

Ci with jenkins and rtifactory - DevOps ®

Version 1.0

Sela College

© 2013 Sela college All rights reserved.

All other trademarks are the property of their respective owners.

This course material has been prepared by:

Sela Software Labs Ltd.

14-18 Baruch Hirsch St. Bnei Brak 51202 Israel

Tel: 972-3- 6176666 Fax: 972-3- 6176667

Copyright: © Sela Software Labs Ltd.

All Materials contained in this book were prepared by Sela Software Labs Ltd. All rights of this book are reserved solely for Sela Software Labs Ltd. The book is intended for personal, noncommercial use. All materials published in this book are protected by copyright, and owned or controlled by Sela Software Labs Ltd, or the party credited as the provider of the Content. You may not modify, publish, transmit, participate in the transfer or sale of, reproduce, create new works from, distribute, perform, store on any magnetic device, display, or in any way exploit, any of the content in whole or in part. You may not alter or remove any trademark, copyright or other notice from copies of the content. You may not use the material in this book for the purpose of training of any kind, internal or for customers, without beforehand written approval of Sela Software Labs Ltd.

The Use of this book

The material in this book is designed to assist the student during the course. It does not include all of the information that will be referred to during the course and should not be regarded as a replacement for reference manuals.

Limits of Responsibility

Sela Software Labs Ltd invests significant effort in updating this book, however, Sela Software Labs Ltd is not responsible for any errors or material which may not meet specific requirements. The user alone is responsible for decisions based on the information contained in this book.

Protected Trademarks

In this book, protected trademarks appear that are under copyright. All rights to the trademarks in this material are reserved to the authors.

SELA wishes you success in the course!

Table of Contents

Module 01 - Introduction

<i>Agenda</i>	3
<i>DevOps Pipelines</i>	3
<i>What is Continuous Integration?</i>	4
<i>The benefits of CI</i>	4
<i>Continuous Integration Practices</i>	5
<i>How Jenkins Fit In?</i>	6
<i>Most Used CI Tools</i>	6
<i>History of Jenkins</i>	7

Module 02 - A little bit about Linux

<i>Agenda</i>	3
<i>History of Linux</i>	3
<i>Linux Distributions</i>	4
<i>Filesystem Hierarchy Standard</i>	5
<i>The Terminal</i>	5
<i>The Terminal – Basic commands</i>	6
<i>VIM a Linux Editor</i>	6
<i>A look to Ubuntu Distribution</i>	7
<i>A look to Ubuntu Distribution - Features</i>	8
<i>Lab: Lab 1: Working with Ubuntu</i>	9

Module 03 - Jenkins Overview

<i>Agenda</i>	3
<i>Architecture</i>	3
<i>Installation</i>	4
<i>Backup and Restore</i>	4
<i>Configuration</i>	5
<i>Secure Jenkins</i>	5
<i>Jenkins Slaves</i>	6
<i>Jenkins Plugins</i>	7
<i>System Logs</i>	7
<i>Views</i>	8
<i>Jenkins Overview</i>	9

Module 04 - Anatomy of a Jenkins Job

<i>Agenda</i>	3
<i>General Configuration</i>	3
<i>Source Code Management</i>	4
<i>Build Triggers</i>	4
<i>Build Environment</i>	5
<i>Build Steps</i>	5
<i>Post-Build steps</i>	6
<i>Using plugins</i>	6
<i>Lab: Lab 02: Creating the first freestyle Jenkins job</i>	7

Module 05 - Package Management with Artifactory

<i>Agenda</i>	3
<i>What is Package Management?</i>	3
<i>Package Management Systems</i>	4
<i>What is JFrog Artifactory</i>	4
<i>Automating Processes</i>	5
<i>Artifactory Editions</i>	5
<i>Repository Types</i>	6
<i>Generic Repositories Overview</i>	6
<i>Lab: Lab 03: Deploying build artifacts from Jenkins to Artifactory</i>	8

Module 06 - Jenkins pipelines as code

<i>Agenda</i>	3
<i>Pipeline as Code</i>	3
<i>What is Jenkins Pipelines?</i>	4
<i>Benefits of Jenkins Pipelines?</i>	4
<i>Meeting the Jenkinsfile</i>	5
<i>Declarative Pipelines</i>	5
<i>Scripted Pipelines</i>	10
<i>Lab: Lab 4: CI jobs with Jenkins pipelines</i>	13

Module 07 - Jenkins with Blue Ocean

<i>Agenda</i>	3
<i>What is Jenkins Blue Ocean</i>	4
<i>Main Features</i>	4
<i>Blue Ocean installation</i>	7
<i>Blue Ocean Overview</i>	7
<i>Lab: Lab 5: Creating a Jenkins job using Blue Ocean</i>	11

Module 08 - Summary

<i>DevOps Pipelines</i>	3
<i>Linux Overview</i>	3
<i>Jenkins Overview</i>	4
<i>Jenkins Freestyle Jobs</i>	4
<i>Package Management with Artifactory</i>	5
<i>Jenkins Pipelines as Code</i>	5
<i>Jenkins Blue Ocean</i>	6
<i>Itshak Eli</i>	7

Module 01 - Introduction

Contents:

Agenda	3
DevOps Pipelines.....	3
What is Continuous Integration?	4
The benefits of CI.....	4
Continuous Integration Practices	5
How Jenkins Fit In?.....	6
Most Used CI Tools.....	6
History of Jenkins.....	7



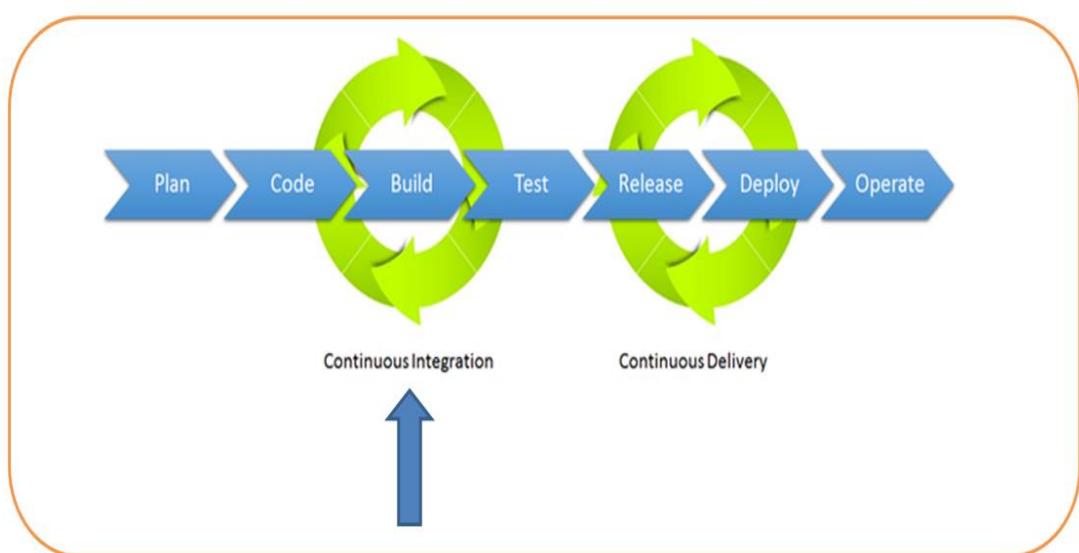
Module 01: Introduction

Continuous Integration with Jenkins and Artifactory

Agenda

- ❖ DevOps Pipelines
- ❖ What is Continuous Integration?
- ❖ The Benefits of CI
- ❖ Continuous Integration Practices
- ❖ How Jenkins Fit In?
- ❖ Most Used CI Tools
- ❖ History of Jenkins

DevOps Pipelines



What is Continuous Integration?

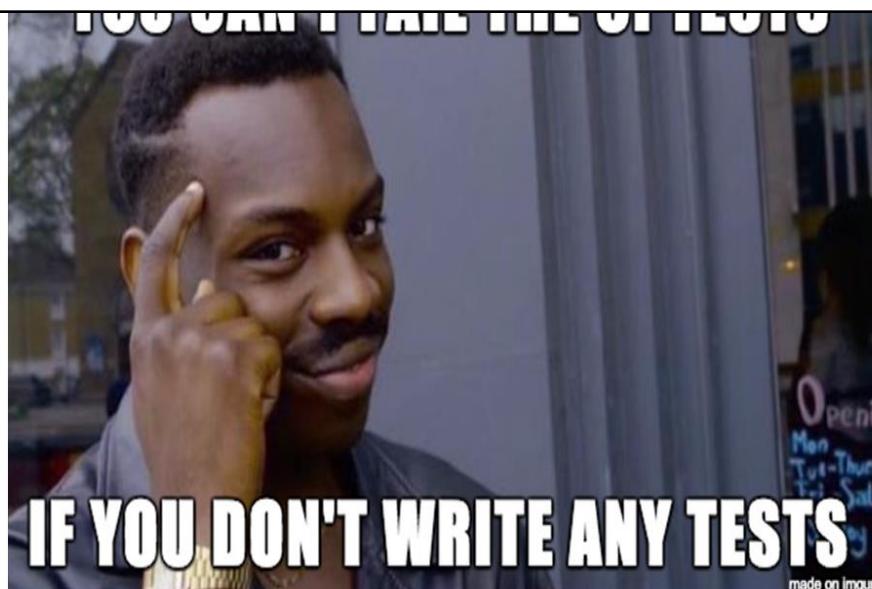
- Software development strategy that increases the speed of development while ensuring the quality of the code.
- Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.
- Continuous integration works hand-in-hand with Agile methodologies.

The benefits of CI

- Improve team productivity and efficiency.
- Catch issues early.
- Accelerate time to market.
- Release higher quality, more stable products.
- Increase customer satisfaction.
- Say goodbye to long and tense integrations.

Continuous Integration Practices

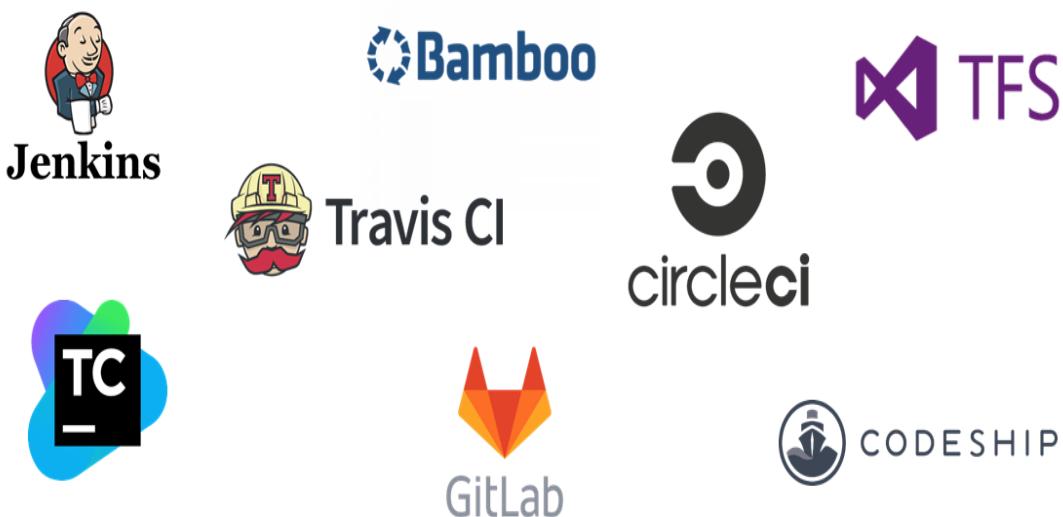
- Maintain a code repository
- Automate the build
- Make the build self-testing
- Everyone commits to the baseline every day
- Every commit (to baseline) should be built
- Keep the build fast
- Make testing an integral part of the development process



How Jenkins Fit In?

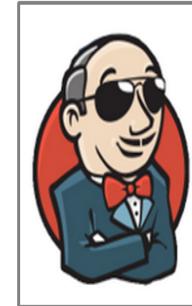
- Jenkins is a open source automation server which can be used to **automate** all sorts of tasks related to building, testing, and delivering or deploying software.
- Jenkins offers a simple way to set up a continuous integration environment for almost any combination of languages and source code repositories using pipelines.
- Many plugins that can do just about anything.

Most Used CI Tools



History of Jenkins

- ★ Jenkins was originally developed as the Hudson project in 2004 by Kohsuke Kawaguchi.
- ★ Kohsuke was a developer at Sun and got tired of incurring the wrath of his team every time his code broke the build.
- ★ In 2011 the project was split from Hudson and renamed to Jenkins.
- ★ Open Source and written in Java.



Questions



Module 02 - A little bit about Linux

Contents:

Agenda	3
History of Linux	3
Linux Distributions.....	4
Linux Distributions.....	4
Filesystem Hierarchy Standard.....	5
The Terminal	5
The Terminal – Basic commands	6
VIM a Linux Editor.....	6
VIM a Linux Editor.....	7
A look to Ubuntu Distribution.....	7
A look to Ubuntu Distribution - Features	8
Lab: Lab 1: Working with Ubuntu.....	9



Module 02: A Little Bit About Linux

Continuous Integration with Jenkins and Artifactory

Agenda

- ❖ History of Linux
- ❖ Linux Distributions
- ❖ Filesystem Hierarchy Standard
- ❖ The Terminal
- ❖ VIM a Linux Editor
- ❖ A look to Ubuntu Distribution
- ❖ Lab 01: Working with Ubuntu

History of Linux

- ❖ Bases on UNIX operation system.
- ❖ Created by a Finnish student "Linus Torvalds" in 1991.
- ❖ Linus created Linux because he wanted a free open-source OS.
- ❖ Development was done on MINIX using the GNU C Compiler.



Linux Distributions

- ★ A **Linux distribution** is an operating system made from a software collection, which is based upon the Linux kernel and, often, a package management system.
- ★ There are multiple different Linux distributions, Many have different philosophies.

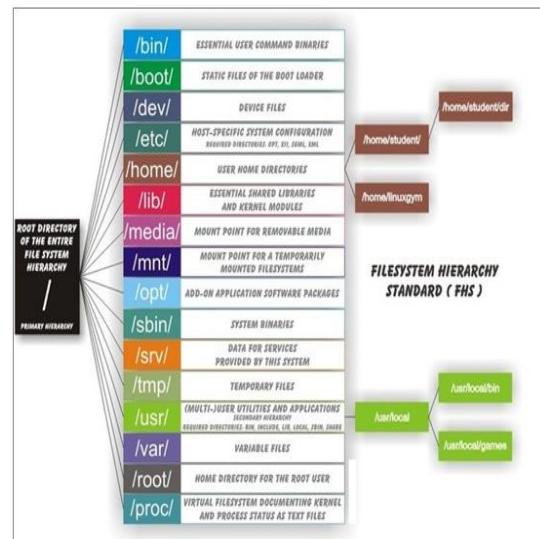
Linux Distributions



Filesystem Hierarchy Standard

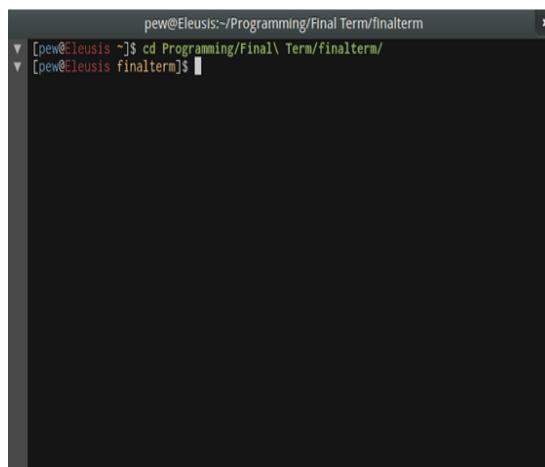
- 👉 The **FHS** defines the directory structure and directory contents in Linux distributions.

- 👉 All the files and directories appear under the root directory `/`



The Terminal

- 👉 The terminal is the command line of Linux.
- 👉 To the most Linux distributions has no UI, only terminal.



The Terminal – Basic commands

- 👉 ls - List information about file(s)
- 👉 cd - Change Directory
- 👉 mv - Move or rename files or directories
- 👉 man - Help manual
- 👉 mkdir - Create new folder(s)
- 👉 rmdir - Remove folder(s)
- 👉 touch - Change file timestamps
- 👉 rm - Remove files
- 👉 locate - Find files
- 👉 clear - terminal screen

VIM a Linux Editor

- 👉 Vim is a powerful text editor program for Linux.
- 👉 The name "Vim" is an acronym for "Vi IMproved" because Vim is an extended version of the vi editor.
- 👉 To install Vim you run this command in the terminal:

```
sudo apt-get install vim
```



VIM a Linux Editor

Basic Vim Commands

- :w Write the current file
- :wq Write the current file and exit.
- :q! Quit without writing
- To change into insert mode: i or a
 - Use escape to exit
- search forward /, repeat the search backwards: N
- Basic movement:
 - h l k j character left, right; line up, down (also arrow keys)
 - b w word/token left, right
 - g e end of word/token left, right
 - 0 \$ jump to first/last character on the line
- x delete
- u undo

```
#include "Arduino.h"
#define WLED 13 // Most Arduino boards already have an LED attached to pin 13 on the board itself
void setup()
{
  pinMode(WLED, OUTPUT); // set pin as output
}

void loop()
{
  digitalWrite(WLED, HIGH); // set the LED on
  delay(1000); // wait for a second
  digitalWrite(WLED, LOW); // set the LED off
  delay(1000); // wait for a second
}

[env:arduino_prosv]
platform = atmelavr
framework = arduino
board = proMk2atzmega168

# put here your device port.
# For Windows OS use COM1...COM9 ports
upload_port = /dev/ttyUSB0

[env:platformio]
# Uncomment lines below if you have problems with $PATH
SHELL := /bin/bash
PATH := /usr/local/bin:$PATH

all:
  platformio run -t upload
clean:
  platformio run -t clean
```

NORMAL > ~/Downloads/PlatformIO/Makefile make

A look to Ubuntu Distribution

- ➔ Ubuntu is a Linux-based operating system.
- ➔ Designed for computers, smartphones, and network servers.
- ➔ Have a Desktop version with UI.
- ➔ Of course – Open Source!



A look to Ubuntu Distribution - Features

- Office software
- Web browsing
- Email
- Photos
- Edit and illustrate
- Videos
- Gaming

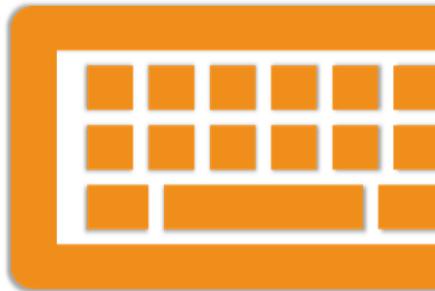


Questions



Lab: Lab 1: Working with Ubuntu

Lab



Module 03 - Jenkins Overview

Contents:

Agenda	3
Architecture	3
Installation	4
Backup and Restore.....	4
Configuration.....	5
Secure Jenkins.....	5
Jenkins Slaves	6
Jenkins Slaves	6
Jenkins Plugins	7
System Logs	7
Views	8
Jenkins Overview	9



Module 03: Jenkins Overview

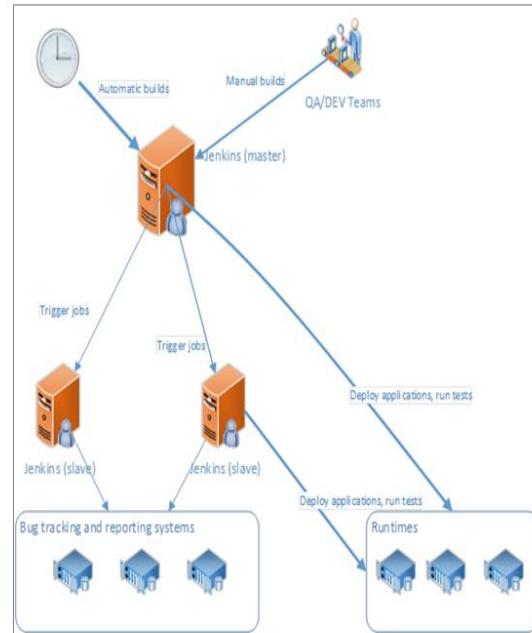
Continuous Integration with Jenkins and Artifactory

Agenda

- Architecture
- Installation
- Backup and Restore
- Configuration
- Secure Jenkins
- Jenkins Slaves
- Jenkins Plugins
- System Logs
- Jenkins Views

Architecture

- Jenkins uses a Master-Slave architecture to manage distributed builds



Installation

Jenkins can be installed on:

- Windows
- Linux
- Mac
- Docker.



For most platforms you have native packages.

Backup and Restore

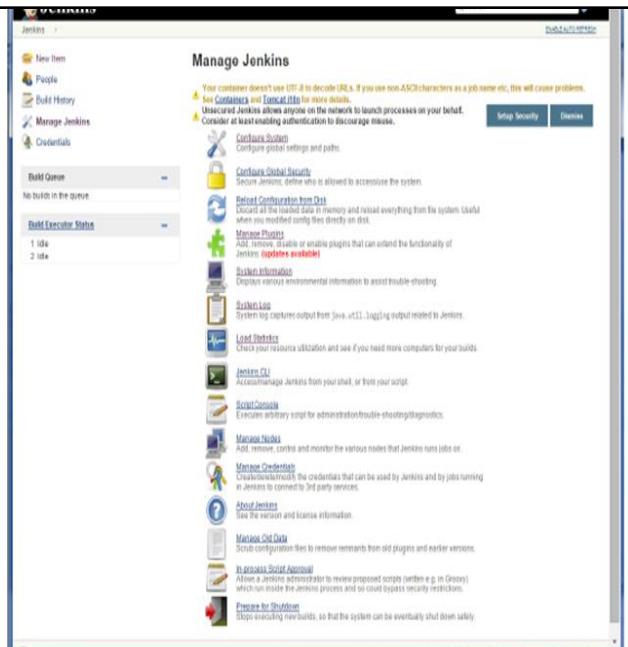
➤ Jenkins stores the job configuration in **config.xml** file that is present in the root directory of job.

➤ There are several plugins in Jenkins for backup.

```
<project>
  <actions/>
  <description>
    task to merge develop branch to integration (MANUAL BUILD)
  </description>
  <keepDependencies>false</keepDependencies>
  <properties>
    <><com.coravy.hudson.plugins.github.GithubProjectProperty plugin="github@1.11.3">
      <projectUrl>https://github.com/rahulkhengare/automergejenkins</projectUrl>
    </com.coravy.hudson.plugins.github.GithubProjectProperty>
    </properties>
    <><scm class="hudson.plugins.git.GitSCM" plugin="git@2.3.5">
      <configVersion>2</configVersion>
      <userRemoteConfigs>
        <><hudson.plugins.git.UserRemoteConfig>
          <url>https://github.com/rahulkhengare/automergejenkins</url>
          <credentialsId>[REDACTED]</credentialsId>
        </hudson.plugins.git.UserRemoteConfig>
      </userRemoteConfigs>
      <branches>
        <><hudson.plugins.git.BranchSpec>
          <name>/develop</name>
        </hudson.plugins.git.BranchSpec>
      </branches>
      <doGenerateSubmoduleConfigurations>false</doGenerateSubmoduleConfigurations>
      <submoduleCfg class="list"/>
    </scm>
    <extensions>
      <><hudson.plugins.git.extensions.impl.PreBuildMerge>
        <options>
          <mergeRemote>origin</mergeRemote>
          <mergeTarget>integration</mergeTarget>
          <mergeStrategy>default</mergeStrategy>
          <fastForwardMode>FF</fastForwardMode>
        </options>
        </hudson.plugins.git.extensions.impl.PreBuildMerge>
      </extensions>
    </scm>
    <assignedNode>[REDACTED]</assignedNode>
    <canRun>false</canRun>
    <disabled>false</disabled>
    <blockBuildWhenDownstreamBuilding>false</blockBuildWhenDownstreamBuilding>
  </project>
```

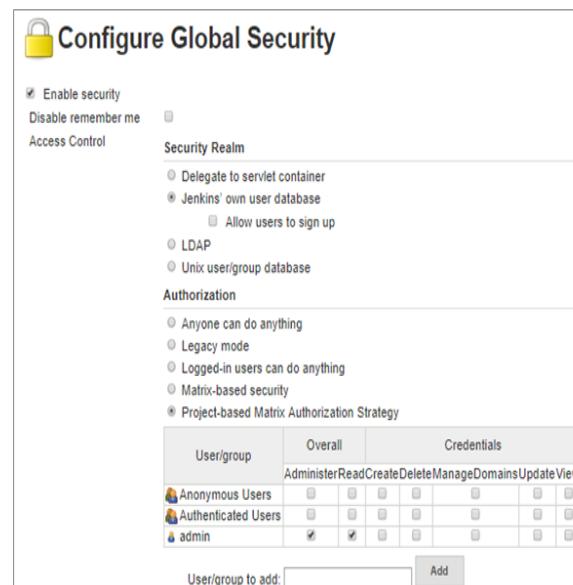
Configuration

- In the manage Jenkins page we can configure:
 - Workspace root directory,
 - Jobs naming pattern,
 - Email notification,
 - Jobs configuration,
 - Global security
 - Plugins



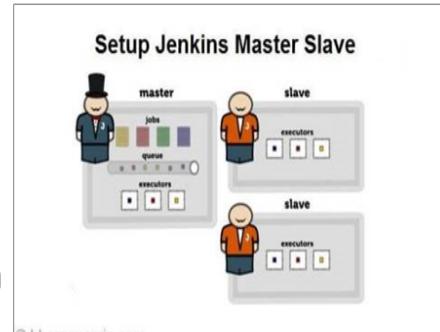
Secure Jenkins

- ★ In the configure global security we can add users or groups, and then decided which permissions we give them read only, create, delete etc.

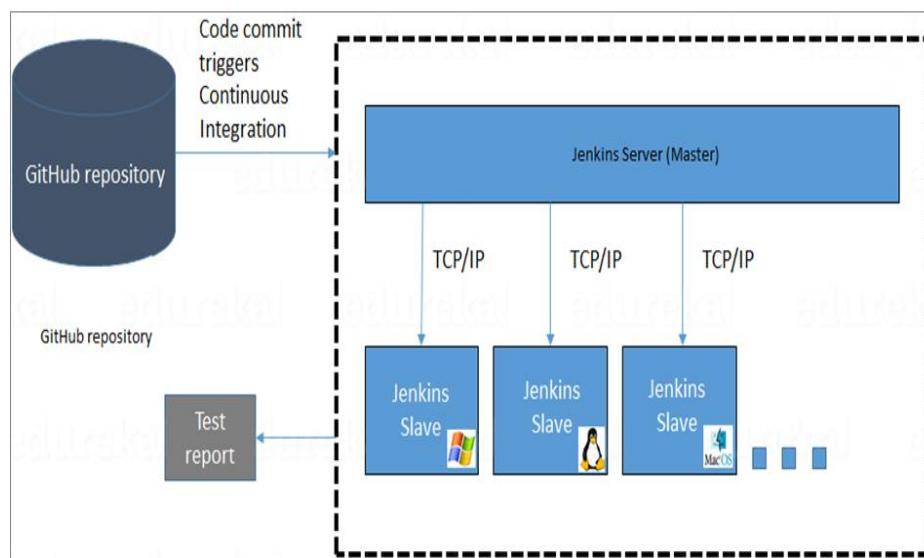


Jenkins Slaves

- 👉 A Slave is a Java executable that runs on a remote machine.
- 👉 It hears requests from the Jenkins Master instance.
- 👉 Slaves can run on a variety of operating systems.
- 👉 You can configure a project to always run on a particular Slave machine
- 👉 Slaves are the "build servers"

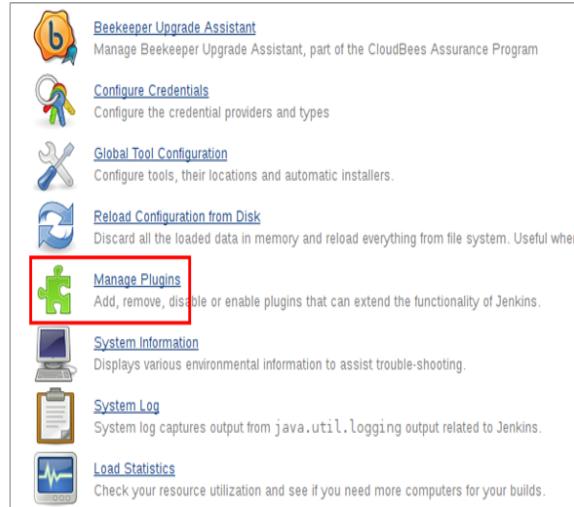


Jenkins Slaves



Jenkins Plugins

- ★ Are the primary way to enhance the functionality of a Jenkins environment, to suit organization or user-specific needs.
- ★ There are over a thousand different plugins available to be installed



System Logs

- ★ Every build, job, and failure in the Jenkins environment is saved in the system log.
- ★ System log can be configured and new loggers can be created.

Jenkins Log

```

May 27, 2018 7:24:58 AM INFO org.apache.jasper.servlet.TldScanner scanTlds
At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a complete list of JARS that we
improve startup time and JSP compilation time.

May 27, 2018 7:26:16 AM INFO jenkins.InitParameterRunner$1 onAttained
Started initialization
May 27, 2018 7:26:16 AM INFO jenkins.InitParameterRunner$1 onAttained
Listed all plugins
May 27, 2018 7:26:16 AM INFO jenkins.InitParameterRunner$1 onAttained
Prepared all plugins
May 27, 2018 7:26:16 AM INFO jenkins.InitParameterRunner$1 onAttained
Started all plugins
May 27, 2018 7:26:16 AM INFO hudson.Extension$Descriptor$DescriptorImpl error
Failed to instantiate optional component hudson.plugins.build_timeout.operations.AbortAndRestartOperationsDescriptorImpl; skipping
May 27, 2018 7:26:16 AM INFO jenkins.InitParameterRunner$1 onAttained
Augmented all extensions
May 27, 2018 7:26:17 AM INFO hudson.model.AsyncPeriodicTask$1 run
Loaded all jobs
May 27, 2018 7:26:17 AM INFO jenkins.InitParameterRunner$1 onAttained
Started Download metadata
May 27, 2018 7:26:18 AM INFO org.springframework.context.support.AbstractApplicationContext prepareRefresh
Refreshing org.springframework.web.context.support.StaticWebApplicationContext@6add8af72: display name [Root WebApplicationContext];
May 27, 2018 7:26:18 AM INFO org.springframework.context.support.AbstractApplicationContext obtainResourceUpdateFactory
Bean factory for application context [org.springframework.web.context.support.StaticWebApplicationContext@66d0af72]: org.springframework
May 27, 2018 7:26:18 AM INFO org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@7919ad64: defining beans [authen
May 27, 2018 7:26:18 AM INFO org.springframework.context.support.AbstractApplicationContext prepareRefresh
Refreshing org.springframework.web.context.support.StaticWebApplicationContext@1991e9b: display name [Root WebApplicationContext];
May 27, 2018 7:26:18 AM INFO org.springframework.context.support.AbstractApplicationContext obtainResourceUpdateFactory

```

Views

Allow us to organize jobs and content into tabbed categories.

Views are displayed on the main dashboard.

As a Jenkins instance expands, it is logical to create associated views for appropriate groups and categories.

Questions



Jenkins Overview

Demo



Module 04 - Anatomy of a Jenkins Job

Contents:

Agenda	3
General Configuration	3
Source Code Management	4
Build Triggers	4
Build Environment	5
Build Steps	5
Post-Build steps	6
Using plugins.....	6
Lab: Lab 02: Creating the first freestyle Jenkins job	7



Module 04: Anatomy of a Jenkins Job

Continuous Integration with Jenkins and Artifactory

Agenda

- ❖ General Configuration
- ❖ Source Code Management
- ❖ Build Triggers
- ❖ Build Environment
- ❖ Build Steps
- ❖ Post-Build steps
- ❖ Using plugins
- ❖ Lab 2: Creating the first freestyle Jenkins job

General Configuration

- ❖ Contains general information about the project, such as a unique name and description, and other information about how and where the build job should be executed.

The screenshot shows the 'General' configuration tab of a Jenkins job. It includes fields for 'Project name' (set to 'test3'), 'Description' (a large text area), and several checkboxes for project settings: 'Enable project-based security', 'Discard old builds', 'GitHub project', 'This project is parameterized' (which is checked), 'Throttle builds', 'Disable this project', and 'Execute concurrent builds if necessary'. There is also an 'Advanced...' button at the bottom right.

The screenshot shows the 'String Parameter' configuration dialog. It has a checkbox 'This project is parameterized' which is checked. It includes fields for 'Name' (a text input field), 'Default Value' (a text input field), and 'Description' (a text area). At the bottom, there are buttons for '[Plain text] Preview' and 'Trim the string'.

Source Code Management

- ★ Jenkins supports CVS and Subversion out of the box, with built-in support for Git, and also integrates with a large number of other version control systems via plugins.

The screenshot shows the 'Source Code Management' section of a Jenkins job configuration. The 'Git' option is selected. Under 'Repositories', there is a field for 'Repository URL' with a red error message: 'Please enter Git repository.' Below it is a 'Credentials' dropdown set to '- none -'. There are 'Advanced...' and 'Add Repository' buttons. Under 'Branches to build', a 'Branch Specifier' field contains '*master'. A 'Repository browser' dropdown is set to '(Auto)'. Under 'Additional Behaviours', there is an 'Add' button. The 'Subversion' option is also present.

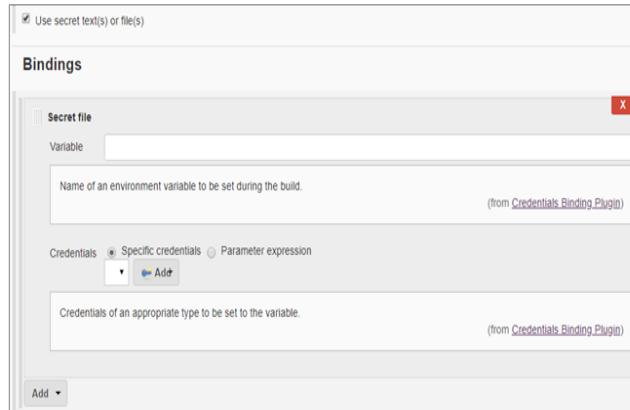
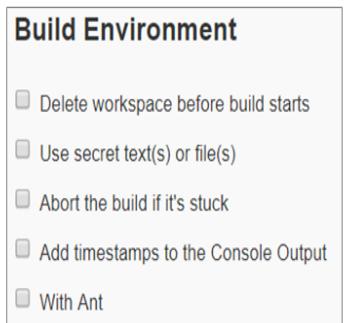
Build Triggers

- ★ Once you have configured your version control system, you need to tell Jenkins when to kick off a build. You set this up in the Build Triggers section.

The screenshot shows the 'Build Triggers' section of a Jenkins job configuration. It lists several trigger options: 'Trigger builds remotely (e.g., from scripts)', 'Build after other projects are built', 'Build periodically', and 'Poll SCM' (which is checked). Below these is a 'Schedule' field containing '*****'. A warning message at the bottom says: '⚠ Do you really mean "every minute" when you say "*****"? Perhaps you meant "H *****" to poll once per hour'. At the bottom, there is an 'Ignore post-commit hooks' checkbox.

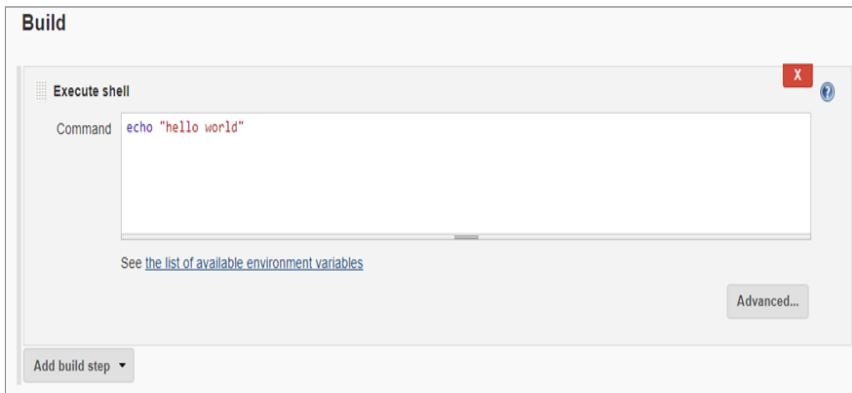
Build Environment

- Now that you have your build triggers lets configure the environment in which Jenkins will operate.



Build Steps

- This section allows the developer to add any batch commands they wish to have run as part of the Jenkins freestyle job



Post-Build steps

- Once the build is completed, there are still a few things you need to look after. You might want to archive some of the generated artifacts, to report on test results, and to notify people about the results.

Post-build Actions

E-mail Notification

Recipients

Whitespace-separated list of recipient addresses. May reference build parameters like \$PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.

Send e-mail for every unstable build

Send separate e-mails to individuals who broke the build

Add post-build action ▾

Using plugins

- Plugins make Jenkins what it is, very adaptable and customizable.
- To enable a plugin first make sure it is selected in the plug in manager.

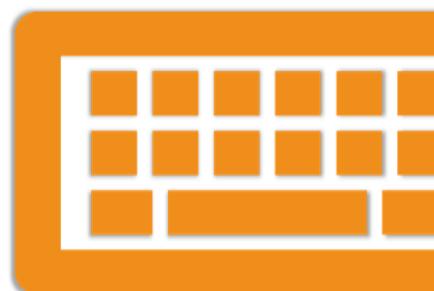
Enabled	Name ↓	Version	Previously installed version	Uninstall
<input checked="" type="checkbox"/>	Ant Plugin Adds Apache Ant support to Jenkins	1.8		Uninstall
<input checked="" type="checkbox"/>	Apache HttpClient 4.x API Bundles Apache HttpClient 4.x and allows it to be used by Jenkins plugins.	4.5.3-2.0		Uninstall
<input checked="" type="checkbox"/>	Authentication Tokens API Plugin This plugin provides an API for converting credentials into authentication tokens in Jenkins.	1.3		Uninstall
<input checked="" type="checkbox"/>	bouncycastle API Plugin This plugin provides an stable API to Bouncy Castle related tasks.	2.16.2		Uninstall
<input checked="" type="checkbox"/>	Branch API This plugin provides an API for multiple branch based projects.	2.0.19		Uninstall
<input checked="" type="checkbox"/>	Build Timeout This plugin allows builds to be automatically terminated after the specified amount of time has elapsed.	1.19		Uninstall
<input checked="" type="checkbox"/>	ChuckNorris Plugin ChuckNorris plugin displays a picture of Chuck Norris (instead of Jenkins the buller) and a random Chuck Norris 'The Programmer' fact on each build page.	1.1		Uninstall

Questions



Lab: Lab 02: Creating the first freestyle Jenkins job

Lab



Module 05 - Package Management with Artifactory

Contents:

Agenda	3
What is Package Management?	3
Package Management Systems	4
What is JFrog Artifactory.....	4
Automating Processes	5
Artifactory Editions	5
Repository Types	6
Generic Repositories Overview	6
Lab: Lab 03: Deploying build artifacts from Jenkins to Artifactory	8



Module 05: Package Management with Artifactory

Continuous Integration with Jenkins and Artifactory

Agenda

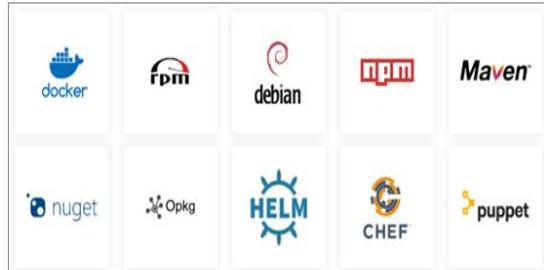
- What is Package Management?
- Package Management Systems
- What is JFrog Artifactory?
- Artifactory Editions
- Generic Repositories Overview
- Lab 3: Deploying build artifacts from Jenkins to Artifactory

What is Package Management?

- Is a collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs for a computer's operating system in a consistent manner.



Package Management Systems



What is JFrog Artifactory

- ★ JFrog Artifactory is an open source project
- ★ It's an advanced repository manager
- ★ Manage all binary artifacts efficiently in a single place
- ★ Accelerate your development workflow
- ★ Manage full artifact lifecycle



Automating Processes

- ★ JFrog CLI
 - ★ Client that provides a simple interface to interact with Jfrog products
- ★ Rest API
 - ★ Artifactory exposes a useful Rest API that can be used to interact with the server.



Artifactory Editions

- ★ Artifactory OSS
- ★ Artifactory Pro
- ★ Artifactory SaaS
- ★ Artifactory SaaS (dedicated server)
- ★ Artifactory Enterprise

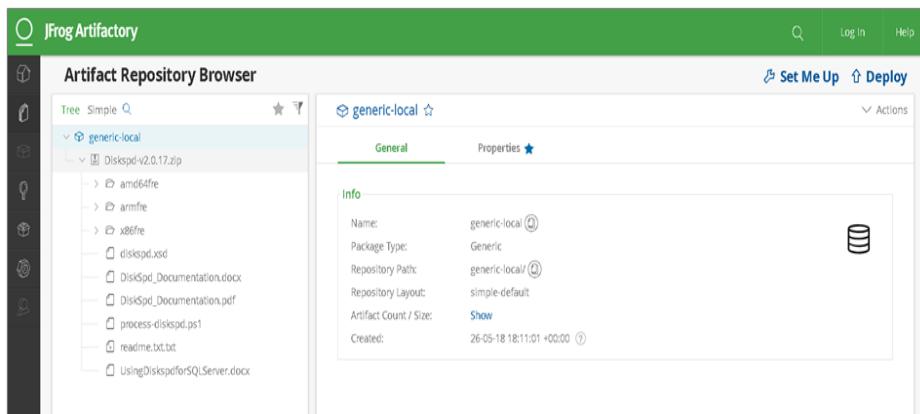


Repository Types

- 👉 Local Repositories
 - 👉 Are physical, locally-managed repositories into which you can deploy artifacts.
- 👉 Remote Repositories
 - 👉 Serves as a caching proxy for a repository managed at a remote URL.
- 👉 Virtual Repositories
 - 👉 Aggregates several repos with the same package type under a common URL.

Generic Repositories Overview

- 👉 It's a repository without any particular packaging type
- 👉 You can upload packages of any type (zip/tar files for example)



Demo

Generic Repositories

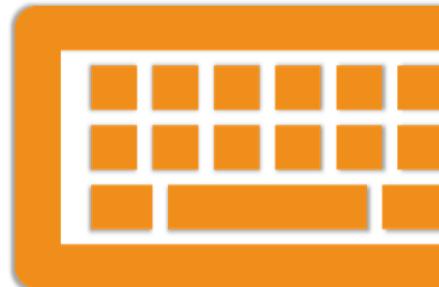


Questions



Lab: Lab 03: Deploying build artifacts from Jenkins to Artifactory

Lab



Module 06 - Jenkins pipelines as code

Contents:

Agenda	3
Pipeline as Code	3
What is Jenkins Pipelines?	4
Benefits of Jenkins Pipelines?	4
Meeting the Jenkinsfile.....	5
Declarative Pipelines.....	5
Declarative Pipelines.....	6
Declarative Pipelines.....	6
Declarative Pipelines.....	7
Declarative Pipelines.....	7
Declarative Pipelines.....	8
Declarative Pipelines.....	8
Declarative Pipelines.....	9
Declarative Pipelines.....	9
Scripted Pipelines	10
Scripted Pipelines	10
Scripted Pipelines	11
Scripted Pipelines	11
Scripted Pipelines	12
Scripted Pipelines	12
Lab: Lab 4: CI jobs with Jenkins pipelines	13



Module 06: Jenkins Pipelines as Code

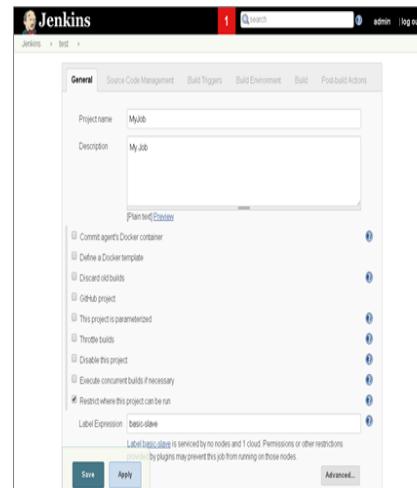
Continuous Integration with Jenkins and Artifactory

Agenda

- Pipeline as Code
- What is Jenkins Pipeline?
- Benefits of Jenkins Pipelines?
- Meeting the Jenkinsfile
- Declarative Pipelines
- Scripted Pipelines
- Lab 4: Continuous Integration jobs with Jenkins pipelines

Pipeline as Code

- The default interaction model with Jenkins, historically, has been very web UI driven.
- Requires users to manually create jobs, then manually fill in the details through a web browser.
- With the introduction of the Jenkins Pipeline, users now can implement a project's entire build/test/deploy pipeline as code.



What is Jenkins Pipelines?

It's just a plugin!

Benefits of Jenkins Pipelines?

- 👉 Code review/iteration on the Pipeline
- 👉 Audit trail for the Pipeline
- 👉 Single source of truth for the Pipeline
- 👉 Is Resilient: pipeline executions can survive master restarts
- 👉 Is Pausable: pipelines can pause and wait for human input/approval
- 👉 Is Efficient: pipelines can restart from saved checkpoints
- 👉 Is Visualized: Pipeline StageView provides status at-a-glance dashboards



Meeting the Jenkinsfile

- ★ File used to defining the deployment pipeline through code rather than configuring a running CI/CD tool

```
1 stage('Integration Testing') {
2     steps {
3         echo 'Running Integration Tests..'
4         sh 'cypress run --record --key xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx -c baseUrl=https://<url.com>/'  
5
6         echo 'Running Visual Regression tests..'
7         sh 'npm run image-compare-ci'  
8     }
9 }
10 }
11 }
```

- ★ Pipeline supports two syntaxes, Declarative and Scripted Pipeline

Declarative Pipelines

- ★ All Declarative Pipelines must start with pipeline and include these 4 directives to be syntactically correct: agent, stages, stage, and steps.

```
1 pipeline {
2     agent any
3     stages {
4         stage('Build') {
5             steps {
6                 sh 'npm --version'  
7             }
8         }
9     }
10 }
```

Declarative Pipelines

- 👉 Step is a single command which performs a single action.
- 👉 A stage is a collection of related steps that share a common execution env.

```

1 pipeline {
2     agent any
3     stages {
4         stage('Build') {
5             steps {
6                 sh 'echo "Hello World"'
7                 sh '''
8                     echo "Multiline shell steps"
9                     ls -lah
10                '''
11            }
12        }
13    }
14 }
```

Declarative Pipelines

- 👉 A stage may be optionally skipped in a Pipeline based on criteria defined in the when section of the stage.

```

1 pipeline {
2     agent any
3     stages {
4         stage('Build') {
5             when {
6                 expression {
7                     "foo" == "bar"
8                 }
9             }
10            steps {
11                echo 'Building'
12            }
13        }
14        stage('Test') {
15            when {
16                environment name: 'JOB_NAME', value: 'foo'
17            }
18            steps {
19                echo 'Testing'
20            }
21        }
22        stage('Deploy') {
23            when {
24                branch 'master'
25            }
26            steps {
27                echo 'Deploying'
28            }
29        }
30    }
31 }
```

Declarative Pipelines

- ★ To maximize efficiency of your Pipeline some stages can be run in parallel if they do not depend on each other.

```

1 pipeline {
2   agent any
3   stages {
4     stage('Browser Tests') {
5       parallel {
6         stage('Chrome') {
7           steps {
8             echo "Chrome Tests"
9           }
10        }
11        stage('Firefox') {
12          steps {
13            echo "Firefox Tests"
14          }
15        }
16      }
17    }
18  }
19 }
```

Declarative Pipelines

- ★ The agent directive tells Jenkins where and how to execute the Pipeline, or subset thereof.
- ★ It is also possible to use a custom workspace directory on each agent using a relative or absolute path

```

1 pipeline {
2   agent {
3     node { label 'my-agent' }
4   }
5 }
```

```

1 pipeline {
2   agent {
3     node {
4       label 'my-defined-label'
5       customWorkspace '/some/other/path'
6     }
7   }
8 }
```

Declarative Pipelines

- ★ Pipeline is designed to easily use Docker images and containers.

```

1 pipeline {
2     agent {
3         node { label 'my-docker' }
4     }
5     stages {
6         stage("Build") {
7             agent {
8                 docker {
9                     reuseNode true
10                    image 'maven:3.5.0-jdk-8'
11                }
12            }
13            steps {
14                sh 'mvn install'
15            }
16        }
17    }
18 }
```

Declarative Pipelines

- ★ Environment variables can be set globally or per stage.
- ★ Build secrets or API tokens can be easily inserted into environment variables.

```

1 pipeline {
2     agent any
3     environment {
4         DISABLE_AUTH = 'true'
5         DB_ENGINE      = 'sqlite'
6     }
7     stages {
8         stage('Build') {
9             steps {
10                sh 'printenv'
11            }
12        }
13    }
14 }
```

```

1 environment {
2     AWS_ACCESS = credentials('AWS_KEY')
3     AWS_SECRET = credentials('AWS_SECRET')
4 }
```

Declarative Pipelines

- This Post section of the is guaranteed to run at the end of the enclosing Pipeline or Stage.

```

1 pipeline {
2   agent any
3   stages {
4     stage('No-op') {
5       steps {
6         sh 'ls'
7       }
8     }
9   }
10  post {
11    always {
12      echo 'I have finished'
13      deleteDir() // clean up workspace
14    }
15    success {
16      echo 'I succeeded!'
17    }
18    unstable {
19      echo 'I am unstable :/'
20    }
21    failure {
22      echo 'I failed :('
23    }
24    changed {
25      echo 'Things are different...'
26    }
27  }
28 }
```

Declarative Pipelines

- Email/Slack Notification

```

1 post {
2   failure {
3     mail to: 'team@example.com',
4       subject: 'Failed Pipeline',
5       body: "Something is wrong"
6   }
7 }
```

```

1 post {
2   success {
3     slackSend channel:'#ops-room',
4       color: 'good',
5       message: 'Completed successfully.'
6   }
7 }
```

- Often, when passing between stages, especially environment stages, you may want human input before continuing.

```

1 stage('Sanity check') {
2   steps {
3     input "Does the staging environment look ok?"
4   }
5 }
```

Scripted Pipelines

- ★ Unlike Declarative, Scripted Pipeline is effectively a general purpose DSL built with Groovy.
- ★ It can be a very expressive and flexible tool with which one can author continuous delivery pipelines.
- ★ The first block to be defined is the "node" block:

```
node{  
}
```

Scripted Pipelines

- ★ The pipeline will consist of several steps that can be grouped in stages.

```
node {  
  
    stage ('Build') {  
        bat "msbuild ${C:\\\\Jenkins\\\\my_project\\\\workspace\\\\test\\\\my_project.sln}"  
    }  
  
    stage('Selenium tests') {  
        dir(automation_path) {  
            bat "mvn clean test -Dsuite=SMOKE_TEST -Denvironment=QA"  
        }  
    }  
}
```

Scripted Pipelines

- ★ You can use Groovy expressions, such as the if/else conditionals

```
node {  
  
    stage('Example') {  
        if (env.BRANCH_NAME == 'master') {  
            echo 'I only execute on the master branch'  
        } else {  
            echo 'I execute elsewhere'  
        }  
    }  
}
```

Scripted Pipelines

- ★ You can use Groovy expressions, such as the try/catch/finally conditionals

```
node {  
    stage('Example') {  
        try {  
            sh 'exit 1'  
        }  
        catch (exc) {  
            echo 'Something failed, I should sound the klaxons!'  
            throw  
        }  
    }  
}
```

Scripted Pipelines

- 👉 Use stash/unstash to do some more stuff on a different slave

```
node("unix1") {  
    git url: "https://github/project.git"  
    sh("mvn clean package")  
    stash includes: "target/artifact.jar" name "artifact"  
}  
  
node("unix2") {  
    unstash name: "artifact"  
    sh("cf push -n artifact.jar")  
}
```

Scripted Pipelines

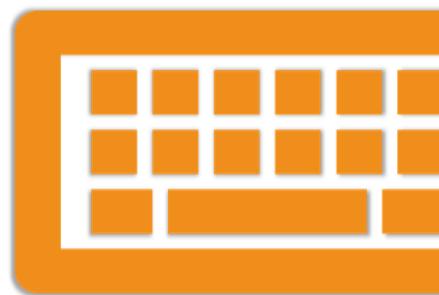
```
node() {  
    git poll: false, changelog: false, url: workflowLibsRepo, credentialsId:  
gitCredentialsIdForWorkflowLibs, branch: branch  
  
    common = load "lib/common.groovy"  
    common.assertMandatoryParameters(parameters)  
    common.logParameterValues(parameters)  
}  
node() {  
    git poll: false, changelog: false, url: workflowLibsRepo, credentialsId:  
gitCredentialsIdForWorkflowLibs, branch: branch  
  
    cf = load "lib/cf.groovy"  
    cf.mapRoute(appName, majorHostName, cfSpace, cfOrg, cfApiEndpoint,  
pcfCredentialsId)  
}
```

Questions



Lab: Lab 4: CI jobs with Jenkins pipelines

Lab



Module 07 - Jenkins with Blue Ocean

Contents:

Agenda	3
What is Jenkins Blue Ocean	4
Main Features	4
Main Features	5
Main Features	5
Main Features	6
Main Features	6
Blue Ocean installation	7
Blue Ocean Overview	7
Blue Ocean Overview	8
Blue Ocean Overview	8
Blue Ocean Overview	9
Blue Ocean Overview	9
Blue Ocean Overview	10
Blue Ocean Overview	10
Lab: Lab 5: Creating a Jenkins job using Blue Ocean.....	11

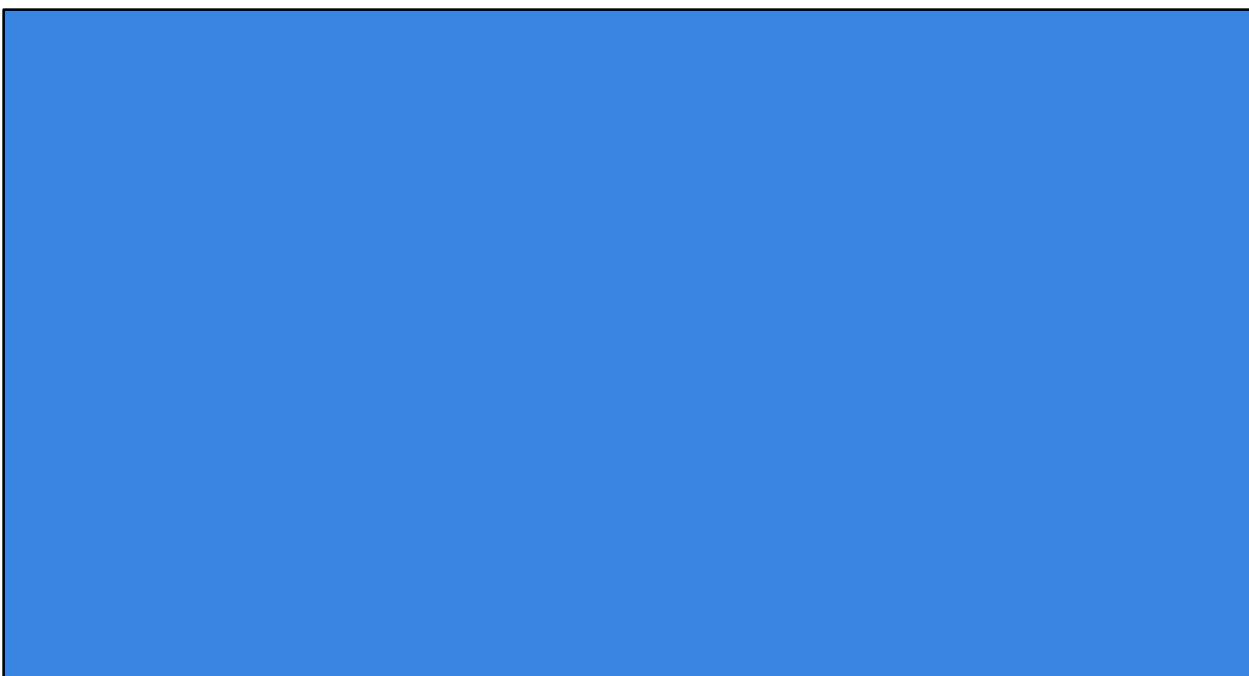


Module 07: Jenkins with Blue Ocean

Continuous Integration with Jenkins and Artifactory

Agenda

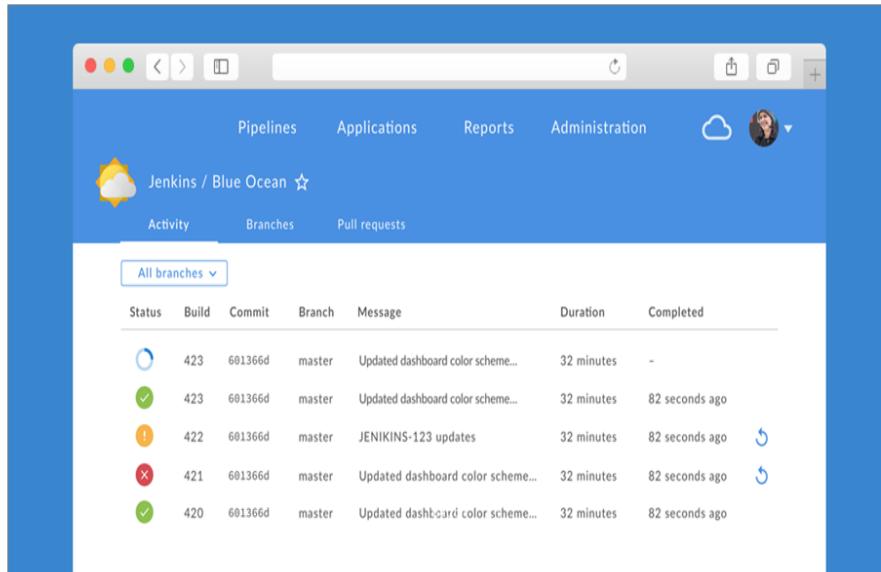
- ❖ What is Jenkins Blue Ocean?
- ❖ Main Features
- ❖ Installation
- ❖ Overview
- ❖ Lab 5: Creating a Jenkins job using Blue Ocean



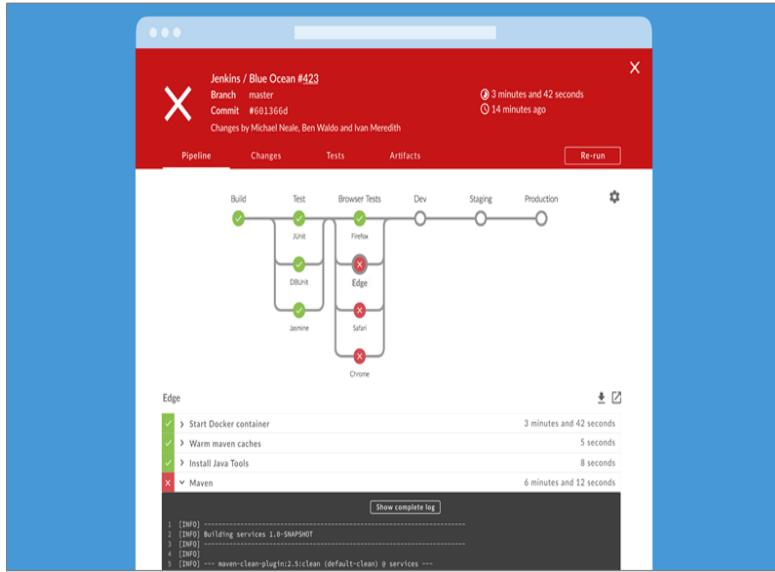
What is Jenkins Blue Ocean

- 👉 Blue Ocean is an open source project that rethinks the user experience of Jenkins.
- 👉 Jenkins users can install Blue Ocean side-by-side with the Jenkins Classic UI via a plugin.
- 👉 Designed from the ground up for Jenkins Pipeline, but still compatible with freestyle jobs

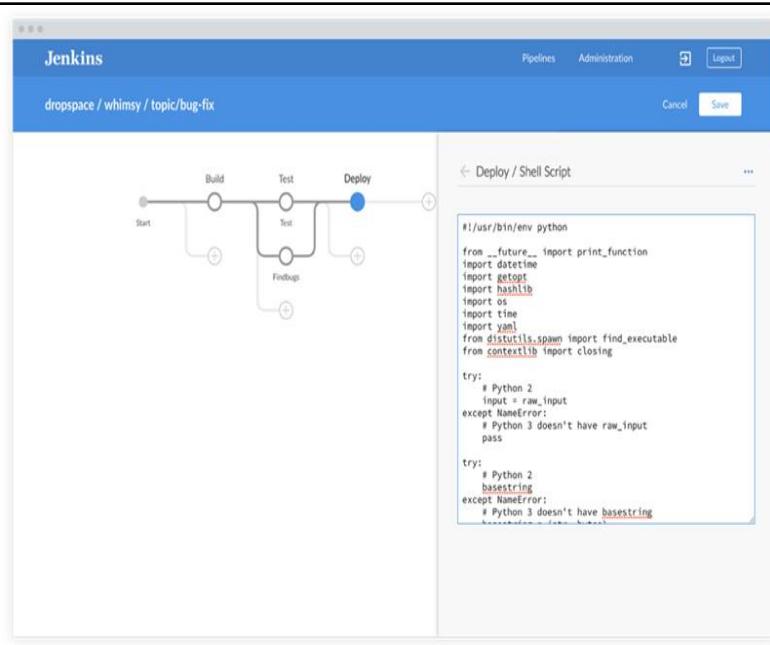
Main Features



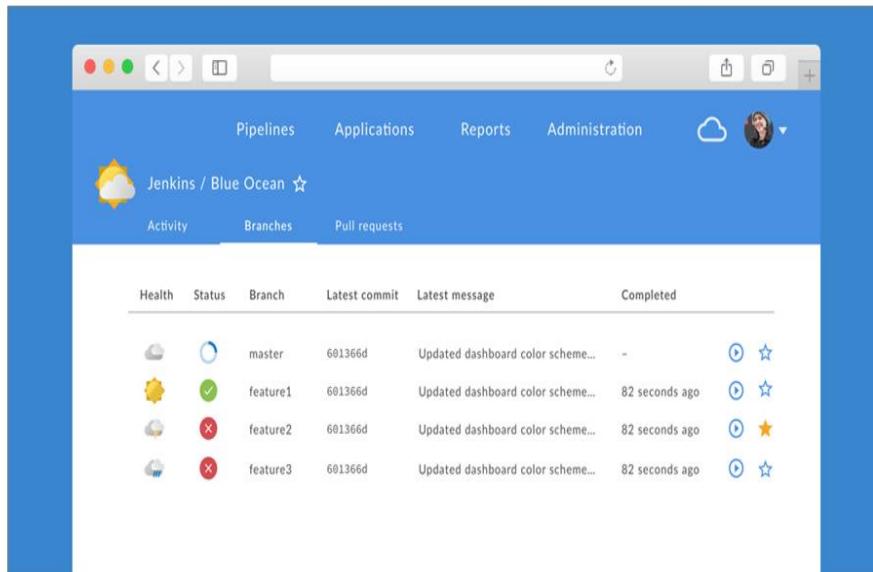
Main Features



Main Features



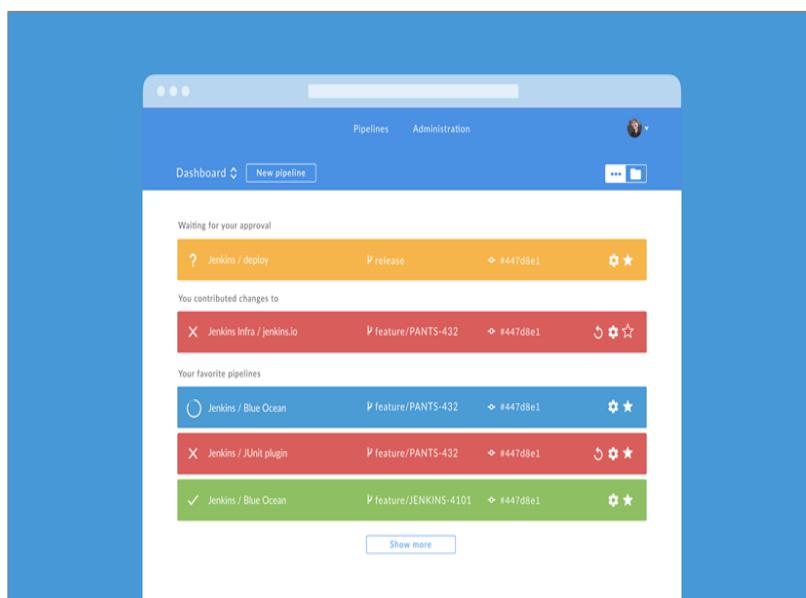
Main Features



The screenshot shows the Jenkins / Blue Ocean dashboard. At the top, there are tabs for Pipelines, Applications, Reports, and Administration. Below the tabs, there's a navigation bar with Activity, Branches, and Pull requests. The main content area displays a table of recent activity:

Health	Status	Branch	Latest commit	Latest message	Completed
		master	601366d	Updated dashboard color scheme...	-
		feature1	601366d	Updated dashboard color scheme...	82 seconds ago
		feature2	601366d	Updated dashboard color scheme...	82 seconds ago
		feature3	601366d	Updated dashboard color scheme...	82 seconds ago

Main Features



The screenshot shows the Jenkins / Blue Ocean dashboard. At the top, there are tabs for Pipelines and Administration. Below the tabs, there's a navigation bar with Dashboard, New pipeline, and other icons. The main content area displays a pipeline status card:

Waiting for your approval

	Jenkins / deploy		release		#447d8e1	
--	------------------	--	---------	--	----------	--

You contributed changes to

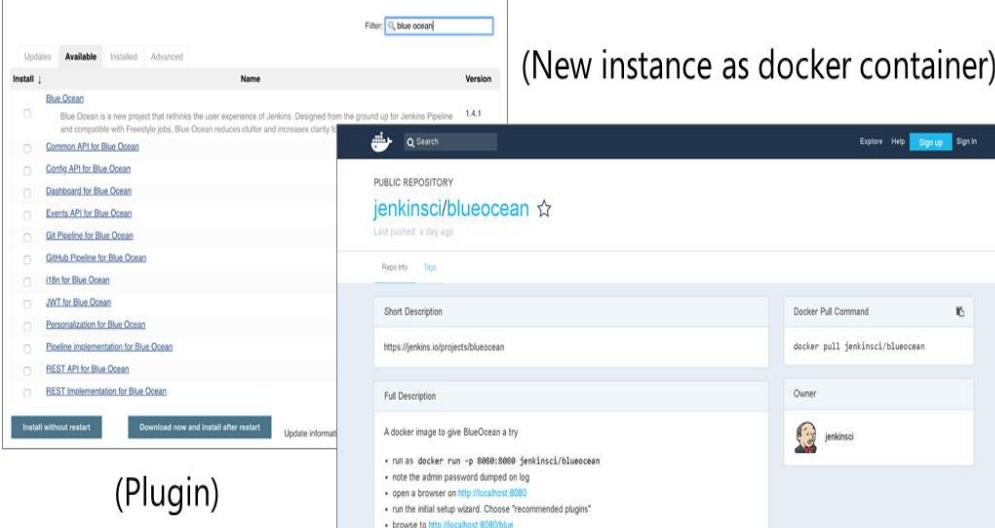
	Jenkins Infra / jenkins.io		feature/PANTS-432		#447d8e1	
--	----------------------------	--	-------------------	--	----------	--

Your favorite pipelines

	Jenkins / Blue Ocean		feature/PANTS-432		#447d8e1	
	Jenkins / JUnit plugin		feature/PANTS-432		#447d8e1	
	Jenkins / Blue Ocean		feature/JENKINS-4101		#447d8e1	

[Show more](#)

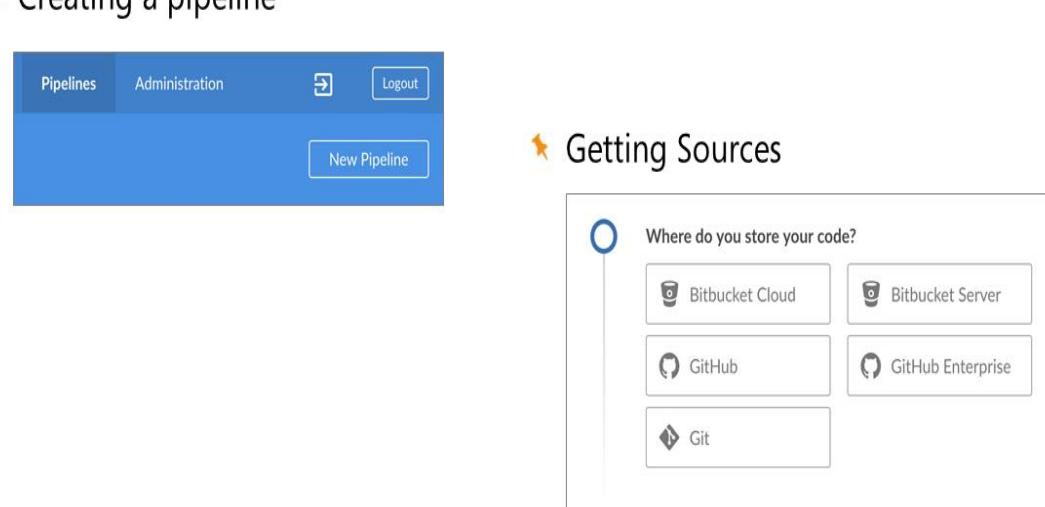
Blue Ocean installation



(New instance as docker container)

(Plugin)

Blue Ocean Overview

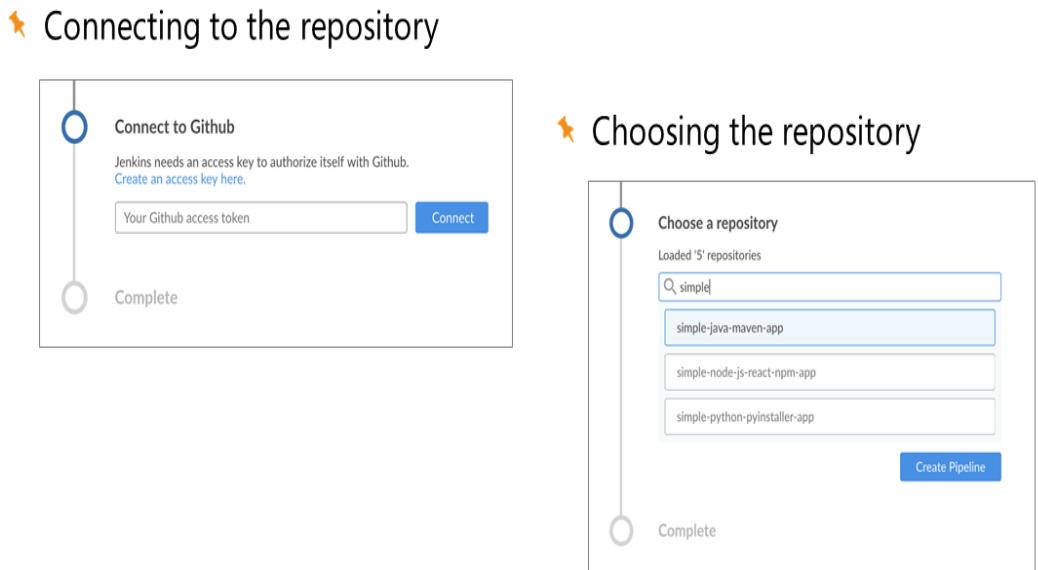


Creating a pipeline

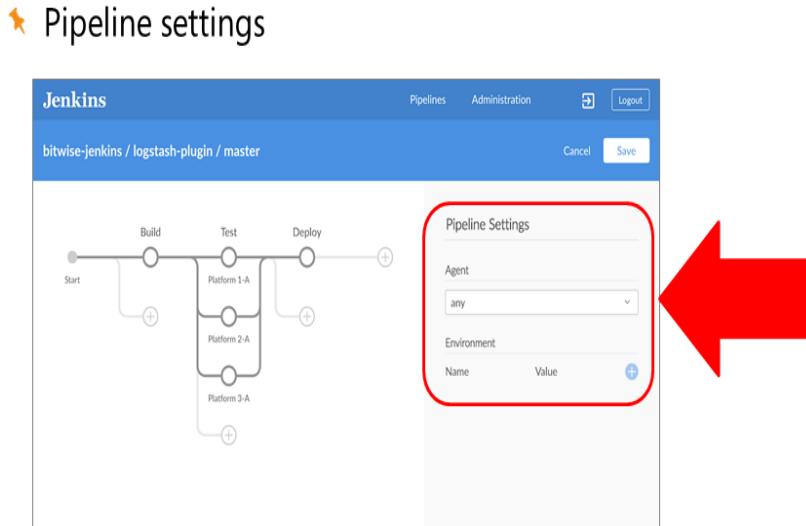
Getting Sources

- Where do you store your code?
- Bitbucket Cloud
- Bitbucket Server
- GitHub
- GitHub Enterprise
- Git

Blue Ocean Overview



Blue Ocean Overview



Blue Ocean Overview

Stage editor

Name your stage

There are no steps, at least one is required.

+ Add step

Choose step type

file

- Evaluate a Groovy source file into the Pipeline script
- Load a resource file from a shared library
- Read file from workspace

Step configuration

Blue Ocean Overview

Save pipeline

Save Pipeline

Saving the pipeline will commit a Jenkinsfile to the repository.

Description

What changed?

Commit to master

Commit to new branch

my-new-branch

Save & run Cancel

Activity View

Jenkins

bitwise-jenkins / junit-plugin

Status	Run	Commit	Branch	Message	Duration	Completed
Green	3	b518058	PR-7	-	4m 51s	a minute ago
Green	1	63a7f47	master	-	5s	8 minutes ago
Red	1	86b2229	PR-7	-	48s	7 minutes ago
Grey	6	d3d9a39	blog/blue-ocean-editor	-	1m 52s	12 minutes ago
Green	5	d3d9a39	blog/blue-ocean-editor	-	11m 8s	4 hours ago

Blue Ocean Overview

⭐ Dashboard

The screenshot shows the Jenkins Blue Ocean dashboard. On the left, there's a sidebar with a 'Dashboard' icon. The main area has a 'Favorites' section with five pipelines listed: 'building-a-multibranch-pipeline-project' (development branch, build #2ec7747, 1 minute ago), 'simple-node-js-react-npm-app' (master branch, build #be4fffd6, 31 minutes ago), 'simple-java-maven-app' (master branch, build #e616863, 24 minutes ago), 'building-a-multibranch-pipeline-project' (master branch, build #87393ab, a few seconds ago), and 'building-a-multibranch-pipeline-project' (production branch, build #e85d115, 7 minutes ago). Below this is a table showing project health: 'building-a-multibranch-pipeline-project' is failing (1 failing), 'simple-java-maven-app' is neutral, 'simple-node-js-react-npm-app' is neutral, and 'simple-python-pyinstaller-app' is passing (1 passing).

Blue Ocean Overview

⭐ Dashboard - Health Icons

Icon	Health
	Sunny, more than 80% of Runs passing
	Partially Sunny, 61% to 80% of Runs passing
	Cloudy, 41% to 60% of Runs passing
	Raining, 21% to 40% of Runs passing
	Storm, less than 21% of Runs passing

⭐ Dashboard - Run Status

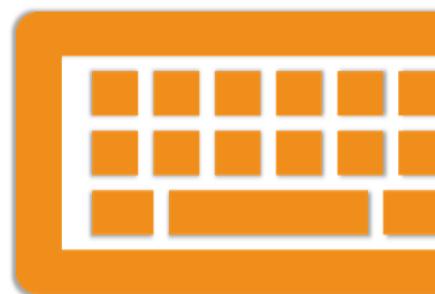
Icon	Status
	In Progress
	Passed
	Unstable
	Failed
	Aborted

Questions



Lab: Lab 5: Creating a Jenkins job using Blue Ocean

Lab



Module 08 - Summary

Contents:

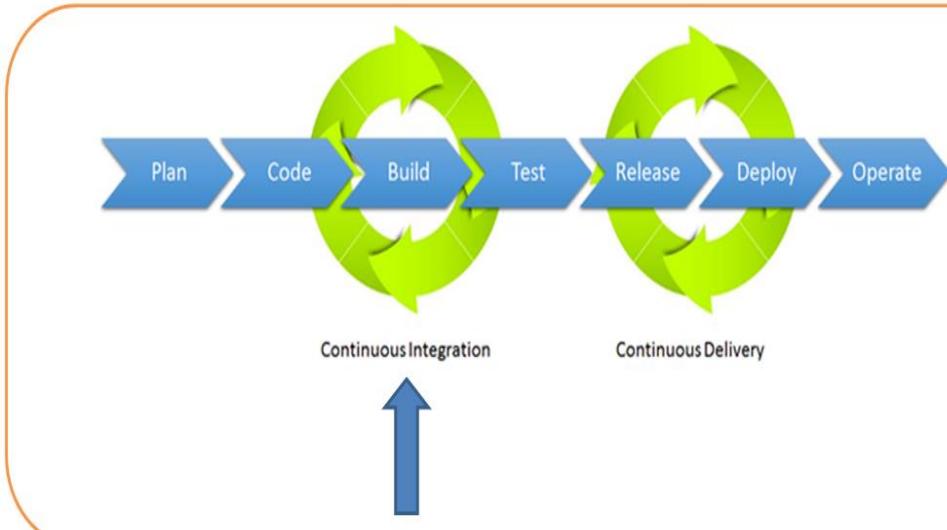
DevOps Pipelines.....	3
Linux Overview.....	3
Jenkins Overview	4
Jenkins Freestyle Jobs.....	4
Package Management with Artifactory	5
Jenkins Pipelines as Code	5
Jenkins Blue Ocean	6
Itshak Eli.....	7



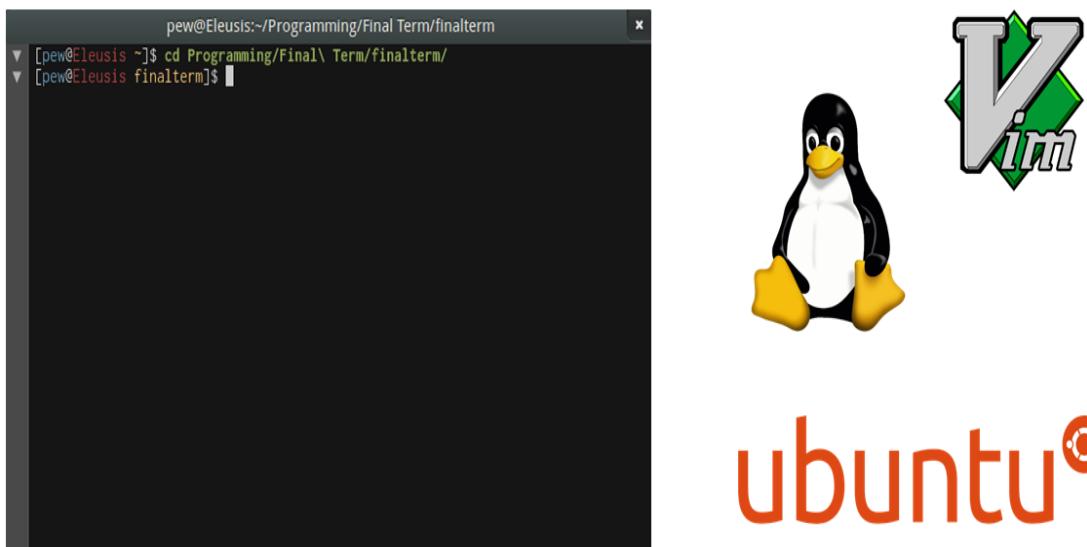
Module 08: Summary

Continuous Integration with Jenkins and Artifactory

DevOps Pipelines



Linux Overview



Jenkins Overview

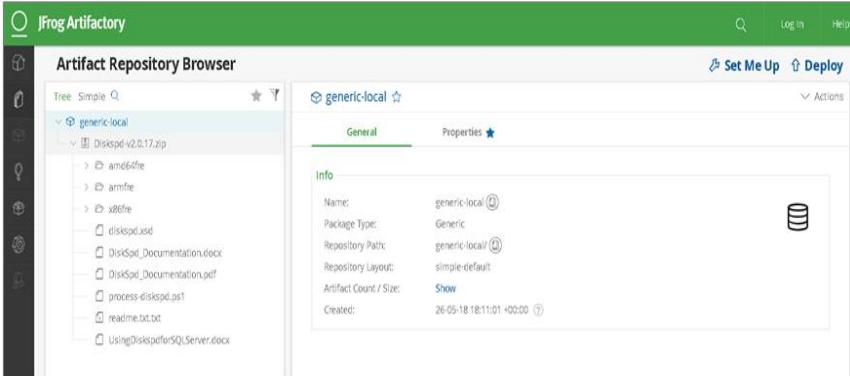


Jenkins Freestyle Jobs

The screenshot shows the configuration page for a Jenkins Freestyle job named "test2". The General tab is selected, showing the following settings:

- Project name:** test2
- Description:** (empty)
- Source Code Management:** Git (selected)
- Build Triggers:**
 - Trigger builds remotely (e.g., from scripts)
 - Build after other projects are built
 - Build periodically
 - Poll SCM** (checkbox checked)
- Schedule:** * * * * *
- Build:**
 - Execute shell**
 - Command:** echo "hello world"
- Advanced...** button

Package Management with Artifactory



The screenshot shows the JFrog Artifactory interface. On the left, there's a tree view of the repository structure under 'generic-local'. Inside 'generic-local' are several items: 'DiskSpd', 'DiskSpd.amd64fe', 'DiskSpd.armfe', 'DiskSpd.x86fe', 'DiskSpd.x86', 'DiskSpd.documentation.docx', 'DiskSpd.documentation.pdf', 'process-DiskSpd.ps1', and 'readme.txt.txt'. Below these is a file named 'UsingDiskSpdForSQL.Serverdeck'. On the right, there's a detailed view of the 'generic-local' repository. It shows the following properties:

- Name: generic-local
- Package Type: Generic
- Repository Path: generic-local/
- Repository Layout: simple-default
- Artifact Count / Size: Show
- Created: 26-05-18 18:11:01 +0000



Jenkins Pipelines as Code

```

1 pipeline {
2     agent any
3     stages {
4         stage('Browser Tests') {
5             parallel {
6                 stage('Chrome') {
7                     steps {
8                         echo "Chrome Tests"
9                     }
10                }
11                stage('Firefox') {
12                    steps {
13                        echo "Firefox Tests"
14                    }
15                }
16            }
17        }
18    }
19 }
```

```

node {

    stage('Example') {
        if (env.BRANCH_NAME == 'master') {
            echo 'Executed on the master branch'
        } else {
            echo 'I execute elsewhere'
        }
    }
}
```

Jenkins Blue Ocean

The screenshot shows the Jenkins Blue Ocean interface. On the left, there's a list of branches: feature/UX-311 (red), bug/UX-202 (red), master (green), UX-458 (yellow), UX-163 (red), and task/JENKINS-36884 (green). The master branch has a green checkmark and the status 'JENKINS-35'. A modal window is open for the 'jenkins / DevOps Camp / Parallel Docker #2' pipeline. The pipeline consists of three steps: 'whatever', 'run in docker', and 'deploy'. The 'run in docker' step is further divided into 'php7' and 'php5'. The 'run in docker' step has a green checkmark. The pipeline status is 'Completed' with a green checkmark. The last update was 'a few seconds' ago at '3 minutes ago'. Below the pipeline view, there's a code editor showing a 'Shell Script' step with the following content:

```
1 [workspace] Running shell script
2 + php -v
3 PHP 7.0.11 (cli) (built: Sep 23 2016 21:47:48) ( NTS )
4 Copyright (c) 1997-2016 The PHP Group
5 Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
```

Itshak Eli



Thank you for your attention!

Continuous Integration with Jenkins and Artifactory

Itshak Eli

itshak.elis@gmail.com