Multi-scenario Jenkins pipeline breakdown

What is a Declarative Pipeline?

- A **Declarative Pipeline** is a simplified, opinionated syntax to define Jenkins jobs using a Jenkinsfile.
- Introduced to make pipeline-as-code **more readable, maintainable, and structured**.
- It uses a pipeline {} block and clearly defined stages.

Basic Structure of a Declarative Pipeline

```
pipeline {
  agent any

  environment {
    VAR_NAME = "value"
  }

  options {
    timeout(time: 10, unit: 'MINUTES')
  }

  triggers {
    pollSCM('* * * * *')
  }

  tools {
    maven 'Maven 3'
  }

  stages {
    stage('Build') {
      steps {
        sh 'mvn clean package'
      }
    }
    stage('Test') {
      steps {
        sh 'mvn test'
      }
    }
  }

  post {
    success {
      echo 'Build completed successfully!'
    }
    failure {
      echo 'Build failed!'
    }
```

```
    }
}
```

Jenkins Declarative Pipeline Sections & Directives

Section	Directive	Purpose	Example
pipeline	Main wrapper	Root block to define everything in a Declarative pipeline	pipeline { ... }
agent	any, none, label	Where the pipeline or stage runs (Jenkins agent/label)	agent any / agent { label 'ubuntu' }
environment	Define environment vars	Set key-value pairs as env vars in the pipeline	environment { ENV = "prod" }
tools	JDK, Maven, etc.	Automatically install and use tools configured in Jenkins	tools { maven 'Maven 3' }
options	Timeout, retry, skip	Configure behavior like timeouts or skipping default checkout	timeout(time: 15, unit: 'MINUTES')
triggers	Poll SCM, cron	Automatically trigger builds via schedule or source changes	pollSCM('@daily')
parameters	Build inputs	Accept input at build time (strings, booleans, choices)	string(name: 'ENV', defaultValue: 'dev')
stages	stage blocks	Define the major units of work (build, test, deploy, etc.)	stage('Build') { steps { ... } }
steps	Actual commands	Execute commands/scripts inside a stage	sh 'npm install', echo 'Hello'
script	Scripted block	Use Groovy inside declarative for complex logic	script { if (a == b) { ... } }
post	Success, failure, etc.	Run cleanup or notifications after the pipeline or stage completes	post { success { echo "Deployed!" } }
when	Conditional logic	Run stage conditionally (e.g., on branch, env, tag)	when { branch 'main' }
input	Wait for approval	Pause pipeline and wait for manual input or approval	input { message 'Deploy?' }
parallel	Run stages in parallel	Run multiple steps or stages at the same time	parallel { stageA { ... } stageB { ... } }

Common Use Cases

Directive	Use Case
agent	Define where (on which node or container) the pipeline runs
environment	Set deployment env (dev, qa, prod)
stages	Separate Build, Test, Deploy clearly

Directive	Use Case
post	Notify Slack/email, clean up, archive artifacts
tools	Use Jenkins-managed Maven/Java version
parameters	Let user choose branch or tag at runtime
when	Skip or include stages conditionally
input	Ask QA or manager to approve deploy
parallel	Run frontend and backend tests together

Example: Basic Declarative CI Pipeline for Java Maven App

```
pipeline {
  agent any

  tools {
    maven 'Maven 3'
  }

  stages {
    stage('Checkout') {
      steps {
        git 'https://github.com/mgsgoms/vanakkam-world.git'
      }
    }

    stage('Build') {
      steps {
        sh 'mvn clean package'
      }
    }

    stage('Test') {
      steps {
        sh 'mvn test'
      }
    }
  }

  post {
    success {
      echo 'Build and test completed successfully!'
    }
    failure {
      echo 'Pipeline failed. Check logs.'
    }
  }
}
```

👉 https://github.com/mgsgoms/vanakkam-world (Java/Maven-based app).

Pre-requisites

Ensure Jenkins has:

- Git plugin
- Maven integration (Global Tool Config)
- Docker CLI installed on Jenkins agent
- DockerHub credentials configured (dockerhub-cred)
- Kubeconfig credentials (kubeconfig-secret)

Scenario 1: Jenkins + GitHub (Clone Only)

```
pipeline {
  agent any

  stages {
    stage('Clone GitHub Repo') {
      steps {
        git url: 'https://github.com/mgsgoms/vanakkam-world.git'
        echo 'Code pulled successfully.'
      }
    }
  }
}
```

Scenario 2: Jenkins + GitHub + Maven (Build WAR)

```
pipeline {
  agent any

  tools {
    maven 'Maven 3'  // Define this in Jenkins → Global Tool Config
  }

  stages {
    stage('Clone Repo') {
      steps {
        git url: 'https://github.com/mgsgoms/vanakkam-world.git'
      }
    }

    stage('Build WAR with Maven') {
      steps {
        sh 'mvn clean package'
      }
    }

    stage('Archive WAR') {
      steps {
        archiveArtifacts artifacts: '**/target/*.war', fingerprint: true
      }
    }
  }
}
```

Scenario 3: Jenkins + Docker (Build Docker Image)

```
pipeline {
  agent any

  environment {
    DOCKER_IMAGE = 'yourdockerhubuser/vanakkam-world'
  }

  stages {
    stage('Clone Repo') {
      steps {
        git url: 'https://github.com/mgsgoms/vanakkam-world.git'
      }
    }

    stage('Build Docker Image') {
      steps {
        script {
          sh 'docker build -t $DOCKER_IMAGE .'
        }
      }
    }

    stage('Run Locally') {
      steps {
        sh 'docker run -d -p 8080:8080 $DOCKER_IMAGE'
      }
    }
  }
}
```

Scenario 4: Jenkins + Kubernetes (Deploy to K8s)

```
pipeline {
  agent any

  environment {
    APP_NAME = 'vanakkam-world'
    K8S_NAMESPACE = 'default'
  }

  stages {
    stage('Deploy to Kubernetes') {
      steps {
        withCredentials([file(credentialsId: 'kubeconfig-secret', variable:
'KUBECONFIG')]) {
          sh '''
            export KUBECONFIG=$KUBECONFIG
            kubectl apply -f k8s/deployment.yaml -n $K8S_NAMESPACE
            kubectl apply -f k8s/service.yaml -n $K8S_NAMESPACE
          '''
        }
      }
    }
  }
}
```

You'll need:

- A working k8s/deployment.yaml and service.yaml in your repo.
- kubeconfig-secret saved in Jenkins (secret file type).

Scenario 5: Full CI/CD

GitHub → Jenkins → Maven → Docker → DockerHub → Kubernetes

```
pipeline {
  agent any

  tools {
    maven 'Maven 3'
  }

  environment {
    DOCKER_IMAGE = 'cubensquare/vanakkam-world'
    DOCKER_TAG = 'latest'
  }

  stages {
    stage('Clone Repo') {
      steps {
        git url: 'https://github.com/mgsgoms/vanakkam-world.git'
      }
    }

    stage('Maven Build') {
      steps {
        sh 'mvn clean package'
      }
    }

    stage('Docker Build & Push') {
      steps {
        script {
          docker.withRegistry('', 'dockerhub-cred') {
            def image = docker.build("$DOCKER_IMAGE:$DOCKER_TAG")
            image.push()
          }
        }
      }
    }

    stage('Deploy to Kubernetes') {
      steps {
        withCredentials([file(credentialsId: 'kubeconfig-secret', variable:
'KUBECONFIG')]) {
          sh '''
            export KUBECONFIG=$KUBECONFIG
            kubectl apply -f k8s/deployment.yaml
            kubectl apply -f k8s/service.yaml
          '''
        }
      }
    }
  }
}
```