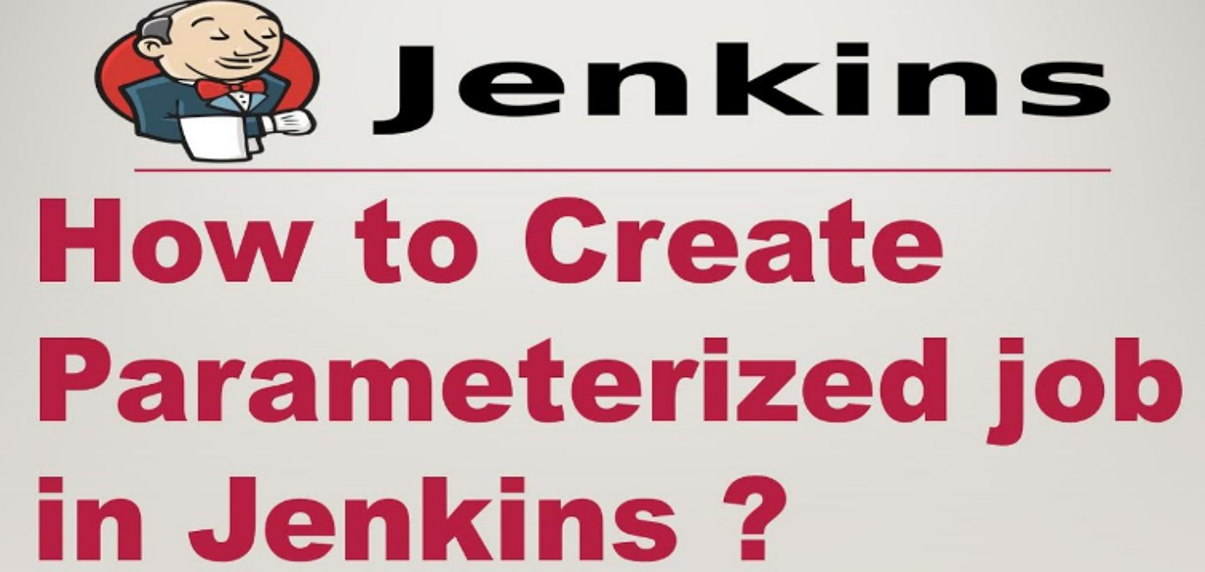
**Parameterization in Jenkins job**

****

**Parameters in Jenkins:**

Parameters can be used in 2 ways in Jenkins job.

* Define parameter value directly inside a build job (or in shell/batch commands) as fixed value
* Define using Build Parameters - Setting build parameter and passing values to the parameter during execution.

**1) Define parameter value(fixed) directly inside job**

**A) Traditional Jobs** (with Shell or batch command)

* Jobs with batch commands, we use the native Windows syntax:

${packageType}

* We can also create build steps that execute Gradle tasks or Maven

%packageType%

**B) Pipelines Jobs**

* Inside a Jenkins Pipeline, we can access a build parameter (static parameter variable) in multiple ways.
* First, all build parameters are placed into a params variable. This means we can access a parameter value using dot notation:

pipeline {

agent any

stages {

stage('Build') {

when {

expression { params.jdkVersion == "14" }

}

}

}

}

Second, the build parameters are added to the environment of the pipeline. This means we can use the shorter shell syntax inside a step that executes a shell script:

pipeline {

agent any

stages {

stage('Build') {

steps {

echo "${packageType}"

}

}

}

}

**2)Build parameters – Passing values during execution/run time of Jenkins Job**



* A build parameter allows us to pass data into our Jenkins jobs.
* Using build parameters, we can pass any data we want: git branch name, secret credentials, hostnames and ports, and so on.
* We can define Build parameters using

->**UI web console** (UI menu options)

->**Defined in Pipeline job** (UI pipeline script section)

->**Defined in Pipeline job** (pipeline in Jenkins file)

**Types of Build Parameters**

Jenkins supports several parameter types. Below is a list of the most common ones, but keep in mind that different plugins may add new parameter types:

**String**: any combination of characters and numbers

**Choice**: a pre-defined set of strings from which a user can pick a value

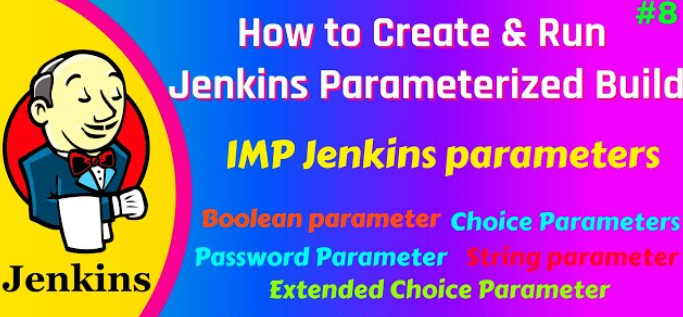
**Credentials**: a pre-defined Jenkins credential

**File**: the full path to a file on the filesystem

**Multi-line String**: same as String, but allows newline characters

**Password**: similar to the Credentials type, but allows us to pass a plain text parameter specific to the job or pipeline

**Run**: an absolute URL to a single run of another job



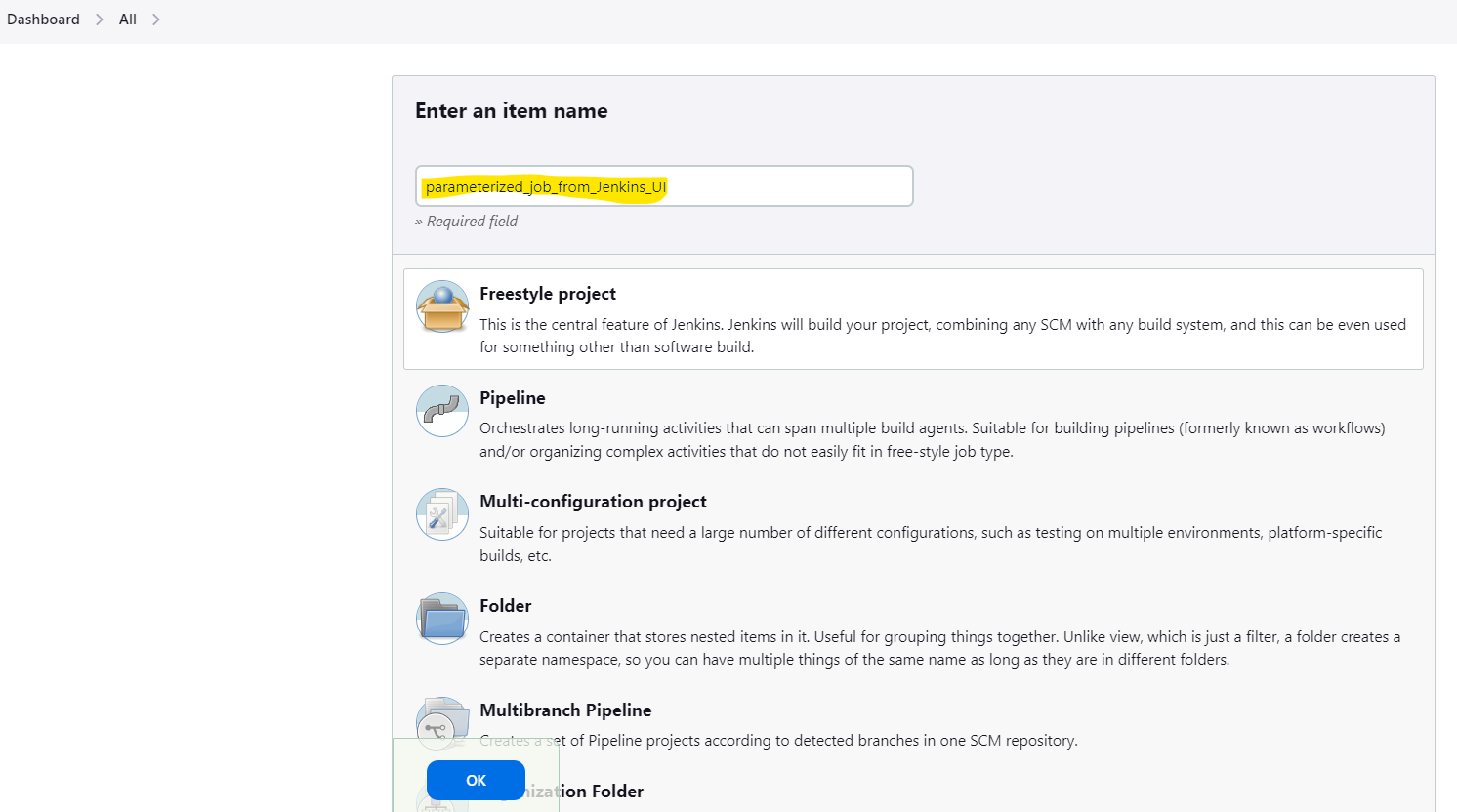
**Jenkins Plugin to use Build Parameters feature:**

Install the Parameterized Build Plugin (if not already installed): If you don’t have the Parameterized Build plugin installed in your Jenkins instance, you need to install it first.

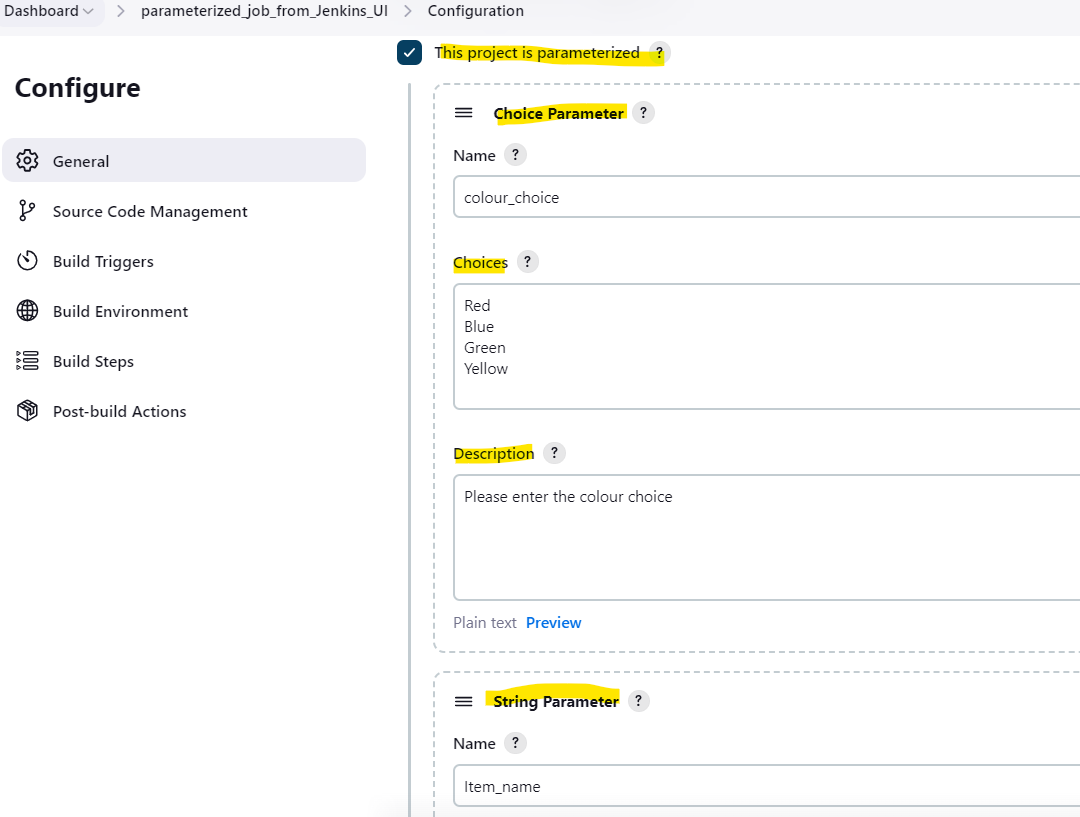
Manage Jenkins” -> “Manage Plugins” -> “Available” tab, searching for “Parameterized Build” plugin, and installing it.

**Example Job - : Parameter Build (using Freestyle job in Web UI )**

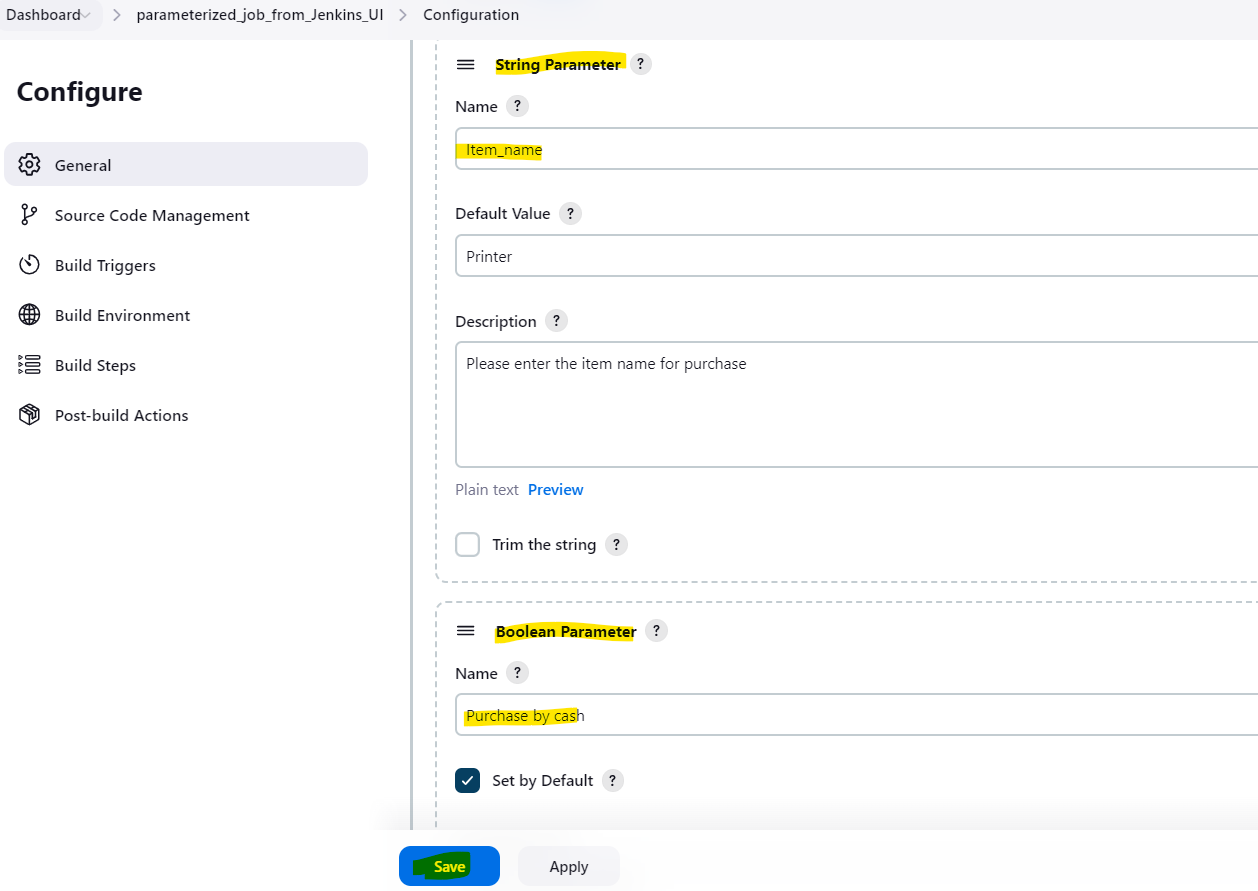
Create a new job –in console



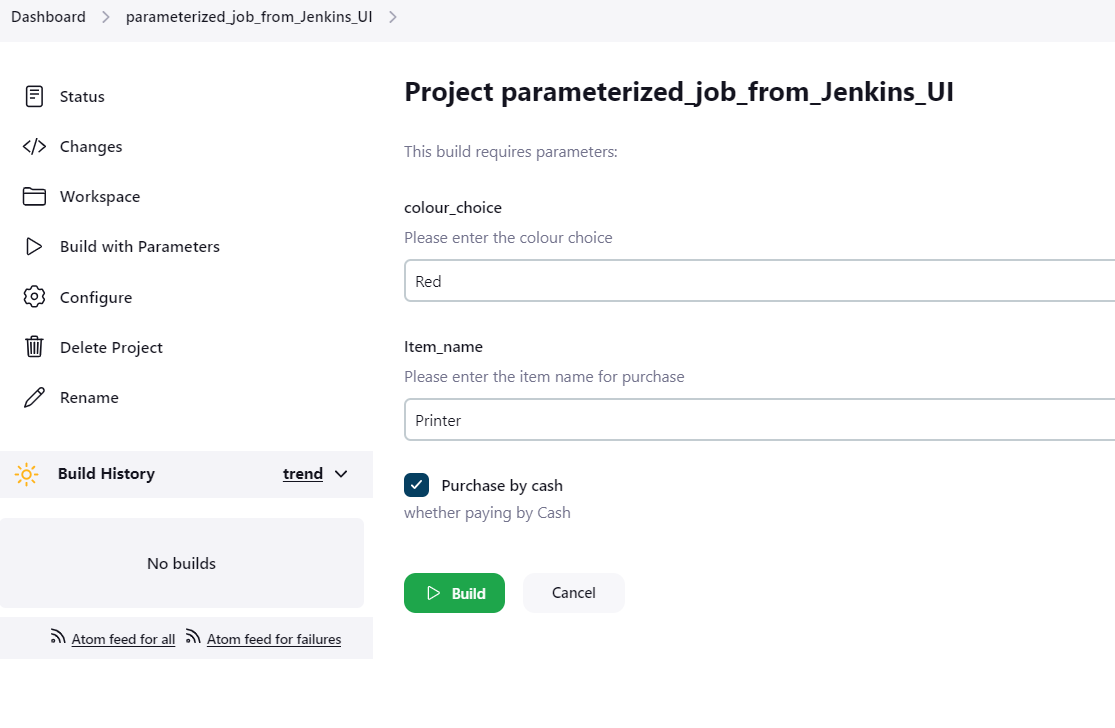
Configure the job 🡪This build is parameterized ( under Build Triggers)



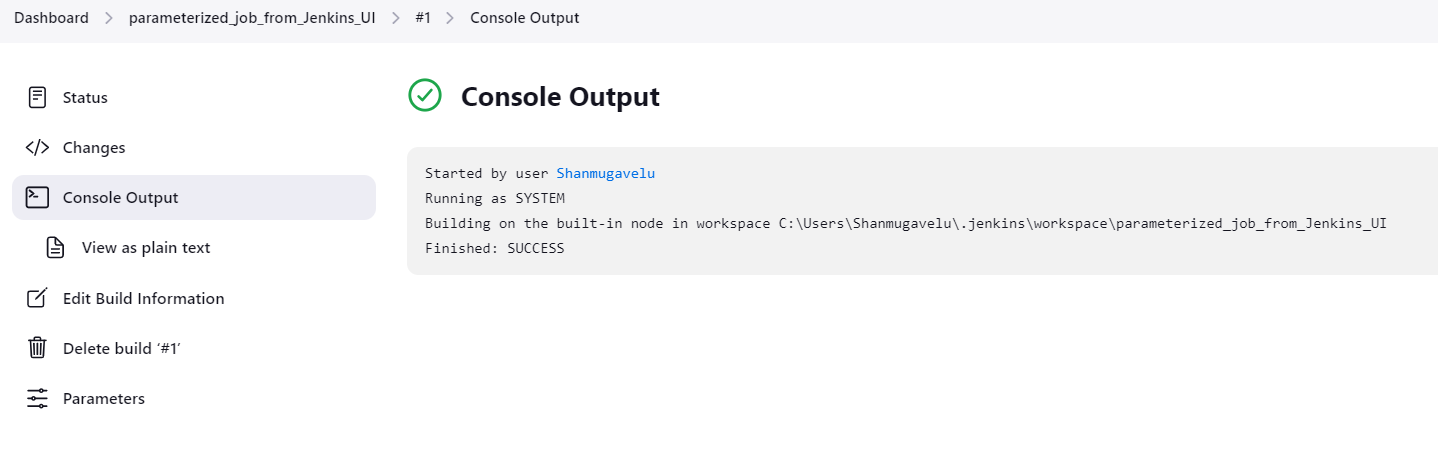
Add parameters –different kinds of parameters.



Trigger the build using the option – Build with Parameters .

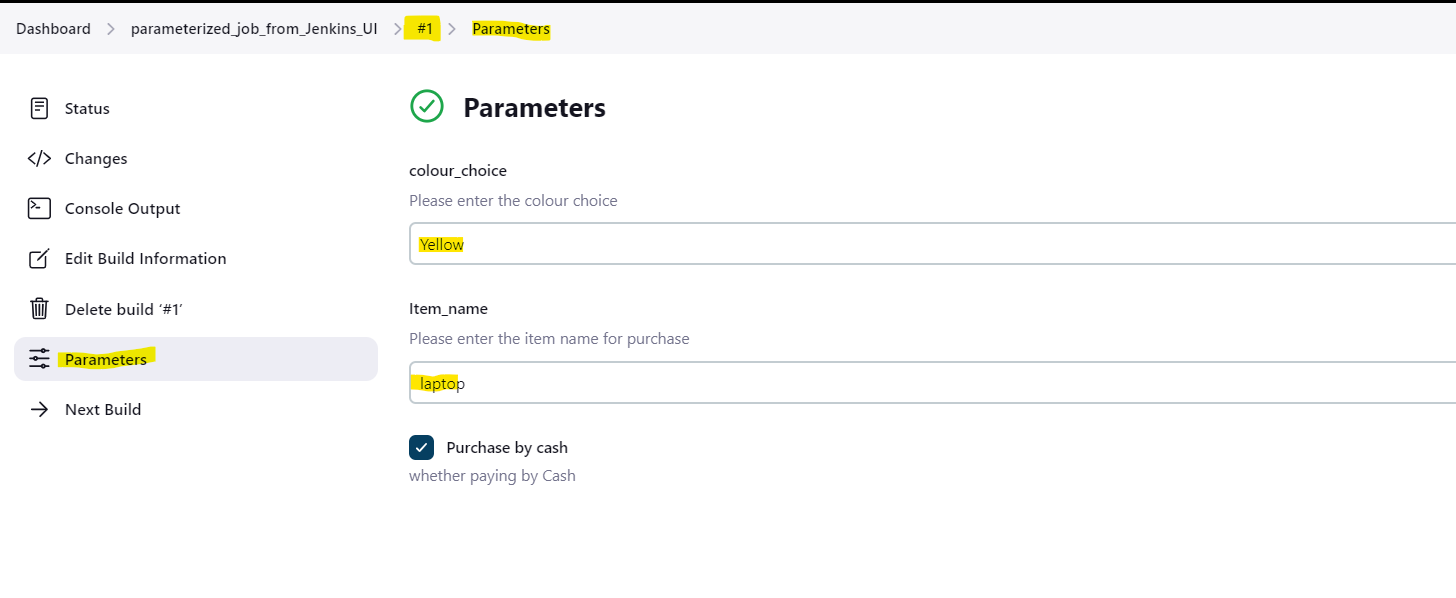


Check the console output:

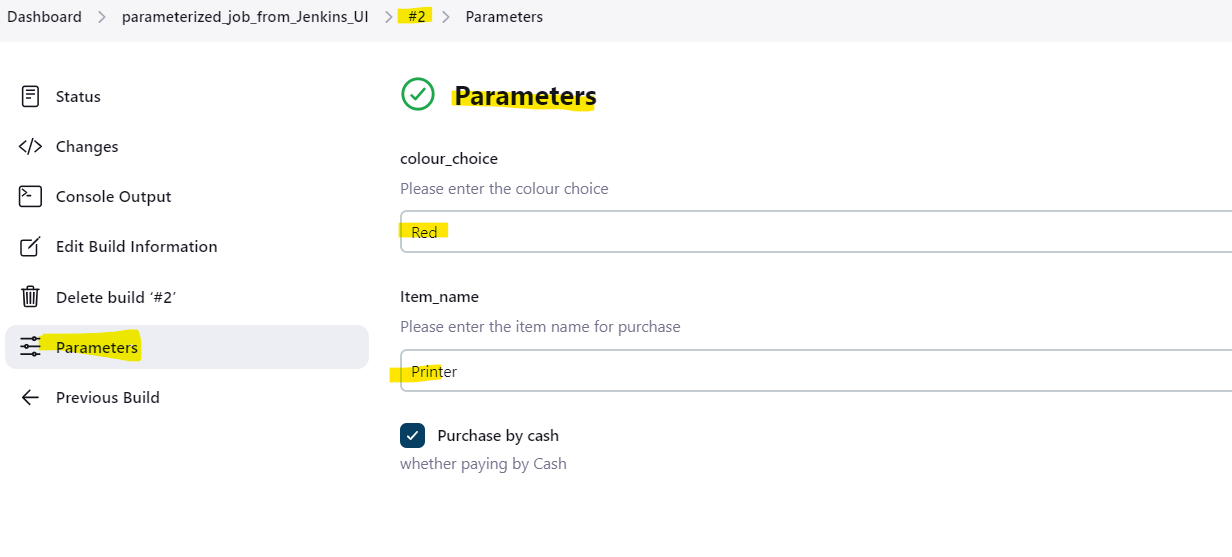


--We can check the parameters used for each Build – Based on BUILD number

Check the parameter used for Job #1



Check the parameter used for Job #2



**Example Job - : Parameter Build in Declarative Pipeline job**

// Declarative Pipeline code //

pipeline {

agent any

parameters {

string(name: 'yyyyy', defaultValue: 'XXX', description: 'Hello world')

text(name: 'Demo', defaultValue: '', description: 'Demo parameter')

booleanParam(name: 'Boolean', defaultValue: true, description: 'Boolean value')

choice(name: 'CHOICE', choices: ['A', 'B', 'C'], description: 'Choose one')

password(name: 'PASSWORD', defaultValue: 'Key', description: 'Enter a password')

file(name: "FILE", description: "file to upload")

}

stages {

stage('Example') {

steps {

echo "Hello ${params.yyyyy}"

echo "Biography: ${params.Demo}"

echo "Toggle: ${params.Boolean }"

echo "Choice: ${params.CHOICE}"

echo "Password: ${params.PASSWORD}"

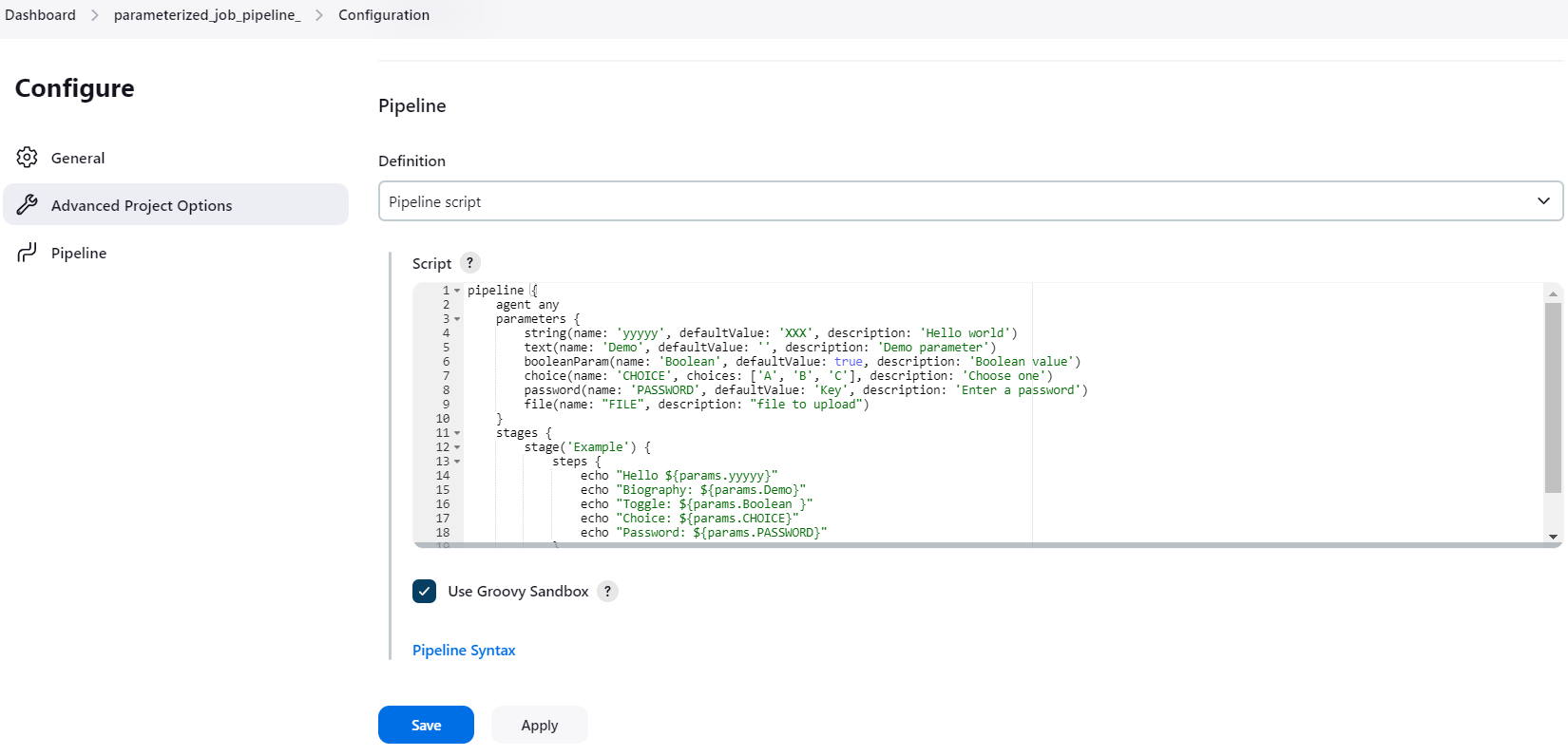
}

}

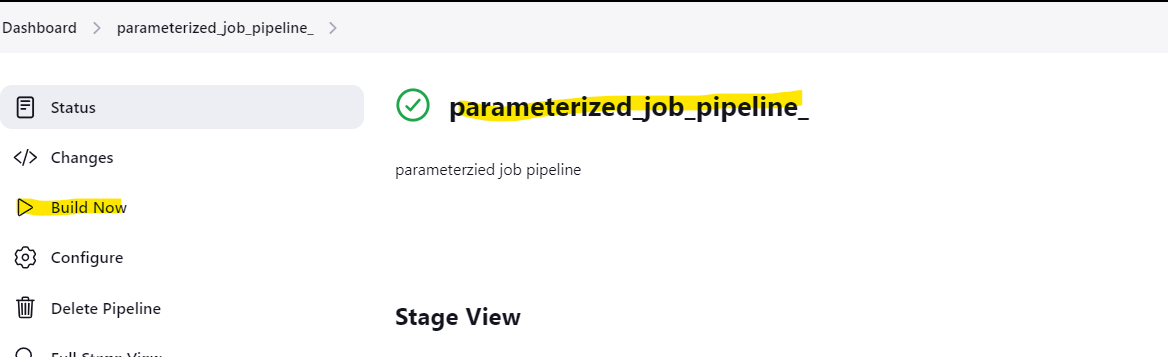
}

}

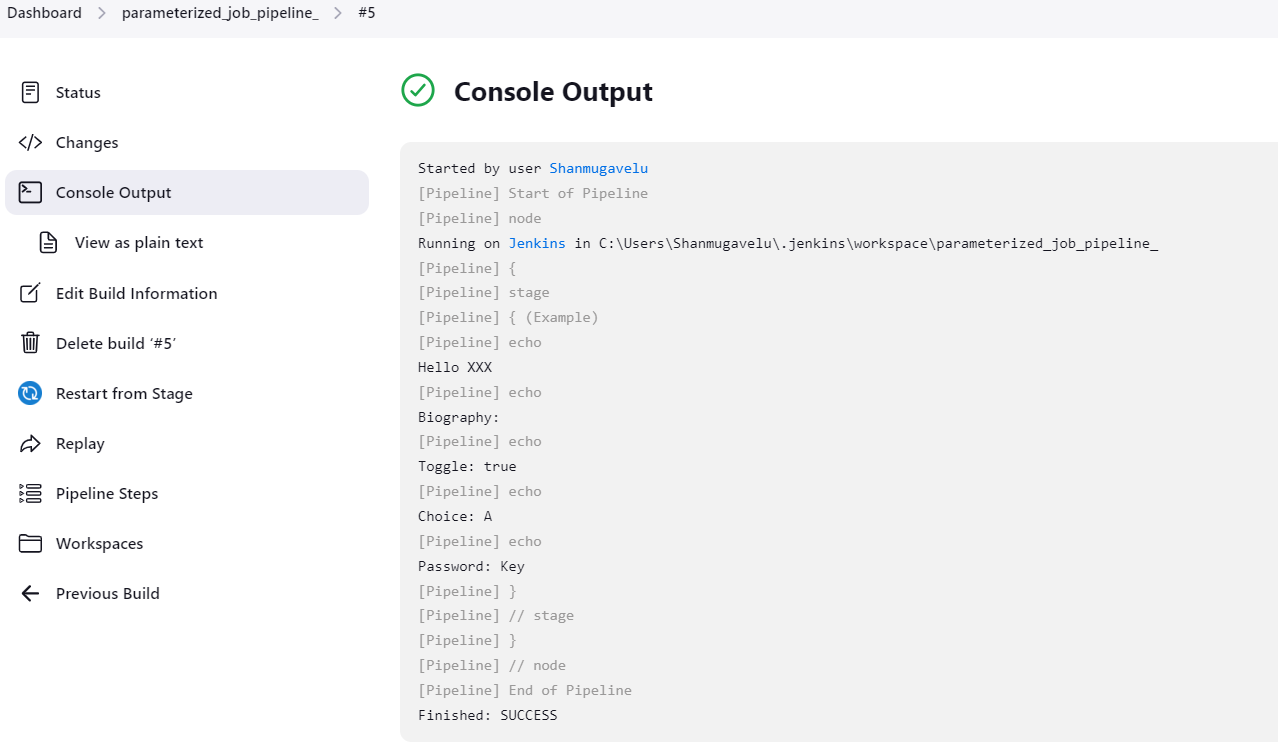
Enter the pipeline code under the pipeline script section in Web UI



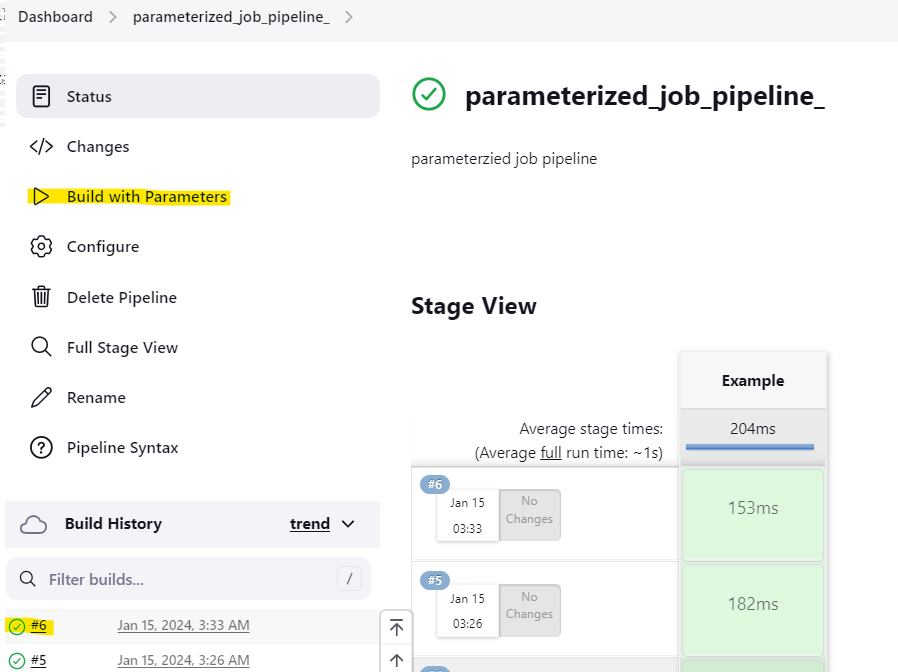
Enter the Build Now option – for the first RUN it will NOT show Build with parameters



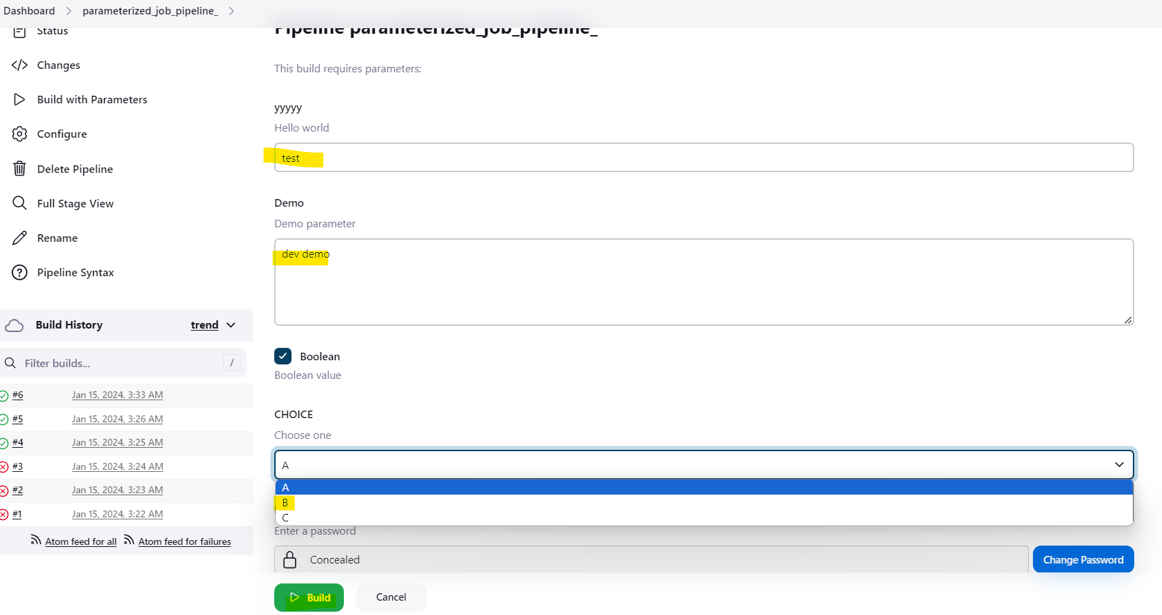
Check the console output for the first RUN:



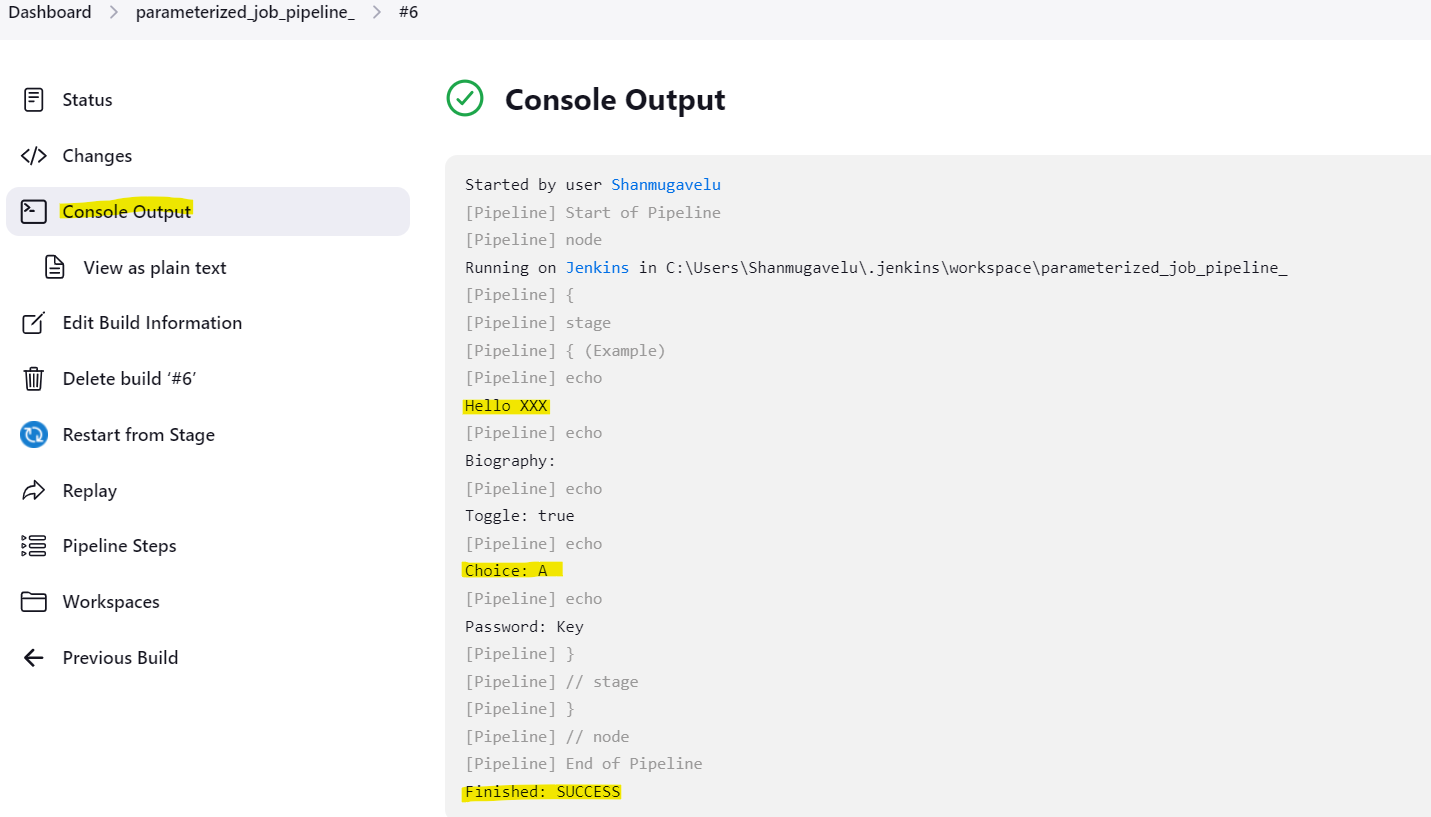
Perform the second run with option - Build with Parameters.



Enter the value under each filed and submit the BUILD.



Check the console output:



**Example Job - : Parameter Build in Scripted Pipeline job**

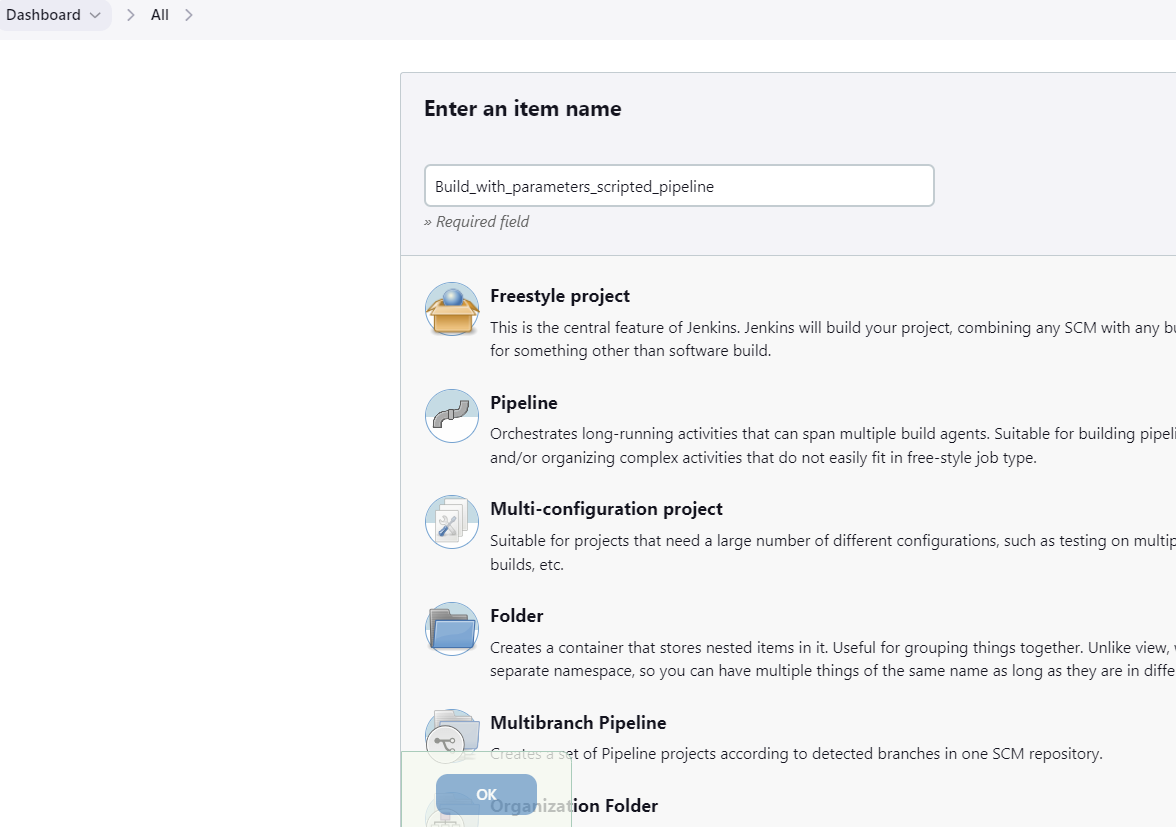
// Script Piple - Build with PARAMETERS //

properties([parameters([string(defaultValue: 'Hello', description: 'How should I greet the world?', name: 'Greeting')])])

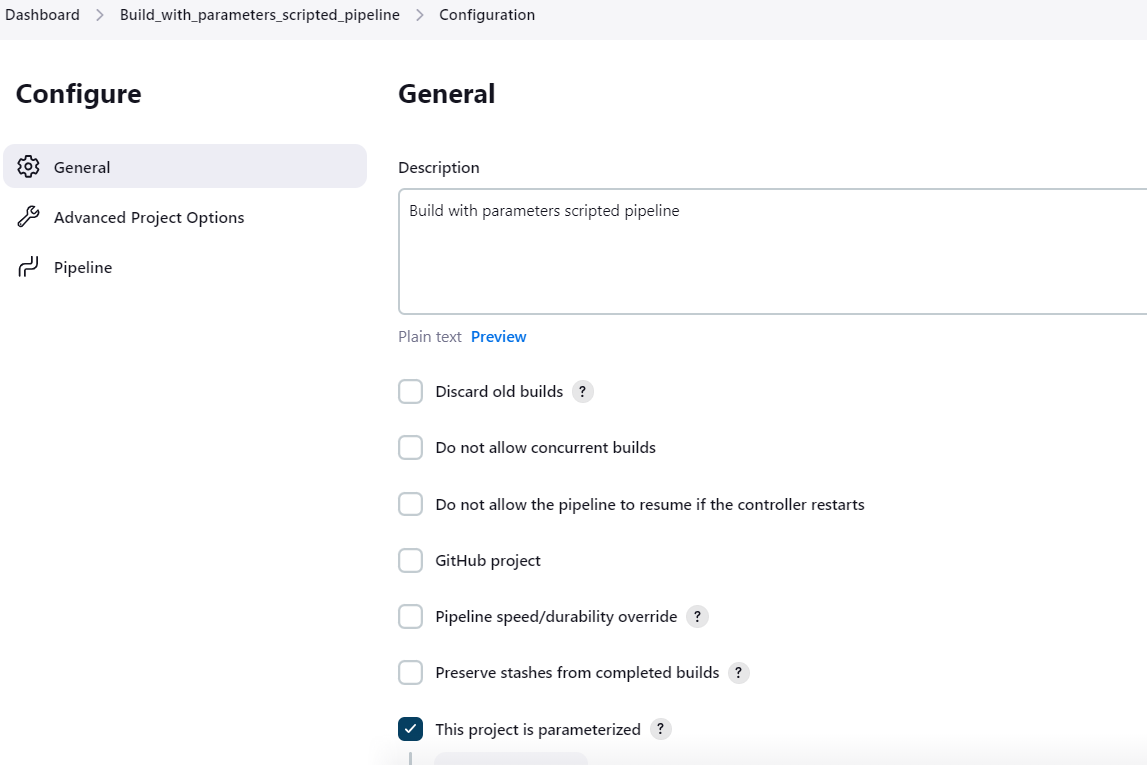
node {

echo "${params.Greeting} World!"

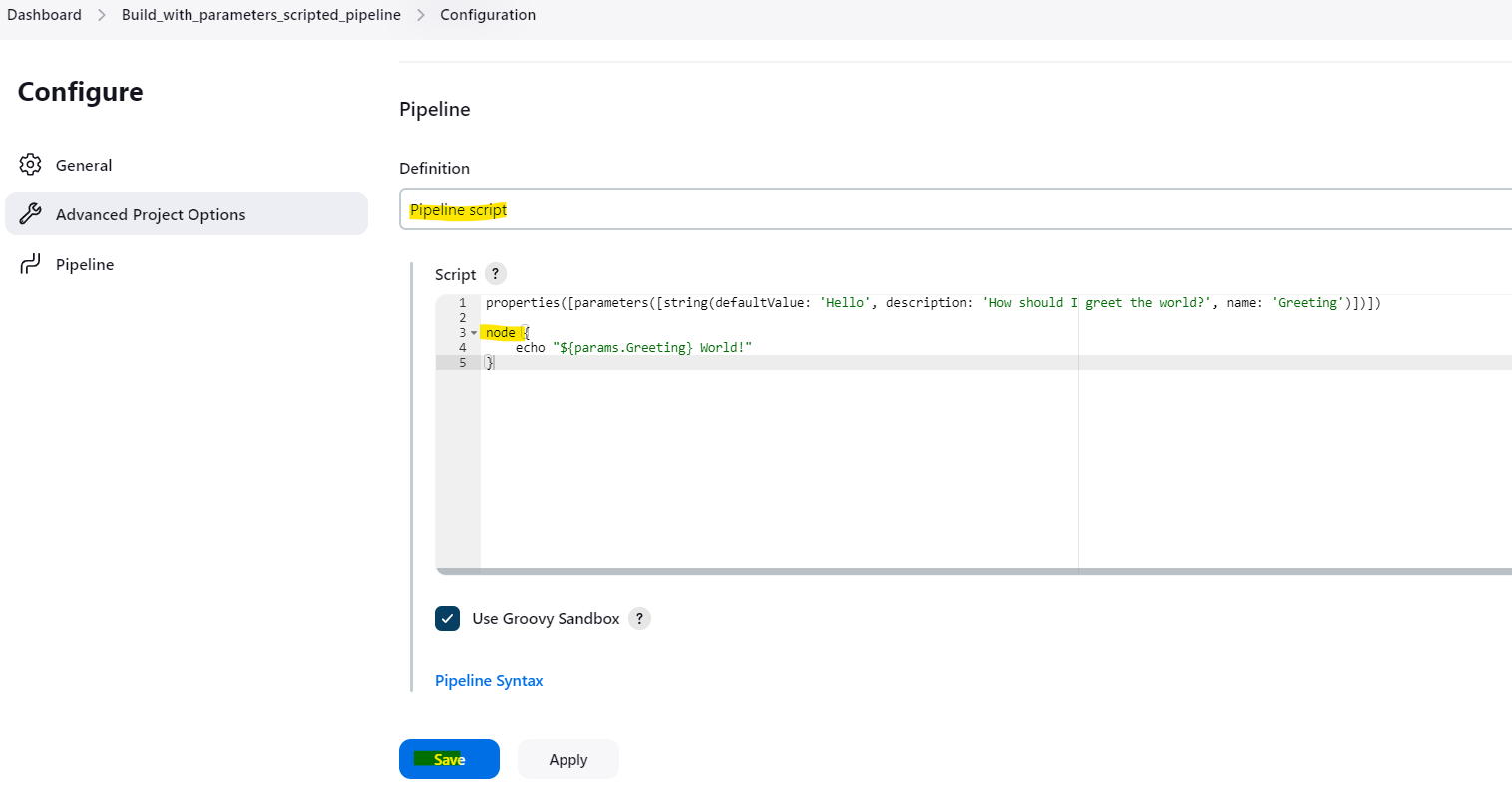
}  
Create the job in console:



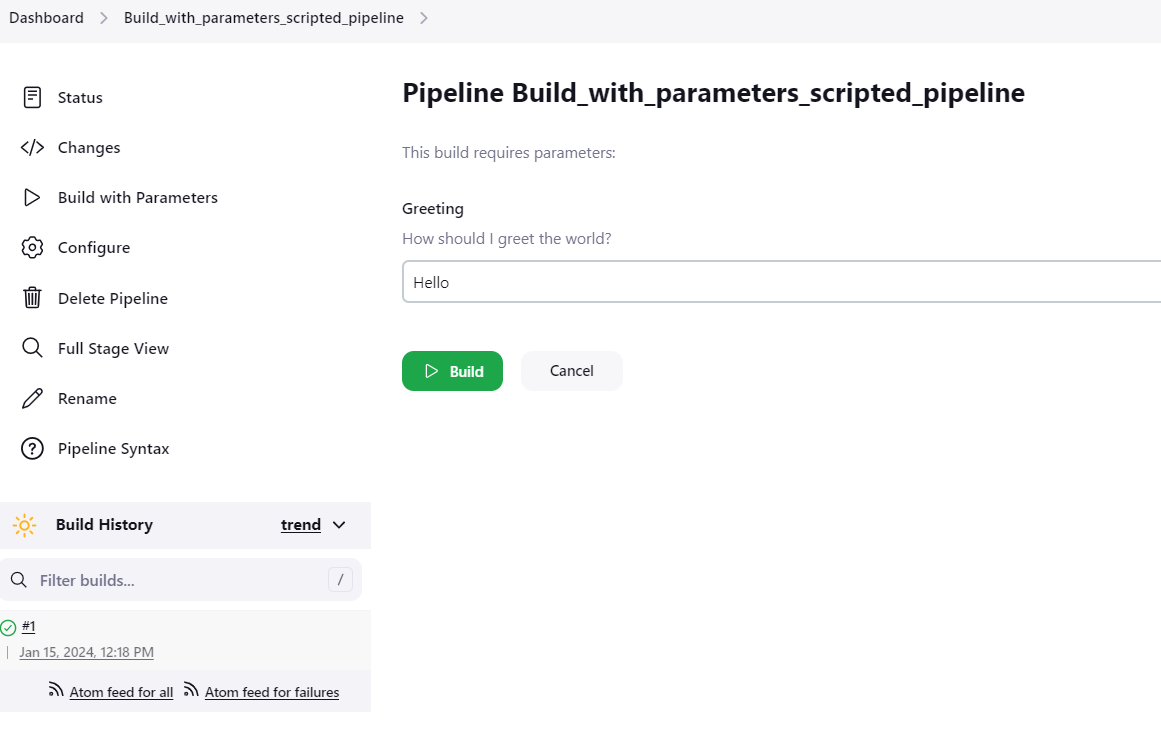
Configure the job for scripted pipeline



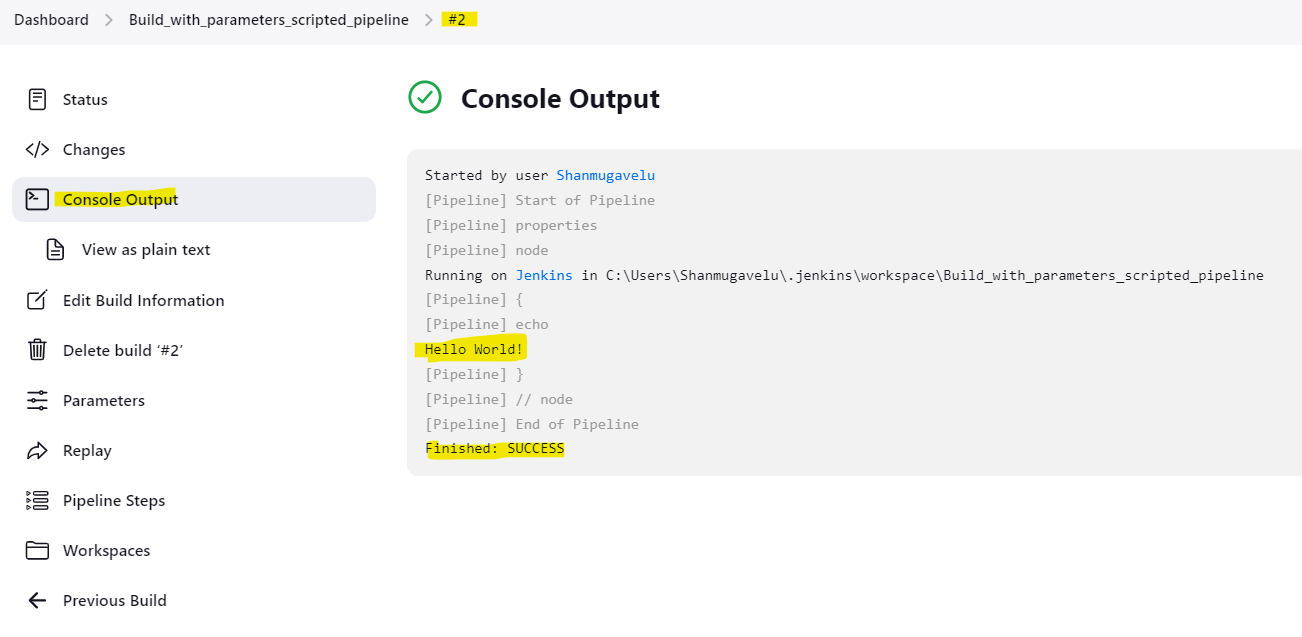
Add the code in console for scripted pipeline :



Build the code with option -Build with parameters during the second RUN



Check the console output :



**Example Job - : Parameter Build using JENKINS file**

// Pipline code stored in the Jenkins file //

pipeline {

agent any

stages {

stage('Setup parameters') {

steps {

script {

properties([

parameters([

choice(

choices: ['ONE', 'TWO'],

description: 'Choice for selecing the course',

name: 'COURSE\_number'

),

booleanParam(

defaultValue: true,

description: 'Will you join the course immediately',

name: 'Course\_joining\_time'

),

text(

defaultValue: '''

this is a multi-line

string parameter example

''',

description: 'Share the idea to start the new course ',

name: 'Idea\_regarding\_new\_course'

),

string(

defaultValue: 'java',

name: 'NEW\_course\_name',

description: 'Provide the course name to start newly ',

trim: true

)

])

])

}

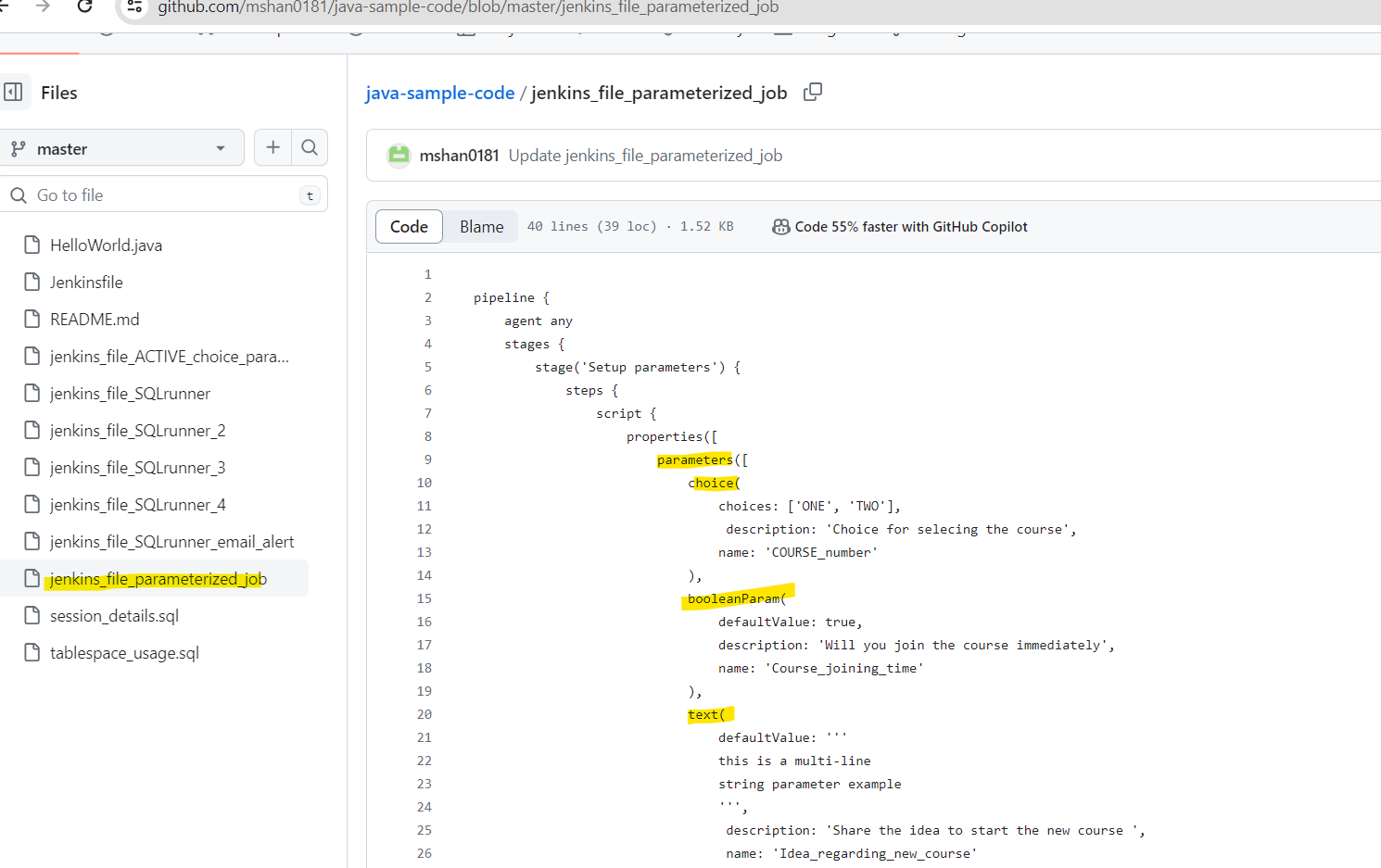
}

}

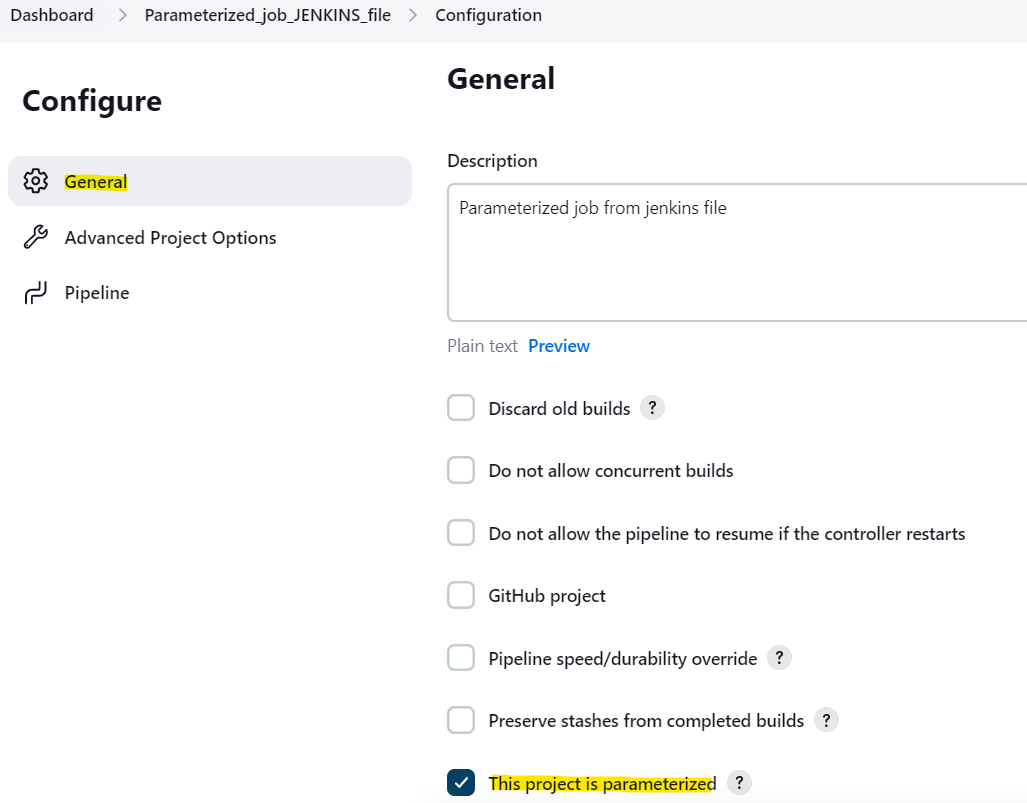
}

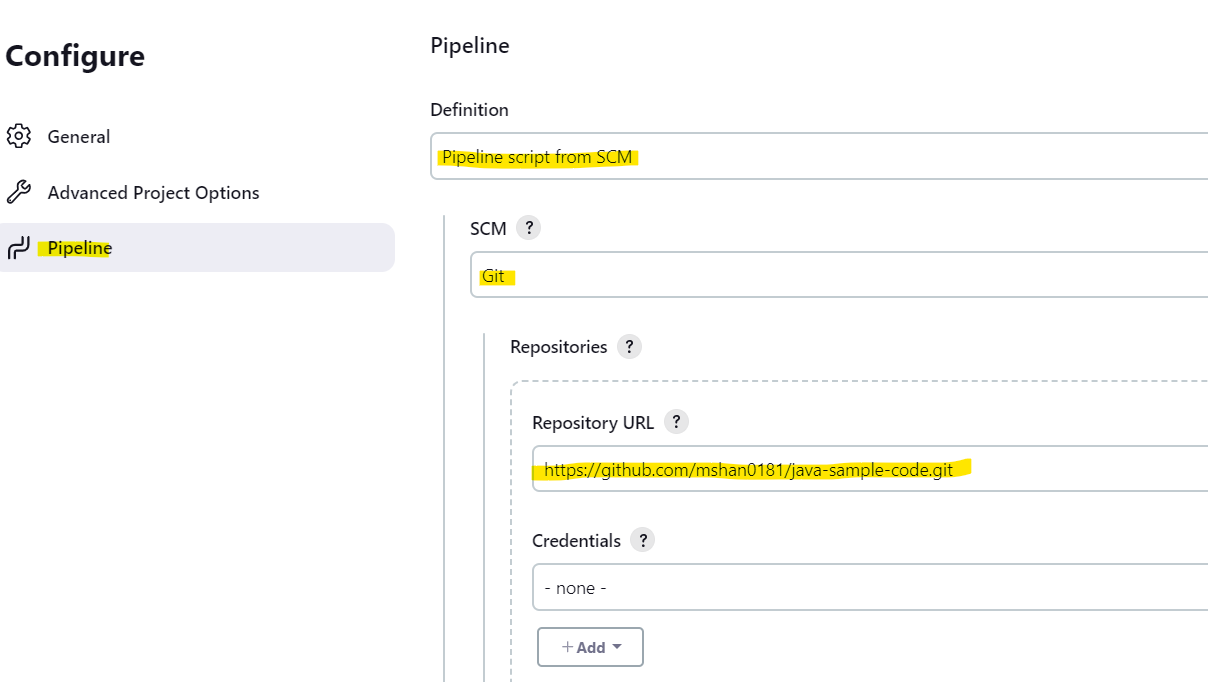
}

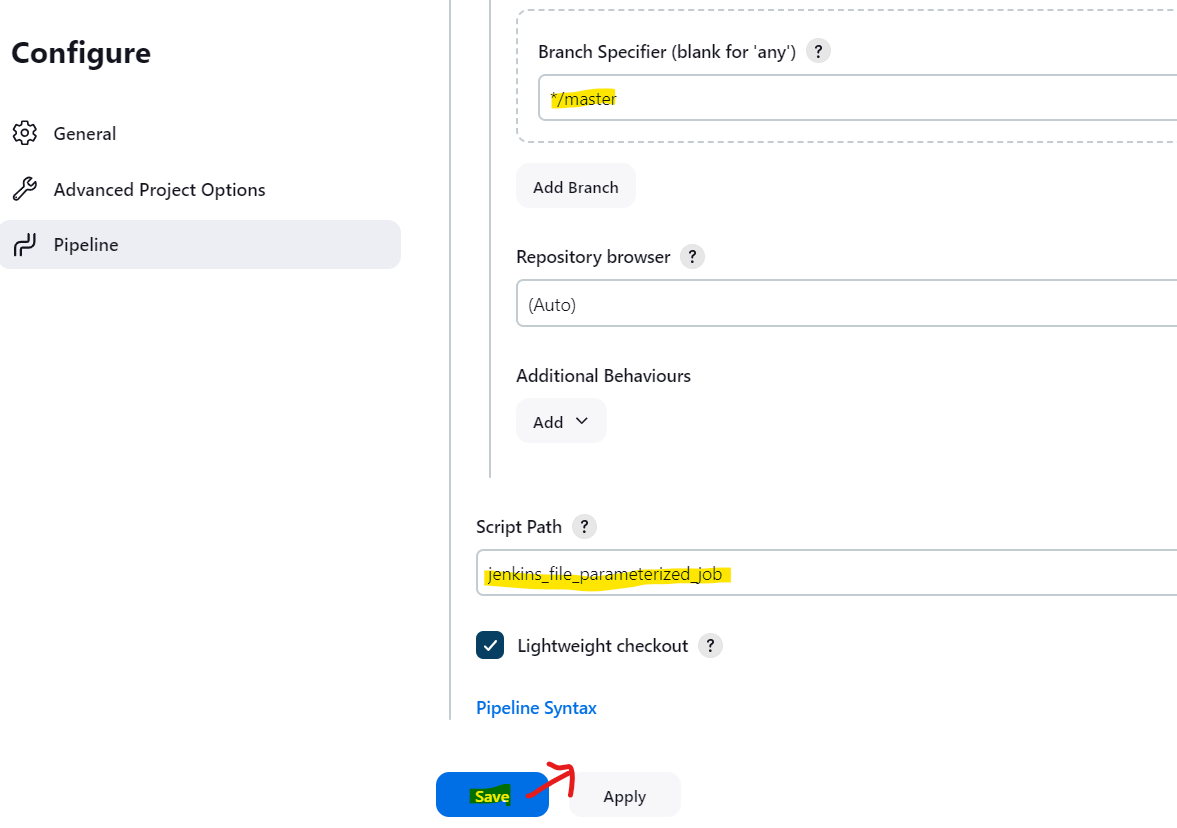
Save the code in the Jenkins file (in GIT Repo)



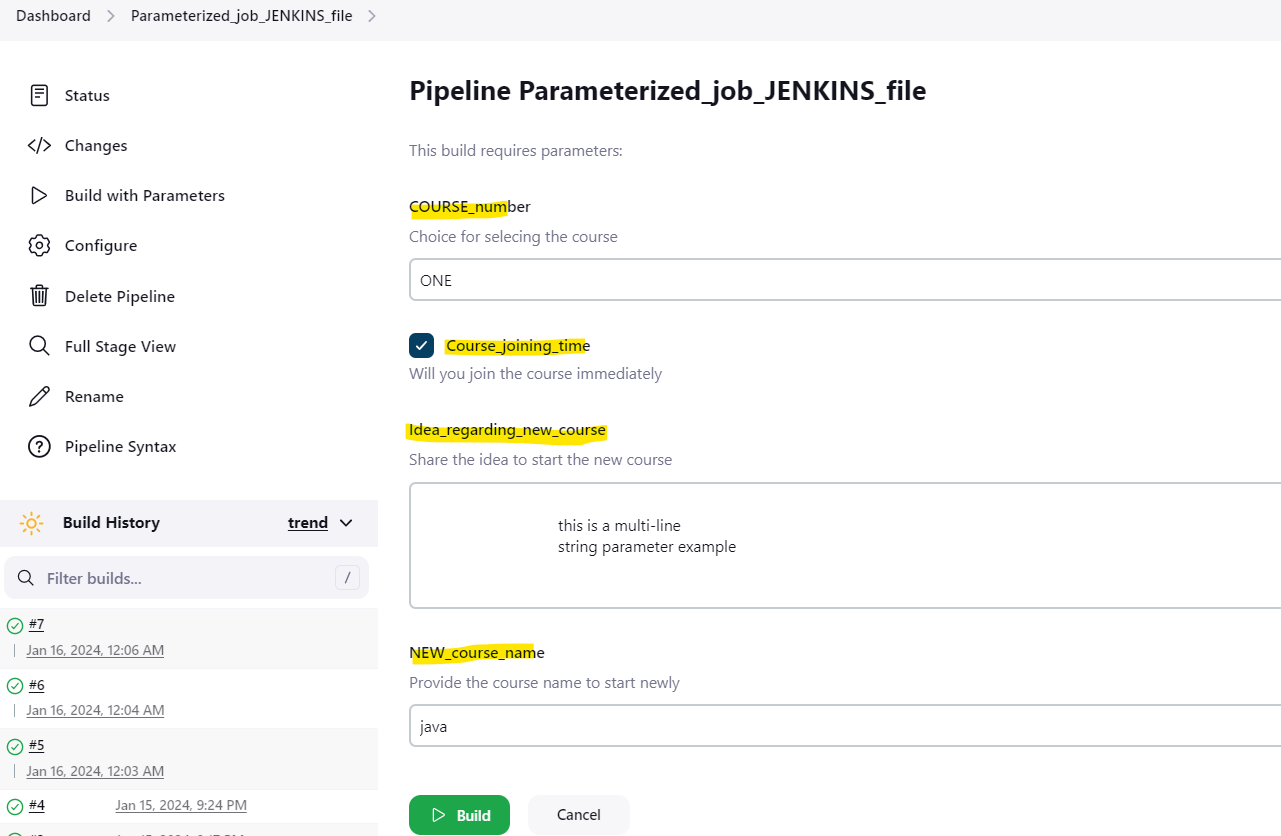
Add configuration details in Web UI



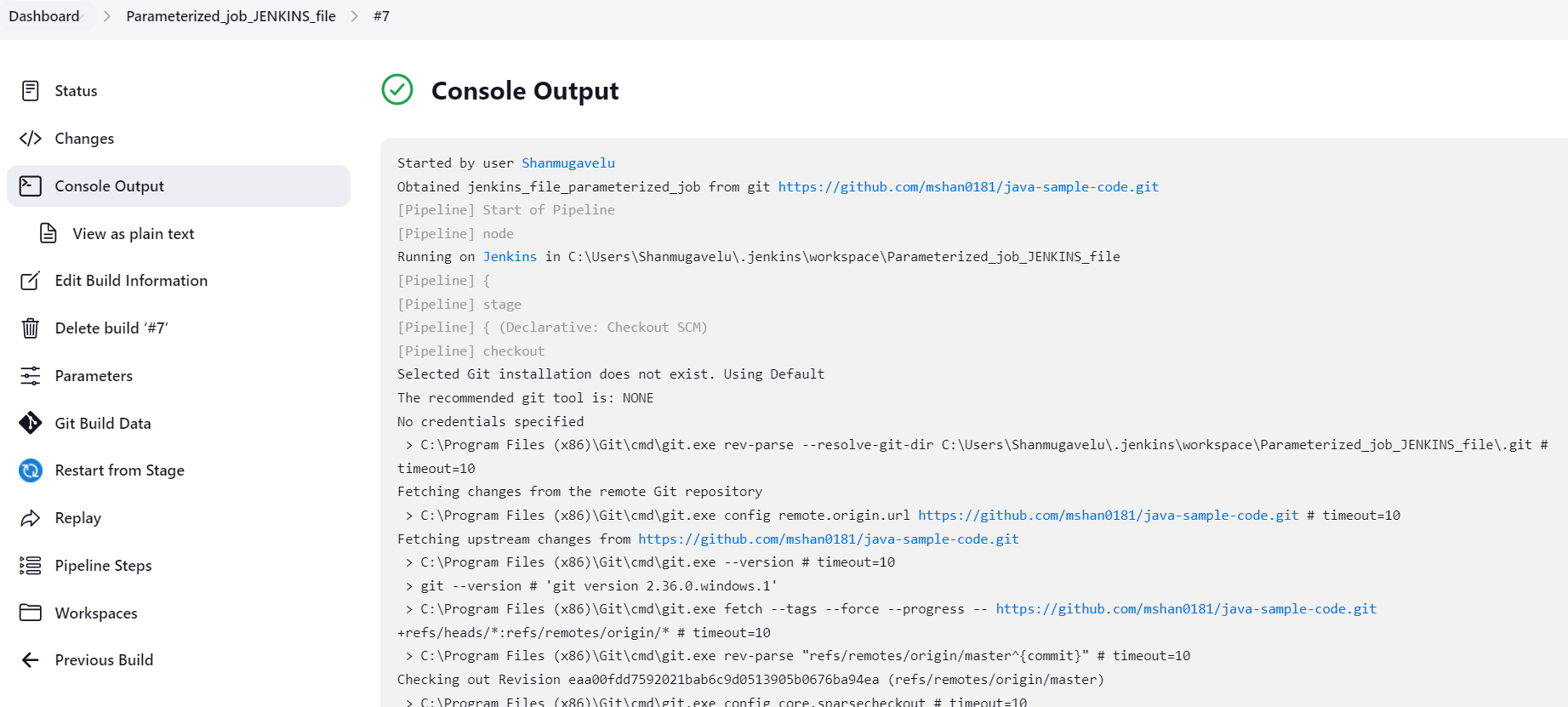


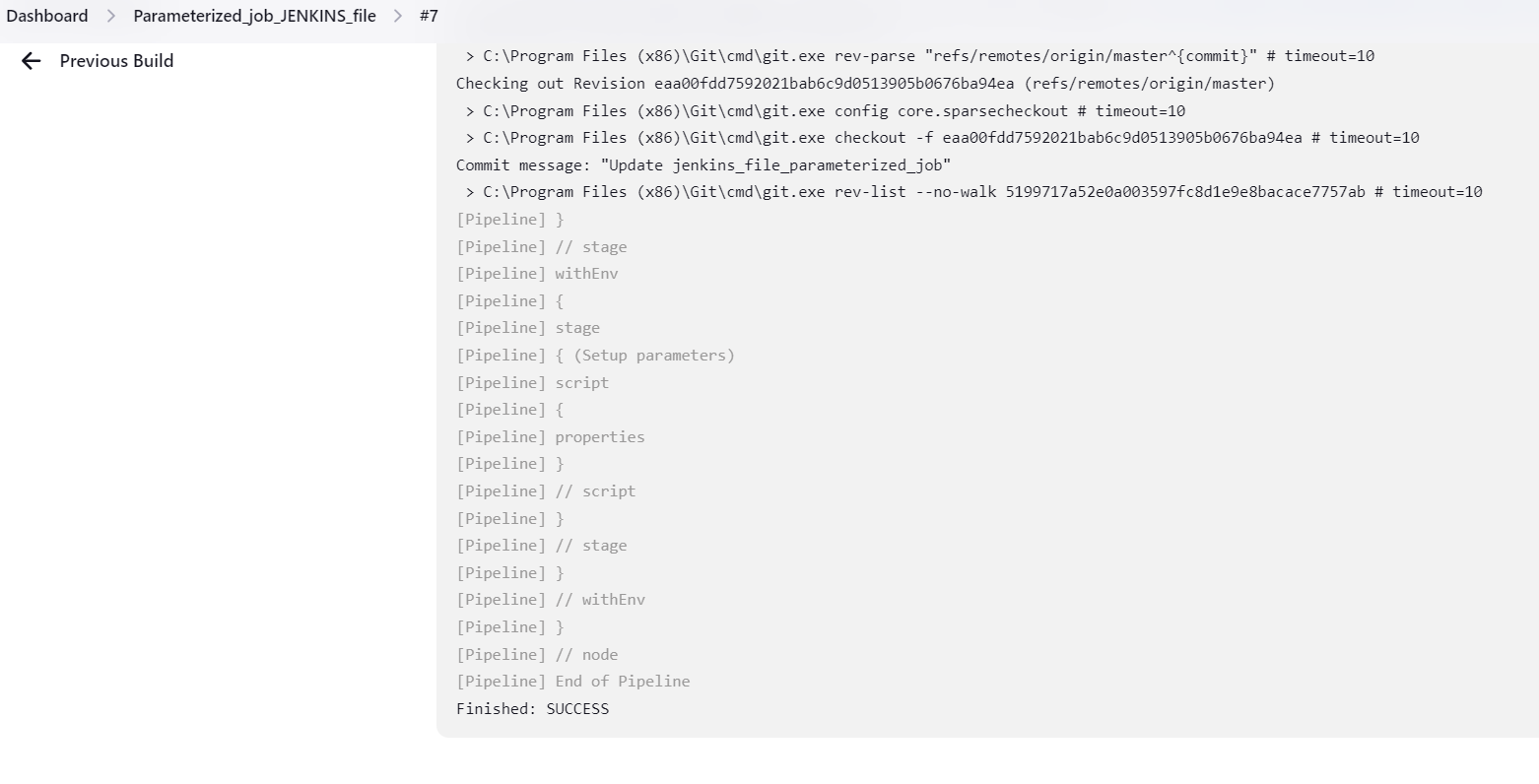


Build the code with option -Build with Paramter ( from the second RUN)

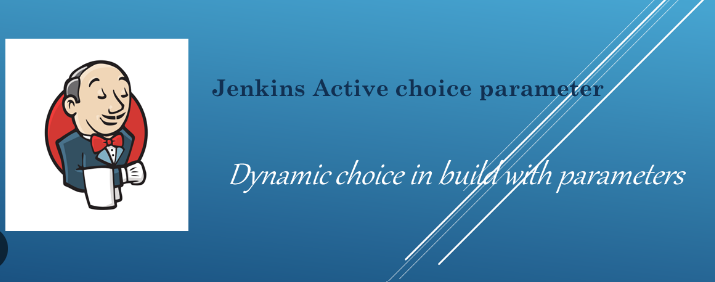


Check the console output:



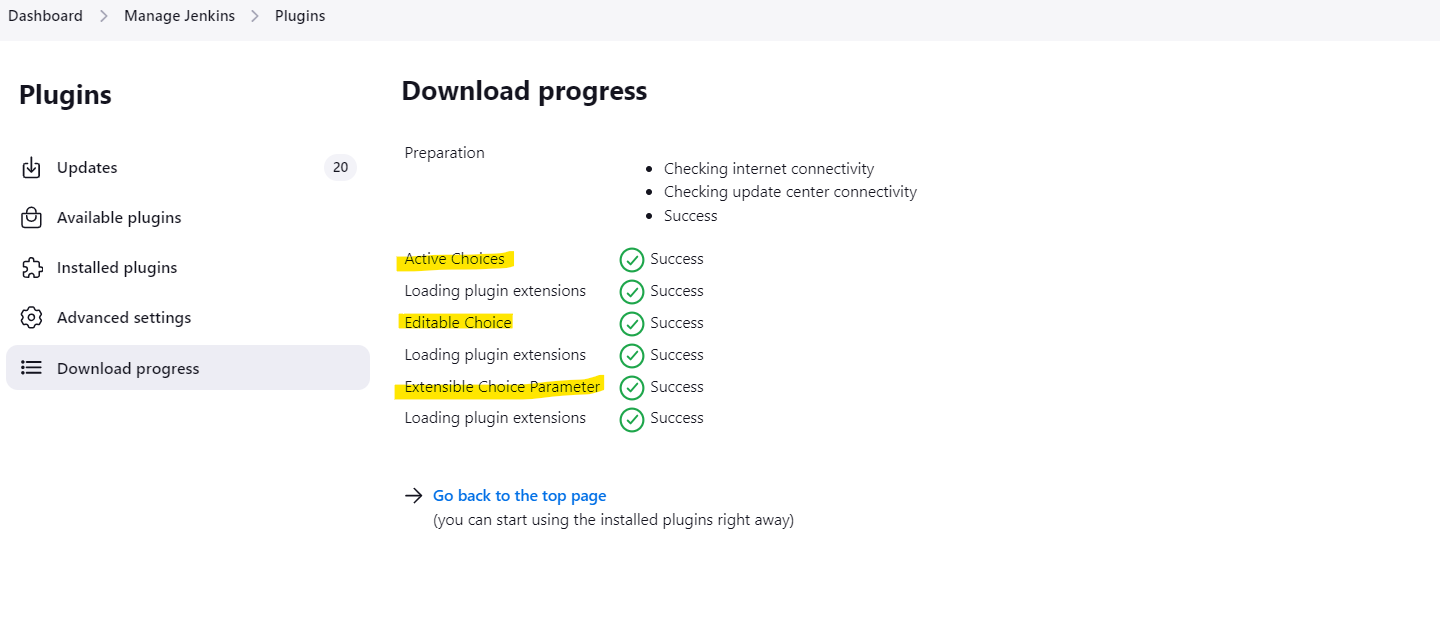


**How to use Jenkins Build Parameters Dynamically**



* Sometimes you want to render the parameters dynamically based on the value selected in any other parameter.
* Given the declarative nature of Jenkins jobs, we cannot achieve the use-case with the native Jenkins parameters available.
* Here comes the [Active Choices parameter plugin](https://plugins.jenkins.io/uno-choice/) to the rescue, which can help us render parameters/parameter’s value dynamically.

**Active Choice Plugins required:**



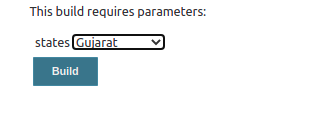
Active Choices parameter plugin provides 3 types of parameters.

1. Active Choices Parameter
2. Active Choices Reactive Parameter
3. Active Choices Reactive Reference Parameter

**Active Choices Parameter**

* An Active Choices parameter dynamically generates a list of value options for a build parameter

Example: Listing all the state names in a drop-down menu (based on parameter STATE).

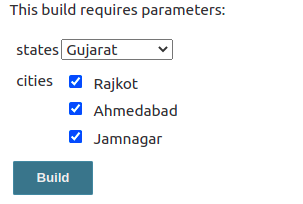


**Active Choice Reactive Parameter**

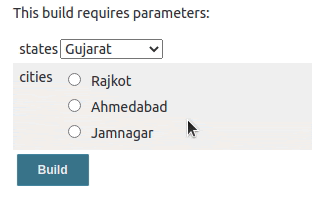
* An Active Choices Reactive Parameter can generate the same set of value options as of Active choice parameter
* in addition to that, it can be dynamically updated with the values based on the selection in some other build parameter

Example: Selecting a City name based on the state name

Example A) Select many cities based on State



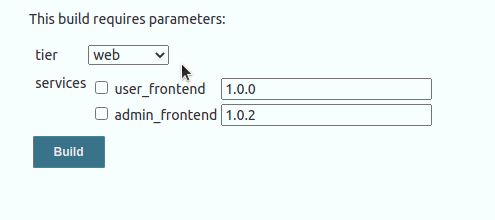
Example B) Select only one city under a State



**Active Choice Reactive Reference Parameter**

* This parameter provides the same features as the Active Choices Reactive Reference parameters.
* In addition to that, it can enhance the Job UI form with custom HTML controls like list, images, text boxes, etc.

Example: Adding additional text box in the side in HTML format to enter release details of the service



**Example Job - Parameter Build – Active choice Reactive Parameter**

**// Pipeline code for Build with Parameter -- Active choice reactive parameter //**

pipeline {

agent any

stages {

stage('Parameters'){

steps {

script {

properties([

parameters([

[$class: 'ChoiceParameter',

choiceType: 'PT\_SINGLE\_SELECT',

description: 'Select the Environemnt from the Dropdown List',

filterLength: 1,

filterable: false,

name: 'Env',

script: [

$class: 'GroovyScript',

fallbackScript: [

classpath: [],

sandbox: false,

script:

"return['Could not get The environemnts']"

],

script: [

classpath: [],

sandbox: false,

script:

"return['dev','stage','prod']"

]

]

],

[$class: 'CascadeChoiceParameter',

choiceType: 'PT\_SINGLE\_SELECT',

description: 'Select the AMI from the Dropdown List',

name: 'AMI List',

referencedParameters: 'Env',

script:

[$class: 'GroovyScript',

fallbackScript: [

classpath: [],

sandbox: false,

script: "return['Could not get Environment from Env Param']"

],

script: [

classpath: [],

sandbox: false,

script: '''

if (Env.equals("dev")){

return["ami-sd2345sd", "ami-asdf245sdf", "ami-asdf3245sd"]

}

else if(Env.equals("stage")){

return["ami-sd34sdf", "ami-sdf345sdc", "ami-sdf34sdf"]

}

else if(Env.equals("prod")){

return["ami-sdf34sdf", "ami-sdf34ds", "ami-sdf3sf3"]

}

'''

]

]

],

[$class: 'DynamicReferenceParameter',

choiceType: 'ET\_ORDERED\_LIST',

description: 'Select the AMI based on the following information',

name: 'Image Information',

referencedParameters: 'Env',

script:

[$class: 'GroovyScript',

script: 'return["Could not get AMi Information"]',

script: [

script: '''

if (Env.equals("dev")){

return["ami-sd2345sd: AMI with Java", "ami-asdf245sdf: AMI with Python", "ami-asdf3245sd: AMI with Groovy"]

}

else if(Env.equals("stage")){

return["ami-sd34sdf: AMI with Java", "ami-sdf345sdc: AMI with Python", "ami-sdf34sdf: AMI with Groovy"]

}

else if(Env.equals("prod")){

return["ami-sdf34sdf: AMI with Java", "ami-sdf34ds: AMI with Python", "ami-sdf3sf3: AMI with Groovy"]

}

'''

]

]

]

])

])

}

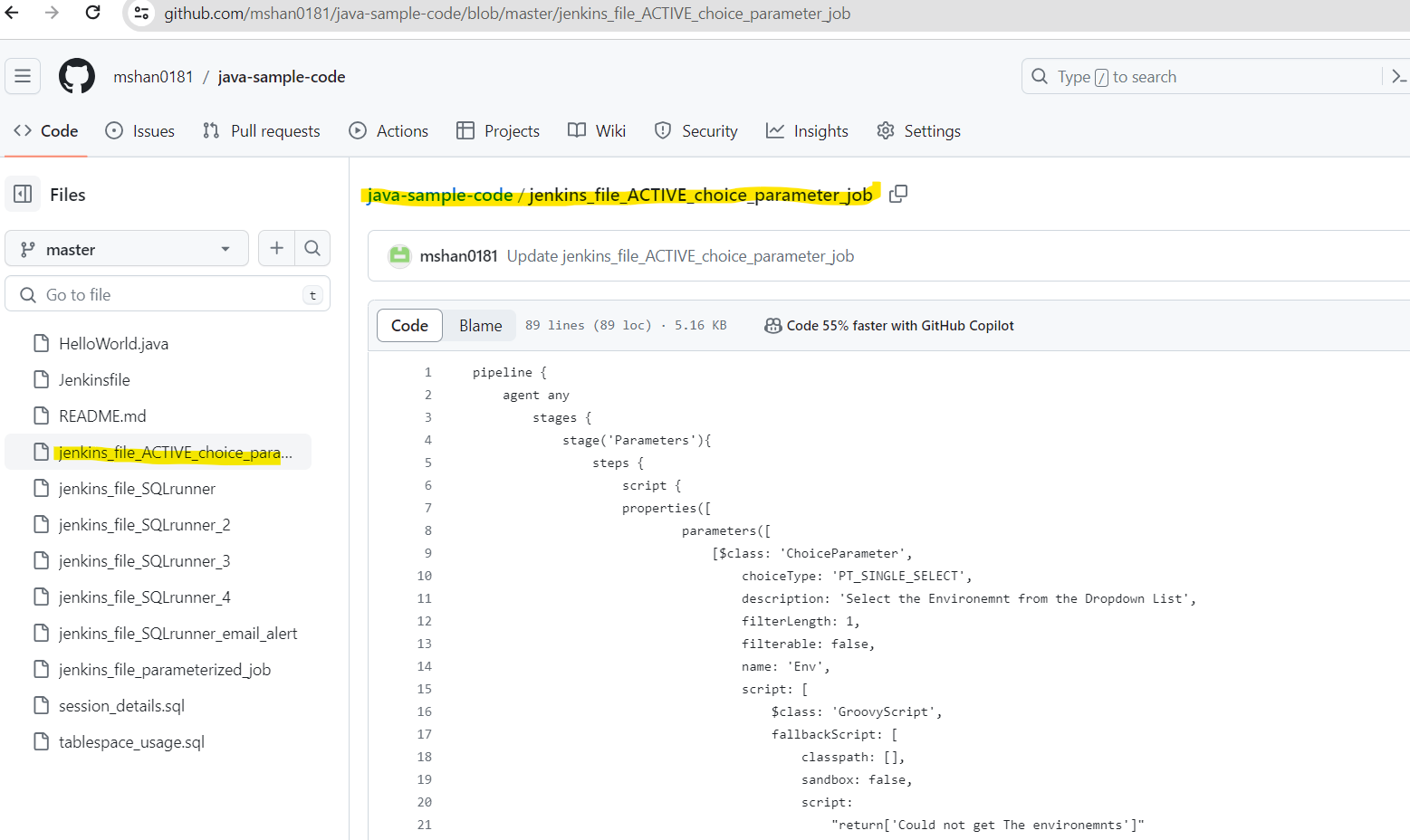
}

}

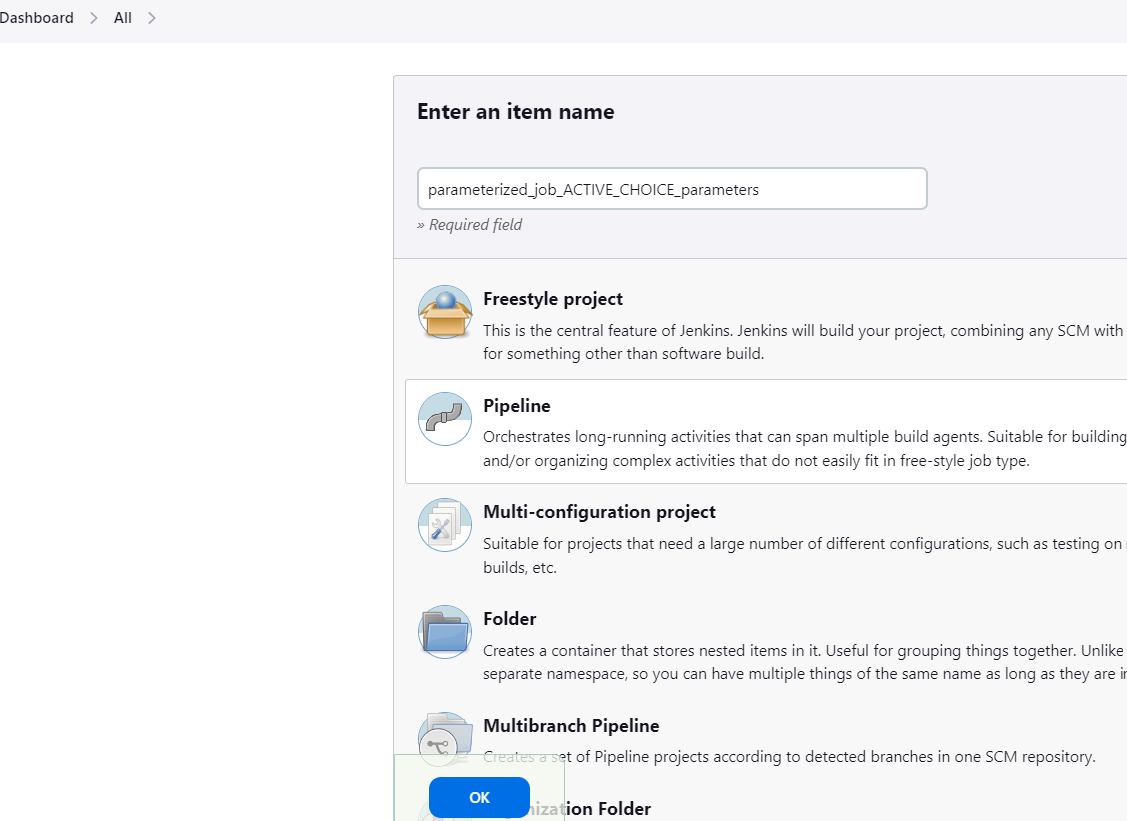
}

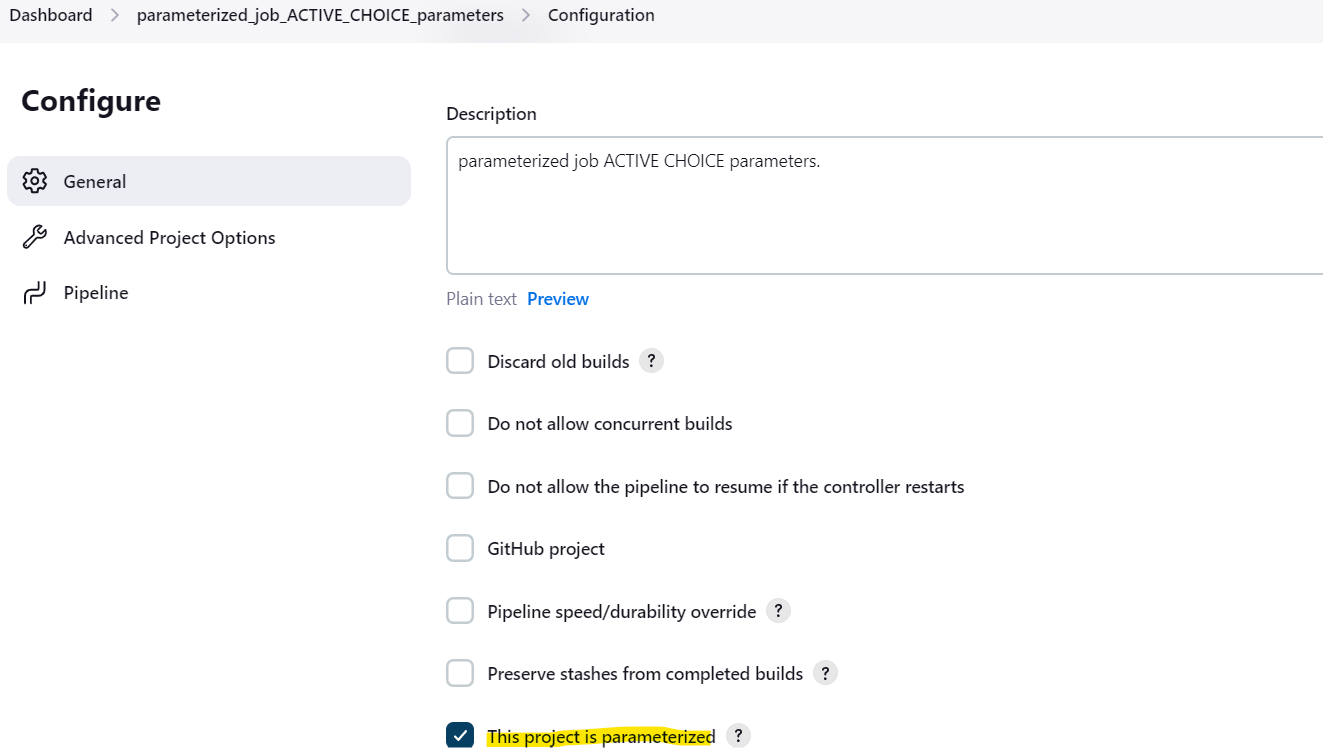
}

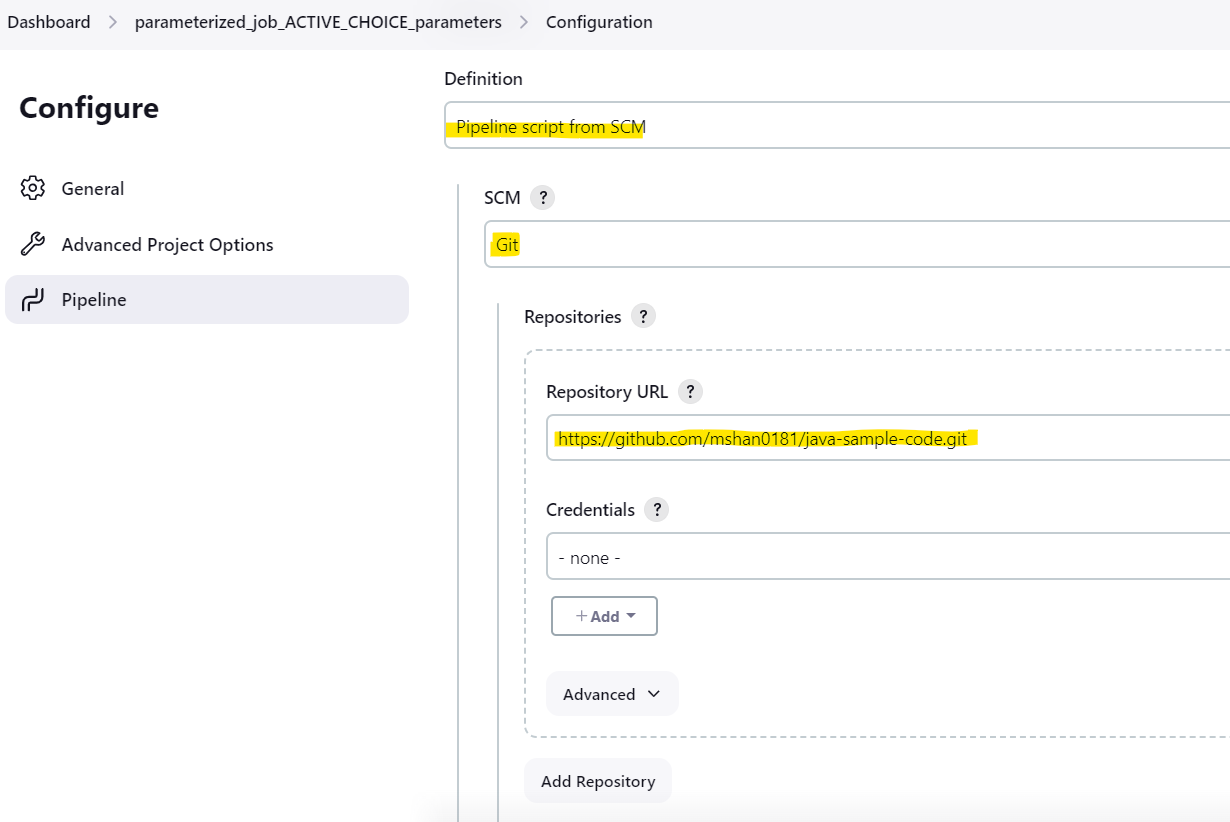
Create a Jenkins file – code for Active choice Reactive parameter

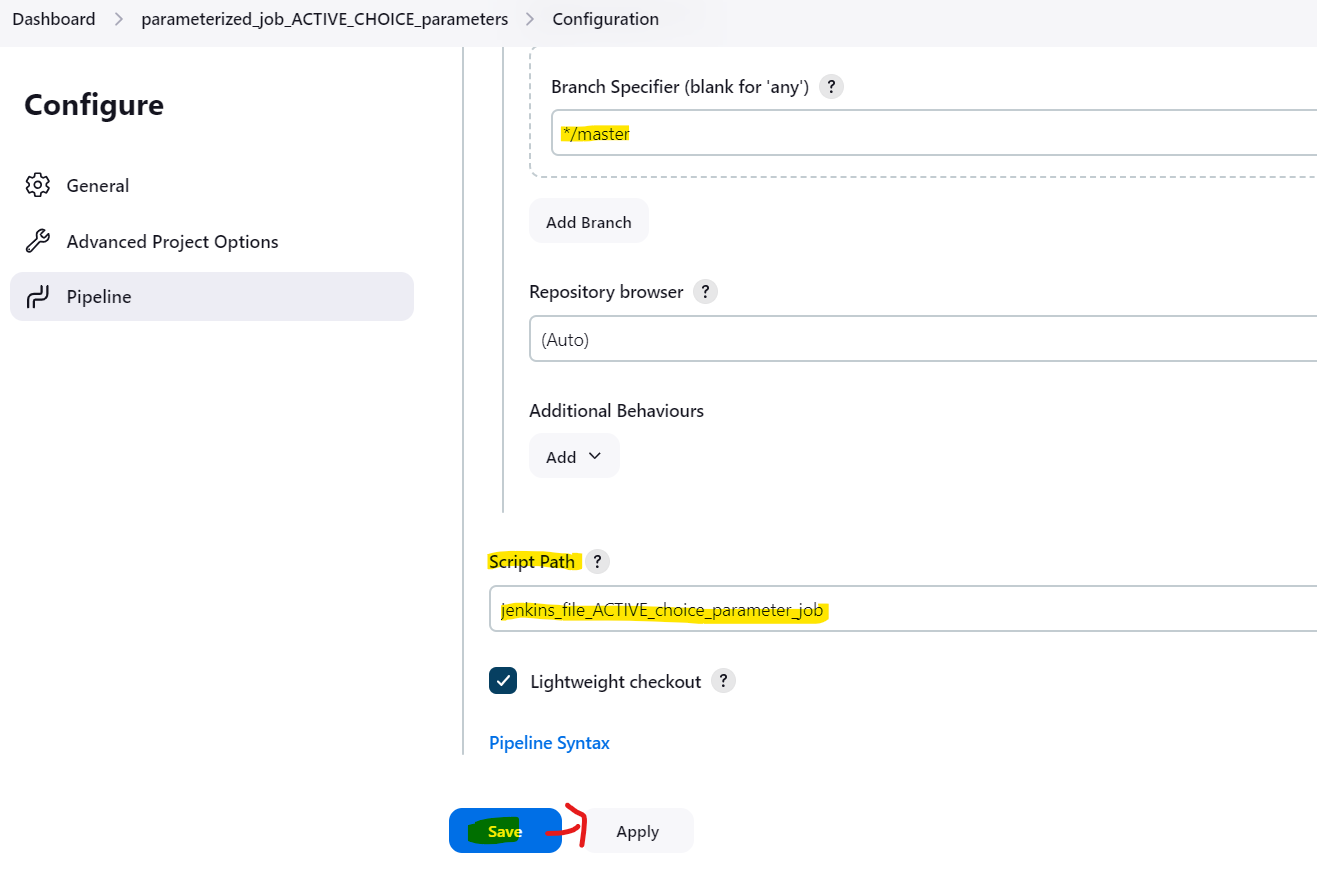


Create a new job – for Active choice Reactive parameter

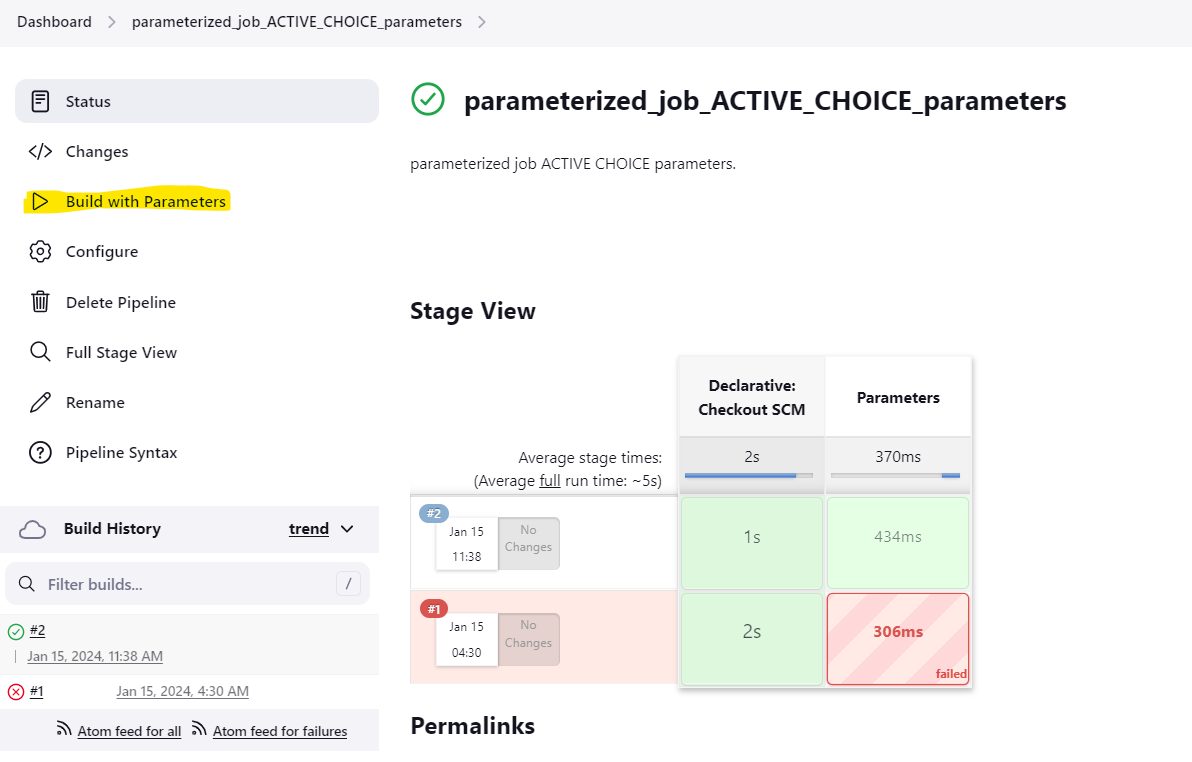




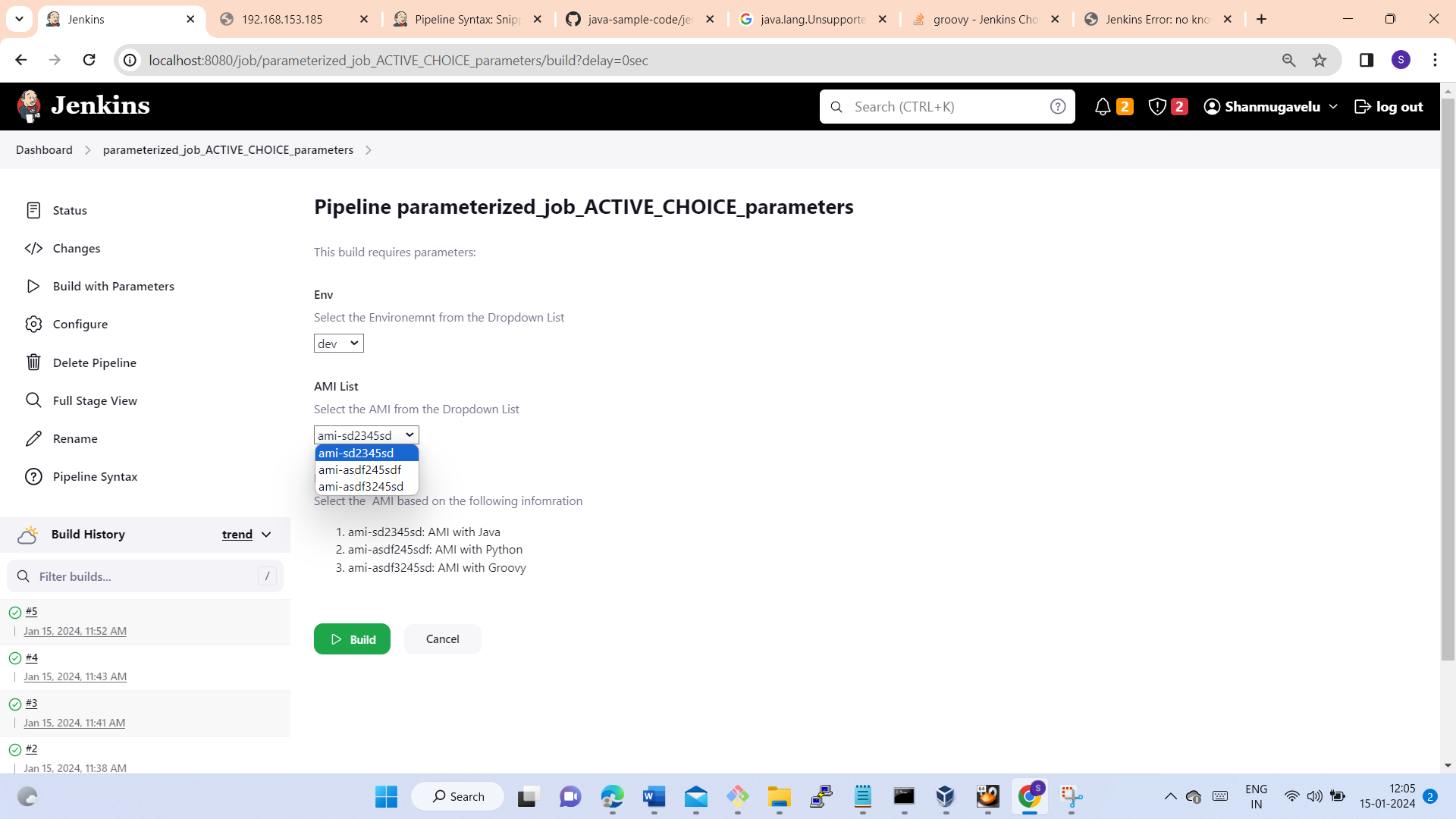




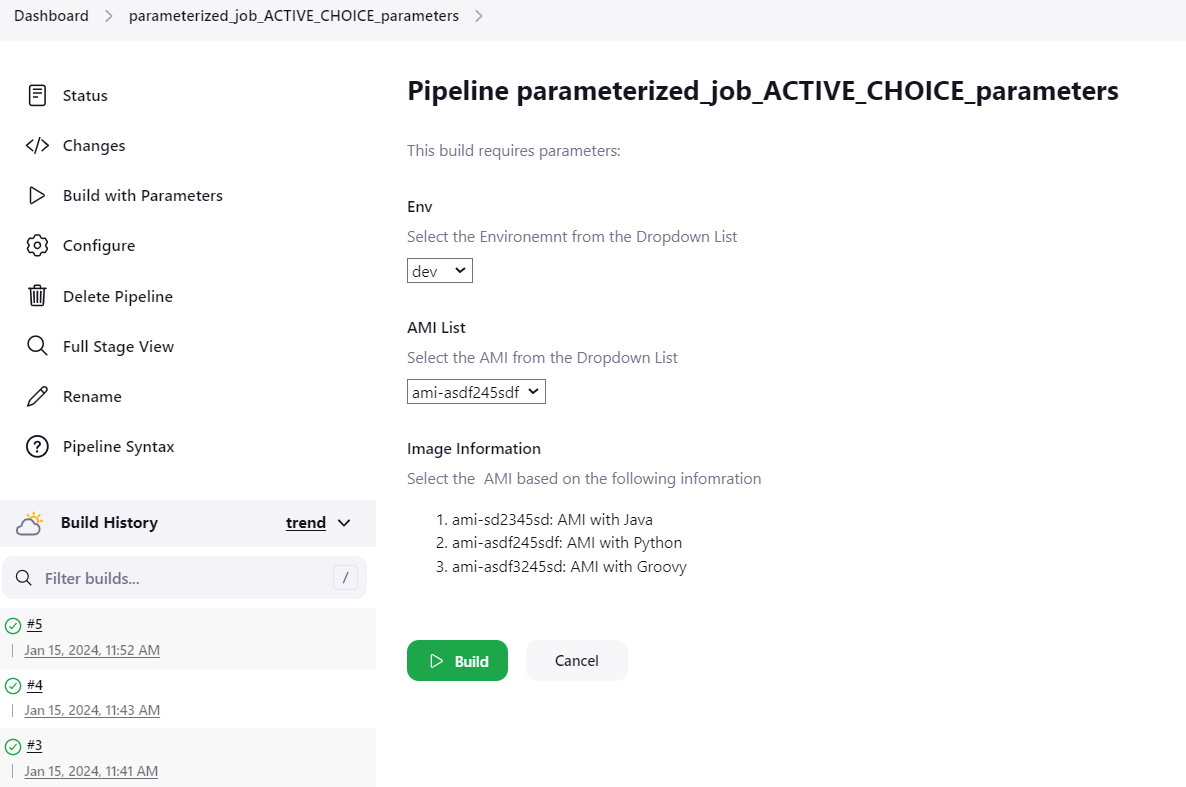
Click the Build with Parameter option: from the second RUN



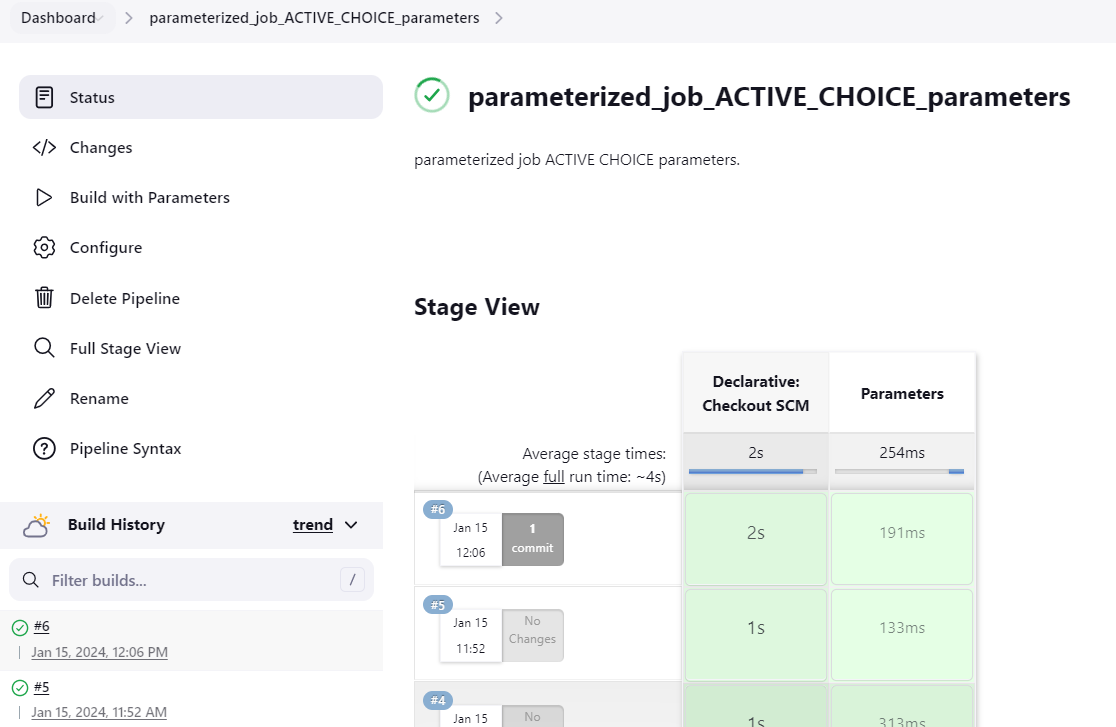
Check the drop down of the second parameter – based on first parameter selection



Select the second parameter (from the drop down) and click BUILD button



Check for the execution of the Job : for successful completion



Check the console output of the job :



**Jenkinsfile Parameter Best Practices**

### The following are some of the best practices you can follow while using parameters in a Jenkinsfile.

### 1)Never pass passwords in the String or Multi-line parameter block. Instead, use the password parameter of access Jenkins credentials with credential id as the parameter.

### 2)Try to use parameters only if required. Alternatively, you can use a config management tool to read configs or parameters in the runtime.

### 3) Handle the wrong parameter execution in the stages with a proper exception handling. It avoids unwanted step execution when a wrong parameter is provided. It happens typically in multi-line and string parameters

**Jenkinsfile Parameter FAQs**

### 1)How to dynamically populate the choice parameter in the declarative pipeline?

### Dynamic parameters can be achieved by using an active choice parameter. It uses a groovy script to dynamically populate choice parameter values.

### 2)How are the parameters used in the declarative pipeline?

### In the declarative pipeline, parameters can be incorporated using the properties block. It supports all types of Jenkins parameters.

### 3)How to generate pipeline code for parameters?

### You can use the native Jenkins pipeline syntax generator to generate the code block for any type of pipeline parameter.

## Written by Vinod Sairam

The Database Administrator who helps to manage Enterprise Databases who develops Devops Engineering ideas to support Database tasks, Infrastructure tasks and to reach the right audience.

**Assisted by Shanmugavelu**