



# Introduction to Declarative Pipeline

In the beginning there was  
the Freestyle job...





freestylingit Config [Jenkins]

Craig

05202017259.jenkins.beedemo.net/job/freestylingit/configure

Managed Master

Open Blue Ocean

search

cvitter | log out

Jenkins05202017259freestylingit

GeneralSource Code ManagementBuild TriggersBuild EnvironmentBuildPost-build Actions

Project namefreestylingit

Description

[Plain text] Preview

☐ Discard old builds

☐ GitHub project

☐ This project is parameterized

☐ Throttle builds

☐ Disable this project

☐ Execute concurrent builds if necessary

☒ Restrict where this project can be run

Label Expression

Advanced...

Source Code Management

None

Git

Save

Apply

cloudbees

© 2017 CloudBees, Inc. All Rights Reserved.

3

# What's Wrong With Freestyle Jobs?

While the Freestyle job type has served the Hudson/Jenkins community well for years it has some major issues including:

- **UI Bound** - The configuration of a job is limited to what can be expressed via the limits of the Jenkins' UI and doesn't allow for building complicated workflows with features like:
  - Control over where builds are executed
  - Flow control (if-then-else, when, try-catch-finally)
  - Ability to run steps in parallel
- **Not Auditable** - The creation and editing of jobs isn't auditable without using additional plugins
- **Too Many Jobs** - Freestyle pipelines are difficult to maintain

# Enter Jenkins Pipeline...



# What is a Jenkins Pipeline?

Jenkins Pipeline (formerly known as Workflow) was introduced in **2014** and built into Jenkins 2.0 when it was released.

Pipelines are:

- A **Job** type - The configuration of the job and steps to execute are defined in a script (**Groovy** or **Declarative** based with a Domain Specific Language) that can be stored in an external SCM
- **Auditable** - changes can be audited via your SCM
- **Durable** - can keep running even if the master fails
- **Distributable** - pipelines can be run across multiple agents including execution of steps in parallel
- **Pausable** - can wait for user input before proceeding
- **Logic** - Flow control can be added to your pipelines
- **Visualizable** - enables status-at-a-glance dashboards like the built in Pipeline Stage View and Blue Ocean

# Pipeline is Awesome but...

```
if (env.BRANCH_NAME == "master"){
    /Something
} else {
    //Something else
}
library "lib@${env.BRANCH_NAME}"
node(){
    sh "mkdir ${Workspace}"
    dir(Workspace){
        checkout([$class: 'GitSCM', branches: [[name: "*/${env.BRANCH_NAME}"]],
            doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [],
            userRemoteConfigs: [[credentialsId: 'git', url:ssh://git@git.example.com/pipeline-sharedlib.git]]])
        try{
            //This
        }catch(Exception e){
            //Something went wrong
            throw e
        }finally{
            //Clean it all up
        }
    }
}
```

# Why You Should Use Declarative Instead of Scripted

While Declarative Pipelines use the same execution engine as Scripted pipelines Declarative adds the following benefits:

- **Easier to Learn** - the Pipeline DSL (Domain Specific Language) is more approachable than Groovy making it quicker to get started
- **Docker Pipeline Integration** - ability to execute builds within one or more docker containers is built into Declarative
- **Syntax Checking** - Declarative syntax adds the following types of syntax checking that don't exist for Scripted pipelines:
  - Immediate runtime syntax checking with explicit error messages.
  - API and CLI based file linting
- **Round Trip Visual Editing** - The Blue Ocean pipeline editor can read and write Declarative syntax (but not Scripted)



# Creating a Pipeline



# The Simplest Declarative Jenkins File vs Scripted

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

```
node {  
  stage 'Say Hello'  
    echo 'Hello World!'  
}
```

# Hands On Exercise 1.0

---

## Setup Workshop Workspace

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-01.md#exercise-10>

# Hands On Exercise 1.1

---

## Create a Simple Declarative Pipeline

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-01.md#exercise-11>



# The Jenkins Snippet Generator

The screenshot shows the Jenkins Snippet Generator web interface. The browser address bar indicates the URL is `localhost:8080/job/BasicPipeline/pipeline-syntax/`. The page header includes the CloudBees Jenkins Enterprise logo, an "Open Blue Ocean" button, a notification badge with the number "1", a search bar, and links for "admin" and "log out". The breadcrumb trail shows the path: Jenkins > BasicPipeline > Pipeline Syntax.

On the left sidebar, there are links for "Back", "Snippet Generator" (active), "Step Reference", "Global Variables Reference", "Online Documentation", and "IntelliJ IDEA GDSL".

The main content area is titled "Overview" and contains the following text: "This **Snippet Generator** will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click **Generate Pipeline Script**, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)"

Below the overview is a "Steps" section with a dropdown menu labeled "Sample Step" showing "stash: Stash some files to be used later in the build".

Configuration fields include:

- Name:** A text input field containing "test.lib".
- Includes:** An empty text input field.
- Excludes:** An empty text input field.
- Use Default Ant Excludes:** A checked checkbox.

A "Generate Pipeline Script" button is located below the configuration fields. Below this button, a text area displays the generated Pipeline Script snippet: `stash 'test.lib'`.

The footer of the page shows the URL `localhost:8080/job/BasicPipeline/pipeline-syntax/globals` and the heading "Global Variables".

# Pipeline Replay

The screenshot shows the CloudBees Jenkins Enterprise web interface. The browser address bar indicates the URL `localhost:8080/job/BasicPipeline/13/replay/`. The page title is "Replay #13 [Jenkins]". The main header bar includes the CloudBees logo, the text "CloudBees Jenkins Enterprise", a button "Open Blue Ocean", a red notification badge with the number "1", a search bar, and links for "admin" and "log out". The breadcrumb navigation shows "Jenkins > BasicPipeline > #13 > Replay".

On the left sidebar, there are several navigation links: "Back to Project", "Status", "Changes", "Console Output", "Edit Build Information", "Delete Build", "Replay" (highlighted with a green arrow), "Pipeline Steps", and "Previous Build".

The main content area is titled "Replay #13" and includes a description: "Allows you to replay a Pipeline build with a modified script. If any load steps were run, you can also modify the scripts they loaded." Below this, there is a section for the "Main Script" with a line-numbered code editor. The script is a Jenkins Pipeline script:

```
1 pipeline {  
2   agent any  
3  
4   stages {  
5     stage('Say Hello') {  
6       steps {  
7         echo 'Hello World!'  
8       }  
9       post {  
10        always {  
11          echo "Running ${env.JOB_NAME} (${env.BUILD_ID}) on ${env.JENKINS_URL}"  
12        }  
13      }  
14    }  
15  }  
--
```

Below the code editor, there is a link "Pipeline Syntax" and a blue "Run" button.

The footer of the page contains the text "CloudBees Jenkins Platform" on the left, "Page generated: May 30, 2017 4:57:06 PM" in the center, and "CloudBees Jenkins Enterprise 2.46.2.1-rolling" on the right.

# Blue Ocean Editor

Create Pipeline

Where do you store your code?

Github

Github Enterprise

Bitbucket

Bitbucket Server

Git

Connect to Github

Jenkins needs a access key to authorize itself with Github. [Learn how to create an access key.](#)

Connect

In which Github organization are your repositories located?

Build any repository that contains a Jenkinsfile?

Completed

Create Pipeline

Discard Changes

Save

Start

Build

Build

Sit Around

+

Test

Test

Browser test

+

Deploy

+

cloudbees

© 2017 CloudBees, Inc. All Rights Reserved.

15

# Beyond Hello World





# Specifying Agents

- **agent** keyword can be used for an entire pipeline or within a **stage**
- **agent none** specifies that no agent will be used. Used primarily when stage definition is used
- **agent any** specifies that any agent will do
- **docker** {} should use *Pipeline Model Definition* or **label** syntax

```
pipeline {  
  agent {  
    docker {  
      image 'maven:3.3-jdk-8'  
      label 'dockerd'  
    }  
  }  
  stages { ... }  
}
```

```
pipeline {  
  agent any  
  stages { ... }  
}
```

```
pipeline {  
  agent none  
  stages {  
    stage('Build') {  
      agent { label 'java && maven' }  
      steps {  
        sh '''  
          mvn clean  
          mvn package  
          mvn verify  
          '''  
      }  
    }  
  }  
}
```

# Hands On Exercise 1.2

---

## Define a Docker Based Agent

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-01.md#exercise-12>

# Environmental Variables

```
pipeline {
  agent any

  environment {
    A_VALUE = 'Some Value'
  }

  stages {
    stage('Build') {
      steps {
        echo "${A_VALUE}"
        echo "${env.BUILD_ID}"
        echo "${currentBuild.result}"
      }
    }
  }
}
```

```
pipeline {
  agent any

  environment {
    SONAR = credentials('sonar')
  }

  stages {
    stage('Build') {
      steps {
        echo "${SONAR_USR}"
        echo "${SONAR_PSW}"
      }
    }
  }
}
```

<http://localhost:8080/job/BasicPipeline/pipeline-syntax/globals>

# Environmental Variables

```
pipeline {
  agent any
  environment {
    A_VALUE = 'Some Value'
  }
  stages {
    stage('1') {
      environment { A_VALUE = 'Changed' }
      steps {
        echo "${A_VALUE}" //stage scope
      }
    }
    stage('2') {
      steps {
        echo "${A_VALUE}" //pipeline scope
      }
    }
  }
}
```



# Credentials

```
pipeline {
  agent any

  environment {
    SONAR = credentials('sonar')
  }

  stages {
    stage('Build') {
      steps {
        echo "${SONAR_USR}"
        echo "${SONAR_PSW}"
      }
    }
  }
}
```

# Hands On Exercise 1.3

---

## Add Environment Variables

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-01.md#exercise-13>

# Parameters

```
pipeline {
  agent any

  parameters {
    string(name: 'Greeting', defaultValue: 'Hello',
           description: 'How should I greet the world?')
  }

  stages {
    stage('Example') {
      steps {
        echo "${params.Greeting} World!"
      }
    }
  }
}
```

# Hands On Exercise 1.4

---

## Capture Input Parameters

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-01.md#exercise-14>



# Capturing User Input

```
stage('Deploy') {  
  input {  
    message "Should we continue?"  
  }  
  steps {  
    echo "Continuing with deployment"  
  }  
}
```

```
stage('Input') {  
  input {  
    message "Need some input"  
    parameters {  
      string(name: 'PARAM1', defaultValue: '')  
    }  
  }  
  agent any  
  steps {  
    echo "${PARAM1}"  
  }  
}
```

Steps - Deploy

Wait for interactive input 1m 54s

Should I Deploy?

Proceed Abort

Steps - Input

Wait for interactive input 41s

Need some input

Proceed Abort

# Retry, Timeout, and Sleep

```
stage('Deploy') {  
  steps {  
    retry(3) {  
      sh './flakey-deploy.sh'  
    }  
  
    timeout(time: 3, unit: 'MINUTES') {  
      sh './health-check.sh'  
    }  
  }  
}
```

```
stage('Deploy') {  
  steps {  
    sleep time: 15, unit: 'SECONDS'  
  }  
}
```

```
stage('Deploy') {  
  steps {  
    timeout(time: 3, unit: 'MINUTES') {  
      retry(5) {  
        sh './flakey-deploy.sh'  
      }  
    }  
  }  
}
```

# Hands On Exercise 1.5

---

## Capture User Input During Run Time

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-01.md#exercise-15>

# Hands On Exercise 1.6

---

More User Input at Run Time

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-01.md#exercise-16>

# Post Actions

```
pipeline {
  agent any

  stages { ... }
  post {
    always {
      echo 'I always run!'
    }
    success { ... }
    failure { ... }
    aborted { ... }
    unstable { ... }
    changed { ... }
  }
}
```

```
pipeline {
  agent any

  stages {
    stage('Build') {
      steps {
      }
      post {
        always {
          echo 'I always run!'
        }
        success { ... }
      }
    }
  }
}
```

# Hands On Exercise 1.7

---

## Handling Post Actions

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-01.md#exercise-16>



# Script Block

```
stage('Get Kernel') {  
    steps {  
        script {  
            try {  
                KERNEL_VERSION = sh (script: "uname -r", returnStdout: true)  
            } catch (err) {  
                echo "CAUGHT ERROR: ${err}"  
                throw err  
            }  
        }  
    }  
}
```

# Hands On Exercise 1.8

---

## Script Block

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-01.md#exercise-18>

# Shared Libraries

```
// Groovy Library located in
// github.com/example/CraigsLibs/vars/helloWorld.groovy
def call(name) {
    echo "Hello ${name}"
    echo "Have a great day!"
}
```

The screenshot shows the Jenkins configuration page for a shared library named 'CraigsLibs'. The 'Name' field is 'CraigsLibs' and the 'Default version' is 'master'. Below this, it states 'Currently maps to revision: f45ffa4f941692e971fbb9618b6034174ed60a1e'. The 'Load implicitly' checkbox is checked. Under 'Retrieval method', 'Modern SCM' is selected. In the 'Source Code Management' section, 'GitHub' is selected as the provider. The 'Owner' is 'cvitter', the 'Scan credentials' is 'cvitter/\*\*\*\*\* (GitHub Token)', and the 'Repository' is 'jenkins-pipeline-examples'. There are 'Advanced...' and 'Delete' buttons at the bottom.

Library	
Name	CraigsLibs
Default version	master
Currently maps to revision: f45ffa4f941692e971fbb9618b6034174ed60a1e	
Load implicitly	<input checked="" type="checkbox"/>
Allow default version to be overridden	<input checked="" type="checkbox"/>
Retrieval method	
<input checked="" type="radio"/> Modern SCM	
Source Code Management	
<input type="radio"/> Git	
<input checked="" type="radio"/> GitHub	
Owner	cvitter
Scan credentials	cvitter/***** (GitHub Token) <span>Add</span>
Repository	jenkins-pipeline-examples
<input type="radio"/> Legacy SCM	
<span>Advanced...</span>	
<span>Delete</span>	

```
library 'CraigsLibs'

pipeline {
    agent any
    stages {
        stage('Example') {
            steps {
                helloWorld("Bob")
            }
        }
    }
}
```

# Hands On Exercise 1.9

---

## Using Shared Libraries

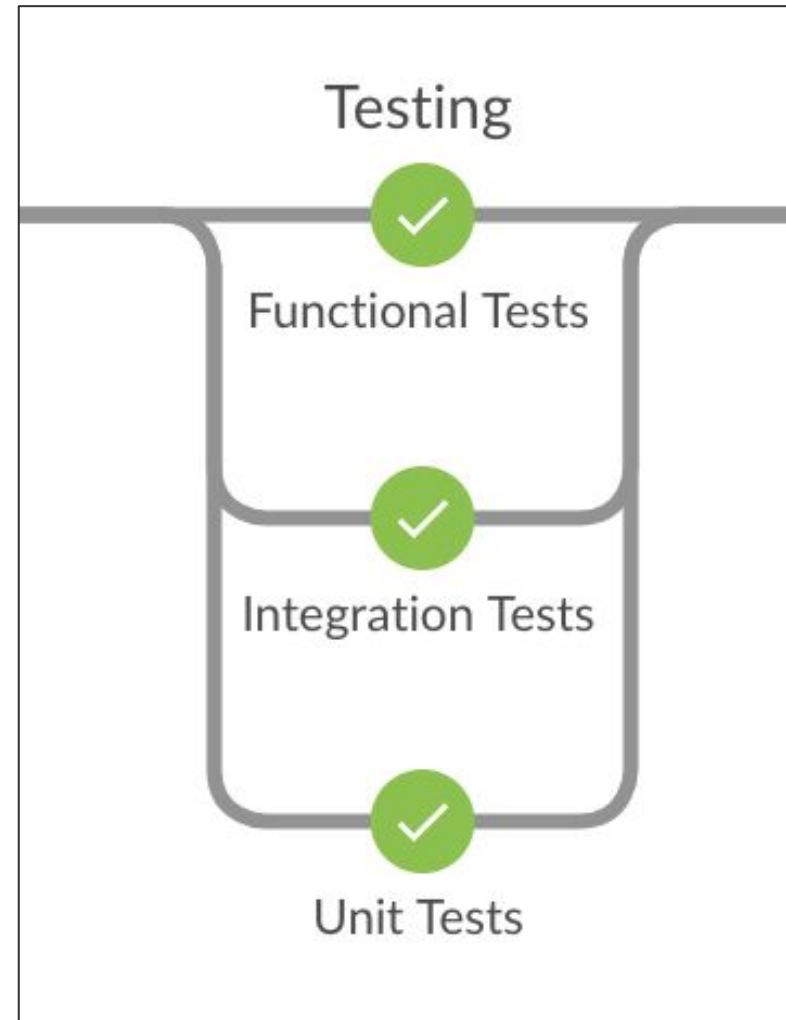
<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-01.md#exercise-17>

# Conditional Flow Control

```
stage('Deploy') {  
    when {  
        beforeAgent true  
        expression {  
            currentBuild.result == null || currentBuild.result == 'SUCCESS'  
        }  
    }  
    steps {  
        ...  
    }  
}  
  
stage('Build Master') {  
    when {  
        branch 'master'  
    }  
    steps {  
        ...  
    }  
}
```

# Executing Steps in Parallel

```
pipeline {
  agent any
  stages {
    stage("Testing") {
      parallel {
        stage("Unit Tests") {
          agent { docker 'openjdk:7-jdk-alpine' }
          steps {
            sh 'java -version'
          }
        }
        stage("Functional Tests") {
          agent { docker 'openjdk:8-jdk-alpine' }
          steps {
            sh 'java -version'
          }
        }
        stage("Integration Tests") {
          steps {
            sh 'java -version'
          }
        }
      }
    }
  }
}
```





# Hands On Exercise 1.10

---

## Executing Parallel Stages

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-01.md#exercise-18>

# !! Warning !!

```
pipeline {
  agent any

  environment {
    APP_VERSION = "0.0.1"
  }

  stages {
    stage('Parse POM') {
      steps {
        script {
          pom = readMavenPom file: 'pom.xml'
          APP_VERSION = pom.version
        }
      }
    }
  }
}
```



# Building a Multibranch Pipeline



# What is a Multibranch Pipeline?

The **Multibranch Pipeline** project type enables you to implement different Jenkinsfiles for different branches of the same project. In a Multibranch Pipeline project, Jenkins **automatically discovers, manages and executes** Pipelines for branches which contain a Jenkinsfile in source control.

A **Github Organization** or **Bitbucket Organization** scans for projects that have a Jenkinsfile and creates a **Multibranch Pipeline** project for each one it finds.

# Hands On Exercise 2.1

---

Fork The sample-rest-server Repo

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-02.md#exercise-21>

# Hands On Exercise 2.2

---

## Create a Github Organization Project

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-02.md#exercise-22>

# Hands On Exercise 2.3

---

## Add Branch Based Flow Control

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-02.md#exercise-23>



# Hands On Exercise 2.4

---

## Handling Feature Branches and Pull Requests

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-02.md#exercise-24>

# Enterprise Only Pipeline Features



# Checkpoints\*

```
stage("Checkpoint") {  
    agent none  
    steps {  
        checkpoint 'Completed Docker Image Testing'  
    }  
}
```

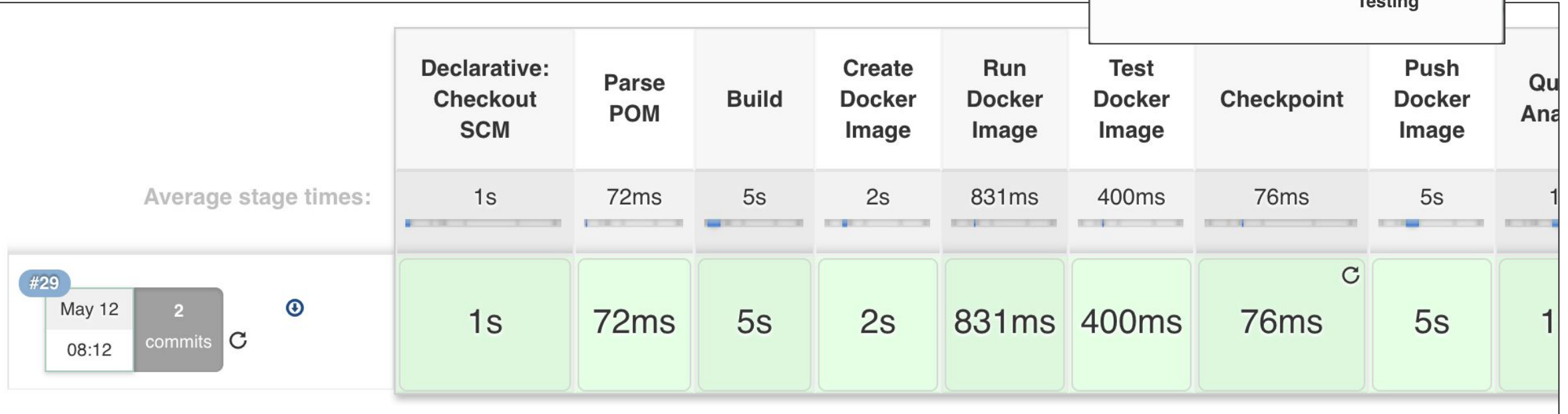
**Resume** ×

This Pipeline run can be restarted from the following Checkpoint(s)

Delete

Restart

Completed Docker Image Testing



# Hands On Exercise 3.1

---

## Create a Checkpoint

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-03.md#exercise-31>

## Geolocation Team



### Nearest ATM

Successful completion of  
**team://geolocation/nearest-atm/build/v1.x**  
**send event**  
**maven://nearest-atm:1.1-snapshot:jar**

```
...  
publishEvent \  
  'maven://nearest-atm:1.1-snapshot:jar'
```

Application  
**www:LATEST**  
**depends on**  
**maven://nearest-atm:1.1-snapshot:jar**

```
...  
trigger \  
  'maven://nearest-atm:1.1-snapshot:jar'
```

- Trigger defined on the downstream pipeline as desired
- Friendly syntax with the team name
- Foreign key on generated artifact as desired
- Would be greatly improved building the triggers and publishing the events automatically being maven / gradle / npm aware (similar to the withMaven plugin)

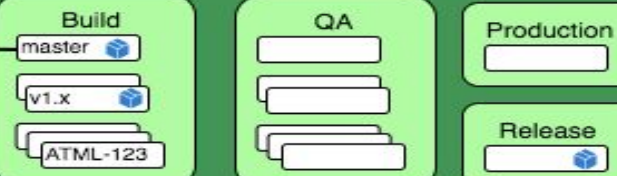


- How can the author of the upstream pipeline inject the version of his "trigger" step based on the build manifest instead of hardcoding in the Jenkins file

## WWW Team



### WWW



# Hands On Exercise 3.2

---

## Cross Team Collaboration

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-03.md#exercise-32>

# Custom Markers\*

Custom script

Marker file

Pipeline

Definition 

Pipeline script

Script

1

try sample Pipeline...

☒ Use Groovy Sandbox

[Pipeline Syntax](#)

Custom script

Marker file

Pipeline

Definition 

Pipeline script from SCM

SCM

None

Script Path

Jenkinsfile

Lightweight checkout

☒

[Pipeline Syntax](#)



# Hands On Exercise 3.3

---

## Use a Custom Marker File

<https://github.com/pipeline-training/jenkins-declarative/blob/master/Exercise-03.md#exercise-33>

# Best Practices



# Pipeline Best Practices

A few best practices for creating pipelines in Jenkins:

- **Use a Jenkinsfile** - your pipeline should be treated like code
- **Keep it simple** - limit the amount of logic you use and don't treat declarative like a general purpose programming language (**hint**: every step should be executable from outside of Jenkins)
- **Parallelize your pipeline** - if stages can run in parallel do it to improve execution time
- **Shift important steps to the left of your pipeline** - fail faster
- **Wrap Inputs in Timeouts** - don't leave jobs waiting indefinitely for input blocking executors
- **Prefer Stash to Archiving** - to share files between stages so that you can move execution of stages across multiple agents seamlessly
- **Use Plugins vs custom code** - easier to develop and maintain
- **Prefer external scripts/tools for complex or CPU-expensive processing** - limit processing requirements on the master
- **Use trusted global libraries** - increases reusability/reduces complexity, but beware of requirements for processing scripts on the master

Thank You!

