**Jenkins**

**Agenda:**

1. Installing Jenkins, setting up, configuration ()

2. Security LDAP - authorization / security realm – Jenkins in built user management

3. How to schedule a job and different type of jobs

4. Integration:

Master Slave Architecture -https://github.com/miztiik/DevOps-Demos

* Bring an instance
* Move to HOME\_DIR and “Ssh-key gen” to generate public and private keys
* Modify the permission
* Copy the public key to Authorized\_key folder
* In master copy the publickey to .ssh/know\_hosts folder
* Ssh-key-Scan to connect to slave
* In Jenkins UI create gobbal credentials with slave username and private key

Integration with GIT/GITHUB

Integration with Maven – Gobal tool configuration

Integration with Nexus/jfrog (configure system and give the repo url , ID and password)

Integration with Ansible

5. Troubleshooting (location of log files of startup, failed plugin and etc)

6. How to schedule backups

7. Restore from backups

8. Configure Slaves

9. Monitoring the workload

10. Managing plugins

11. Difference b/w declarative and scripted pipeline

12. Git parameter

13. Environment variable

14. Credentials

15. Environment directive

16. Various plugin

17. Build Errors

**What is Jenkins:**

Leading open source continuous Integration server

Highly configurable system by itself

400+ community developed plugins

Multi platform

Easy to use

Free & Open source

<https://jenkins.io/index.html>

**Features:**

* Jenkins provides continuous Integration services for software deployment.
* Based on tomcat
* Originally started as Hudson then renamed as Jenkins
* leading open source continuous integration server.
* Flexibility
* Jenkins is a highly configurable system by itself.
* 400+ community developed plugins
* Less Effort to Integrate existing existing test buckets.
* Multi platform
* Easy to use
* Free & Open Source
* Documentation: https://jenkins.io/index.html

Failing is Ok, but we need to fail fast

Catching a bug in test costs less than catching in production

Code quality review

Proper documentation

This can be done using custom scripts, but as the project increases it is difficult to maintain.

There was split in Hudson when oracle took over SUN,

90% Hudson and Jenkins are same

Build can be started by

Commit a version in version control

Scheduling via cron like mechanism

Tests can be executed every week/hour etc

If A fails do B, if A passes do C – jobs can be dependent like a pipeline

**What is a build:**

Build has many components – one of them is version control

No matters how many times a code changes, build has to happen, code has to be compiled.

Build also has to be tested

Communicate the results to all stake holders

* Product owner
* System admin
* QA
* Developer

History – can be maintained

Continuous delivery vs Continuous deployment

Continuous Deployment: deploying the product to customers continuously without approval, this is automatic

Continuous delivery: approval is needed, ready to deploy. This is not automatic

Test cases have to be automated for Continuous Integration

**Download Jenkins:**

Download LTS – long term support , not weekly build which is not safe

Download native packages not war packages

Jenkins rpm comes with webserver

Download Stable Jenkins rpm from <http://pkg.jenkins-ci.org/redhat-stable/>

1. Install it using rpm –ivh

root@ctx3p12 MISC]# rpm -ivh jenkins-2.9-1.1.noarch.rpm

warning: jenkins-2.9-1.1.noarch.rpm: Header V4 DSA/SHA1 Signature, key ID d50582e6: NOKEY

Preparing... ################################# [100%]

Updating / installing...

1:jenkins-2.9-1.1 ################################# [100%]

1. Install Java using yum
2. Start the Jenkins service

[root@ctx3p12 MISC]# sudo systemctl start jenkins.service

[root@ctx3p12 MISC]# cat /var/lib/jenkins/secrets/initialAdminPassword

a45c1a99da7c49518ce07b7db91950e4

[root@ctx3p12 MISC]#

firewall-cmd --permanent --zone=public --add-port=8080/tcp

to open port 8080 on Jenkins server

Jenkins by default listens on port 8080

So open <http://FQDN:8080>

To change port

**/etc/sysconfig/jenkins**

http port to 8089

jenkins home

/var/lib/jenkins/config.xml

How to install

sudo yum -y install java

java is a prerequisite for Jenkins.

yum whatprovides service

sudo wget -O /etc/yum.repos.d/jenkins.repo <http://pkg.jenkins-ci.org/redhat/jenkins.repo>

* Downloads the repo file from Jenkins site

sudo rpm --import <https://jenkins-ci.org/redhat/jenkins-ci.org.key>

* Imports the key from Jenkins site for authentication

sudo yum install jenkins

* Downloads and installs the Jenkins packages from Jenkins repository

sudo systemctl status jenkins.service

* Start the service and check the status – it should be Active

Download Jenkins

Install it

Configure plugins

Install all of them

Install GIT related them

Installing all may slow down Jenkins

Its not recommended to enable auto refresh

URL/restart 🡪 will restart Jenkins

Manage Jenkins 🡪 Configure global security 🡪 click on enable security

Authorization 🡪 any one can do any thing

Security realm Jenkins own user database

LDAP is for corporate security to sign up using intranet id

Manage Jenkins 🡪 project based authority 🡪 enable everything for admin enable all permissions

Setup security

Click on Enable security

Difference between save and apply

Apply will remain in the same, save will go back to previous page

Manage Jenkins -> configure system

No. of executors – no. of core you have on CPU

GIT hub server - create user id in git hub server

Add git hub server

Go to client

Git clone <https://githubg.com/RSCTFVT/>

Commit back from SVN

Add a file here

Git add \*

Git commit –m “some commit”

Jenkins was developed by Kohsuke Kawaguchi.

**Backup of Jenkins:**

1. Take backup of home directory using tar -cvf command

tar -cvf jenkins-backup-21dec.tar Jenkins

gzip jenkins-backup-21dec.tar

then copy it to other machine

Incase of crash, extract this file in newly created Jenkins server

1. Take backup of single job using xml

[http://ctx3p11.in.company.com:8080/job/RSCTBAT/config.xml](http://ctx3p11.in.ibm.com:8080/job/RSCTBAT/config.xml)

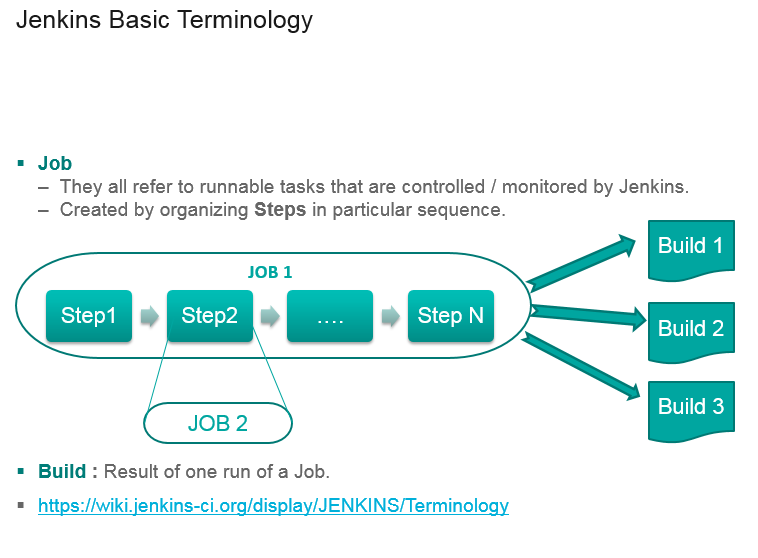
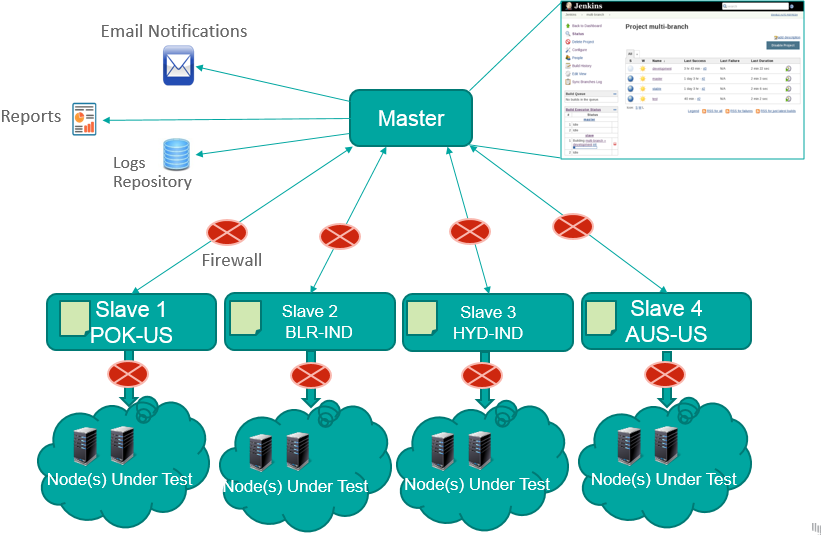
open a job with config.xml extension and take backup in a file.

After installation:

[root@jupiter-vm751 ~]# cat /var/lib/jenkins/secrets/initialAdminPassword

1b5561f487c14ea3b9c6c386eca33cfc

**Configuring Slaves:**



**GIT**

Git is becoming popular in Devops world

Git is to keep track of changes

Ansible/puppet/chef/vagrant – files we store on GITHUB

Why GIT :

1. Build & Release - devops eng works with developers
2. For storing automation scripts
3. Open source - so many open source projects are hosted on GIT (Linux, chef, puppet etc)

System admins take backup by copying.

Cp file file\_1

Cp file\_1 file\_2

Its difficult to track

Or we can name with date

That is also difficult to track

If there are many scripts – how to handle ?

There should be a centralized location to keep track of all changes – that is called version control system

Don’t keep track

Save numbered zip file

Fomal version control

Diff , windiff – can see the difference

Even that is difficult

Developed by Linux torwalds – keep track of source code

Used for just for source code

Do not use for file storage

Its fast

Don’t need access to server

Good at merging simultaneous changes

Everyone is using it

Bitbucket is alternative

GIT HUB – to store our scripts

Singup to github

Just give username and password

Login to it

Create new repository

Other CI tools:

Hudson

Bamboo

Microsoft tfs – team foundation server

Mercuyrail

**Configuring slaves on the Jenkins master:**

1. Create jenkins user on the slave

***useradd -d "/home/jenkins" -c "jenkins user" -s "/bin/bash" -m jenkins***

and set the password to slave as root

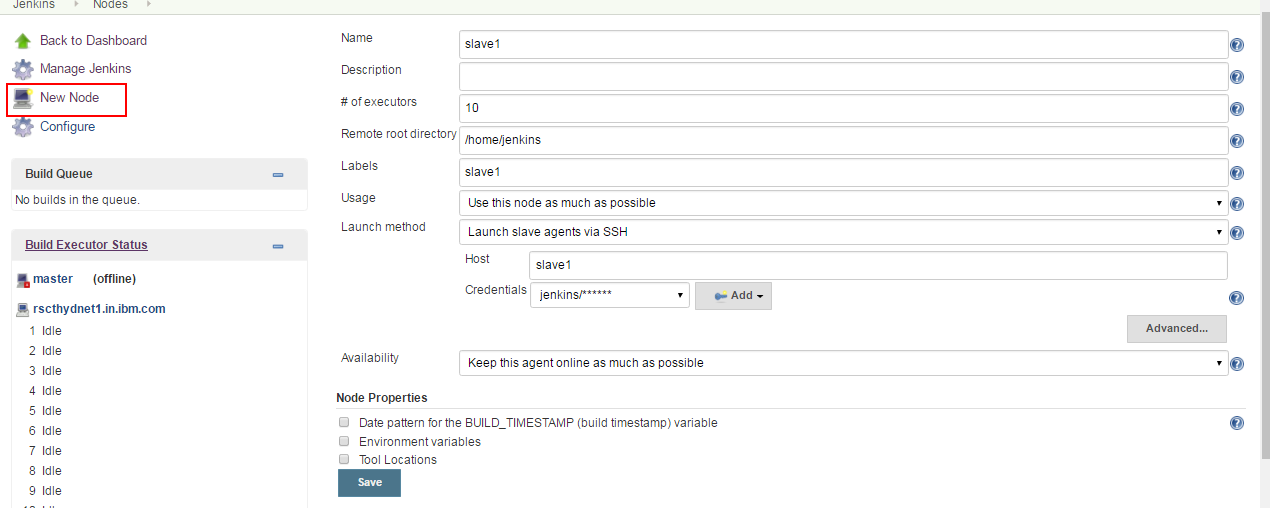
***passwd Jenkins***

1. Configure password less authentication on the server and slave for the Jenkins user

***Login as Jenkins to master and execute***

***ssh-copy-id -i ~/.ssh/id\_rsa.pub jenkins@rscthydnet1***

1. Go to Jenkins 🡪 manage nodes 🡪 New node,
2. Select as Permanent agent
3. Select options as below



Once slave is added, make it online by

Clicking on

Manage Jenkins 🡪 Manage Node 🡪 Click on Slave 🡪 Launch agent

This will make the slave online

**Integration with GIT:**

Install git on slave

Register your self on github and create a sample project

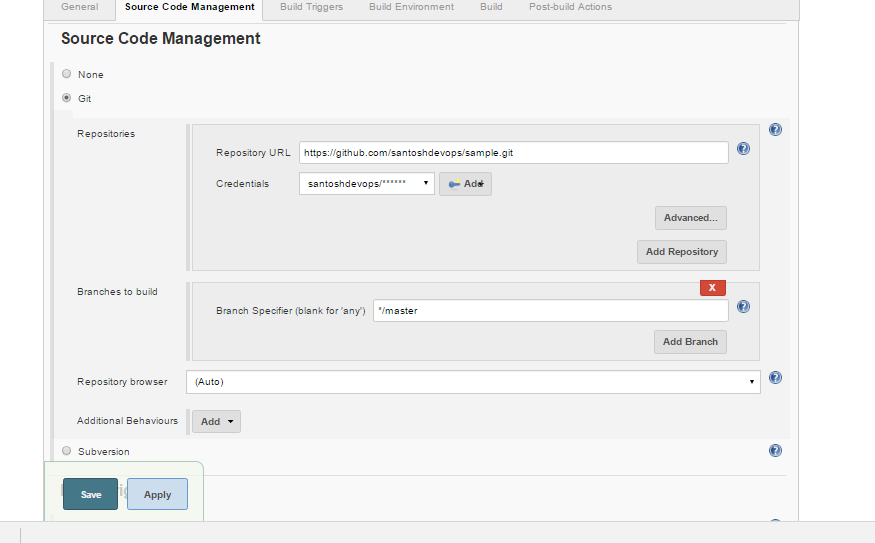
Add giit plugin to Jenkins

Create a new job

Select Source Code Management as git

Provide the Repository URL as the git url (copy from clone or download option on git hub)

Provide the credentials (Same as github)



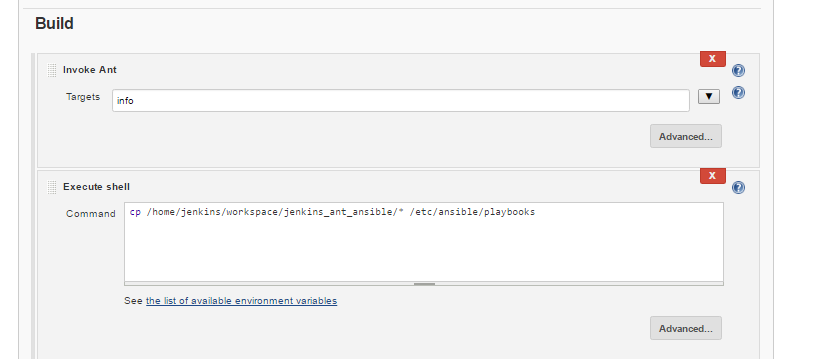
Select build triggers as Poll SCM for every minute - \* \* \* \* \*

To run this project on a particular slave

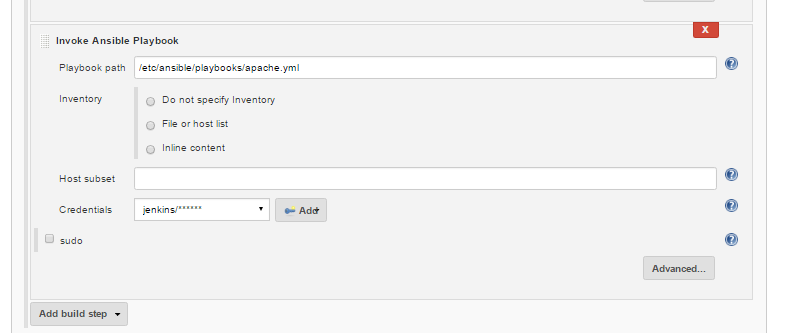
Click the checkbox

Restrict where this project can be run and select lable of the slave.

**Ant:**



**Ansible**



**Integration with Nexus build repository:**

Nexus is a build repository stores the builds in a hierarchical directory structure.

Nexus can

* Gives a centralized repository for managing all popular component formats.
* Gain insight into component security, license, and quality issues.
* Improve productivity by efficiently distributing components to developers around the corner, or around the world.
* Modernize software development with intelligent staging and release functionality.
* Sleep comfortably with world-class support and training.

**How to install Nexus:**

cd /usr/local/

wget http://sonatype.org/downloads/nexus-latest-bundle.tar.gz

gunzip nexus-latest-bundle.tar.gz

tar -xvf nexus-latest-bundle.tar

mv nexus-2.14.3-02 sonatype-work /usr/local

ln -s nexus-2.14.3-02 nexus

cd nexus

export RUN\_AS\_USER=root

***./nexus start***

Make sure nexus is running by executing “ ps -ef | grep -i nexus

“

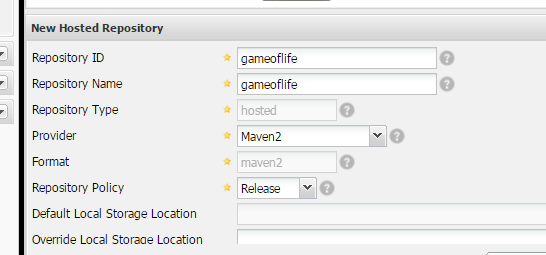
admin/admin123 – default password for nexus login.

Open nexus GUI:

**URL: http://rscthydnet2:8081/nexus/#welcome**

Create a repository in Nexus to host builds

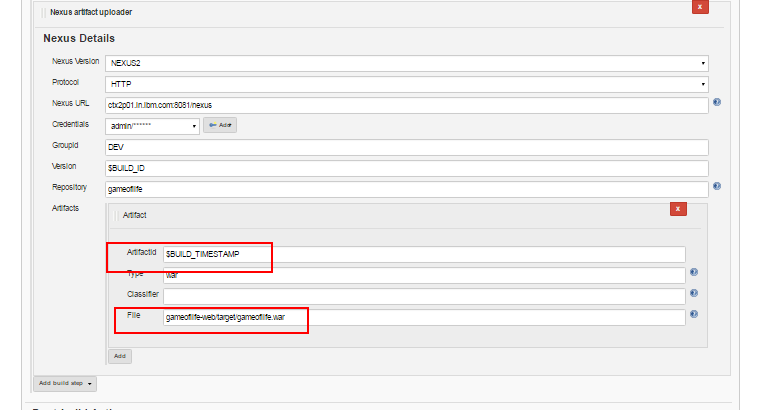
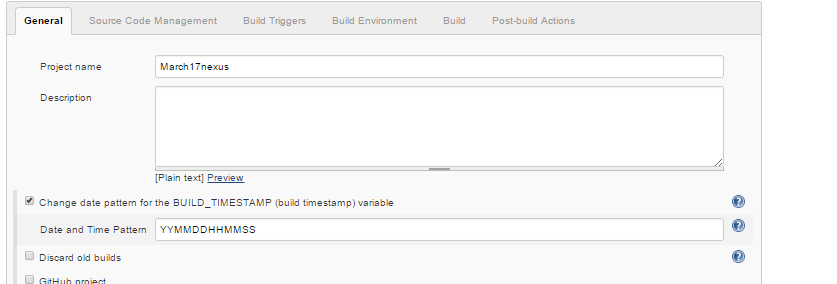
Go to repository 🡪 add hosted repository



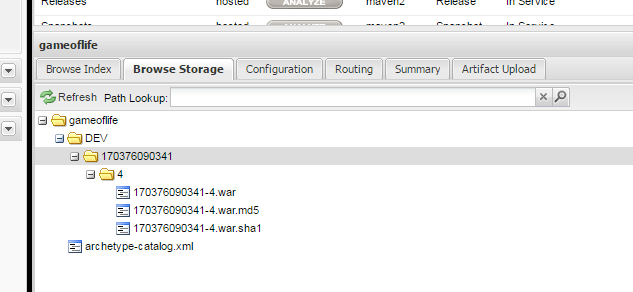
In the Jenkins add the plugin **Nexus Artifact Uploader**

Also install the zen timestamp plugin to use $BUILD\_TIMESTAMP variable in the job

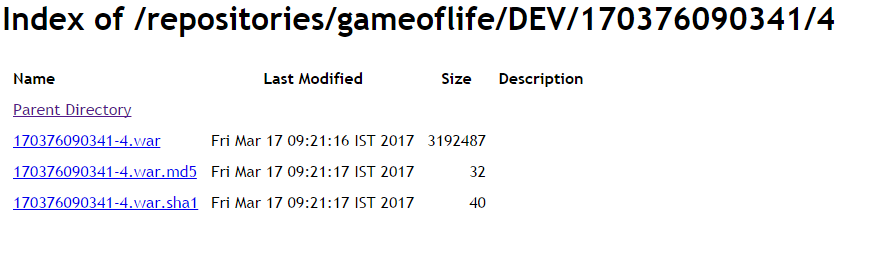
Configure the job as below



After job is executed nexus creates repos as below



And it can be accessed using a link



**Jenkins logs:**

Jenkins logs will be stored under :

/var/log/jenkins/jenkins.log

**Some commonly used plugins:**

Extended choice parameter

Build with parameters

Rebuild now

Html report generator

Junit report generator

Thinkbackup

Monitoring

Crontab syntax:

\* \* \* \* \*

minutes 0-59

hours 0-12

day of the month 1-31

month of the year 1-12

day of the week sun,mon,tue,wed,thu,fri,sat

command/script

crontab - utility to schedule the jobs

10AM - good morning

9AM, 5PM - cleanup the logs

1st of month --> msg

10 \* \* \* \* echo Hi --> every 10 minutes

\* 10 \* \* \* date >> /file --> every day 10AM

\* 17 \* \* \* /scr --> every day 17th hour - 5PM

\* 10,17 \* \* \* /script

\* 10,11,12,17 \* \* /script

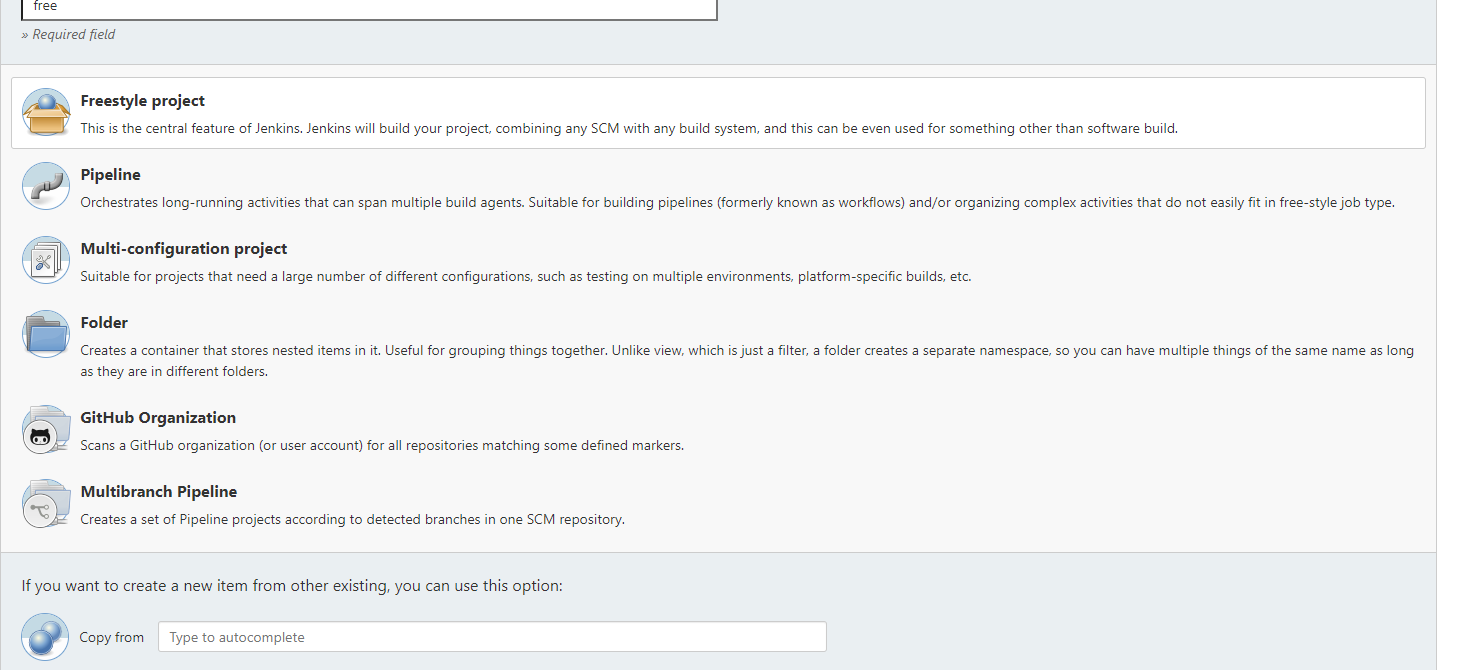
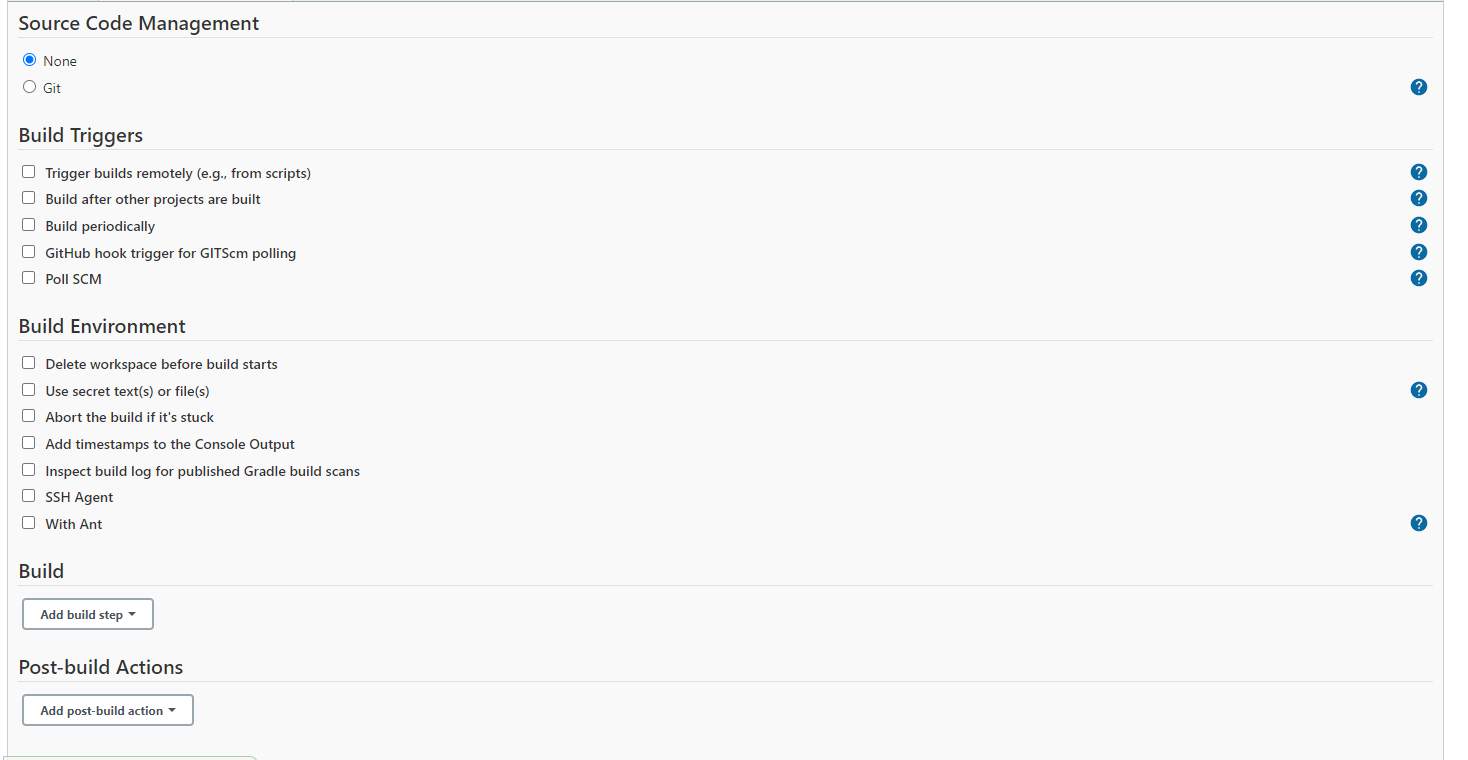
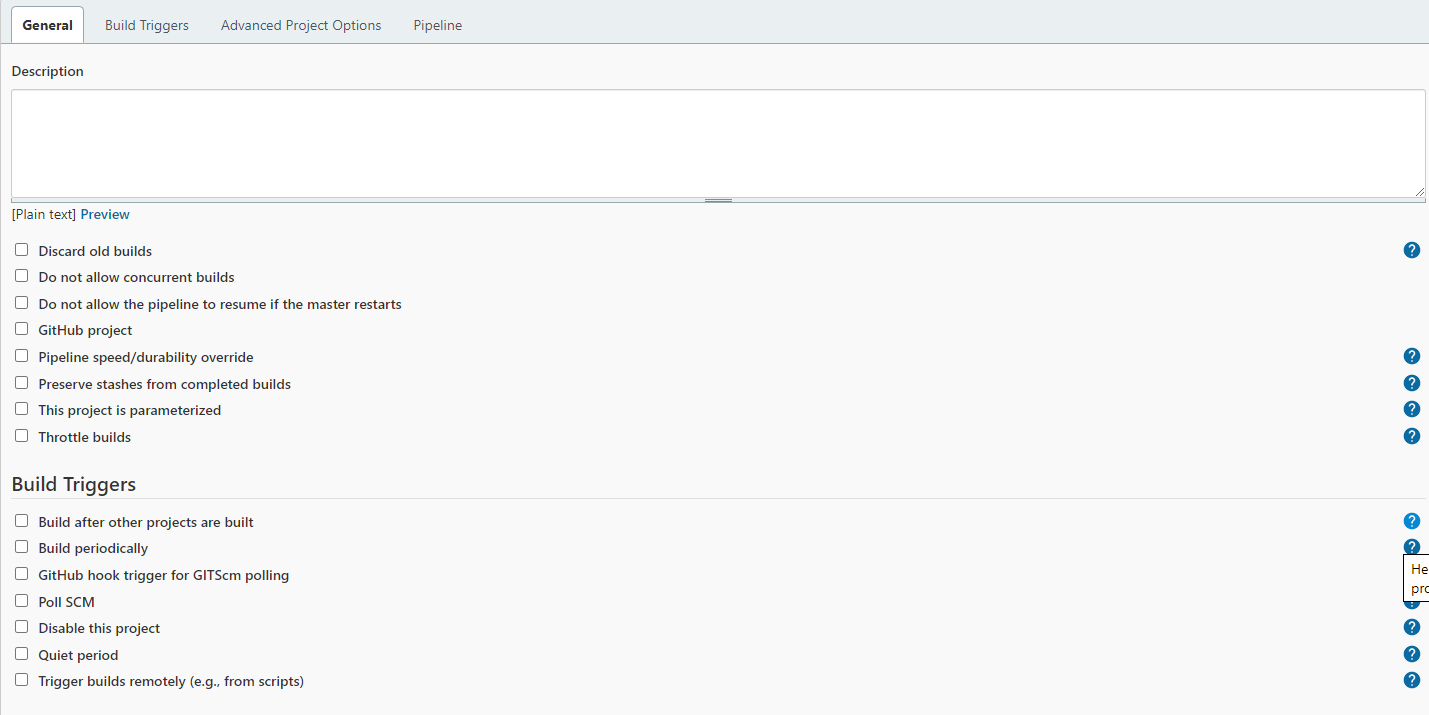
\* \* 1 \* \* /scr --> every month on 1st

**To have upstream job and downstream job:**

Use a plugin called Jobfan in

While creating job, we can have a condition like, if Job A is built successfully then only start the job B.

If build is successful, then go for test.



Valaxy- miztiik , ravdy,

Yb madu- jmstechhome8 and ybmadhu

Deekshith SN

|  |  |
| --- | --- |
| Decelarative | groovy |
| Syntax Starts with pipeline  Simple and flexible  Input Parameter are declared in properties block and variable are defined in environment block  We can track the changes in VCS using Jenkins file  Try/catch structure is not allowed like any other Groovy syntax. The custom step warnError is used to manage build state  To created/ access the function in declarative pipeline we can use Shared libraries/ Scripts  warnError  git parameter , parameterized job, blue ocean, thin backup, publish over ssh GitHub plugin, GitHub Branch Source plugin, GitHub API plugin, Gitclient plugin, Pipeline Utility Steps | Syntax Starts with agent  Strict groovy based syntax  Parameter are decelerated in Properties block  And variables are defined in Groovy syntax [def artifact]  Script should be written in UI  Error control is managed with a try/catch clause in Groovy Syntax |

|  |
| --- |
| pipeline { |
|  | agent { |
|  | label "master" |
|  | } |
|  | tools { |
|  | // Note: this should match with the tool name configured in your jenkins instance (JENKINS\_URL/configureTools/) |
|  | maven "Maven3" |
|  | } |
|  | environment { |
|  | // This can be nexus3 or nexus2 |
|  | NEXUS\_VERSION = "nexus3" |
|  | // This can be http or https |
|  | NEXUS\_PROTOCOL = "http" |
|  | // Where your Nexus is running |
|  | NEXUS\_URL = "localhost:8081" |
|  | // Repository where we will upload the artifact |
|  | NEXUS\_REPOSITORY = "dvs-evn-spring" |
|  | // Jenkins credential id to authenticate to Nexus OSS |
|  | NEXUS\_CREDENTIAL\_ID = "nexus\_credentials" |
|  | } |
|  | stages { |
|  | stage("clone code") { |
|  | steps { |
|  | script { |
|  | // Let's clone the source |
|  | git 'https://github.com/ybmadhu/spring3-mvc-maven-xml-hello-world.git'; |
|  | } |
|  | } |
|  | } |
|  | stage("mvn build") { |
|  | steps { |
|  | script { |
|  | // If you are using Windows then you should use "bat" step |
|  | // Since unit testing is out of the scope we skip them |
|  | bat(/${MAVEN\_HOME}**\b**in**\m**vn -Dmaven.test.failure.ignore clean package/) |
|  | } |
|  | } |
|  | } |
|  | stage("publish to nexus") { |
|  | steps { |
|  | script { |
|  | // Read POM xml file using 'readMavenPom' step , this step 'readMavenPom' is included in: https://plugins.jenkins.io/pipeline-utility-steps |
|  | pom = readMavenPom file: "pom.xml"; |
|  | // Find built artifact under target folder |
|  | filesByGlob = findFiles(glob: "target/\*.${pom.packaging}"); |
|  | // Print some info from the artifact found |
|  | echo "${filesByGlob[0].name} ${filesByGlob[0].path} ${filesByGlob[0].directory} ${filesByGlob[0].length} ${filesByGlob[0].lastModified}" |
|  | // Extract the path from the File found |
|  | artifactPath = filesByGlob[0].path; |
|  | // Assign to a boolean response verifying If the artifact name exists |
|  | artifactExists = fileExists artifactPath; |
|  | if(artifactExists) { |
|  | echo "\*\*\* File: ${artifactPath}, group: ${pom.groupId}, packaging: ${pom.packaging}, version ${pom.version}"; |
|  | nexusArtifactUploader( |
|  | nexusVersion: NEXUS\_VERSION, |
|  | protocol: NEXUS\_PROTOCOL, |
|  | nexusUrl: NEXUS\_URL, |
|  | groupId: pom.groupId, |
|  | version: '${BUILD\_NUMBER}', |
|  | repository: NEXUS\_REPOSITORY, |
|  | credentialsId: NEXUS\_CREDENTIAL\_ID, |
|  | artifacts: [ |
|  | // Artifact generated such as .jar, .ear and .war files. |
|  | [artifactId: pom.artifactId, |
|  | classifier: '', |
|  | file: artifactPath, |
|  | type: pom.packaging], |
|  | // Lets upload the pom.xml file for additional information for Transitive dependencies |
|  | [artifactId: pom.artifactId, |
|  | classifier: '', |
|  | file: "pom.xml", |
|  | type: "pom"] |
|  | ] |
|  | ); |
|  | } else { |
|  | error "\*\*\* File: ${artifactPath}, could not be found"; |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |

**Answer:** Maven is a build management tool. Using a simple pom.xml, one can configure all the dependencies needed to build, test and run the code. Maven manages the full lifecycle of a test project. Once integrated with Jenkins, the maven Webdriver will build the project and execute all tests efficiently.+

Container technology, also simply known as just a container, is a method to package an application along with its dependencies, so it can be portable, lightweight and can be run isolated from other processes

Docker is based on Linux LXC container technology developed in go language

1. **Docker Client**: This is how you interact with your containers. Call it the user interface for Docker.
2. **Docker Daemon**: A background process responsible for receiving commands and passing them to the containers via command line.
3. **Docker Registry**: Commonly known as Docker Hub, this is where your container images are stored and retrieved.

Docker is primarily used to package an application’s code and its dependencies. The same container can be shared from Dev to QA and later to IT, thus bringing portability to the development pipeline.

You can use Docker to isolate individual applications, and use Virtual Machines to isolate entire systems. They are operating at different levels of abstraction.

## Advantages of Virtual Machines

* The tools associated with a virtual machine are easier to access and simpler to work with. Docker has a more complicated tooling ecosystem that consists of both Docker-managed and third-party tools.
* As mentioned earlier, once you have a virtual machine up and running, you can start a Docker instance within that VM, and run docker container within the VM (which is the predominant method of running containers at present). This way, containers and virtual machines are not mutually exclusive and can co-exist alongside each other.

|  |  |
| --- | --- |
| Docker | VM |
| 1. Lite weight, portable and easy to share with other team and helps in reducing the configuration error in different environment 2. OS level virtualization 3. No guest OS | 1. Requires more resource and boot time is slow 2. Hardware level virtualization 3. Run application in guest OS |

Docker Commands - http://containertutorials.com/get\_started/index.html

The difference between “docker run” and “docker exec” is that “docker exec” executes a command on a running container. On the other hand, “docker run” creates a temporary container, executes the command in it and stops the container when it is done.

When running “**docker exec**” with the “-i” option, **you are binding the standard input of your host to the standard input of the process you are running in the container.**

1. Docker build –t (tag) –f(filename)
2. –w workdirectory
3. –e environemt variable
4. –net=”bridge” to mention the network
5. –v or –mnt containerolume:hostvolumepath
6. –p port = hostport:container port

**Docker Compose** is used to run multiple containers as a single service. For example, suppose you had an application which required NGNIX and MySQL, you could create one file which would start both the containers as a service without the need to start each one separately.

🔥 What is Kubernetes – container orchestration tool – manage application running in containers

What problems does Kubernetes solve? - HA, scalability and load balancing & DR

What features do container orchestration tools offer?

Main K8s Components ------------------------------------------------

* Node – Virtual machine where Pods run
* Pod – abstraction over container so that we can communicate with k8 componenet
* Containers in the same pod share a local network and the same resources, allowing them to easily communicate with other containers in the same pod as if they were on the same machine while at the same time maintaining a degree of isolation.
* Service – service component is like **permanent IP** used to communicate between pods(application POD and DB pod) incase pod dies the service will be available and attached to new POD
* External service – connecting application via node IP: port hhtp:192.10.10.0:8080
* Ingress – used to route traffic to the cluster in Worker node with domain name hhtps://myapp.com
* ConfigMap – like environment variable holds the DB\_URL which can be changed in future
* Secret – holds the DB\_passswd/credentials with encrypted form
* Volumes – are used for data persistent
* Deployment – blue print of PODS, we can use replica set in Deployment to bring new apllication
* StatefulSet – are used for replication of DB, since the data is sateful

K8s Architecture --------------------------------------------------

* Worker Nodes – requires [container runtime, **kubelet**- interact with container runtime and node- helps in run the container and assign resources, **kubeproxy**- forward the request via SERVICE helps in load balancing ]
* Master Nodes –[API server- used to intract with master node via CLI, also checks the authentication]
* Api Server - - [used to intract with master node via CLI, also checks the authentication and frwd request]
* Scheduler – [schedules the POD in Worker node, check the available resources]
* Controller Manager –[POD die on nodes are detect through state change and re schedule the PODS]
* etcd - the cluster brain –[Key-value store of the cluster information ]

### **2. What is ClusterIP?**

The ClusterIP is the default Kubernetes service that provides a service inside a cluster (with no external access) that other apps inside your cluster can access.

### **23. What is NodePort?**

The NodePort service is the most fundamental way to get external traffic directly to your service. It opens a specific port on all Nodes and forwards any traffic sent to this port to the service.

### **24. What is the LoadBalancer in Kubernetes?**

The LoadBalancer service is used to expose services to the internet. A Network load balancer, for example, creates a single IP address that forwards all traffic to your service.

Top 30 Kubernetes Interview Questions and Answers in 2021[Updated]

Lesson 4 of 5By Simplilearn

Last updated on Apr 20, 202182890

PreviousNext

Kubernetes comes from a Greek word meaning ‘captain,’ ‘helmsman,’ or ‘governor.’ The term is now also used in the DevOps and on-premises software development world to refer to a powerful bundle of solutions that equips operations engineers to scale and service server (and box) setups effortlessly. Kubernetes was created by Joe Beda, Craig McLuckie, and Brendan Burns, who were later joined by Google engineers before officially releasing it in 2014. Today, Kubernetes is maintained by Cloud Native Computing Foundation (CNCF) and has evolved into a fast-growing and widely used ecosystem.

By now, you’re probably wondering, what does Kubernetes do? Well, the answer to this question can’t be fully explained within the scope of this Kubernetes interview questions article. After all, there are entire Kubernetes courses that are designed to answer this question, including how to use it.

However, what we’ll cover here are some frequently asked Kubernetes interview questions and answers. These questions and answers will help you prepare for any interview or certification exam that you may need to take once you’ve completed the Kubernetes training. So, without further ado, let's jump right in and learn the top Kubernetes interview questions and answers.

Enhance your Kubernetes skills and gain credibility in the field with the Certified Kubernetes Administrator Certification Training. Enroll now!

Kubernetes Interview Questions and Answers

1. What is Kubernetes?

Kubernetes is an open-source container orchestration tool or system that is used to automate tasks such as the management, monitoring, scaling, and deployment of containerized applications. It is used to easily manage several containers (since it can handle grouping of containers), which provides for logical units that can be discovered and managed.

2. What are K8s?

K8s is another term for Kubernetes.

3. What is orchestration when it comes to software and DevOps?

Orchestration refers to the integration of multiple services that allows them to automate processes or synchronize information in a timely fashion. Say, for example, you have six or seven microservices for an application to run. If you place them in separate containers, this would inevitably create obstacles for communication. Orchestration would help in such a situation by enabling all services in individual containers to work seamlessly to accomplish a single goal.

FREE DevOps Certification Training

Master in-demand DevOps tools and skillsENROLL FOR FREEFREE DevOps Certification Training

4. How are Kubernetes and Docker related?

Docker is an open-source platform used to handle software development. Its main benefit is that it packages the settings and dependencies that the software/application needs to run into a container, which allows for portability and several other advantages. Kubernetes allows for the manual linking and orchestration of several containers, running on multiple hosts that have been created using Docker.

5. What are the main differences between the Docker Swarm and Kubernetes?

Docker Swarm is Docker’s native, open-source container orchestration platform that is used to cluster and schedule Docker containers. Swarm differs from Kubernetes in the following ways:

Docker Swarm is more convenient to set up but doesn’t have a robust cluster, while Kubernetes is more complicated to set up but the benefit of having the assurance of a robust cluster

Docker Swarm can’t do auto-scaling (as can Kubernetes); however, Docker scaling is five times faster than Kubernetes

Docker Swarm doesn’t have a GUI; Kubernetes has a GUI in the form of a dashboard

Docker Swarm does automatic load balancing of traffic between containers in a cluster, while Kubernetes requires manual intervention for load balancing such traffic

Docker requires third-party tools like ELK stack for logging and monitoring, while Kubernetes has integrated tools for the same

Docker Swarm can share storage volumes with any container easily, while Kubernetes can only share storage volumes with containers in the same pod

Docker can deploy rolling updates but can’t deploy automatic rollbacks; Kubernetes can deploy rolling updates as well as automatic rollbacks

6. What are the main components of Kubernetes architecture?

There are two primary components: the master node and the worker node. Each of these components has individual components in them.

7. What is a node in Kubernetes?

A node is the smallest fundamental unit of computing hardware. It represents a single machine in a cluster, which could be a physical machine in a data center or a virtual machine from a cloud provider. Each machine can substitute any other machine in a Kubernetes cluster. The master in Kubernetes controls the nodes that have containers.

8. What does the node status contain?

The main components of a node status are Address, Condition, Capacity, and Info.

9. What process runs on Kubernetes Master Node?

The Kube-api server process runs on the master node and serves to scale the deployment of more instances.

10. What is a pod in Kubernetes?

Pods are high-level structures that wrap one or more containers. This is because containers are not run directly in Kubernetes. Containers in the same pod share a local network and the same resources, allowing them to easily communicate with other containers in the same pod as if they were on the same machine while at the same time maintaining a degree of isolation.

11. What is the job of the kube-scheduler?

The kube-scheduler assigns nodes to newly created pods.

12. What is a cluster of containers in Kubernetes?

A cluster of containers is a set of machine elements that are nodes. Clusters initiate specific routes so that the containers running on the nodes can communicate with each other. In Kubernetes, the container engine (not the server of the Kubernetes API) provides hosting for the API server.

13. What is the Google Container Engine?

The Google Container Engine is an open-source management platform tailor-made for Docker containers and clusters to provide support for the clusters that run in Google public cloud services.

14. What are Daemon sets?

Daemon set ensure all the nodes have the copy of PODS

A Daemon set is a set of pods that runs only once on a host. They are used for host layer attributes like a network or for monitoring a network, which you may not need to run on a host more than once.

What is a Namespace in Kubernetes?

Namespaces are used for dividing cluster resources between multiple users. They are meant for environments where there are many users spread across projects or teams and provide a scope of resources.

kubectl Commands

kubectl CREATE EDIT DELETE Deployment [name]

kubectl get nodes or pod or replicaset or deployment or service

kubectl logs POD NAME

kubectl exec –it POD

namespace

ingress

volume and persist data

Cloudformation ability to code our infra and helps in reusability reduce time to bring the infra

Templates components (one course section for each):

1. Resources: your AWS resources declared in the template (MANDATORY)

2. Parameters: the dynamic inputs for your template received from user

3. Mappings: the static variables for your template **!FindInMap[]**

4. Outputs: References to what has been created

5. Conditionals: List of conditions to perform resource creation

6. Metadata

Templates helpers:

1. References – refer your parameter **!Ref**

2. Functions

AWS EBS volume resize

* Modify the EBD volume and increase the size
* aws ec2 modify-volume --region <regionName> --volume-id <volumeId> --size <newSize> --volume-type <newType> --iops <newIops>
* Run the diskmgmt.msc and extend the volume in windows
* Linux Extend the partition by typing sudo growpart /dev/xvda 1
* df –hT and get the file system type
* lsblk
* To extend the volume in file system XFS, use the **xfs\_growfs** command
* To extend the volume in file system EXT4, use the , resize2fs /dev/xvda1 .

Jenkins

|  |  |
| --- | --- |
| Global Variable Reference: --------------------------------------  Env  Params  currentBuild | Example :  env.PATH or env.BUILD\_ID  Exposes all parameters defined for the Pipeline as a read-only [Map](http://groovy-lang.org/syntax.html#_maps), for example: params.MY\_PARAM\_NAME.  currentBuild.result, currentBuild.displayName |
| Input, options, when , tool, stash, Post | Options = such as buildDiscarder, timeout, timestamp.  Tools = A section defining tools to auto-install and put on the PATH. This is ignored if agent none is specified |
| Secrets  Different types   1. **Secret text** - a token such as an API token (e.g. a GitHub personal access token), 2. **Username and password** - 3. **Secret file** - 4. **SSH Username with private key** - **Username**, **Private Key** and optional **Passphrase** into their respective fields. 5. **Certificate** - a [PKCS#12 certificate file](https://tools.ietf.org/html/rfc7292) and optional password, or 6. **Docker Host Certificate Authentication** credentials. | While creating secrets choose **scope** system(not available for jobs) or **global(available to jobs)**  Credentials and Credential Bindings Plugin are required  To maintain the security and anonymity of these credentials, if the job displays only the \*\*\*\* in build output value  environment {  AWS\_ACCESS\_KEY\_ID = credentials('jenkins-aws-secret-key-id')  AWS\_SECRET\_ACCESS\_KEY = credentials('jenkins-aws-secret-access-key')  /home/user/.jenkins/workspace/cred\_test@tmp/secretFiles/546a5cf3-9b56-4165-a0fd-19e2afe6b31f/kubeconfig.txt  MY\_KUBECONFIG = credentials('my-kubeconfig')  withCredentials(bindings: [sshUserPrivateKey(credentialsId: 'jenkins-ssh-key-for-abc',keyFileVariable: 'SSH\_KEY\_FOR\_ABC')])  withCredentials([string(credentialsId: 'mytoken', variable: 'TOKEN')])  Note: The single-quotes will cause the secret to be expanded by the shell as an environment variable. The double-quotes are potentially less secure as the secret is interpolated by Groovy |
| PLUGIN – active choice plugin,  Publish over shh  Archive artifacts  Pipeline utiliy plugin |  |
| Shared library  Folder structure  Var – contains groovy code with.groovy extension  Src- add java class path for every script execution  Resource – non groovy files are placed in this folder | Concept of have common scripts/ groovy code in repo which can be reused in different pipeline  Example – filter logs code which can be re used in all pipeline for |
|  |  |

|  |  |
| --- | --- |
| Declarative Pipeline supports an [environment](https://www.jenkins.io/doc/book/pipeline/syntax/#environment) directive.  pipeline {  agent any  environment {  CC = 'clang'  } | , whereas users of Scripted Pipeline must use the withEnv step.  node {  /\* .. snip .. \*/  withEnv(["PATH+MAVEN=**${**tool 'M3'**}**/bin"]) {  sh 'mvn -B verify'  }  } |
| Parameter block (different type include =string, choice, text, Boolean)  Declarative Pipeline supports parameters out-of-the-box, allowing the Pipeline to accept user-specified parameters at runtime via the [parameters directive](https://www.jenkins.io/doc/book/pipeline/syntax/#parameters).  *Jenkinsfile (Declarative Pipeline)*  pipeline {  agent any  parameters {  string(name: 'Greeting', defaultValue: 'Hello', description: 'How should I greet the world?')  } | Configuring parameters with Scripted Pipeline is done with the properties step, which can be found in the Snippet Generator  properties([parameters([string(defaultValue: 'Hello', description: 'How should I greet the world?', name: 'Greeting')])])  node {  echo "**${**params.Greeting**}** World!"  } |
| Handling failure Declarative Pipeline supports robust failure handling by default via its [post section](https://www.jenkins.io/doc/book/pipeline/syntax/#post) which allows declaring a number of different "post conditions" such as: always, unstable, success, failure, and changed. The [Pipeline Syntax](https://www.jenkins.io/doc/book/pipeline/jenkinsfile/#syntax) section provides more detail on how to use the various post conditions.  post {  always {  junit '\*\*/target/\*.xml'  }  failure {  mail to: team@example.com, subject: 'The Pipeline failed :('  }  }  } | Scripted Pipeline however relies on Groovy’s built-in try/catch/finally semantics for handling failures during execution of the Pipeline.  node {  /\* .. snip .. \*/  stage('Test') {  **try** {  sh 'make check'  }  **finally** {  junit '\*\*/target/\*.xml'  }  } |
|  |  |
|  |  |

Roof

3 tracktor jali

Sengal Stone = 1load = 32000 \* 5 =

Cement = 1 bag = 350\* 200

Steel:

Jail Small stone= 1 unit \* 4

Sand: 20000

Labour = 1000 per day

Wood work = (including labour and stell)

Window = 3\*3 =total 7

Tiles & plumbing & painting