cloudbees UNIVERSITY

CloudBees University
Jenkins Pipeline - Intermediate

Table of Contents

Introduction

This workbook supplements your CloudBees Jenkins training. It consists of a sequence of lab exercises through which you will learn to work with Pipelines.

These lab exercises should be completed in sequence and in the presence of a Certified Trainer.

All of the plugins and dependencies required for this course are bundled with the installation.

IMPORTANT	Never try to upgrade plugins on the Lab's Jenkins instances !

If you encounter problems with your software installation, or if you do not understand any of the instructions, please ask your instructor for help.

Lab exercises

The solution to each task is located at the end of that task. Please try to solve the assignment by yourself and look at the solutions only if you get stuck or want to validate your work.

IMPORTANT	This training is language-agnostic, so you are not expected to know the language. While we recommend that you familarize yourself with ancillary technologies such as Apache Maven, Gradle, Ant, NPM, Apache Groovy, Docker and Git/GitHub, you should be able to complete the lab exercises by copying commands and text that are given in the class. You will not need any additional tools.

Understand your lab environment

Before diving into the exercises, you need to familiarize yourself with your lab environment. Your home page contains links to the facilities you need. When you click a link from this page, it opens in a new tab.

The first two links are for documentation:

- *Slides* — Slide set used for the lecture portions of this class.
- *Labs Document* — Workbook for the lab portions of this course.

The last three links are to work environments:

- *Jenkins Master* — The Jenkins Master dashboard; this is the environment you will use for most of the work in this course.
- *Gitserver* — Git repository page for the projects in this course. If you are used to GitHub, this page will look familiar although it is actually running Gitea, which gives you a local advanced Git server with a web interface from which to browse repositories, authenticate, do pull requests and reviews.
- *DevBox* — A `bash` shell environment that provides a command line interface.

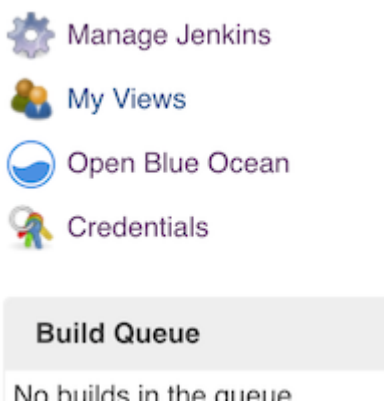
For the labs associated with this class, we will not be using *DevBox*.

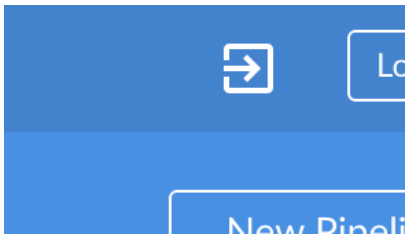
Credentials To Use

Use the id of `butler` and the password of `butler` for all credentials in your lab environment.

Blue Ocean

The Blue Ocean plugin is installed in your lab environment. To open it:

- Open the Jenkins Master dashboard
- Click "Open Blue Ocean" in the left frame:
  - 

- Switch to Classic Web UI
  - Click on the arrow button to switch to "Jenkins Web UI":
    - 
  - Click "Open Blue Ocean" in the side bar to switch back.

NOTE "Switch to Classic Web UI" is available at the top of most pages in Blue Ocean

Blue Ocean Refresher

In the exercise for this section you will:

- Review Pipeline resources
- Create and run a Pipeline in another feature branch
- Move some steps to `post` sections
- Add a `when` directive to skip Deploy stages when not on `master` branch
- Add and use parameters
- Save a Pipeline to the master branch

NOTE In order to open the Blue Ocean Pipeline Code Editor, type `Ctrl`+`S` on Windows or `Cmd`+`S` on macOS

Task: Review Pipeline resources

The Jenkins Pipeline features in this lab are not accessible via the Blue Ocean Pipeline Editor. The rest of this lab will require some combination the Classic Web UI and the Blue Ocean Pipeline Code Editor.

Snippet Generator

- Most steps are visible from the Blue Ocean Pipeline Editor, but sometimes you may want options that Blue Ocean does not yet support.
- Open the Snippet generator, select a step, add some value and generate a code snippet.
- Repeat this for a few different steps

Declarative Directive Generator

- The Declarative Directive Generator allows you to generate the Pipeline code for a Declarative Pipeline directive, such as agent, options, when and more.
- Open Declarative Directive Generator, select a directive, add some value and generate a declarative directive.
- Repeat this for a few different directives

Declarative Pipeline Documentation

- Declarative Pipeline syntax cannot be generated by the Snippet Generator.
- As you perform the tasks in this lab you will need to refer to the Declarative Pipeline Syntax to know what Declarative Pipeline code to use.

Task: Create and run a Pipeline in another feature branch

- Edit your pipeline from `master`
- Save the pipeline. In the Save dialog, provide the description "Start finished pipeline" and save to a new branch named `finished-pipeline`.

Task: Move some steps to `post` sections

- Edit your pipeline from `finished-pipeline`
- Open the Blue Ocean Pipeline Code Editor
  - Add a post section to any stage that contains any of the following steps:
    - `archiveArtifacts`
    - `junit`
    - `stash`
  - Move the `archiveArtifacts` and `stash` steps into `post { success { … } }` conditions.
  - Move the `junit` steps into `post { always { … } }` conditions.

- Click update to apply your changes. The steps will no longer be visible in the Blue Ocean Pipeline Editor, but they are still there.

- Save the pipeline. In the Save dialog, provide the description "Move steps to post sections" and save to the default branch (`finished-pipeline` branch).

Task: Add a `when` directive to skip Deploy stages when not on `master` branch

- Edit your pipeline from `finished-pipeline`
- Open the Blue Ocean Pipeline Code Editor
  - Add a when directive with a `branch` condition in both the "Deploy" stages so that the stages are skipped when the pipeline is run from a branch other than `master`.
  - Click "Update" to apply your changes. The changes will not be visible in the Blue Ocean Pipeline Editor, but they are still there.
- Save the pipeline. In the Save dialog, provide the description "Skip Deploy when not on master" and save to the default branch (`finished-pipeline` branch).
- Verify that both "Deploy" stages were skipped.

Task: Add and use a parameter

- Edit your pipeline from `finished-pipeline`
- Open the Blue Ocean Pipeline Code Editor
  - Add a parameters directive.
  - Add a `string` parameter named "DEPLOY_TO" with the default value of "dev".
  - Update the call to the shell script for `./jenkins/deploy.sh`, replacing "staging" with the `DEPLOY_TO` parameter.
  - Click "Update" to apply your changes. The parameter will not be visible, but you can see that the shell script is changed when you view the Jenkinsfile.
- Save the pipeline. In the Save dialog, provide the description "Add DEPLOY_TO parameter" and save to the default branch (`finished-pipeline` branch).
- Since you are not on the `master` branch yet, you won't see what effect the parameter has during the job execution. You'll review that in the next task.

WARNING There is a known issue where parameters are not present the first time the pipeline runs after they are added. Once the pipeline completes, run it again and Blue Ocean will ask you to enter the parameter before continuing.

Task: Save a Pipeline to the master branch

- Edit your pipeline from `finished-pipeline`
- Save the pipeline. In the Save dialog, provide the description "Change to Finished pipeline in master" and save to the `master` branch.

- Verify that both "Deploy" stages were included in the run.
- Look at the output of the "Fluffy Deploy" stage and verify the parameter value from the previous task has been set.

Solution

Jenkinsfile

```
pipeline {
  agent none
  stages {
    stage('Fluffy Build') {
      parallel {
        stage('Build Java 8') {
          agent {
            node {
              label 'java8'
            }
          }
          steps {
            sh './jenkins/build.sh'
          }
          post {
            success {
              stash(name: 'Java 8', includes: 'target/**')
            }
          }
        }
        stage('Build Java 7') {
          agent {
            node {
              label 'java7'
            }
          }
          steps {
            sh './jenkins/build.sh'
          }
          post {
            success {
              archiveArtifacts 'target/*.jar'
              stash(name: 'Java 7', includes: 'target/**')
            }
          }
        }
      }
    }
    stage('Fluffy Test') {
      parallel {
        stage('Backend Java 8') {
          agent {
            node {
              label 'java8'
            }
          }
          steps {
```

```
        unstash 'Java 8'
        sh './jenkins/test-backend.sh'
      }
      post {
        always {
          junit 'target/surefire-reports/**/TEST*.xml'
        }
      }
    }
    stage('Frontend') {
      agent {
        node {
          label 'java8'
        }
      }
      steps {
        unstash 'Java 8'
        sh './jenkins/test-frontend.sh'
      }
      post {
        always {
          junit 'target/test-results/**/TEST*.xml'
        }
      }
    }
    stage('Performance Java 8') {
      agent {
        node {
          label 'java8'
        }
      }
      steps {
        unstash 'Java 8'
        sh './jenkins/test-performance.sh'
      }
    }
    stage('Static Java 8') {
      agent {
        node {
          label 'java8'
        }
      }
      steps {
        unstash 'Java 8'
        sh './jenkins/test-static.sh'
      }
    }
    stage('Backend Java 7') {
      agent {
        node {
          label 'java7'
        }
      }
      steps {
        unstash 'Java 7'
        sh './jenkins/test-backend.sh'
```

```
        }
        post {
          always {
            junit 'target/surefire-reports/**/TEST*.xml'
          }
        }
      }
      stage('Frontend Java 7') {
        agent {
          node {
            label 'java7'
          }
        }
        steps {
          unstash 'Java 7'
          sh './jenkins/test-frontend.sh'
        }
        post {
          always {
            junit 'target/test-results/**/TEST*.xml'
          }
        }
      }
      stage('Performance Java 7') {
        agent {
          node {
            label 'java7'
          }
        }
        steps {
          unstash 'Java 7'
          sh './jenkins/test-performance.sh'
        }
      }
      stage('Static Java 7') {
        agent {
          node {
            label 'java7'
          }
        }
        steps {
          unstash 'Java 7'
          sh './jenkins/test-static.sh'
        }
      }
    }
  }
  stage('Confirm Deploy') {
    when {
      branch 'master'
    }
    steps {
      timeout(time: 3, unit: 'MINUTES' ) {
        input(message: "Okay to Deploy to Staging?", ok: "Let's Do it!")
      }
    }
```

```
    }
    stage('Fluffy Deploy') {
      when {
        branch 'master'
      }
      agent {
        node {
          label 'java7'
        }
      }
      steps {
        unstash 'Java 7'
        sh "./jenkins/deploy.sh ${params.DEPLOY_TO}"
      }
    }
  }
  parameters {
    string(name: 'DEPLOY_TO', defaultValue: 'dev', description: '')
  }
}
```

Configure a Global Pipeline Library

In the exercise for this section you will:

- Configure a Global Pipeline Library
- Create a simple Jenkinsfile to verify that the library is set up correctly

Task: Configure a Global Pipeline Library

- Click on `Manage Jenkins` in the left navigation bar
- Click on `Configure System`
- Scroll down to `Global Pipeline Libraries`
- Click on the `Add` button
- Set the following values:
  - Name: `shared-library`
  - Default version: `master`
  - Load implicitly: unchecked
  - Allow default version to be overridden: checked
  - Include @Library changes in job recent changes: checked
- Under `Retrieval method`, click on `Modern SCM`
- Select the radio button for `Git` and enter/select the following values:
  - Project Repository: http://localhost:5000/gitserver/butler/shared-library
  - Credentials: butler

- Click Save

Task: Create a simple Jenkinsfile to verify that the library is set up correctly

- Click on `New Item`
- Enter the item name `test-shared-library` and select `Pipeline`
- Click `OK`
- Scroll down to the Pipeline text area and paste the following in:

```
@Library('shared-library') _
pipeline {
    agent { label 'java' }
    stages {
        stage('verify') {
            steps {
                helloWorld(name: 'fred')
            }
        }
    }
}
```

- Click `Save`
- Click `Build Now`
- Click on the Blue Ball to the left hand side of the #1
- Scroll down and verify that you see

```
Hello world, fred
```

Create a Simple Custom Step

In the exercise for this section you will:

- Create a custom step
- Modify existing Pipeline to use the custom step

Task: Create Custom Step

For this task, we will use the Gitea editor to create the custom step.

- Navigate to the `shared-library` Gitea repository
- Make sure the Branch is set to `master`

- Click on the `vars` directory
- Click on `New File` in the upper right hand corner
- Name the file `postBuildSuccess.groovy`
- Paste in the following content

vars/postBuildSuccess.groovy

```
def call(Map config = [:]) {
    archiveArtifacts 'target/*.jar'
    stash(name: "${config.stashName}", includes: 'target/**')
}
```

- Add a commit message and click `Commit Changes`

Use a Custom Step

In the exercise for this section you will:

- Modify existing Pipeline to use the custom step from the previous lab

Task: Modify existing Pipeline to use the custom step

For this task, we will use the Gitea editor to modify the existing Pipeline.

- Navigate to the `pipeline-lab` Gitea repository
- Make sure the Branch is set to `master`
- Click on `Jenkinsfile`
- Click on the pencil in the upper right hand corner to enter edit mode
- Add the following line at the top of the file

```
@Library('shared-library') _
```

- Scroll down to the `post { success … } }` section of the **Build Java 7** stage
- Replace

```
archiveArtifacts 'target/*.jar'
stash(name: 'Java 7', includes: 'target/**')
```

with

```
postBuildSuccess(stashName: "Java 7")
```

Scroll to the bottom and click on `Commit Changes`

- The job should automatically start once the commit has completed.

Solution

Jenkinsfile

```
@Library('shared-library') _
pipeline {
  agent none
  stages {
    stage('Fluffy Build') {
      parallel {
        stage('Build Java 8') {
          agent {
            node {
              label 'java8'
            }
          }
          steps {
            sh "./jenkins/build.sh"
          }
          post {
            success {
              stash(name: 'Java 8', includes: 'target/**')
            }
          }
        }
        stage('Build Java 7') {
          agent {
            node {
              label 'java7'
            }
          }
          steps {
            sh './jenkins/build.sh'
          }
          post {
            success {
              postBuildSuccess(stashName: "Java 7")
            }
          }
        }
      }
    }
    stage('Fluffy Test') {
      parallel {
        stage('Backend Java 8') {
          agent {
            node {
              label 'java8'
            }
          }
          steps {
            unstash 'Java 8'
            sh './jenkins/test-backend.sh'
```

```
    }
    post {
      always {
        junit 'target/surefire-reports/**/TEST*.xml'
      }
    }
  }
  stage('Frontend') {
    agent {
      node {
        label 'java8'
      }
    }
    steps {
      unstash 'Java 8'
      sh './jenkins/test-frontend.sh'
    }
    post {
      always {
        junit 'target/test-results/**/TEST*.xml'
      }
    }
  }
  stage('Performance Java 8') {
    agent {
      node {
        label 'java8'
      }
    }
    steps {
      unstash 'Java 8'
      sh './jenkins/test-performance.sh'
    }
  }
  stage('Static Java 8') {
    agent {
      node {
        label 'java8'
      }
    }
    steps {
      unstash 'Java 8'
      sh './jenkins/test-static.sh'
    }
  }
  stage('Backend Java 7') {
    agent {
      node {
        label 'java7'
      }
    }
    steps {
      unstash 'Java 7'
      sh './jenkins/test-backend.sh'
    }
    post {
```

```
            always {
              junit 'target/surefire-reports/**/TEST*.xml'
            }
          }
        }
        stage('Frontend Java 7') {
          agent {
            node {
              label 'java7'
            }
          }
          steps {
            unstash 'Java 7'
            sh './jenkins/test-frontend.sh'
          }
          post {
            always {
              junit 'target/test-results/**/TEST*.xml'
            }
          }
        }
        stage('Performance Java 7') {
          agent {
            node {
              label 'java7'
            }
          }
          steps {
            unstash 'Java 7'
            sh './jenkins/test-performance.sh'
          }
        }
        stage('Static Java 7') {
          agent {
            node {
              label 'java7'
            }
          }
          steps {
            unstash 'Java 7'
            sh './jenkins/test-static.sh'
          }
        }
      }
    }
    stage('Confirm Deploy') {
      when {
        branch 'master'
      }
      steps {
        timeout(time: 3, unit: 'MINUTES' ) {
          input(message: "Okay to Deploy to Staging?", ok: "Let's Do it!")
        }
      }
    }
    stage('Fluffy Deploy') {
```

```
        when {
          branch 'master'
        }
        agent {
          node {
            label 'java7'
          }
        }
        steps {
          unstash 'Java 7'
          sh "./jenkins/deploy.sh ${params.DEPLOY_TO}"
        }
      }
    }
    parameters {
      string(name: 'DEPLOY_TO', defaultValue: 'dev', description: '')
    }
}
```

Create and Use a Resource File

In the exercise for this section you will:

- Create a `resources` file
- Create a custom step that loads the `resources` file
- Modify existing Pipeline to use the custom step

Task: Create a `resources` file

For this task, we will copy the contents of jenkins/build.sh from pipeline-lab to resources/scripts/build.sh in the shared-library repository.

- Navigate to the `pipeline-lab` Gitea repository
- Click on the `jenkins` directory
- Copy the contents of `build.sh` to the clipboard
- Navigate to the `shared-library` Gitea repository
- Click on `New File` in the upper right hand corner
- Enter `resources/scripts/build.sh` in the "Name your file..." input field
- Paste in the `build.sh` contents from the clipboard
- Add a commit message and click `Commit Changes`

Task: Create a custom step that loads the `resources` file

For this task, we will use the Gitea editor to create the custom step.

- Navigate to the `shared-library` Gitea repository

- Click on the `vars` directory
- Click on `New File` in the upper right hand corner
- Name the file `runLinuxScript.groovy`
- Paste in the following content

```
def call(Map config = [:]) {
  def scriptcontents = libraryResource "scripts/${config.name}"
  writeFile file: "${config.name}", text: scriptcontents
  sh """
    chmod a+x ./${config.name}
    ./${config.name}
  """
}
```

- Add a commit message and click `Commit Changes`

Task: Modify existing Pipeline to use the custom step

- Navigate to the `pipeline-lab` Gitea repository
- Click on `Jenkinsfile`
- Click on the pencil in the upper right hand corner to enter edit mode
- Replace

```
sh './jenkins/build.sh'
```

with

```
runLinuxScript(name: "build.sh")
```

- Commit the changes. The job should automatically start.
- Verify the job ran successfully.

Solution

Jenkinsfile

```
@Library('shared-library') _
pipeline {
  agent none
  stages {
    stage('Fluffy Build') {
      parallel {
        stage('Build Java 8') {
          agent {
```

```
            node {
              label 'java8'
            }
          }
          steps {
            runLinuxScript(name: "build.sh")
          }
          post {
            success {
              stash(name: 'Java 8', includes: 'target/**')
            }
          }
        }
        stage('Build Java 7') {
          agent {
            node {
              label 'java7'
            }
          }
          steps {
            runLinuxScript(name: "build.sh")
          }
          post {
            success {
              postBuildSuccess(stashName: "Java 7")
            }
          }
        }
      }
    }
    stage('Fluffy Test') {
      parallel {
        stage('Backend Java 8') {
          agent {
            node {
              label 'java8'
            }
          }
          steps {
            unstash 'Java 8'
            sh './jenkins/test-backend.sh'
          }
          post {
            always {
              junit 'target/surefire-reports/**/TEST*.xml'
            }
          }
        }
        stage('Frontend') {
          agent {
            node {
              label 'java8'
            }
          }
          steps {
            unstash 'Java 8'
```

```
        sh './jenkins/test-frontend.sh'
      }
      post {
        always {
          junit 'target/test-results/**/TEST*.xml'
        }
      }
    }
    stage('Performance Java 8') {
      agent {
        node {
          label 'java8'
        }
      }
      steps {
        unstash 'Java 8'
        sh './jenkins/test-performance.sh'
      }
    }
    stage('Static Java 8') {
      agent {
        node {
          label 'java8'
        }
      }
      steps {
        unstash 'Java 8'
        sh './jenkins/test-static.sh'
      }
    }
    stage('Backend Java 7') {
      agent {
        node {
          label 'java7'
        }
      }
      steps {
        unstash 'Java 7'
        sh './jenkins/test-backend.sh'
      }
      post {
        always {
          junit 'target/surefire-reports/**/TEST*.xml'
        }
      }
    }
    stage('Frontend Java 7') {
      agent {
        node {
          label 'java7'
        }
      }
      steps {
        unstash 'Java 7'
        sh './jenkins/test-frontend.sh'
      }
```

```
          post {
            always {
              junit 'target/test-results/**/TEST*.xml'
            }
          }
        }
        stage('Performance Java 7') {
          agent {
            node {
              label 'java7'
            }
          }
          steps {
            unstash 'Java 7'
            sh './jenkins/test-performance.sh'
          }
        }
        stage('Static Java 7') {
          agent {
            node {
              label 'java7'
            }
          }
          steps {
            unstash 'Java 7'
            sh './jenkins/test-static.sh'
          }
        }
      }
    }
    stage('Confirm Deploy') {
      when {
        branch 'master'
      }
      steps {
        timeout(time: 3, unit: 'MINUTES' ) {
          input(message: "Okay to Deploy to Staging?", ok: "Let's Do it!")
        }
      }
    }
    stage('Fluffy Deploy') {
      when {
        branch 'master'
      }
      agent {
        node {
          label 'java7'
        }
      }
      steps {
        unstash 'Java 7'
        sh "./jenkins/deploy.sh ${params.DEPLOY_TO}"
      }
    }
  }
  parameters {
```

```
        string(name: 'DEPLOY_TO', defaultValue: 'dev', description: '')
    }
}
```

Create a corporate pipeline

In the exercise for this section you will:

- Create a custom step
- Modify existing Pipeline to use the custom step

Task: Create Custom Step

For this task, we will use the Gitea editor to create the custom step.

- Navigate to the `shared-library` Gitea repository
- Click on the `vars` directory
- Click on `New File` in the upper right hand corner
- Name the file `corporatePipeline.groovy`
- Paste the following into `corporatePipeline.groovy`:

```
def call(body) {
    def pipelineParams= [:]
    body.resolveStrategy = Closure.DELEGATE_FIRST
    body.delegate = pipelineParams
    body()

    !!!REPLACEME!!!
}
```

- Replace **!!!REPLACEME!!!** with the contents from the `Jenkinsfile` from the `pipeline-lab` repository. Be sure to **not** copy over the `@Library` annotation.
- Add a commit message and click `Commit Changes`

Task: Modify the new custom step to use a parameter passed through corporatePipeline

- Click on `corporatePipeline.groovy`
- Click on the pencil in the upper right hand corner to enter edit mode
- Remove the `parameters` directive
- Change the `${params.DEPLOY_TO}` parameter to `${pipelineParams.deployTo}`
- Add a commit message and click `Commit Changes`

Task: Modify existing Pipeline to use the custom step

For this task, we will use the Gitea editor to modify the existing Pipeline.

- Navigate to the `pipeline-lab` Gitea repository
- Click on `Jenkinsfile`
- Click on the pencil in the upper right hand corner to enter edit mode
- Replace all of the contents of Jenkinsfile with

```
@Library('shared-library') _
corporatePipeline {
    deployTo = "dev"
}
```

- Commit the changes. The job should automatically start.
- Verify the job ran successfully.

Solution

vars/corporatePipeline.groovy

```
def call(body) {
    def pipelineParams= [:]
    body.resolveStrategy = Closure.DELEGATE_FIRST
    body.delegate = pipelineParams
    body()

  pipeline {
    agent none
    stages {
      stage('Fluffy Build') {
        parallel {
          stage('Build Java 8') {
            agent {
              node {
                label 'java8'
              }

            }
            post {
              success {
                stash(name: 'Java 8', includes: 'target/**')

              }

            }
            steps {
              runLinuxScript(name: "build.sh")
            }
          }
          stage('Build Java 7') {
```

```
        agent {
          node {
            label 'java7'
          }

        }
        post {
          success {
            postBuildSuccess(stashName: "Java 7")
          }

        }
        steps {
          runLinuxScript(name: "build.sh")
        }
      }
    }
  }
  stage('Fluffy Test') {
    parallel {
      stage('Backend Java 8') {
        agent {
          node {
            label 'java8'
          }

        }
        post {
          always {
            junit 'target/surefire-reports/**/TEST*.xml'

          }

        }
        steps {
          unstash 'Java 8'
          sh './jenkins/test-backend.sh'
        }
      }
      stage('Frontend') {
        agent {
          node {
            label 'java8'
          }

        }
        post {
          always {
            junit 'target/test-results/**/TEST*.xml'

          }

        }
        steps {
          unstash 'Java 8'
          sh './jenkins/test-frontend.sh'
        }
      }
```

```
        stage('Performance Java 8') {
          agent {
            node {
              label 'java8'
            }

          }
          steps {
            unstash 'Java 8'
            sh './jenkins/test-performance.sh'
          }
        }
        stage('Static Java 8') {
          agent {
            node {
              label 'java8'
            }

          }
          steps {
            unstash 'Java 8'
            sh './jenkins/test-static.sh'
          }
        }
        stage('Backend Java 7') {
          agent {
            node {
              label 'java7'
            }

          }
          post {
            always {
              junit 'target/surefire-reports/**/TEST*.xml'

            }

          }
          steps {
            unstash 'Java 7'
            sh './jenkins/test-backend.sh'
          }
        }
        stage('Frontend Java 7') {
          agent {
            node {
              label 'java7'
            }

          }
          post {
            always {
              junit 'target/test-results/**/TEST*.xml'

            }

          }
          steps {
```

```
            unstash 'Java 7'
            sh './jenkins/test-frontend.sh'
          }
        }
        stage('Performance Java 7') {
          agent {
            node {
              label 'java7'
            }

          }
          steps {
            unstash 'Java 7'
            sh './jenkins/test-performance.sh'
          }
        }
        stage('Static Java 7') {
          agent {
            node {
              label 'java7'
            }

          }
          steps {
            unstash 'Java 7'
            sh './jenkins/test-static.sh'
          }
        }
      }
    }
    stage('Confirm Deploy') {
      when {
        branch 'master'
      }
      steps {
        timeout(time: 3, unit: 'MINUTES') {
          input(message: 'Okay to Deploy to Staging?', ok: 'Let\'s Do it!')
        }

      }
    }
    stage('Fluffy Deploy') {
      agent {
        node {
          label 'java7'
        }

      }
      when {
        branch 'master'
      }
      steps {
        unstash 'Java 7'
        sh "./jenkins/deploy.sh ${pipelineParams.deployTo}"
      }
    }
  }
}
```

```
}
```

Durability

In the exercise for this section you will:

- Modify the existing custom step to override the global durability value

Task: Review the logs from a previous job run

- Using Classic view, review the most recent log from a successful master branch run.
- Search for `Running in Durability level`. You should see a value of `PERFORMANCE_OPTIMIZED`.

Task: Modify corporatePipeline to use a different durability option

- Navigate to the `shared-library` Gitea repository.
- Click on the `vars` directory.
- Click on `corporatePipeline.groovy`.
- Click on the pencil in the upper right hand corner to enter edit mode.
- Using the information from the slides, add an `option` to set the durability to the default value.
- Commit the change.
- Back in Jenkins, start a build from the master branch of the `pipeline-lab` job.
- Verify that the job ran successfully.

Task: Review the log

- Using Classic view, review the most recent log from a successful master branch run.
- Search the log for `Running in Durability level`. You should see a value of `PERFORMANCE_OPTIMIZED`.
  - This is a known issue much like the parameters issue we saw in the Blue Ocean Refresher lab.
- Start another build from the master branch of the `pipeline-lab` job.
- Verify that the job ran successfully.
- Search the most recent log for `Running in Durability level`. You should see the value that you set.

Solution

vars/corporatePipeline.groovy

```
def call(body) {
  def pipelineParams = [:]
  body.resolveStrategy = Closure.DELEGATE_FIRST
```

```
body.delegate = pipelineParams
body()

pipeline {
  agent none
  stages {
    stage('Fluffy Build') {
      parallel {
        stage('Build Java 8') {
          agent {
            node {
              label 'java8'
            }
          }
          post {
            success {
              stash(name: 'Java 8', includes: 'target/**')
            }
          }
          steps {
            runLinuxScript(name: "build.sh")
          }
        }
        stage('Build Java 7') {
          agent {
            node {
              label 'java7'
            }
          }
          post {
            success {
              postBuildSuccess(stashName: "Java 7")
            }
          }
          steps {
            runLinuxScript(name: "build.sh")
          }
        }
      }
    }
    stage('Fluffy Test') {
      parallel {
        stage('Backend Java 8') {
          agent {
            node {
              label 'java8'
            }
          }
          post {
            always {
              junit 'target/surefire-reports/**/TEST*.xml'
            }
          }
          steps {
            unstash 'Java 8'
            sh './jenkins/test-backend.sh'
          }
        }
```

```
    stage('Frontend') {
      agent {
        node {
          label 'java8'
        }
      }
      post {
        always {
          junit 'target/test-results/**/TEST*.xml'
        }
      }
      steps {
        unstash 'Java 8'
        sh './jenkins/test-frontend.sh'
      }
    }
    stage('Performance Java 8') {
      agent {
        node {
          label 'java8'
        }
      }
      steps {
        unstash 'Java 8'
        sh './jenkins/test-performance.sh'
      }
    }
    stage('Static Java 8') {
      agent {
        node {
          label 'java8'
        }
      }
      steps {
        unstash 'Java 8'
        sh './jenkins/test-static.sh'
      }
    }
    stage('Backend Java 7') {
      agent {
        node {
          label 'java7'
        }
      }
      post {
        always {
          junit 'target/surefire-reports/**/TEST*.xml'
        }
      }
      steps {
        unstash 'Java 7'
        sh './jenkins/test-backend.sh'
      }
    }
    stage('Frontend Java 7') {
      agent {
        node {
          label 'java7'
```

```
            }
          }
          post {
            always {
              junit 'target/test-results/**/TEST*.xml'
            }
          }
          steps {
            unstash 'Java 7'
            sh './jenkins/test-frontend.sh'
          }
        }
        stage('Performance Java 7') {
          agent {
            node {
              label 'java7'
            }
          }
          steps {
            unstash 'Java 7'
            sh './jenkins/test-performance.sh'
          }
        }
        stage('Static Java 7') {
          agent {
            node {
              label 'java7'
            }
          }
          steps {
            unstash 'Java 7'
            sh './jenkins/test-static.sh'
          }
        }
      }
    }
    stage('Confirm Deploy') {
      when {
        branch 'master'
      }
      steps {
        timeout(time: 3, unit: 'MINUTES') {
          input(message: 'Okay to Deploy to Staging?', ok: 'Let\'s Do it!')
        }
      }
    }
    stage('Fluffy Deploy') {
      agent {
        node {
          label 'java7'
        }
      }
      when {
        branch 'master'
      }
      steps {
        unstash 'Java 7'
        sh "./jenkins/deploy.sh ${pipelineParams.deployTo}"
```

```
        }
      }
    }
    options {
      durabilityHint('MAX_SURVIVABILITY')
    }
  }
}
```

Sequential Stages

In the exercise for this section you will:

- Modify existing custom step to use sequential stages

Task: Make a copy of the corporatePipeline

- Navigate to the `shared-library` Gitea repository
- Click on the `vars` directory
- Click on `corporatePipeline.groovy`
- Copy the contents of the file
- Click on the `vars` directory
- Create a new file
- Name it `corporatePipelineSequential.groovy`
- Paste the contents into the body of the text editor
- Commit the changes to the `master` branch

Task: Create Sequential Stages for Java 7 and Java 8

- Click on the pencil in the upper right hand corner of `corporatePipelineSequential.groovy` to enter edit mode
- Modify the pipeline to run parallel sequential stages to replace the existing `Fluffy Build` and `Fluffy Test` stages. There should be one sequential stage for `Java 7` and one sequential stage for `Java 8`.
- Commit the changes to `corporatePipelineSequential.groovy`
- Open the `Jenkinsfile` in the `pipeline-lab` repo and change `corporatePipeline` to `corporatePipelineSequential`.
- Commit the changes to the `master` branch
- The job should start automatically on the `master` branch
- Verify the job completed successfully
- Go take a look at the job visualization using Blue Ocean. Notice the difference from prior runs.

Solution

vars/corporatePipelineSequential.groovy

```
def call(body) {
  def pipelineParams= [:]
  body.resolveStrategy = Closure.DELEGATE_FIRST
  body.delegate = pipelineParams
  body()

  pipeline {
    agent none
    stages {
      stage('Build and Test Java') {
        parallel {
          stage('java8') {
            agent { label 'java8' }
            stages {
              stage("build8") {
                steps {
                  runLinuxScript(name: "build.sh")
                }
                post {
                  success {
                    stash(name: 'Java 8', includes: 'target/**')
                  }
                }
              }
              stage('Backend Java 8') {
                steps {
                  unstash 'Java 8'
                  sh './jenkins/test-backend.sh'
                }
                post {
                  always {
                    junit 'target/surefire-reports/**/TEST*.xml'
                  }
                }
              }
              stage('Frontend') {
                steps {
                  unstash 'Java 8'
                  sh './jenkins/test-frontend.sh'
                }
                post {
                  always {
                    junit 'target/test-results/**/TEST*.xml'
                  }
                }
              }
              stage('Performance Java 8') {
                steps {
                  unstash 'Java 8'
                  sh './jenkins/test-performance.sh'
                }
              }
              stage('Static Java 8') {
```

```
            steps {
              unstash 'Java 8'
              sh './jenkins/test-static.sh'
            }
          }
        }
      }
      stage('java7') {
        agent { label 'java7' }
        stages {
          stage("build7") {
            steps {
              runLinuxScript(name: "build.sh")
            }
            post {
              success {
                postBuildSuccess(stashName: "Java 7")
              }
            }
          }
          stage('Backend Java 7') {
            steps {
              unstash 'Java 7'
              sh './jenkins/test-backend.sh'
            }
            post {
              always {
                junit 'target/surefire-reports/**/TEST*.xml'
              }
            }
          }
          stage('Frontend Java 7') {
            steps {
              unstash 'Java 7'
              sh './jenkins/test-frontend.sh'
            }
            post {
              always {
                junit 'target/test-results/**/TEST*.xml'
              }
            }
          }
          stage('Performance Java 7') {
            steps {
              unstash 'Java 7'
              sh './jenkins/test-performance.sh'
            }
          }
          stage('Static Java 7') {
            steps {
              unstash 'Java 7'
              sh './jenkins/test-static.sh'
            }
          }
        }
      }
    }
  }
```

```
        stage('Confirm Deploy') {
          when { branch 'master' }
          steps {
            timeout(time: 3, unit: 'MINUTES') {
              input(message: 'Okay to Deploy to Staging?', ok: 'Let\'s Do it!')
            }
          }
        }
        stage('Fluffy Deploy') {
          agent { label 'java7' }
          when { branch 'master' }
          steps {
            unstash 'Java 7'
            sh "./jenkins/deploy.sh ${pipelineParams.deployTo}"
          }
        }
      }
      options {
        durabilityHint('MAX_SURVIVABILITY')
      }
    }
}
```

Restart Stage

In the exercise for this section you will:

- Modify existing custom step to use be able to successfully restart a stage

Task: Modify the companyPipelineSequential step

- Navigate to the `shared-library` Gitea repository
- Click on the `vars` directory
- Click on `corporatePipelineSequential.groovy`
- Click on the pencil in the upper right hand corner to enter edit mode
- Add an option to retain the last 5 stashes
- Commit the changes

Task: Run the job from the Classic UI

- Open the job run from the master branch in the Classic UI
- Click on `Build Now`
- Click on the progress bar for the job in the left nav to open the scrolling console log
- Wait for the input stage and then click on `Abort`
- The job should finish in an `ABORTED` state

Task: Restart the stage from the Classic UI

- In the breadcrumb at the top of the page, click on the job number that you just aborted
- On the left nav, click on `Restart from Stage`
- From the dropdown, select `Confirm Deploy`
- Click `Run`
- Click on the progress bar for the job in the left nav to open the scrolling console log
- Wait for the input stage and then click on `Let's Do It!`
- The job should complete successfully

Task: Restart the stage from Blue Ocean

- In the breadcrumb at the top of the page:
  - make note of the job number
  - click on `master`
- Click on the `Open Blue Ocean` link in the left nav
- Click on the `Activity` tab on the right hand side of the screen
- Click on the line that has the run number that matches the job number for the `master` branch
- Click on the `Confirm Deploy` green ball in the pipeline
- Click on the blue `Restart Confirm Deploy` link in the lower right hand corner

> **NOTE** You may have to refresh the page (Cmd+R/Ctrl+R) in order to see the correct rendering. This is known issue with Blue Ocean and Restart from Stage.

- Click on `Let's Do It!`
- The job should complete successfully

Solution

vars/corporatePipelineSequential.groovy

```
def call(body) {
  def pipelineParams= [:]
  body.resolveStrategy = Closure.DELEGATE_FIRST
  body.delegate = pipelineParams
  body()

  pipeline {
    agent none
    stages {
      stage('Build and Test Java') {
        parallel {
          stage('java8') {
            agent { label 'java8' }
```

```
    stages {
      stage("build8") {
        steps {
          runLinuxScript(name: "build.sh")
        }
        post {
          success {
            stash(name: 'Java 8', includes: 'target/**')
          }
        }
      }
      stage('Backend Java 8') {
        steps {
          unstash 'Java 8'
          sh './jenkins/test-backend.sh'
        }
        post {
          always {
            junit 'target/surefire-reports/**/TEST*.xml'
          }
        }
      }
      stage('Frontend') {
        steps {
          unstash 'Java 8'
          sh './jenkins/test-frontend.sh'
        }
        post {
          always {
            junit 'target/test-results/**/TEST*.xml'
          }
        }
      }
      stage('Performance Java 8') {
        steps {
          unstash 'Java 8'
          sh './jenkins/test-performance.sh'
        }
      }
      stage('Static Java 8') {
        steps {
          unstash 'Java 8'
          sh './jenkins/test-static.sh'
        }
      }
    }
  }
  stage('java7') {
    agent { label 'java7' }
    stages {
      stage("build7") {
        steps {
          runLinuxScript(name: "build.sh")
        }
        post {
          success {
            postBuildSuccess(stashName: "Java 7")
          }
```

```
            }
          }
          stage('Backend Java 7') {
            steps {
              unstash 'Java 7'
              sh './jenkins/test-backend.sh'
            }
            post {
              always {
                junit 'target/surefire-reports/**/TEST*.xml'
              }
            }
          }
          stage('Frontend Java 7') {
            steps {
              unstash 'Java 7'
              sh './jenkins/test-frontend.sh'
            }
            post {
              always {
                junit 'target/test-results/**/TEST*.xml'
              }
            }
          }
          stage('Performance Java 7') {
            steps {
              unstash 'Java 7'
              sh './jenkins/test-performance.sh'
            }
          }
          stage('Static Java 7') {
            steps {
              unstash 'Java 7'
              sh './jenkins/test-static.sh'
            }
          }
        }
      }
    }
    stage('Confirm Deploy') {
      when { branch 'master' }
      steps {
        timeout(time: 3, unit: 'MINUTES') {
          input(message: 'Okay to Deploy to Staging?', ok: 'Let\'s Do it!')
        }
      }
    }
    stage('Fluffy Deploy') {
      agent { label 'java7' }
      when { branch 'master' }
      steps {
        unstash 'Java 7'
        sh "./jenkins/deploy.sh ${pipelineParams.deployTo}"
      }
    }
  }
  options {
```

```
        durabilityHint('MAX_SURVIVABILITY')
        preserveStashes(buildCount: 5)
      }
    }
  }
```