1️⃣ Snippet Generator

- Found in **Jenkins → Pipeline Syntax**.
- Helps generate **Pipeline steps** (mostly for **Scripted Pipeline** but works for Declarative too).
- Useful for complex steps like `archiveArtifacts`, `junit`, `bat`, etc.
- Interactive UI where you select a step and get the corresponding Groovy code.

Example (Using Snippet Generator)

If you select `archiveArtifacts`, it generate: it will archive.

```
archiveArtifacts artifacts: 'output/*.jar', onlyIfSuccessful: true
```

How to Use Snippet Generator?

1. Go to **Jenkins Dashboard**.
2. Open any **Pipeline job**.
3. Click **Pipeline Syntax** (on the left menu).
4. Select the **step** you want.
5. Fill in the parameters.
6. Click **"Generate Pipeline Script"** → Copy & paste the code.

2️⃣ Declarative Directive Generator

- Found in **Jenkins → Pipeline Syntax**.
- Specifically for **Declarative Pipelines**.
- Helps generate **pipeline directives** (like `options`, `triggers`, `parameters`).
- Useful when configuring **agent, tools, environment variables, post actions, etc.**

Example (Using Declarative Directive Generator)

If you select `options`, it will generate:

```
pipeline {

    agent any

    options {

        timeout(time: 10, unit: 'MINUTES')
```

```
    }
}
```

How to Use Declarative Directive Generator?

1. Go to **Jenkins Dashboard**.
2. Open any **Pipeline job**.
3. Click **Pipeline Syntax**.
4. Scroll down and click **Declarative Directive Generator**.
5. Select the **directive** (e.g., `options`, `triggers`).
6. Fill in the parameters.
7. Click **"Generate Declarative Directive"** → Copy & paste the code.

Feature	Snippet Generator	Declarative Directive Generator
Works for	Scripted & Declarative	Only Declarative
Generates	Pipeline steps (e.g., sh, bat, archiveArtifacts)	Directives (e.g., agent, options, environment)
Best for	Complex build steps (e.g., running scripts, tests, archives)	Pipeline structure & configuration
Example Output	sh 'mvn clean package'	options { timeout(time: 10, unit: 'MINUTES') }

What Are Directives in Jenkins?

In **Jenkins Declarative Pipelines**, **directives** are special blocks that define how the pipeline behaves. They control **agent selection, environment variables, triggers, stages, options, and more**.

Directive	Purpose
pipeline	Defines the entire pipeline
agent	Specifies where to run the pipeline

stages	Groups multiple stage blocks
stage	Defines a phase (e.g., Build, Test)
steps	Commands to run inside a stage
environment	Sets environment variables
post	Defines actions after success/failure
options	Configures pipeline settings (timeouts, logs)
parameters	Adds user input options
triggers	Automates pipeline execution

What is `build` in Jenkins?

In **Jenkins**, `build` refers to the process of executing a **job or pipeline** to produce an output (e.g., compiled code, test results, or deployment).

You can **trigger a build** in multiple ways:

- **Manually** (clicking "Build Now")
- **Automatically** (via Webhooks, SCM polling, or scheduled triggers)
- **From another pipeline/job** using the `build` step

1️⃣ Triggering a Build from Another Pipeline

You can start another **Jenkins job** from a pipeline using the `build` step.

Example: Trigger Another Job

```
pipeline {

    agent any

    stages {

        stage('Trigger Another Build') {

            steps {

                build job: 'Other_Job_Name'

            }
```

```
        }
    }
}
```

👉 This triggers the job named **"Other_Job_Name"**.

2️⃣ Passing Parameters to Another Build

If the target job expects parameters, you can pass them like this:

groovy

```
pipeline {
    agent any
    stages {
        stage('Trigger with Parameters') {
            steps {
                build job: 'Other_Job_Name', parameters: [
                    string(name: 'BRANCH', value: 'develop'),
                    booleanParam(name: 'DEPLOY', value: true)
                ]
            }
        }
    }
}
```

3️⃣ Checking Build Status & Handling Failures

By default, if the triggered build fails, the current pipeline also fails.
To handle this, use **`propagate: false`**.

groovy

```
pipeline {

    agent any

    stages {

        stage('Trigger with Error Handling') {

            steps {

                script {

                    def result = build job: 'Other_Job_Name',
propagate: false

                    echo "Triggered build result: ${result.result}"

                }

            }

        }

    }

}
```

👉 The variable **`result.result`** stores the status (`SUCCESS`, `FAILURE`, etc.).

4️⃣ Building a Project in a Stage

A **build stage** typically includes:

- **Fetching code** from Git
- **Compiling** the project
- **Running tests**

Example: Build a Java Project

groovy

```
pipeline {

    agent any

    stages {

        stage('Checkout') {

            steps {

                git branch: 'main', url:
'https://github.com/example/repo.git'

            }

        }


        stage('Build') {

            steps {

                bat 'clean install'

            }

        }

                stage('Test') {
```

```
            steps {
                sh './mvn test'
            }
        }
    }
}
```