Jenkins

Jeeva S. Chelladhurai
CEO, Comorin Consulting Services
+91 97319 77222
jeeva@comorin.co

11. Jenkins Pipeline / DSL

- Introduction
- Prerequisites
- Why use Jenkins Pipeline?
- Jenkins Pipeline Advantages
- What is a Jenkinsfile?
- Pipeline Concepts
  - Pipeline | Node | Agents | Stages | Steps
- Creating Jenkins Pipeline
- Directives in Pipeline

Introduction

Jenkins Pipeline

- A suite of plugins which supports implementing & integrating continuous delivery pipelines into Jenkins
- Provides an extensible set of tools for modelling simple-to-complex delivery pipelines "as code" via the Pipeline DSL

Prerequisites

- Jenkins 2.x or later
  - (older versions back to 1.642.3 may work but not recommended)
- Pipeline plugin
  - installed as a part of “suggested plugins” during Jenkins post installation

Why Use Jenkin's Pipeline?

- A collection of jobs that brings the software from version control into the hands of the end users by using automation tools
- A feature used to **incorporate continuous delivery** in our software development workflow

Key Features

- To define the entire deployment flow through code
- Meaning all the standard jobs defined by Jenkins are manually written as one whole script & they can be stored in a version control system
- Follows the '**pipeline as code**' discipline
- Instead of building several jobs for each phase, you can now code the entire workflow & put it in a ***Jenkinsfile***

Jenkins Pipeline Advantage

- 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
- Durable in terms of unplanned restart of the Jenkins master
- Can restart from saved **checkpoints**
- Supports complex pipelines by incorporating conditional loops, fork or join operations & allowing tasks to be performed in parallel
- Can integrate with several other plugins

What is a Jenkinsfile?

- A text file that stores the entire workflow as code
- Can be checked into a SCM on your local system

How is this advantageous?

- Enables the developers to **access, edit and check the code at all times**
- Written using the Groovy DSL
- Can be created through a text/groovy editor or through the configuration page on the Jenkins instance

Based on two Syntaxes

Scripted pipeline syntax

- Traditional way of writing the code
- ***Jenkinsfile*** is written on the **Jenkins UI instance**
- Uses stricter groovy based syntaxes as it was the first pipeline to be built on the groovy foundation
- Defined within a ‘**node**’
- Both are based on the **groovy DSL**
- Since this Groovy script was not typically desirable to all the users, the **declarative pipeline** introduced to offer a simpler & more optioned Groovy syntax

Declarative pipeline syntax

- A relatively new feature that supports the pipeline as code concept
- Makes the pipeline code easier to read and write
- Written in a ***Jenkinsfile*** that can be checked into a **SCM** system such as Git
- Defined within a block labelled ‘pipeline’

Pipeline Concepts

Pipeline

- User defined block that contains all the processes such as build, test, deploy, etc.
- Collection of all the stages in a ***Jenkinsfile***
- All the stages & steps are defined within this block.
- Key block for a declarative pipeline syntax.

Example:

```
Pipeline {

}
```

Cont'd...

Node:

- Machine that executes an entire workflow
- Key part of the scripted pipeline syntax

Example:

```
Node {

}
```

Cont'd...

Agent:

- Directive that run multiple builds with only one instance of Jenkins
- Instructs Jenkins to **allocate an executor** for the builds
- A single agent can be specified for an entire pipeline
- Few of the parameters used with agents are:
  - **Any** - Runs the pipeline/ stage on any available agent
  - **None** - Applied at the root of the pipeline, indicates no global agent for the entire pipeline
  - **Label -** Executes the pipeline/stage on the labelled agent
  - **Docker –** Uses docker container as an execution environment for the pipeline or a specific stage

In the example we are using docker to pull an ubuntu image that can be used as an execution environment to run multiple commands

Example:

```
pipeline {
        agent {
          docker {
            image 'busybox'
          }
        }
    }
```

Cont’d…

Stages:

- This block contains all the work that needs to be carried out
- The work is specified in the form of stages
- There can be more than one stage within this directive
- Each stage performs a specific task
- In the example, created multiple stages, each performing a specific task

```
Ex:   pipeline {
        agent any
        stages {
          stage (“build”) {
            …
          }

          stage (“test”) {
            …
          }

          stage (“deploy”) {
            …
          }
        }
      }
```

Cont'd…

Steps:

- A series of steps can be defined within a stage block
- Carried out in sequence to execute a stage
- Must be at least one step within steps directive
- In the example Implemented an echo command within the build stage
- This command is executed as a part of the 'Build' stage

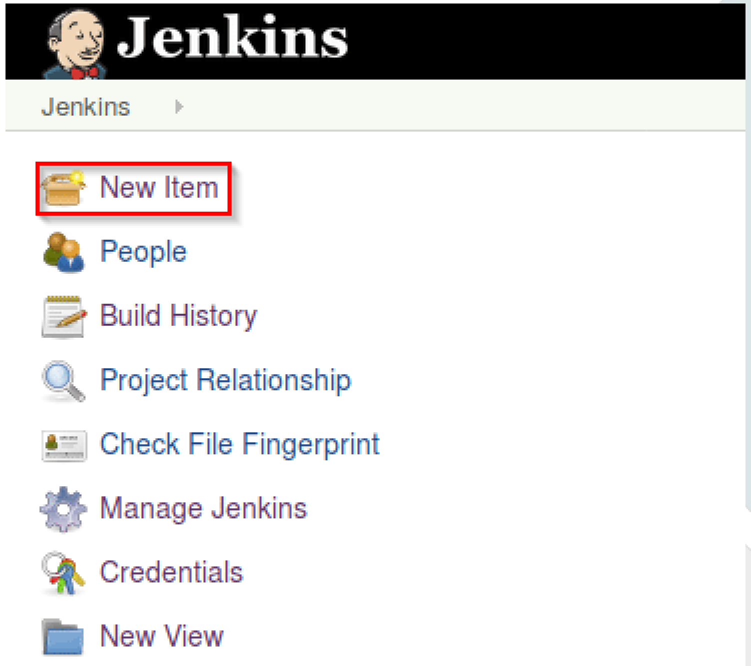
Example:

```
pipeline {
    agent any
    stages {
        stage ("Build") {
            stage {
                echo "Running build
phase..."
            }
        }
    }
}
```

Creating Jenkins Pipeline Step 1

Step 1

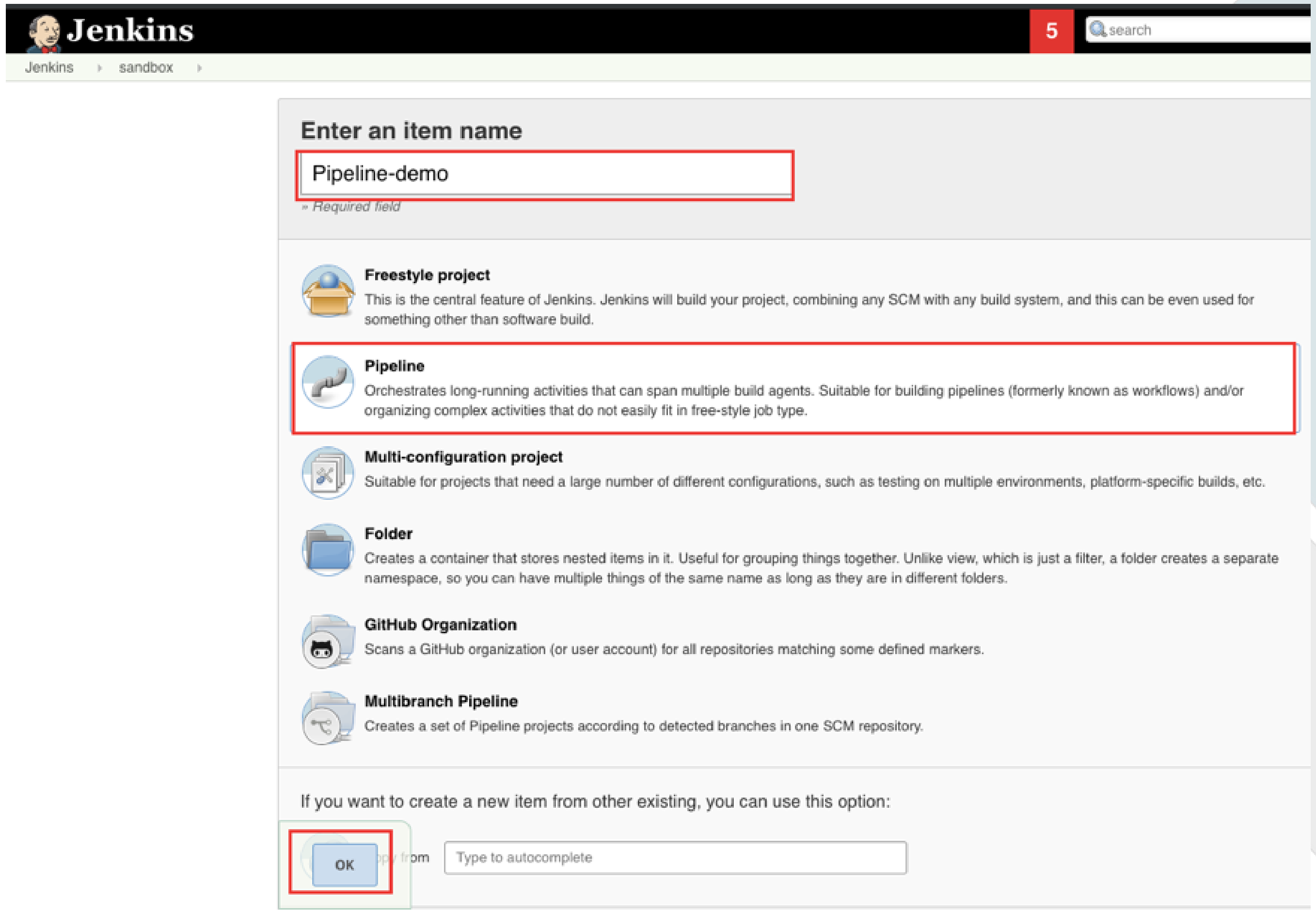
- Log into Jenkins
- Select 'New item' from the dashboard



Creating Jenkins Pipeline Step 2

Step 2

- Enter a name for your pipeline
- Select ***pipeline*** project
- Click on ***OK*** to proceed



Enter an item name

Pipeline-demo

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

GitHub Organization
Scans a GitHub organization (or user account) for all repositories matching some defined markers.

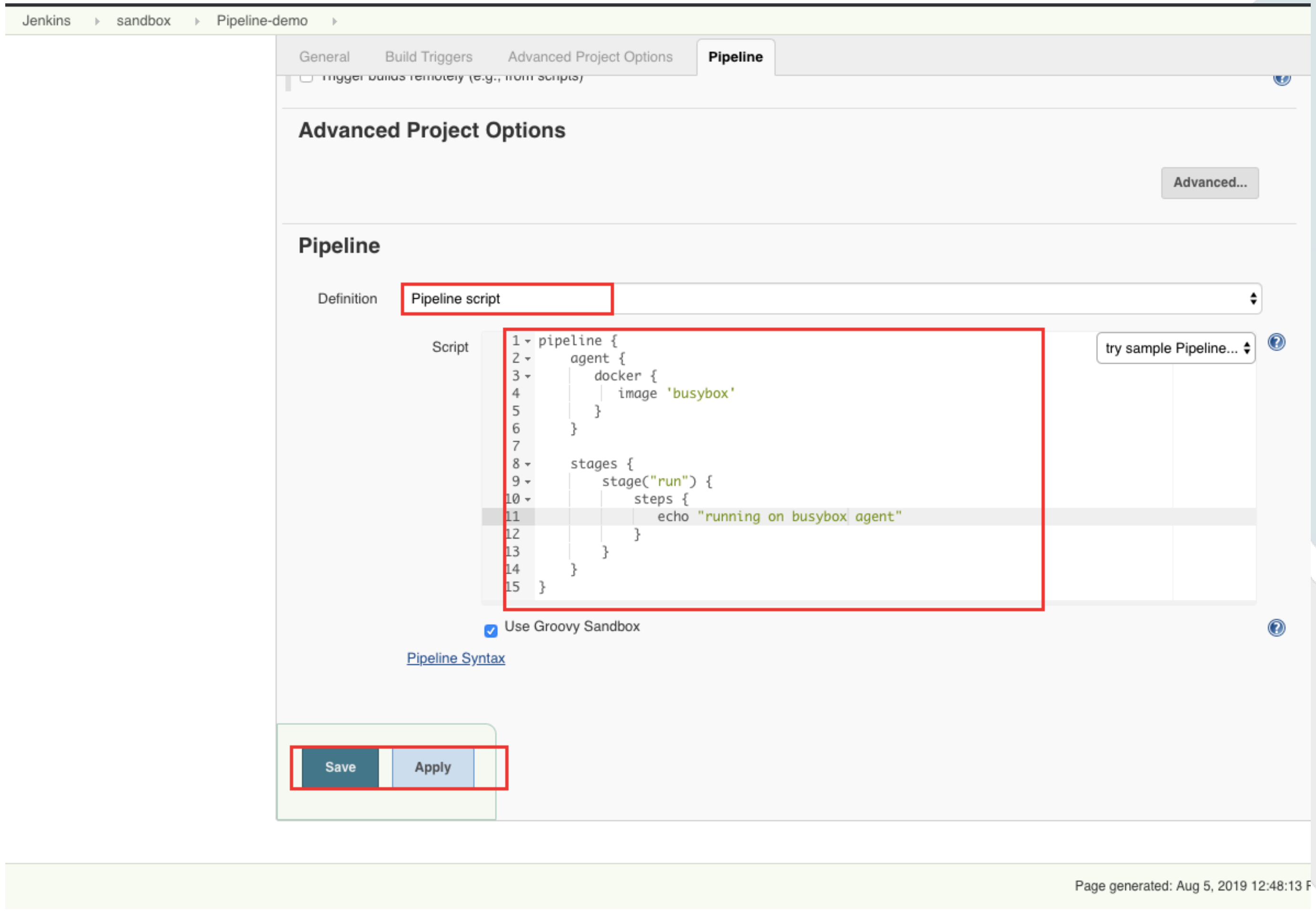
Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

If you want to create a new item from other existing, you can use this option:

Creating Jenkins Pipeline Step 3

Step 3

- Scroll down to the pipeline
- Choose if you want a declarative pipeline or a scripted one



Advanced Project Options

Pipeline

Definition: Pipeline script

Script:

```
pipeline {
    agent {
        docker {
            image 'busybox'
        }
    }

    stages {
        stage("run") {
            steps {
                echo "running on busybox agent"
            }
        }
    }
}
```

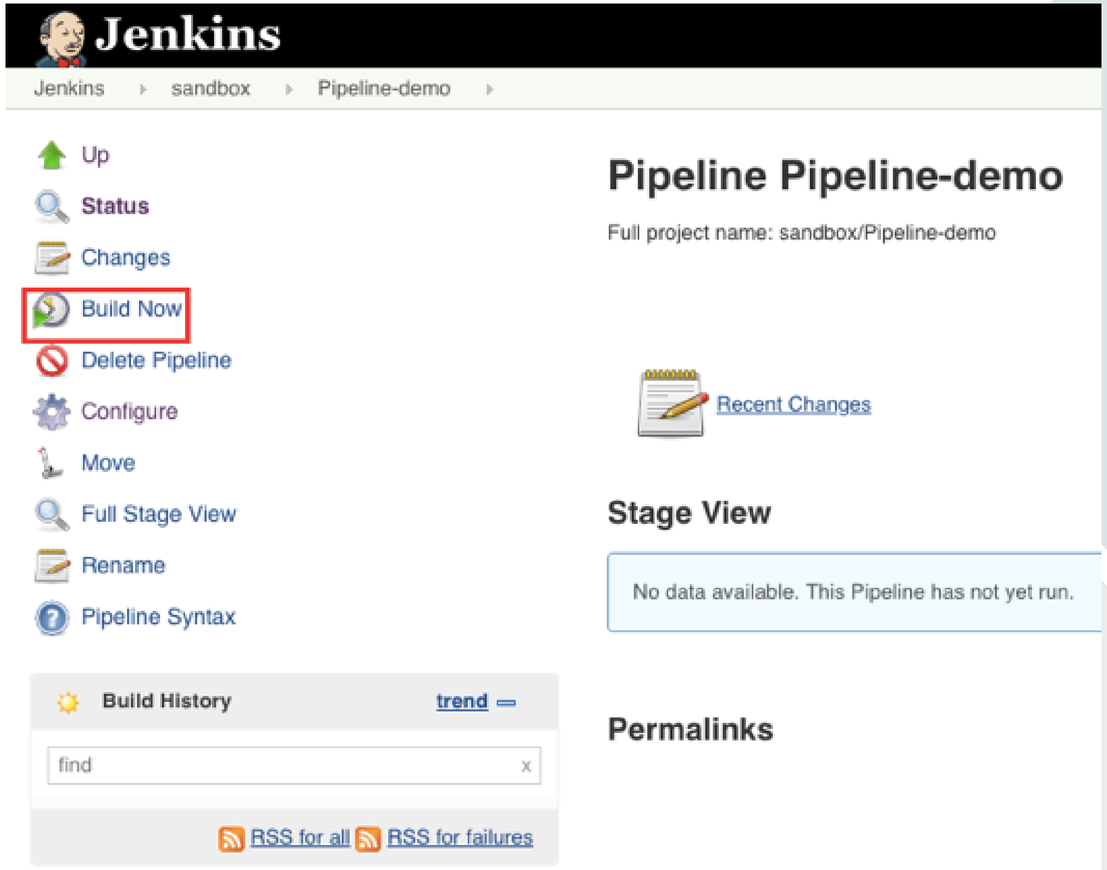
☑ Use Groovy Sandbox

Page generated: Aug 5, 2019 12:48:13 F

Creating Jenkins Pipeline Step 4

Step 4

- Build & go to console



Jenkins

Jenkins › sandbox › Pipeline-demo ›

Pipeline Pipeline-demo

Full project name: sandbox/Pipeline-demo

Recent Changes

Stage View

No data available. This Pipeline has not yet run.

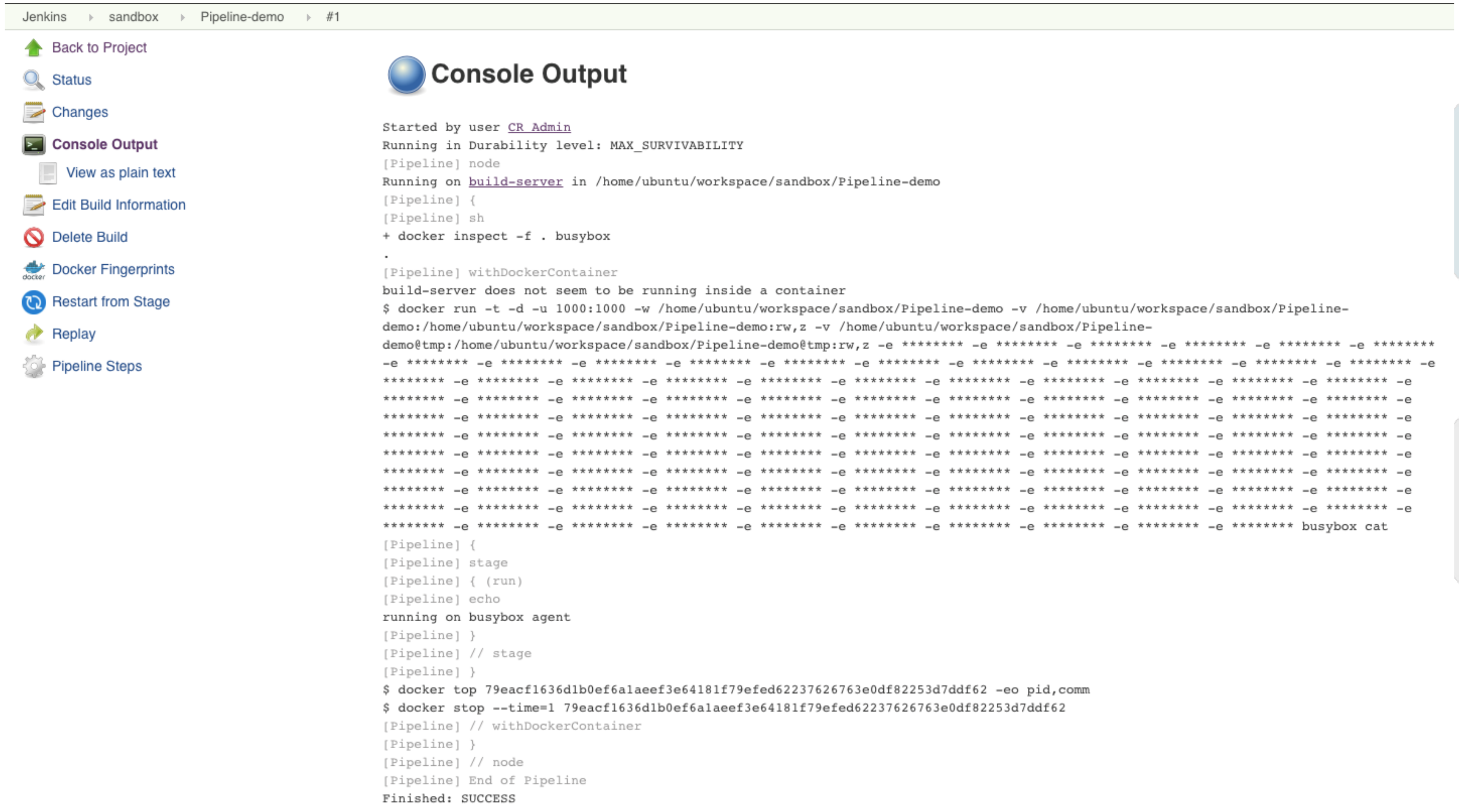
Build History trend

find

RSS for all RSS for failures

Permalinks

Console Output



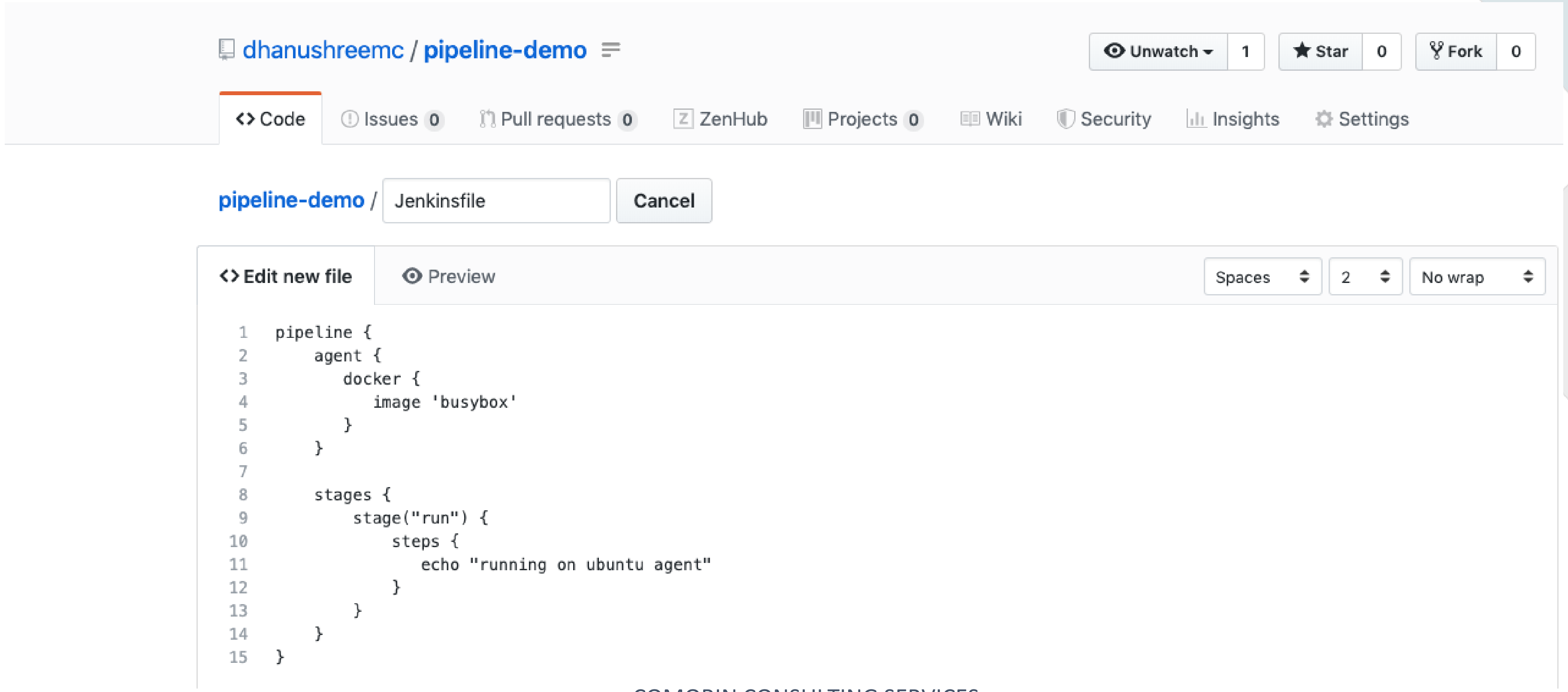
Console Output

```
Started by user CR Admin
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] node
Running on build-server in /home/ubuntu/workspace/sandbox/Pipeline-demo
[Pipeline] {
[Pipeline] sh
+ docker inspect -f . busybox
.
[Pipeline] withDockerContainer
build-server does not seem to be running inside a container
$ docker run -t -d -u 1000:1000 -w /home/ubuntu/workspace/sandbox/Pipeline-demo -v /home/ubuntu/workspace/sandbox/Pipeline-
demo:/home/ubuntu/workspace/sandbox/Pipeline-demo:rw,z -v /home/ubuntu/workspace/sandbox/Pipeline-
demo@tmp:/home/ubuntu/workspace/sandbox/Pipeline-demo@tmp:rw,z -e ******** -e ******** -e ******** -e ******** -e ******** -e ********
-e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** busybox cat
[Pipeline] {
[Pipeline] stage
[Pipeline] { (run)
[Pipeline] echo
running on busybox agent
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
$ docker top 79eacf1636d1b0ef6a1aeef3e64181f79efed62237626763e0df82253d7ddf62 -eo pid,comm
$ docker stop --time=1 79eacf1636d1b0ef6a1aeef3e64181f79efed62237626763e0df82253d7ddf62
[Pipeline] // withDockerContainer
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Example 1

Pipeline Script from SCM

- Create a GitHub repo
- Push the pipeline script(Jenkinsfile) to repo

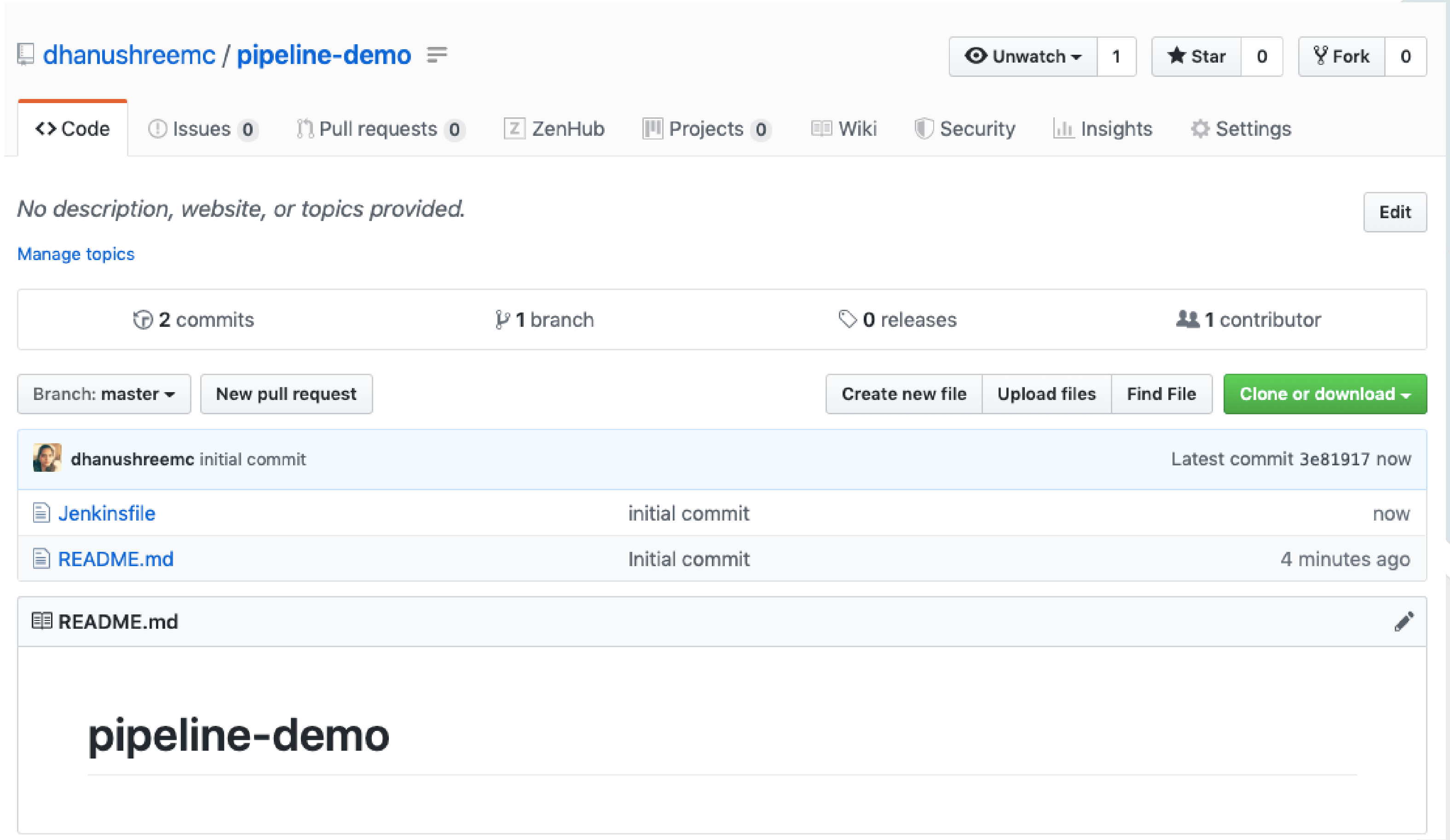


```
pipeline {
    agent {
       docker {
          image 'busybox'
       }
    }

    stages {
        stage("run") {
            steps {
               echo "running on ubuntu agent"
            }
        }
    }
}
```

Cont'd…

- Commit the file to repo by giving commit message



dhanushreemc / pipeline-demo

Unwatch 1 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | ZenHub | Projects 0 | Wiki | Security | Insights | Settings

No description, website, or topics provided. Edit

Manage topics

2 commits | 1 branch | 0 releases | 1 contributor

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

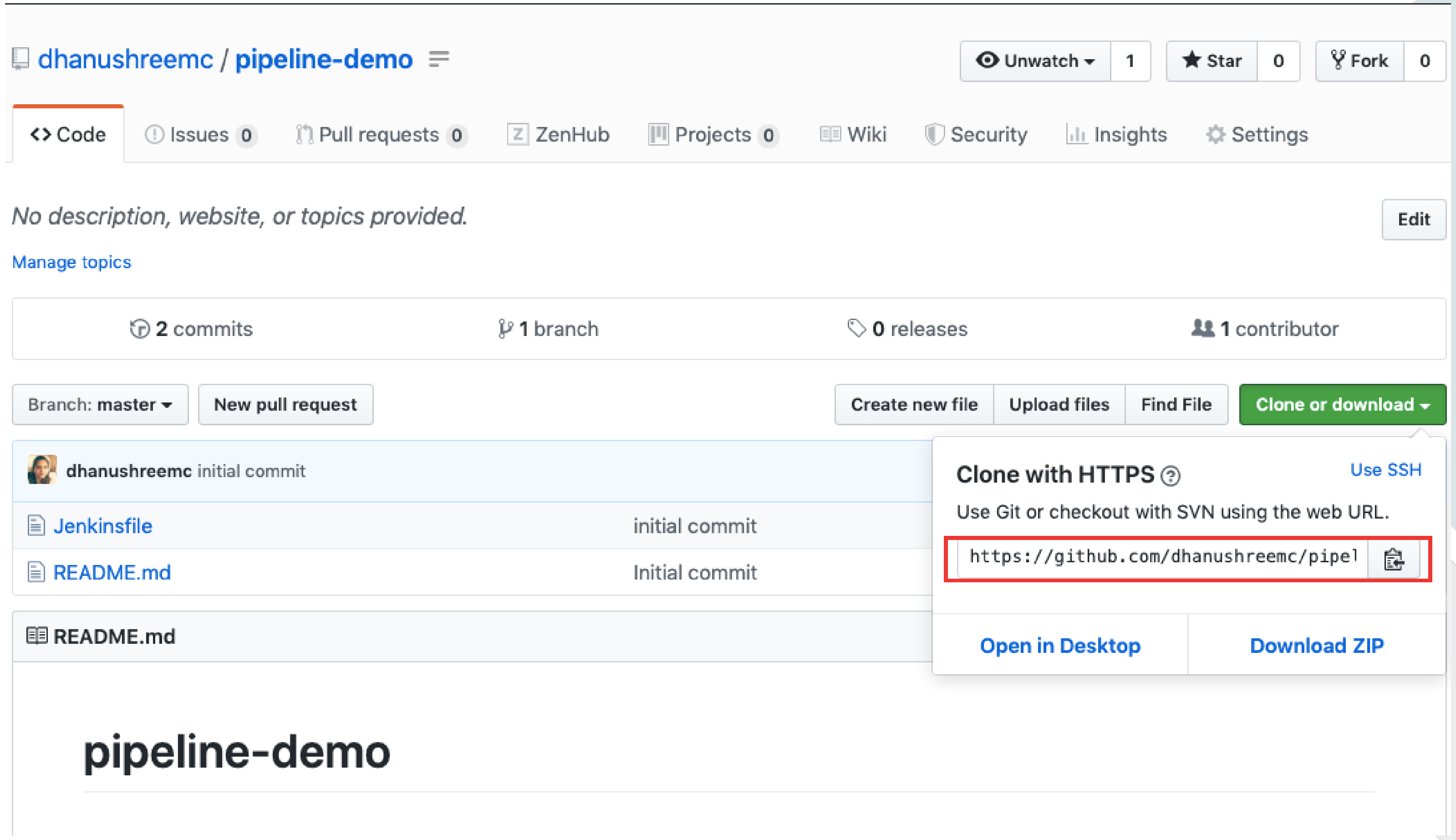
dhanushreemc initial commit		Latest commit 3e81917 now
Jenkinsfile	initial commit	now
README.md	Initial commit	4 minutes ago

README.md

pipeline-demo

Cont'd…

- Come back to same Jenkins job instead of pipeline script
- Choose pipeline script by SCM as shown



dhanushreemc / pipeline-demo

No description, website, or topics provided.

2 commits | 1 branch | 0 releases | 1 contributor

dhanushreemc initial commit

Jenkinsfile	initial commit
README.md	Initial commit

README.md

pipeline-demo

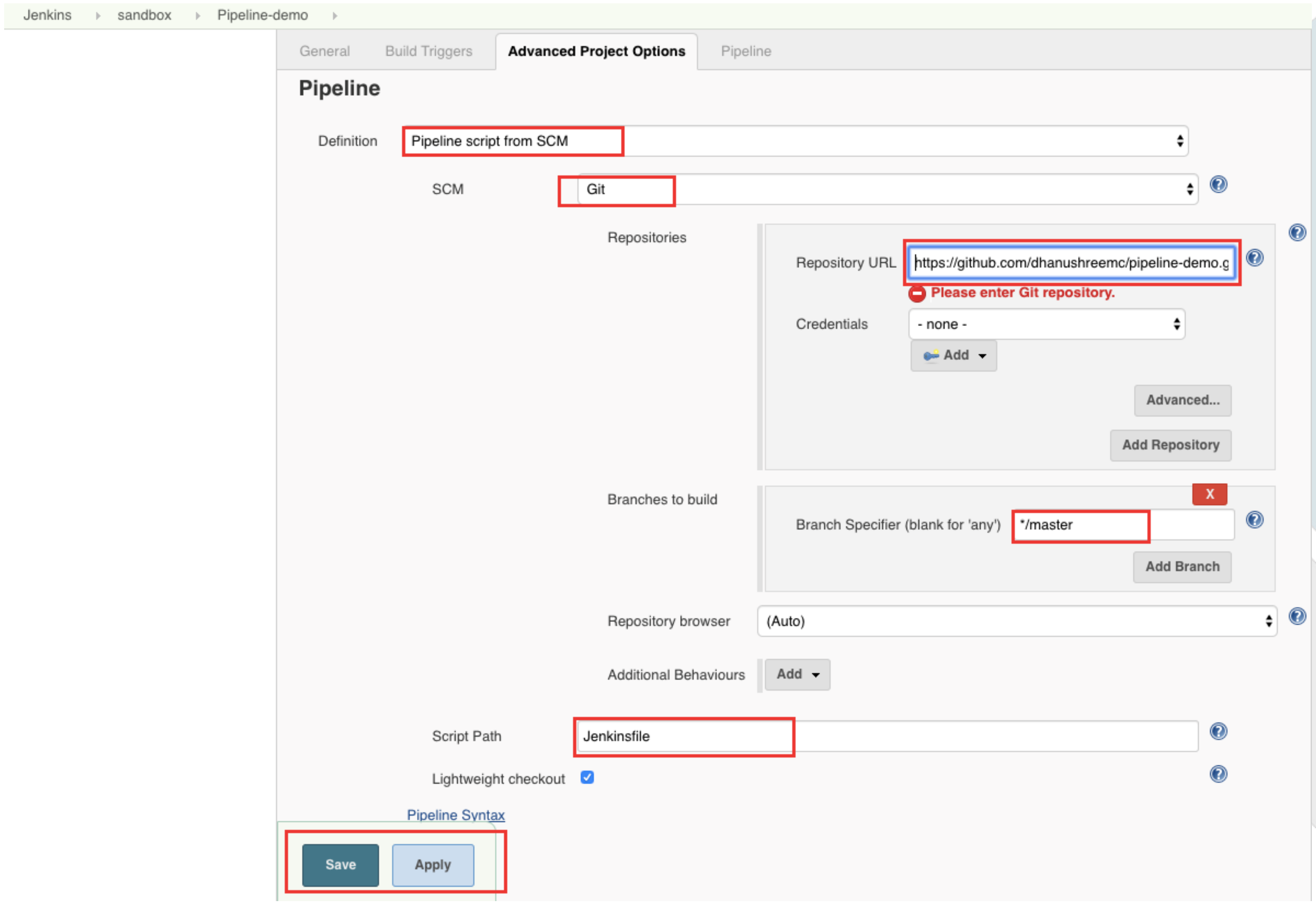
Clone with HTTPS

Use Git or checkout with SVN using the web URL.

```
https://github.com/dhanushreemc/pipel
```

Cont'd…

- Clone the GitHub URL
- Configure as shown
- Save & Apply Build



Jenkins › sandbox › Pipeline-demo ›

General | Build Triggers | Advanced Project Options | Pipeline

Pipeline

Definition: Pipeline script from SCM

SCM: Git

Repositories

Repository URL: https://github.com/dhanushreemc/pipeline-demo.g

Please enter Git repository.

Credentials: - none -

Branches to build

Branch Specifier (blank for 'any'): */master

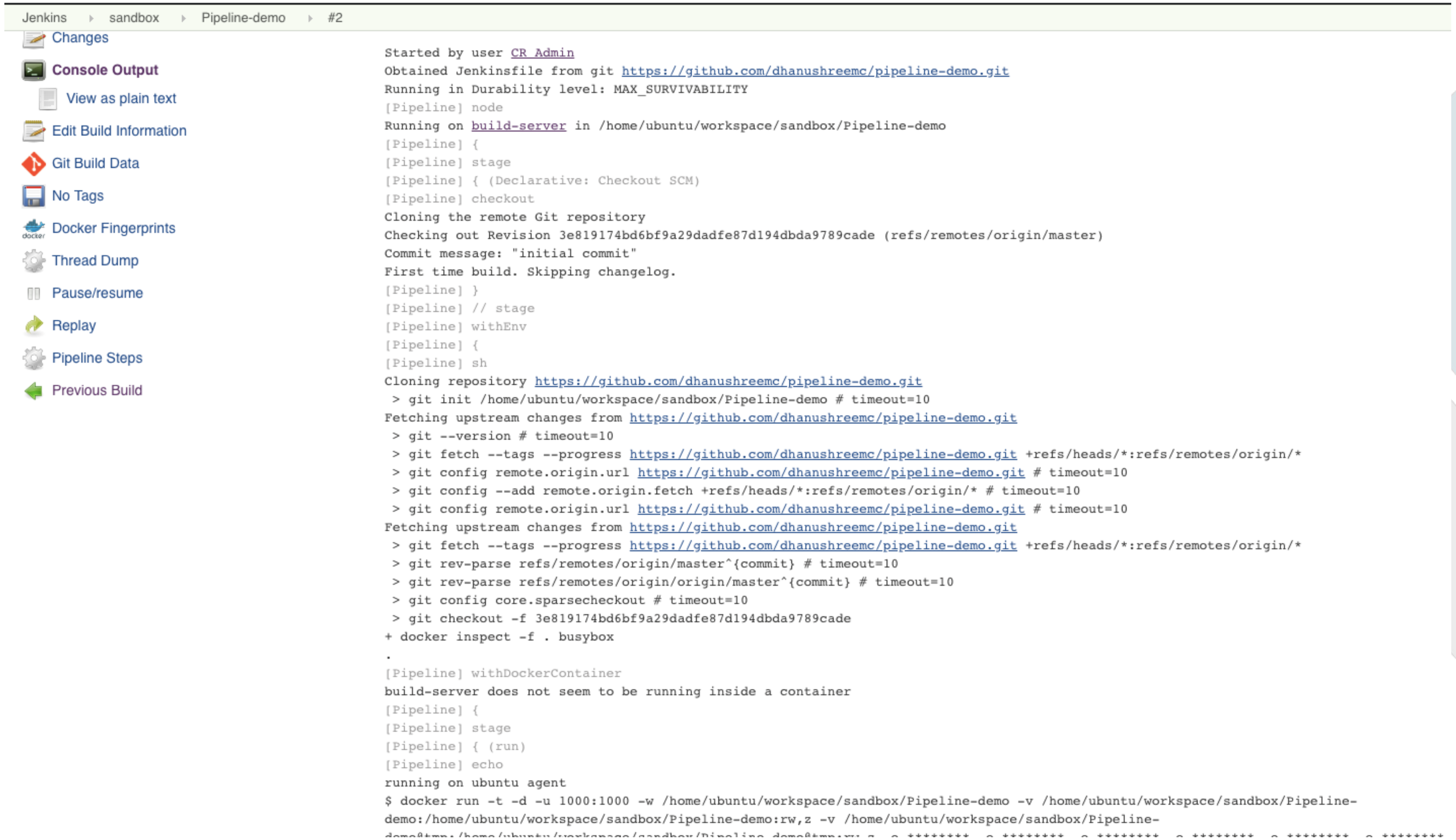
Repository browser: (Auto)

Additional Behaviours: Add

Script Path: Jenkinsfile

Lightweight checkout: ✓

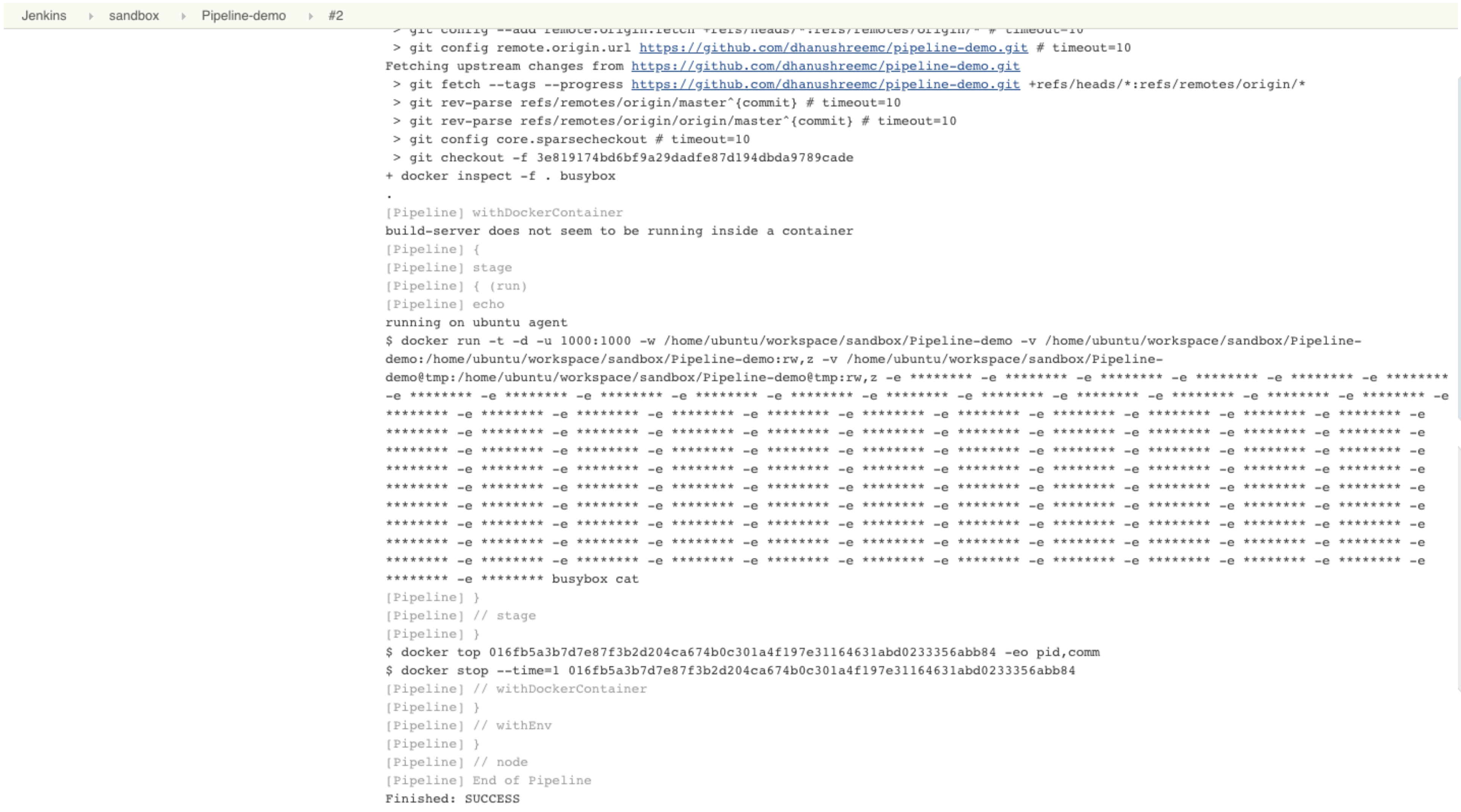
Console Output


Jenkins › sandbox › Pipeline-demo › #2

- Changes
- Console Output
  - View as plain text
- Edit Build Information
- Git Build Data
- No Tags
- Docker Fingerprints
- Thread Dump
- Pause/resume
- Replay
- Pipeline Steps
- Previous Build

```
Started by user CR Admin
Obtained Jenkinsfile from git https://github.com/dhanushreemc/pipeline-demo.git
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] node
Running on build-server in /home/ubuntu/workspace/sandbox/Pipeline-demo
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
Cloning the remote Git repository
Checking out Revision 3e819174bd6bf9a29dadfe87d194dbda9789cade (refs/remotes/origin/master)
Commit message: "initial commit"
First time build. Skipping changelog.
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] sh
Cloning repository https://github.com/dhanushreemc/pipeline-demo.git
 > git init /home/ubuntu/workspace/sandbox/Pipeline-demo # timeout=10
Fetching upstream changes from https://github.com/dhanushreemc/pipeline-demo.git
 > git --version # timeout=10
 > git fetch --tags --progress https://github.com/dhanushreemc/pipeline-demo.git +refs/heads/*:refs/remotes/origin/*
 > git config remote.origin.url https://github.com/dhanushreemc/pipeline-demo.git # timeout=10
 > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git config remote.origin.url https://github.com/dhanushreemc/pipeline-demo.git # timeout=10
Fetching upstream changes from https://github.com/dhanushreemc/pipeline-demo.git
 > git fetch --tags --progress https://github.com/dhanushreemc/pipeline-demo.git +refs/heads/*:refs/remotes/origin/*
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 3e819174bd6bf9a29dadfe87d194dbda9789cade
+ docker inspect -f . busybox
.
[Pipeline] withDockerContainer
build-server does not seem to be running inside a container
[Pipeline] {
[Pipeline] stage
[Pipeline] { (run)
[Pipeline] echo
running on ubuntu agent
$ docker run -t -d -u 1000:1000 -w /home/ubuntu/workspace/sandbox/Pipeline-demo -v /home/ubuntu/workspace/sandbox/Pipeline-
demo:/home/ubuntu/workspace/sandbox/Pipeline-demo:rw,z -v /home/ubuntu/workspace/sandbox/Pipeline-
```

Console Output



Jenkins › sandbox › Pipeline-demo › #2

```
 > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git config remote.origin.url https://github.com/dhanushreemc/pipeline-demo.git # timeout=10
Fetching upstream changes from https://github.com/dhanushreemc/pipeline-demo.git
 > git fetch --tags --progress https://github.com/dhanushreemc/pipeline-demo.git +refs/heads/*:refs/remotes/origin/*
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 3e819174bd6bf9a29dadfe87d194dbda9789cade
+ docker inspect -f . busybox
.
[Pipeline] withDockerContainer
build-server does not seem to be running inside a container
[Pipeline] {
[Pipeline] stage
[Pipeline] { (run)
[Pipeline] echo
running on ubuntu agent
$ docker run -t -d -u 1000:1000 -w /home/ubuntu/workspace/sandbox/Pipeline-demo -v /home/ubuntu/workspace/sandbox/Pipeline-
demo:/home/ubuntu/workspace/sandbox/Pipeline-demo:rw,z -v /home/ubuntu/workspace/sandbox/Pipeline-
demo@tmp:/home/ubuntu/workspace/sandbox/Pipeline-demo@tmp:rw,z -e ******** -e ******** -e ******** -e ******** -e ******** -e ********
-e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e ******** -e
******** -e ******** busybox cat
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
$ docker top 016fb5a3b7d7e87f3b2d204ca674b0c301a4f197e31164631abd0233356abb84 -eo pid,comm
$ docker stop --time=1 016fb5a3b7d7e87f3b2d204ca674b0c301a4f197e31164631abd0233356abb84
[Pipeline] // withDockerContainer
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

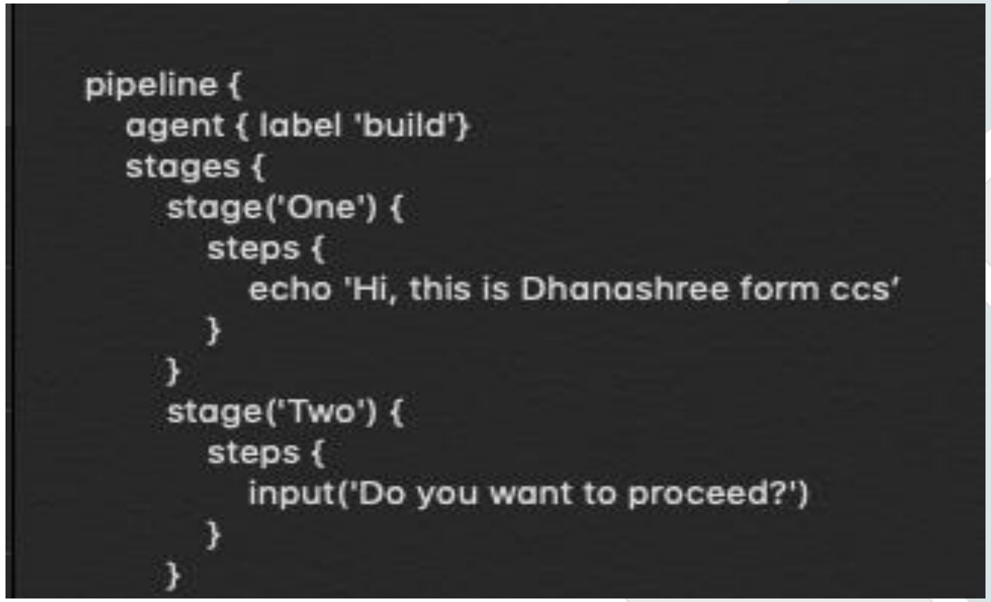
Example 2

Building a pipeline to run multiple stages, each performing a specific task

- Declarative pipeline: defined by writing the code within a pipeline block
- Within the block, an agent is defined with the tag ‘any’ meaning the pipeline runs on any available executor
- Next create four stages, each performing a simple task

Cont'd…

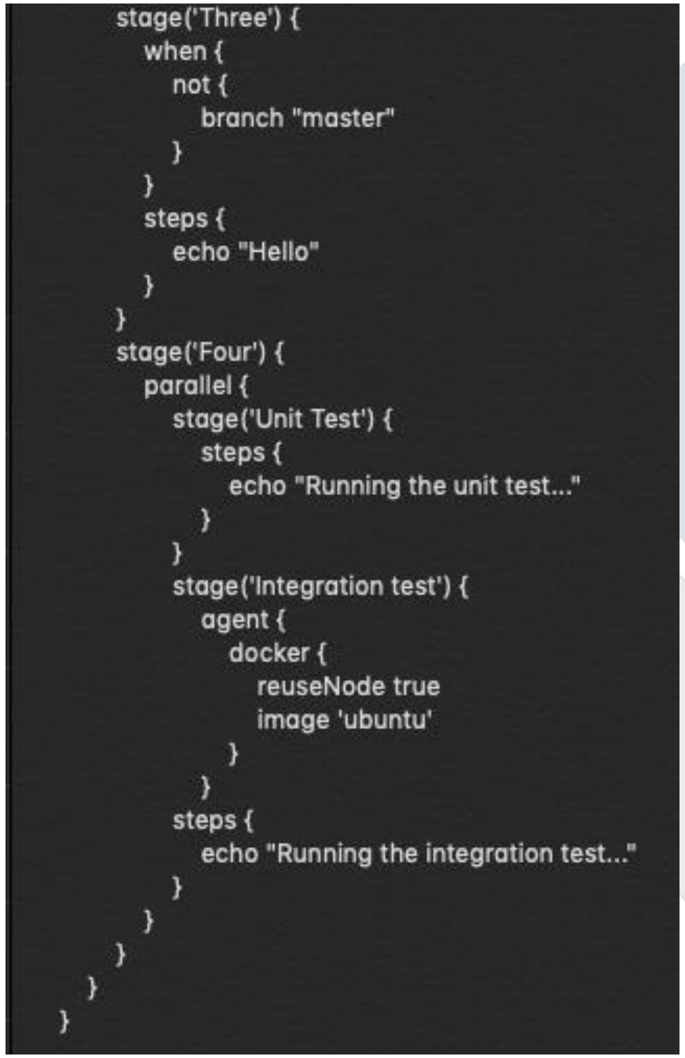
- **Stage one** executes a simple echo command which is specified within the 'steps' block
- **Stage two** executes an input directive allows to prompt a user input & displays a message
- Waits for the user input if the input is approved, then it will trigger further deployments
- In this demo a simple input message 'Do you want to proceed?' is displayed
- On receiving the user input the pipeline either proceeds with the execution or aborts



```
pipeline {
    agent { label 'build'}
    stages {
        stage('One') {
            steps {
                echo 'Hi, this is Dhanashree form ccs'
            }
        }
        stage('Two') {
            steps {
                input('Do you want to proceed?')
            }
        }
```

Cont'd…

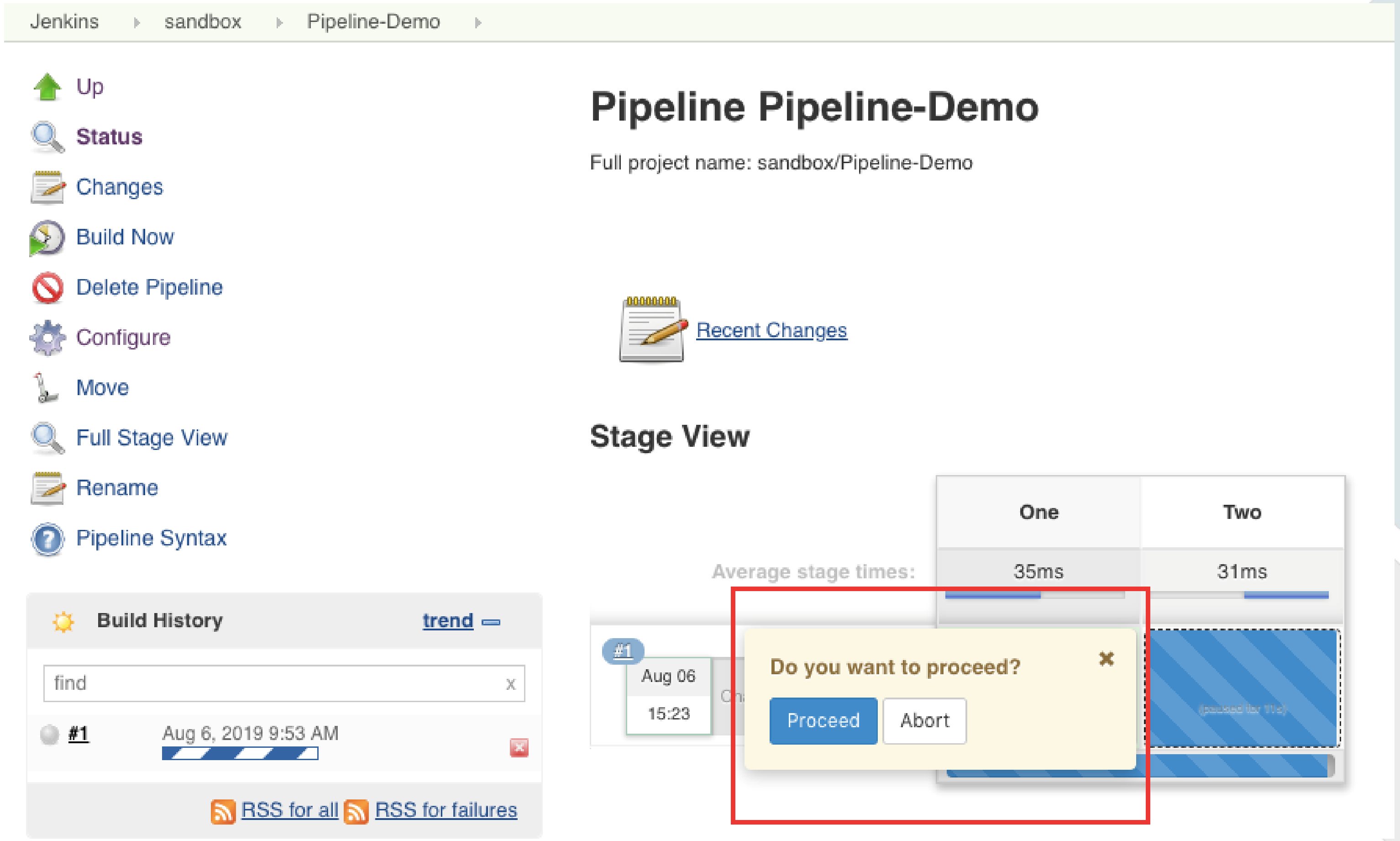
- **Stage three** runs a 'when' directive with a 'not' tag
  - Allow to execute a step depending on the **conditions defined** within the 'when' loop
  - If the conditions are met, the corresponding stage will be executed
- In this demo, used '***not***' tag.
  - executes a stage when the nested condition is **false**. Hence when the 'branch is master' holds false, the echo command in the following step is executed
- **Stage four** runs a parallel directive
  - Allows you to run nested stages in parallel
  - In the given example two nested stages in parallel, namely, 'Unit test' & 'Integration test'
  - Within the integration test stage, a stage specific docker agent is defined
  - This docker agent will execute the 'Integration test' stage



```
    stage('Three') {
      when {
        not {
          branch "master"
        }
      }
      steps {
        echo "Hello"
      }
    }
    stage('Four') {
      parallel {
        stage('Unit Test') {
          steps {
            echo "Running the unit test..."
          }
        }
        stage('Integration test') {
          agent {
            docker {
              reuseNode true
              image 'ubuntu'
            }
          }
          steps {
            echo "Running the integration test..."
          }
        }
      }
    }
  }
}
```

Running the Pipeline

- The pipeline waits for the user input and on clicking '**proceed**', the execution resumes



Jenkins › sandbox › Pipeline-Demo ›

Up
Status
Changes
Build Now
Delete Pipeline
Configure
Move
Full Stage View
Rename
Pipeline Syntax

Build History trend

find

#1 Aug 6, 2019 9:53 AM

RSS for all RSS for failures

Pipeline Pipeline-Demo

Full project name: sandbox/Pipeline-Demo

Recent Changes

Stage View

	One	Two
Average stage times:	35ms	31ms

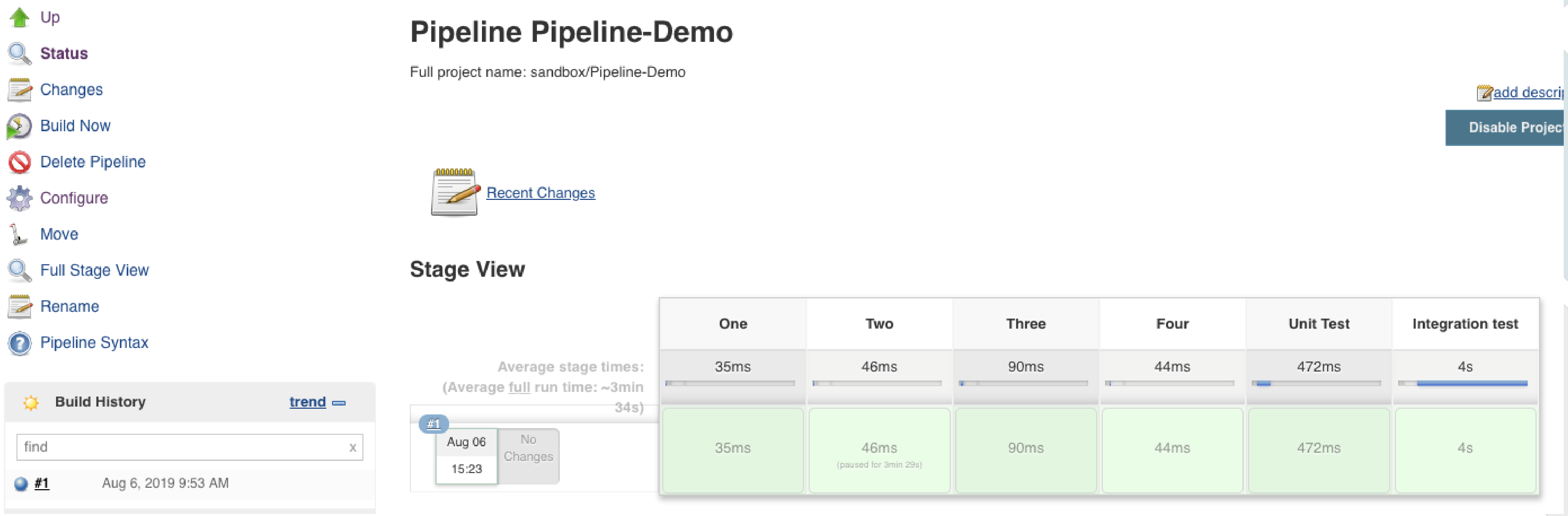
#1 Aug 06 15:23

Do you want to proceed?

Proceed Abort

Cont'd…

- After proceed is clicked, the pipeline execution proceeds further



- Up
- Status
- Changes
- Build Now
- Delete Pipeline
- Configure
- Move
- Full Stage View
- Rename
- Pipeline Syntax

Build History trend

find x

#1 Aug 6, 2019 9:53 AM

Pipeline Pipeline-Demo

Full project name: sandbox/Pipeline-Demo

add descri

Disable Projec

Recent Changes

Stage View

Average stage times:
(Average full run time: ~3min 34s)

#1 Aug 06 15:23 No Changes

	One	Two	Three	Four	Unit Test	Integration test
Average stage times	35ms	46ms	90ms	44ms	472ms	4s
#1 Aug 06 15:23	35ms	46ms (paused for 3min 29s)	90ms	44ms	472ms	4s

Cont’d…

POST:

- Defines one or more additional steps that are run upon the completion of a Pipeline’s or stage’s run
- Supports the following post conditional blocks mentioned below
    - ***always***
    - ***changed***
    - ***fixed***
    - ***regression***
    - ***failure***
    - ***success***
    - ***unstable***
    - ***unsuccessful***
    - ***cleanup***

Example:

```
pipeline {
   agent any
   stages {
      stage('Example') {
         steps {
            echo 'Hello World'
         }
      }
   }
   post {
      always {
         echo 'I will always say Hello again!'
      }
   }
}
```

Directives in Pipeline

Stages

- Containing a sequence of one or more stage directives
- The stages section is where the bulk of the "work" described by a Pipeline will be located
- At a minimum it is recommended that stages contain at least one stage directive for each discrete part of the continuous delivery process, such as Build, Test, and Deploy
- This is required in pipeline block and allowed only once inside pipeline
- Stages section typically follow the directives such as agent & options

Example:

```
pipeline {    agent any
   stages {
      stage('Example') {
         steps {
            echo 'Hello World'
         }
      }
   }
}
```

Cont'd...

- **Steps:**
- The steps section defines a series of one or more steps to be executed in a given stage directive
- Required inside each stage block

Example:

```
pipeline {    agent any
    stages {
        stage('Example') {
            steps {
                echo 'Hello World'
            }
        }
    }
}
```

Cont'd…

environment:

- The environment directive specifies a sequence of key-value pairs which will be defined as environment variables for the all steps, or stage-specific steps, depending on where the environment directive is located within the Pipeline.
- not mandatory in pipeline
- can be inside pipeline block or stage block

Example:

```
pipeline {
agent any
environment {
   CC = 'clang'
}
stages {
   stage('Example1') {
      environment {
         dd = 'test'
      }
      steps {
         sh "printenv"
      }
   }

   stage('Example2') {
      steps {
         sh "printenv"
      }
   }
}
}
```

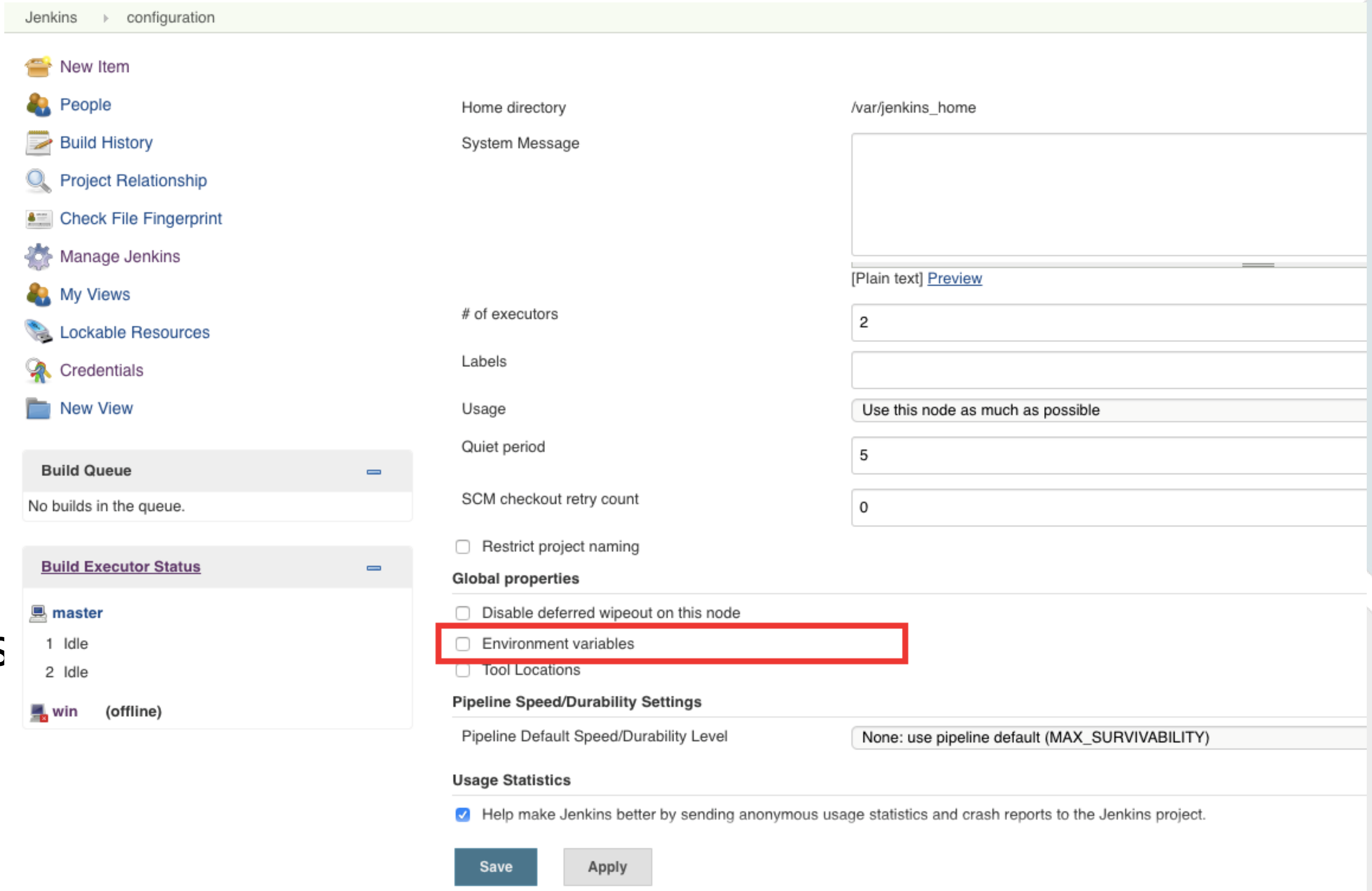
Cont’d…

- env CC will be available for all the stages in the pipeline but env dd will be available only for stage ‘Example1’
- So CC is called as global to pipeline and dd is local to a particular stage

Global Environments

Global environments

- To set Global environments
- Go to Manage Jenkins → Configure System → Global properties → check for Environment variables → click on Add
- Give variable and value as below



Jenkins › configuration

Home directory /var/jenkins_home
System Message
[Plain text] Preview
# of executors 2
Labels
Usage Use this node as much as possible
Quiet period 5
SCM checkout retry count 0
☐ Restrict project naming

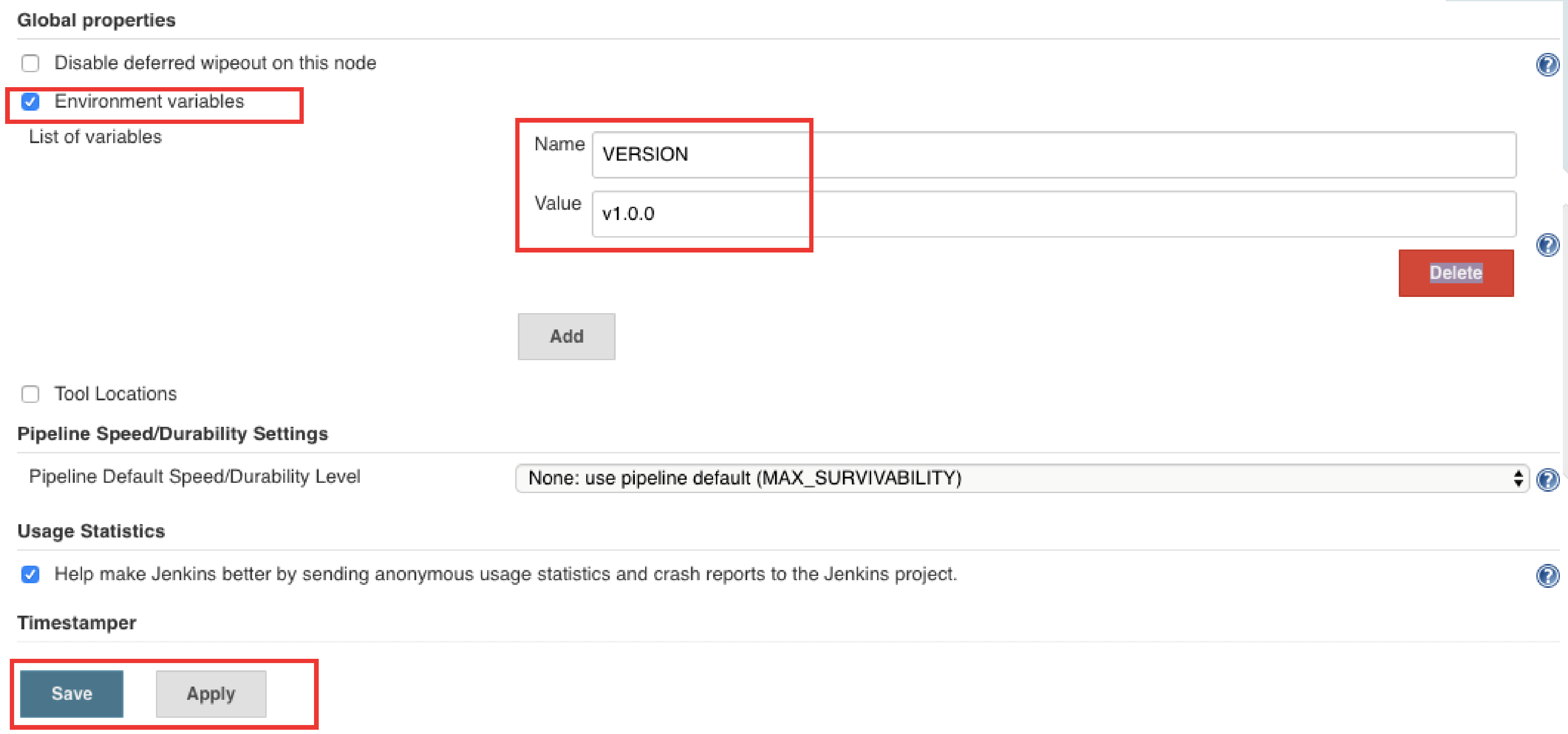
Global properties
☐ Disable deferred wipeout on this node
☐ Environment variables
☐ Tool Locations

Pipeline Speed/Durability Settings
Pipeline Default Speed/Durability Level None: use pipeline default (MAX_SURVIVABILITY)

Usage Statistics
☑ Help make Jenkins better by sending anonymous usage statistics and crash reports to the Jenkins project.

Cont'd…

- Then check on Environment variables and add variable and value as given
- The variable VERSION will be available for all slave-nodes and all environments in Jenkins
- The variables declared can be accessed globally



Global properties

☐ Disable deferred wipeout on this node

☑ Environment variables

List of variables

Name: VERSION

Value: v1.0.0

☐ Tool Locations

Pipeline Speed/Durability Settings

Pipeline Default Speed/Durability Level: None: use pipeline default (MAX_SURVIVABILITY)

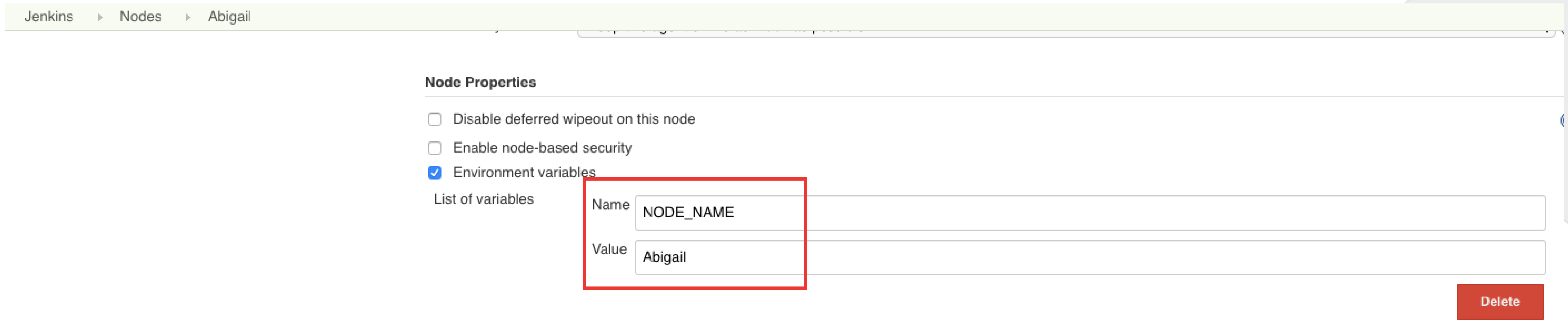
Usage Statistics

☑ Help make Jenkins better by sending anonymous usage statistics and crash reports to the Jenkins project.

Timestamper

Node Environment

- We can even declare environments which are specific to slave node only
- To do this, go to Jenkins → go to slave node → Configure → Node properties → check for Environment variables



Jenkins › Nodes › Abigail

Node Properties

- [ ] Disable deferred wipeout on this node
- [ ] Enable node-based security
- [x] Environment variables

List of variables

Name	NODE_NAME
Value	Abigail

Scripted Pipeline Example

- Like Declarative_Pipeline, is built on top of the underlying Pipeline sub-system
- Unlike Declarative, Scripted Pipeline is effectively a general-purpose DSL built with Groovy
- Most functionality provided by the Groovy language is made available to users of Scripted Pipeline, which means it can be a very expressive and flexible tool with which one can author continuous delivery pipelines

Example

```
node {     stage('Example') {
        if (env.BRANCH_NAME == 'master') {
            echo 'I only execute on the master branch'
        } else {
            echo 'I execute elsewhere'
        }
    }
}
```

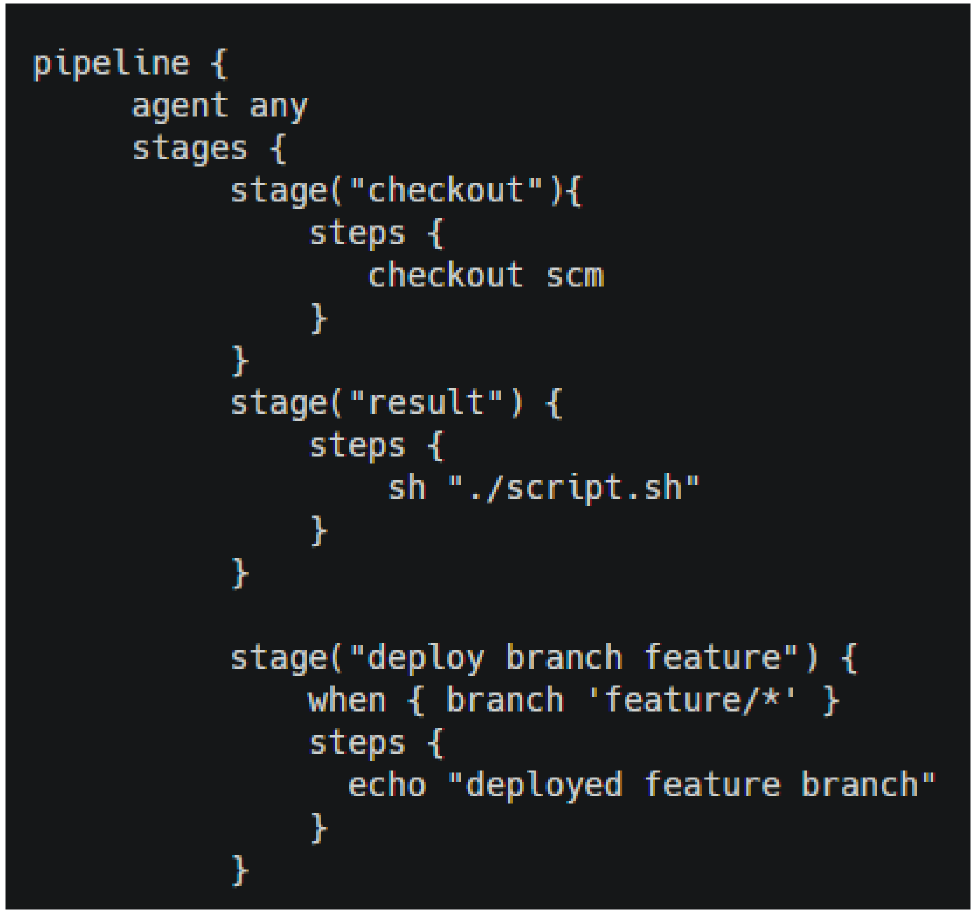
Cont'd…

- Scripted pipeline supports Groovy's **exception handling** support

Example

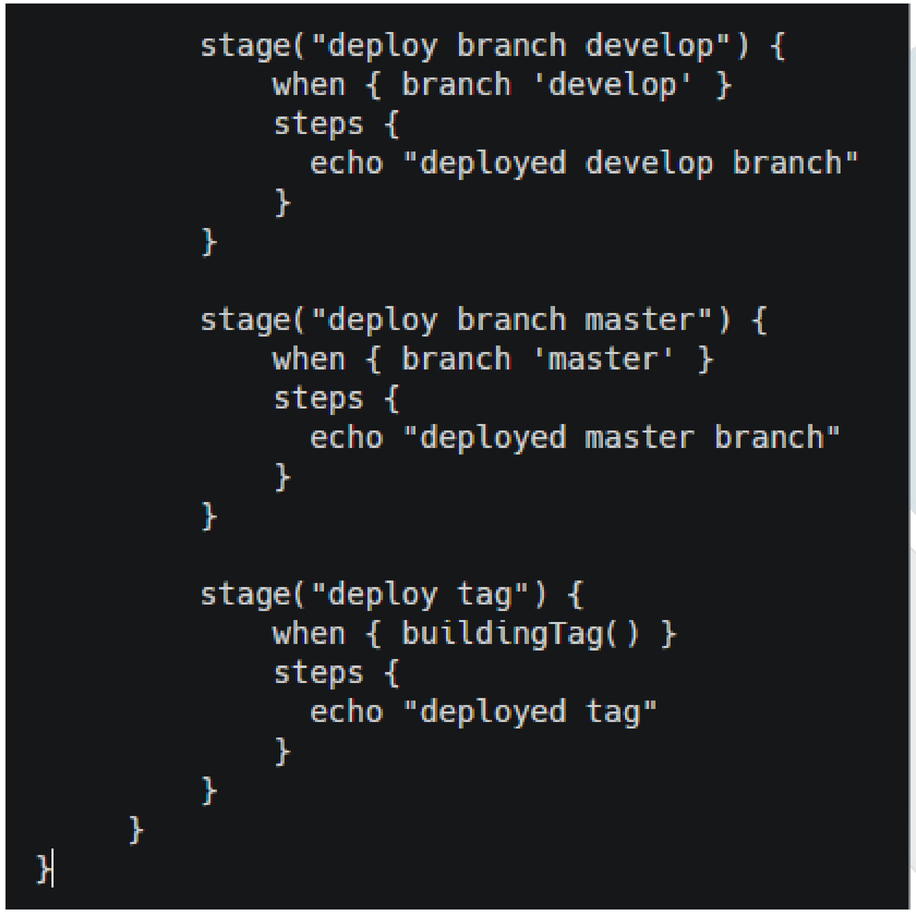
```
node {
    stage('Example') {
        try {
            sh 'exit 1'
        }
        catch (exc) {
            echo 'Something failed, I should sound the klaxons!'
        }
    }
}
```

Example for *when* condition



```
pipeline {
     agent any
     stages {
          stage("checkout"){
              steps {
                 checkout scm
              }
          }
          stage("result") {
              steps {
                  sh "./script.sh"
              }
          }

          stage("deploy branch feature") {
              when { branch 'feature/*' }
              steps {
                echo "deployed feature branch"
              }
          }
```



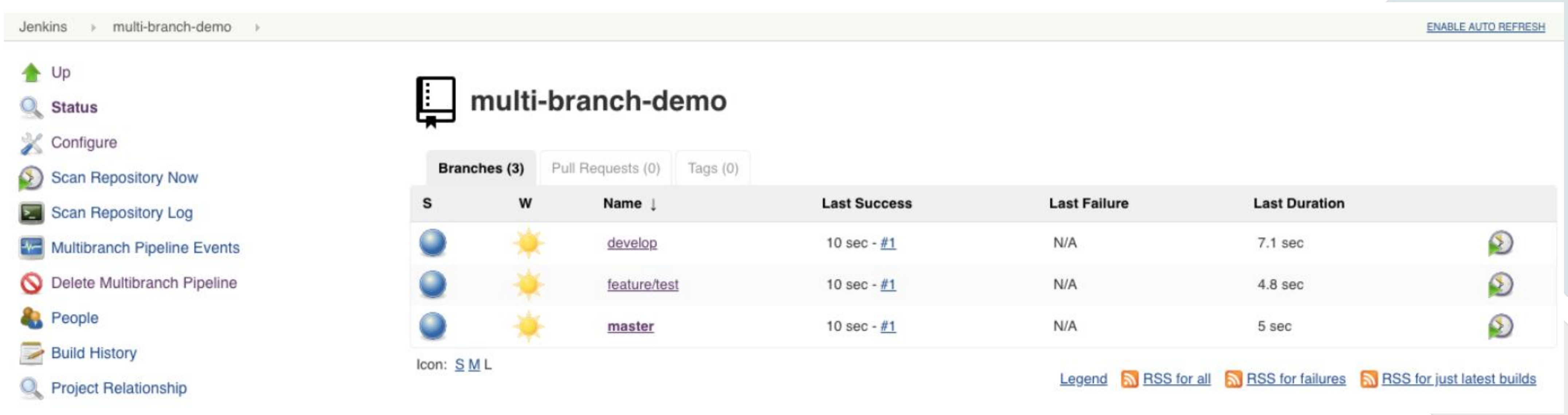
```
          stage("deploy branch develop") {
              when { branch 'develop' }
              steps {
                echo "deployed develop branch"
              }
          }

          stage("deploy branch master") {
              when { branch 'master' }
              steps {
                echo "deployed master branch"
              }
          }

          stage("deploy tag") {
              when { buildingTag() }
              steps {
                echo "deployed tag"
              }
          }
     }
}
```

Implementation

- Note that this works only on Multibranch Pipeline Projects



Jenkins › multi-branch-demo ›

ENABLE AUTO REFRESH

- Up
- Status
- Configure
- Scan Repository Now
- Scan Repository Log
- Multibranch Pipeline Events
- Delete Multibranch Pipeline
- People
- Build History
- Project Relationship

multi-branch-demo

Branches (3) | Pull Requests (0) | Tags (0)

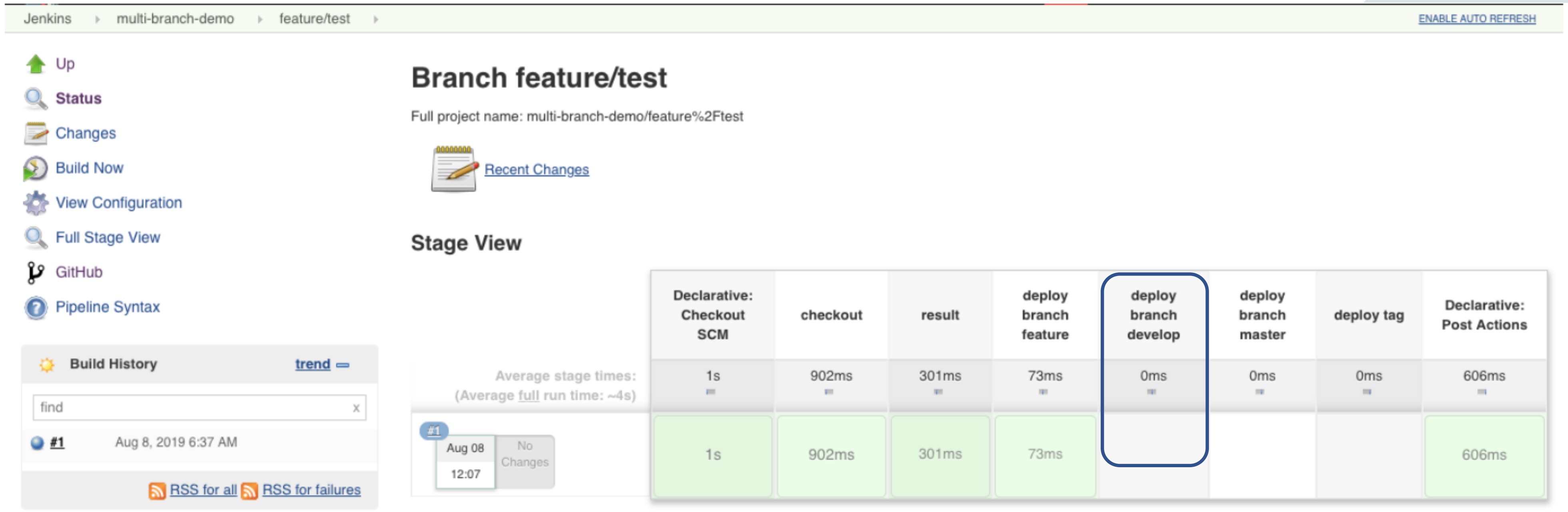
S	W	Name ↓	Last Success	Last Failure	Last Duration	
●	☀	develop	10 sec - #1	N/A	7.1 sec	
●	☀	feature/test	10 sec - #1	N/A	4.8 sec	
●	☀	master	10 sec - #1	N/A	5 sec	

Icon: S M L

Legend RSS for all RSS for failures RSS for just latest builds

Cont'd…

- ***when*** condition is true for ***feature/test***



Jenkins › multi-branch-demo › feature/test ›

ENABLE AUTO REFRESH

- Up
- Status
- Changes
- Build Now
- View Configuration
- Full Stage View
- GitHub
- Pipeline Syntax

Build History trend

find

#1 Aug 8, 2019 6:37 AM

RSS for all RSS for failures

Branch feature/test

Full project name: multi-branch-demo/feature%2Ftest

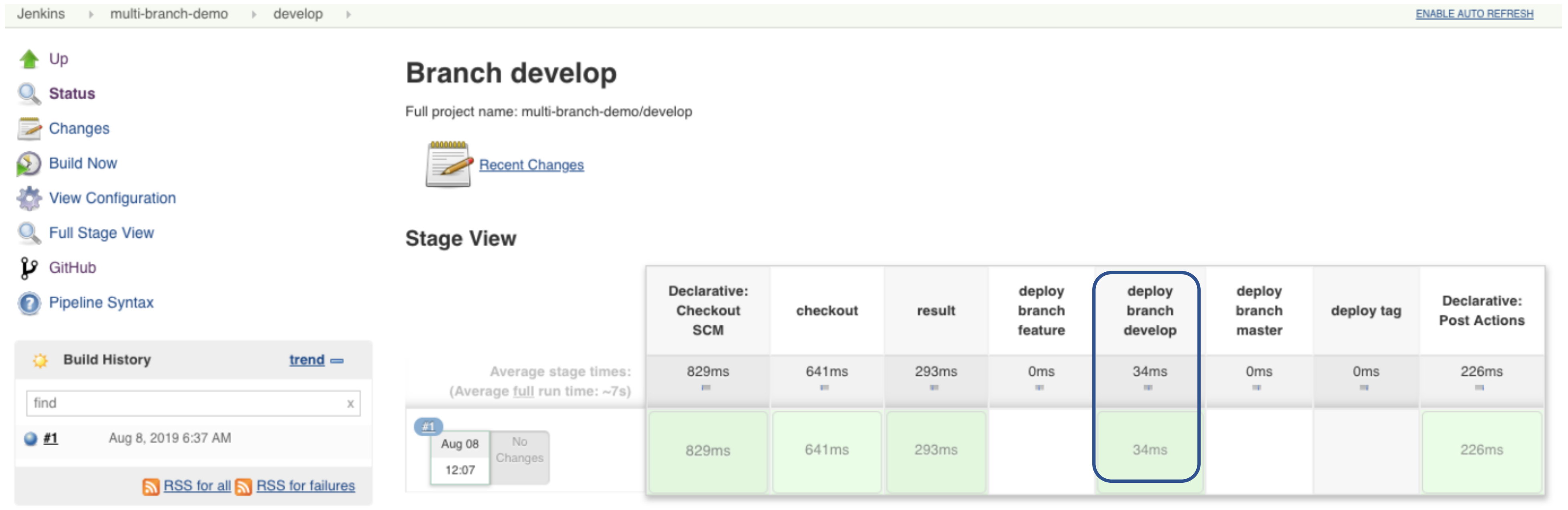
Recent Changes

Stage View

	Declarative: Checkout SCM	checkout	result	deploy branch feature	deploy branch develop	deploy branch master	deploy tag	Declarative: Post Actions
Average stage times: (Average full run time: ~4s)	1s	902ms	301ms	73ms	0ms	0ms	0ms	606ms
#1 Aug 08 12:07 No Changes	1s	902ms	301ms	73ms				606ms

Cont'd…

- ***when*** condition is true for ***develop branch***



Jenkins › multi-branch-demo › develop ›

ENABLE AUTO REFRESH

- Up
- Status
- Changes
- Build Now
- View Configuration
- Full Stage View
- GitHub
- Pipeline Syntax

Build History trend

find

#1 Aug 8, 2019 6:37 AM

RSS for all RSS for failures

Branch develop

Full project name: multi-branch-demo/develop

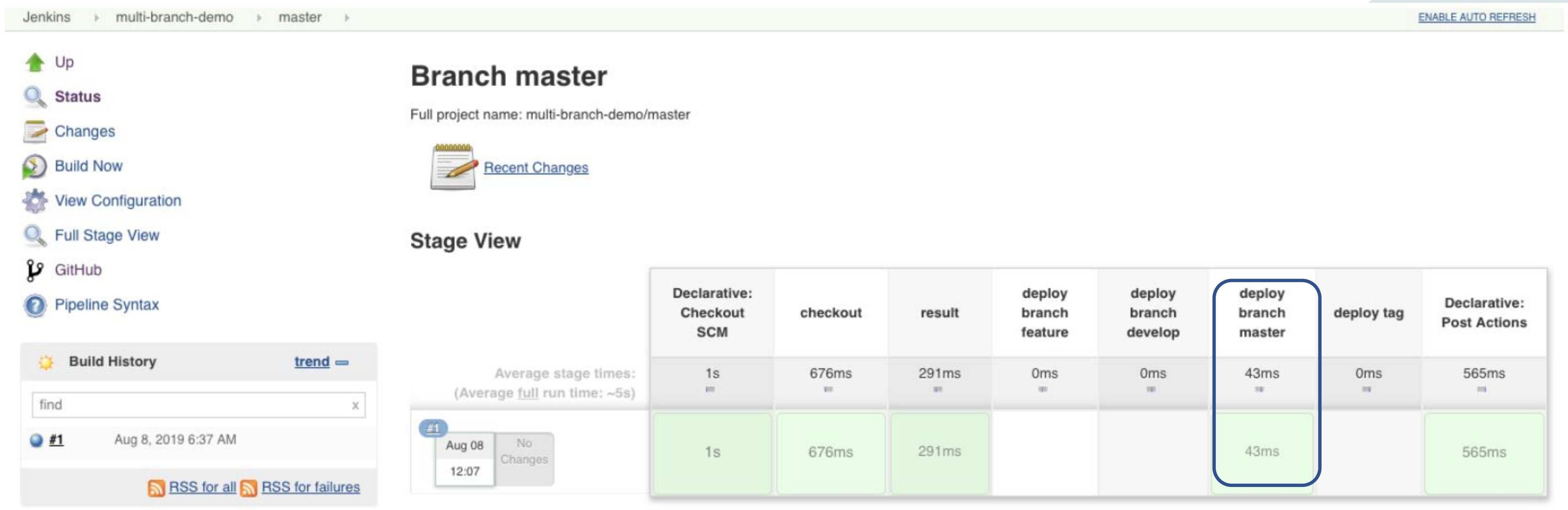
Recent Changes

Stage View

	Declarative: Checkout SCM	checkout	result	deploy branch feature	deploy branch develop	deploy branch master	deploy tag	Declarative: Post Actions
Average stage times: (Average full run time: ~7s)	829ms	641ms	293ms	0ms	34ms	0ms	0ms	226ms
#1 Aug 08 12:07 No Changes	829ms	641ms	293ms		34ms			226ms

Cont'd…

- ***when*** condition is true for ***master branch***



Jenkins › multi-branch-demo › master ›

ENABLE AUTO REFRESH

- Up
- Status
- Changes
- Build Now
- View Configuration
- Full Stage View
- GitHub
- Pipeline Syntax

Build History trend

find

#1 Aug 8, 2019 6:37 AM

RSS for all RSS for failures

Branch master

Full project name: multi-branch-demo/master

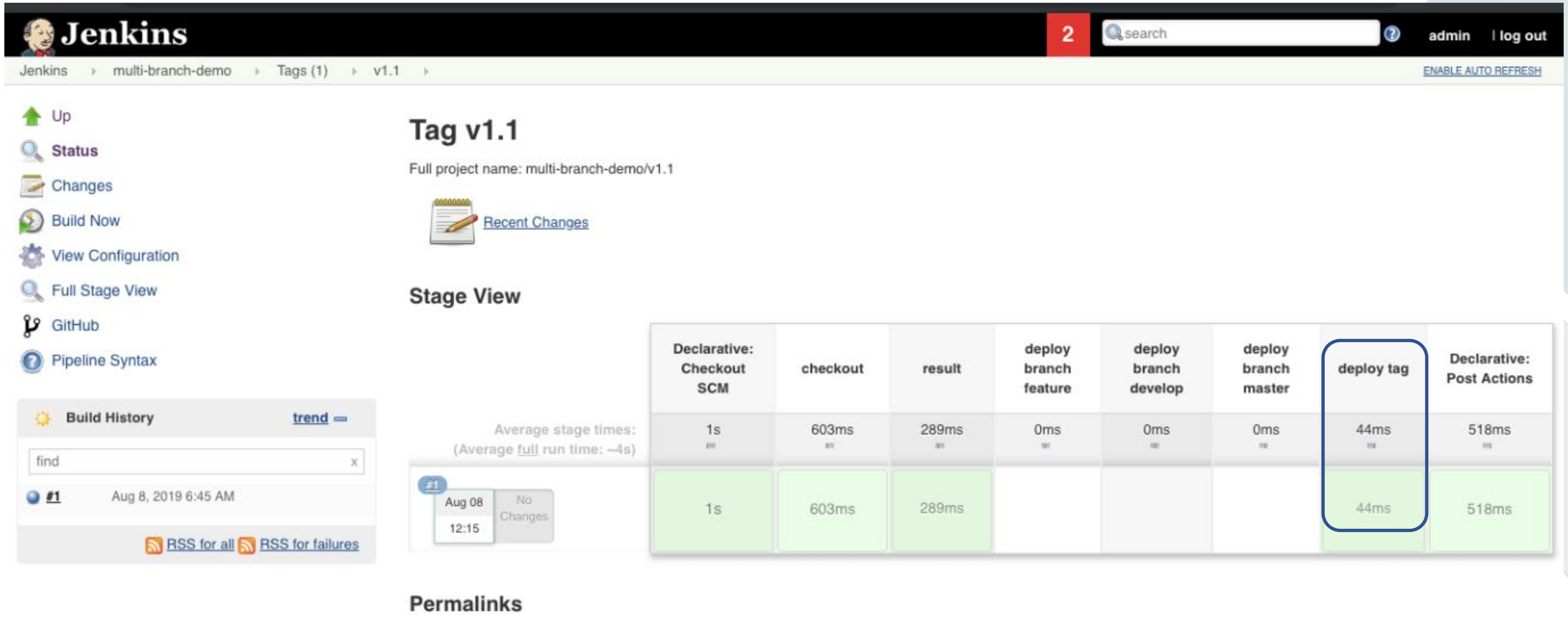
Recent Changes

Stage View

	Declarative: Checkout SCM	checkout	result	deploy branch feature	deploy branch develop	deploy branch master	deploy tag	Declarative: Post Actions
Average stage times: (Average full run time: ~5s)	1s	676ms	291ms	0ms	0ms	43ms	0ms	565ms
#1 Aug 08 12:07 No Changes	1s	676ms	291ms			43ms		565ms

Cont'd…

- ***when*** condition is true for ***tag***



Jenkins | 2 | search | admin | log out

Jenkins › multi-branch-demo › Tags (1) › v1.1 ›

ENABLE AUTO REFRESH

Up
Status
Changes
Build Now
View Configuration
Full Stage View
GitHub
Pipeline Syntax

Build History trend

find

#1 Aug 8, 2019 6:45 AM

RSS for all RSS for failures

Tag v1.1

Full project name: multi-branch-demo/v1.1

Recent Changes

Stage View

	Declarative: Checkout SCM	checkout	result	deploy branch feature	deploy branch develop	deploy branch master	deploy tag	Declarative: Post Actions
Average stage times: (Average full run time: ~4s)	1s	603ms	289ms	0ms	0ms	0ms	44ms	518ms
#1 Aug 08 12:15 No Changes	1s	603ms	289ms				44ms	518ms

Permalinks