Jenkins 6

Declarative Jenkins Pipeline

Category of jobs/projects managed using Jenkins

- Software/applications [build, test and deploy]
- System monitoring
- Database backup
- Infrastructure creation/provisioning
- Infrastructure configuration management

Differences between Freestyle and Pipeline Projects

Aspect	**Freestyle Projects**	**Pipeline Projects**
Flexibility	Maximum flexibility in configuring build steps	Relatively rigid structure with defined stages
Customization	Define custom build steps and configurations	Define build workflows as code
Manual Configuration	Requires manual configuration of each aspect	Configuration defined in code/scripts
User Interface	Configuration primarily through UI	Configuration defined as code/scripts
Legacy Support	Often used in legacy systems	Embraced for modern CI/CD practices
Automation	Manual intervention may be required	Highly automated with defined pipelines
Integration with SCM	May require separate triggers for SCM integration	Native integration with SCM events
Containerization	Limited support for containerization	Supports containerization for build environments

CI/CD	Suitable for basic CI, limited CD capabilities	Ideal for implementing CI/CD pipelines
Scalability	May become complex to scale for large projects	Designed for scalability with complex workflows

Jenkins Declarative Pipeline Script

```
pipeline {
    agent any

    environment {
        PATH = '/usr/local/bin'
    }

    options {
        timeout(time: 1, unit: 'HOURS')
        disableConcurrentBuilds()
        retry(3)
    }

    stages {
        stage('Build') {
            steps {
                // Build steps go here
                echo 'Building...'
            }
        }
        stage('Test') {
            steps {
                // Test steps go here
                echo 'Testing...'
            }
        }
        stage('Deploy') {
            steps {
                // Deployment steps go here
                echo 'Deploying...'
            }
        }
    }

    post {
        success {
            // Actions to take on success
            echo 'Pipeline succeeded'
            mail to: 'team@example.com', subject: 'Pipeline succeeded'
        }
        failure {
            // Actions to take on failure
            echo 'Pipeline failed'
            mail to: 'team@example.com', subject: 'Pipeline failed'
            archiveArtifacts 'target/*.jar'
        }
    }
}
```

- **pipeline**: Defines the beginning of the pipeline block.
- **agent any**: Specifies that the pipeline can execute on any available agent (Jenkins executor).
- **stages**: Defines a block containing multiple stages of the pipeline.
- **stage**: Defines a single stage within the pipeline.
- **steps**: Contains the steps to be executed within the stage.
- **environment:** Sets the PATH environment variable.
- **options**: Specifies a timeout of 1 hour, disables concurrent builds, and defines a retry strategy with a maximum of 3 retries.
- **post:** Defines actions to be taken after the completion of all stages,
- **echo**: A simple step that prints a message to the console log.

Differences between Scripted pipeline and Declarative pipeline in Jenkins

Aspect	Scripted Pipeline	Declarative Pipeline
Syntax Complexity	More verbose and complex syntax	Simplified and concise syntax
Stage Definition	Stages and steps defined programmatically	Stages and steps defined declaratively
Flow Control	Offers complete flow control with loops, conditions, etc.	Limited flow control options
Error Handling	Manual error handling and exception handling	Automatic error handling and retries
Readability	Can be less readable, especially for complex pipelines	More readable and easier to understand syntax
Configuration	Requires explicit configuration for each step	Simplified configuration with predefined structure
Best Suited For	Experienced users comfortable with scripting	Beginners and users preferring simpler syntax
Adoption	Widely used, especially in older Jenkins setups	Increasingly adopted for new projects and pipelines
Official Support	Well-supported by Jenkins and community	Officially recommended by Jenkins

Example:

```
pipeline{
  agent any
  tools {
    maven "maven3.6.0"
  }
  stages {
    stage('1GetCode'){
      steps{
        sh "echo 'cloning the latest application version' "
        git branch: 'feature', credentialsId: 'gitHubCredentials', url:
https://github.com/fewaitconsulting/maven-web-app'
      }
    }
    stage('2Test+Build'){
      steps{
        sh "echo 'running JUnit-test-cases' "
        sh "echo 'testing must passed to create artifacts ' "
        sh "mvn clean package"
      }
    }
    stage('3CodeQuality'){
      steps{
        sh "echo 'Perfoming CodeQualityAnalysis' "
        sh "mvn sonar:sonar"
      }
    }
    stage('4uploadNexus'){
      steps{
        sh "mvn deploy"
      }
    }
    stage('5deploy2prod'){
      steps{
        deploy adapters: [tomcat8(credentialsId: 'tomcat-credentials', path: '',
url: 'http://35.170.249.131:8080/')], contextPath: null, war: 'target/*war'
      }
    }
}
  post{
    always{
      emailext body: '''Hey guys
Please check build status.

Thanks
TeamCameroon
+237650661631''', recipientProviders: [buildUser(), developers()], subject:
'success', to: 'paypal-team@gmail.com'
    }
    success{
      emailext body: '''Hey guys
Good job build and deployment is successful.

Thanks
FewaItConsulting
+237650661631''', recipientProviders: [buildUser(), developers()], subject:
'success', to: 'paypal-team@gmail.com'
    }
```

```
    failure{
      emailext body: '''Hey guys
Build failed. Please resolve issues.

Thanks
FewaItConsulting
+237650661631''', recipientProviders: [buildUser(), developers()], subject:
'success', to: 'paypal-team@gmail.com'
    }
  }
```

Multi-branch Project

A Jenkins Multi-Branch Pipeline project is a Jenkins job type that automatically creates individual Jenkins Pipeline jobs for each branch in a repository.

It's commonly used for projects hosted in version control systems like Git, where each branch may represent a feature, bug fix, or other development effort.

Overview:

- **Automatic Branch Discovery**: Jenkins automatically discovers branches in a repository and creates a separate pipeline job for each branch.
- **Branch Source Configuration**: You specify the repository and branch sources to be monitored. Jenkins supports Git, Mercurial, and other version control systems.
- **Pipeline Script Configuration**: Each pipeline job's configuration is based on a Jenkinsfile found in the root directory of each branch. The Jenkinsfile defines the stages and steps for the pipeline.

Key Features:

1. **Branch Management**:
    - Jenkins automatically manages pipeline jobs for branches, including creation, deletion, and updating.
2. **Branch Indexing**:
    - Jenkins periodically scans the repository for changes and updates the pipeline jobs accordingly.

3. **Automatic Pipeline Creation**:
    - When a new branch is created in the repository, Jenkins automatically creates a corresponding pipeline job.
4. **Branch Filtering**:
    - You can specify inclusion and exclusion patterns to control which branches trigger pipeline jobs.

Usage:

1. **Setup**:
    - Create a new Multi-Branch Pipeline job in Jenkins.
    - Configure the branch sources (e.g., repository URL, credentials).
    - Define any additional settings, such as build triggers or build options.
2. **Pipeline Configuration**:
    - Include a Jenkinsfile in the root directory of each branch.
    - The Jenkinsfile defines the stages, steps, and other pipeline configurations.
3. **Branch Management**:
    - Jenkins manages pipeline jobs for each branch automatically.
    - Pipeline jobs are created, updated, or removed based on changes in the repository.

Benefits:

1. **Automation**:
    - Eliminates the need to manually create and manage pipeline jobs for each branch.
    - Automates the build and test process for all branches in the repository.
2. **Consistency**:

- Ensures consistency in build and test configurations across different branches.

3. **Scalability**:
    - Scales efficiently for projects with many branches, reducing manual overhead.
4. **Integration**:
    - Integrates seamlessly with version control systems like Git, allowing continuous integration across multiple branches.