**Jenkins Pipeline:**

* Is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.
* Pipeline provides an extensible set of tools for modelling simple to complex delivery pipelines “as code”.
* The definition of a Jenkins pipeline is written into a text file (a Jenkinsfile) which in turn is checked into a project’s source control repository.

**Note: Continuous delivery pipeline:**

* Is an automated expression of process of getting software from version control right to the Users or customers.

Creating a **Jenkinsfile** and **committing** it to **source control** provides a number of immediate **benefits:**

* Automatically creates a Pipeline build process for all branches and pull requests.
* Code review/iteration on the Pipeline (along with the remaining source code).
* Audit trail for the Pipeline.
* Single source of truth for the Pipeline, which can be view and edited by multiple team members.

**Defining a Pipeline:**

* Through Blue Ocean
* Through classic UI
* In SCM.

**Declarative** or **Scripted Pipeline:**

A Jenkinsfile can be written using two types of syntax (**Declarative** and **Scripted**)

Declarative Pipeline is most recent feature which provides richer syntactical feature and designed to make writing and reading Pipeline code easier.

**Pipeline:**

* A Pipeline is a user-defined model of a CD pipeline.
* A Pipeline’s code defines **entire build process**, which typically includes **stages** for **building an application**, **testing it** and **delivering it**.

**Node:**

A ***node*** is a machine which is part of Jenkins environment and capable of executing Pipeline.

**Stage:**

A ***stage*** block defines a conceptually distinct ***subset of tasks*** performed through the entire Pipeline. (e.g. “Build”, “Test” and “Deploy” stages)

**Step:**

A ***single task***, fundamentally, a ***step tells Jenkins*** what to do at a particular point in time (or “**step**” in the process). e.g. to execute the shell command make use the sh step: ***sh ‘make’***.

[\* Note: **stages** and **steps** are common elements in both Declarative and Scripted Pipeline syntax]

**Key Difference between Declarative Pipeline and Scripted Pipeline:**

* **Declarative pipeline** is a relatively new feature that supports the pipeline as code concept.it makes the pipeline code easier to read and write. ***This code is written in Jenkinsfile which can be checked into source control management system such as git***.
* ***Scripted pipeline*** is a traditional way of writing the code. In this pipeline, the **Jenkinsfile** is written on the ***Jenkins UI instance***.
* Though both these pipelines are based on the groovy DSL, the scripted pipeline uses stricter groovy based syntaxes.
* The ***Declarative Pipeline*** is ***defined*** a block labelled ‘***pipeline***” whereas the ***Scripted pipeline***is defined within a ”***node***”.

|  |  |
| --- | --- |
| *Jenkinsfile (Declarative Pipeline)*  pipeline {  agent any  stages {  stage('Build') {  steps {  //  }  }  stage('Test') {  steps {  //  }  }  stage('Deploy') {  steps {  //  }  }  }  } | *Jenkinsfile (Scripted Pipeline)*  node {  stage('Build') {  //  }  stage('Test') {  //  }  stage('Deploy') {  //  }  } |
| |  | | --- | | ***agent any***: Execute this Pipeline or any of its stages, on any available agent.  Defines the ***"Build" stage***: Perform some steps related to the "Build" stage.  Defines the ***"Test" stage***: Perform some steps related to the "Test" stage.  Defines the ***"Deploy" stage***: Perform some steps related to the "Deploy" stage. | |  |  | |  |  | |  |  | | Example:#  git “url…….git” -- to pull code  bat “mvn clean install” -- to execute |  | | Or  bat label: '', script: 'mvn clean install' |  | | bat label: '', script: 'clean install -Dcucumber.options="--tags @regression"' |  | |  |  | | ***node***: Execute this Pipeline or any of its stages, on any available agent.  Defines the ***"Build" stage***: Perform some steps related to the "Build" stage.  Defines the **"Test" stage**: Perform some steps related to the "Test" stage.  Defines the **"Deploy" stage**: Perform some steps related to the "Deploy" stage.  stage blocks are optional in Scripted Pipeline syntax.  However, implementing stage blocks in a Scripted Pipeline provides clearer visualization of each `stage’s subset of tasks/steps in the Jenkins UI. |

|  |
| --- |
| **Stages in Pipeline can be:**   * Cloning the code from SCM * Building the project * Running the Unit Test cases * Deploying the code * Other functional and performance tests. |

**Shell Script:** is a computer program designed to run by the ***Unix shell*** *(* a command-line interpreter that provides command line user interface), generally more used to mean the automated mode of running an operating system shell.

The typical operations performed by the shell scripts include file manipulation, program execution and printing text.

A script which setup environment, runs the program, does any necessary cleanup, logging, etc is called a Wrapper.

**Batch script:** in windows, the batch file is a file that stores commands in serial order. Command line interpreter takes the file as an input and executes in same order. A batch fileis simply a text file saved with the .bat file extension.

**Plugins:** (are packaged as self-contained .hpi files which have code, images and other resources).

Plugins are primary means of enhancing the functionality of a Jenkins environment to suit organization or user specific needs. Can be downloaded from Jenkins Update Centre.

There are bunch of different plugins which can be installed on Jenkins master and to integrate various build tools, cloud providers, analysis tools and much more.

**Installing a plugin:**

* Using the Plugin Manager in the web UI.

Manage Jenkins> Manage plugins

* Using the Jenkins CLI install-plugin command.

> java -jar jenkins-cli.jar -s http://localhost:8080/ install-plugin SOURCE ... [-deploy] [-name VAL] [-restart]

**CORN Expression:**

Jenkins uses the cron syntax to schedule various items within the tool.

The cron syntax is represented by five asterisks, with each one separated by a space. The first asterisk represents minutes, the second represents hours, the third the day of the month, the fourth the month itself and the fifth the day of the week. For example, to schedule a build job to pull from GitHub every Friday at 5:30 p.m., the syntax would be: 30 17 \* \* 4.

Q. Name a Jenkins environment variable you have used in a shell script or batch file.

There are a number of environment variables that are available by default in any Jenkins build job. A few commonly used ones include:

$JOB\_NAME

$NODE\_NAME

$WORKSPACE

$BUILD\_URL

$JOB\_URL

Note that, as new Jenkins plug-ins are configured, more environment variables become available. For example, when the Jenkins Git plug-in is configured, new Jenkins Git environment variables, such as $GIT\_COMMIT and $GIT\_URL, become available to be used in scripts.

Q. Name three security mechanisms Jenkins uses to authenticate users.

Jenkins can authenticate users in one of three ways:

Jenkins can use an internal database to store user data and credentials. (This is the default.)

Jenkins can be configured to authenticate against a Lightweight Directory Access Protocol server.

Jenkins can be configured to employ the authentication mechanism used by the application server upon which it is deployed.

Q. Describe the standard process to configure and use third-party tools within Jenkins?

The process to use a third-party tool, such as Artifactory, Node, SonarQube or Git typically follows a four-step process.

The third-party software must be installed.

A Jenkins plug-in that supports the third-party tool must be installed through the Jenkins admin console.

The third-party tool must be configured in the Tools tab of the Manage Jenkins section of the admin console.

Finally, the plug-in can be used from within a Jenkins build job. The plug-in will then facilitate communication between the Jenkins build job and the third-party tool.

Not every third-party tool is configured in exactly the same way. For example, Jenkins can be configured to install Maven itself, rather than requiring a pre-existing installation. Similarly, third-party tools, like Checkstyle or JaCoCo, can be downloaded at build time by Maven. So these four steps are not always adhered to strictly, but at a high level, these are the typical steps required to install and configure a third-party Jenkins tool.

Q. Name two ways a Jenkins node agent can be configured to communicate back with the Jenkins master.

The tool provides two mechanisms for starting a Jenkins node agent:

Launch a Jenkins node agent from a browser window.

Launch a Jenkins node agent from the command line.

When a Jenkins node agent is launched from a browser, a JNLP file is downloaded. When it runs, the JNLP file launches a new process on the client machine that runs Jenkins jobs.

To launch from the command line, the agent.jar file is required on the client. This executable JAR file is run from the command line, along with a reference to the slave agent's JNLP file that is hosted on the server. Like the JNLP file downloaded through a web browser, running this command launches a process on the client that can communicate with the Jenkins master and run Jenkins build jobs when it has idle clock cycles.

Q. How do you take a backup of your Jenkins build jobs in order to prepare for disaster recovery?

Each Jenkins build stores its configuration as XML in a subdirectory of the JENKINS\_HOME\jobs folder. By copying this folder to a secondary location, the configuration of all of the build jobs managed by the Jenkins master will be backed up as a result. Checking this folder into a source code management tool like Git isn't a bad idea either. Knowledge of ways to perform Jenkins Git integration is always looked upon fondly in a DevOps interview.

By simply copying the contents of this folder to a new Jenkins server instance, all of the build jobs described in this folder will be restored the next time the Jenkins server is started.

Q. Name three steps or stages a typical Jenkins pipeline might include.

A full-blown Jenkins pipeline will build a project from source code, put it through a variety of unit, integration, performance and user acceptance tests, and then, finally, if every test succeeds, deploy a packaged application to an application server, Nexus repository or Docker container. So, three fundamental stages would be:

Build

Test

Deploy

expand on these three stages and describe a more complete Jenkins pipeline as code example.

|  |
| --- |
| pipeline { |
|  | agent any |
|  | stages { |
|  | stage('One') { |
|  | steps { |
|  | echo 'This is stage one running ....................' |
|  | } |
|  | } |
|  | stage('Two') { |
|  | steps {  echo ‘ executing mvn clean install command’ |
|  | bat 'mvn clean install’ |
|  | } |
|  | } |
|  | stage('Three') { |
|  | steps { |
|  | echo 'This is stage three running .....................' |
|  | } |
|  | } |
|  | stage('Four') { |
|  | steps { |
|  | echo 'This is stage four running .....................' |
|  | } |
|  | } |
|  |  |
|  | stage('Input') { |
|  | steps { |
|  | input('Do you want to proceed......?') |
|  | } |
|  | } |
|  |  |
|  | stage('Five') { |
|  | steps { |
|  | echo 'This is stage five running .....................' |
|  | } |
|  | } |
|  |  |
|  |  |
|  | stage('Six') { |
|  | steps { |
|  | echo 'This is stage six running .....................' |
|  | } |
|  | } |
|  | } |
|  | } |