



Introduction to Declarative Pipeline

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

☆ a k G

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

☒ Restrict where this project can be run

Label Expression

Advanced...

Source Code Management

None

Git

Save

Apply

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

PipelinesAdministration

Logout

intro-pipeline / master

Cancel

Save

Start

Say Hello

Testing

Java 7

Java 8

Pipeline Settings

Agent

any

Environment

Name	Value	
------	-------	--

Hands On Exercise

Blue Ocean Pipeline 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

```
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_UR  
12        }  
13      }  
14    }  
15  }  
16 }
```

☒ Use Groovy Sandbox

[Pipeline Syntax](#)

Save Apply

Hands On Exercise

Basic Declarative Syntax Structure

Blue Ocean Dashboard

CloudBees Jenkins Enterprise

PipelinesAdministration

Logout

beedemo

Search pipelines...

New Pipeline

Favorites

×

CloudBees Solutions Architecture Pipeline Examples / mobile-deposit-ui

master

#9c418e8

22 days ago

↺

▶

★

×

CloudBees Solutions Architecture Pipeline Examples / mobile-deposit-api

master

#67a83bc

25 days ago

↺

▶

★

✓

CloudBees Solutions Architecture Pipeline Examples / cje-mm

kube

#352a14c

5 days ago

↺

▶

★

✓

CloudBees Solutions Architecture Pipeline Examples / 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	-	★

cloudbees


© 2018 CloudBees, Inc. All Rights Reserved.

14


Blue Ocean Activity View

 CloudBees Jenkins Enterprise

PipelinesAdministration



Logout

 CloudBees Solutions Architecture Pipeline Examples / cje-mm ★

ActivityBranchesPull 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

1 ✓ bitwise-jenkins

2 / junit-plugin

3 3

4 Pipeline

Changes

Tests

Artifacts

5 ↺

6 ✎

7 ↗

8 ✕

9 Pull Request: PR-7 ↗

11 ⌚ 4m 51s

13 No changes

10 Commit: b518058

12 ⌚ a minute ago

Initialize

Build

Report

14

Steps - Report

↗

⬇

✓ > General Build Step

1s

✓ > General Build Step

<1s

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 script: `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 tab is titled "Replay #13 [Jenkins]". The address bar shows "localhost:8080/job/BasicPipeline/13/replay/". The top navigation 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". Below the navigation bar, a breadcrumb trail shows "Jenkins > BasicPipeline > #13 > Replay".

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

The main content area is titled "Replay #13". Below the title, a description states: "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 description is a text area labeled "Main Script" containing 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 script text area, 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.

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

Agent Labels

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>

Hands On Exercise

Environment Variables

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

Job 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?

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

Interactive Input

Hands On Exercise

Capture User Input with Parameters

Post Actions

```
pipeline {
  agent any

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

```
pipeline {
  agent any

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

Hands On Exercise

Post Actions

Script Block

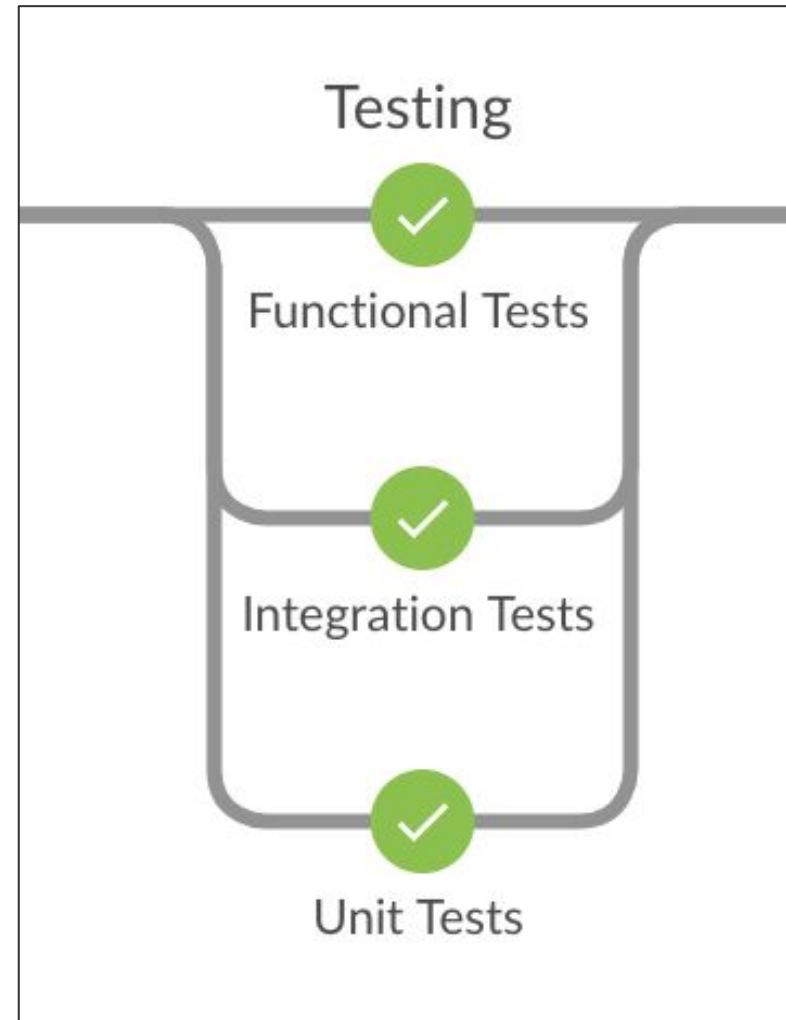
```
stage('Get Kernel') {  
    steps {  
        script {  
            try {  
                KERNEL_VERSION = sh (script: "uname -r", returnStdout: true)  
            } catch (err) {  
                echo "CAUGHT ERROR: ${err}"  
                throw err  
            }  
        }  
    }  
}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

Hands On Exercise

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

Executing Parallel Stages

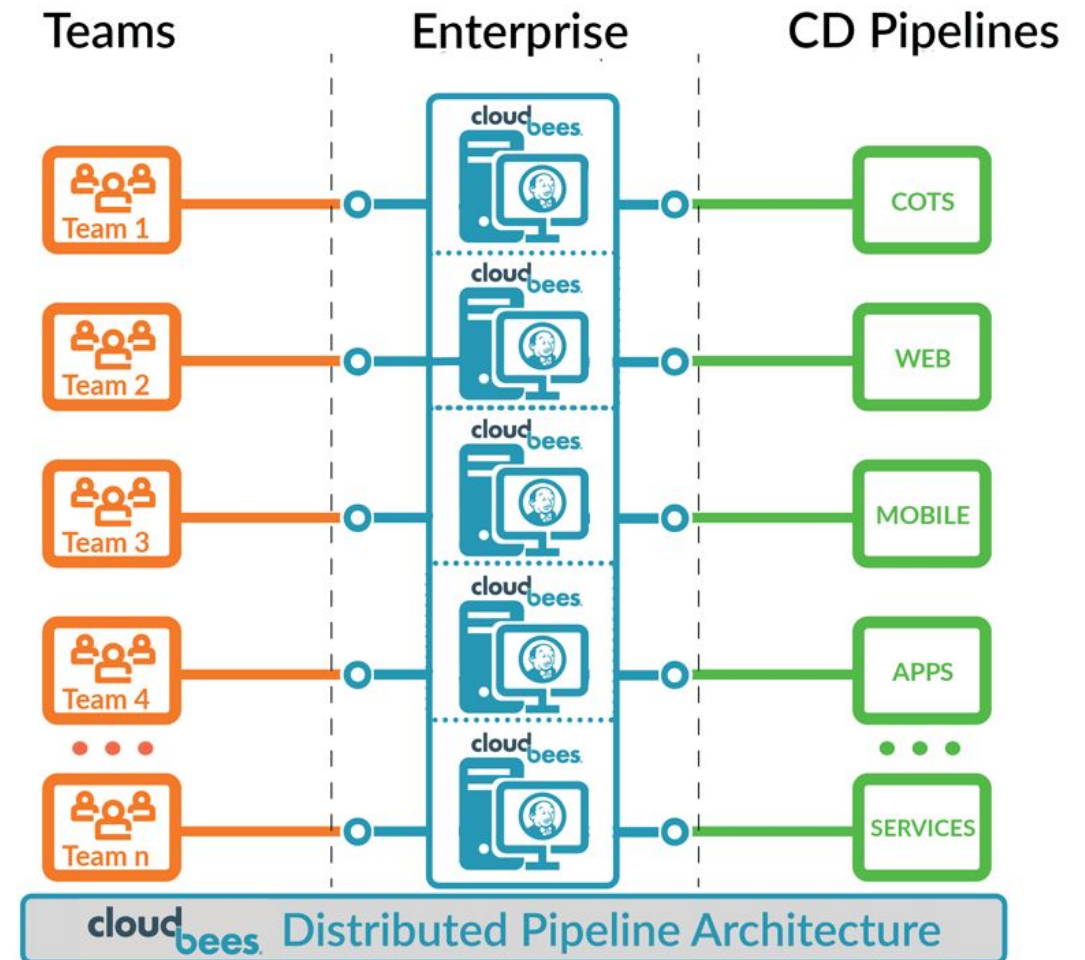
Distributed Pipelines with CloudBees



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

Checkpoints*

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

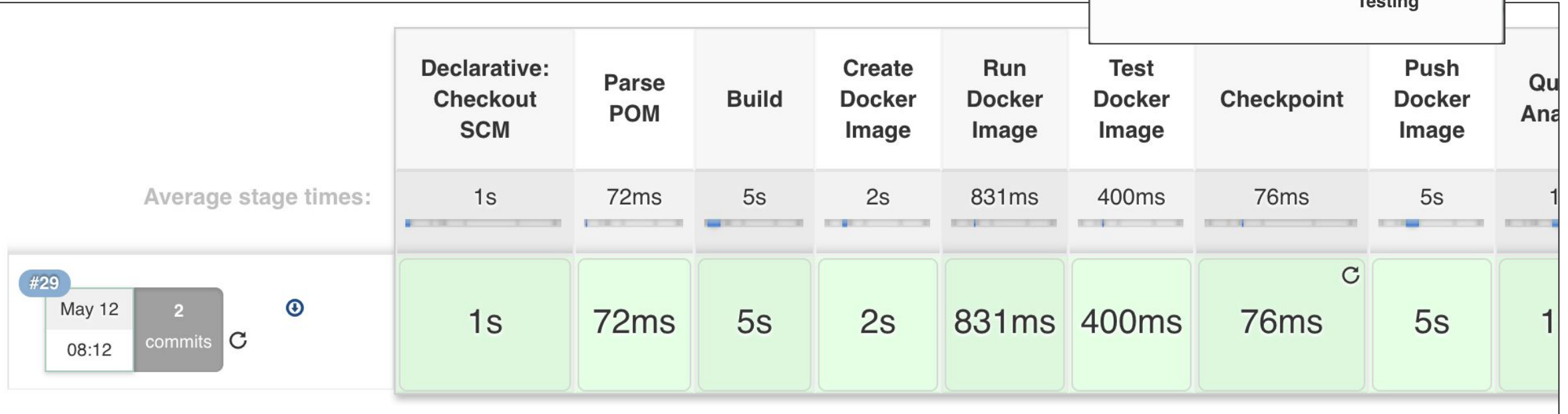
Resume ×

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

Delete

Restart

Completed Docker Image Testing



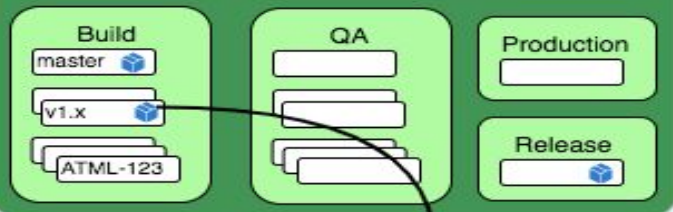
Hands On Exercise

Create a Checkpoint

Geolocation Team



Nearest ATM



- 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

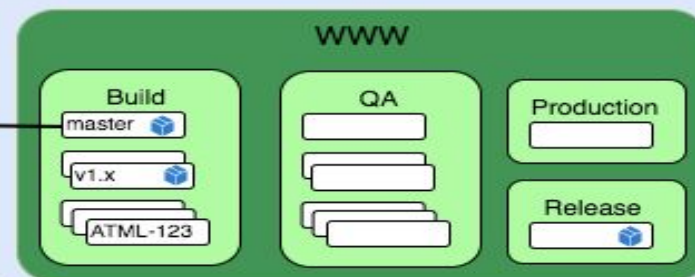
Successful completion of
team://geolocation/nearest-atm/build/v1.x
send event
maven://nearest-atm:1.1-snapshot:jar

```
...  
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'
```

WWW Team



- 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

Cross Team Collaboration

Dynamic & Ephemeral Agents with Kubernetes

- Jenkins plugin to run dynamic agents in a Kubernetes cluster
- The Jenkins Kubernetes plugin creates a Kubernetes Pod for each agent started, defined by the Docker image to run, and stops it after each build
- All of the agents you have used up to this point have been provided by the Kubernetes cluster and defined in a Pod Template
- The Pod Template allows adding additional job specific Docker containers that you can run Pipeline steps inside

Hands On Exercise

Agents with Kubernetes

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

Create a Github Organization Project

Hands On Exercise

Use a Custom Marker File

More Pipeline as Code



Shared Libraries

```
// Groovy Library located in
// github.com/example/SharedLibs/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 'Default version' is set to 'master', which maps to the revision 'f45ffa4f941692e971fbb9618b6034174ed60a1e'. The 'Load implicitly' and 'Allow default version to be overridden' options are checked. Under 'Retrieval method', 'Modern SCM' is selected. In the 'Source Code Management' section, 'GitHub' is chosen as the provider. The 'Owner' is 'cvitter', the 'Repository' is 'jenkins-pipeline-examples', and the 'Scan credentials' field contains 'cvitter/***** (GitHub Token)' with an 'Add' button next to it. There are 'Advanced...' and 'Delete' buttons at the bottom right of the configuration area.

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

Hands On Exercise

Using 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 one it finds.

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

Add Branch Based Flow Control

Hands On Exercise

Handling Feature Branches and Pull Requests

Hands On Exercise

Advanced Jenkins Kubernetes Agents

Config Ad Code for Masters

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!