A **Jenkins pipeline** is an automated process for defining, managing, and executing tasks in Jenkins using **Groovy scripting**.

It provides a flexible way to describe the steps (or "stages") involved in your build, test, and deployment processes

**Components of a Jenkins Pipeline:**

1. **Stages**: Logical sections (like build, test, deploy).
2. **Steps**: Specific tasks in a stage (running a script, building an application).
3. **Agents**: Specify where to run the pipeline (e.g., on a specific Jenkins node).

node {

stage('Build')

{

echo 'Building the project...'

// Build commands here

}

stage('Test')

{

echo 'Running tests...'

// Test commands here

}

stage('Deploy')

{

echo 'Deploying application...'

// Deploy commands here

}

}

**Key Terminology in Cypress:**

1. **Cypress**: A JavaScript end-to-end testing framework used for testing web applications.
2. **Parallel Execution**: Running multiple test files or test suites concurrently to reduce test execution time.
3. **Reports**: Test results generated in a human-readable format, often in JSON, HTML, or XML format.
4. **Artifacts**: Files generated by the build (like test reports) that Jenkins can archive for later use.
5. **Caching**: Storing dependencies (like node\_modules) to speed up build times in CI/CD pipelines.

**Key Jenkins Terminology:**

1. **Pipeline**: A Jenkins pipeline is a suite of steps your Jenkins job will follow, defined using **Groovy scripts**. It automates the process of building, testing, and deploying code.
2. **Agent**: Defines where the pipeline or stage runs. You can specify a Jenkins node or use agent any to run on any available node.
3. **Stage**: A block that defines a distinct part of the pipeline, such as **Build**, **Test**, or **Deploy**. It's a logical group of steps.
4. **Steps**: A single task that Jenkins executes (e.g., running a shell command, building code).
5. **When**: Conditional logic that specifies whether a particular stage should run, based on branch names, conditions, etc.
6. **Parallel**: Allows stages or steps to run in parallel, reducing the time required to execute a pipeline.
7. **sh**: A step that runs shell commands on Unix-based systems.
8. **junit**: A step that processes test results, such as JUnit XML reports, for displaying them in Jenkins.

Ex. pipeline {

agent any

stages {

stage('Install Dependencies')

{

steps {

sh 'npm install'

}

}

stage('Run Cypress Tests')

{

steps {

sh 'npx cypress run'

}

}

}

post {

always {

archiveArtifacts artifacts: '\*\*/cypress/videos/\*\*/\*.mp4', allowEmptyArchive: true

archiveArtifacts artifacts: '\*\*/cypress/screenshots/\*\*/\*.png', allowEmptyArchive: true

}

}

}

**2. Running Cypress in Parallel**

This pipeline runs Cypress tests in parallel across multiple machines or nodes using the --parallel flag (in conjunction with Cypress Dashboard).

pipeline {

agent any

stages {

stage('Install Dependencies')

{

steps {

sh 'npm install'

}

}

stage('Run Cypress in Parallel')

{

steps {

sh 'npx cypress run --record --parallel --key <your\_cypress\_dashboard\_key>'

}

}

}

}

**3. Cypress with Test Reports (JUnit and Mochawesome)**

This pipeline generates **JUnit XML** and **Mochawesome HTML** reports and archives them in Jenkins.

pipeline {

agent any

stages {

stage('Install Dependencies') {

steps {

sh 'npm install'

}

}

stage('Run Cypress Tests with JUnit Reporting')

{

steps {

sh 'npx cypress run --reporter junit --reporter-options "mochaFile=reports/junit/results-[hash].xml"'

}

}

stage('Generate Mochawesome Report') {

steps {

sh 'npx mochawesome-merge reports/\*.json > mochawesome.json'

sh 'npx marge mochawesome.json --reportDir mochawesome-report'

}

}

}

post {

always {

junit 'reports/junit/\*.xml'

archiveArtifacts artifacts: 'mochawesome-report/\*.html', allowEmptyArchive: true

}

}

}

**Conditional Cypress Execution Based on Branch**

This pipeline runs Cypress tests only on specific branches, like master or develop.

pipeline {

agent any

stages {

stage('Install Dependencies') {

steps {

sh 'npm install'

}

}

stage('Run Cypress Tests') {

when {

branch 'master'

}

steps {

sh 'npx cypress run'

}

}

}

}

**Caching Dependencies (node\_modules) to Speed Up Pipeline**

This pipeline caches the node\_modules directory to avoid reinstalling dependencies on every run, improving performance.

pipeline {

agent any

options {

cache(path: 'node\_modules', key: 'npm-cache', fallbackToEmpty: true)

}

stages {

stage('Install Dependencies') {

steps {

sh 'npm ci'

}

}

stage('Run Cypress Tests') {

steps {

sh 'npx cypress run'

}

}

}

}

**npm ci**: A faster installation command that uses the exact versions specified in package-lock.json.

**6. /Running Cypress Tests in Parallel (Using GitHub Actions or Jenkins with Different Test Files)**

Run multiple Cypress test files in parallel by assigning specific test files to different agents or nodes.

pipeline {

agent any

stages {

stage('Install Dependencies') {

steps {

sh 'npm install'

}

}

stage('Run Tests in Parallel') {

parallel {

stage('Test Group 1') {

steps {

sh 'npx cypress run --spec "cypress/integration/tests/test1.spec.js,cypress/integration/tests/test2.spec.js"'

}

}

stage('Test Group 2') {

steps {

sh 'npx cypress run --spec "cypress/integration/tests/test3.spec.js,cypress/integration/tests/test4.spec.js"'

}

}

}

}

}

}

**parallel {}**: This directive allows Cypress tests to run in parallel across multiple Jenkins stages, reducing test execution time.

**--spec**: Specifies which test files to run. You can distribute tests across multiple nodes.

**Using Cypress with Environment Variables and Configurations**

This pipeline runs Cypress tests using environment-specific configurations (such as staging or production).

pipeline {

agent any

stages {

stage('Install Dependencies') {

steps {

sh 'npm install'

}

}

stage('Run Cypress Tests on Staging') {

steps {

sh 'npx cypress run --config-file cypress/config/staging.json'

}

}

stage('Run Cypress Tests on Production') {

steps {

sh 'npx cypress run --config-file cypress/config/production.json'

}

}

}

}

**--config-file**: This flag specifies different Cypress configuration files (like staging.json and production.json), allowing you to run environment-specific tests with different base URLs, timeouts, etc.

#staging.json

{

"baseUrl": "https://staging.example.com",

"env": {

"username": "staging\_user",

"password": "staging\_password",

"apiUrl": "https://api.staging.example.com"

},

"viewportWidth": 1280,

"viewportHeight": 720,

"defaultCommandTimeout": 10000,

"pageLoadTimeout": 30000,

"retries": {

"runMode": 2,

"openMode": 0

},

"video": true,

"screenshotOnRunFailure": true

}

 **Workflow** (GitHub Actions) = **Pipeline** (Jenkins)

 **Job** (GitHub Actions) = **Stage** (Jenkins)

 **Runner** (GitHub Actions) = **Agent/Node** (Jenkins)

 **Actions** (GitHub Actions) = **Plugins** (Jenkins)

 **Secrets** (GitHub Actions) = **Credentials** (Jenkins)

 **Logs** (Both) = Detailed output of each stepname: CI Pipeline

on: [push, pull\_request]

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Install dependencies

run: npm install

- name: Run Cypress tests

run: npm run cypress:run

pipeline {

agent any

stages {

stage('Checkout') {

steps {

checkout scm

}

}

stage('Install dependencies') {

steps {

sh 'npm install'

}

}

stage('Run Cypress tests') {

steps {

sh 'npm run cypress:run'

}

}

}

}