Conditional environment variables in Jenkins Declarative Pipeline

I'm trying to get a declarative pipeline that looks like this:

```
pipeline {
    environment {
        ENV1 = 'default'
        ENV2 = 'default also'
    }
}
```

The catch is, I'd like to be able to override the values of ENV1 or ENV2 based on an arbitrary condition. My current need is just to base it off the branch but I could imagine more complicated conditions.

Is there any sane way to implement this? I've seen some examples online that do something like:

```
stages {
    stage('Set environment') {
        steps {
            script {
                ENV1 = 'new1'
            }
        }
    }
}
```

But i believe this isn't setting the actually environment variable, so much as it is setting a local variable which is overriding later calls to ENV1. The problem is, I need these environment variables read by a nodejs script, and those need to be real machine environment variables.

Is there any way to set environment variables to be dynamic in a jenkinsfile?

jenkins jenkins-pipeline

4 Answers

use `withEnv` to set environment variables dynamically for use in a certain part of your pipeline (when running your node script, for example). like this (replace the contents of an sh step with your node script):

```
pipeline {
    agent { label 'docker' }
    environment {
        ENV1 = 'default'
    }
    stages {
        stage('Set environment') {
            steps {
                sh "echo $ENV1" // prints default
                // override with hardcoded value
                withEnv(['ENV1=newvalue']) {
                    sh "echo $ENV1" // prints newvalue
                }
                // override with variable
                script {
                    def newEnv1 = 'new1'
                    withEnv(['ENV1=' + newEnv1]) {
                        sh "echo $ENV1" // prints new1
                    }
                }
            }
```

```
        }
    }
}
```

1 Ah yes, this does work. It's unfortunate that there is no way to do it for all stages, but this will have to do for now. Thanks! – Paul Sachs May 17 '17 at 15:48

```
pipeline {
    agent none
    environment {
        ENV1 = 'default'
        ENV2 = 'default'
    }
    stages {
        stage('Preparation') {
            steps {
                script {
                    ENV1 = 'foo' // or variable
                    ENV2 = 'bar' // or variable
                }
                echo ENV1
                echo ENV2
            }
        }
        stage('Build') {
            steps {
                sh "echo ${ENV1} and ${ENV2}"
            }
        }
        // more stages...
    }
}
```

This method is more simple and looks better. Overridden environment variables will be applied to all other stages also.

3 Actually this doesn't overwrite the environment variable, but defines a new global-scope Groovy variable (I think) Try to echo `${env.ENV1}` int the *Build* stage and you'll see that it still has the original value! – Kutzi Feb 27 at 15:54

environment{} can be used at the stage level for that. However it seems that it needs to be repeated for every stage where it is needed. – Tos May 7 at 14:58

I tried to do it in a different way, but unfortunately it does not entirely work:

```
pipeline {
  agent any
  environment {
    TARGET = "${changeRequest() ? CHANGE_TARGET:BRANCH_NAME}"
  }

  stages {
    stage('setup') {
        steps {
            echo "target=${TARGET}"
            echo "${BRANCH_NAME}"
        }
    }
  }
}
```

Strangely enough this works for my pull request builds (changeRequest() returning true and TARGET becoming my target branch name) but it does not work for my CI builds (in which case the branch name is e.g. release/201808 but the resulting TARGET evaluating to null)

I managed to get this working by explicitly calling shell in the environment section, like so:

```
UPDATE_SITE_REMOTE_SUFFIX = sh(returnStdout: true, script: "if [ \"$GIT_BRANCH\" ==
\"develop\" ]; then echo \"\"; else echo \"-$GIT_BRANCH\"; fi").trim()
```

however I know that my Jenkins is on nix, so it's probably not that portable

answered Sep 24 at 0:55

Karrde
292 1 7
