No such DSL method `stages`

Asked 7 years, 10 months ago Modified 2 years, 6 months ago Viewed 123k times

Part of CI/CD Collective

48

I'm trying to create my first Groovy script for Jenkins:

After looking here https://jenkins.io/doc/book/pipeline/, I created this:

```
node {
  stages {

    stage('HelloWorld') {
      echo 'Hello World'
    }

    stage('git clone') {
      git clone "ssh://git@mywebsite.example/myrepo.git"
    }

  }
}
```

However, I'm getting:

```
java.lang.NoSuchMethodError: No such DSL method "stages" found among steps
```

What am I missing?

Also, how can I pass my credentials to the Git Repository without writing the password in plain text?

CI/CD jenkins groovy jenkins-pipeline

Share

Improve this question

Follow

edited Jun 23, 2022 at 22:34

Stephen Ostermiller ♦

25.5k • 16 • 94 • 114

asked Feb 8, 2017 at 12:52

bsky

20.2k • 54 • 166 • 282

My problem was just the inverse of this. I was getting the error "no such DSL method `steps`" – Daniel Watrous Nov 7, 2017 at 16:06

2 Answers

Sorted by: Highest score (default)

102

You are confusing and mixing `Scripted Pipeline` with `Declarative Pipeline`, for complete difference see here. But the short story:

- **declarative pipelines** is a new extension of the pipeline DSL (it is basically a pipeline script with only one step, a pipeline step with arguments (called directives), these directives should follow a specific syntax. The point of this new format is that it is more strict and therefor should be easier for those new to pipelines, allow for graphical editing and much more.
- **scripted pipelines** is the fallback for advanced requirements.

So, if we look at your script, you first open a `node` step, which is from scripted pipelines. Then you use `stages` which is one of the directives of the `pipeline` step defined in `declarative pipeline`. So you can for example write:

```
pipeline {
  ...
  stages {
    stage('HelloWorld') {
      steps {
        echo 'Hello World'
      }
    }
    stage('git clone') {
      steps {
        git clone "ssh://git@mywebsite.example/myrepo.git"
      }
    }
  }
}
```

So if you want to use `declarative pipeline` that is the way to go.

If you want to `scripted pipeline`, then you write:

```
node {
  stage('HelloWorld') {
    echo 'Hello World'
  }

  stage('git clone') {
    git clone "ssh://git@mywebsite.example/myrepo.git"
  }
}
```

E.g.: skip the stages block.

Share

Improve this answer

Follow

edited Jun 23, 2022 at 22:32

Stephen Ostermiller ♦

25.5k ● 16 ● 94 ● 114

answered Feb 8, 2017 at 13:09

Jon S

16.3k ● 4 ● 46 ● 46

You are missing a steps block around your `echo` step in the declarative pipeline example. – pmr Feb 10, 2017 at 10:46

This is the key answer that resolves my confusion I was struggling for such a long time! Thanks! – Calvin Zhou Aug 8, 2019 at 8:42

any idea about this issue stackoverflow.com/questions/60001629/... – Kumaresan Sd Feb 1, 2020 at 6:22

4 Man, what a mess. – jaques-sam Feb 13, 2020 at 8:54

1 @FrancisVinluan use `node('label') { ... }` – Jon S Jun 10, 2021 at 6:22

16

A Jenkinsfile can be written using two types of syntax - **Declarative** and **Scripted**.

Declarative and Scripted Pipelines are constructed fundamentally differently. Declarative Pipeline is a more recent feature of Jenkins Pipeline which:

- provides richer syntactical features over Scripted Pipeline syntax, and
- is designed to make writing and reading Pipeline code easier.

Many of the individual syntactical components (or "steps") written into a Jenkinsfile, however, are common to both Declarative and Scripted Pipeline. Example:

Declarative Pipeline fundamentals

In Declarative Pipeline syntax, the `pipeline` block defines all the work done throughout your entire Pipeline.

Jenkinsfile (Declarative Pipeline):

```
pipeline {
    agent any 1
    stages {
        stage('Build') { 2
            steps {
                // 3
            }
        }
        stage('Test') { 4
            steps {
                // 5
            }
        }
        stage('Deploy') { 6
            steps {
                // 7
            }
        }
```

```
    }
}
```

1. Execute this Pipeline or any of its stages, on any available agent.
2. Defines the "Build" stage.
3. Perform some steps related to the "Build" stage.
4. Defines the "Test" stage.
5. Perform some steps related to the "Test" stage.
6. Defines the "Deploy" stage.
7. Perform some steps related to the "Deploy" stage.

Scripted Pipeline fundamentals

In Scripted Pipeline syntax, one or more `node` blocks do the core work throughout the entire Pipeline. Although this is not a mandatory requirement of Scripted Pipeline syntax, confining your Pipeline's work inside of a `node` block does two things:

1. Schedules the steps contained within the block to run by adding an item to the Jenkins queue. As soon as an executor is free on a node, the steps will run.
2. Creates a workspace (a directory specific to that particular Pipeline) where work can be done on files checked out from source control.
   Caution: Depending on your Jenkins configuration, some workspaces may not get automatically cleaned up after a period of inactivity. See tickets and discussion linked from JENKINS-2111 for more information.

Jenkinsfile (Scripted Pipeline):

```
node { 1
    stage('Build') { 2
        // 3
    }
    stage('Test') { 4
        // 5
    }
    stage('Deploy') { 6
        // 7
    }
}
```

1. Execute this Pipeline or any of its stages, on any available agent.
2. Defines the "Build" stage. `stage` blocks are optional in Scripted Pipeline syntax. However, implementing `stage` blocks in a Scripted Pipeline provides clearer visualization of each `stage's subset of tasks/steps in the Jenkins UI.

3. Perform some steps related to the "Build" stage.
4. Defines the "Test" stage. 5
5. Perform some steps related to the "Test" stage.
6. Defines the "Deploy" stage.
7. Perform some steps related to the "Deploy" stage.

Pipeline example

Here is an example of a `Jenkinsfile` using Declarative and it's equivalent scriptive Pipeline syntax:

Jenkinsfile (Declarative Pipeline):

```
pipeline {
    agent any
    options {
        skipStagesAfterUnstable()
    }
    stages {
        stage('Build') {
            steps {
                sh 'make'
            }
        }
        stage('Test'){
            steps {
                sh 'make check'
                junit 'reports/**/*.xml'
            }
        }
        stage('Deploy') {
            steps {
                sh 'make publish'
            }
        }
    }
}
```

Jenkinsfile (Scripted Pipeline):

```
node {
    stage('Build') {
        sh 'make'
    }
    stage('Test') {
        sh 'make check'
        junit 'reports/**/*.xml'
    }
    if (currentBuild.currentResult == 'SUCCESS') {
        stage('Deploy') {
            sh 'make publish'
```

```
        }
    }
}
```

Share

Improve this answer

Follow

edited Jun 24, 2019 at 8:26

answered Jun 24, 2019 at 8:21

M-Razavi

3,467 ● 2 ● 35 ● 47