SIGMA ACADEMY CI COURSE DAY 3

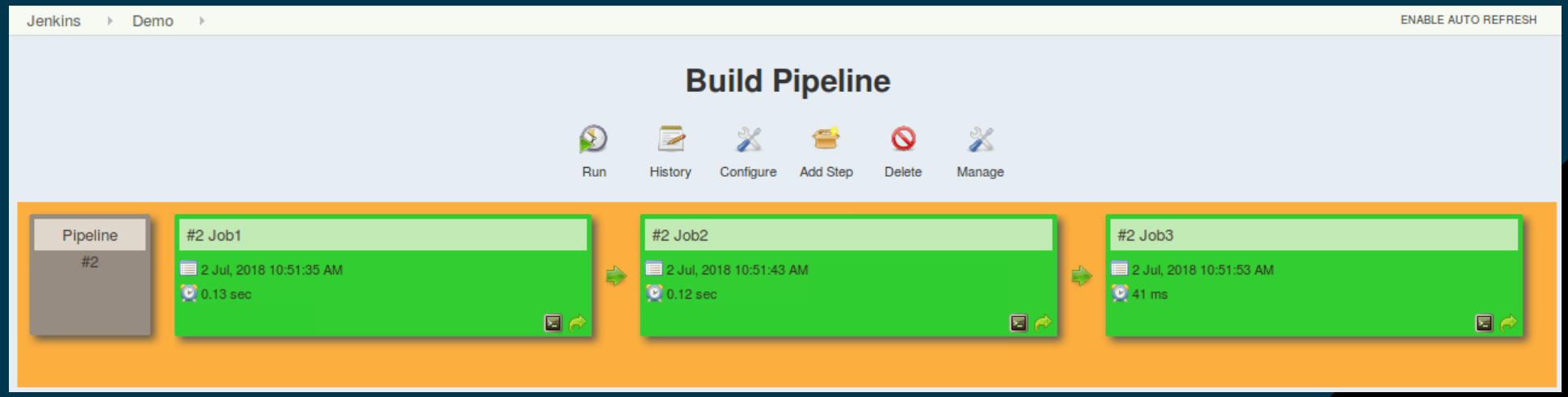
BY-
MANASA, NIKOLAI &
DAVID (SIGMA R&D)

SIGMA

RECAP FROM LAST SESSION

- DEVOPS LIFECYCLE AND PRACTICES
- INTRO TO CI/CD – WHAT & WHY?
- INTRO TO JENKINS - WHAT & WHY?
- LIVE JENKINS DEMO
- CI/CD PIPELINES USING JENKINS

JENKINS PIPELINES

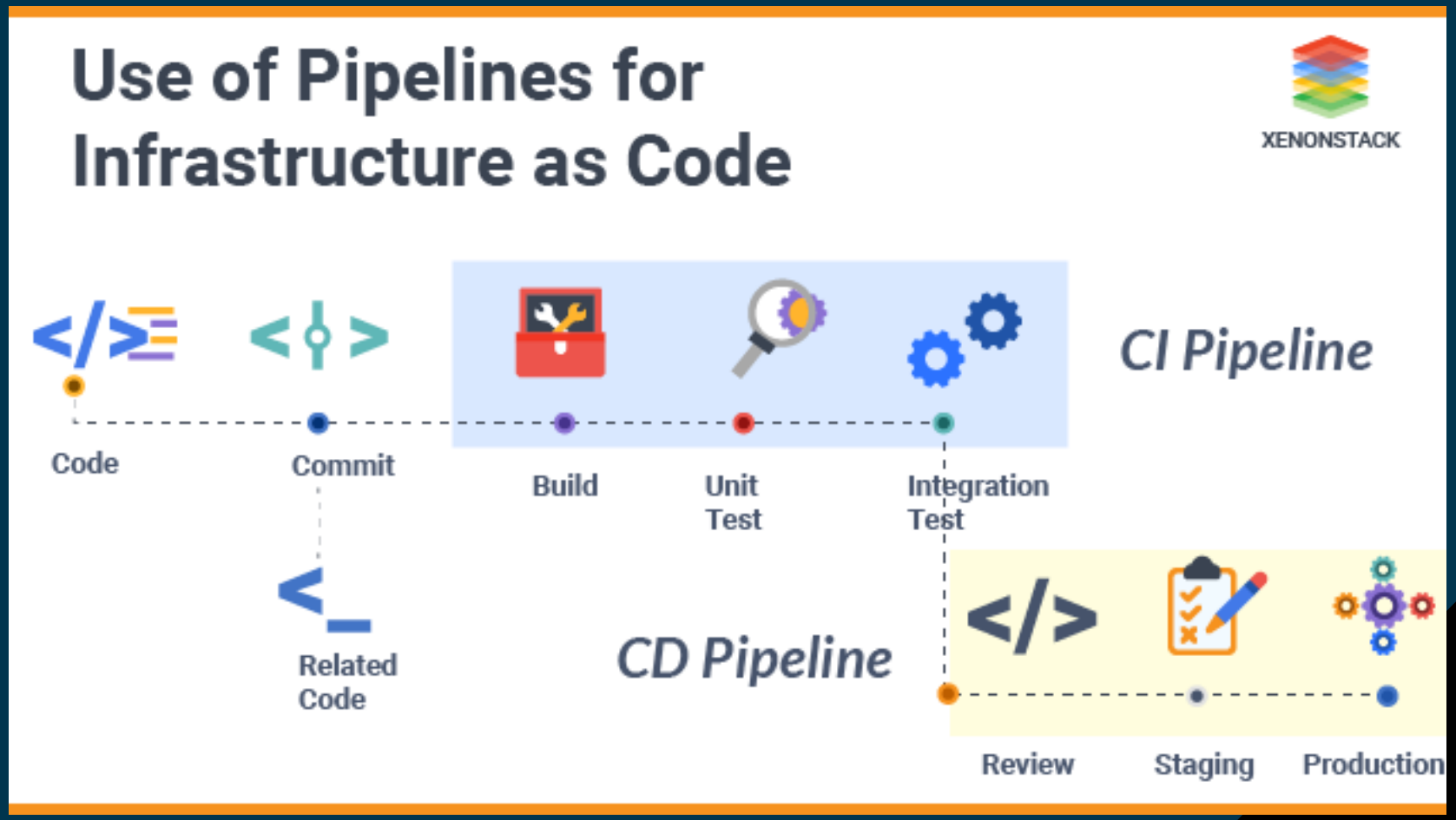


- Pipelines leverage the power of multiple steps to execute both simple and complex tasks according to parameters that you establish.

- Once created, pipelines can build code and orchestrate the work required to drive applications from commit to delivery.

SIGMA

WHAT IS INFRASTRUCTURE AS CODE / PIPELINE AS CODE?

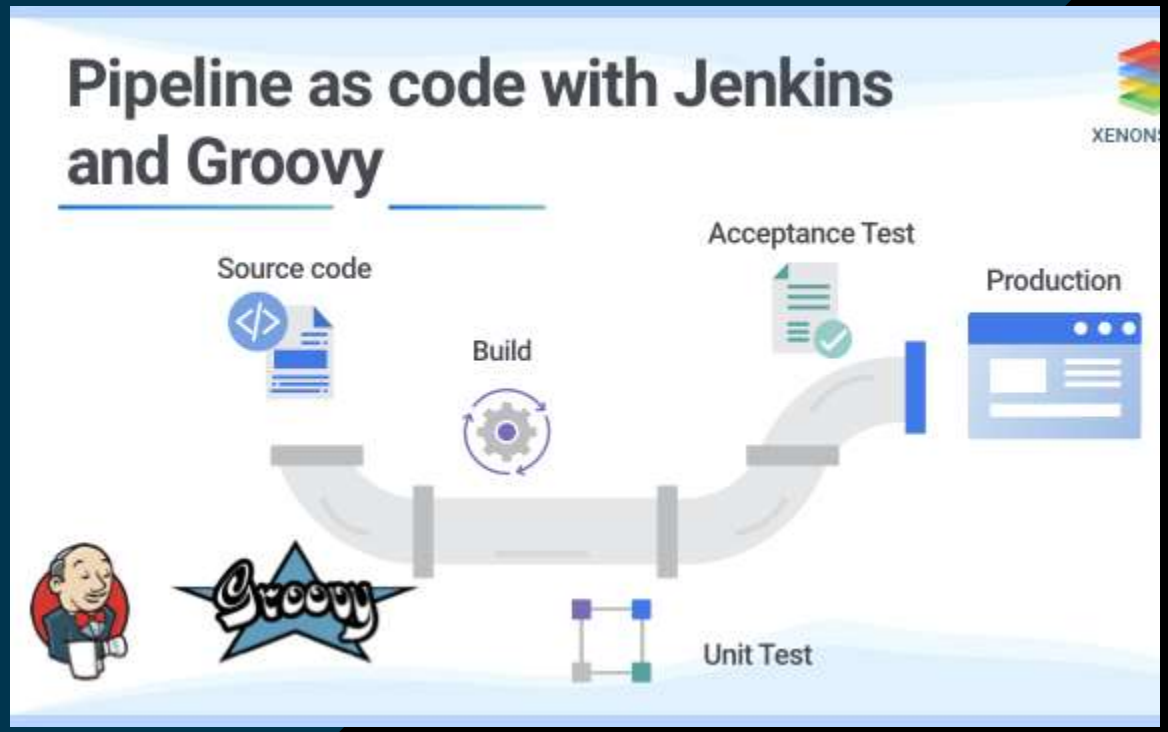


SIGMA

PIPELINES AS CODE

- It means that all the standard jobs defined by Jenkins are manually written as one whole script and they can be stored in a version control system

- It basically follows the '**pipeline as code**' discipline. Instead of building several jobs for each phase, you can now code the entire workflow and put it in a **Jenkinsfile**



SIGMA

ADVANTAGES OF PIPELINE AS CODE

It models simple to complex pipelines as code by using **Groovy DSL** (Domain Specific Language)

The code is stored in a text file called the Jenkinsfile which can be **checked into a SCM** (Source Code Management)

Improves user interface by incorporating **user input** within the pipeline

It is durable in terms of unplanned restart of the Jenkins master

It can restart from saved **checkpoints**

It supports complex pipelines by incorporating conditional loops, fork or join operations and allowing tasks to be performed in parallel

It can integrate with several other plugins

SIGMA

WHAT IS A JENKINSFILE?

```
1  node('test'){
2  stage('stage1') {
3  sh '''echo  stage1 steps'''
4  }
5  stage('stage2') {
6  sh '''echo stage2 steps'''
7  }
8  stage('stage3') {
9  sh '''echo stage3 steps'''
10 }
11 }
```

- A ***Jenkinsfile*** is a text file that stores the entire workflow as code and it can be checked into a SCM on your local system
- This enables the developers to **access, edit and check the code at all times**
- The Jenkinsfile is written using the Groovy DSL and it can be created through a text/groovy editor or through the configuration page on the Jenkins instance.

SIGMA

INTRODUCTION TO GROOVY

- Groovy is an object-oriented programming language used for JVM platform. This dynamic language has a lot of features drawing inspiration from Python, Smalltalk & Ruby.

- It can be used to orchestrate your pipeline in Jenkins and it can glue different languages together meaning that teams in your project can be contributing in different languages

- Groovy can seamlessly interface with the Java language and the syntax of Java and Groovy is very similar

SIGMA

JENKINSFILE GROOVY BASICS

- Variables in a Jenkinsfile can be defined by using the **def** keyword
- Jenkins Environment Variable is a global variable exposed through the **env** variable and used anywhere in the Jenkinsfile

Using environment variables

Jenkins Pipeline exposes environment variables via the global variable `env`, which is available from anywhere within a `Jenkinsfile`. The full list of environment variables accessible from within Jenkins Pipeline is documented at ${YOUR_JENKINS_URL}/pipeline-syntax/globals#env and includes:

BUILD_ID
The current build ID, identical to BUILD_NUMBER for builds created in Jenkins versions 1.597+

BUILD_NUMBER
The current build number, such as "153"

BUILD_TAG
String of jenkins-${JOB_NAME}-${BUILD_NUMBER}. Convenient to put into a resource file, a jar file, etc for easier identification

BUILD_URL
The URL where the results of this build can be found (for example http://buildserver/jenkins/job/MyJobName/17/)

EXECUTOR_NUMBER
The unique number that identifies the current executor (among executors of the same machine) performing this build. This is the number you see in the "build executor status", except that the number starts from 0, not 1

JAVA_HOME
If your job is configured to use a specific JDK, this variable is set to the JAVA_HOME of the specified JDK. When this variable is set, PATH is also updated to include the bin subdirectory of JAVA_HOME

JENKINS_URL
Full URL of Jenkins, such as https://example.com:port/jenkins/ (NOTE: only available if Jenkins URL set in "System Configuration")

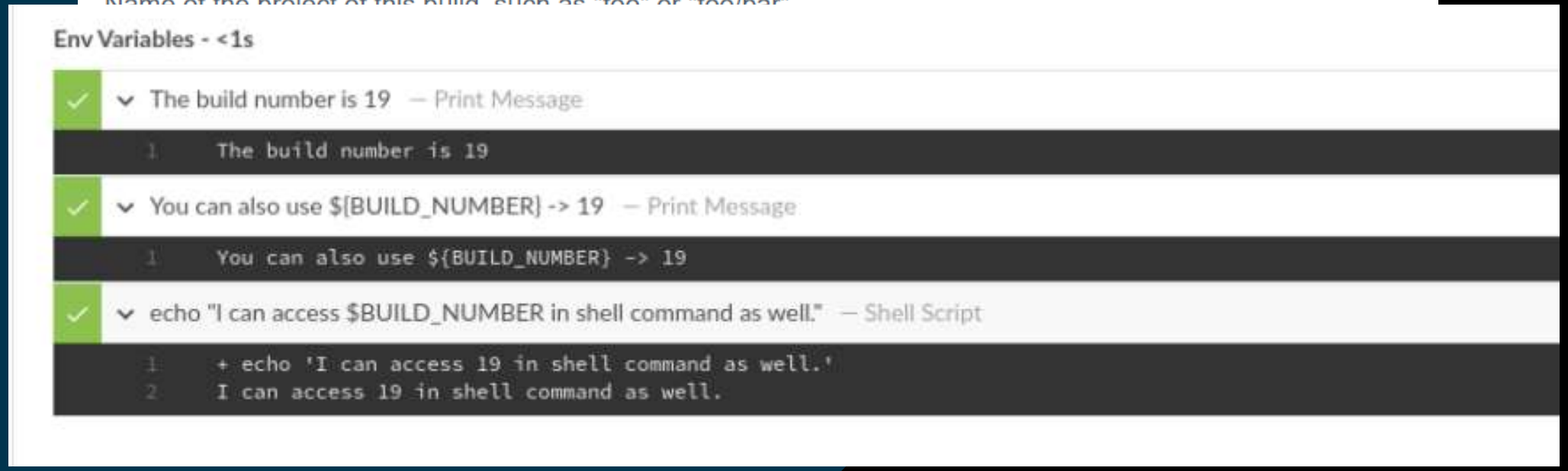
SIGMA

JENKINS ENVIRONMENT VARIABLES

```
echo "The build number is ${env.BUILD_NUMBER}"
echo "You can also use \${BUILD_NUMBER} -> ${BUILD_NUMBER}"
sh 'echo "I can access $BUILD_NUMBER in shell command as well."'
```

JOB_NAME

Name of the project of this build, such as "foo" or "foo/bar"



SIGMA

DEFINING A JENKINS PIPELINE

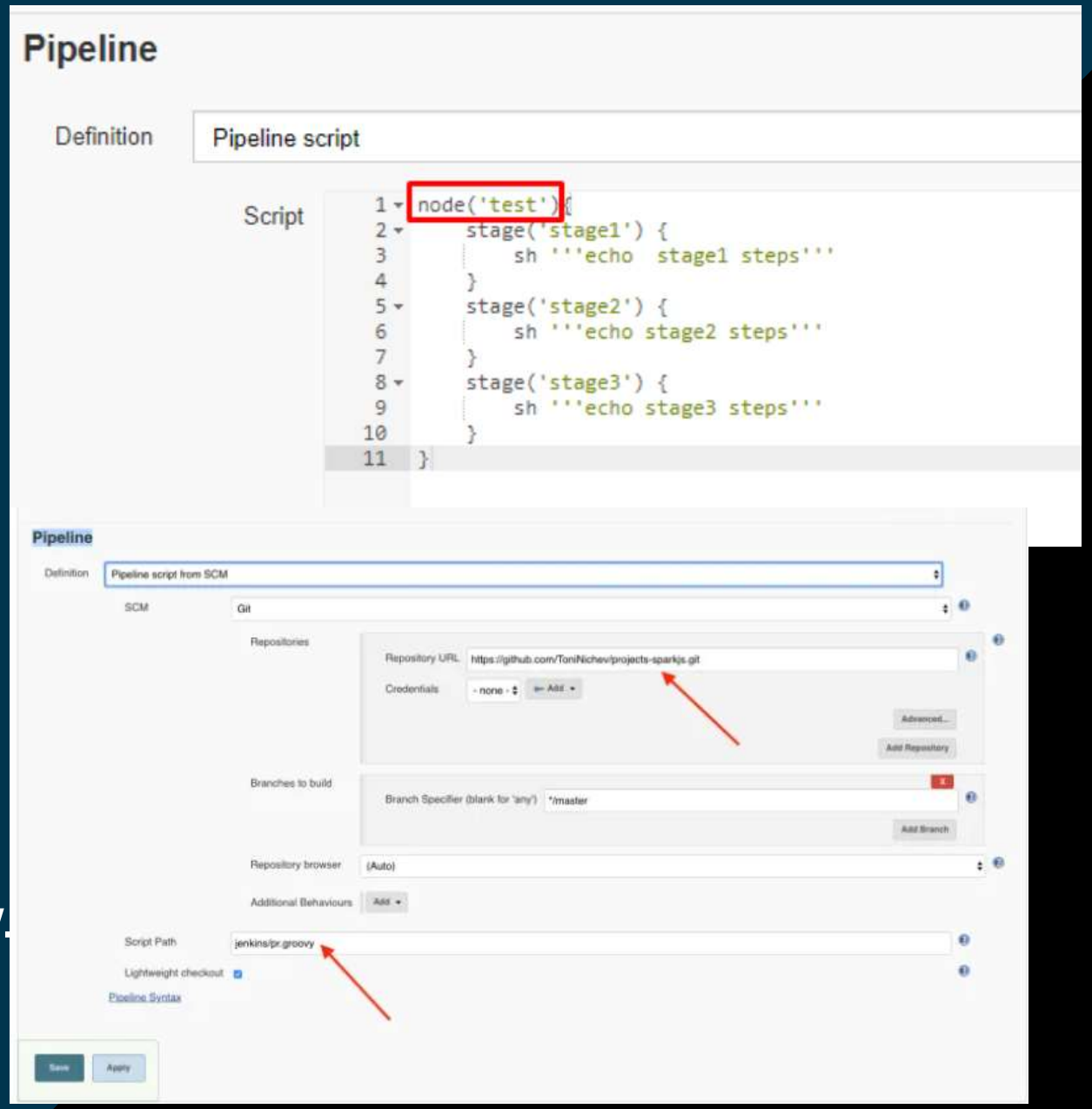
Jenkins Pipeline can be created in the following ways:

- ***Through the classic UI*** -

You can directly enter the basic Pipeline into Jenkins using the classic UI

- ***In SCM*** -

Jenkinsfile can be written manually and submitted to the project's source control repository.



SIGMA

DEFINING A JENKINS PIPELINE

Instead of relying on creating freestyle jobs and configuring it, Jenkins pipeline has a set of instructions for the Jenkins job to execute.

Jenkinsfile created by the classic UI is saved directly by Jenkins.

Pipeline created from the Jenkins classic UI is saved in the root directory of Jenkins and script is executed in Jenkins script console.

The Jenkins file is a base code for Jenkins which executes it as a Groovy script in Jenkins script console.

SIGMA

TYPES OF JENKINSFILE

DECLARATIVE PIPELINE SYNTAX

```
// Jenkinsfile (Declarative Pipeline)
pipeline {
  agent any
  stages {
    stage('Stage 1') {
      steps {
        echo 'Hello world!'
      }
    }
  }
}
```

SCRIPTED PIPELINE SYNTAX

```
// Jenkinsfile (Scripted Pipeline)
node { // node/agent
  stage('Stage 1') {
    echo 'Hello World' // echo Hello World
  }
}
```

SIGMA

DECLARATIVE VS SCRIPTED JENKINSFILE SYNTAX

- The scripted pipeline is a traditional way of writing the Jenkins pipeline as code

- Unlike Declarative pipeline, the scripted pipeline strictly uses groovy based syntax. Because of this, the scripted pipeline provides huge control over the script and can manipulate the flow of script extensively. This helps developers to develop advance and complex pipeline as code

SCRIPTED JENKINSFILE CONCEPTS

```
node {

}
```

NODE

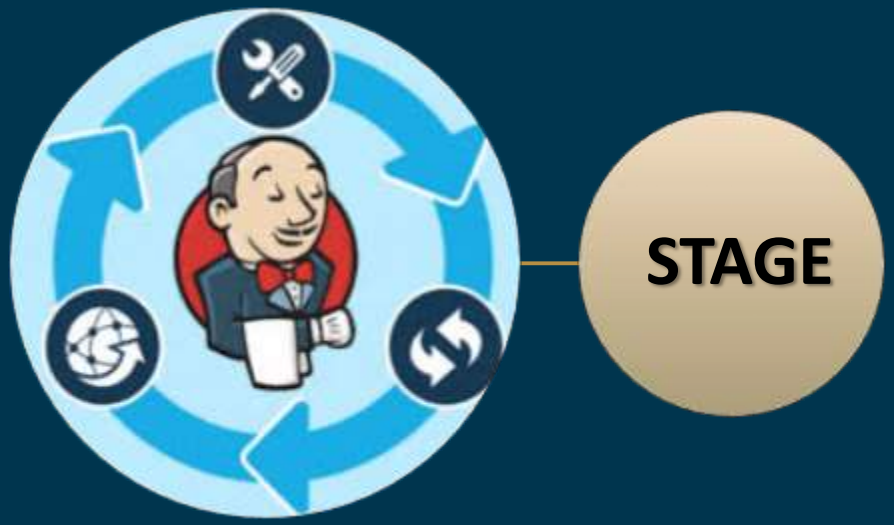
A node is a machine that executes an entire workflow

```
// Jenkinsfile (Scripted Pipeline)
node { // node/agent
  stage('Stage 1') {
    echo 'Hello World' // echo Hello World
  }
}
```

STAGE

Stage block can be a single stage or multiple as the task goes. That is, the build, test, and deploy processes all come together in a stage. Generally, a stage block is used to visualize the Jenkins pipeline process.

SCRIPTED JENKINSFILE CONCEPTS



STAGE

- Cloning the code from SCM
- Building the project
- Deploying the code
- Running the Unit Test cases
- Other functional and performance tests

```
node ('node-1') {
  stage('Source') {
    git 'https://github.com/digitalvarys/jenkins-tutorials.git''
  }
  stage('Compile') {
    def gradle_home = tool 'gradle4'
    sh "'${gradle_home}/bin/gradle' clean compileJava test"
  }
}
```

SIGMA

USING A JENKINSFILE WITH EXAMPLES

Pipeline

Definition: Pipeline script

Script

```
node('test'){
    stage('stage1') {
        sh '''echo  stage1 steps'''
    }
    stage('stage2') {
        sh '''echo stage2 steps'''
    }
    stage('stage3') {
        sh '''echo stage3 steps'''
    }
```

The **sh** step is used to execute a shell command in a Pipeline

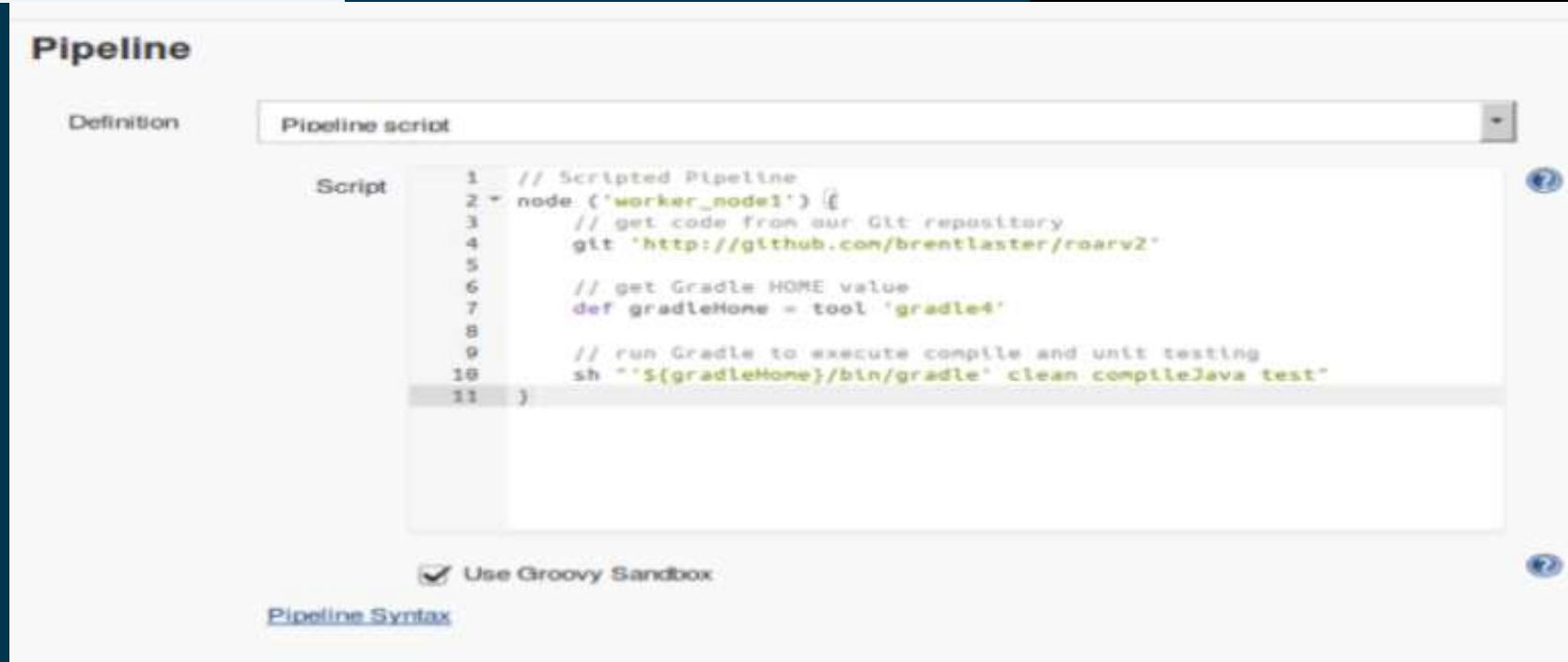
```
String branchName = env.BRANCH_NAME
String gitCredentials = "CREDENTIAL_ID"
String repoUrl = "https://github.com/username/repo-name.git"

node ('worker_node1') {
    stage('Source') {
        // Get some code from our Git repository
        git 'http://github.com/brentlaster/roarv2'
    }
}

    dir('build') {
        git branch: branchName, credentialsId:       gitCredentials, url: repoUrl
    }
  }
}
```

Get code from a git repository in jenkinsfile

SIGMA

USING A JENKINSFILE WITH EXAMPLES

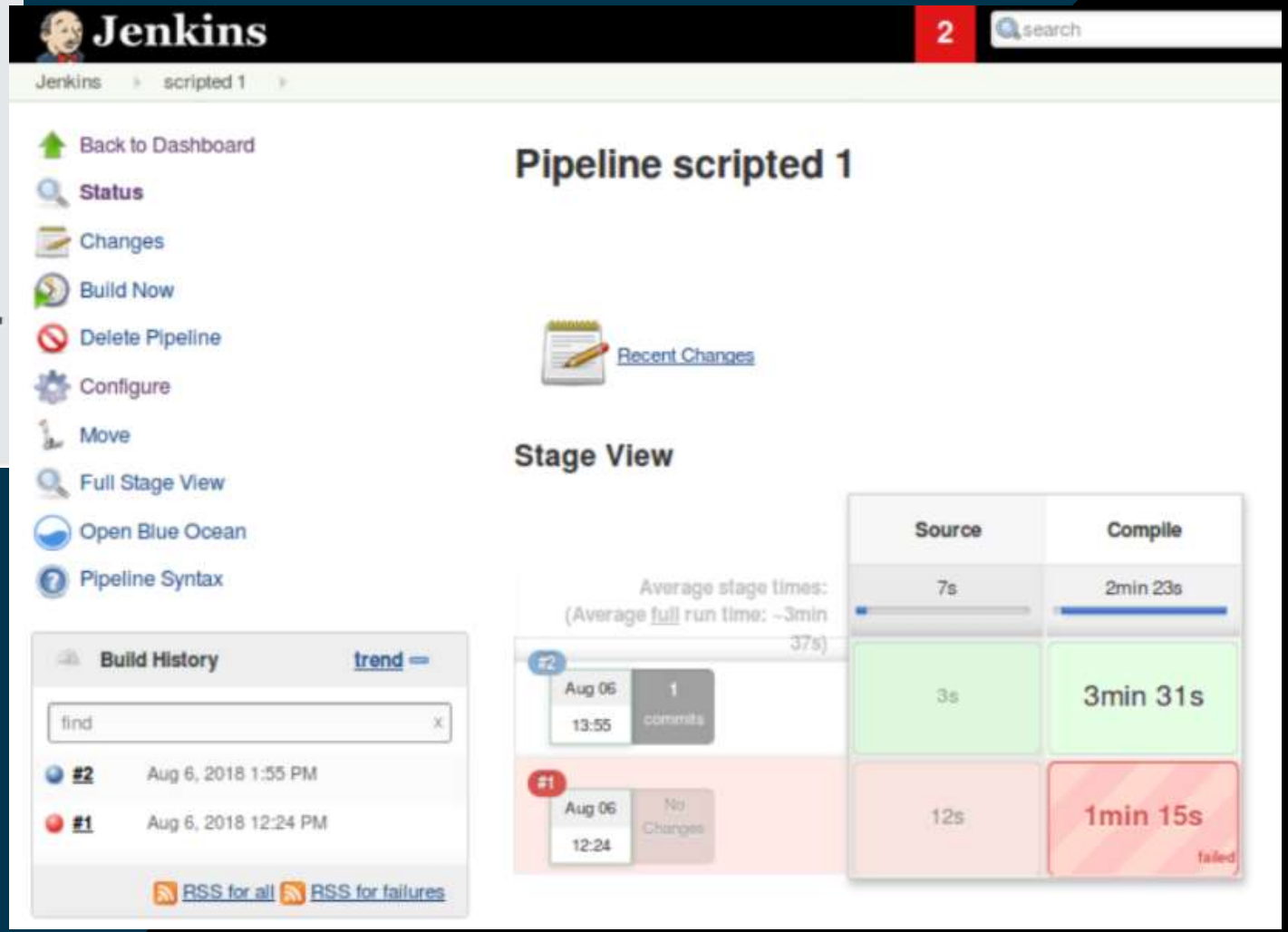
```
// Scripted Pipeline
node ('worker_node1') {
  // get code from our Git repository
  git 'http://github.com/brentlaster/roarv2'
   // get Gradle HOME value
  def gradleHome = tool 'gradle4'
   // run Gradle to execute compile and unit testing
  sh "'${gradleHome}/bin/gradle' clean compileJava test"
}
```



SIGMA

USING A JENKINSFILE WITH EXAMPLES

```
// Scripted Pipeline with stages
node ('worker_node1') {
  stage('Source') { // Get code
    // get code from our Git repository
    git 'http://github.com/brentlaster/roarv2'
  }
  stage('Compile') { // Compile and do unit testing
    // get Gradle HOME value
    def gradleHome = tool 'gradle4'
    // run Gradle to execute compile and unit testing
    sh "'${gradleHome}/bin/gradle' clean compileJava test"
  }
}
```



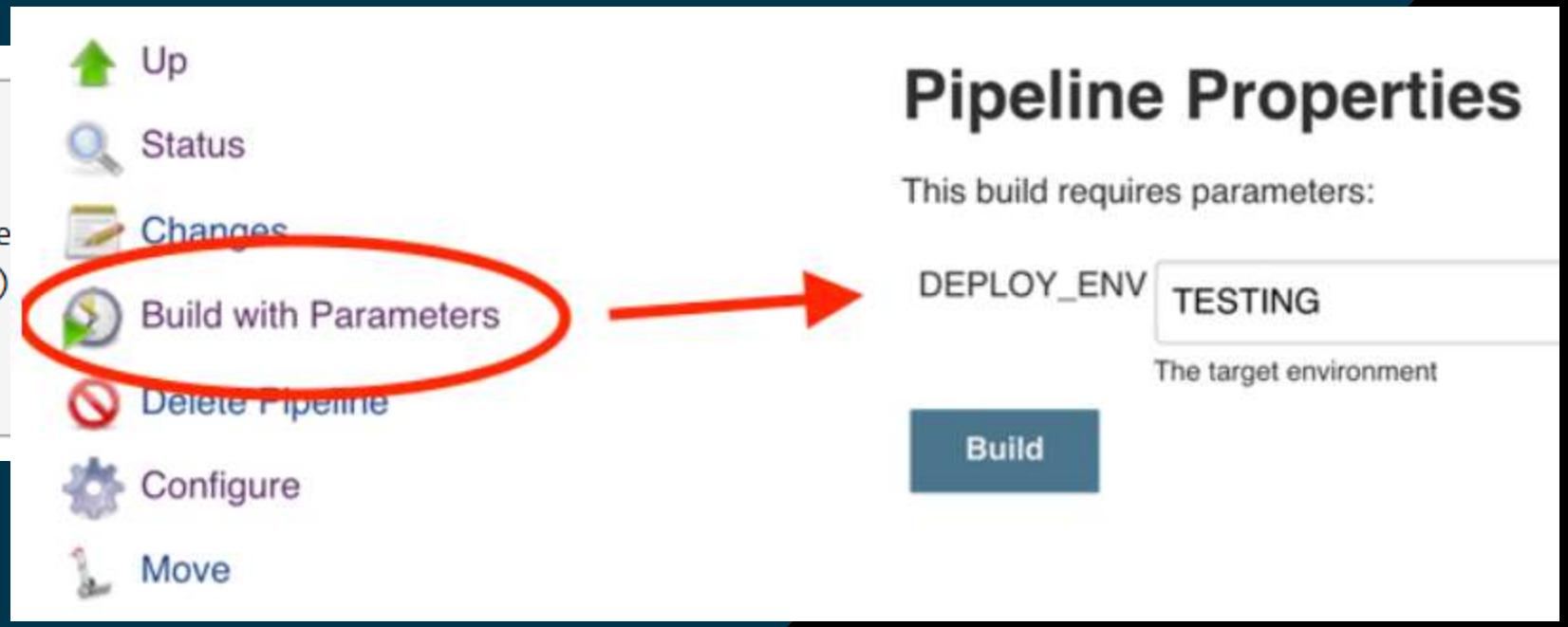
SIGMA

USING A JENKINSFILE WITH EXAMPLES

`ws` allocates a new workspace. you would use this to ensure that nothing else interferes with the location on disk where you are running the enclosed steps.

- this is not as heavy-handed as the `node` step, since `node` will also ensure that it gets run with a separate executor.
- this provides more isolation than the `dir` step, since `dir` will not ensure an isolated location on the filesystem the way `ws` will.

```
properties([
  parameters([
    string(name
environment', )
  ])
])
```



SIGMA

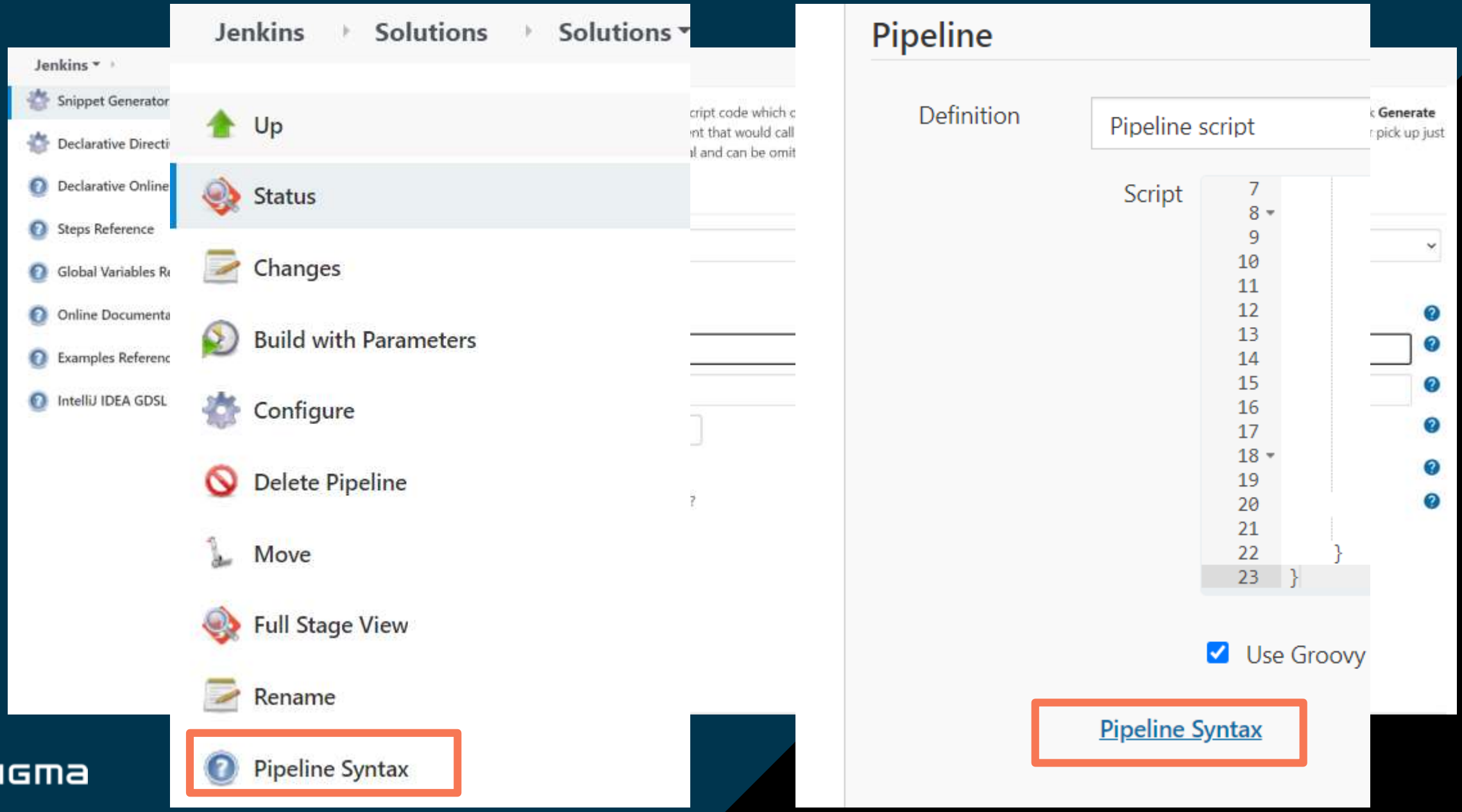
USING A JENKINSFILE WITH EXAMPLES

`build`: Build a job

Triggers a new build for a given job.

- `job`
  Name of a downstream job to build. May be another Pipeline job, but more commonly a freestyle or other project. Use a simple name if the job is in the same folder as this upstream Pipeline job; otherwise can use relative paths like `../sister-folder/downstream` or absolute paths like `/top-level-folder/nested-folder/downstream`.
  - **Type:** `String`
- `parameters` (optional)
  Array/List
  Nested Choice of Objects
  - ⊞ `booleanParam`
  - ⊞ `buildMasterRelease`

JENKINS PIPELINE SNIPPET GENERATOR



JENKINS DEMO