✅ What is a Declarative Pipeline (Simple Meaning)

👉 A **Declarative Pipeline** is just a **step-by-step instruction file** that tells Jenkins:

“First do this… then do this… if success do next…”

Instead of clicking many buttons in Jenkins UI, you write everything in **one structured script**.

Think of it like:

```
Recipe for Jenkins.
```

🧠 In Your Project — What YOU are Doing

Right now your freestyle job does steps manually.

Pipeline job = same work but automated in order:

```
Stage 1 → Download code
Stage 2 → Check code quality (Sonar)
Stage 3 → Build WAR (Maven)
Stage 4 → Store artifact (Nexus)
Stage 5 → Deploy to Tomcat
Stage 6 → Send Slack message
```

You are basically writing:

👉 “My whole DevOps flow as code.”

☑️ Why Companies Use Declarative Pipeline

Very simple reasons:

✓ Everything saved as code

✓ Easy to repeat builds

✓ Easy debugging

✓ Version control (Git can track pipeline)

✓ No manual clicking every time

🧱 Structure of Declarative Pipeline (Super Easy View)

Every pipeline always looks like this:

```
pipeline {
   agent
   stages {
      stage('Step Name') {
         steps {
            commands
         }
      }
   }
}
```

Meaning:

Word	Simple Meaning
pipeline	Start of Jenkins automation
agent	Where to run (your Jenkins container)
stages	Big sections of work
stage	One step (Git, Maven, Deploy)
steps	Actual commands

🎯 Real-Life Analogy

Imagine you're cooking biryani:

```
pipeline = full recipe
stage = wash rice
```

```
stage = cook masala
stage = mix rice
stage = serve
```

You are just telling Jenkins the cooking order 😄 .

⚠ Important For YOUR Setup

Since Jenkins runs inside Docker:

👉 Pipeline commands run **inside Jenkins container**, not EC2 host.

That's why:

```
docker cp ...
```

still works — Jenkins talks to host docker using docker.sock.

🧪 What Will Change For You?

Before:

```
Freestyle Job
→ clicking UI
→ many configs
```

Now:

```
One Pipeline Script
→ press Build
→ everything runs automatically
```

🧩 Difference: Declarative vs Scripted (Don’t worry too much)

Declarative = structured, beginner friendly, safe

Scripted = full coding, advanced

You are using:

`Declarative = BEST for interviews + real projects`

💡 One Line Summary (VERY IMPORTANT)

👉 Declarative Pipeline = **CI/CD workflow written as structured code instead of manual Jenkins configuration.**

👉 **Interview one-liner:**

“Declarative Pipeline is Jenkins pipeline-as-code where CI/CD stages like build, scan, deploy, and notify are defined in a structured Jenkinsfile.”

✅ Basic Example Pipeline (Practice Version)

```
pipeline {
  agent any

  stages {

    stage('Say Hello') {
      steps {
        sh 'echo Hello'
      }
    }
```

```
    stage('Create File') {
      steps {
        sh 'touch test.txt'
      }
    }

  }
}
```

🧠 What Each Line Means (SUPER SIMPLE)

1️⃣ `pipeline {}`

👉 Start of Jenkins automation.

You are telling Jenkins:

`"Everything inside this is my CI/CD flow"`

2️⃣ `agent any`

👉 Where to run the job.

In your case:

`Runs inside Jenkins container`

You don’t need to change this.

3️⃣ `stages {}`

👉 Big sections of work.

Think:

```
Main chapters of a book.
```

Your real pipeline chapters will be:

```
Git Clone
SonarQube
Maven
Nexus
Deploy
Slack
```

4️⃣ stage('Name')

Example:

```
stage('Say Hello')
```

👉 Just a label in Jenkins UI.

You will literally see boxes like:

```
[Git Clone] → [Sonar] → [Maven]
```

5️⃣ steps {}

👉 Actual commands Jenkins runs.

Example:

```
sh 'echo Hello'
```

Meaning:

```
Run Linux command inside Jenkins container
```

Since your Jenkins runs in Docker:

```
commands execute INSIDE container
```

🔥 Now Map This to YOUR Real Pipeline

Let’s convert one real stage.

☑ Git Clone Stage

```
stage('Git Clone') {
  steps {
    git branch: 'feature-1.1', url: 'repo-url'
  }
}
```

What happens:

👉 Jenkins downloads code into:

```
/var/jenkins_home/workspace/JOB_NAME/
```

Inside container.

☑ SonarQube Stage

```
withSonarQubeEnv('SonarQube') {
   sh 'sonar-scanner ...'
}
```

Meaning:

👉 Jenkins connects to Sonar EC2

👉 Sends code for quality check

Simple idea:

```
Send code → Sonar analyzes → return report
```

☑ Maven Stage

```
sh 'mvn clean install'
```

Meaning:

👉 Compile Java

👉 Create WAR file

WAR location:

```
target/hiring-0.1.war
```

inside workspace.

☑ Nexus Stage

```
nexusArtifactUploader(...)
```

Meaning:

👉 Upload WAR to Nexus repository.

Simple thinking:

```
Build result → store in artifact storage
```

☑ Deploy Stage

```
docker cp target/hiring-0.1.war tomcat-container:/usr/local/tomcat/webapps/
```

Meaning:

👉 Copy WAR from Jenkins container

👉 Into Tomcat container.

Because docker.sock is mounted:

```
jenkins → host docker → tomcat
```

☑ Slack Notification

```
post {
 success { slackSend ... }
}
```

Meaning:

👉 After pipeline finishes, send message.

🎯 How You Can Practice (VERY EASY)

Create a dummy pipeline job and try this:

```
pipeline {
 agent any
 stages {

   stage('Step1') {
     steps {
       sh 'pwd'
     }
```

```
    }

    stage('Step2') {
      steps {
        sh 'ls -la'
      }
    }

  }
}
```

Run it and watch console output.

You'll instantly understand pipelines.

💡 Mental Model (Remember This)

Freestyle Job:

```
Click → Configure → Click → Configure
```

Declarative Pipeline:

```
Write steps once → Jenkins executes automatically
```

You are basically turning Jenkins into a **robot following written instructions**.

👉 **Interview one-liner:**

"Declarative pipeline defines CI/CD stages as structured blocks where each stage runs shell commands or plugins sequentially inside the Jenkins execution agent."