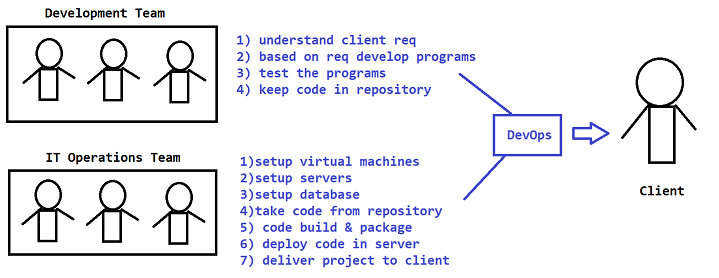
**DevOps**

* DevOps is a culture
* DevOps is a process
* DevOps is set of practices
* DevOps culture is used to Collaborate Development and Operations in Software Project
* Using DevOps culture we can simplify software project delivery process to clients
* DevOps is used throughout software development life cycle process



**Software Development Life Cycle (SDLC)**

* The process of developing and delivering software project is called as SDLC.

**In SDLC we have several phases:**

1) Requirements Gathering

2) Requirements Analysis

3) Design / Planning

4) Development / Coding / Implementation

5) Testing

6) Deployment

7) Maintenance

**Waterfall Methodology**

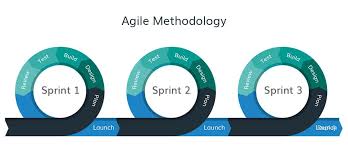
* Earlier people used to follow Waterfall Methodology to develop projects
* Waterfall is a linear methodology to develop and deliver projects
* Everything will happen step by step
* If one step completed then only we will go to next step
* We will move only in forward direction (No backward direction)
* Requirements are fixed
* Budget is fixed
* Client involvement is very less
* Client will see the project at the end



**Note:** Waterfall Methodology is not suitable for big projects

**Agile Methodology**

* Agile is an iterative approach to develop and deliver the projects
* Development and testing will happen parallelly
* Client involvement will be very high
* We will deliver project in multiple releases (Sprints)
* For every release we will take client feedback
* Requirements are not fixed
* Budget is not fixed
* Project Development, Testing & Delivery is very frequent is Agile
* Using DevOps culture we can adopt agile methodology very easily
* DevOps is promoting Agile methodology
* Using DevOps we can achieve Continuous Integration (CI) & Continuous Deployment / Delivery (CD) **CI CD**



**DevOps Advantages**

1) Speed

2) Rapid Development

3) Quick Releases

4) Reliability

5) Security

6) Client Satisfaction

7) Teams Collaboration

Note: DevOps is not one person job; it is everyone's job in the project

**DevOps tools overview**

Build Tools ( Ant / Maven / Gradle )

Repository Tools (SVN / Git Hub / BitBucket )

Code Review Tools ( PMD / Sonar Qube / Sonar lint)

Code Deployment Tools ( Jenkins / UDeploy )

Containerization Tools ( Docker )

Orchestration Tools ( Kubernetes )

Configuration Tools ( Chef / Ansible )

Infrastructure as a Code (IaaC) ( Terraform)

Monitoring Tools ( Nagios / Graphana )

Project Management Tools ( Jira )

**MAVEN**

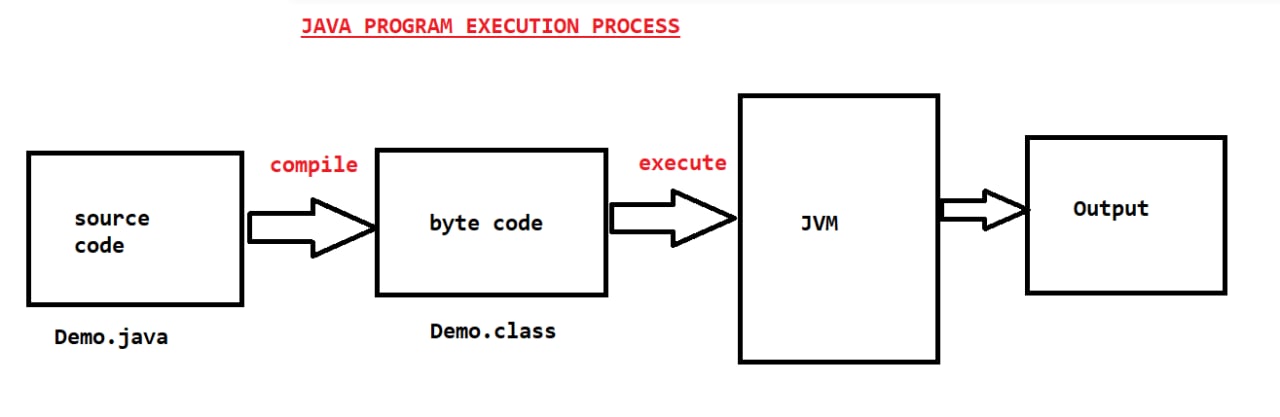
* Maven is a free and open source software given by **Apache Organization**
* Maven s/w is developed using **Java programming language**
* Maven is used to perform **Build Automation** for java projects
* Maven is called as **Java Build Tool**

**JAVA**

* Java is a programming language
* Java language developed by sun Microsystem Company
* Oracle Company acquired Sun Microsystem
* Java is under license of **Oracle Company**
* Java is a high level programming language
* Java is simple programming language
* Java program files will have .java as extension

Ex: Demo.java, Hello.java, Driver.java, Calculator.java etc.

* We can't execute .java files directly
* Java Programs should be converted into Machine understandable format to execute
* Java Programs (.java file) contains source code



* We need to compile java source code into byte code using java compiler (javac)

**Ex: javac Demo.java**

* When we compile java code it will create .class file
* We need to execute .class file to run the java program

**Ex: java Demo**

* When we run java program using java command, JVM will start and it will execute java program

**Note:** JVM stands for Java Virtual Machine

* JVM will convert byte code into machine understandable code
* Java project contains several java programs (.java files)
* We need to compile project source code into byte code
* When we compile project source code we will get .class files
* To deploy java project, we will package all .class files as JAR or WAR file

JAR: Java Archive

WAR: Web archive

**What we can do using maven**

1) We can create **initial project folder structure** using maven

2) We can download "**project dependencies**" using maven

(ex: springboot, hibernate, kafka, redis, email, log4j, junit, security...)

* Java is a free s/w given by sun microsystem
* To develop one java project we will use several frameworks like spring, hibernate etc. along with Java
* We need to download those frameworks and we should add to our java project
* These frameworks we are using in our project are called as our project dependencies
* Instead of we are downloading dependencies, we can tell to maven s/w to download dependencies

**Note:** Required dependencies we will add in **"pom.xml"** file then maven s/w will download them

* pom stands for **project object model**
* When we create maven project then pom.xml file will be created automatically
* **pom.xml** will act as **input file** for maven software

3) We can **compile project source code** using maven

**Note:** Compilation means converting java source code into byte code

Compilation

Demo.java Demo.class

**Note:** java project means collection of java programs.

4) We can package java project as **jar or war file** using maven

JAR - Java Archive

WAR - Web Archive

**Standalone java applications** will be executed as a **jar file.**

**Java Web Applications** will be executed as a **war file.**

**Maven Installation**

1) Download and install Java software

* When we install java we will below 2 things

**a) JDK (Java Development Kit)**

**b) JRE (Java Runtime Environment)**

* JDK contains set of tools to develop java programs
* JRE contains platform/environment which is used to run java programs

**Link To Download Java:** <https://www.oracle.com/in/java/technologies/javase/javase8-archive-downloads.html>

2) Set JAVA\_HOME in Environment Variables (System Environment Variables)

* User Environment Variables: Specific to particular account which logged in our PC
* System Environment Variables: For All Accounts

JAVA\_HOME = C:\Program Files\Java\jdk1.8.0\_202

**Note: Environment Variables will be used by operating system to find out location of software**

3) Set Path for JAVA (Go to System Environment Variables Environment Variables System Variables Select Path and Click on Edit then add JDK path)

Path = C:\Program Files\Java\jdk1.8.0\_202\bin

4) Verify Java installation by executing below command in "**Command Prompt"**

**java -version**

**Note:** It should display java version which we have installed

5) Download Maven software from Apache website

Link to download Maven: https://maven.apache.org/download.cgi

File Name: apache-maven-3.8.5-bin.zip (Binary Archive)

6) Extract Maven Zip file -> Copy extracted maven folder and paste it in "C" drive

7) Set MAVEN\_HOME in System Environment Variables

MAVEN\_HOME = C:\apache-maven-3.8.5

8) Set Path for Maven in System Environment Variables

Path: C:\apache-maven-3.8.5\bin

9) Open Command Prompt and verify Maven Installation using below command

**mvn –version**

**Maven Terminology**

Archetype

Groupid

Artifactid

Packaging

* Archteype represents what type of project we want to create

maven-archetype-**quickstart**: It represents java **standalone application**

maven-archetype-**webapp**: It represents java **web application**

**Note:** Maven providing 1500+ archetypes

* groupId represents company name or project name
* artifactId represents project name or project module name
* packaging represents how we want to package our java application (jar or war)

**Creating standalone application using maven**

1) Create one folder for maven practice

2) Open Command prompt from that folder

3) Execute below command to create maven project

* **mvn archetype:generate -DgroupId=in.ashokit -DartifactId=01-Maven-App -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false**

4) Once project created verify project folder structure

01-Maven-App

- src

- main

-java

- test

-java

- pom.xml

src/main/java: Application source code (.java files)

src/test/java: Application Unit Test code (.java files)

pom.xml: Project Object Model (Maven configuration file)

5) We can add dependencies in pom.xml file

6) We can find maven dependencies in www.mvnrepository.com website

7) Add below dependency in pom.xml file

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.2.22.RELEASE</version>

</dependency>

**How maven will download dependencies**

* Maven will download dependencies using repository

In Maven we have 3 types of repositories

1) Central Repository

2) Remote Repository

3) Local Repository

Central repository is maintaining by **apache organization**

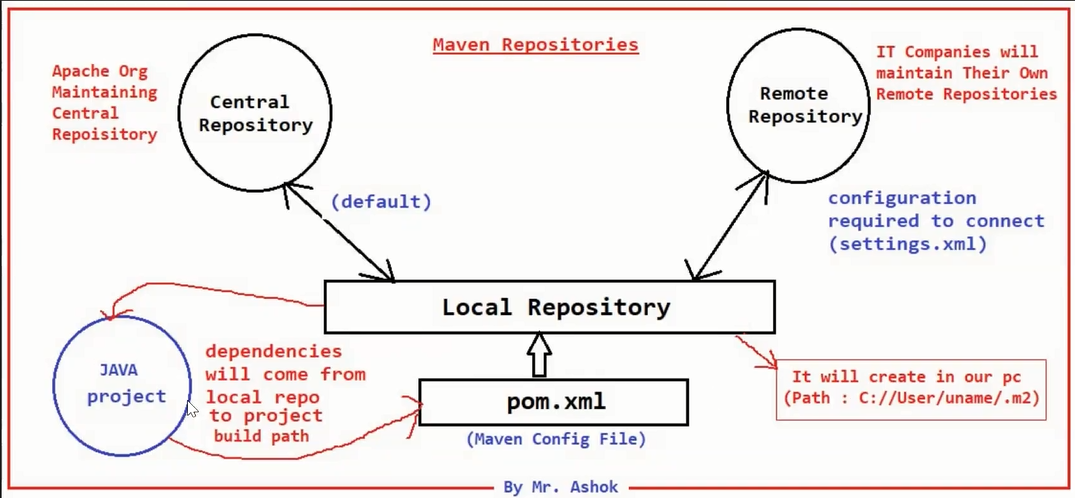
**Every company** will maintain their **own remote repository**

**Local repository** will be created in our system (Location: C://users/<uname>/**.m2**)

* When we add dependency in pom.xml, maven will search for that dependency in local repository. If it is available it will add to project build path.
* If dependency not available in local repository then maven will connect to Central Repository or Remote Repository based on our configuration.

**Note**: By default maven will connect with central repository. If we want to use remote repository then we need to configure remote repository details.

**Note**: Every software company will maintain their own remote repository (Ex: JFrog)



**Configuring Remote Repository (Configure in pom.xml file)**

<repositories>

<repository>

<id>id</id>

<url>jfrong-repo-url/</url>

</repository>

</repositories>

**Maven Goals**

To perform project build activities maven provided several goals for us

1. **Clean**
2. **Compile**
3. **Test**
4. **Package**
5. **Install**

* **Clean goal** is used **to delete target folder**
* Compile goal is used to compile project source code. Compiled code will be stored in target folder

Compile

.java ------------> .class

* **Test goal** is used to **execute unit test code** of our application (junit code)
* **Package goal** is used **to generate jar or war file** for our application based on packaging type available in pom.xml file.

**Note**: jar or war file will be created in target folder.

* **Install goal** is used **to install our project as a dependency** in maven local repository.

**Note**: Every maven goal is associated with maven plugin. When we execute maven goal then respective maven plugin will execute to perform the operation.

**Syntax**: mvn <goal-name>

**Note**: We need to execute maven goals from project folder

Creating web application using maven

* **mvn archetype:generate -DarchetypeArtifactId=maven-archetype-webapp -DgroupId=in.ashokit -DartifactId=01-maven-web-app -DinteractiveMode=false**

**SOURCE CODE REPOSITORY TOOLS / VERSION CONTROL SOFTWARE**

* Multiple developers will work for project development
* Developers will be working from multiple locations
* All developers code should be store at one place (Code Integration Should Happen)
* To integrate all the developers source code at one place we will use Source code Repository Software

**Advantages with Source code repository software**

1) All the developers can connect to repository server and can integrate the code

2) Repository server will provide monitored access

**- Who**

**- When**

**- Why**

**- What**

**Repository Tools**

* **SVN (outdated)**
* **Git Hub**
* **BitBucket**

**Environment Setup to work with Git Hub**

1) Create Github account ( www.github.com )

2) Download and install Git Client software ( https://git-scm.com/downloads )

3) Once installation completed, right click on the mouse and verify git options display (If git options displaying our git client installation completed successfully)

**Working with GitHub**

* Login into github account with your credentials
* Create Repository in github

**Note:** Repository is used to store project source code. Every Project will have one repository

* When we create a repository, unique URL will be generated with Repository Name (i.e Repo URL)
* All the developers will connect to repository using Repository URL
* We can create 2 types of Repositories in Git Hub

**1) Public repository**

**2) Private repository**

* Public Repository means everybody can access but we can choose who can modify our repository
* Private Repository means we will choose who can access and who can modify

**Repo URL: https://github.com/ashokitschool/01-devops-app.git**

**Working with Git Bash**

Git Bash we can use as Git Client software to perform Git Operations

Download and install git client (https://git-scm.com/downloads)

Right Click on Mouse and choose **"Open Git Bash Here"**

**git help :** It will display frequently used git commands

**git help <cmd-name> :** It will open documentation for given command

**Configure Your Email and Name in GitBash with Commands**

**$ git config --global user.email "youremail@yourdomain.com"**

**$ git config --global user.name "name"**

**$ git init :** To initialize our folder as git working tree folder

**$ git clone :** To clone git repository to our machine from github.com

**Syntax: $ git clone <project-repo-url>**

**$ git status : It will display staged , un-staged and un-tracked files**

**Syntax: $ git status**

**Staged Files:** The files which are added for commit

**Un-Staged Files:** The files which are modified but not added for commit

**Un-tracked files:** Newly created files

**Note:** To commit a file(s), we should add to staging area first

**$ git add :** It is used to add file(s) to staging area

**Syntax: $ git add <file-name>**

**$ git add .**

**$ git commit :** It is used to commit staged files to git local repository

**Syntax: $ git commit -m 'reason for commit'**

**$ git push :** To push changes from git local repository to git central repository

**Syntax: $ git push**

**Steps to push code to github central repository**

1) Create one public repository in git hub (take github repo url)

2) Clone github repository using 'git clone' command

**$ git clone 'repo-url'**

3) Navigate to repository folder

4) Create one file in repository folder

**$ touch Demo.java**

5) Check status of the file using 'git status' command

**$ git status (It will display as untracked file)**

6) Add file to staging area using 'git add' command

**$ git add .**

7) Commit file to git local repository

**$ git commit -m 'commit-msg'**

8) Push file from git local repository to git central repository using 'git push' command

**$ git push**

**Note:** If you are doing 'git push' for first time it will ask to enter your github account password**.**

**Note:** Git bash will ask our password only for first time. It will save our git credentials in Credential Manager in Windows machine.

**Go to Credential Manager -> Windows Credentials -> Select Github -> We can modify and delete saved credentials from here**

* When we do git commit then it will geneate a commit-id with 40 characters length
* From this commit-id it will display first 7 characters in git hub central repository
* We can check commit history using 'git log' command

**Steps to commit Maven Project to Github Repository**

1) Create Maven Project

2) Create GitHub Repository

**Note:** After creating git repository, it will display set of commands to execute

3) Open gitbash from project folder and execute below commands

**$ git init**

**$ git status**

**$ git add .**

**$ git commit -m 'commit-msg'**

**$ git branch -M main**

**$ git remote add origin <repo-url>**

**$ git push -u origin master**

**When we are working on one task suddenly we may get some other priority task.**

**Use case**

* Manager assigned task id : 101
* I am working on that task (i am in middle of the task)
* Manager told that stop the work for 101 and complete 102 on priority.
* Once 102 is completed then resume your work on 101

When manager asked me to start 102 task, i have already done few changes for 101 (Partially completed)

We can't push partial changes to repository because with our partial changes existing functionality may break.

We can't delete our changes because we have spent few hours of time to implement those changes

**In this scenario we will go for 'git stash' option**

Git stash is used to save working tree changes to temporary location and make working tree clean.

After priority work completed we can get stashed changes back using **'gitstash apply'**

**Git Branches**

Branches are used to maintain separate code bases for our project

In Git repository we can create multiple branches

* **Main**
* **Develop**
* **QA**
* **Release**
* **Research**

Development team will integrate the code in 'develop' branch

Bug-fixing team will integrate the code in 'QA' branch

R & D team will integrate the code in 'research' branch

In github we can create branches

**To clone particular branch in git repo we will use below command**

**$ git clone -b <branch-name> <repo-url>**

What is Git branch?

Why we need git branches?

How to create git branches?

How to clone particular branch?

How to switch from one branch to another branch?

**Branch Merging**

* The process of merging changes from one branch to another branch is called as Branch merging
* We will use Pull Request for Branch Merging which is available in github GUI

**Steps to do branch merging**

1) Create feature branch from main branch

2) clone feature branch

3) Create new file in feature branch then commit and push to central repo

4) Go to central repository then create pull request to merge feature branch changes to main branch

**Note: Once feature branch changes are merged to main branch then we can delete feature branch (if required)**

**What is .gitignore?**

* This .gitignore is used to configure the files or folders which we want to ignore from our commits
* The files and folders which are not required to commit to central repository those things we can configure in .gitnore file
* Ex: In maven project 'target' folder will be available which is not required to commit to central repository. This we can configure in .gitignore file.

**--------------------------------------------.gitignore-------------------------------------------------------**

HELP.md

target/

!.mvn/wrapper/maven-wrapper.jar

!\*\*/src/main/\*\*/target/

!\*\*/src/test/\*\*/target/

### STS ###

.apt\_generated

.classpath

.factorypath

.project

.settings

.springBeans

.sts4-cache

### IntelliJ IDEA ###

.idea

\*.iws

\*.iml

\*.ipr

### NetBeans ###

/nbproject/private/

/nbbuild/

/dist/

/nbdist/

/.nb-gradle/

build/

!\*\*/src/main/\*\*/build/

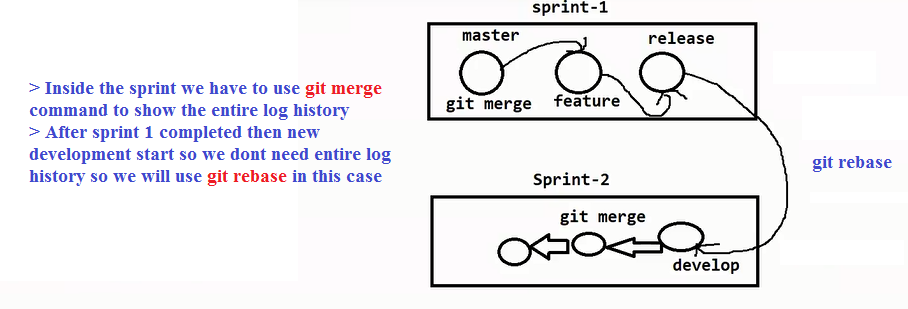
!\*\*/src/test/\*\*/build/

### VS Code ###

.vscode/

**git merge vs git rebase**

* These commands are used to merge changes from one branch to another branch
* git merge will maintain commit history
* git rebase will not maintain that rebase history
* When we are working on particular sprint and we want to merge changes from one branch to another branch then we will use 'git merge' command
* Once sprint-1 is delivered then we want to take latest code of sprint-1 to start sprint-2 development. In this scenario we don't need commit history so we will use 'git rebase' command.



**What is git pull command?**

Pull command is used to take latest changes from repository to local

When we want to make some changes to code, it is always recommended to take git pull

**Note: When we execute 'git pull' there is a chance of getting conflicts. We need to resolve the conflict and we should push the code without conflicts.**

What is Source Code Repository?

Why we need source code repository

What are the source code repository servers available?

What is Code Integration?

What is Monitored Access?

What is git hub?

What is git

What is version control?

What is Repository?

Public Repository vs Private Repository

Cloning Repository

Staged vs Unstaged vs Untracked File

Adding Files to Stating Area

Unstaging the files from staging

Discarding local changes

What is working tree?

What is Local Repostiory

What is Central Repository?

Commit from working tree to local repo

Push from local repo to central repo

Taking latest code changes

Push vs pull

What is conflict?

How to resolve conflicts

What is branch in git hub?

How to create branches

How to clone particular branch

how to switch to particular branch

How to merge branches

What is pull request?

git merge vs rebase

What is .gitignore

**git init**

**git help**

**git config**

**git clone**

**git status**

**git add .**

**git add <file-name>**

**git restore**

**git commit**

**git push**

**git pull**

**git log**

**git rm**

**git branch**

**git checkout**

**git merge**

**git rebase**

**Assignment**

1) Create Maven Web Application

2) Added 'Spring-Core' dependency in project pom.xml file

3) Package maven project as war file using maven goal

4) Create Git repository in github.com (public repo)

5) Push maven project into github repo using gitbash

(Target folder shouldn't be commited, add this is .gitignore file)

6) Make changes in pom.xml and push changes to github repo using git bash

7) Create 'feature' branch in git repo from main branch

8) Clone feature branch from git bash using git clone

9) Make changes in 'feature' branch pom.xml file and push changes to central repo

10) Create pull request and merge 'feature' branch changes to 'main' branch

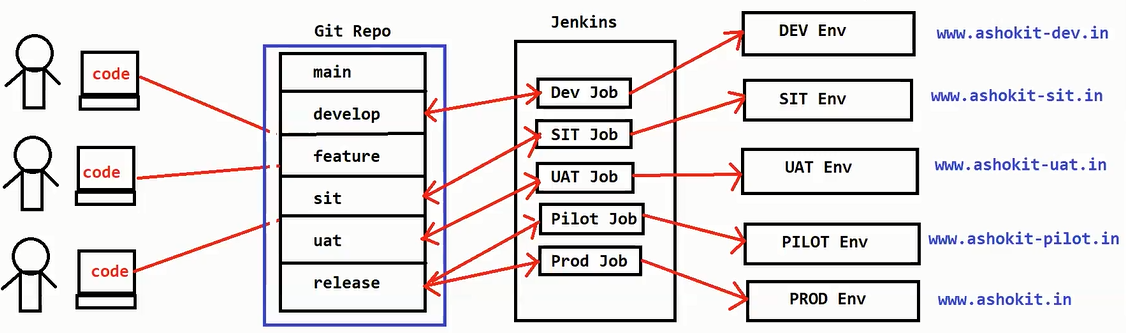
**ghp\_9iUitPbtyTYPkZXxy8KoLQDK5HwQmH01w3YT**

**Saisunil3012**

**ghp\_9g317tqHqjI12ogdDxJAFoFdk76svP3P97hr**

**APPLICATION ENVIRONMENTS**

* Environment means the platform which is used to run our application
* In Real-time, we will use multiple environments to run our application
* DEV Environment
* SIT Environment
* UAT Environment
* PILOT Environment
* PROD Environment



* Developers will use DEV environment for integration testing.
* SIT environment will be used by Software Testing team to test our application functionality. SIT stands for System Integration Testing.
* UAT means User Acceptance Testing. UAT will be done by Client. For client testing we will give UAT environment.

**Note:** After UAT testing, client will decide GO or No-GO. GO means go for production deployment. No-GO means production deployment cancelled.

* When client identify some critical bugs in UAT then they will say No GO.
* PILOT environment is called as pre-prod environment. It is used for live data testing.
* PROD environment is called live environment. The application which is deployed in PROD environment will be available for public user’s access.

**Note: To deploy the project into these environments we will use Jenkins software.**

**APACHE TOMACT SERVER**

* Apache Tomcat is a web server
* Apache Tomcat is used to run Java Web Applications
* Apache Tomcat is free & open source
* Apache Tomcat runs on 8080 port by default (we can change that port)

**Apache Tomcat Folder Structure**

**bin:** It contains commands to start and stop tomcat server

**conf:** It contains configuration files

**lib:** It contains libraries (jar files)

**logs:** It contains server log files

**temp:** Temp files will be created here (we can delete them)

**webapps: This is called as deployment folder**

**Note: We will keep war file in webapps folder for deployment**

**Working with Apache Tomact in Linux**

* Login into AWS Management Console
* Create EC2 Instance (Amazon Linux AMI)
* Connect to EC2 instance using MobaXterm / Putty
* install java software using below command

**$ sudo yum install java-1.8.0-openjdk**

* Verify the version of java installed in our machine

**$ java -version**

**Note: If we have multiple java versions installed then we can switch to particular version using below command**

**$ alternatives --config java**

* We can download apache tomcat from official website

**URL: https://tomcat.apache.org/download-90.cgi**

* We can find apache tomcat urls to download in official website downloads page
* Copy the URL of tar file and execute below command in linux machine

**$ wget <tomact-tar-file-url>**

**Note: I am using**

**"https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.63/bin/apache-tomcat-9.0.63.tar.gz"**

* After tomact tar file got downloaded then extract Tomcat Tar file using below command

**$ tar -xvf <tomact-tar-file-name>**

* Go inside tomcat folder and see folder structure

**$ cd tomact-folder**

**$ ls -ltr**

* Go to tomact bin directory and run tomact server

**$ cd bin**

**$ ./startup.sh**

**Note: Tomcat Server runs on 8080 port by default. Enable this port in security group as custom tcp**

**Type: Custom TCP**

**Protoal : TCP**

**Port Range: 8080**

**Source : Custom (0.0.0.0/0)**

* Access Tomact server from your browser

**URL : http://EC2-VM-Public-IP:8080/**

**Note: It should open tomact server home page.**

* By default the Host Manager is only accessible from a browser running on the same machine as Tomcat. If you wish to modify this restriction, you'll need to edit the Host Manager's **context.xml file.**

**File Location : <tomcat>/webapps/manager/META-INF/context.xml**

* In Manager context.xml file, change <Valve> section like below (allow attribute value changed)

**<Context antiResourceLocking="false" privileged="true" >**

**<Valve className="org.apache.catalina.valves.RemoteAddrValve" allow=".\*" />**

**</Context>**

**Add tomact users in "tomact-folder/conf/tomact-users.xml" file like below**

<role rolename="manager-gui" />

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

<role rolename="admin-gui" />

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

**Stop the tomact server and start it**

* **We can change tomcat server default port in tomact/conf/server.xml file**

**Note: When we change tomact port number in server.xml file then we have to enable that port in Security Group which is associated with our EC2 instance.**

**Steps to display Maven Web Application in Tomcat Server**

**1)** Create Maven Web application

**2)** Edit "index.jsp" file like below (File Location : project-folder\src\main\webapp)

**<html>**

**<body>**

**<h1><font color='red'>Welcome to Ashok IT..!!<font></h1>**

**<h2>Learn Here.. Lead Anywhere..!! </h2>**

**<a href="https://ashokitech.com/online-training-schedules">Click Here To See Training Schedules</a>**

**</body>**

**</html>**

**3)** Package maven web application as war file using maven goals

**$ mvn clean package**

**4)** Go to Tomcat Server Admin Dashboard and click on "Manager App"

**5)** Select War file to upload and click on 'deploy' button

**6)** War file will be deployed and it will display in applications

**7)** Click on Application Path (It will open the application in browser)

**Conclusion**

Stop Apache Tomact Server

Stop Ec2 instance

**--------------------------------------------------------------------------------------------------**

**sudo amazon-linux-extras install java-openjdk11**

**sudo yum install java-11-openjdk**

**sudo alternatives --config java**

**\*1 java-1.8.0-openjdk.x86\_64 (/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.332.b09-1.amzn2.0.2.x86\_64/jre/bin/java)**

**+2 java-11-openjdk.x86\_64(/usr/lib/jvm/java-11-openjdk-11.0.13.0.8 1.amzn2.0.3.x86\_64/bin/java)**

**In this location we will get which java version is installed for jenkins.**

**/usr/lib/systemd/system/jenkins.service**

**JENKINS**

**What is Jenkins?**

* Jenkins is an open source automation tool for CI and CD
* Jenkins tool developed using Java
* Jenkins is part of Hudson Project
* Initially it is called as **Hudson** then later it renamed to Jenkins

**About CI CD**

* CI and CD are two most frequently used terms in modern development practices and DevOps practices.
* CI stands for Continuous Integration. It is fundamental DevOps best practice where developers frequently merge code changes to central repository where automated builds and tests runs.
* CD means Continuous Delivery or Continous Deployment.
* Jenkins is a self-contained, open-source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

**Build & Deployment Process**

1) Take latest source code from repository

2) Compile source code

3) Execute Unit tests (Junits)

4) Perform Code Review

5) Package code as war file

6) Deploy the war file into server

**Note: All the above build and deployment tasks can be automated using Jenkins tool.**

**Components of Jenkins:**

**Master Node:** The system where you install and run Jenkins.

**Managed Node:** Target systems managed by Jenkins.

**Repository:** Version controlled system where you keep your code.

**UI:** User interface to manage and configure Jenkins.

**Pipeline:** Means of continuous deployment of your code to target systems from the version control system.

**Plugins:** Means of enhancing the functionality of a Jenkins.

**Jenkins Installation:**

**Prerequisite:**

One EC2 Amazon Linux 2 instance with internet access

**Java 11** runtime environments

**Step1- Jenkins installation on AWS EC2**

* Create an EC2 instance with Amazon Linux 2 AMI
* Amazon Linux 2 AMI (HVM), SSD Volume Type
* Connect to your EC2 instance
* Update all packages

**$ sudo yum update -y**

* Install Java

**$ sudo amazon-linux-extras install java-openjdk11**

**Step2-Jenkins installation on AWS EC2 using YUM**

* The stepwise procedure is available on [www.jenkins.io](http://www.jenkins.io)

GoTo Documentation---> Installing Jenkins--> Linux---> Ubuntu/Redhat

* Add Jenkins repo to your yum repository

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

* Import a key file from Jenkins-CI to enable installation from the package

**$ sudo rpm --import https://pkg.jenkins.io/redhat/jenkins.io.key**

* For Amazon Linux 2

**$ sudo amazon-linux-extras install epel**

* Install Jenkins

**$ sudo yum install jenkins -y**

* Start and enable Jenkins service

**$ sudo systemctl enable jenkins**

**$ sudo systemctl start jenkins**

**$ sudo systemctl status jenkins**

* Get the initial administrative password

**$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword**

**pwd : 5fe6ddcc9db244cab6aca5ccf2d6a83a**

**Step3:**

* Open your EC2 instance public DNS or public IP (http://<PUBLIC\_DNS/PUBLIC\_IP>:8080/) along with port 8080 in your favorite browser and provide the administrative password obtained during the installation.

**Note: Make sure you enabled 8080 port in Security Group Inbound Rules which is attached to EC2 instance.**

**URL : http://43.204.214.21:8080/**

* Provide pwd which we have copied to unlock jenkins
* Select "Install Suggested Plugins" card (it will install those plugins)
* Create Admin account

**If we forgot jenkins Password then how to recover it?**

* Go to **/var/lib/jenkins/**
* Open : config.xml file
* Set Value for useSecurity as false
* ex: **<useSecurity>false</useSecurity>**
* Re-start jenkins and try to access

**How to change Jenkins Port Number**

* Go to root directory
* Go to **/etc/sysconfig**
* Open jenkins file and change Jenkins port number
* Restart Jenkins server

**JOB NO.1**

**Creating First Job in Jenkins**

1) Goto Jenkins Dashboard

2) Click on New Iteam

* + - Enter Item Name (Job Name)
    - Select Free Style Project & Click OK
    - Enter some description
    - Click on 'Build' tab
    - Click on 'Add Build Step' and select 'Execute Shell'

3) Enter below shellscript

echo "Hello Guys,"

touch ashokit.txt

echo "Hello Guys, Welcome to Jenkins Classes" >> ashokit.txt

echo "Done..!!"

4) Apply and Save

**Note:** With above steps we have created JENKINS Job

5) Click on 'Build Now' to start Job execution

6) Click on 'Build Number' and then click on 'Console Ouput' to see job execution details.

=> Jenkins Home Directory in EC2 : /var/lib/jenkins/workspace/

**$ cd /var/lib/jenkins/workspace/**

7) Go to Jenkins home directory and check for the job name --> check the file created inside the job

**JOB NO.2**

**Jenkins Job with GIT Hub Repo + Maven - Integration**

* Pre-Requisites: Java, Git & Maven
* Git installation In Ec2:

**$ sudo yum install git -y**

* JDK Installation In Jenkins:

Jenkins Dashboard -> Manage Jenkins -> Global Tools Configuration -> Add JDK -> JDK Name (JDK 1.8v/you can give any name) -> Choose version (Java SE Development kit 8u221) -> Configure Oracle account credentials to download JDK.

* Maven Installation In Jenkins:

Jenkins Dashboard -> Manage Jenkins --> Global Tools Configuration -> Add maven-> Name (maven 3.6) -> Version (3.6.1) ->click on save.

**Sample Git Repo URLS For Practice**

Git Hub Repo URL-1 : https://github.com/ashokitschool/JAVA-MAVEN-WEB-APP.git

Git Hub Repo URL-2 : https://github.com/ashokitschool/maven-web-app.git

**Steps To Create Jenkins Job with Git Repo + Maven**

1) Connect to EC2 instance in which jenkins server got installed

2) Start Jenkins Server

3) Access Jenkins Server Dashboard and Login with your jenkins credentials

4) Create Jenkins Job with Git Hub Repo

* + - New Item
    - Enter Item Name (Job Name)
    - Select Free Style Project & Click OK
    - Enter some description
    - Go to "Source Code Management" Tab and Select "Git"
    - Enter Project "Git Repo URL"
    - Branch to build (**specify the branch available in github)**
    - Add Your Github account credentials
    - Go to "Build tab"
    - Click on Add Build Step and Select '**Inovke Top Level Maven Targets'**
    - Select Maven and enter goals '**clean package'**
    - Click on Apply and Save

**Note:** With above steps we have created JENKINS Job

5) Click on 'Build Now' to start Job execution

6) Click on 'Build Number' and then click on 'Console Ouput' to see job execution details.

7) Jenkins Home Directory in EC2: /var/lib/jenkins/workspace/

8) Go to jenkins workspace and then go to job folder then goto target folder then see war file created.

9) Access below URL in browser to stop Jenkins Server

URL : http://EC2-VM-IP:8080/exit

(Click on Retry using Post button)

**JOB NO.3**

**Steps To Create Jenkins Job with Git Repo + Maven + Tomcat Server**

1) Go to Tomcat server folder and configure below users in "**tomcat-users.xml**" file (it will be available in tomcat server **conf** folder)

Jenkins is going to use that user to execute the script to do this **automated deployment** that user should have **manager-script** permission. User will have a permission to execute script with that script the war file deploy the server.

<role rolename="manager-gui" />

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

<role rolename="admin-gui" />

**<role rolename="manager-script" />**

<user username="admin" password="admin" roles="manager-gui,**manager-script**,admin-gui"/>

2) Go to Jenkins Dashboard -> Manage Jenkins --> Manage Plugins -> Goto Available -> Search for "**Deploy To Container Plugin**” -> Install it.

3) Create Jenkins Job

* + - New Item
    - Enter Item Name (Job Name)
    - Select Free Style Project & Click OK
    - Enter some description
    - Go to "Source Code Management" Tab and Select "Git"
    - Enter Project "Git Repo URL"
    - Add Your Github account credentials
    - Go to "Build tab"
    - Click on Add Build Step and Select **'Inovke Top Level Maven Targets'**
    - Select Maven and enter goals 'clean package'
    - Click on **'Post Build Action'** and select **'Deploy war/ear to container'** option
    - Give path of war file (You can give like this also: \*\*/\*.war )
    - Enter Context Path (give project name)
    - Click on 'Add Container' and select Tomcat version 9.x
    - Add Tomcat server credentials (give the username & pwd which is having manager-script role)
    - Enter Tomact Server URL (http://ec2-vm-ip:tomcat-server-port)
    - Click on Apply and Save

4) Run the job now using 'Build Now' option and see 'Console Output' of job

5) Once Job Executed successfully, go to tomcat server dashboard and see application should be displayed.

6) Click on the application name (it should display our application)

**JOB NO.4**

**JENKINS PIPELINE**

* Sequence of Jobs execution is called as Pipeline
* For our application, we will have multiple environments like below
  + - * DEV
      * SIT
      * UAT
      * PILOT
      * PROD
* For every environment, one JENKINS Job will be created

JENKINS\_DEV\_JOB -------> DEV Environment (Dev Server)

JENKINS\_SIT\_JOB -------> SIT Environment (SIT Server)

JENKINS\_UAT\_JOB ------> UAT Environment (UAT Server)

JENKINS\_PILOT\_JOB -----> Pilot Environment (Pilot Server)

JENKINS\_PROD\_JOB ------> Prod Environment (Prod Server)

* If we want to deploy code changes to all environments then it is recommended to create Build Pipeline
* By Using Build Pipeline we can execute JENKINS Jobs sequentially.

**Requirement**

* If code commits happend in git hub then deploy code into DEV Server
* If DEV Server Deployment successful, then deploy code into SIT environment
* If SIT Server deployment successful, then deploy code into UAT environment.
* **“Build pipeline plugin”** to install.

**Steps To Create Jenkins Pipeline**

1) Create EC2 VM and install Tomcat Server (Dev Server VM)

2) Create EC2 VM and install Tomcat Server (SIT Server VM)

3) Create EC2 VM and install Tomcat Server (UAT Server VM)

4) Create EC2 VM and install Jenkins (Jenkins Server VM)

5) Install JDK, Maven, Git, Deploy To Container in Jenkins Server VM

6) Create Jobs in Jenkins Server

DEV-Job ----> Dev Server

SIT-Job ----> SIT Server (SIT Job should execute if DEV-JOB is stable)

UAT-Job ----> UAT-Server (UAT job should execute if SIT-JOB is stable)

7) Create Jenkins Build Pipeline to execute Jobs in sequence

**JOB NO.5**

**How to Create Jenkins Jobs with Build Parameters**

Create New Item

Enter Item Name & Select Free Style Project

Select "This Project is parameterized" in General Section

Select Choice Parameter

Name: BranchName

Choices: Enter every branch name in next line

Branches to Build: \*/${BranchName}

**Note: we need to select branch name dynamically so we can deploy the code from any branch.**

**Creating Users in Jenkins**

Manage Jenkins -> Manage Users

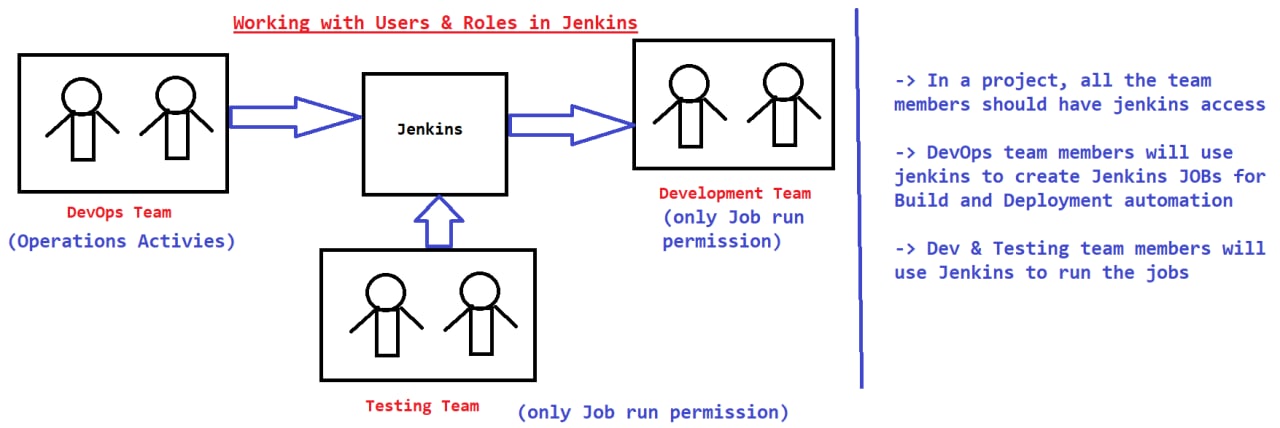
Create User

Note: 2 developer users and 2 devops users

Go to /var/lib/jenkins/users

View users.xml file (you can see user accounts created)

Configure Global Security for Users

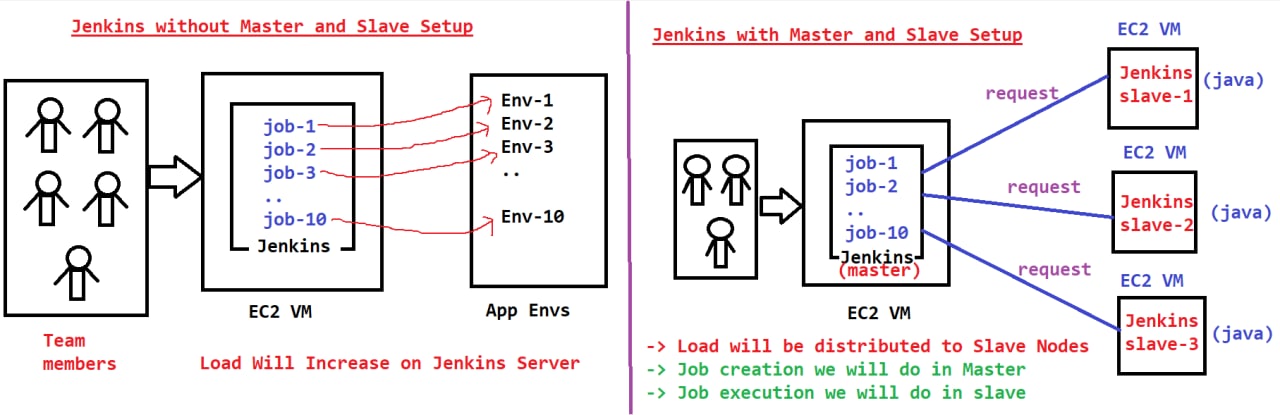


Excercise

Create 1 account for DevOps team member

Create 1 account for Development team member

Configure Roles for DevOps team member and Development team member



**Jenkins Master and Slave Configuration**

* When we build the Jenkins job in a single Jenkins master node then Jenkins uses the resource of the base machine and if no executor is available then the jobs are queued in the Jenkins server.
* Sometimes you might need several different environments to test your builds. This cannot be done by a single Jenkins server.
* It is recommended not to run different jobs in the same system that required a different environment. In such a scenario we need a different machine with a different environment that takes the specific job from the master to build.
* On the same Jenkins setup, multiple teams are working with their jobs. All jobs are running on the same base operating system and the base operating system has limited resources.
* **To overcome this problem, Jenkins provided Distributed Architecture i.e Jenkins Master Slave Architecture.**
* Jenkins uses A Master-Slave architecture to manage distributed builds. The machine where we install Jenkins software will be Jenkins master and that run’s on port 8080 by default. On the slave machine, we install a program called Agent. This agent requires JVM. This agent executes the tasks provided by Jenkins master. We can launch n numbers of agents and we can configure which task will be run on which agent server from Jenkins master by assigning the agent to the task.

**Jenkins Master**

Your main Jenkins server is the Master. The Master’s job is to handle:

a) Scheduling build jobs.

b) Dispatching builds to the slaves for the actual execution.

c) Monitor the slaves (possibly taking them online and offline as required).

d) Recording and presenting the build results.

e) A Master instance of Jenkins can also execute build jobs directly.

**Jenkins Slave**

A Slave is a Java executable that runs on a remote machine. Following are the characteristics of Jenkins Slaves:

1) It hears requests from the Jenkins Master instance.

2) Slaves can run on a variety of operating systems.

3) The job of a Slave is to do as they are told to, which involves executing build jobs dispatched by the Master.

4) You can configure a project to always run on a particular Slave machine or a particular type of Slave machine, or simply let Jenkins pick the next available Slave.

**JOB NO.6**

**Working with Jenkins Master Slave Architecture**

**Step-1: Create Jenkins Master**

1) Create EC2 instance

2) Connect EC2 using Mobaxterm

3) Install Git client

4) Install Java Software

5) Install jenkins server

6) Add Git, JDK and Maven Plugins

7) Enable Jenkins Port Number in Security Group

8) Access Jenkins Server in Browser and Login

**Step-2: Create Jenkins Slave**

1) Create EC2 instance

2) Connect to EC2 using Mobaxterm

3) Install Git client

4) Install Java Software

5) Create one directory in /home/ec2-user (ex: slavenode)

**Step-3: Configure Slave Node in Jenkins Master Node**

1) Go to Jenkins Dashboard

2) Go to Manage Jenkins

3) Go to Manage Nodes & Clouds

4) Click on 'New Node' -> Enter Node Name -> Select Permanent Agent

5) Enter Remote Root Directory ( /home/ec2-user/slavenode )

6) Enter Label name as Slave-1

7) Select Launch Method as 'Launch Agent Via SSH'

8) Give Host as 'Slave VM DNS URL'

9) Add Credentials (Select Kind as: SSH Username with private key)

10) Enter Username as: ec2-user

11) Select Private Key as Enter Directly and add private key

**Note:** Open **gitbash** from your .pem file location and execute below command to get private key from pem file

$ cat <key-pair-file-name>.pem

**Note:** It will display private key on Git Bash terminal (Just copy and paste in Jenkins)

12) Select Host Key Strategy as 'Manually Trusted Key Verification Strategy'

13) Click on Apply and Save (We can see configure slave)

**\*\*\*\*\*\*\*\*\*\*\*\*With above steps Master and Slave Configuration Completed\*\*\*\*\*\*\*\*\*\***

* Then Go to Jenkins Server and Create Jenkins Job
* **Note:** Under General Section of Job creation process, Select "Restrict Where This Project Can Run" and enter Slave Nodel Label name and finish job creation.
* Execute the Job using 'Build Now' option
* **Note:** Job will be executed on Slave Node (Go to Job Console Output and verify execution details)

**JENKINS - Pipeline**

* Jenkins Pipeline is an automation solution that lets you create simple or complex pipelines.
* Jenkins Pipeline is a combination of Plugins which automates number of tasks and makes the CI/CD pipeline efficient, high in quality and reliable.
* Jenkins provides two ways of developing a pipeline

1) Scripted

2) Declarative

* Traditionally, Jenkins jobs were created using Jenkins UI called Freestyle jobs.
* In Jenkins 2.0, Jenkins introduced a new way to create jobs using the technique called **pipeline as code**.
* In pipeline as code technique, jobs are created using a script file that contains the steps to be executed by the job.
* In Jenkins, that scripted file is called Jenkinsfile.

**What is Jenkinsfile?**

* Jenkinsfile is nothing but a simple text file which is used to write the Jenkins Pipeline and to automate the Continuous Integration process.
* Jenkinsfile usually checked in along with the project’s source code in Git repo. Ideally, every application will have its own Jenkinsfile.
* Jenkinsfile can be written in two ways

1) Scripted pipeline syntax

2) Declarative pipeline syntax

**What is Jenkins Scripted Pipeline?**

* Jenkins pipelines are traditionally written as scripted pipelines. Ideally, the scripted pipeline is stored in Jenkins webUI as a Jenkins file. The end-to-end **scripted pipeline script is written in Groovy**.
* It requires knowledge of Groovy programming as a prerequisite.
* Jenkinsfile **starts with** the word **node**.
* It can contain standard programming constructs like if-else block, try-catch block, etc.

**Sample Scripted Pipeline**

node {

stage('Stage 1') {

echo 'hello'

}

}

**What is Jenkins Declarative Pipeline?**

* The Declarative Pipeline subsystem in Jenkins Pipeline is relatively new, and provides a simplified, opinionated syntax on top of the Pipeline subsystems.
* It is a latest addition in Jenkins pipeline job creation technique.
* Jenkins declarative pipeline needs to use the predefined constructs to create pipelines. Hence, it is not flexible as a scripted pipeline
* Jenkinsfile **starts with** the word **pipeline**.
* Jenkins declarative pipeline should be the preferred way to create a Jenkins job as they offer a rich set of features, come with less learning curve & no prerequisite to learn a programming language like Groovy just for the sake of writing pipeline code.
* We can also validate the syntax of the Declarative pipeline code before running the job. It helps to avoid a lot of runtime issues with the build script.

**Our First Declarative Pipeline**

pipeline {

agent any

stages {

stage('Welcome Step') {

steps {

echo 'Welcome to Jenkins Scripting'

}

}

}

}

* **pipeline**: Entire Declarative pipeline script should be written inside the pipeline block. It’s a mandatory block.
* **agent**: Specify where the Jenkins build job should run. agent can be at pipeline level or stage level. It’s mandatory to define an agent.
* **stages**: stages block constitutes different executable stage blocks. At least one stage block is mandatory inside stages block.
* **stage**: stage block contains the actual execution steps. Stage block has to be defined within stages block. It’s mandatory to have at least one stage block inside the stage block. Also its mandatory to name each stage block & this name will be shown in the Stage View after we run the job.
* **steps**: steps block contains the actual build step. It’s mandatory to have at least one step block inside a stage block.
* Depending on the Agent’s operating system (where Jenkins job runs), we can **use shell, bat, etc., inside the steps command**.

**JOB NO 7-BUILD PIPELINE SCRIPT**

pipeline{

agent any

environment {

PATH = "$PATH:/opt/apache-maven-3.6.3/bin"

}

stages{

stage('GetCode'){

steps{

git branch: 'main',

url: 'https://github.com/ashokitschool/maven\_web\_app\_jenkins\_pipeline.git'

}

}

stage('Build'){

steps{

sh 'mvn clean package'

}

}

}

}

**JOB NO 8-BUILD PIPELINE + SONAR QUBE SERVER – SCRIPT**

pipeline{

agent any

environment {

PATH = "$PATH:/opt/apache-maven-3.6.3/bin"

}

stages{

stage('GetCode'){

steps{

git 'https://github.com/ashokitschool/maven-web-app.git'

}

}

stage('Build'){

steps{

sh 'mvn clean package'

}

}

stage('SonarQube analysis') {

steps{

withSonarQubeEnv('Sonar-Server-7.8') {

sh "mvn sonar:sonar"

}

}

}

}

}

**JOB NO 9-JENKINS PIPELINE ( JENKINS + MAVEN + GIT HUB + SONAR + TOMCAT )**

**Note: Install ssh-agent plugin and generate code using pipeline syntax**

pipeline{

agent any

environment {

PATH = "$PATH:/opt/apache-maven-3.6.3/bin"

}

stages{

stage('GetCode'){

steps{

git 'https://github.com/ashokitschool/maven-web-app.git'

}

}

stage('Build'){

steps{

sh 'mvn clean package'

}

}

stage('SonarQube Analysis') {

steps{

withSonarQubeEnv('Sonar-Server-7.8') {

sh "mvn sonar:sonar"

}

}

}

stage('Code deploy') {

steps{

sshagent(['Tomcat-Server-Agent']) {

sh 'scp -o StrictHostKeyChecking=no target/01-maven-web-app.war ec2-user@13.235.68.29:/home/ec2-user/apache-tomcat-9.0.63/webapps'

}

}

}

}

}

**MY JENKINS PIPELINE ( JENKINS + MAVEN + GIT HUB + SONAR + TOMCAT )**

pipeline{

agent any

environment {

PATH = "$PATH:/opt/apache-maven-3.6.3/bin"

}

stages{

stage('GetCode'){

steps{

git branch: 'main', credentialsId: 'f6dad8a3-84e7-45b1-9d9f-8a23cb51540e', url: 'https://github.com/Saisunil3012/maven-web-app.git'

}

}

stage('Build'){

steps{

sh 'mvn clean package'

}

}

stage('SonarQube analysis') {

steps{

withSonarQubeEnv('Sonar-Server-7.8') {

sh "mvn sonar:sonar"

}

}

}

stage('Code deploy') {

steps{

sshagent(['5e7646a0-cca5-4b25-b153-cda343a1f345']) {

sh 'scp -o StrictHostKeyChecking=no target/01-maven-web-app.war ec2-user@43.205.144.82:/home/ec2-user/apache-tomcat-9.0.65/webapps'

}

}

}

}

}

**Email Notifications in Jenkins**

* We can configure Email notifications in Jenkins
* With this option we can send email notification to team members after jenkins job execution completed
* We need to configure SMTP properties to send emails
* Go To Manage Jenkins
* Add Email Extension Server
* We will add company provided SMTP server details

Note: For practice we can use GMAIL SMTP Properties

* Once SMTP properties added then we can configure email notification as 'Post Build Action' in Jenkins job

**JOB NO 10**

pipeline {

agent any

stages {

stage('Hello') {

steps {

echo "Hello world"

}

}

}

post{

always{

mail to: "ashokitschool@gmail.com",

subject: "Test Email",

body: "Test"

}

}

}

**SONARQUBE**

**Sonar Qube**

* Sonar Qube is Continuous Code Quality Checking Tool
* We can do Code Review using Sonar Qube tool

**What is Code Coverage & Code Review?**

Code Coverage: **How many lines of source code is tested by unit test cases**

Note: **Industry standard Code Coverage is 80 %**

Code Review: **Checking Coding Conventions**

* Sonar Qube is an open source software quality management tool
* It will continuosly analyze and measures quality of the source code
* It will generate code review report in html format / pdf format
* It is a web based tool and it supports 29 Programming Languages
* It will support multi OS platform
* It will support multiple databases (MySQL, Oracle, SQL Server, PostGres SQL...)
* It supports multiple browsers
* Sonar Qube will identify below category of issues in project source code

**1) Duplicate Code**

**2) Coding Standards**

**3) Unit Tests**

**4) Code Coverage**

**4) Complex Code**

**5) Commented Code**

**6) Potential Bugs**

* Initially Sonar Qube was developed only for Java Projects
* Today Sonar Qube is supporting for 29 Languages

**Open source:** Java, Java Script, C#, Web Technologies (HTML, JSP), XML, Python, Groovy, PHP, Puppet etc...

**Commercial:** ABAP, C, C++, COBOL, PL/SQL, Visual Basic, VB.Net, Swift etc..

**Environment Setup**

Java is the pre-requisite software

7.6 --> Java 1.8v

7.8 - 8.x --> Java 11v

Note: We can check this compatibility in official sonar website

**Hardware Requirements**

**Minimum RAM: 2 GB**

**t2.micro ---> 1 GB RAM**

**t2.medium ---> 4 GB RAM**

**Sonar Qube Architecture**

1) SonarQube Scanner

(it will take source code as input and it will generate report)

2) SonarQube Server

SonarQube Server contains 3 components

1) Compute Engine

2) Database

3) WebServer

4) Search Engine

**Compute Engine**

- Vulnerabilities

- Bugs

- Code Smells

Note: Compute Engine will store report into H2 DB (we can configure external db also)

**WebServer** (it will display report in webserver)

Note: Developers will see sonar report and will fix the issues

**Search Engine** (it will give search results)

**Sonarqube set up**

Create EC2 instance with 4 GB RAM (t2.medium)

Connect with EC2 instance

Check space (free -h)

* $ sudo su
* $ yum install wget -y
* $ cd /opt
* $ sudo yum install java-1.8.0-openjdk
* $ java -version
* $ yum install wget unzip -y
* $ wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-7.8.zip
* $ unzip sonarqube-7.8.zip

**Note: SonarQube server will not run with root user**

**Create new user**

* $ useradd sonar
* $ visudo

**Configure sudo user without pwd**

* sonar ALL=(ALL) NOPASSWD: ALL

**Change ownership for sonar folder**

* $ chown -R sonar:sonar /opt/sonarqube-7.8/
* $ chmod -R 775 /opt/sonarqube-7.8
* $ su - sonar (switching to sonar user)

**Goto bin directory then goto linux directory and run sonar server**

* $ sh sonar.sh start

**Check sonar server status**

* $ sh sonar.sh status

**Note: Sonar Server runs on 9000 port number by default**

Enable port number in EC2 VM - Security Group

Access Sonar Server in Browser

URL: http://VM-IP:9000/

Note: We can change sonar port at this location (/opt/sonarqube-7.8/conf)

**Default Credentials of Sonar User is admin & admin**

After login, we can go to Security and we can enable Force Authentication.

Note: Once your work got completed then stops your EC2 instance because we have t2.medium so bill will be generated**.**

**Start sonar server**

**Go to sonar server bin folder**

**Switch to sonar user**

* $ sh sonar.sh start

**Note: Search Engine can't run with root user**

* $ sh sonar.sh status

**Note: If sonar not started, then go to log file and see**

* $ sudo rum -rf /opt/sonar-folder/temp/
* $ cd ../bin/
* $ sh sonar.sh start
* $ sh sonar.sh status

**Access sonar server in browser and login into that**

**After shut down when we want to connect again then use following commands**

* $ sudo su
* $ su - sonar
* $ cd /opt
* $ /sonarqube-7.8/bin/linux-x86-64
* $ sh sonar.sh start

**Integrate Sonar server with Java Maven App**

* Configure Sonar Properties under <properties/> tag in pom.xml

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

**<sonar.login>username</sonar.login>**

**<sonar.password>pwd</sonar.password>**

* Go to project pom.xml file location and execute below goal

**$ mvn sonar:sonar**

* After build success, goto sonar dashboard and verify that

**Note: Instead of username and pwd we can configure token also in “pom.xml”**

**Working with sonar token**

* Goto sonar-> Login -> click on profile -> Adminstrator -> Myaccount -> Security -> Generate Token
* Copy the token and configure that token in pom.xml

**<sonar.login>token</sonar.login>**

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

**<sonar.login>d0a49d314f5d169eccce35b158eb166e238f3fae</sonar.login>**

* Then build the project using "mvn sonar:sonar" goal
* Clone project repository

**git clone** [**https://github.com/ashokitschool/SB-REST-H2-DB-APP**](https://github.com/ashokitschool/SB-REST-H2-DB-APP)

**Quality Profile**

* For each language sonar qube provided one quality profile with set of rules
* Quality Profile means set of rules to perform code review
* We can create our own quality profile based on project requirement
* Create One Quality Profile

**- Name: SBI\_Project**

**- Language: Java**

**- Parent: None**

**Note: We can make our quality profile as default one then it will be applicable for all the projects which gets reviewed under this sonar server.**

**Note: If we have any common ruleset for all projects then we can create one quality profile and we can use that as parent quality profile for other projects.**

* We can configure quality profile to specific project

**- click on project name**

**- Go to administration**

**- Click on quality profile**

**- Select profile required**

**Quality Gate**

* Quality Gate represents set of metric to identify project quality is Passed or Failed
* Every Project Quality Gate should be passed
* In Sonar We have default Quality Gate
* If required, we can create our own Quality Gate also

**Note:**

* **If project quality gate is failed then we should not accept that code for deployment.**
* **If project is having Sonar issues then development team is responsible to fix those issues**
* **As a DevOps engineer, we will perform Code Review and we will send Code Review report to Development team (we will send sonar server URL to development team)**

**Sonar Server with Jenkins Integration**

**Pre-Requisites**

1) Sonar Qube Server

2) Jenkins Server

* On SonarQube Server Generate a Token
* On Jenkins Server

- install apache maven

- Install Sonar Plugin

- Configure SonarServer with Token

- Install Sonar Scanner

- Run Jenkins Pipeline Job

* Connect to Jenkins server VM using mobaxterm and start sonar server
* Access Sonar Server in browser & generate token
* Connect to Jenkins Server using MobaXterm
* Execute below commands in Jenkins Server VM

**$ sudo su**

**$ cd /opt**

**$ wget https://mirrors.estointernet.in/apache/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz**

**$ tar -xvf apache-maven-3.6.3-bin.tar.gz**

Manage Jenkins -> Plugins -> Available **-> Sonar Qube Scanner Plugin ->** Install it

Manage Jenkins **-> Configure System -> Sonar Qube Servers -> Add Sonar Qube Server**

- Name: Sonar-Server-7.8

- Server URL: http://52.66.247.11:9000/ (Give your sonar server url here)

- Add Sonar Server Token

**(Token we should add as secret text)** 31258f2c2e33b9826f4de365742e0d32543377ac

(Save it)

Manage Jenkins -> Global Tools Configuration **- > SonarQube Scanner**

**- Name: Sonar-Scanner-4.7**

**- Select Sonar Version ( SonarQube Scanner - 4.7)**

**(Save it)**

Goto jenkins dashboard -> click on 'New Item -> Enter Item name -> Select Pipeline -> Click on Next and Enter below pipeline script in given text box

**pipeline{**

agent any

environment {

PATH = "$PATH:/opt/apache-maven-3.6.3/bin"

}

stages{

stage('GetCode'){

steps{

git ''https://github.com/ashokitschool/maven-web-app.git"

}

}

stage('Build'){

steps{

sh 'mvn clean package'

}

}

stage('SonarQube analysis') {

// def scannerHome = tool 'SonarScanner 4.0';

steps{

withSonarQubeEnv('Sonar-Server-7.8') {

// If you have configured more than one global server connection, you can specify its name

// sh "${scannerHome}/bin/sonar-scanner"

sh "mvn sonar:sonar"

}

}

}

}

}

**MY SCRIPT**

**pipeline{**

agent any

environment {

PATH = "$PATH:/opt/apache-maven-3.6.3/bin"

}

stages{

stage('GetCode'){

steps{

git branch: 'main', credentialsId: 'f6dad8a3-84e7-45b1-9d9f-8a23cb51540e', url: 'https://github.com/Saisunil3012/maven-web-app.git'

}

}

stage('Build'){

steps{

sh 'mvn clean package'

}

}

stage('SonarQube analysis') {

steps{

withSonarQubeEnv('Sonar-Server-7.8') {

sh "mvn sonar:sonar"

}

}

}

}

}

**ANSIBLE**

**Ansible Tutorial**

1) What is Ansible?

2) Configuration Management

3) Push Based v/s Pull Based

4) How to install Ansible

5) Host Inventory

6) Ansible Modules

7) YAML

8) Playbooks

9) Hands On

10) Conclusion

**Configuration Management**

* It is a method through which we automate admin tasks.
* Configuration management tool turns your code into infrastructure.
* So your code would be testable, repeatable and versionable.
* Infrastructure refers to the composite of

Software

Network

People

Process

**ANSIBLE**

* Ansible is one among the DevOps configuration management tools which is famous for its simplicity.
* It is open source software developed by Michael DeHaan and its ownership is on **RedHat**
* Ansible is an open source IT Configuration Management, Deployment & Orchestration tool.
* This tool is very simple to use yet powerful enough to automate complex multi-tier IT application environments.
* Ansible is an automation tool that provides a way to define **Infrastructure as code.**
* **Infrastructure as code (IAC)** simply means that managing infrastructure by writing code rather than using manual processes.
* The best part is that you don’t even need to know the commands used to accomplish a particular task.
* You just need to specify what state you want the system to be in and Ansible will take care of it.
* The main components of Ansible are **playbooks, configuration management and deployment.**
* **Ansible uses playbooks to automate deploy, manage, build, test and configure anything**
* Ansible was written in **Python.**

**Ansible Features**

* Ansible manages machines in an **agent-less manner using SSH**
* Built on top of Python and hence provides a lot of Python's functionality
* **YAML based playbooks**
* Uses SSH for secure connections
* Follows push based architecture for sending configuration related notifications

**Push Based V/s Pull Based**

* Tools like **Puppet and Chef are pull based**
* Agents on the server periodically checks for the configuration information from central server (Master)
* **Ansible is push based**
* Central server pushes the configuration information on target servers.

**What Ansible can do?**

1) Configuration Management

2) App Deployment

3) Continous Delivery

**How Ansible works?**

* Ansible works by connecting to your nodes and pushing out a small program called Ansible modules to them.
* Then Ansible executed these modules and removed them after finished. The library of modules can reside on any machine, and there are no daemons, servers, or databases required.
* The Management Node is the controlling node that controls the entire execution of the playbook.
* The inventory file provides the list of hosts where the Ansible modules need to be run.
* The Management Node makes an SSH connection and executes the small modules on the host’s machine and install the software.
* Ansible removes the modules once those are installed so expertly.
* It connects to the host machine executes the instructions, and if it is successfully installed, then remove that code in which one was copied on the host machine.
* Ansible basically consists of three components
* Ansible requires the following components in order to automate Network Infrastructure.

1) Controlling Nodes

2) Managed Nodes

3) Ansible Playbook

**Controlling Nodes**

These Network Devices are referred to as the Managed Nodes.

**Managed Nodes**

Managed Nodes are stored in the hosts file for Ansible automation.

**Ansible Playbook**

Ansible **Playbooks** are expressed in **YAML** format and serve as the repository for the various tasks that will be executed on the Managed Nodes (hosts).

Playbooks are a collection of tasks that will be run on one or more hosts.

**Inventory file**

* Ansible's inventory hosts file is used to list and group your servers.
* Its default locaton is /etc/ansible/hosts

**Note: In inventory file we can mention IP address or Hostnames also**

**Few Important Points about Inventory File**

Comments begins with '#' character

Blank lines are ignore

Groups of hosts are delimited by '[header]' elements

You can enter hostnames or ip addresses

**A hostname/ip can be a member of multiple groups**

**Sample Inventory File**

#Blank lines are ignore

#Ungrouped hosts are specifying before any group headers like below

192.168.122.1

192.168.122.2

192.168.122.3

[webservers]

192.168.122.1

#192.168.122.2

192.168.122.3

[dbserver]

192.168.122.1

192.168.122.2

ashokit-db1.com

ashokit-db2.com

**Ansible Setup**

Create 2 Red Hat Systems in AWS (Free Tier Eligible)

* 1 - Ansible System
* 1 - Host System

Connect to all the systems and create ansible user

* $ sudo useradd ansible
* $ sudo passwd ansible
* pwd
* confirm pwd
* $ sudo visudo
* ansible ALL=(ALL) NOPASSWD: ALL
* $ sudo vi /etc/ssh/sshd\_config

comment PasswordAuthentication no

un-comment PasswordAuthentication yes

Restart the server

* $ sudo service sshd restart

**Note: Do the above steps in all the 2 machines**

**Install Ansible in Control Node**

Switch to Ansible user

* $ sudo su ansible

Install Python

* $ sudo yum install python3 -y

Check python version

* $ python --version (it will fail bcz we used python3)
* $ python3 --version

Update python alternatives

* $ sudo alternatives --set python /usr/bin/python3

Check python version

* $ python --version

Install PIP (It is a python package manager)

* $ sudo yum -y install python3-pip

Install Ansible using Python PIP

* $ pip3 install ansible –user

Verify ansible version

* $ ansible --version

Create ansible folder under /etc

* $ sudo mkdir /etc/ansible

Create ansible.cfg file under /etc/ansible and paste complete content from below git link.

Open: **https://raw.githubusercontent.com/ansible/ansible/devel/examples/ansible.cfg**

Copy the content and paste it in ansible.cfg file with below command

* $ sudo vi /etc/ansible/ansible.cfg

Create hosts file under /etc/ansible. Sample content can found in below git link

Open: **https://raw.githubusercontent.com/ansible/ansible/devel/examples/hosts**

Copy the content and paste it in hosts file with below command

* $ sudo vi /etc/ansible/hosts

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* With this Ansible setup completed \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Generate SSH Key in Control Node and Copy SSH key into Host Node**

1) Now generate SSH key in Ansible Server (Control Node):

* $ sudo su ansible
* $ ssh-keygen

2) Copy it to Host servers as ansible user:

$ ssh-copy-id ansible@<Host-Private-IP>

Ex : $ ssh-copy-id ansible@172.31.43.23

**Note:** Repeat below command by updating HOST IP for all the HOST Servers.

3) Update Host Inventory in Ansible Server to add host servers details.

4) goto the location /etc/ansible/hosts

* $ vi /etc/ansible/hosts

HOST-NODE-IP <172.31.43.23>

5) Use ping module to test Ansible and after successful run you can see the below

output.

* $ ansible all -m ping

**Ansible AD-HOC Commands**

Switch to ansible user and run ansible ad-hoc commands

* $ sudo su ansible

To run any ansible command we will follow below syntax

**# ansible [ all / groupName / HostName / IP ] -m <<Module Name>> -a <<args>>**

Note: Here -m is the module name and -a is the arguments to module.

Example:

It will display date from all host machines.

* **$ ansible all -m shell -a date**

It will display uptime from all host machines.

* **$ ansible all -m shell -a uptime**

There are two default groups, all and ungrouped. All contains every host. Ungrouped contains all hosts that don’t have another group

It will display the all the modules available in Ansible.

* **$ ansible-doc -l**

To display particular module information

* **$ ansible-doc <moduleName>**

To display shell module information

* **$ ansible-doc shell**

To display details of copy module

* **$ ansible-doc -l | grep "copy"**

It will display more information about yum module

* **$ ansible-doc yum**

**Ping Module**

It will ping all the servers which you have mentioned in **inventory file (/etc/ansible/hosts)**

* **$ ansible all -m ping**

It will display the output in single line.

* **$ ansible all -m ping -o**

Redhat release of all the machines

* **$ ansible all -m shell -a 'cat /etc/\*release'**

Kind of mount on all the machines

* **$ ansible all -m shell -a 'mount'**

Check the service status on all the machines

* **$ ansible all -b -m shell -a 'service sshd status'**

Here it will check the disk space use for all the nodes which are from dbservers group

* **$ ansible dbservers -a "df -h"**

Here it will check the disk space use for all the nodes which are from webservers group

* **$ ansible webservers -a "free -m"**

Here it will display date from from webservers group

* **$ ansible webservers -a "date"**

**Yum Module**

It will install vim package in all node machine which you have mentioned in host inventory file.

* **$ ansible all -b -m yum -a "name=vim"**

Check git version in all machines

* **$ ansible all -m shell -a "git --version"**

To install git client in all node machines

* **$ ansible all -m shell -b -a "yum install git -y"**

To installl git only in webserver nodes

* **$ ansible webservers -m shell -b -a "yum install git -y"**

To install webserver only in particular machine

* **$ ansible 172.1921.1.0 -m shell -b -a "yum install git -y"**

$ ansible all -m shell -b -a "name=git state=present"

$ ansible all -m shell -b -a "name=git state=latest"

$ ansible all -m shell -b -a "name=git state=absent"

**present: install**

**latest: update to latest**

**absent: un-install**

To install any software in **ubuntu** server then we should use **apt** package manager

* **$ ansible all -m apt -a "name="git state="present"**

To install httpd package in all node machines

* **$ ansible all -b -m yum -a "name=httpd state=present"**

**Note: Here state=present, is not a mandatory, it is by default.**

To update httpd package in all node machines.

* **$ ansible all -b -m yum -a "name=httpd state=latest"**

To remove httpd package in all node machines.

* **$ ansible all -b -m yum -a "name=httpd state=absent"**
* **$ ansible all -m copy -a "src="index.html dest=/var/www/html/index.html"**

Start httpd service

* **$ ansible all -b -m service -a "name=httpd state=started"**
* **$ ansible all -b -m shell -a "service httpd start"**

**Note: For privilege escalations we can use -b option**

Q) Irrespective of underlying OS which module we can use to manage packages (softwares) using package manager in Ansible?

Ans) Ansible introduced "**package manager**" to work with underlying package manager

**YAML (Yet another Markup Language)**

* YAML Ain’t markup language
* We can make use of this language to store data and configuration in a human-readable format.
* YAML files will have **.yml as an extension**
* Official Website: https://yaml.org/

**Key-Value Pair**

Fruit: Apple

Vegetable: Carrot

Liquid: Water

Meet: Chicken

**Array/List**

Fruits:

- Orange

- Apple

- Banana

- Guava

**Vegetables:**

- Carrot

- Cauliflower

- Tomoto

Here - dash indicate the element of any array.

**PLAYBOOKS**

* Playbook is a **single YAML file**, containing one or more ‘plays’ in a list.
* Plays are ordered sets of tasks to execute against host servers from your inventory file.
* Play defines a set of activities (tasks) to be run on hosts.
* Task is an action to be perform on the host

Examples are

a) Execute a command

b) Run a shell script

c) Install a package

d) Shutdown/Restart the hosts

**Note:** Playbooks start with the YAML three dashes (---) and end with …

Playbook contains the following sections:

1) Every playbook starts with 3 hyphens ‘- - - ‘

2) Host section – Defines the target machines on which the playbook should run. This is based on the Ansible inventory file.

3) Variable section – This is optional and can declare all the variables needed in the playbook. We will look at some examples as well.

4) Tasks section – This section lists out all the tasks that should be executed on the target machine. It specifies the use of Modules. Every task has a name which is a small description of what the task will do and will be listed while the playbook is run.

**Playbook to Ping All Host Nodes**

---

- hosts: all

tasks:

- name : Test connection

ping:

remote\_user: ansible

hosts: The tasks will be executing in specified group of servers.

name: which is the task name that will appear in your terminal when you run the playbook.

remote\_user: This parameter was formerly called just user. It was renamed in Ansible 1.4 to make it more distinguishable from the user module (used to create users on remote systems).

**Note:** Remote users can also be defined per task.

**Run the playbook Using below command**

* **$ ansible-playbook <<Playbbok file name>>**

It will run the playbook.yml playbook in verbose

* **$ ansible-playbook playbook.yml -v**
* **$ ansible-playbook playbook.yml -vv**
* **$ ansible-playbook playbook.yml -vvv**

It will provide help on ansible playbook command

* **$ ansible-playbook --help**

It will check the syntax of a playbook

* **$ ansible-playbook playbook.yml --syntax-check**

It will do in dry run.

* **$ ansible-playbook playbook.yml --check**

It will display the which hosts would be effected by a playbook before run

* **$ ansible-playbook playbook.yml --list-hosts**

It execute one-step-at-a-time, confirm each task before running with (N)o/(y)es/(c)ontinue

* **$ ansible-playbook playbook.yml --step**

**Install HTTPD + copy index.html + Start Service**

---

- hosts: all

become: true

tasks:

- name: Install Httpd

yum:

name: httpd

state: present

- name: Copy index.html

copy:

src: index.html

dest: /var/www/html/index.html

- name: Start Http Server

service:

name: httpd

state: started

**VARIABLES**

- hosts: all

become: true

tasks:

- name: Install Httpd

yum:

name: "{{package\_name}}"

state: present

- name: Copy index.html

copy:

src: index.html

dest: /var/www/html/index.html

- name: Start Http Server

service:

name: "{{package\_name}}"

state: started

**We can pass variable value in run time like below**

* **$ ansible-playbook filename.yml --extra-vars package\_name=httpd**

**we can define variables with in the playbook**

---

- hosts: all

become: true

vars:

package\_name: httpd

tasks:

- name: Install Httpd

yum:

name: "{{package\_name}}"

state: present

- name: Copy index.html

template:

src: index.html

dest: /var/www/html/index.html

- name: Start Http Server

service:

name: "{{package\_name}}"

state: started

**Group Variables**

For webservers i want to install git-2.1

For dbservers i want to install git-2.3

We can achieve these using group variables

**group vars files should be created at host inventory location**

host-inventory location : **/etc/ansible**

group\_vars/all.yml

group\_vars/<groupName>.yml

Ex: **group\_vars/webservers.yml**

**group\_vars/dbservers.yml**

**group\_vars/all.yml**

/etc/ansible/group\_vars/all.yml

Create mkdir /etc/ansible/group\_vars

vi /etc/ansible/group\_vars/all.yml

test: abc

vi /etc/ansible/group\_vars/appservers.yml

test: xyz

Note: as we have group vars file for appservers it will consider xyz for appservers group and for remaining servers group it will take all.yml file value

**Host variables**

Server specific variables

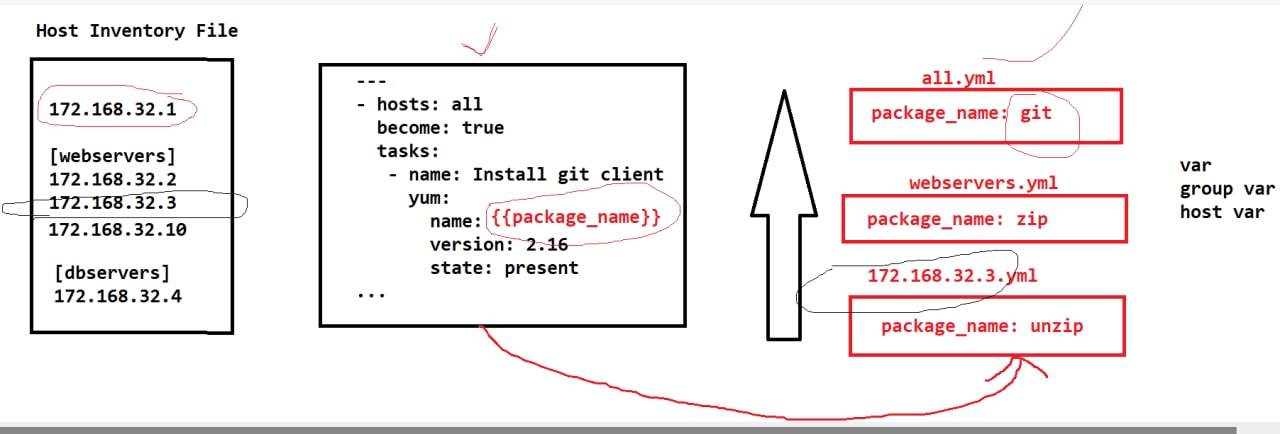
In one group we will have multiple servers

For every host if we want separate variables then we should go for host variables

-> mkdir /etc/ansible/host\_vars

-> create a file with host name or ip

-> vi /etc/ansible/host\_vars/172.138.1.1.yml



**ANSIBLE VAULT**

ansible-vault create <filename>.yml

ansible-vault encrypt <filename>.yml

ansible-vault view <filename>.yml

ansible-vault edit <filename>.yml

ansible-vault decrypt <filename>.yml

ansible-vault rekey <filename>.yml

* To encrypt a playbook we need to set one vault password
* While executing playbook we need to pass vault password
* **$ ansible-playbook <filename>.yml --ask-vault-pass**
* You can store vault password in a file and you can give that file as input to execute playbook
* **$ vi valutpass**
* **$ ansible-playbook filename.yml --vault-password-file=~/vaultpass**

We can see encrypted file in human readable format

* **$ ansible-vault view /etc/ansible/group\_vars/all.yml**

We can edit encrypted file in human readable format

* **$ ansible-vault edit /etc/ansible/group\_vars/all.yml**

We can decrypt the file

* **$ ansible-vault edit /etc/ansible/group\_vars/all.yml**

To update vault password we can use rekey

* **$ ansible-vault rekey /etc/ansible/group\_vars/all.yml**

**HANDLERS AND TAGS**

* Sometimes you want a task to run only when a change is made on a machine. For example, you may want to restart a service if a task updates the configuration of that service, but not if the configuration is unchanged. Ansible uses handlers to address this use case. **Handlers are tasks that only run when notified.**
* **If you have a large playbook, it may become useful to be able to run only a specific part of it rather than running everything in the playbook**. Ansible supports a “tags:” attribute for this reason.
* Tags can be applied tomany structures in Ansible, but its simplest use is with individual tasks.

---

- hosts: all

become: true

gather\_facts: no

vars:

package\_name: httpd

tasks:

- name: install httpd

yum:

name: "{{package\_name}}"

state: present

- name: Copy index.html

copy:

src: index.html

dest: /var/www/html/

notify:

Start Httpd Server

tags:

- copy

- name: debug message

msg: Copy completed

tags:

- debug

- install

handlers:

- name: Start Httpd Server

service:

name: {{package\_name}}

state: started

To display all tags available in playbook

* **$ ansible-playbook filename.yml --list-tags**
* **$ ansible-playbook filename.yml --tags "install"**
* **$ ansible-playbook filename.yml --tags "debug,copy"**
* **$ ansible-playbook filename.yml --skip-tags**

**Installing Multiple Softwares**

- hosts: all

tasks:

- name: install softwares

yum:

name: "{{item}}"

state: present

with\_items:

- wget

- zip

- unzip

**Another approach**

- hosts: all

tasks:

- name: install softwares

yum:

name: ['wget', 'zip', 'unzip']

state: present

Scope of the variable is within the playbook. We can refer vars in the tasks or templates which we are using the playbook.

- hosts: all

tasks:

- name: vars demo

debug:

msg: "var test value: {{test}}"

- name: copy file

template:

src: test.txt

dest: /tmp/test.txt

- name: cat the file

shell: "cat /tmp/test.txt"

register: output

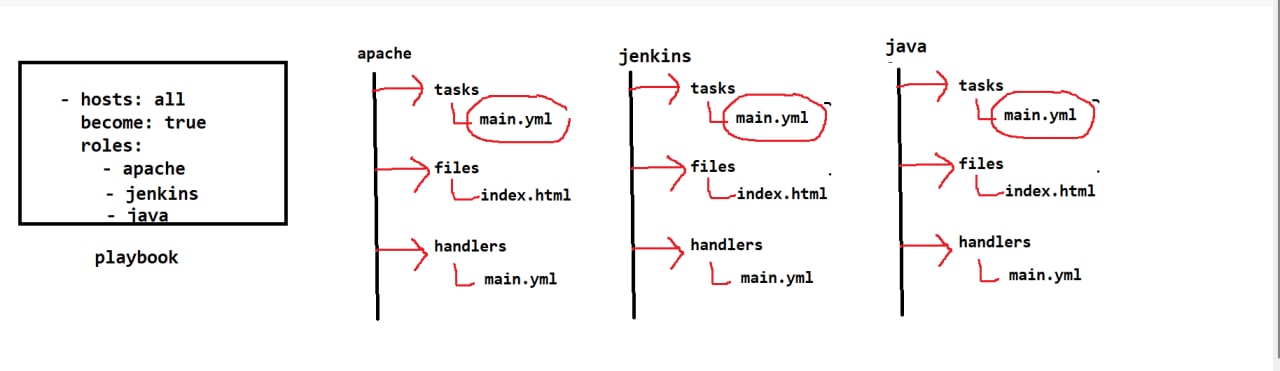
- name: print file content

debug:

msg: "File Content {{output.stdout}}"

**ANSIBLE ROLES**

* Roles are a level of abstraction on top of tasks and playbooks that let you structure your Ansible configuration in a modular and reusable format
* **As you add more and more functionality to your playbooks, they can become difficult to maintain**
* **Roles allow you to break down a complex playbook into separate, smaller chunks** that can be coordinated by a central entry point.



# Sample playbook with Role

---

- hosts: all

become: true

roles:

- apache

1. Ansible roles are consists of many playbooks, which is similar to modules in puppet and cook books in chef. We term the same in ansible as roles.

2. Roles are a way to group multiple tasks together into one container to do the automation in very effective manner with clean directory structures.

3. Roles are set of tasks and additional files for a certain role which allow you to break up the configurations.

4. It can be easily reuse the codes by anyone if the role is suitable to someone.

5. It can be easily modify and will reduce the syntax errors.

**How do we create Ansible Roles?**

* To create an Ansible role, use **"ansible-galaxy"** command which has the templates to create it.
* **$ sudo su ansible**
* **$ cd /home/ansible**
* **$ mkdir roles**
* **$ ansible-galaxy init roles/apache**
* Where, ansible-glaxy is the command to create the roles using the templates.
* init is to initiliaze the role.
* apache is the name of the role
* List out the directory created under roles/apache
* **$ sudo yum install tree**
* **$ tree roles/apache/**

We have got the clean directory structure with the ansible-galaxy command. Each directory must contain a main.yml file, which contains the relevant content.

**Directory Structure:**

tasks – contains the main list of tasks to be executed by the role.

handlers – contains handlers, which may be used by this role or even anywhere outside this role.

defaults – default variables for the role.

vars – other variables for the role. Vars has the higher priority than defaults.

files – contains files required to transfer or deployed to the target machines via this role.

templates – contains templates which can be deployed via this role.

meta – defines some data / information about this role (author, dependency, versions, examples, etc,.)

**Lets take an example to create a role for Apache Web server.**

Below is a sample playbook codes to deploy Apache web server. Lets convert this playbook codes into Ansible role.

- hosts: all

become: true

tasks:

- name: Install Httpd

yum:

name: httpd

state: present

- name: Copy index.html

template:

src: index.html

dest: /var/www/html/index.html

- name: Start Http Server

service:

name: httpd

state: started

First, move on to the Ansible roles directory and start editing the yml files.

**$ cd roles/apache**

**1. Tasks**

Edit main.yml available in the tasks folder to define the tasks to be executed.

**$ vi tasks/main.yml**

---

# tasks file for roles/apache

- name: install httpd

yum:

name: httpd

state: present

- name: Copy index.html

copy:

src=index.html

dest=/var/www/html/

notify:

- restart apache

- name: restart apache

service:

name=httpd

state=restarted

**2. Files**

Copy required files into files directory or create index.html file with content

**3. Handlers**

Edit handlers main.yml to restart the server when there is a change. Because we have already defined it in the tasks with notify option. Use the same name “restart apache” within the main.yml file as below.

**$ cat handlers/main.yml**

- name: restart apache

service:

name: httpd

state: restarted

We have got all the required files for Apache roles. Lets apply this role into the ansible playbook “runsetup.yml” as below to deploy it on the client nodes.

**$ cat /home/ansible/runsetup.yml**

---

- hosts: all

become: true

roles:

- apache

...

Execute playbook which contains apache role

**$ ansible-playbook runsetup.yml**

If you have created **multiple roles**, you can use the below format to add them in the playbook

---

- hosts: all

become: true

roles:

**- apache**

**- jenkins**

**- java**

**- maven**

**- sonar**

What We Covered in Ansible?

1) What is Configuration Management?

2) What is Ansible?

3) Advantages of Ansible

4) Push Based V/s Pull Based Mechanism

5) Ansible Installation

6) Ansible Architecture

7) Host Inventory File

8) Host Groups in Inventory

9) Ansible Ad Hoc Commands

10) YAML

11) Working with YAML in VS Code IDE

12) Playbook Introduction

13) Playbook Commands

14) Writing Playbooks

15) Variables

- Runtime variables

- Playbook variables

- Group variables

- Host variables

16) Handlers

17) Tags

18) Roles

19) Ansible Vault

20) Playbook for Multiple OS Family Based Hosts

**ANSIBLE TOWER**

* Ansible Tower is a **web based GUI** tool which is used to manage infrastructure configurations.
* We can centralize our infrastructure configurations using this UI
* We can do job scheduling
* It is providing graphical inventory management

**Ansible Tower features**

1) Clean dashboard

2) Manage Iventory Dynamically

3) Realtime Job Status

4) Job Scheduling

5) Integrate Internal Notification

6) Rolebase access control (RABAC)

7) Audit Job and Tower Resource

8) Store Credentials safely

9) REST API Integration

10) Self Service UI

**Pre-Requisites to Install Ansible Tower**

* Red Hat Enterprise Linux 6 64-bit
* 20 GB Hard Disk
* 4 GB Ram
* Note: m4.large instance required

**Step-1: Install Ansible Software**

1. Update package manager

* **$ sudo yum -y update**

2. Install epel package manager

* **$ sudo dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm -y**

3. Check epl repositories install

* **$ sudo dnf repolist epel**

4. Ansible tower uses ansible playbook to deploy so we should install Ansible first

* **$ sudo yum -y install ansible vim curl**

**Step-2: Download Ansible Tower Archieve Folder**

* **$ mkdir /tmp/tower && cd /tmp/tower**
* **$ curl -k -O https://releases.ansible.com/ansible-tower/setup/ansible-tower-setup-latest.tar.gz**
* **$ ls**

Extract tar file

* **$ tar xvf <tar-file>**

Go to extracted folder and update admin\_pwd and postgres\_db pwd in inventory file

* **$ vim inventory**

Run the script file to run playbook

* **$ sudo ./setup.sh**

**Note: It will take 2 to 3 minutes**

check ansible tower status

* **$ sudo ansible-tower-service status**

Start Ansible Tower

* **$ sudo ansible-tower-service start**

**Note: Open 443 port because Ansible Tower Running on HTTPS**

Access EC2 VM IP in browser and login

default uname : admin

pwd : we have given in inventory file (ashokit)

**Note: This is enterprise product so we need license. We can get trail version**

Create account in red hat and request for license

www.redhat.com/en/technologies/management/ansible/try-it

Click on 'Request license' -> Complete process

(https://access.redhat.com/management/subscriptions)

Loing with redhat account credentials

**Configure 2 Host Nodes IN ec2**

1) Login into tower

2) Create Groups and add hosts

3) Create credentials with Machine and provide ssh private key

Create Project

Give SCM as git repo url of playbooks

**TERRAFORM**

* **Terraform is an open source s/w created by HashiCorp and written in Go programming language**
* Terraform is an **infrastructure as code** (IaaC) software tool,
* Infrastructure as code is the process of managing infrastructure in a file or files rather than manually configuring resources in a user interface (UI)
* In Terraform resources are nothing but Virtual machines, Elastic IP, Security Groups, Network interfaces
* Terraform Code is written in the **HashiCorp Configuration language (HCL)** in files with the extension **.tf**
* Terraform allows users to use HashiCorp Configuration Language (HCL) to create the files containing definitions of the their desired resources
* Terraform supports all most all cloud providers (AWS, AZURE, GCP, Openstack etc.). To automate **infrastructure creation we will use Terraform**.

**Terraform v/s Cloud Formation**

|  |  |
| --- | --- |
| **Terraform** | **Cloud Formation** |
| Terraform developed by HashiCorp | CloudFormation developed by AWS |
| Terraform supports many cloud providers | Cloud Formation will support only in AWS |
| Terraform uses HashiCorp configuration language (HCL) which built by HashiCorp. It is fully compatible with JSON. | AWS Cloud Formation utilizes either JSON or YAML. Cloud formation has a limit of 51,000 bytes for the template body itself. |

**Terraform V/s Ansible**

|  |  |
| --- | --- |
| **Terraform** | **Ansible** |
| Terraform developed by HashiCorp | Ansible is also an open source software |
| Terraform is an infrastructure as a Code, which means they are designed to provision the servers themselves. | Ansible is a configuration management tool. This means ansible designed to install and manage software on existing servers. |
| Terraform is ideal for creating, managing and improving infrastructure. | Ansible is ideal for software provisioning, application deployment and configuration management. |

**Pre-Requisites**

1) Any Cloud Provider (AWS, Azure, GCP, Openstack etc.)

2) IAM User account (Secret Key and Access Key)

3) IAM User should have AmazonEc2FullAccess

1) Create EC2 instance (RED HAT Linux)

2) Connect to EC2 VM using Mobaxterm

3) Swith to root user

**$ sudo su -**

4) Install unzip software

**$ yum install wget unzip vim -y**

5) Download Terraform Software (https://www.terraform.io/downloads)

**$ sudo yum install -y yum-utils**

**$ sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/RHEL/hashicorp.repo**

**$ sudo yum -y install terraform**

6) Check Terraform Version

**$ terraform -v**

7) Create IAM User with Programmatic Access and Provide 'AmazonEc2FullAccess'

8) Download Secret Key and Access Key

9) Write Terraform Script

**$ mkdir terraformscript**

**$ cd terraformscripts**

**$ vi FirstTFScript.tf**

provider "aws" {

region = "ap-south-1"

access\_key = "your-access-key"

secret\_key = "your-secret-key"

}

resource "aws\_instance" "AWSServer" {

ami = "ami-05c8ca4485f8b138a"

instance\_type = "t2.micro"

key\_name = "linux"

security\_groups = ["launch-wizard-1"]

tags = {

Name = "Terrafrom Server"

}

}

10) Initialize Terraform using init command

**$ terraform init**

11) Format your script (indent spaces)

**$ terraform fmt**

12) Validate Your Script

**$ terraform validate**

13) Create Execution Plan For Your Script

**$ terraform plan**

14) Create Infrastructure

**$ terraform apply**

**$ terraform -auto-aprove**

**Note:** When the script got executed it will store that state in a file. If we execute script again it will not create. If you delete that state file and execute script again then it will create it.

15) Destory Infrastructure

**$ terraform destroy -auto-aprove**

**In first script we kept provider and resources info in single script file. We can keep provider and resources information in separate files**

Ex: proder.tf & main.tf

**Script to create multiple Ec2 instances**

provider "aws" {

region = "ap-south-1"

access\_key = "your-access-key"

secret\_key = "your-secret-key"

}

resource "aws\_instance" "AWSVM\_Server" {

count = "3"

ami = "ami-05c8ca4485f8b138a"

instance\_type = "t2.micro"

key\_name = "linux"

security\_groups = ["ashokit\_security\_group"]

tags = {

Name = "REDHAT-EC2-VM1"

}

}

**Note: Once it is created, then destory infrastructure using below command**

**$ terraform destroy -auto-approve**

Variables in TypeScript

We can maintain variables in seperate file

**$ vi vars.tf**

variable "ami"{

description="Amazon Machine Image value"

default = "ami-05c8ca4485f8b138a"

}

variable "instance\_type"{

description="Amazon Instance Type"

default = "t2.micro"

}

variable "instances\_count"{

description="Total No.of Instances"

default = "2"

}

Create main tf file using variables

**$ vi main.tf**

provider "aws" {

region = "ap-south-1"

access\_key = "AKIA4GJG"

secret\_key = "d04co9pmnE5xNj8XiubpUPhI"

}

resource "aws\_instance" "AWSServer" {

count="${var.instances\_count}"

ami = "${var.ami}"

instance\_type = "${var.instance\_type}"

key\_name = "linux"

security\_groups = ["launch-wizard-1"]

tags = {

Name = "EC2 VM - ${count.index}"

}

}

**Note: We can supply variables in runtime also**

Remove instances\_count variable from var.tf file and pass like below

**$ terraform apply -var instances\_count="2" -auto-aprove**

Comments in Terraform Script

# - single line comment

// - single line comment (java style)

/\* and \*/ - Multi line comments

**Dealing with Secret Key and Access Key**

We have configure secret\_key and access\_key in terraform script file. Instead of that we can configure them as environment variables.

**$ export AWS\_ACCESS\_KEY\_ID="AKIA4MGQ5UW7B"**

**$ export AWS\_SECRET\_ACCESS\_KEY="kzpje2T8S1lHsywFLTOI89XU7sRy8"**

To verify environment variables we can use echo command

**$ echo $AWS\_ACCESS\_KEY**

**$ echo $AWS\_SECRET\_ACCESS\_KEY**

Now remove credentials from terraform script and execute it.

**Note: We are setting provider credentials in terminal so these variables will be available for current session. If we want to set permanently add them in .bashrc file**

**Working with User Data**

It is used to execute script when instance launched for first time.

Create Userdata in one file

**$ vi installHttpd.sh**

#!/bin/bash

sudo su

yum install httpd -y

cd /var/www/html

echo "<html><h1>Welcome to Ashok IT...!!</h1></html>" > index.html

service httpd start

$ chmod u+x installHttpd.sh

create main scrit in main.tf file

vi main.tf

provider "aws" {

region = "ap-south-1"

access\_key = "your-access-key"

secret\_key = "your-secret-key"

}

resource "aws\_instance" "AWSServer" {

ami = "ami-05c8ca4485f8b138a"

instance\_type = "t2.micro"

key\_name = "linux"

security\_groups = ["ashokit\_security\_group"]

user\_data = "${file("installHttpd.sh")}"

tags = {

Name = "Terrafrom Server"

}

}

**Creating S3 bucket using Terraform script**

Add S3 policy for IAM user

Execute below terraform script to create s3 bucket in AWS

provider "aws"{

region = "ap-south-1"

access\_key = "your-access-key"

secret\_key = "your-secret-key"

}

resource "aws\_s3\_bucket" "s3bucketashokit"{

bucket = "s3bucketashokit"

acl="private"

versionining{

enabled = true

}

tags = {

Name = "S3 Bucket By Ashok"

}

}

**Create MySQL DB in AWS using Terraform**

Provider RDS access for IAM user

Execute below script to create MySQL DB in AWS cloud

provider "aws"{

region = "ap-south-1"

access\_key = "your-access-key"

secret\_key = "your-secret-key"

}

resource "aws\_db\_instance" "default" {

allocated\_storage = 100

engine = "mysql"

engine\_version = "5.7"

instance\_class = "db.t3.micro"

name = "mydb"

username = "foo"

password = "foobarbaz"

parameter\_group\_name = "default.mysql5.7"

skip\_final\_snapshot = true

}

**DOCKER**

**Application Development**

* Collection of programs is called as software project
* Software project contains several components

1) Front end components (User interface logic)

2) Backend components (Business Logic)

3) Database Components (Persistence Logic)

* In order to deploy our application in a machine we need to setup all the Softwares which are required to our application

Ex: OS, Java 1.8v, MYSQL DB, Tomcat Web Server 9.0v etc.....

* In Realtime project should be deployed into multiple environments for testing purpose

Ex : DEV, SIT, UAT, PILOT and PROD

* DEV env will be used by Developers to perform integration testing
* SIT env will be used by Testing team to test functionality of the application
* UAT env will be used by Client to test functionality of the application
* PILOT env means pre-production testing env
* PROD means live environment (It is used to deliver the project)
* To deploy application to these many enivornments we need to take of all the softwars required to run our application in all environments. It is very difficult task.

**Virtualization**

* Installing Multiple Guest Operating Systems in one Host Operating System
* **Hypervisior** S/w will be used to achieve this
* We need to install all the required softwares in HOST OS to run our application
* It is old technique to run the applications
* System performance will become slow in this process
* To overcome the problems of Virtualization we are going for Containerization concept

**Containerization**

* **It is used to package all the softwares and application code in one container for execution**
* Container will take care of everything which is required to run our application
* We can run the containers in Multiple Machines easily
* **Docker is containerization software**
* Using Docker **we will create container** for our application
* Using Docker **we will create image** for our application
* Docker images we can share easily to mulitple machines
* Using Docker image we can create docker container and we can execute it

**Conclusion**

Docker is containerization software

Docker will take care of application and application dependencies for execution

Deployments into multiple environments will become easy if we use Docker containers concept

**++++++++Install Docker in Amazon Linux++++++++++++**

* **$ sudo yum update -y**
* **$ sudo yum install docker -y**
* **$ sudo service docker start**

# add user to docker group by executing below command

* **$ sudo usermod -aG docker ec2-user**
* **$ docker info**

#Restart the session

* **$ exit**

Then press 'R' to restart the session (This is in MobaXterm)

**Docker Commands**

See docker info

* **$ docker info**

To see docker images execute below command

* **$ docker images**

Pulling hello-world docker image

* **$ docker pull hello-world**

See docker image

* **$ docker images**

Running hello-world docker image

* **$ docker run hello-world**

See docker container

* **docker ps -a**

To remove docker image

* **$ docker rmi <image\_id>**
* **$ docker rmi -f <image\_id>**

To remove docker container

* **$ docker rm <container\_id>**
* **$ docker rm -f <container\_id>**

To run docker container

* **$ docker run -d -p (Host machine port)80:80(container port) <image\_name>**
* **$ docker run --name=firstimage -d -p 80:80 nginx**

**Note: Create account in Docker Hub (https://hub.docker.com/)**

**Dockerfile**

Dockerfile is file which contains instructions to create an image which contains Docker Domain Specific Key Words to build image.

**DockerImage**

It's a package which contains everything (Softwares+ENV+Application Code) to run your application.

**DockerContainer**

Run time instance of an image. If you run docker image container will be created that's where our application (process) is running.

**DockerRepo/Registry**

We can store and share the docker images.

**Public Repo**

Docker hub is a public repository which contains all the open source softwares as a docker images. We can think of docker hub as play store for docker images.

**Private Repo**

(Nexus, JFrog, D.T.R(Docker Trusted Registory)),

**AWS ECR**

We can store and share the docker images with in our company network using private repo

**Docker Engine/Daemon/Host**

It's a software or program using which we can create images & contianers.

Docker is cross platform.

Docker CE

Docker CE will not be supported by Redhat.

Docker EE

Docker EE will be support most of the os including redhat.

**What is docker hub?**

It's a public repository for docker images.

You can think as play store for docker images.

Create docker file with below content

* **$ vi Dockerfile**

FROM ubuntu

RUN echo "Run One Updated"

RUN echo "RUN TWO"

CMD echo "Echo From Image"

CMD echo "Echo From Latest"

RUN echo "RUN Three"

Login into Dockerhub account

* **$ docker login**

Build docker image using docker file

* **$ docker build -t(filename) imageone . (dot represent search for current location)**

To save docker image in docker hub Tag is required

* **$ docker tag imageone ashokit/ imageone**

Push docker image

* **$ docker push ashokit/ imageone**

**Dockerfile**

* **Dockerfile contains instructions to build docker image**
* In Dockerfile we will use DSL (**Domain Specific Language**) keywords
* Docker engine will process Dockerfile instructions from top to bottom
* Below are the Dockerfile keywords

FROM

MAINTAINER

COPY

ADD

RUN

CMD

ENTRYPOINT

ENV

LABEL

USER

WORKDIR

EXPOSE

VOLUME

**1. FROM**: It indicates base image to run our application. On top of base image we will create our own image

**Syntax: FROM <IMAGE-NAME>**

Example:

FROM java:jdk-1.8.0

FROM tomcat:9.2

FROM mysql

**2. MAINTAINER:** It represents who is author of Dockerfile

Ex: MAINTAINER Ashok <ashokitschool@gmail.com>

**3. COPY:** It is used to copy files / folders to image while creating an image

**Syntax : COPY <source> <destination>**

Example :

copying war file from target directory to tomcat/webapps directory

COPY target/maven-web-app.war /usr/local/tomcat/webapp/maven-web-app.war

**4. ADD**

ADD is also used to copy files to image while creating an image

ADD keyword can **download files** from remote location (http)

ADD keyword will extract tar file while copying to imgae

Note: zip files we have to extract manually

Syntax:

**ADD <source> <destination>**

**ADD <url-to-download> <destination>**

Q) What is the difference between COPY and ADD?

**5. RUN**

It is used to execute commands on top of base image

**Run command instructions will execute while creating an image**

We can write multiple RUN instructions, they will execute in the order (from top to bottom)

Example:

RUN mkdir workspace

RUN yum install git

**6. CMD**

CMD is also used to execute commands

**CMD instructions will execute while creating container**

Example:

CMD sudo start tomcat

We can write multiple CMD instructions in Dockerfile but Docker will process **only last CMD instruction.**

Note: There is no use of writing multiple CMD instructions in Dockerfile

**Sample Dockerfile**

FROM ubuntu

MAINTAINER Ashok IT

RUN echo "Run One"

RUN echo "Run Two"

CMD echo "CMD One"

CMD echo "CMD Two"

RUN echo "Run Three"

Build image using docker file

* **$ docker build -t imageone .**

Run image

* **$ docker run imageone**

**Note: CMD instruction we can override using runtime CMD**

It will print only date (CMD will not execute)

* **$ docker run imageone date**

We can change docker file name

* **$ mv Dockerfile Dockerfile\_One**

Creating Docker image using Dockerfile\_One

* **$ docker build -f Dockerfile\_One -t imagetwo** .

**7. ENTRYPOINT:** ENTRYPOINT instructions will execute **while creating container**

**Note: CMD instructions we can override where as ENTRYPOINT instructions we can't override**

Example:

ENTRYPOINT [ "echo", "Welcome to Ashok IT "]

**8. WORKDIR:** It is used to set Working Directory for an image / container

Ex: WORKDIR <DIR-PATH>

**Note: The Dockerfile instructions which are available after WORKDIR those those instructions will be process from given working directory**

**9. ENV**

ENV is used to set Environment Variables

Ex: ENV <key> <value>

**10. LABEL**

LABEL will represent data in key value pair

It is used to add meta data for our image

**Ex: LABEL branchName release**

**11. ARG**

It is used to avoid hard coded values in Dockerfile

Ex:

ARG branch=develop

LABEL branch $branch

**Note: We can pass argument values in RUNTIME**

**$ docker build -t imageone --build-arg branch=feature**

**12. USER:** We can set user for an image / container

Note: After USER instruction, remaining instructions will be processed with given USER

**13. EXPOSE:** It represents on which **port number** our container is running

It is just like a documentation to understand container running port number

**14. VOLUME:** It is used for **data storage**

**Dockerizing Spring Boot Application**

FROM java:8-jdk-alpine

COPY ./target/spring-boot-docker-app.jar /usr/app/

WORKDIR /usr/app

ENTRYPOINT ["java","-jar","spring-boot-docker-app.jar"]

**Dockerizing Java Web App**

FROM tomcat:8.0.20-jre8

COPY target/java-web-app\*.war /usr/local/tomcat/webapps/java-web-app.war

Note: After running the container access application using below URL

URL : http://ec2-vm-public-ip:8080/java-web-app/

**Dockerizing Python Flask Application**

FROM python:3.7

WORKDIR /opt/app

COPY . .

RUN pip install --no-cache-dir -r requirements-prod.txt

EXPOSE 5000

CMD ["python3", "-m", "flask", "run", "--host=0.0.0.0"]

To display all running containers

* **$ docker ps**

To display all containers

* **$ docker ps -a**

Stop the container

* **$ docker stop <container-id>**

Remove the container

* **$ docker rm <container-id>**

Remove stopped container & unused images

* **$ docker system prune -a**

**Docker Volumes**

* Docker volumes are used to **persist the data generated by Docker containers**
* Using Docker volume we can de-couple data storage from Docker container
* Using Docker volume we can **share the data among multiple Docker containers**
* **On deletion of Container, Docker Volume will not be deleted.**

**Docker Volume Commands**

To display docker volume commands

* **$ docker volume --help**

Create Docker Volume

* **$ docker volume create <volume-name>**

Display Docker volumes

* **$ docker volume ls**

Inspect the volume

* **$ docker volume inspect <volume-name>**

Remove docker volume

* **$ docker volume rm <volume-name>**

Rmove unused volumes

* **$ docker volume prune**

**Pulling nginx image from dockerhub**

* **$ docker pull nginx**

Running nginx image using docker container with container name as "webapp1"

* **$ docker run --name=webapp1 -d -p 80:80 nginx**

Note: Access static website using EC2 VM public ip which is hosted by nginx

Modify static website content in index.html file (file is available in container)

* **$ docker exec -it webapp1 bash**
* **$ cd /usr/share/nginx/html**
* **$ echo "Welcome to Ashok IT" > index.html**

Note: Access static website using EC2 VM public ip which is hosted by nginx (Modified content shud display here)

Stop docker container which is "webapp1"

* **$ docker stop <container-id>**

Start another docker container with name as "webapp2"

* **$ docker run --name=webapp2 -d -p 80:80 nginx**

Note: Access static website using EC2 VM public ip which is hosted by nginx

(Changes we made in first container will not reflect here)

Stop docker container which is "webapp2"

* **$ docker stop <container-id>**

Create docker volume

* **$ docker volume create new\_vol**

Start docker container by mouting docker volume

* **$ docker run -d --name=webapp20 --mount source=new\_vol,destination=/usr/share/nginx/html -p 80:80 nginx**

Note: Access static website using EC2 VM public ip which is hosted by nginx

* **$ docker exec -it webapp20 bash**
* **$ cd /usr/share/nginx/html**
* **$ echo "Welcome to Ashok IT" > index.html**

Note: Access static website using EC2 VM public ip which is hosted by nginx

Stop the docker container

* **$ docker stop <container-id>**

Run another docker container (modified data should reflect here also)

* **$ docker run -d --name=webapp21 --mount source=new\_vol,destination=/usr/share/nginx/html -p 80:80 nginx**

**Host Directory Mounting to Docker Container**

Create Directory in host machine

* **$ mkdir /tmp/nginx/html**

Run Docker container for nginx image with directory mouting (container name is C1)

* **$ docker run -d -p 80:80 -v /tmp/nginx/html:/usr/share/nginx/html --name c1 nginx:latest**

See the containers which are running (c1 should be running)

* **$ docker container ls**

Note: Acess Nginx webserver using Host Machine Public IP

(It will not displayed bcz index.html file not available in container)

Goto Mounted directory in host machine

* **$ cd /tmp/nginx/html**

Create index.html file

* **$ vi index.html**

Note: write the content in index.html file then save it close it

Access Nginx webserver in browser (Changes will be reflected)

Note: We have created index.html file in host machine mounted directory it is reflecting in container

**Docker Compose**

* In Realworld applications are getting developed using Microservices Architecture
* In Microservices architecture several APIs (**Application program interface**) will be available
* Every API is a project and it should run in a container
* Running multiple containers manually for all APIs is difficult job

Note: To solve this problem Docker Compose came into picture

* Docker compose is a tool which is used to manage multi container based applications
* Using Docker compose we can define & deploy multi container based applications easily
* We will give input to docker compose tool **using YAML file** to run multiple containers
* Docker Compose YML file should have information related to all our service

**Sample Docker Compose YML (docker-compose.yml)**

version:

services:

network:

volumes:

Docker Compose Default File Name: docker-compose.yml

Create and start the containers

* **$ docker-compose up**

List Docker containers started by docker compose

* **$ docker-compose ps**

Stop and remove containers

* **$ docker-compose down**

List down running container images

* **$ docker-compose images**

Using different file

* **$ docker-compose -f <filename> up**

**Docker Compose Setup**

Download docker compose

**$ sudo curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose**

Give permission

**$ sudo chmod +x /usr/local/bin/docker-compose**

How to check docker compose is installed or not

**$ docker-compose --version**

Create a docker compose file for setting up dev environment. Mysql container is linked with wordpress container.

**$ vim docker-compose.yml**

---

services:

mydb:

environment:

MYSQL\_ROOT\_PASSWORD: ashokit

image: "mysql:5"

mysite:

image: wordpress

links:

- "mydb:mysql"

ports:

- "9090:80"

version: "3"

Lets remove all the running container

**$ docker rm -f $(docker ps -aq)**

How to start the above services from yml file

**$ docker-compose up**

We got lot of logs coming on the screen. to avoid it we use -d option

**$ docker-compose stop**

Remove the container

**$ docker rm -f $(docker ps -aq)**

Run containers in detached mode

**$ docker-compose up -d**

To check wordpress site

**URL: http://public\_ip:9090**

To stop both the containers

**$ docker-compose stop**

**Spring Boot App with MSQL DB using Docker Compose**

To run spring boot application we will use below Dockerfile

Dockerfile

----------------

FROM openjdk:jdk1.8

COPY target/sb-app.jar /usr/local/sb-app.jar

WORKDIR /usr/local/

EXPOSE 8080

ENTRYPOINT [ "java" , "-jar", "sb-app.jar" ]

If we are using MySQL DB in spring boot application then we need to MySQL DB in a container

Note: Instead of managing two containers separately we can use Docker Compose.

**Docker Compose Configuration (docker-compose.yml)**

version: "3"

services:

boot-app:

image: sb-rest-api

ports:

- "8080" : 8080

depends\_on:

- mysql-db

mysql-db:

image: mysql:8

environment:

- MYSQL\_ROOT\_PASSWORD = ashokit

- MYSQL\_DATABASE = bootdb

**Docker Network**

* Networking is all about communication among processes
* Docker networking enables a user to link a docker container to as many networks as they require
* Docker network is used to provide complete isolation for Docker containers

**Advantages of Docker Networking**

1) They share single operating system and maintain containers in isolated manner

2) It requires fewer OS instance to run the workload

3) It helps in fast delivery of the software

4) It helps in application portability

When Docker s/w is installed in a machine by default 3 docker networks will be configured

**1) none**

**2) host**

**3) bridge**

Note: One container we can attach to multiple networks

When container is attached to multiple networks then those containers can communicate

Docker providing Networking to containers using Network Drivers

**Docker Network Drivers**

1) Bridge

2) Host

3) None

4) Overlay

5) Macvlan

**Bridge Driver**

Bridge: This the default network driver created on the Docker host machine

Note: Bridge network drivers are very useful **when application running in standalone container**

We can see more details about bridge driver using below command

**$ docker network inspect bridge**

**Host Driver**

It is useful when standalone container is available

**The container will not get any IP address when we enable Host Driver**

Note: For example a container is executed that binds to port 80 with Host Network Driver. In this case we no need to map container port to host machine port.

This is useful when we are running our containers with large no. of ports

**None Driver**

In this type of network, the containers will have no access to network

**Overlay Driver**

We will use Docker Swarm to orchestrate our Docker containers

Overlay Driver will be used when we have Docker Swarm cluster

**Macvlan Driver**

This network driver will assign a MAC address to a container

MAC address will make our device as Physical

Using this MAC address Docker engine routes the traffic to a particular route

Macvlan driver simplifies communication betweek containers

**Working with Docker Networking Commands**

To list all networks

**$ docker network ls**

Running docker container with default network

**$ docker run --name nginx -d -p 80:80 nginx**

**$ docker inspect nginx**

Creating our own bridge network

**$ docker network create --driver bridge my-network**

**Note:** If we don't specify driver then by **default it will take bridge driver**

Run the docker container using custom network which we have created

**$ docker run --name nginx20 -d --network my-network -p 7070:80 nginx**

Check the containers attached to my-network

**$ docker network inspect my-network**

Run 2 Docker containers using My-Network and check connectivity between 2 containers using ping command

**$ docker run --name nginx30 -d --network my-network -p 8080:80 nginx**

**$ docker run --name nginx31 -d --network my-network -p 9090:80 nginx**

Get IP address of Docker Container

Synax: **$ docker inspect -f '{{range.NetworkSettings.Networks}} {{.IPAddress}}{{end}}' <container-id-or-name>**

Nginx30 Container IP: 172.19.0.2

Nginx31 Container IP: 172.19.0.3

Connect to nginx30 container and ping nginx31 container IP using ping command

**$ docker exec -it nginx30 /bin/bash**

**$ apt-get update**

**$ apit-get install iputils-ping**

**$ ping 172.19.0.3**

Connect to nginx31 container and ping nginx30 container IP using ping command

**$ docker exec -it nginx31 /bin/bash**

**$ apt-get update**

**$ apit-get install iputils-ping**

**$ ping 172.19.0.2**

**Running Containers Using Host Network Driver**

When we use Host Network driver port mapping is not required

**$ docker run --name nginx35 --network host -d nginx**

Note: If you observer we are running nginx container without port mapping because we are using Host Network Driver

**Running Containers Using Macvlan Network Driver**

In Docker, a common question that usually comes up is “How do I expose my containers directly to my local physical network?” This is especially so when you are running monitoring applications that are collecting network statistics and want to connect container to legacy applications. A possible solution to this question is to create and implement the macvlan network type.

Macvlan networks are special virtual networks that allow you to create “clones” of the physical network interface attached to your Linux servers and attach containers directly your LAN. To ensure this happens, simple designate a physical network interface on your server to a macvlan network which has its own subnet and gateway.

# Get IP address of Docker Host Machine

$ ifconfig

Note : Take ipaddress from eth0

inet : 172.31.13.126

Subnet : 172.31.13.0/24

Gateway : 172.31.13.1

# Create macvlan network using subnet and gateway

$ docker network create -d macvlan \

--subnet=172.31.13.0/24 \

--gateway=172.31.13.1 \

-o parent=eth0 \

my-macvlan-network

# Check the network created

$ docker network ls

# inspect network created

$ docker network inspect my-macvlan-network

# Run docker container using macvlan network we have created

$ docker run --rm -itd \

--network=my-macvlan-network \

--ip=172.31.13.110 \

alpine:latest \

/bin/sh

# inspect network created

$ docker network inspect my-macvlan-network

What is Virtualization

What is Containeraization

What is Docker

Advantages of Docker

Docker Architecture

Docker Setup

Docker Terminology

Dockerfile

Docker image

Docker hub

Docker Container

Docker image commands

Docker container commands

Writing Dockerfile

SpringBoot App Dockerization

Java Web App Dockerization

Python Flask App Dockerization

Stateless Container Vs Statesfull Container

Docker Volumes

Named Volumes

Binded Mount Volumes

Docker Networking

Docker Network Driver

Creating Custom Networks

Running containers with Custom Network

**Docker Swarm**

* It is a container orchestration sofware
* Orchestration means managing processes/containers
* Docker Swarm is used to setup Docker Cluster (Load Balancing)
* Cluster means group of servers
* Docker swarm is embedded in Docker engine
* We will setup Master and Worker nodes using Docker Swarm cluster
* Master will schedule the tasks (containers) and manage the nodes and node failures
* Worker nodes will perform the action (containers will run here)

**Swarm Features**

1) Cluster Management

2) Decentralize design

3) Declarative service model

4) Scaling

5) Multi Host Network

6) Service Discovery

7) Load Balancing

8) Secure by default

9) Rolling Updates

**Setup**

Create 3 EC2 instances (ubuntu)

Note: Enable 2377 port for Swarm Cluster Communications

**$ curl -fsSL https://get.docker.com -o get-docker.sh**

**$ sudo sh get-docker.sh**

1 - Master

2 - Nodes

Connect to Master Machine and execute below command

**$ sudo docker swarm init --advertise-addr 172.31.42.162**

**$ sudo docker swarm join-token worker**

**$ sudo docker swarm join --token SWMTKN-1-21l3z1izmf6plkgprlzfcr87fcxcsl3k2k79iax7yfyh3k4a00-cfgwhikmbcgsklhsxkroj8vad 172.31.38.222:2377**

Q) what is docker swarm manager quarm?

Ans) If we run only 1 master node then we can't get gaurantee High Availability for the application

Formula : (n-1)/2

If we take 2 master nodes

2-1/2 => 0.5 ( It can't become master )

3-1/2 => 1 (it can be leader when the main leader is down)

Note: Always use odd number for Master machines when we are using Docker Swarm Cluster

If we use "docker run" command then our application will executed in one container

In Docker swarm we need to deploy our application as a service.

**Docker Swarm Service**

Service is collection of one or more containers of same image

There are 2 types of services in docker swarm

1) Replica (default mode)

2) global

**$ sudo docker service create --name <serviceName> -p <hostPort>:<containerPort> <imageName>**

**$ sudo docker service create --name java-web-app -p 8080:8080 ashokit/javawebapp**

Note: By default 1 replica will be created

Check the services created

**$ docker service ls**

We can scale docker service

**$ sudo docker service scale <serviceName>=<no.of.replicas>**

Inspect docker service

**$ sudo docker service inspect --pretty java-web-app**

See service details

**$ sudo docker service ps java-web-app**

Remove one node from swarm cluster

**$ sudo docker swarm leave**

Remove docker service

**$ docker service rm java-web-app**

**KUBERNETES**

* K8S is a Production-Grade **Container Orchestration Platform**
* K8S is open source software (OSS)
* K8S is used to manage containers of our application
* K8S will take care of **container deployment, scaling, de-scaling and containers load balancing**
* K8S is **not replacement for Docker**
* K8S is **replacement for "Docker Swarm"**
* K8S developed by **Google and donated to CNCF in 2014**
* CNCF means Cloud Native Computing Foundation
* K8S s/w **developed by using GO Lang**
* K8S **v1.0 released** to market in the year of **2015**

**K8S Official Website: https://kubernetes.io/**

**K8S Features**

1) Automated Scheduling

2) Self Healing Capabilities

3) Automated Rollouts and Rollbacks

4) Load Balancing

5) Service Discovery

6) Storage Orchestration

7) Secret and configuration management

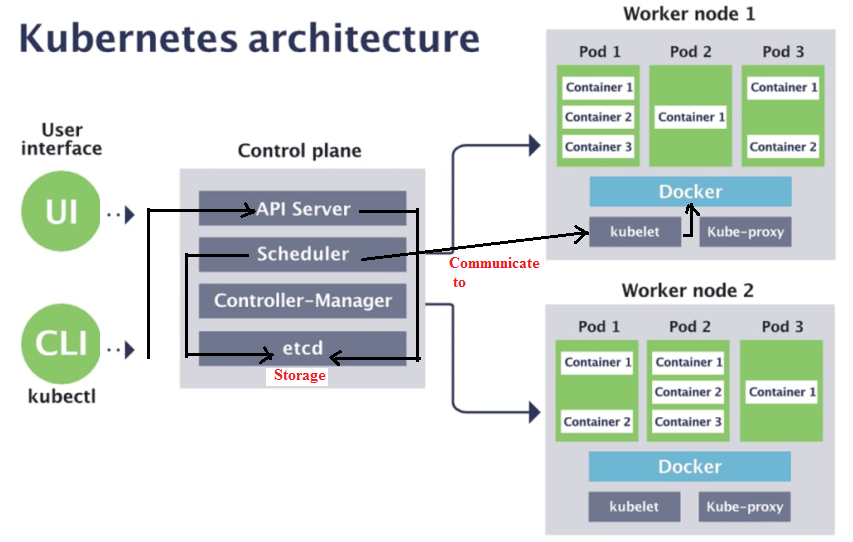
* K8S providing advanced Schedular concept to launch containers depends on our requirement
* Restarts containers that fail, replaces and reschedules containers when nodes die, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.
* If something goes wrong, Kubernetes will rollback the change for you. Take advantage of a growing ecosystem of deployment solutions.
* Scale your application up and down with a simple command, with a UI, or automatically based on CPU usage

**Note: In Docker Swarm Load Balancing is manual process where as K8S supports Auto Scaling**

* No need to modify your application to use an unfamiliar service discovery mechanism. Kubernetes gives Pods their own IP addresses and a single DNS name for a set of Pods, and can load-balance across them
* Automatically mount the storage system of your choice, whether from local storage, a public cloud provider such as GCP or AWS, or a network storage system such as NFS, iSCSI, Gluster, Ceph, Cinder, or Flocker.
* Deploy and update secrets and application configuration without rebuilding your image and without exposing secrets in your stack configuration.

**Kubernetes Architecture**

* K8S works on cluster model
* In K8S cluster we will have master node(s) and worker nodes



To communicate with Kubernetes Cluster we have 2 options

**1) UI (User Interface)**

**2) Kubectl (CLI s/w)**

**Master Node** manages worker nodes in the Cluster. It will **assign tasks to worker nodes** for execution.

Worker Nodes will run the tasks which are assigned by Master Node.

**What is API Server?**

* In K8S cluster we have several services/objects
* PODS
* ReplicationController
* ReplicationSet
* DeamonSet
* Deployment
* Volumes
* Services
* All the above K8S services implemented using GO lang. To use K8S servies we no need to learn GO language. **To use K8S servies K8S provided API server**.
* When we execute a command API server will interact with K8S s/w and it will perform required operation.
* API server will acts as communication channel between Developers / DevOps Engineers and K8S components

**What is ETCD?**

* It is a key-value pair Data Store in K8S
* It acts as database for kubernetes (How many pods, how many nodes, how many containers etc....)
* **When we ask K8S to run our application then API server will recieve that request and it will store into ETCD.**

**What is Schedular**

* It will **schedule PODS for executions** which are un-scheduled based on ETCD
* Schedular will schedule PODS on the nodes with the help of Kubelet
* Kubelet is a worker node component
* **Schedular will talk to kubelet** to to check the resources to our own application

**What is Kubelet?**

* Kubelet will **act as Node Agent**
* Kubelet will ensure that Containers are running healthy in the POD
* Kubelet will **interact with Docker Runtime** to create a container in the POD

**Note: Here we will use Docker Runtime to create our containers**

**What is POD?**

* A POD is the **smallest execution unit** in Kubernetes
* A POD encapsulates one or more applications
* Containers will be grouped as one POD in order to increase the intelligence of resources sharing
* **POD can run single container as well as can multiple container**

**What is Kube-Proxy?**

* Kube-Proxy acts as network proxy
* Kube-Proxy will **maintain network rules** on PODS
* **The network rules allow network communication to your PODS from inside or outside of your cluster**

**What is Controller Manager?**

* Controller Manager **runs controllers** in the background
* It is **responsible to run tasks in K8S cluster**
* **It performs cluster level operations**
* We have several Controllers in K8S
* NodeController
* ReplicationController
* EndpointController
* DeploymentController

**Kubernetes Cluster Setup**

* There are multiple ways to setup kubernetes cluster

**a) Self Managed K8S cluster**

**b) Provider Managed K8S cluster**

* Self Managed Cluster means we have to setup the K8S cluster on our own (Lot of commands to install)
* To create Self Managed Cluster we have 2 options

**1) Mini Kube (Single Node K8S Cluster )**

**2) Kubeadm (Multi Node K8S Cluster )**

* Provider Managed Cluster means we will use K8S cluster which is configured by someone
* EKS : Elastic Kubernetes Service ( AWS )
* AKS : Azure Kubernetes Service ( Microsoft Azure )
* GKE : Google Kubernetes Engine ( Google Cloud Platform )
* IKE : IBM Kubernetes Engine ( IBM Cloud )

**Kubernetes Core Components**

Kubernets Resources / Objects / Workloads

* Container
* POD
* Namespaces
* Service
* Deployment
* ReplicationController
* ReplicationSet
* DaemonSets
* PersistentVolumes
* StatefulSets
* Role
* Secret Config Maps
* We are using Docker to create Containers for our application
* Docker will be used as runtime engine in kubernetes cluster
* **Kubernetes is used to manage our Docker Containers**
* K8S will manage our containers but no directley (**It will use POD to manage containers**)
* **POD is a smallest building block which we can deploy in K8S cluster**
* **Containers will be wrapped under one unit called POD (Logical Grouping)**

**Note: In Docker, container is a smalletst part that we can deploy where as in K8S POD is smallet part we can deploy**

**Note:** To get clarify on PODS, we need to understand Namespaces first in K8S

**What is Namespace?**

Namespace represents a **cluster inside another cluster**

**Kubernetes components will be grouped logically using namespace**

Note: We can consider namespace as a package in java (dao pkg, service pkg, util pkg, controller pkg)

We can have multiple namespaces in k8s cluster

We can get all namespaces using below command

* **$ kubectl get namespaces**

or

* **$ kubectl get ns**

**Note:** When we setup our k8s cluster we will get below 3 namespaces

1) **default:** It will be used by default when we don't specify our namespace

2) **kube-system:** It contains k8s control plan pods

3) **kube-public:** It is reserved for kubernetes system usage

**Note:** It is not recommended to run our pods using default namespaces. We have to **create our own namespace to run our PODS**

Create our own namespace

* **$ kubectl create namespace <namespace-name>**

Ex:

* **$ kubectl create namespace sbi-customer-app**
* **$ kubectl create namespace sbi-agent-app**
* **$ kubectl create namespace sbi-report-app**

We will run our POD using custom namespace

How to get pods belongs to a namespace

* **$ kubectl get pods -n <name-space>**

Get the pods of all namespaces

* **$ kubectl get pods --all-namespaces**

Getting all pods of default namespace

* **$ kubectl get pods**

**Note:** If we delete a namespace, all the objects / resources / components also gets deleted

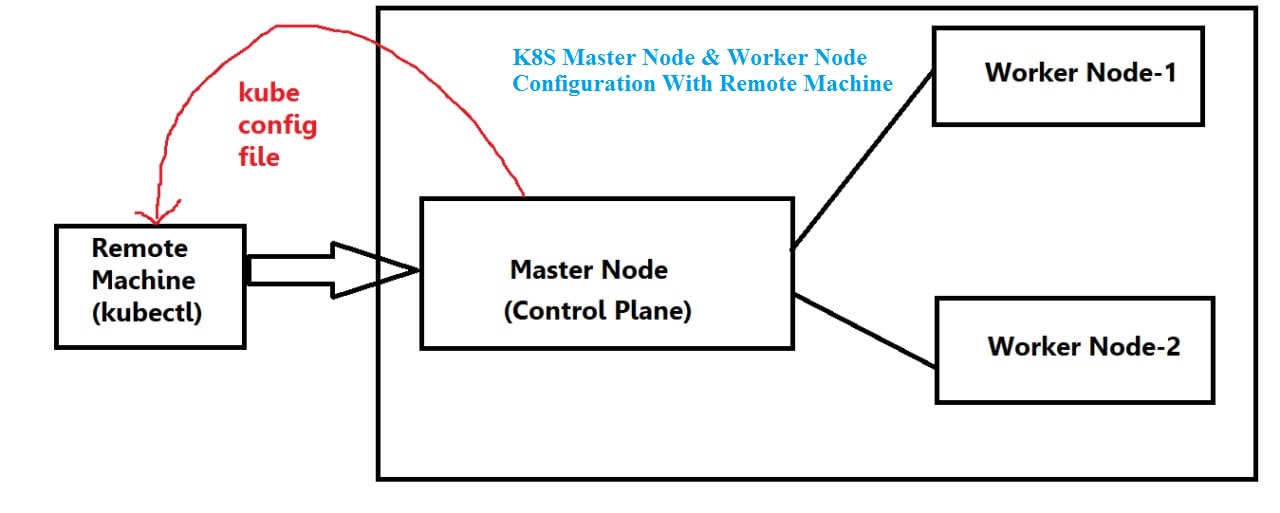
**Kubernetes Cluster Setup**

Create One Security group with Protocol as "All Traffic" Port Range as 0-65535

Create 3 Ubuntu Servers using above created security group

**1 - Master Node (t2.medium instance)**

**2 - Worker Nodes (t2.micro instances )**



**Master & Worker Node Common Commands Execution Start**

Upgrade apt packages

* **$ sudo apt-get update**

Create configuration file for containerd:

* **$ cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf overlay br\_netfilter**

**EOF**

Load modules:

* **$ sudo modprobe overlay**
* **$ sudo modprobe br\_netfilter**

Set system configurations for Kubernetes networking:

* **$ cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf**

**net.bridge.bridge-nf-call-iptables = 1**

**net.ipv4.ip\_forward = 1**

**net.bridge.bridge-nf-call-ip6tables = 1**

**EOF**

Apply new settings:

* **$ sudo sysctl --system**

Install containerd:

* **$ sudo apt-get update && sudo apt-get install -y containerd**

Create default configuration file for containerd:

* **$ sudo mkdir -p /etc/containerd**

Generate default containerd configuration and save to the newly created default file:

* **$ sudo containerd config default | sudo tee /etc/containerd/config.toml**

Restart containerd to ensure new configuration file usage:

* **$ sudo systemctl restart containerd**

Verify that containerd is running.

* **$ sudo systemctl status containerd**

Disable swap:

* **$ sudo swapoff -a**

Disable swap on startup in /etc/fstab:

* **$ sudo sed -i '/ swap / s/^\(.\*\)$/#\1/g' /etc/fstab**

Install dependency packages:

* **$ sudo apt-get update && sudo apt-get install -y apt-transport-https curl**

Download and add GPG key:

* **$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -**

Add Kubernetes to repository list:

* **$ cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list**

**deb https://apt.kubernetes.io/ kubernetes-xenial main**

**EOF**

Update package listings:

* **$ sudo apt-get update**

Install Kubernetes packages (Note: If you get a dpkg lock message, just wait a minute or two before trying the command again):

* **$ sudo apt-get install -y kubelet kubeadm kubectl kubernetes-cni nfs-common**

Turn off automatic updates:

* **$ sudo apt-mark hold kubelet kubeadm kubectl kubernetes-cni nfs-common**

**+++++++++++ Master & Worker Common Commands Execution End ++++++++++++**

**Only Master Node Commands Execution Start**

Initialize the Cluster-

Initialize the Kubernetes cluster on the control plane node using kubeadm

(Note: This is only performed on the Control Plane Node):

* **$ sudo kubeadm init**

Note: if we will get an error as "[ERROR NumCPU]: the number of available CPUs 1 is less than the required 2"

Kubeadm runs a series of pre-flight checks to validate the system state before making changes.

This error means the host don't have minimum requirement of 2 CPU.

You can ignore the error if you still want to go ahead and install kubernetes on this host.

sudo kubeadm init --ignore-preflight-errors=NumCPU

Set kubectl access:

* **mkdir -p $HOME/.kube**
* **sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config**
* **sudo chown $(id -u):$(id -g) $HOME/.kube/config**

Test access to cluster:

* **$ kubectl get nodes**

Install the Calico Network Add-On -

On the Control Plane Node, install Calico Networking:

* **$ kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml**
* **$ kubectl get nodes**

Join the Worker Nodes to the Cluster

In the Control Plane Node, create the token and copy the kubeadm join command (NOTE:The join command can also be found in the output from kubeadm init command):

* **$ kubeadm token create --print-join-command**

Note : In both Worker Nodes, paste the kubeadm join command to join the cluster. Use sudo to run it as root:

**sudo** kubeadm join ...

In the Control Plane Node, view cluster status (Note: You may have to wait a few moments to allow all nodes to become ready):

**What is POD?**

* POD is a smallest build block what we can execute inside K8S cluster
* POD will execute in a node
* One Node can execute multiple PODS
* POD can have one container & more than one container
* POD represents running process
* Containers inside the POD will share a unique network ip, storage and other specifications

**How to run our application in K8S?**

To run our docker image we need to create a pod then k8s will execute that pod in a node

**Note:** If we have pod then we can send request to K8S to schedule that POD execution.

**We can create POD in 2 ways**

1) Interactive

Interactive approach means using commands we can create a pod

Ex: kubectl run --name javawebapppod --image=ashokit/javawebapp

2) Declarative

Declarative approach means using manifest file (YML) we can create a pod

|  |  |
| --- | --- |
| **apiVersion:** | apiVersion represents version of our api like v1, v2, v3.... |
| **kind:** | kind represents what is the purpose of this manifest file |
| **metadata:** | metadata represents data about the (labels) |
| **spec:** | spec represents specification (what you want to use for this manifest) |

* **$ vi javawebapppod.yml**

---

apiVersion: v1

kind: Pod

metadata:

name: javawebapppod

labels:

app: javawebapp

spec:

containers:

- name: javawebappcontainer

image: ashokit/javawebapp

ports:

- containerPort: 8080

...

Get all pods

* **$ kubectl get pods**

Create POD using manifest file

* **$ kubectl apply -f javawebapppod.yml**

Describe the pod using below command

* **$ kubectl describe pod javawebapppod**

Check where the pod is running

* **$ kubectl get pods -o wide**

Note: we can access the POD across the cluster using POD IP.

* **$ curl pod-ip:8080**

**Note: We can't access POD using POD IP outside of the cluster** (this is default behaviour)

**POD Lifecycle**

* Make a request to API server using manifest file (YML) to create a POD
* API server will save the POD info in ETCD
* Schedular find un-scheduled POD info and schedule that POD for execution in NODE
* Kubelet will see that POD Execution schedule and it will trigger DOCKER Runtime
* Docker Runtime will run that container inside the POD.

**Note:** POD is ephemeral (**lives for short period of time**)

* When POD is **re-created then POD IP will change**
* **It is not recommended to access the POD using POD ID**
* We will use **"Kubernetes Service"** component to execute the PODs
* K8S service will make **POD accessible** / discoverable **inside the cluster and outside the cluster also**
* When we create a service we will get one Virtual IP (cluster IP).
* Cluster IP will be registered in K8S DNS with its name.

**What is K8S Service?**

Service is responsible to make our PODS discoverable / accessible inside and outside of the cluster

**Service will identify the POD using** **POD label / selector**

**We have 3 types of services**

1) ClusterIP

2) NodePort

3) Load Balancer

---

apiVersion: v1

kind: Service

metadata:

name: javawebappsvc

spec:

type: ClusterIP

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

...

To get all services

* **$ kubectl get svc**

Schedule a service using manifest

* **$ kubectl apply -f javawebappsvc.yml**
* **$ kubectl get svc**

Note: In CluterIP one VIRTUAL IP will be assigned for our service. **Using that ClusterIP** **we can access service with in the cluster.**

**If we want to expose our service outside cluster we need to use NodePort Service**

---

apiVersion: v1

kind: Service

metadata:

name: javawebappsvc

spec:

type: NodePort

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

#nodePort: 32611

...

For NodePort service kubernetes will assign random port number if we don't specify nodePort in manifest file

We can access our service outside cluster using **any cluster machine public IP** with node port

**Note:** Enable node port in security group.

URL access to app : http://ec2-vm-ip:nodeport/context-path

(http://13.233.63.130:32645/java-web-app/)

Q) What is the range of Node PORT in k8s cluster?

**Ans) 30000 – 32767**

**In the above scenario we have created the POD manually (it is not recommended)**

**If we create the POD then K8S will not provide high availability**

lets test it by deleting our pod

* **$ kubectl delete pod <pod-name>**

**Note: once pod got delete, k8s not creating another pod and application went down (not accessible)**

If we want to achieve high availability then we should not create pods manually

We need to use K8S components to create PODS then k8s will provide high availability for our application

Note: High Availability means always our application should be accessible

* **ReplicationController**
* **ReplicationSet**
* **DaemonSet**
* **Deployment**
* **StatefulSets**

**What is Replication Controller?**

* It is one of the key features in k8s
* It is responsible to manage POD lifecycle
* It will make sure given no. of POD replicas are running at any point of time.

**Note: if any POD got crashed/deleted/dead then Replication Controller will replace it.**

* Replication Controller is providing facility to create multiple PODS and it will make sure PODS always exists to run our application.
* **Using Replication controller we can achieve High Availability**
* Replication Controller and PODS are associated with **Labels and Selectors.**

---

# pod manifest configuration

apiVersion: v1

kind: ReplicationController

metadata:

name: javawebapprc

spec:

replicas: 1

selector:

app: javawebapp

template:

metadata:

name: javawebapppod

labels:

app: javawebapp

spec:

containers:

- name: javawebappcontainer

image: ashokit/javawebapp

ports:

- containerPort: 8080

---

# node-port service manifest

apiVersion: v1

kind: Service

metadata:

name: javawebappsvc

spec:

type: NodePort

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

...

**What is Replica Set?**

* **It is next generation of Replication Controller**
* It is also used to manage POD life cycle
* We can scale up and scale down PODS using Replica Set also
* **The only** **difference** between Replication Controller and Replication Set is **'Selector support'**

We have 2 types of Selectors

**1) Equality Selector**

Ex: selector:

app: javawebappp

**2) Set based Selector**

selector:

matchExpressions:

- key : app

operator : in

values:

- javapp

- javaweb

- javawebapp

---

# pod manifest configuration

apiVersion: apps/v1

kind: ReplicaSet

metadata:

name: javawebapprc

spec:

replicas: 1

selector:

matchLabels:

app: javawebapp

template:

metadata:

name: javawebapppod

labels:

app: javawebapp

spec:

containers:

- name: javawebappcontainer

image: ashokit/javawebapp

ports:

- containerPort: 8080

---

# node-port service manifest

apiVersion: v1

kind: Service

metadata:

name: javawebappsvc

spec:

type: NodePort

selector:

app: javawebapp

ports:

- port: 80

targetPort: 8080

...

**What is DaemonSet? (For every worker it create PODS)**

* A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected.
* Deleting a DaemonSet will clean up the Pods it created.
* Some typical uses of a DaemonSet are:

1) Running a **cluster storage** daemon on every node

2) Running a **logs collection** daemon on every node

3) Running a **node monitoring** daemon on every node

**Note: Replicas not applicable for DaemonSet**

---

apiVersion: apps/v1

kind: DaemonSet

metadata:

name: logging

spec:

selector:

matchLabels:

app: httpd-logging

template:

metadata:

labels:

app: httpd-logging

spec:

containers:

- name: webserver

image: httpd

ports:

- containerPort: 80

...

* **Manually POD Created ( Not recommended )**
* **POD creation using ReplicationController**
* **POD creation using ReplicaSet**
* **POD creation using DaemonSet**
* In above concepts **Auto-Scaling feature not available** (Manually we need to scale our pods)
* There is **no option to rollback** our pods creation.
* **To overcome these problems We have "Deployment" concept**

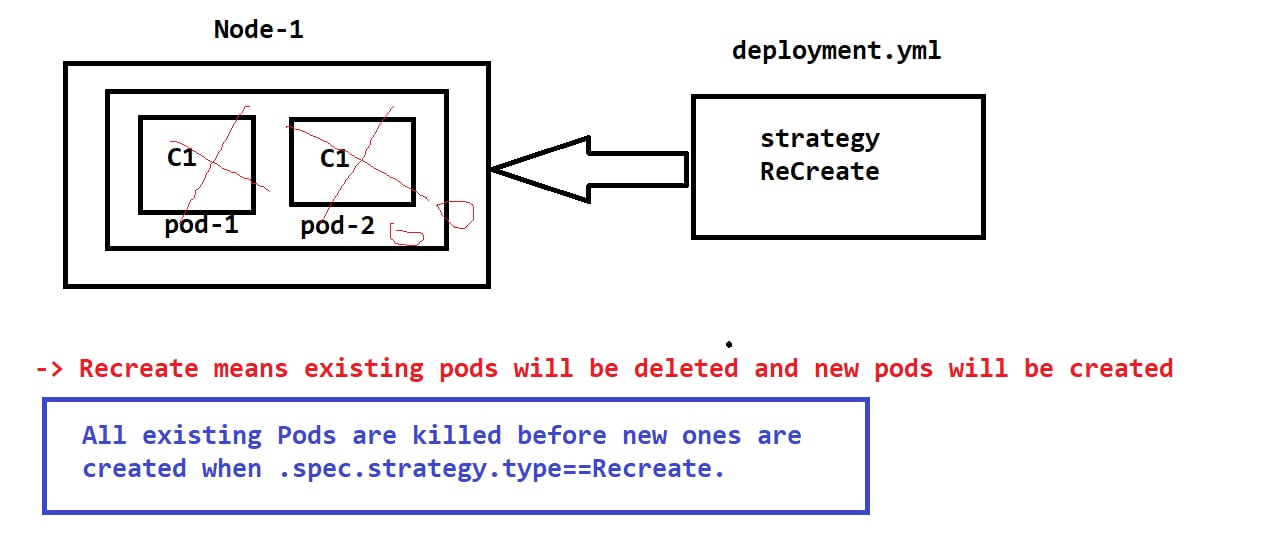
**What is Deployment?**

* Deploymet is used to tell Kubernetes how to create or modify instances of the pods
* **By using Deployment we can rollout and rollback our application deployment (if required)**
* **We can achieve Auto-Scaling by Deployment**
* **Deployment Strategy**

1) ReCreate

2) Rolling Update

3) Blue / Green (Approach)



**K8S deployment manifest file ( POD Manifest + Service Manifest )**

---

**# POD Deployment Manifest**

apiVersion: apps/v1

kind: **Deployment**

metadata:

name: javawebappdeployment

labels:

app: javawebapp

spec:

replicas: 1

strategy:

type: **Recreate**

selector:

matchLabels:

app: javawebapp

template:

metadata:

name: javawebapppod

labels:

app: javawebapp

spec:

containers:

- name: javawebappcontainer

image: ashokit/javawebapp

ports:

- containerPort: 8080

---

---

**# Service Manifest**

apiVersion: v1

kind: Service

metadata:

name: javawebappsvc

spec:

type: NodePort

selector:

app: javawebapp

ports:

- port: 80

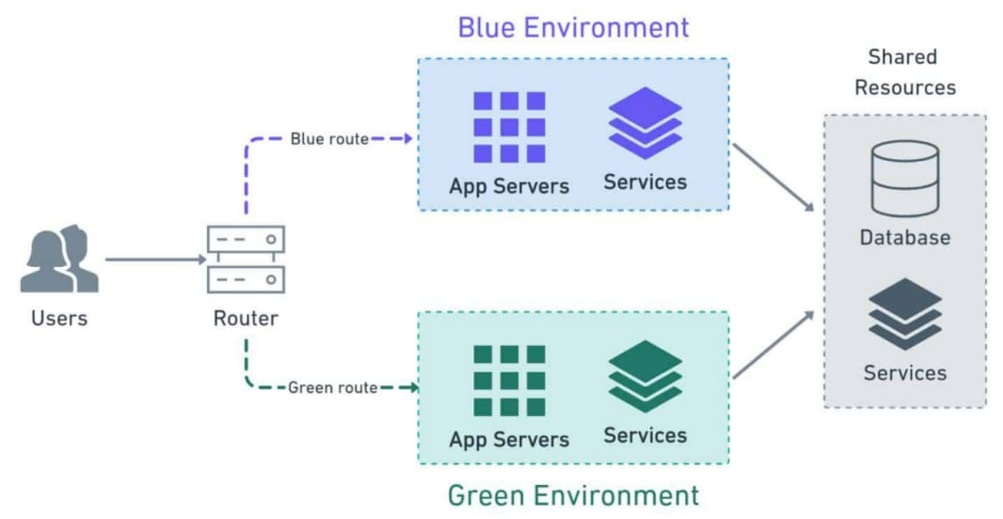
targetPort: 8080

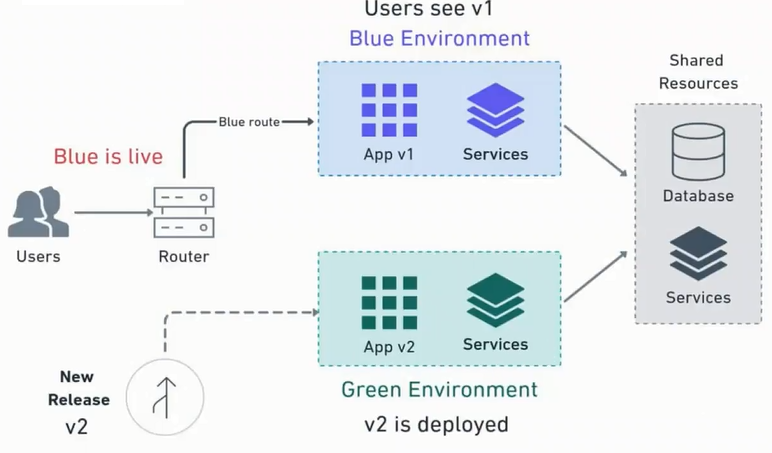
...

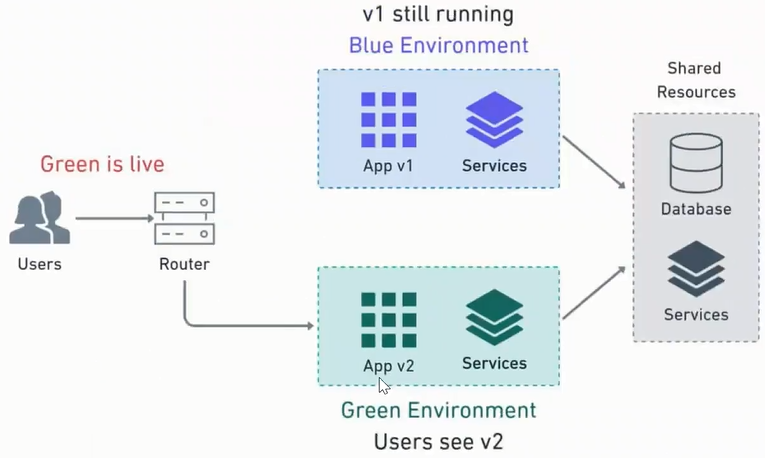
* **$ kubectl get pods**
* **$ kubectl get svc**
* **$ kubectl delete all --all**
* **$ kubectl apply -f deployment.yml**
* **$ kubectl get pods**
* **$ kubectl get svc**
* **$ kubectl get deployment**
* **$ kubectl delete deployment <deployment-name>**

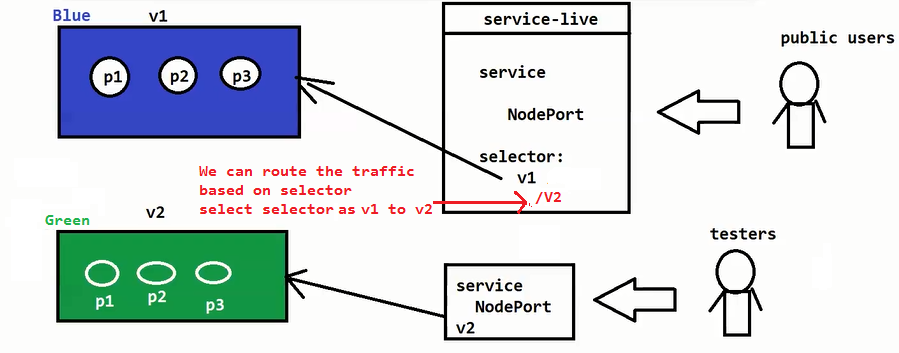
**Blue / Green Deployment (Approach)**

* It is an application release model.
* It reduces risk and minimizes downtime.
* It uses two production environment known as Blue and Green.
* Rapid releasing, Simple rollback, Zero downtime

****



****



**Autoscaling**

It is the process of increasing / decreasing infrastructure based on demand

Autoscaling can be done in 2 ways

1) Horizontal Scaling

2) Vertical Scaling

Horizontal Scaling means **increasing number of instances/systems**

Vertical Scaling means **increasing capacity** of single system

Note: For production we will use Horizontal Scaling

**HPA: Horizontal POD Autoscaling**

**VPA: Vertical POD Autoscaling (we don't use this)**

HPA: Horizontal POD Autoscaler which will **scale up/down number of pod** replicas of deployment, ReplicaSet or Replication Controller dynamically **based on the observed Metrics** (CPU or Memory Utilization).

HPA will interact with Metric Server to identify CPU/Memory utilization of POD.

To get node metrics

* **$ kubectl top nodes**

To get pod metrics

* **$ kubectl top pods**

Note: By default metrics service is not available

Metrics server is an application that collects metrics from objects such as pods, nodes according to the state of CPU, RAM and keeps them in time.

Metric-Server can be installed in the system as an addon. You can take and install it directley from the repo.

1) Clone git repo

* **$ git clone https://github.com/ashokitschool/k8s\_metrics\_server**

2) Check the cloned repo

* **$ cd k8s\_metrics\_server**
* **$ ls deploy/1.8+/**

3) Apply manifest files from manifest-server directlry

* **$ kubectl apply -f deploy/1.8+/**

Note: it will create service account, role, role binding all the stuff

We can see metric server running in kube-system ns

* **$ kubectl get all -n kube-system**

Check the top nodes using metric server

* **$ kubectl top nodes**

Check the top pods using metric server

* **$ kubectl top pods**

Note: When we install Metric Server, it is installed under the kubernetes system namespaces.

* **$ kubectl delete all -all**

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: hpadeployment

labels:

name: hpadeployment

spec:

replicas: 2

selector:

matchLabels:

name: hpapod

template:

metadata:

labels:

name: hpapod

spec:

containers:

- name: hpacontainer

image: k8s.gcr.io/hpa-example

ports:

- name: http

containerPort: 80

resources:

requests:

cpu: "100m"

memory: "64Mi"

limits:

cpu: "100m"

memory: "256Mi"

---

apiVersion: v1

kind: Service

metadata:

name: hpaclusterservice

labels:

name: hpaservice

spec:

ports:

- port: 80

targetPort: 80

selector:

name: hpapod

type: ClusterIP

---

apiVersion: autoscaling/v2beta1

kind: HorizontalPodAutoscaler

metadata:

name: hpadeploymentautoscaler

spec:

scaleTargetRef:

apiVersion: apps/v1

kind: Deployment

name: hpadeployment

minReplicas: 2

maxReplicas: 5

metrics:

- resource:

name: cpu

targetAverageUtilization: 50

type: Resource

------------------------------

Resources & requests

In cluster if none of the pods have this min resources availabile it will not schedue

Min resource and Memory we are configuring to schedule pods using HPA

Note: take hpademo.yml

* **$ kubectl get pods**
* **$ kubectl apply -f hpa.yml**

Note: as of now there is no load on application

Now we need to simulate the load

We can simulate load using busybox

* **$ kubectl run -it --rm loadgenerator --image=busybox**

Note: witht this command we are inside the pod

* **$ wget -q -O- http://hpaclusterservice**

Note: we got response

* **$ while true; do wget -q -O- http://hpaclusterservice; done**

Note: connect to control-pane and check pods

* **$ kubectl top pods**
* **$ kubectl get hpa**

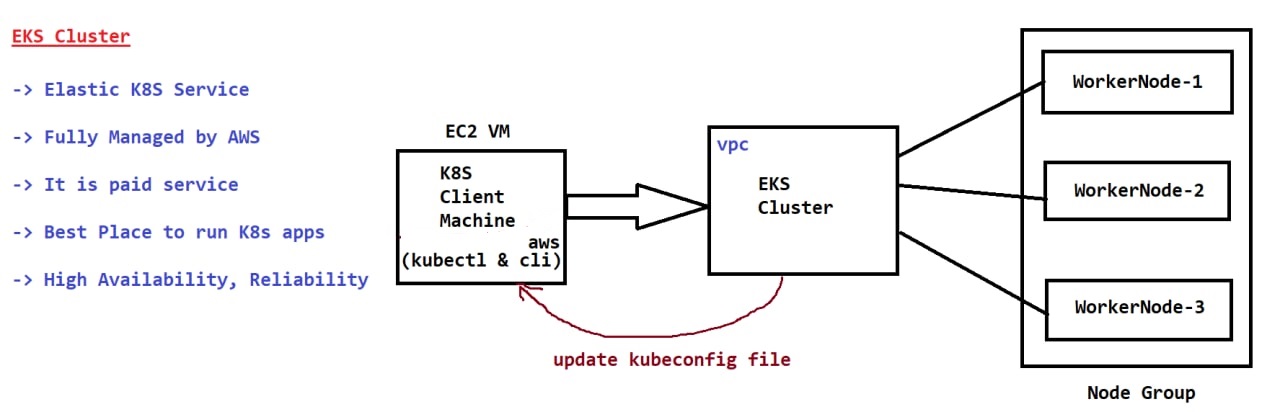
**AWS-EKS (Elastic Kubernetes Service)**

* EKS stands for “Elastic Kubernetes Service"
* EKS is a fully managed K8S service
* EKS is the best place to run K8S applications because of its security, reliability and scalability
* **EKS can be integrated with other AWS services such as ELB, CloudWatch, AutoScaling, IAM and VPC**
* EKS makes it easy to run K8S on AWS without needing to install, operate and maintain your own k8s control plane.
* Amazon EKS runs the K8S control Plane across three availability zones in order to ensure high availability and it automatically detects and replaces unhealthy masters.
* AWS will have complete control over Control Plane. We don't have control on Control Plane.
* We need to create Worker Nodes and attach to Control Plane.

Note: We will create Worker Nodes Group using ASG Group

* Control Plane Charges + Worker Node Charges (Based on Instance Type & No.of Instances)

**Note: $0.10 per hour**



**Pre-Requisites**

AWS account with admin priviliges

Instance to manage/access EKS cluster using Kubectl

AWS CLI access to use kubectl utility

**Steps to Create EKS Cluster in AWS**

**Step-1) Create VPC using Cloud Formation (with below S3 URL)**

**URL:** https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml

Stack name: EKSVPCCloudFormation

**Step-2) Create IAM role in AWS**

Entity Type: AWS Service

Select Usecase as 'EKS' ==> EKS Cluster

Role Name: EKSClusterRole (you can give any name for the role)

**Step-3) Create EKS Cluster using Created VPC and IAM Role**

Cluster endpoint access: Public & Private

**Step-4) Create RedHat ec2 Instance (K8S\_Client\_Machine)**

Connect to K8S\_Client\_Machine using Mobaxterm

**#################### Install Kubectl with below commands ######################**

* $ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
* $ sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
* $ kubectl version --client

**######### Install AWS ClI in K8S\_Client\_Machine with below commands ###########**

* $ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"
* $ sudo yum install unzip
* $ unzip awscliv2.zip
* $ sudo ./aws/install

**###################### Configure AWS Cli with Credentials ####################**

Access Key ID: AKIA4MGQ5UW7R76

Secret Access Key: ZoZZW+063Km49zi19FbPC3Ijo15auV

$ aws configure

Note: We can use root user accesskey and secret key access

**##########################################################################**

$ aws eks list-clusters

$ ls ~/.

### Update kubeconfig file in remote machine from cluster using below command ########

$ aws eks update-kubeconfig --name <cluster-name> --region ap-south-1

**###########################################################################**

**Step-5) Create IAM role for EKS worker nodes (usecase as EC2) with below policies**

a) AmazonEKSWorkerNodePolicy

b) AmazonEKS\_CNI\_Policy

c) AmazonEC2ContainerRegistryReadOnly

**Step-6) Create Worker Node Group**

* Go to cluster -> Compute -> Node Group
* Select the Role we have created for WorkerNodes
* Use t2.large
* Min 2 and Max 5

**Step-7) Once Node Group added then check nodes in K8s\_client\_machine**

$ kubectl get nodes

$ kubectl get pods --all-namespaces

**Step-8) Create POD and Expose the POD using NodePort service**

Note: Enable NODE PORT in security Group to access that in our browser

**Kubernetes Ingress**

* Deploy two application Into K8S using Service using Cluster IP
* $ kubectl apply -f javawebapp.yml
* $ kubectl apply -f mavenwebapp.yml
* Now we have 2 services running in K8S cluster with **Cluster IP service**. **We can't access them outside the cluster.**
* **We will use Ingress to provide routing for these two services from** **external traffic**
* K8S ingress is a resource to add rules for routing traffic from external sources to the services in the k8s cluster
* K8S ingress is a native k8s resource where you can have rules to route traffic from an external source to service endpoints residing inside the cluster.
* It requires an ingress controller for routing the rules specified in the ingress object
* Ingress controller is typically a proxy service deployed in the cluster. It is nothing but a Kubernetes deployment exposed to a service.

**Ingress Setup**

# git clone k8s-ingress

$ git clone https://github.com/ashokitschool/kubernetes\_ingress.git

$ cd kubernetes-ingress

# Create namespace and service-account

$ kubectl apply -f common/ns-and-sa.yaml

# create RBAC and configMap

$ kubectl apply -f common/

# Deploy Ingress controller

-> We have 2 options to deploy ingress controller

1) Deployment

2) DaemonSet

$ kubectl apply -f daemon-set/nginx-ingress.yaml

# Get ingress pods using namespace

$ kubectl get all -n nginx-ingress

# create LBR service

$ kubectl apply -f service/loadbalancer-aws-elb.yaml

Note: It will generate LBR DNS

-> Map LBR dns to route 53 domain

-> Create Ingress kind with rules

============================

Path Based Routing

$ vi ingress-rules2-routes.yml

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: ingress-resource-2

spec:

ingressClassName: nginx

rules:

- host: ashokit.org

http:

paths:

- pathType: Prefix

path: "/java-web-app"

backend:

service:

name: javawebappsvc

port:

number: 80

- pathType: Prefix

path: "/maven-web-app"

backend:

service:

name: mavenwebappsvc

port:

number: 80

...

**K8S HELM**

* We deployed our apps in Kubernetes cluster using Manifest files
* Manifest files we can write in 2 ways

1) JSON

2) YML (more demand)

* It is difficult to write manifest files for our applications
* **Helm is a package manager for k8s applications**
* **Helm allows you to install or deploy applications on kubernetes cluster in a similar manner to yum/apt for linux distributions.**
* Helm lets you fetch, deploy and manage the lifecycle of applications both 3rd party apps and your own applications

Ex: promethus, graphana, nginx-ingress are third party apps

* Helm introduces several familiar concepts such as Helm Chart (**package contains k8s manifests - templates**)
* **Helm Repositories which holds helm charts/packages**
* A CLI with install/upgrade/remove commands

**Why to use Helm?**

Deploying application on K8S cluster is little difficult

As part of app deployment we need to create below k8s objects

Deployment

Service

ConfigMaps/Secrets

Volumes

Ingress Rules

HPA

Helm greatly simplifies the process of creating, deploying and managing applications on k8s cluster

Helm also maintains a versioned history of very chart (application) installation. If something goes wrong, you can simply call **'helm rollback'.**

Setting up a single application can involve creating multiple independent k8s resources and each resource requires a manifest file.

**Purpose of Helm**

* Create new charts from scratch
* Package charts into chart archive (tgz) files
* Interact with chart repositories where charts are stored
* Install and uninstall charts into an existing Kubernetes cluster
* Manage the release cycle of charts that have been installed with Helm

**What is Helm Chart?**

HELM chart is a basically just a **collection of manifest files** organized in a specific directory structure that describe a related K8S resource.

There are two main components in HELM chart

1) Template

2) Value

Templates and values renders a manifest which can understand by k8s

**Helm uses charts to pack all the required k8s components (manifests) for an application to deploy, run and scale.**

Charts are very similar to RPM and DEB packages for Linux.

Ex: yum install git

Note: it will interact with repo and it will download git

**HELM Concepts**

Helm packages are called charts, and they consist of a few YML configuration files and some templates that are rendered into K8S manifest files. Here is the basic directory structure of a chart.

charts : dependent charts will be added here

templates: contains all template files

values : It contains values which are required for templates

**HELM Architecture**

what-the-helm

├── Chart.yaml

├── charts

├── templates

│ ├── NOTES.txt

│ ├── \_helpers.tpl

│ ├── deployment.yaml

│ ├── ingress.yaml

│ ├── service.yaml

│ └── tests

│ └── test-connection.yaml

└── values.yaml

**Helm Installation**

$ curl -fsSl -o get\_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3

$ chmod 700 get\_helm.sh

$ ./get\_helm.sh

$ helm

check do we have metrics server on the cluster

$ kubectl top pods

$ kubectl top nodes

# check helm repos

$ helm repo ls

# Before you can install the chart you will need to add the metrics-server repo to

$ helm repo add metrics-server https://kubernetes-sigs.github.io/metrics-server/

# Install the chart

$ helm upgrade --install metrics-server metrics-server/metrics-server

$ helm list

$ helm delete <release-name>

**KUBERNETES MONITORING**

* Prometheus is an open-source systems monitoring and alerting toolkit
* Prometheus **collects and stores** its metrics as time series data
* It provides out-of-the-box monitoring capabilities for the k8s container orchestration platform.
* Grafana is a database **analysis and monitoring tool**
* Grafana is a multi-platform open source analytics and interactive visualization web application.
* It provides **charts, graphs, and alerts** for the web when connected to supported data sources.
* Grafana allows you to query, visualize, alert on and understand your metrics no matter where they are stored. Create, explore and share dashboards.

**Note: Graphana will connect with Prometheus for data source.**

**How to deploy Grafana & Prometheus in K8S**

Most Efficient way is using Helm Chart to deploy Prometheus Operator

**Install Prometheus & Grafana**

Add the latest helm repository in Kubernetes

* **$ helm repo add stable https://charts.helm.sh/stable**

Add prometheus repo to helm

* **$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts**

Update Helm Repo

* **$ helm repo update**

Search Repo

* **$ helm search repo prometheus-community**

Install prometheus

* **$ helm install stable prometheus-community/kube-prometheus-stack**

Get all pods

* **$ kubectl get pods**

Node: You should see prometheus pods running

Check the services

* **$ kubectl get svc**

**By default prometheus and grafana service is available within the cluster using ClusterIP, to access them outside lets change it either NodePort or Loadbalancer.**

* **$ kubectl edit svc stable-kube-prometheus-sta-prometheus**

Now edit the grafana service

* **$ kubectl edit svc stable-grafana**

Verify the service if changed to LoadBalancer

* **$ kubectl get svc**

To access Prometheus web interface copy Loadbalancer URL and port number 9090

To access Grafana web interface copy Loadbalancer URL and port number 80

**UserName:** admin

**Password:** prom-operator

**ELK Stack**

The ELK Stack is a collection of three open-source products — Elasticsearch, Logstash, and Kibana

ELK stack provides centralized logging in order to identify problems with servers or applications

* It allows you to search all the logs in a single place
* E stands for : Elastic Search --> It is used to store logs
* L stands for : Log Stash --> It is used for processing logs
* K stands for : Kibana --> It is an visualization tool
* FileBeat : Log files
* MetricBeat : Metrics
* PacketBeat : Network data
* HeartBeat : Uptime Monitoring

Filebeat collect data from the log files and sends it to logstash

Logstash enhances the data and sends it to Elastic search

Elastic search stores and indexes the data

Kibana displays the datas stored in Elastic Search

**Installation using HELM**

Pre-requisites :

EKS Cluster

Nodes : 4 GB RAM

Client Machine with kubectl & helm configured

* **$ kubectl create ns efk**
* **$ kubectl get ns**
* **$ helm ls**
* **$ helm repo add elastic https://helm.elastic.co**
* **$ helm repo ls**
* **$ helm show values elastic/elasticsearch >> elasticsearch.values**
* **$ vi elasticsearch.values**

replicas as 1 & masternodes as 1

* **$ helm install elasticsearch elastic/elasticsearch -f elasticsearch.values -n efk**
* **$ helm ls -n efk**
* **$ kubectl get all -n efk**
* **$ helm show values elastic/kibana >> kibana.values**
* **$ vi kibana.values**

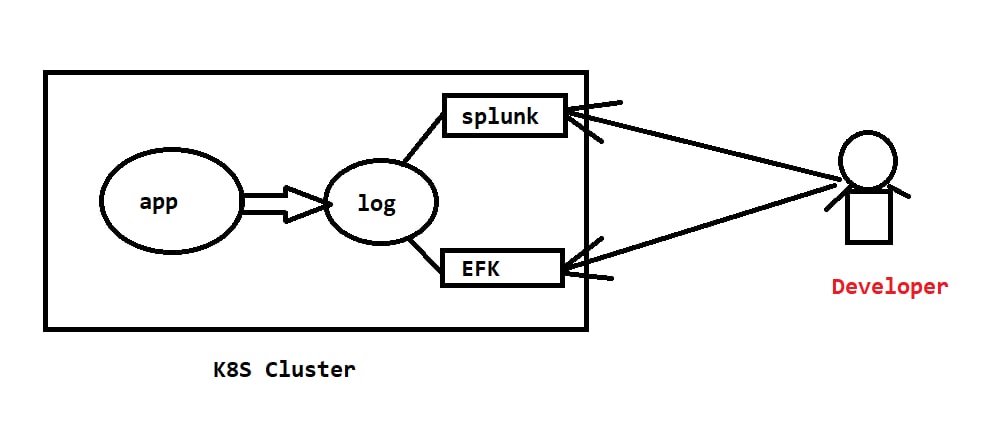
Set replicas as 1

Change Service Type from ClusterIP to LoadBalancer

Change Port to 80

* **$ helm install kibana elastic/kibana -f kibana.values -n efk**
* **$ kubectl get all -n efk**
* **$ helm install filebeat elastic/filebeat -n efk**
* **$ helm install metricbeat elastic/metricbeat -n efk**

Note: Access Kibana using Load Balancer DNS



**NEXUS**

**Sonatype Nexus**

* Nexus is an Open Source Software
* It is an **Artifact Repository Server**
* It is used to **store and retrieve build artifacts**
* We can **store shared liabraries** also (e.g pwd-utils.jar) (company specific remote)
* **Nexus** software developed using **Java**
* Note: To install Nexus s/w we need to install java first
* Currently people are using **Nexus 3.x**

Java: jar, war and ear

Docker : Docker images

Node JS: NPM package

Q) What is difference between Nexus and GitHub ?

Github is Source code management software which is used to **store source code** of the project

Nexus is **Artifact Repository** which is used to **store build artifacts**

**Nexus Setup**

Take t2.medium instance

Java s/w is required to install Nexus

Connect to t2.medium instance using mobaxterm

Nexus S/w Installation Process in Linux OS

* **$ sudo su -**
* **$ cd /opt**
* **$ yum install tar wget -y**

Note: https://help.sonatype.com/repomanager3/product-information/download

Latest version

* **$ wget https://download.sonatype.com/nexus/3/nexus-3.40.1-01-unix.tar.gz**

Old version

* **$ wget http://download.sonatype.com/nexus/3/nexus-3.15.2-01-unix.tar.gz**
* **$ tar -zxvf nexus-3.40.1-01-unix.tar.gz**
* **$ mv /opt/nexus-3.40.1-01 /opt/nexus**

As a good security practice, Nexus is not advised to run nexus service as a root user, so create a new user called nexus and grant sudo access to manage nexus services as follows.

* **$ useradd nexus**

Give the sudo access to nexus user

* **$ visudo**
* **nexus ALL=(ALL) NOPASSWD: ALL**

Change the owner and group permissions to /opt/nexus and /opt/sonatype-work directories.

* **$ chown -R nexus:nexus /opt/nexus**
* **$ chown -R nexus:nexus /opt/sonatype-work**
* **$ chmod -R 775 /opt/nexus**
* **$ chmod -R 775 /opt/sonatype-work**

Open /opt/nexus/bin/nexus.rc file and uncomment run\_as\_user parameter and set as nexus user.

* **$ vi /opt/nexus/bin/nexus.rc**
* **run\_as\_user="nexus"**

Create nexus as a service

* **$ ln -s /opt/nexus/bin/nexus /etc/init.d/nexus**

Install java 1.8v

* **$ sudo yum install java-1.8.0-openjdk**

Switch as a nexus user and start the nexus service as follows.

* **$ su - nexus**

Enable the nexus services

* **$ sudo systemctl enable nexus**

Start the nexus service

* **$ sudo systemctl start nexus**

Access the Nexus server from Laptop/Desktop browser.

URL: http://IPAddess:8081/

**Note:** Enable this 8081 port number in Security Group

Default Username

User Name: admin

We can copy nexus password using below command

* **$ sudo cat /opt/sonatype-work/nexus3/admin.password**

We can change nexus default properties

* **/opt/nexus/etc/nexus.properties**

**Integrate Maven with Nexus**

Create Repositories in Nexus to store build artifacts

We will create 2 types of repositories in Nexus

1) **Snapshot**

**2) Release**

If project is **under development** then that project build artifacts will be stored into **snapshot repository**

If project **development completed** and released to production then that project build artifacts will be stored to **release repository**

Snanpshot Repo URL: http://13.233.238.64:8081/repository/ashokit\_snapshot\_repo/

Release Repo URL: http://13.233.238.64:8081/repository/ashokit\_release\_repo/

Note: Based on <version/> name available in project pom.xml file it will decide artifacts should be stored to which repository

Nexus Repository details we will configure in project pom.xml file like below

**<distributionManagement>**

**<repository>**

<id>nexus</id>

<name>Ashok IT Releases Nexus Repo</name>

<url>http://15.207.19.102:8081/repository/ashokit-release-repository/</url>

**</repository>**

**<snapshotRepository>**

<id>nexus</id>

<name>Ashok IT Snapshots Nexus Repo</name>

<url>http://15.207.19.102:8081/repository/ashokit-snapshot-repository/</url>

**</snapshotRepository>**

**</distributionManagement>**

Nexus Server Credentials will be configured in Maven **"settings.xml"** file.

Maven Location: **C:\apache-maven-3.8.5\conf**

In settings.xml file, under <servers> tag add below <server> tag

<server>

<id>nexus</id>

<username>admin</username>

<password>admin@123</username>

</server>

Once these details are configured then we can run below maven goal to upload build artifacts to Nexus Server

**$ mvn clean deploy**

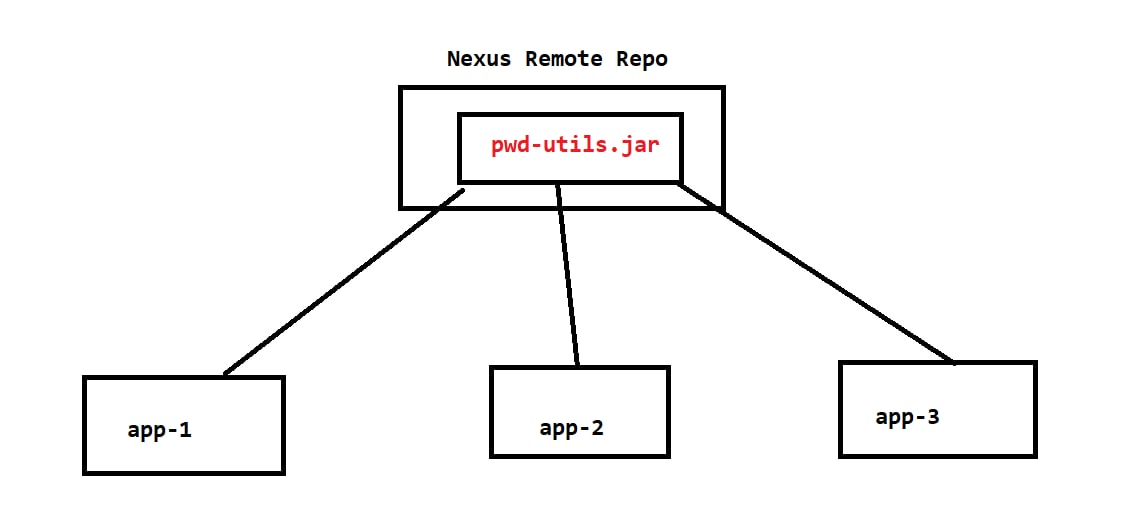
**Remote Repository**

Remote repository used for **shared libraries**

If we want to use **few jar files in multiple projects** in the company then we will **use Remote Repository**

**Remote repository is specific to our company projects**

Create remote repo in nexus and upload a jar file



Take dependency details of uploaded jar file and add in project pom.xml as a dependency

We need to add Remote Repository Details in pom.xml above <dependencies/> tag

<repositories>

<repository>

<id>nexus</id>

<url>repo-url</url>

</repository>

</repositories>

After adding the remote repository details do maven package and see dependency is downloading from nexus repo or not.

**We will create users and will give access for users for our repositories**