Jenkins Pipelines

Jenkins Pipelines

- Identify Git workflows that enable CI and easily integrate into Jenkins
- Use a version-controlled project with multiple branches and build it on Jenkins
- Use the declarative Jenkins pipeline and add pipeline to version control

Scripted pipeline

- Code is written on the Jenkins UI instance and is enclosed within the node block

  - node {
  - scripted pipeline code
  - }

Declarative pipeline

- Code is written locally in a file and is checked into a SCM and is enclosed within the pipeline block
  - pipeline {
  - declarative pipeline code
  - }

Pipeline Concepts

- Pipeline
  - A user-defined block which contains all the stages
  - It is a key part of declarative pipeline syntax
- Node
  - A node is a machine that executes an entire workflow
  - It is a key part of the scripted pipeline syntax.
- Agent
  - instructs Jenkins to allocate an executor for the builds
  - It is defined for an entire pipeline or a specific stage

Agent Parameters

- Any
  - Runs pipeline/ stage on any available agent
- None
  - Indicates that there is no global agent & each stage must specify its own agent
- Label
  - Executes the pipeline/stage on the labelled agent.
- Docker
  - Uses Docker container as an execution environment

Jenkins Pipeline syntax example

```
node {
    stage('SCM checkout') {
        //Checkout from your SCM(Source Control Management)
        //For eg: Git Checkout
    }
    stage('Build') {
        //Compile code
        //Install dependencies and Perform Unit Test, Integration Test
    }
    stage('Test') {
        //Perform UAT
    }
    stage('Deploy') {
        //Deploy code to prod server
}
}
```

The Jenkinsfile

The Jenkinsfile

- A pipeline in Jenkins is defined using a script called the **Jenkinsfile**
- While working with the Jenkins scripted pipeline, we use standard Groovy syntax
- The scripted pipeline has some special directives that perform different functions

The Jenkinsfile

Directive	Explanation node
`node`	This defines where the job is going to be run. We will explore more about this in the next chapter as we cover setting up master-slave relationships on Jenkins.
`dir`	This directive defines what directory/folder to run the following directives on.
`stage`	This defines the stage of your pipeline, for example, what task it's running.
`git`	This points to the remote repository where you pull the changes from.
`sh`	This defines the shell script to run on a UNIX-based environment. On a Windows environment, we would use the `bat` directive instead.
`def`	As mentioned previously, the pipeline is written in Groovy; thus, we can define functions to perform different actions. In this case, we defined a `printMessage` function, which prints out different messages at the start and end of our pipeline.

Creating the Pipeline

- Go to the Jenkins dashboard and select New Item.
- Enter an appropriate name(PipeLine-Project-1) for the project and select Pipeline for the project type
- In the project configuration, under the **General** tab, select **GitHub project** and enter the appropriate URL
  - https://github.com/atingupta2005/simple-java-maven-app
- Under the **Build Triggers** section, select the **GitHub hook trigger for GITScm polling**
  - Need to create a Webhoob in Github Repo – Settings->Webhook
    - http://52.142.55.134:8080/github-webhook/
- Under the **Pipeline** section, select **Pipeline script** under **Definition**.
- In the script section of the configuration, add the snippet of code:

Creating the Pipeline

```
node('master') {
    stage("Fetch Source Code") {
        git ' https://github.com/atingupta2005/jenkins-python-test'
    }
    dir('.') {
        printMessage('Running Pipeline')
        stage("Testing") {
            sh 'python tests.py'
        }
        printMessage('Pipeline Complete')
    }
}
def printMessage(message) {
    echo "${message}"
}
```

Creating the Pipeline

- Press Apply
- Select Save
- Select Build Now
- On the project dashboard, after running our build, the Stage View shows up.

Creating Multibranch Pipelines

Creating Multibranch Pipelines

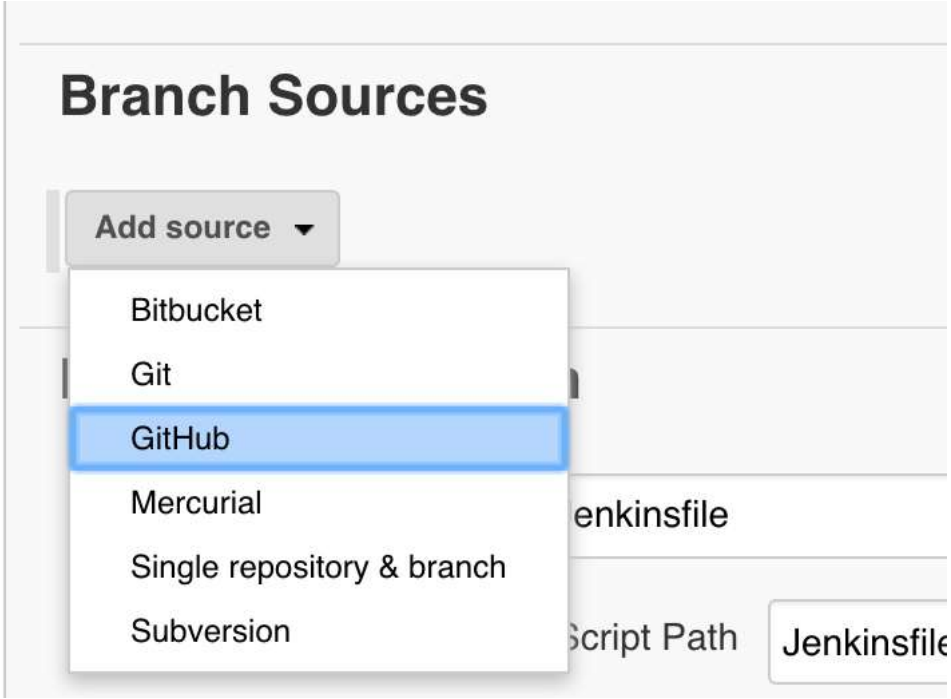
- Multibranch pipelines will enable you to build different branches besides the default.

Global Variables

- A global variable is accessible in any scope within our program
- There are pre-defined global variables. Examples:
  - BRANCH_NAME
  - BUILD_NUMBER
  - BUILD_ID
  - JOB_NAME
  - NODE_NAME
  - JENKINS_HOME
  - BUILD_URL

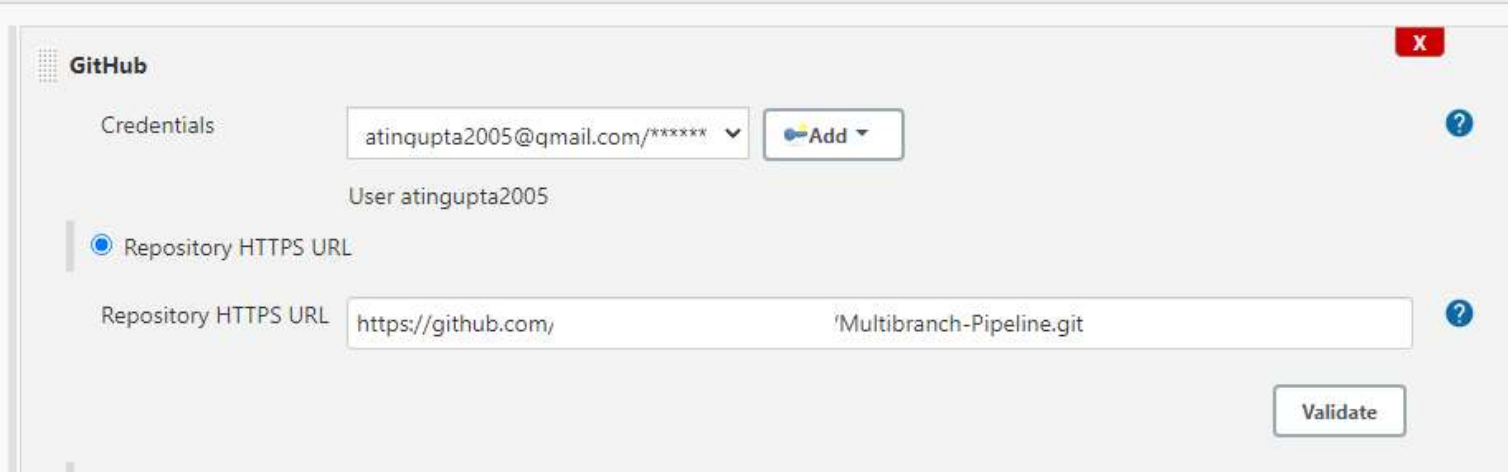
Creating Multibranch Pipelines

- Create a new project on Jenkins and, under the Project type, select Multibranch pipeline and enter an appropriate name for the project.
- Under Branch Sources, select Add source and select GitHub
- Under Kind, select Username with password.

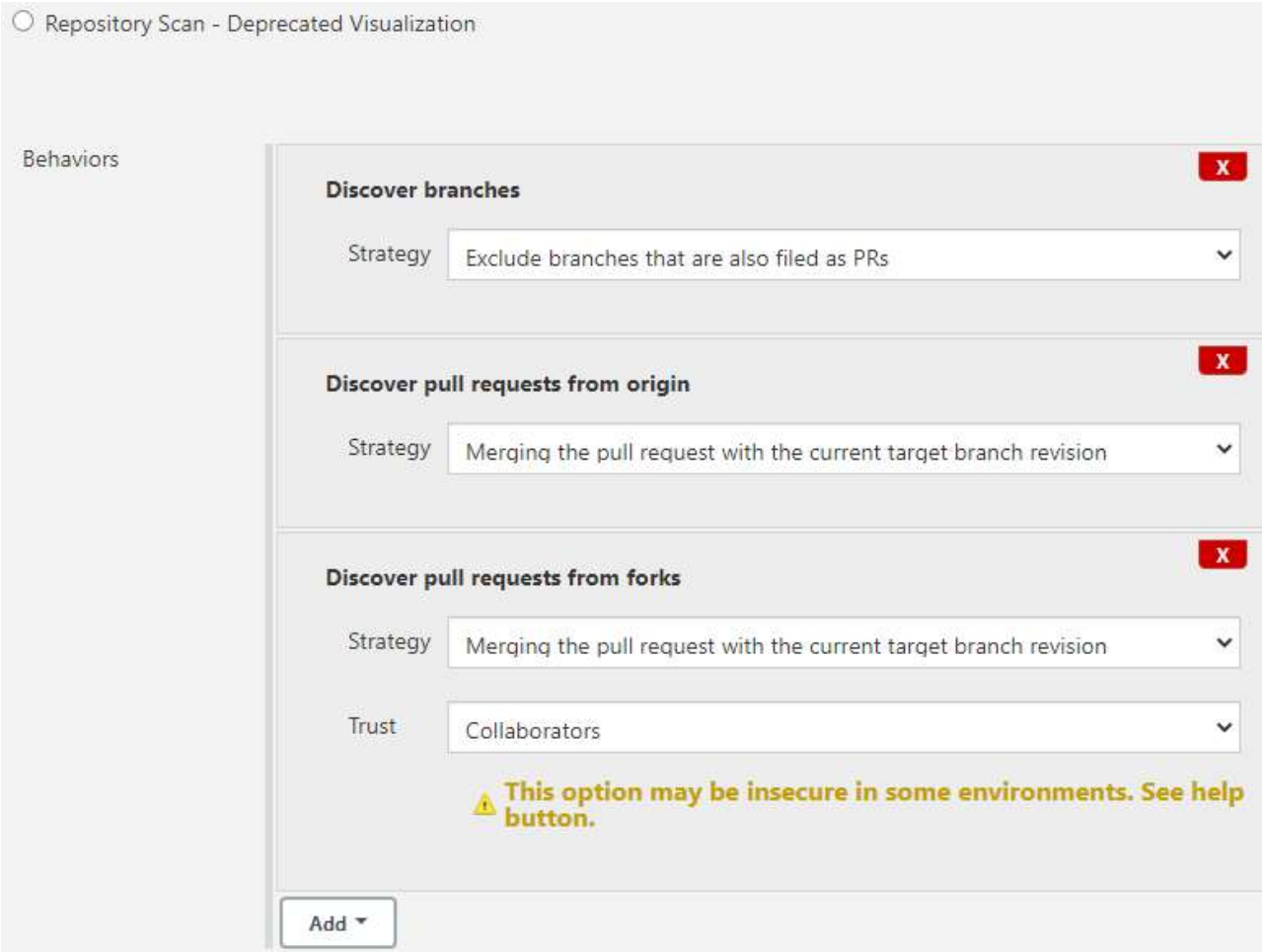


Creating Multibranch Pipelines

- Under the Build Configuration section, set up the project as shown.
- GitHub Project URL:
  - https://github.com/atingupta2005/Multibranch-Pipeline



Creating Multibranch Pipelines



- Select Apply and Save
- Under the Branches tab, select the first item, which in our case is master

Building Pull Requests

- On your Git Bash in the project root, check out to a new branch and do some modifications in the code
  - Multibranch-Pipeline.git

- Push the new branch to the remote and create a new pull request.

- Going back to Jenkins, we can see our new branch and, under the Pull Requests tab
- We can see that the Pull Request we created has been built by Jenkins using the same pipeline stages.

Thanks