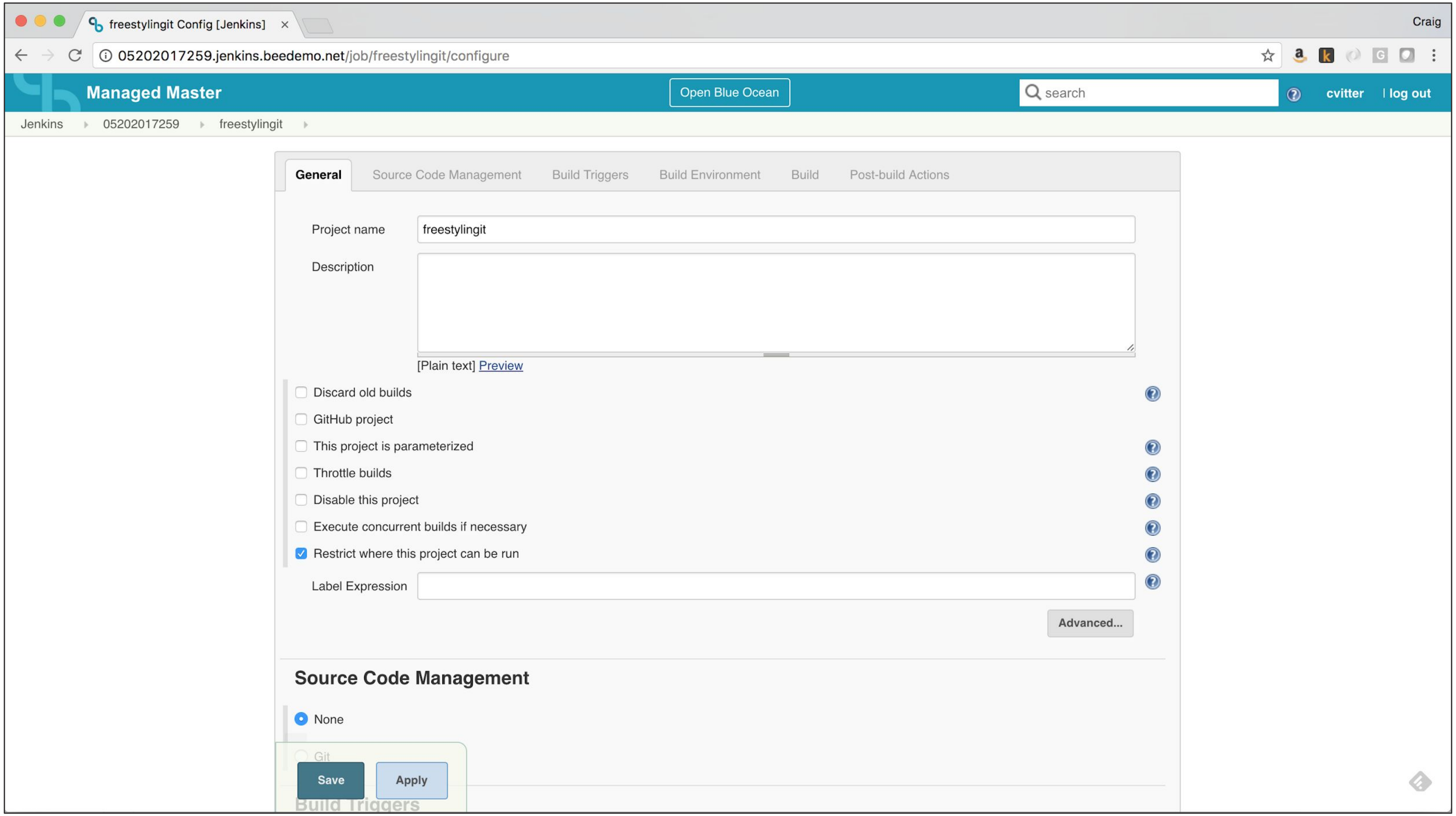
Introduction to Declarative Pipeline

JENKINS DAYS by CloudBees

Set-up

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Setup.md

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

Jenkins › 05202017259 › freestylingit ›

General | Source Code Management | Build Triggers | Build Environment | Build | Post-build Actions

Project name: freestylingit

Description:

[Plain text] Preview

- [ ] Discard old builds
- [ ] GitHub project
- [ ] This project is parameterized
- [ ] Throttle builds
- [ ] Disable this project
- [ ] Execute concurrent builds if necessary
- [x] Restrict where this project can be run

Label Expression:

Advanced...

Source Code Management

(●) None

( ) Git

Save | Apply

Build Triggers

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 on multiple agents
    - Ability to run steps in parallel
- **Not Auditable** - The creation and editing of jobs isn’t auditable without using additional plugins

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 easily 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
- **Visualizable** - enables status-at-a-glance dashboards like the built in Pipeline Stage View and Blue Ocean

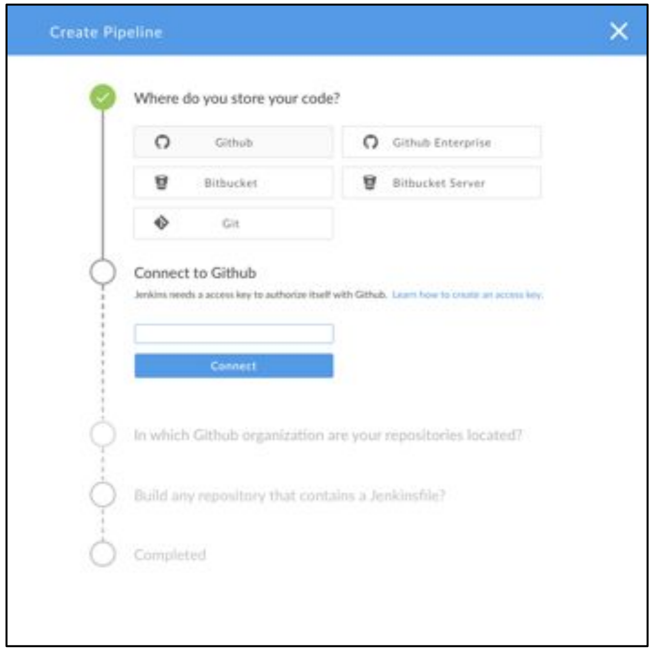
Why You Should Use Declarative Instead of Scripted

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 more straightforward with Declarative syntax
- **Richer Syntax** - Declarative provides richer syntactical features over Scripted Pipeline syntax
- **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)

Declarative Basics

Blue Ocean Pipeline Editor



Create Pipeline

Where do you store your code?

Github | Github Enterprise | Bitbucket | Bitbucket Server | Git

Connect to Github

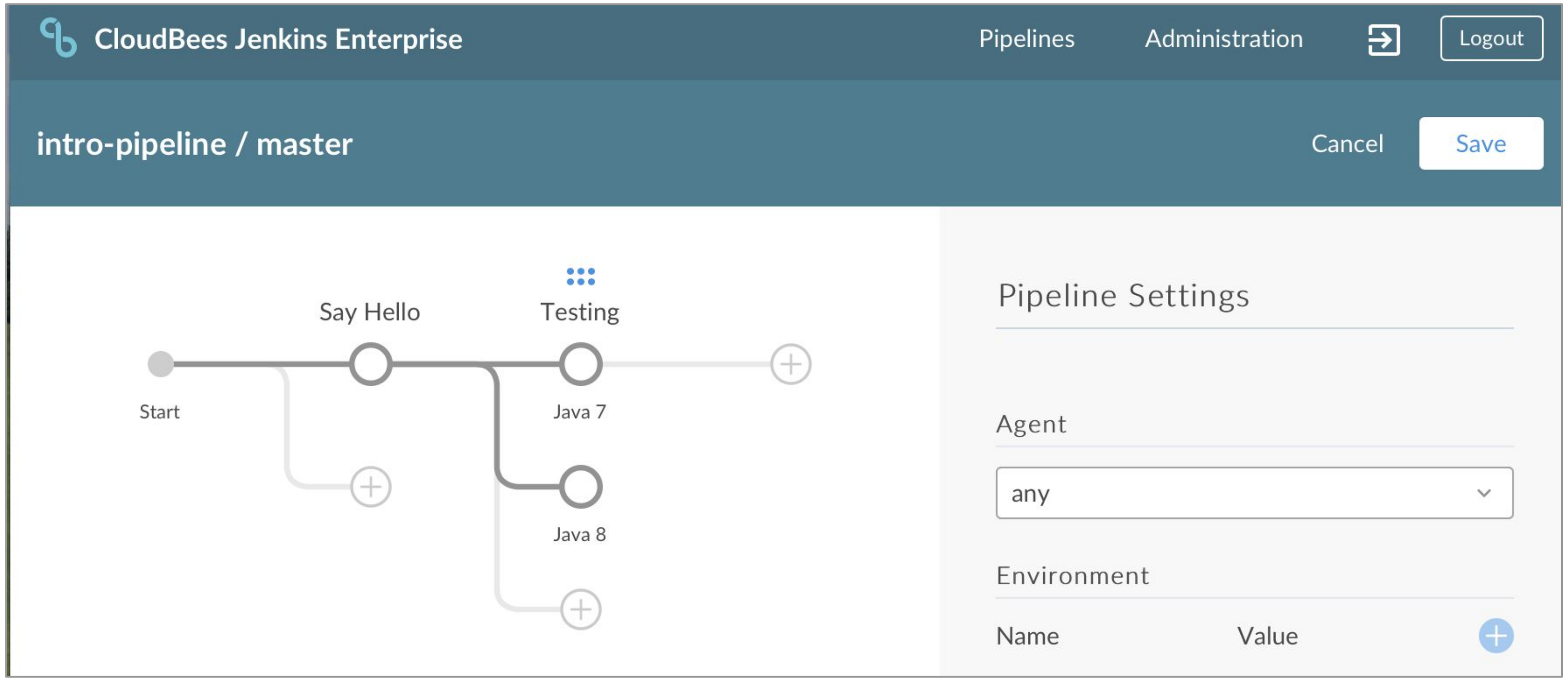
Jenkins needs a access key to authorize itself with Github. Learn how to create an access key.

Connect

In which Github organization are your repositories located?

Build any repository that contains a Jenkinsfile?

Completed



CloudBees Jenkins Enterprise | Pipelines | Administration | Logout

intro-pipeline / master

Cancel | Save

Pipeline Settings

Agent: any

Environment

Name | Value

Hands On Exercise 1.0

Blue Ocean Pipeline Editor

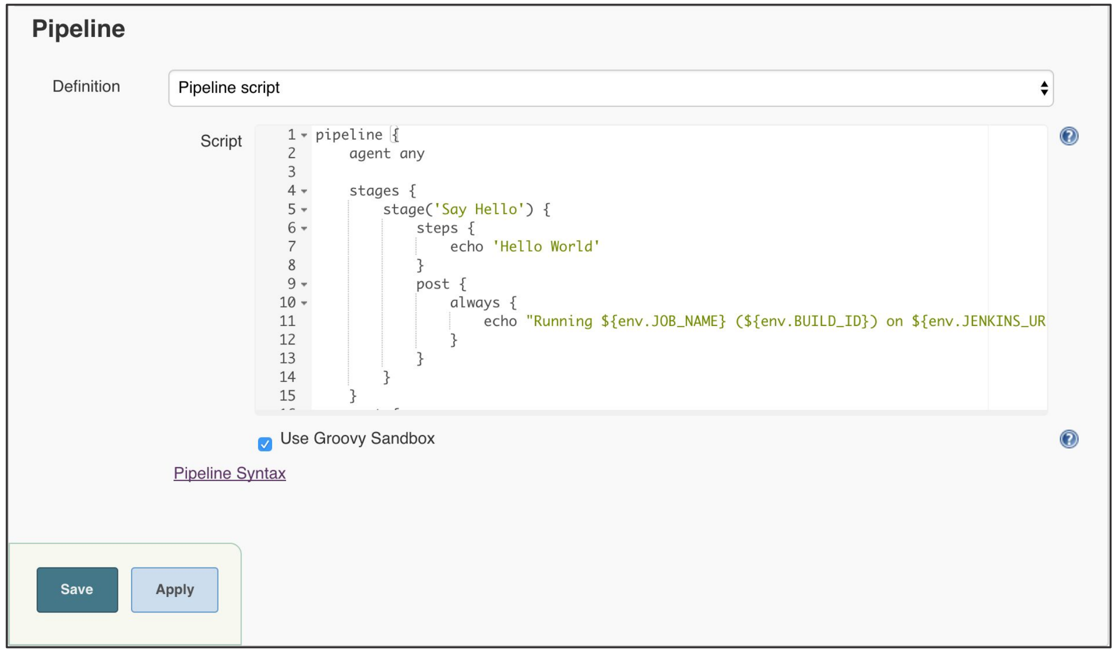
https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-01.md#exercise-10---blue-ocean-editor

The Simplest Declarative Jenkinsfile vs Scripted

```
pipeline {
    agent any

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

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



Pipeline

Definition: Pipeline script

Script

```
pipeline {
    agent any

    stages {
        stage('Say Hello') {
            steps {
                echo 'Hello World'
            }
            post {
                always {
                    echo "Running ${env.JOB_NAME} (${env.BUILD_ID}) on ${env.JENKINS_UR
                }
            }
        }
    }
```

[✓] Use Groovy Sandbox

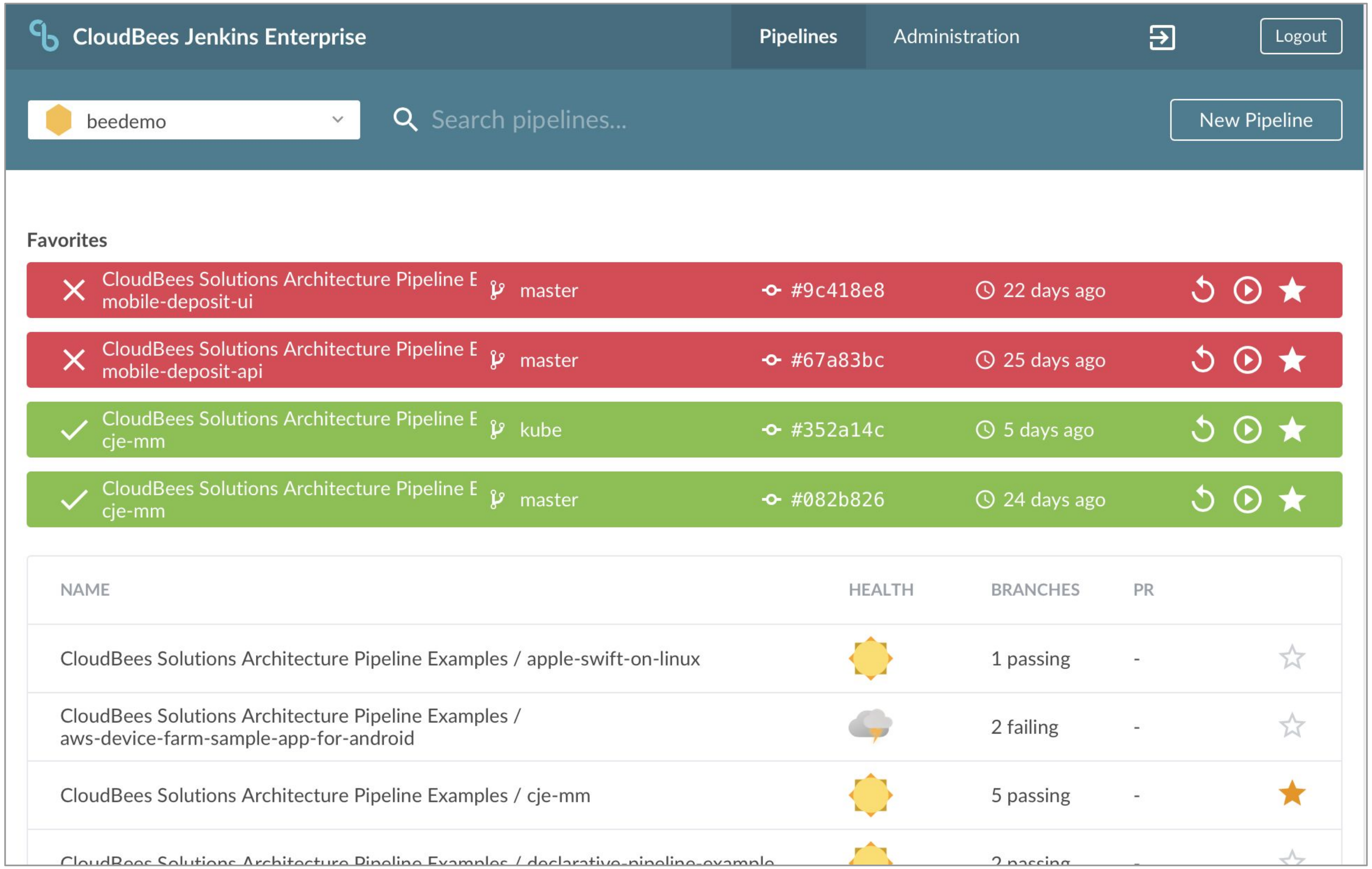
Pipeline Syntax

Hands On Exercise 1.1

Basic Declarative Syntax Structure

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-01.md#exercise-11---basic-declarative-syntax-structure

Blue Ocean Dashboard

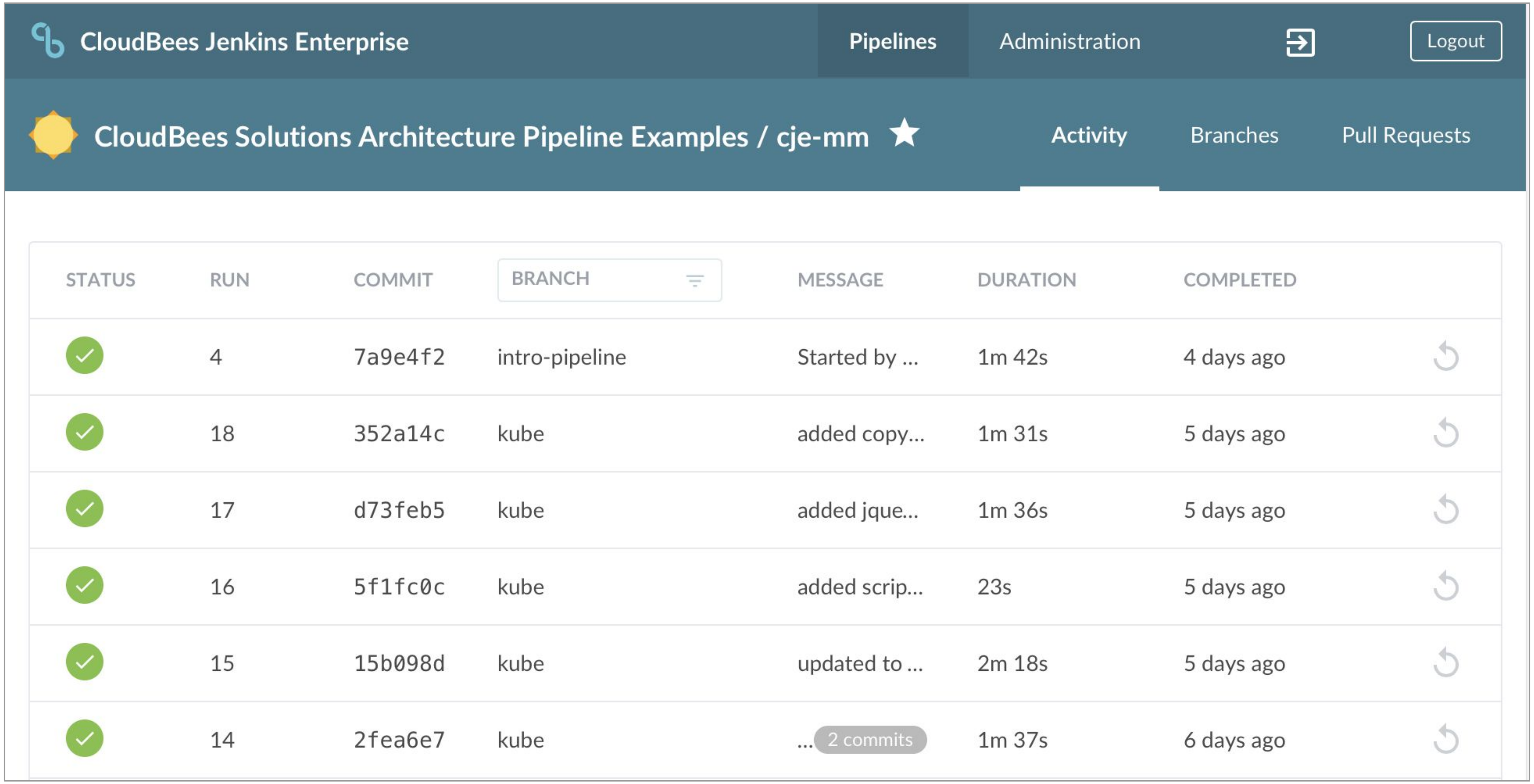


Favorites

✗	CloudBees Solutions Architecture Pipeline E mobile-deposit-ui	master	#9c418e8	22 days ago
✗	CloudBees Solutions Architecture Pipeline E mobile-deposit-api	master	#67a83bc	25 days ago
✓	CloudBees Solutions Architecture Pipeline E cje-mm	kube	#352a14c	5 days ago
✓	CloudBees Solutions Architecture Pipeline E cje-mm	master	#082b826	24 days ago

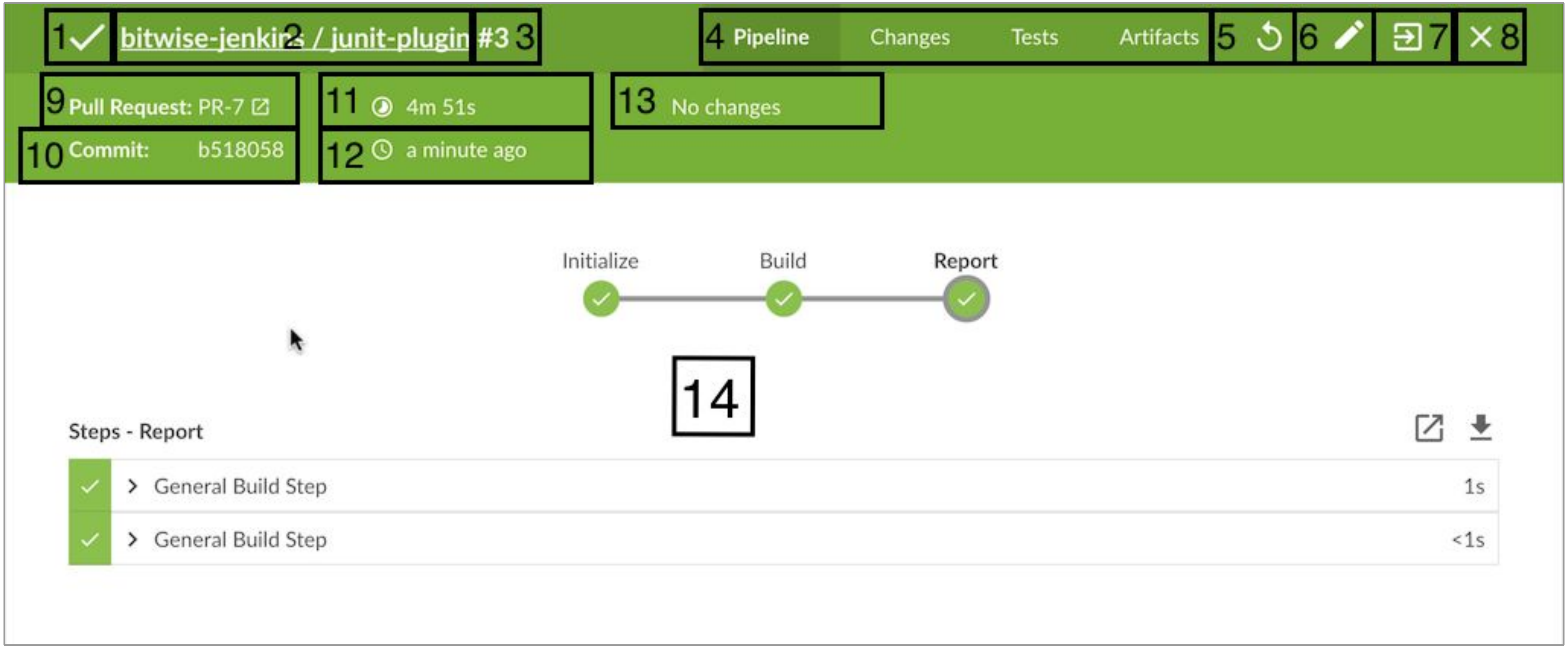
NAME	HEALTH	BRANCHES	PR	
CloudBees Solutions Architecture Pipeline Examples / apple-swift-on-linux	☀	1 passing	-	☆
CloudBees Solutions Architecture Pipeline Examples / aws-device-farm-sample-app-for-android	⛈	2 failing	-	☆
CloudBees Solutions Architecture Pipeline Examples / cje-mm	☀	5 passing	-	★
CloudBees Solutions Architecture Pipeline Examples / declarative-pipeline-example	☀	2 passing	-	☆

Blue Ocean Activity View


CloudBees Jenkins Enterprise
Pipelines
Administration
Logout
CloudBees Solutions Architecture Pipeline Examples / cje-mm
Activity
Branches
Pull Requests

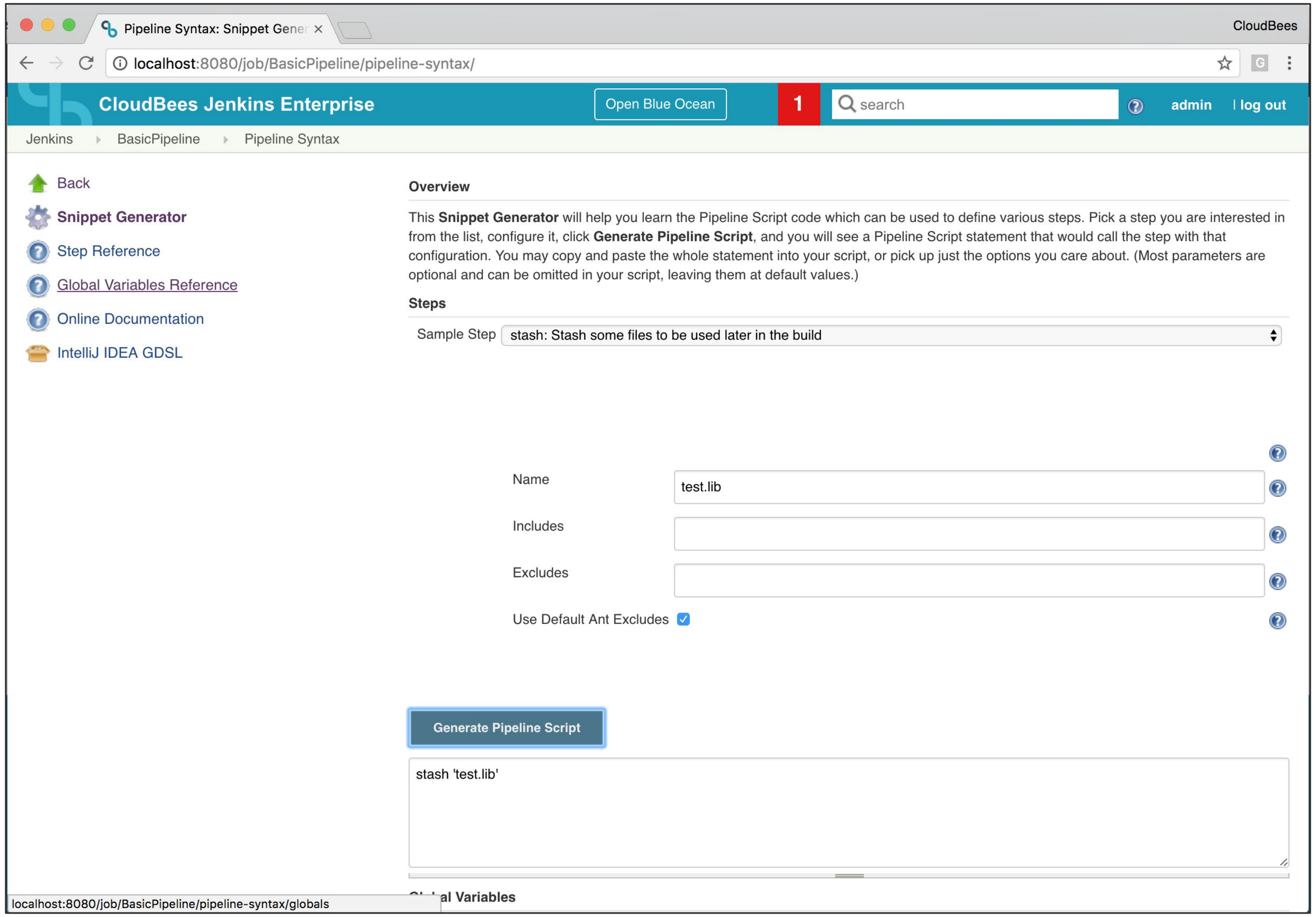
STATUS	RUN	COMMIT	BRANCH	MESSAGE	DURATION	COMPLETED
✓	4	7a9e4f2	intro-pipeline	Started by ...	1m 42s	4 days ago
✓	18	352a14c	kube	added copy...	1m 31s	5 days ago
✓	17	d73feb5	kube	added jque...	1m 36s	5 days ago
✓	16	5f1fc0c	kube	added scrip...	23s	5 days ago
✓	15	15b098d	kube	updated to ...	2m 18s	5 days ago
✓	14	2fea6e7	kube	... 2 commits	1m 37s	6 days ago

Pipeline Run Details View



✓	General Build Step	1s
✓	General Build Step	<1s

The Jenkins Snippet Generator



Pipeline Syntax: Snippet Gener ×

CloudBees

localhost:8080/job/BasicPipeline/pipeline-syntax/

CloudBees Jenkins Enterprise

Open Blue Ocean

1

search

admin | log out

Jenkins › BasicPipeline › Pipeline Syntax

- Back
- Snippet Generator
- Step Reference
- Global Variables Reference
- Online Documentation
- IntelliJ IDEA GDSL

Overview

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.)

Steps

Sample Step: stash: Stash some files to be used later in the build

Name	test.lib
Includes	
Excludes	
Use Default Ant Excludes	✓

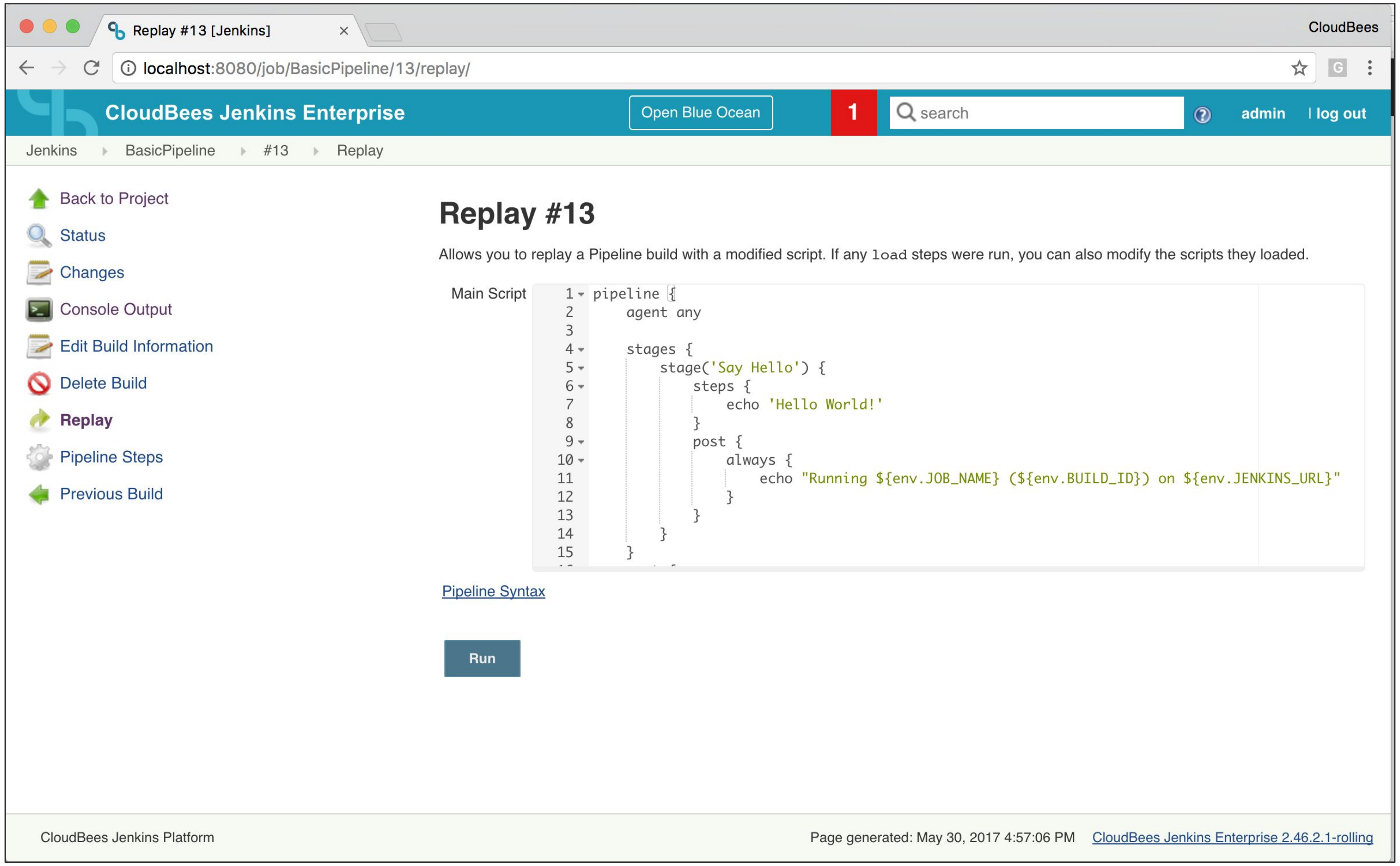
Generate Pipeline Script

```
stash 'test.lib'
```

Global Variables

localhost:8080/job/BasicPipeline/pipeline-syntax/globals

Pipeline Replay



Replay #13

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.

Main Script

```
pipeline {
    agent any

    stages {
        stage('Say Hello') {
            steps {
                echo 'Hello World!'
            }
            post {
                always {
                    echo "Running ${env.JOB_NAME} (${env.BUILD_ID}) on ${env.JENKINS_URL}"
                }
            }
        }
    }
```

CloudBees Jenkins Platform

Page generated: May 30, 2017 4:57:06 PM CloudBees Jenkins Enterprise 2.46.2.1-rolling

Specifying Agents

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

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

```
pipeline {
   agent none
   stages {
       stage('Build') {
         agent any
         steps {
            sh 'make'
            stash includes: '**/target/*.jar', name:
'app'
         }
      }
      stage('Test') {
         agent { label 'linux' }
         steps {
            unstash 'app'
            ...
         }
      }
   }
}
```

Hands On Exercise 1.2

Agent Parameters

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-01.md#exercise-12---agent-labels

Hands On Exercise 1.3

Agents with Docker

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-01.md#exercise-13---agents-with-docker

Environmental Variables

```
pipeline {
    agent any

    environment {
        A_VALUE = 'Some Value'
    }

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

Credentials

```
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

cloudbees

Hands On Exercise 1.4

Add Environment Variables

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-01.md#exercise-14---environment-directive

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 1.5

Job Parameters

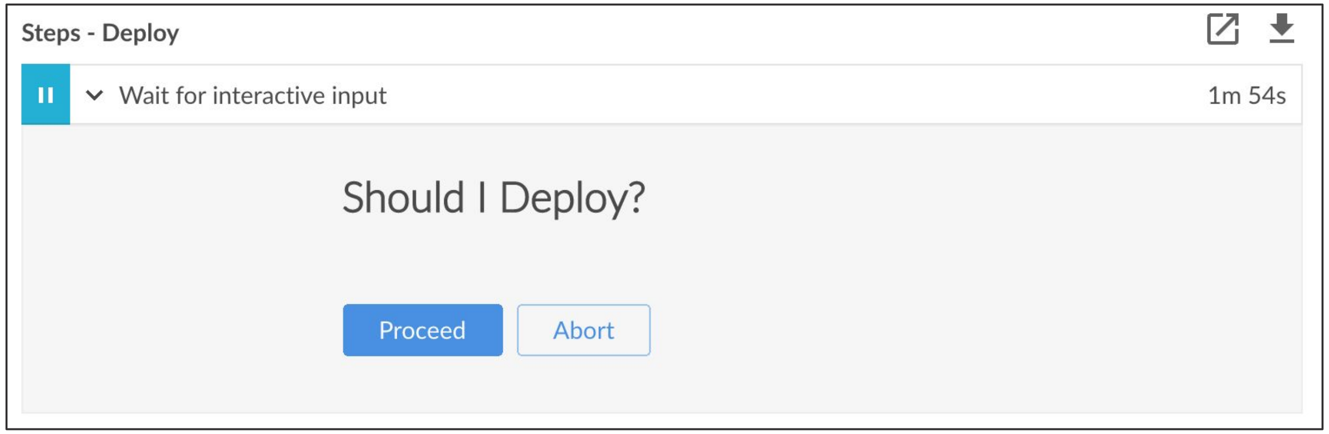
https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-01.md#exercise-15---parameters

Declarative Advanced Syntax

Capturing User Input

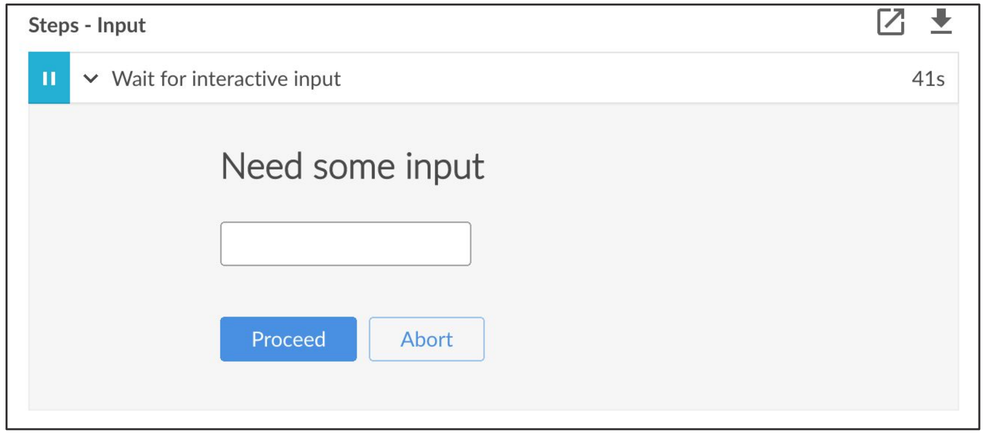
```
stage('Deploy') {
  input {
    message "Should we continue?"
  }
  steps {
    echo "Continuing with deployment"
  }
}
```

```
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?


Steps - Input

Wait for interactive input 41s

Need some input

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 2.1

Capture User Input During Run Time

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-02.md#exercise-21---interactive-input

Hands On Exercise 2.2

Capture User Input with Parameters

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-02.md#exercise-22---input-parameters

Post Actions

```
pipeline {
   agent any

   stages { ... }
   post {
      always {
         echo ‘I always run!’
      }
      success { ... }
      failure { ... }
      aborted { ... }
      unstable { ... }
      changed { ... }
      regression { ... }
      fixed { ... }
   }
}
```

```
pipeline {
   agent any

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

Hands On Exercise 2.3

Handling Post Actions

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-02.md#exercise-23---post-actions

Script Block

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

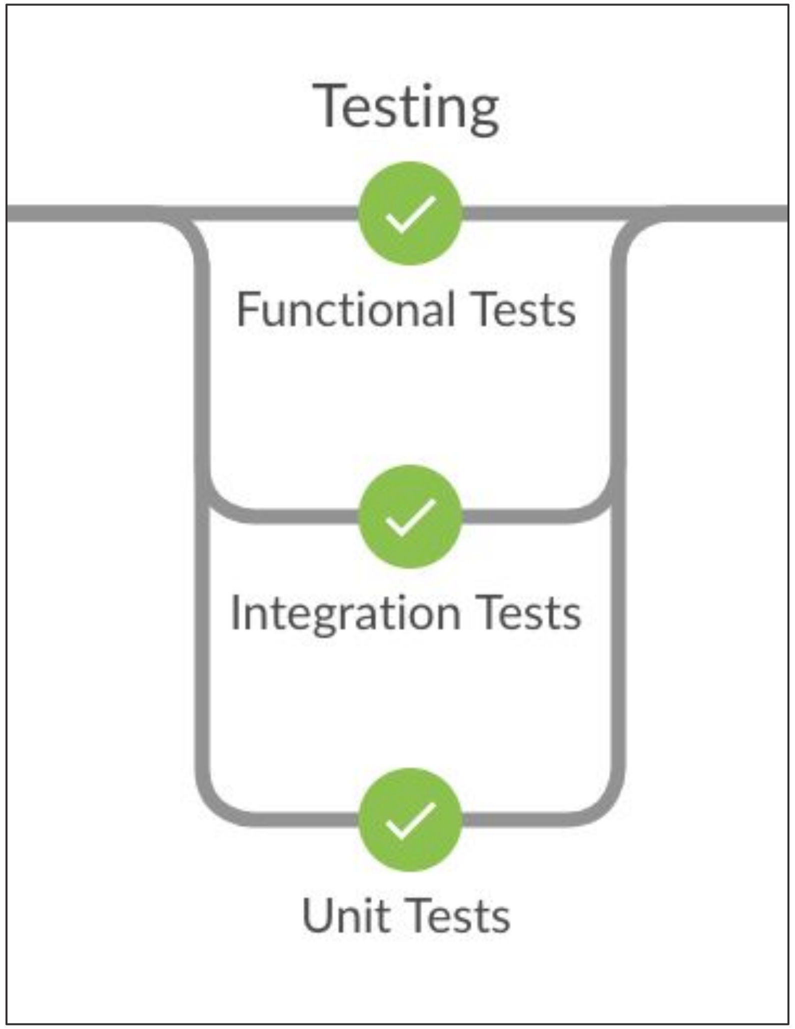
Hands On Exercise 2.4

Script Block

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-02.md#exercise-24---script-block

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 2.5

Executing Parallel Stages

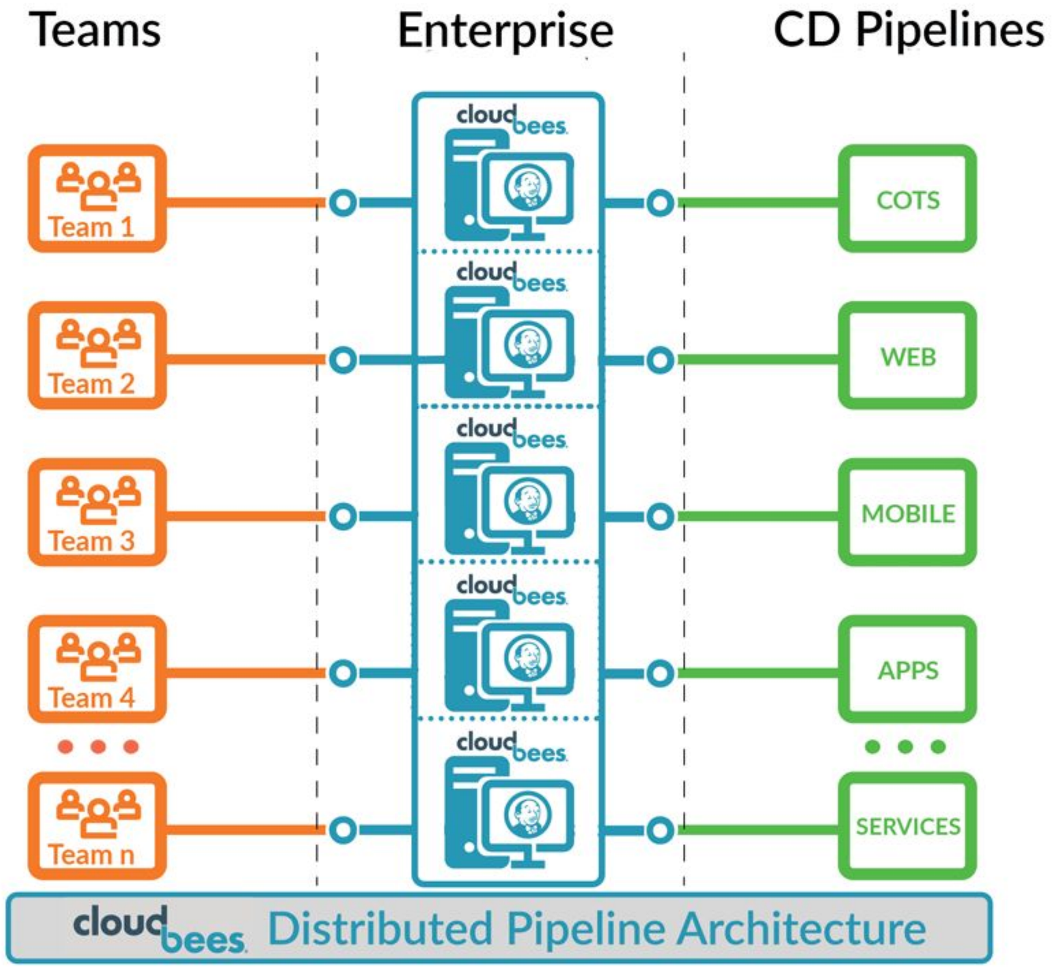
https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-02.md#exercise-25---parallelization

Distributed Pipelines with Pipeline as Code

Distributed Pipelines

An architecture that enables development teams to focus on their CI/CD pipelines:

- DevOps project teams get their own Jenkins Master
- Cross project contamination of workspaces and data is eliminated
- Scaling and elasticity achieved through use of cluster managed containers



Pipeline as Code

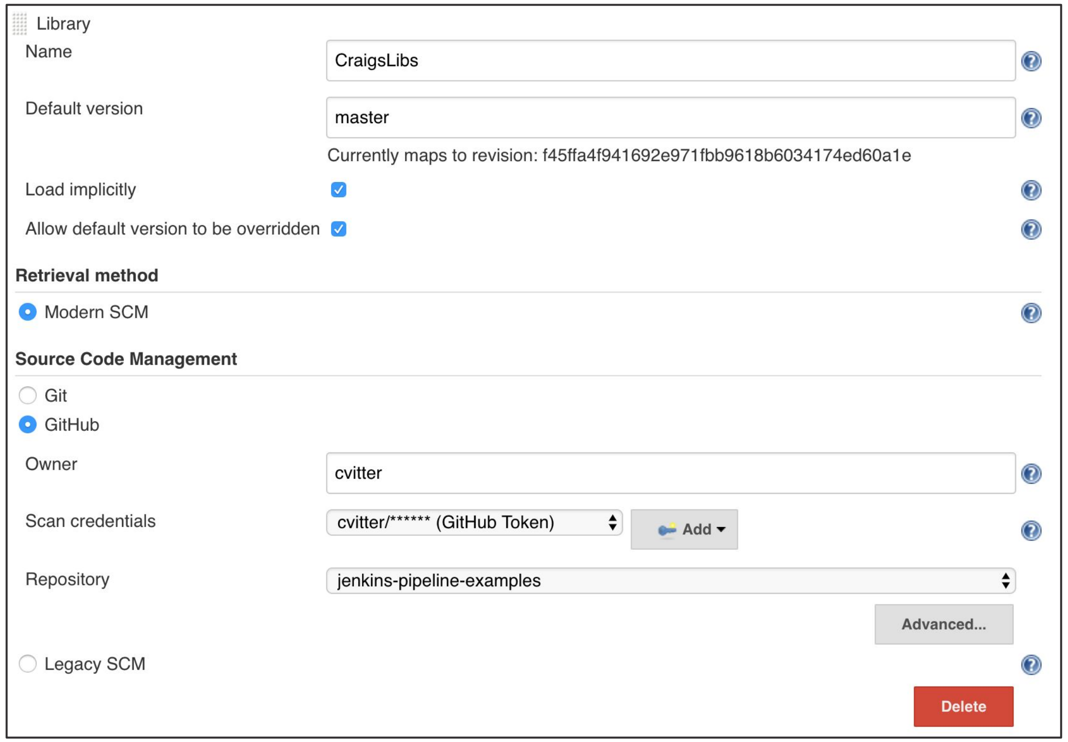
Pipeline as Code is a set of features that allow Jenkins users to define pipelined job processes with code, stored and versioned in a source repository. These features allow Jenkins to discover, manage, and run jobs for multiple source repositories and branches — eliminating the need for manual job creation and management.

To use Pipeline as Code, projects must contain a file named Jenkinsfile in the repository root, which contains a "Pipeline script" and one of the enabling jobs needs to be configured in Jenkins:

- **Multibranch Pipeline**: build multiple branches of a single repository automatically
- **Organization Folders**: scan a GitHub Organization or Bitbucket Team to discover an organization’s repositories, automatically creating managed Multibranch Pipeline jobs for them

Shared Libraries

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



Library

Name	CraigsLibs
Default version	master
	Currently maps to revision: f45ffa4f941692e971fbb9618b6034174ed60a1e
Load implicitly	✓
Allow default version to be overridden	✓

Retrieval method

● Modern SCM

Source Code Management

○ Git

● GitHub

Owner	cvitter
Scan credentials	cvitter/****** (GitHub Token)
Repository	jenkins-pipeline-examples

○ Legacy SCM

```
library 'CraigsLibs'

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

Hands On Exercise 3.2

Using Shared Libraries

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-03.md#exercise-32---shared-libraries

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 on it finds.

Hands On Exercise 3.3

Fork The sample-rest-server Repo

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-03.md#exercise-33--create-github-org-and-fork-repos

Hands On Exercise 3.4

Create a Github Organization Project

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-03.md#exercise-34---github-organization-project

Conditional Flow Control

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

stage('Build Master') {
   when {
      branch 'master'
    }
    steps {
      ...
    }
}
```

Hands On Exercise 3.5

Add Branch Based Flow Control

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-03.md#exercise-35---conditional-execution

Hands On Exercise 3.6

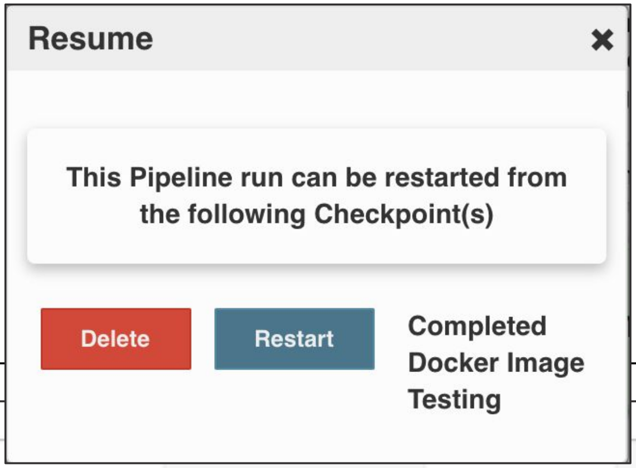
Handling Feature Branches and Pull Requests

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-03.md#exercise-36---prs-and-merging

Distributed Pipelines with CloudBees

Checkpoints*

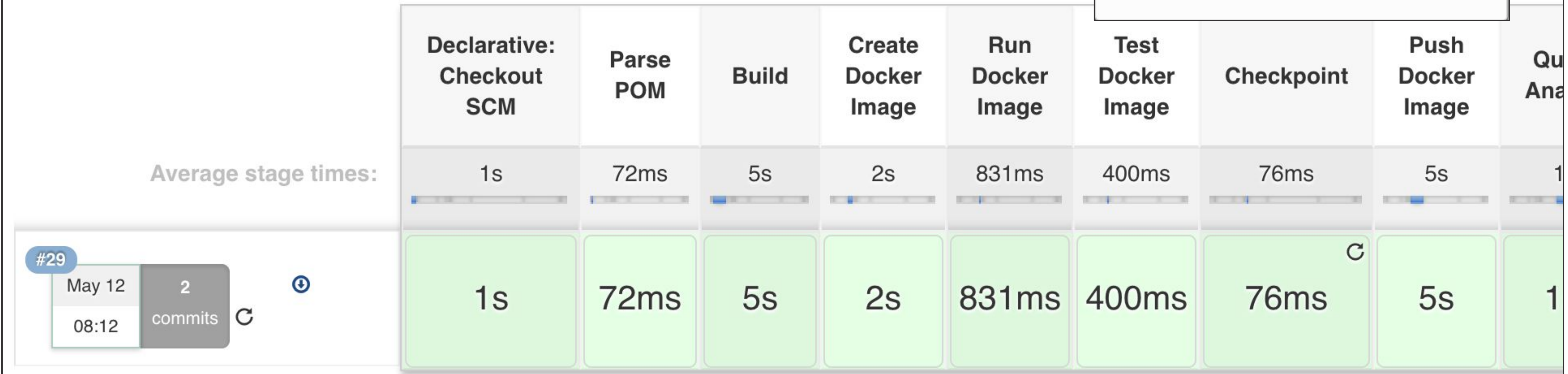
```
stage("Checkpoint") {
    agent none
    steps {
        checkpoint 'Completed Docker Image Testing'
    }
}
```



Resume

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

Completed Docker Image Testing

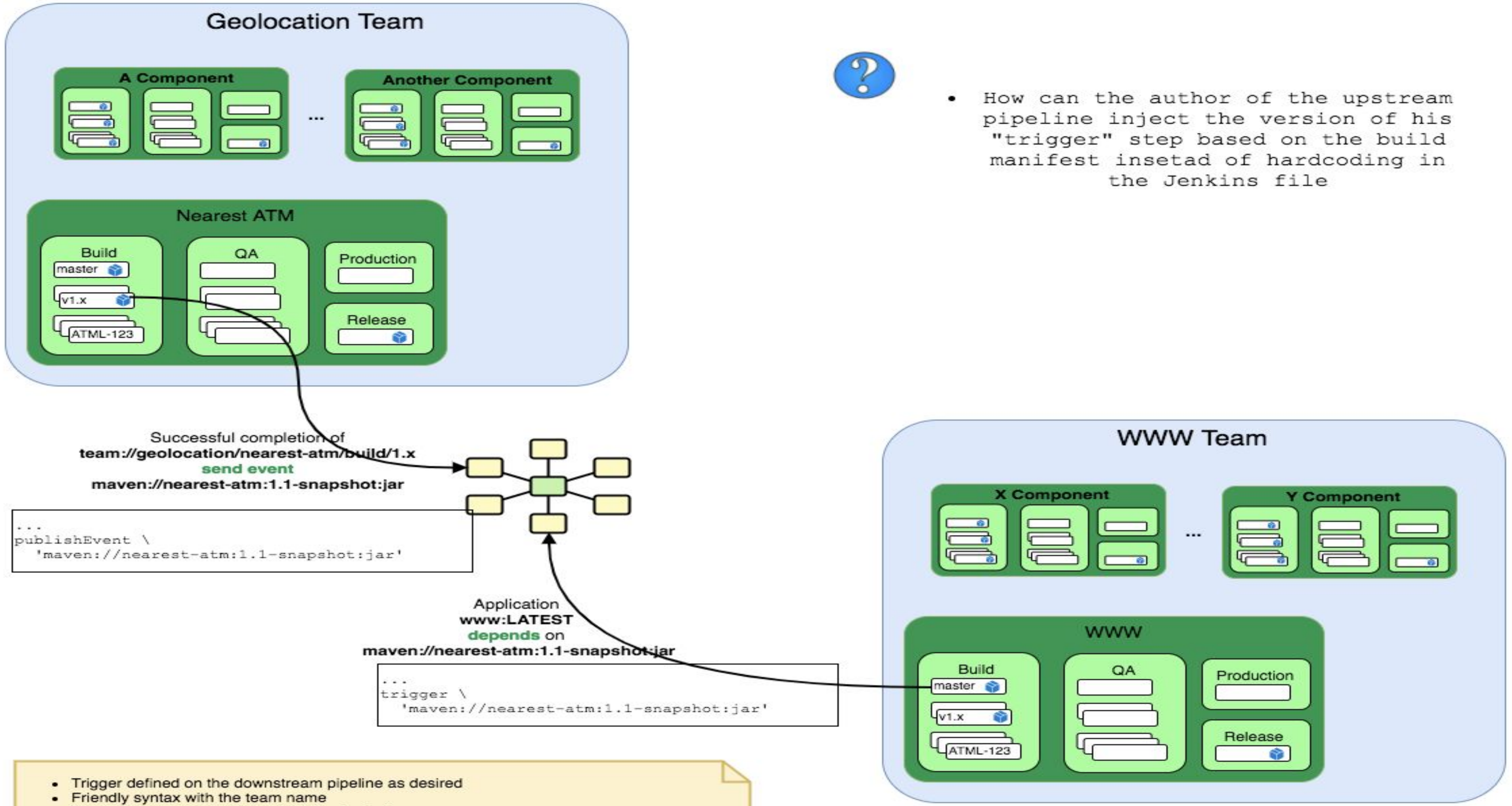


	Declarative: Checkout SCM	Parse POM	Build	Create Docker Image	Run Docker Image	Test Docker Image	Checkpoint	Push Docker Image	Qu Ana
Average stage times:	1s	72ms	5s	2s	831ms	400ms	76ms	5s	1
#29 May 12 08:12 2 commits	1s	72ms	5s	2s	831ms	400ms	76ms	5s	1

Hands On Exercise 4.1

Create a Checkpoint

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-04.md#exercise-41---checkpoints



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

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

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

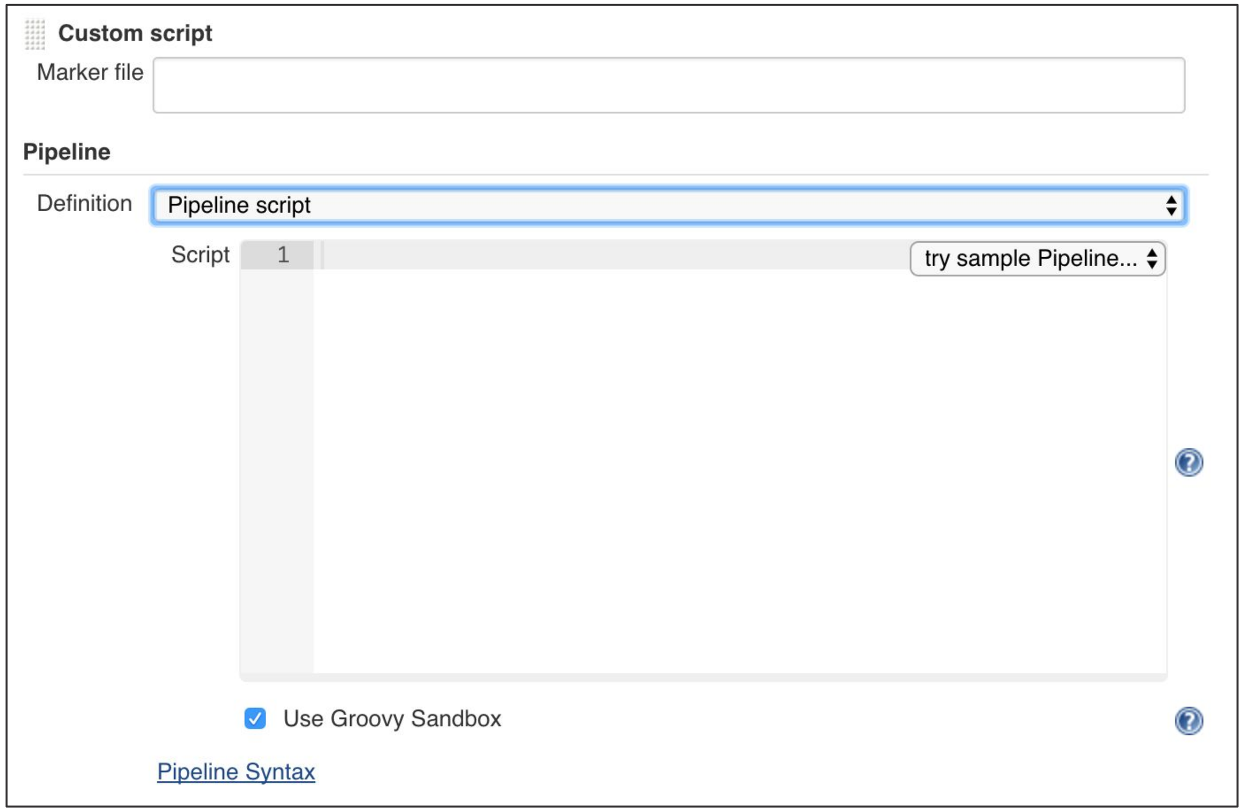
- Trigger defined on the downstream pipeline as desired
- Friendly syntax with the team name
- Foreign key on generated artifact as desired
- Would be greatly improved building the triggers and publishing the events automatically being maven / gradle / npm aware (similar to the withMaven plugin)

Hands On Exercise 4.2

Cross Team Collaboration

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-04.md#exercise-42---cross-team-collaboration

Custom Markers*


Custom script

Marker file

Pipeline

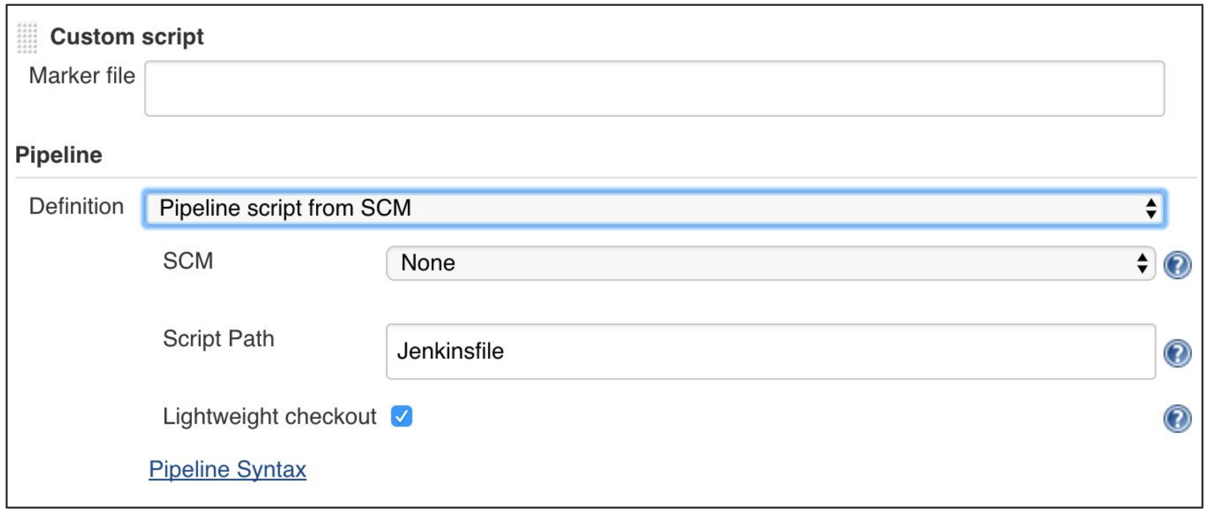
Definition: Pipeline script

Script

```
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 4.3

Use a Custom Marker File

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-04.md#exercise-43---custom-marker-files

Hands On Exercise 4.4

Agents with Kubernetes

https://github.com/cloudbees/intro-to-declarative-pipeline/blob/master/Exercise-01.md#exercise-14---kubernetes-agents

Jenkins Master as Code

CloudBees Jenkins Enterprise allows you to use your own custom Docker images for Managed Masters

For this workshop we used a custom CJE Managed Master Docker image that

- Skipped the Jenkins Setup Wizard
- Installed all of the plugins we will need to complete the workshop
- Configured:
    - Pipeline Shared Libraries
    - Configured the Docker label to use for the docker
    - Enabled the CloudBees Notification API

```
FROM cloudbees/cje-mm:2.107.1.2 1

#skip setup wizard and disable CLI
ENV JVM_OPTS -Djenkins.install.runSetupWizard=false -Djenkins.CLI.disabled=true -server 2


#jenkins master configuration (groovy scripts)
COPY ./init.groovy.d/* /usr/share/jenkins/ref/init.groovy.d/ 3
COPY ./license-activated/*
/usr/share/jenkins/ref/license-activated-or-renewed-after-expiration.groovy.d/
COPY ./quickstart/* /usr/share/jenkins/ref/quickstart.groovy.d/


#install additional plugins
ENV JENKINS_UC http://jenkins-updates.cloudbees.com 4
COPY plugins.txt plugins.txt
COPY jenkins-support /usr/local/bin/jenkins-support
COPY install-plugins.sh /usr/local/bin/install-plugins.sh 5


RUN /usr/local/bin/install-plugins.sh $(cat plugins.txt) 6
```

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!

JENKINS DAYS
by CloudBees