Ci with jenkins and
rtifactory - DevOps ®

Version 1.0

Sela College

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

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

Module 02 - A little bit about Linux

Module 03 - Jenkins Overview

Module 04 - Anatomy of a Jenkins Job

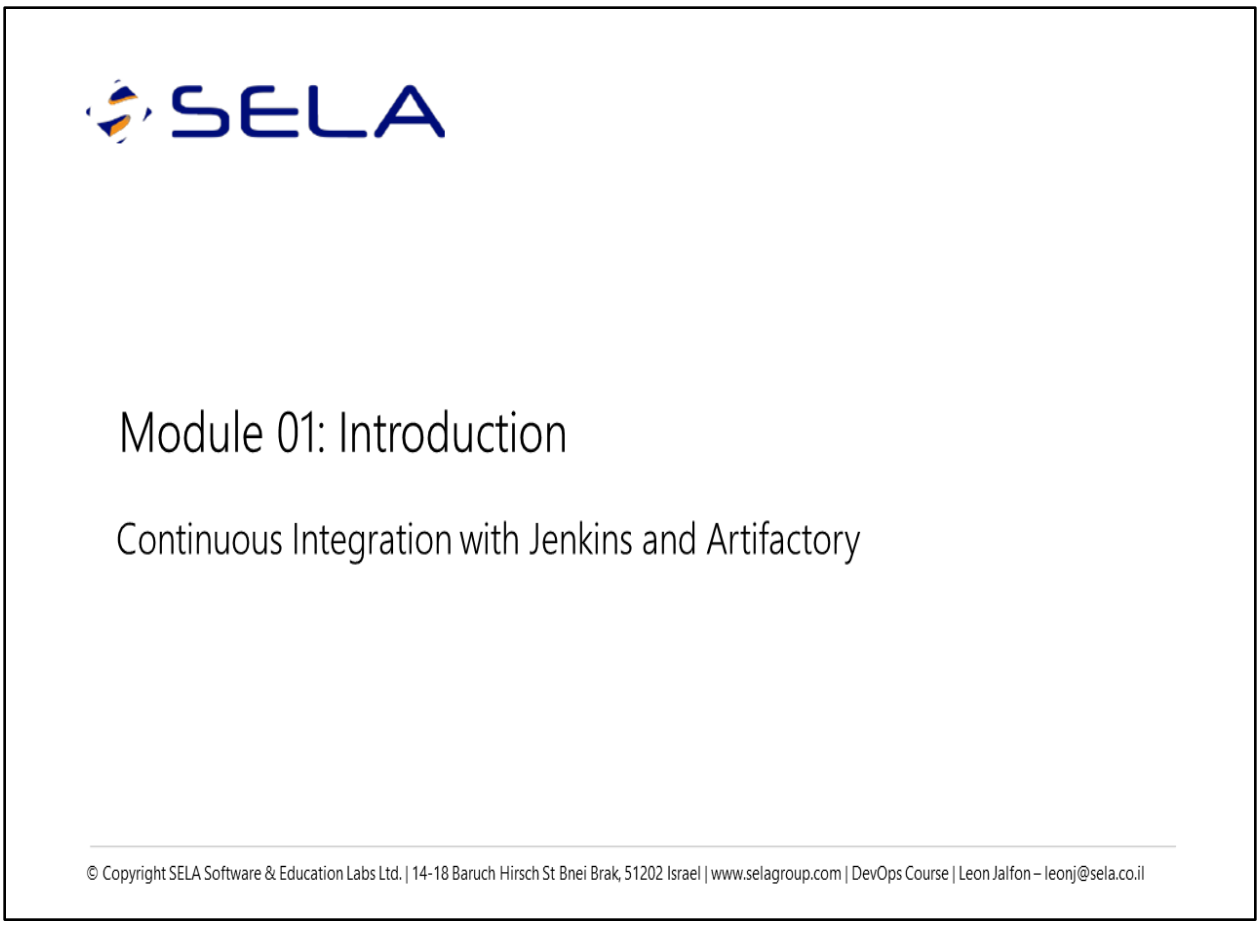
Module 06 - Jenkins pipelines as code

Module 07 - Jenkins with Blue Ocean

Module 08 - Summary

Module 01 - Introduction

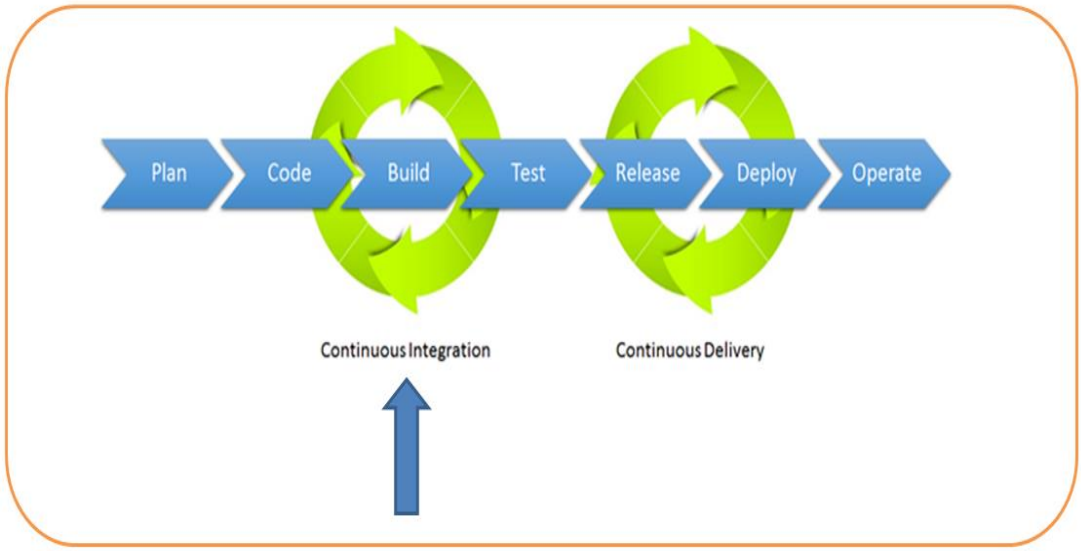
Contents:


SELA
Module 01: Introduction
Continuous Integration with Jenkins and Artifactory
© Copyright SELA Software & Education Labs Ltd. | 14-18 Baruch Hirsch St Bnei Brak, 51202 Israel | www.selagroup.com | DevOps Course | Leon Jalfon – leonj@sela.co.il

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?

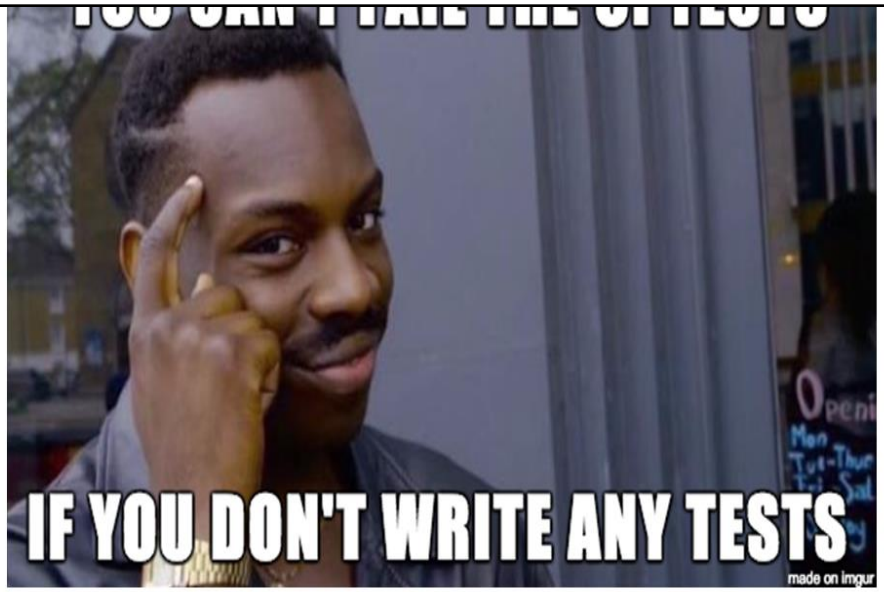
- Software development strategy that increases the speed of development while ensuring the quality of the code.
- Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.
- Continuous integration works hand-in-hand with Agile methodologies.

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

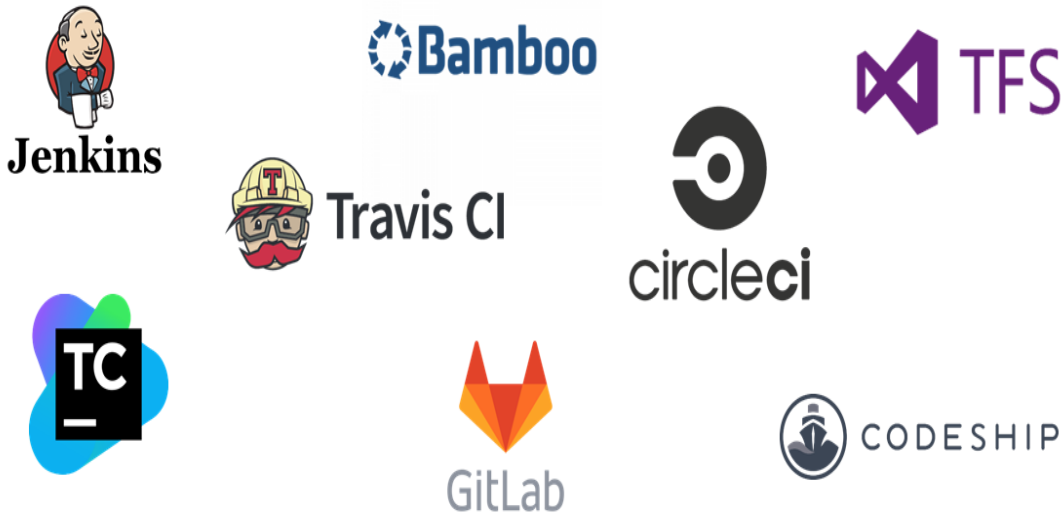
- Maintain a code repository
- Automate the build
- Make the build self-testing
- Everyone commits to the baseline every day
- Every commit (to baseline) should be built
- Keep the build fast
- Make testing an integral part of the development process



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



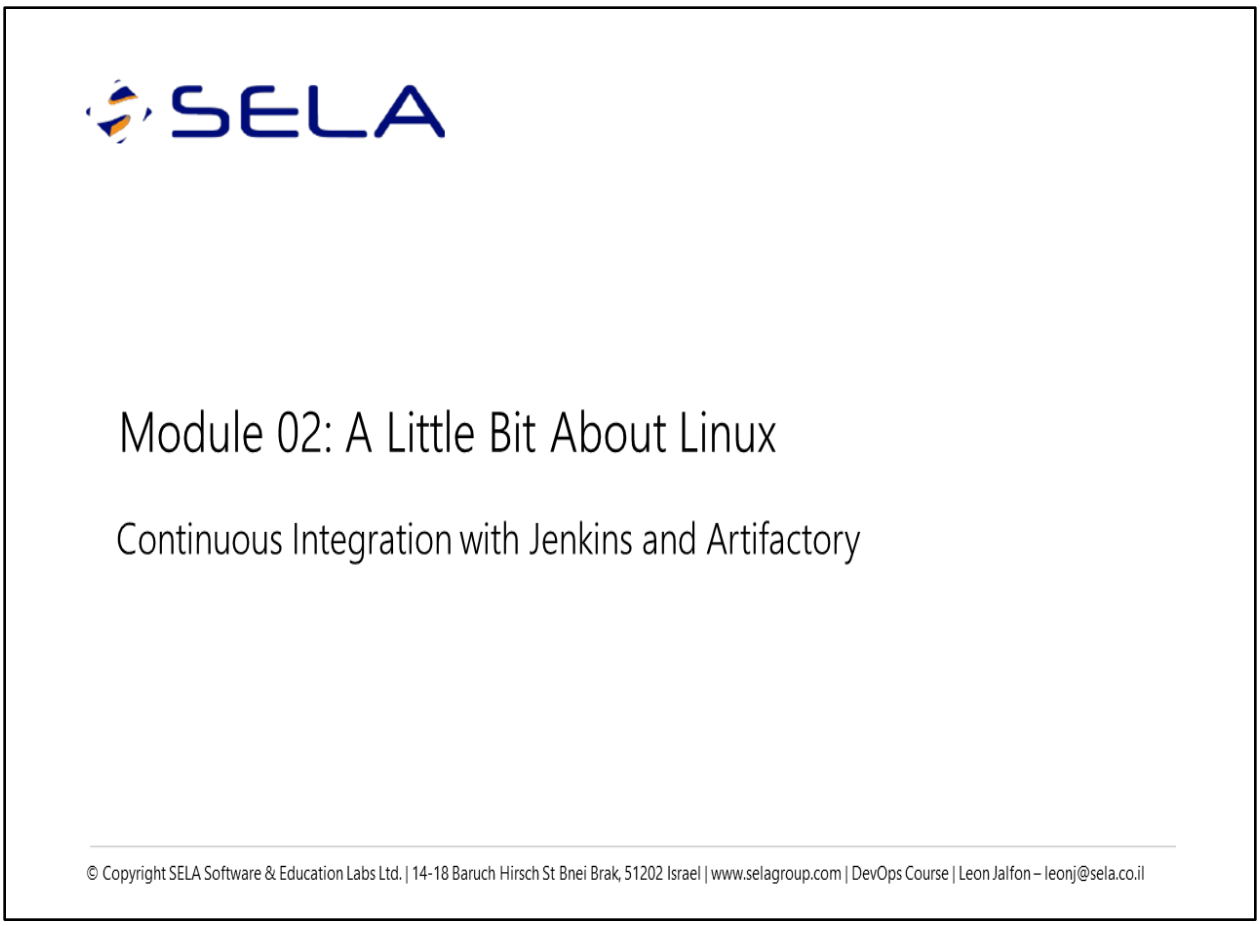
History of Jenkins

- 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:

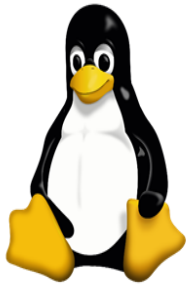

SELA
Module 02: A Little Bit About Linux
Continuous Integration with Jenkins and Artifactory
© Copyright SELA Software & Education Labs Ltd. | 14-18 Baruch Hirsch St Bnei Brak, 51202 Israel | www.selagroup.com | DevOps Course | Leon Jalfon – leonj@sela.co.il

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.



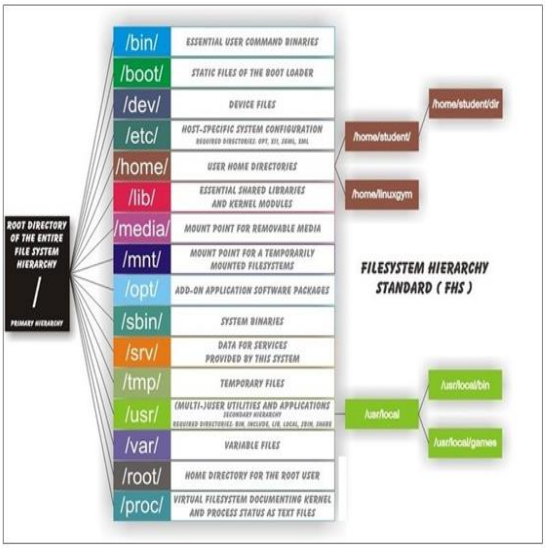
Linux Distributions

- 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

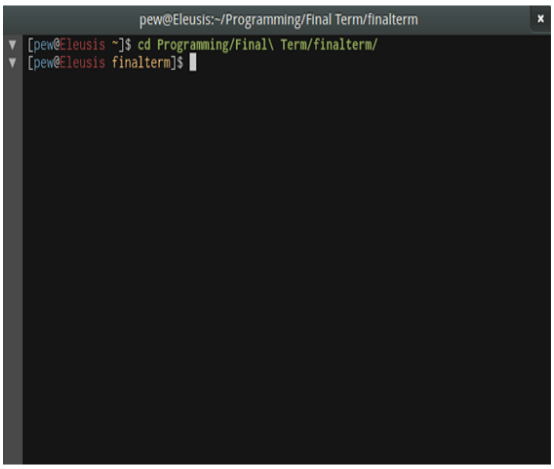
Filesystem Hierarchy Standard

- 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

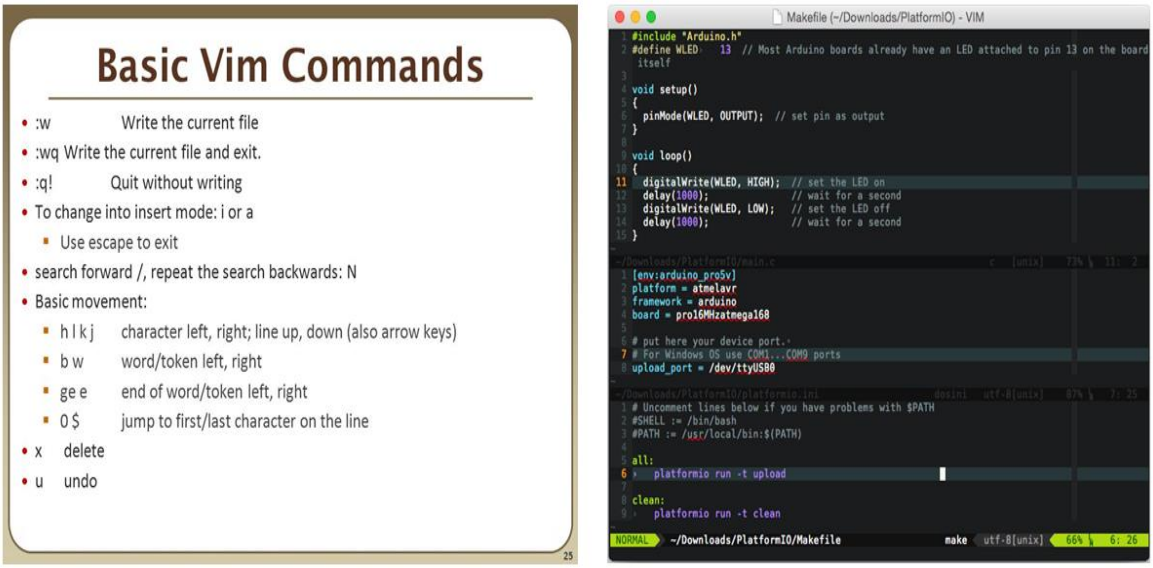
- ls - List information about file(s)
- cd - Change Directory
- mv - Move or rename files or directories
- man - Help manual
- mkdir - Create new folder(s)
- rmdir - Remove folder(s)
- touch - Change file timestamps
- rm - Remove files
- locate - Find files
- clear - terminal screen

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



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!

ubuntu®

A look to Ubuntu Distribution - Features

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

Questions

Lab: Lab 1: Working with Ubuntu

Module 03 - Jenkins Overview

Contents:

Module 03: Jenkins Overview

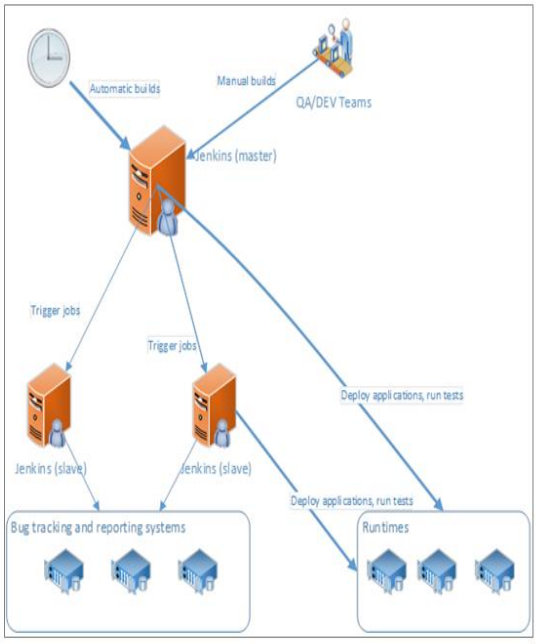
Continuous Integration with Jenkins and Artifactory

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.

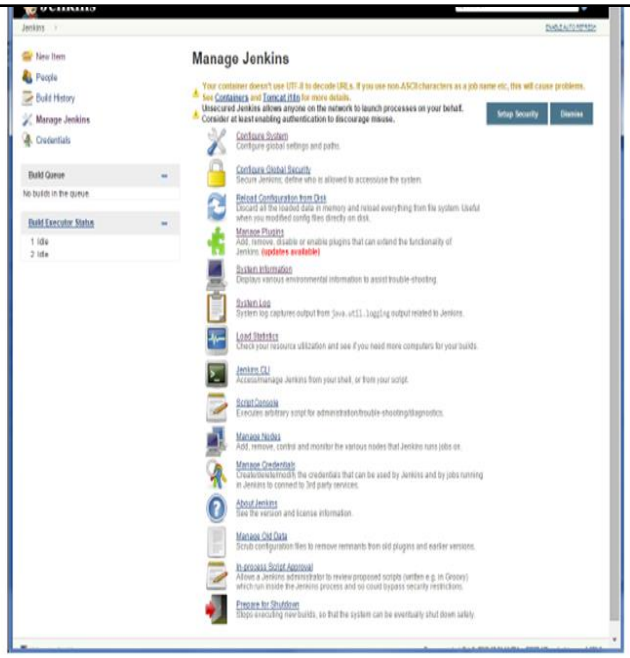
For most platforms you have native packages.

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.

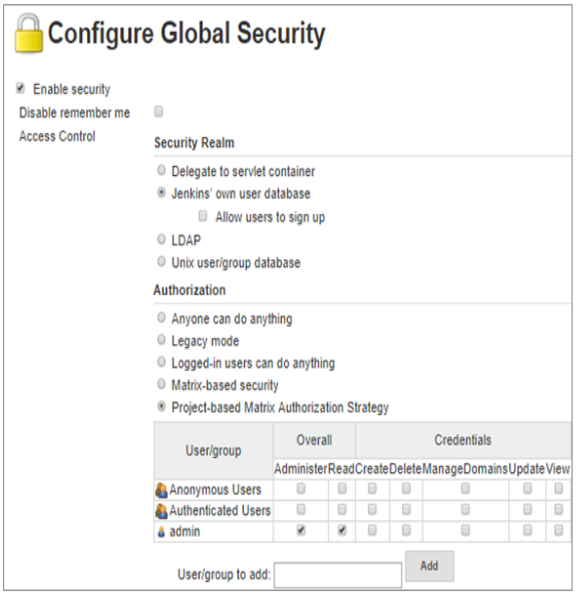
Configuration

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



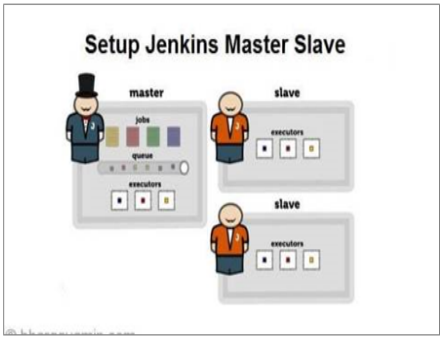
Secure Jenkins

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

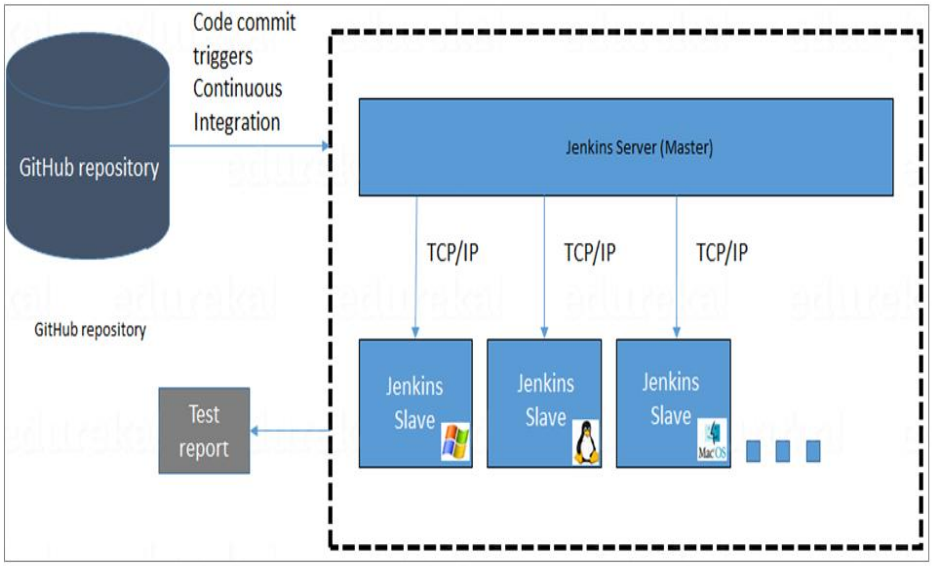


Jenkins Slaves

- 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"

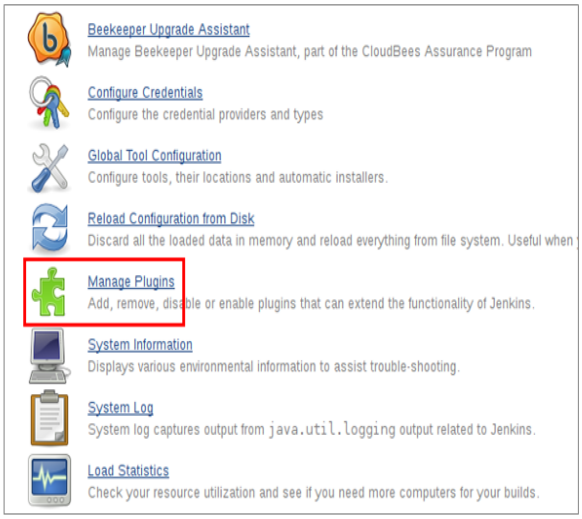


Jenkins Slaves



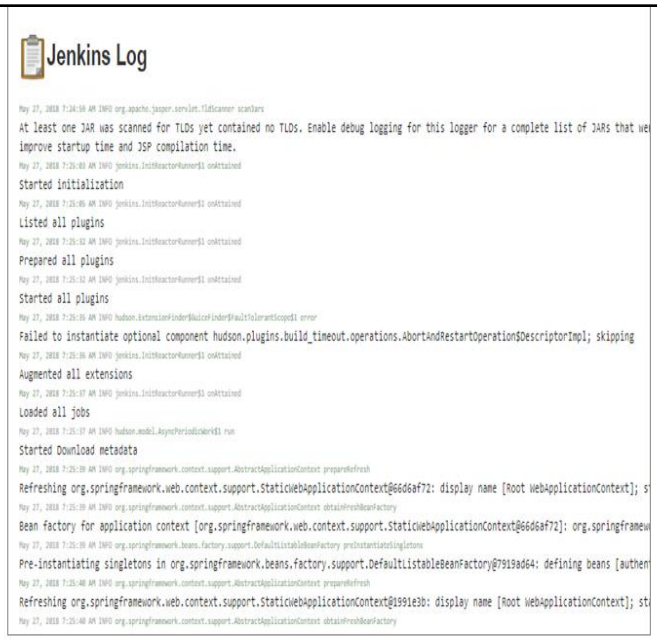
Jenkins Plugins

- 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



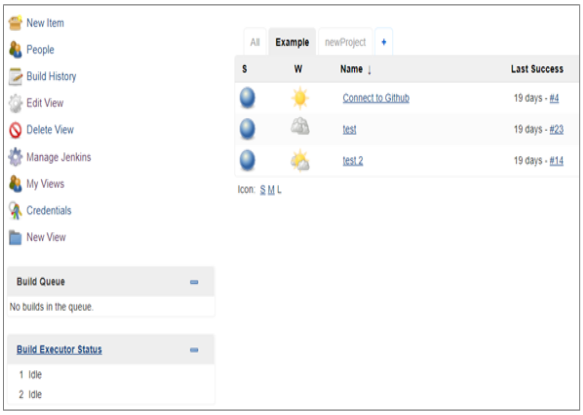
System Logs

- 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.



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.

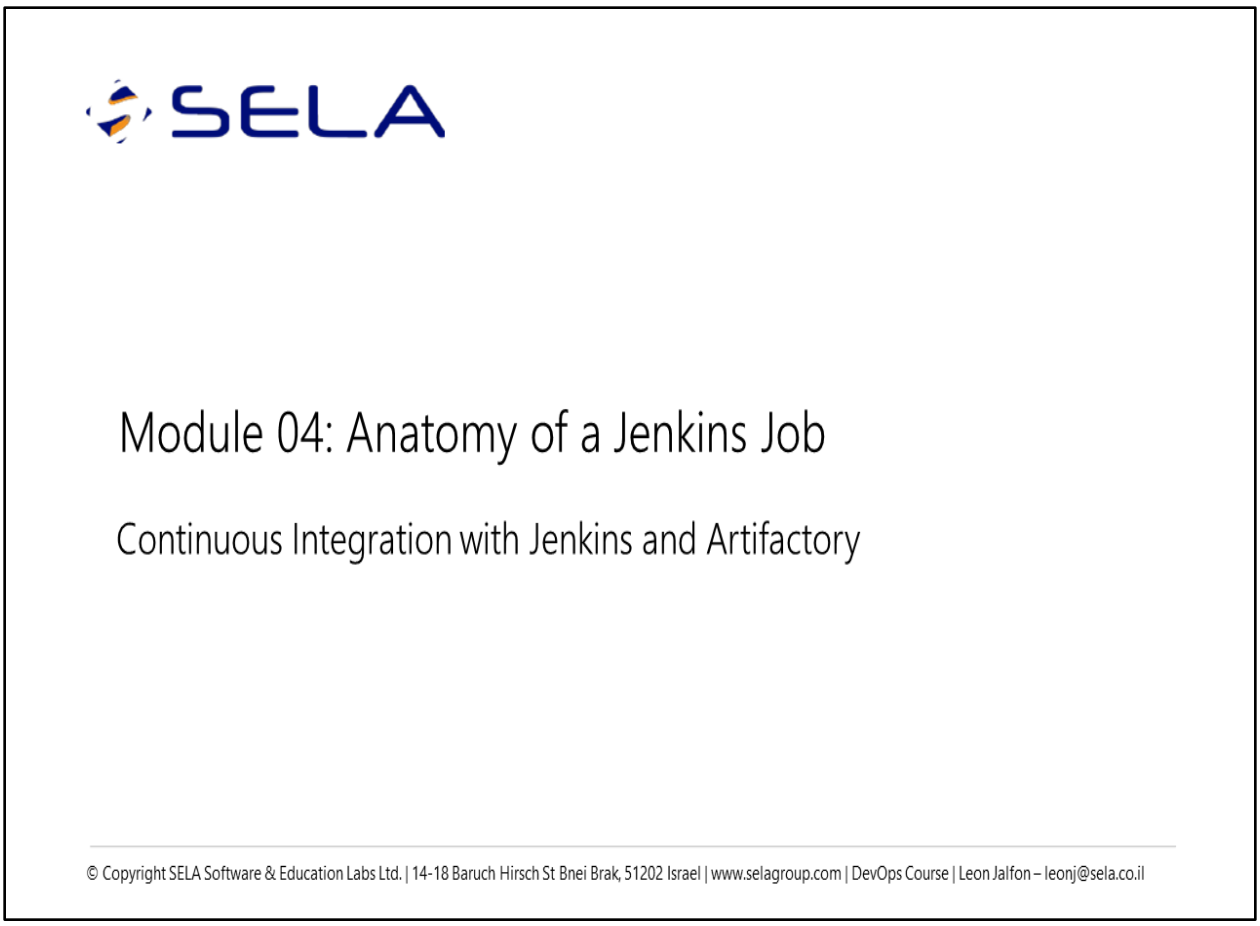


Questions

Jenkins Overview

Module 04 - Anatomy of a Jenkins Job

Contents:

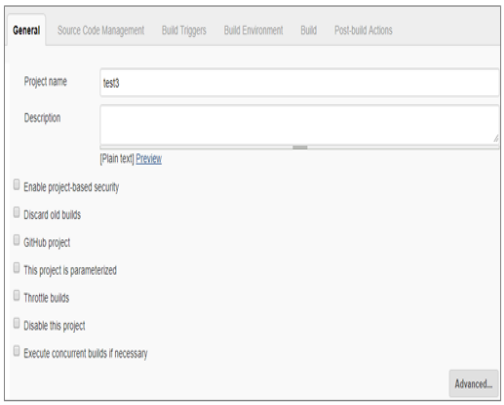

SELA
Module 04: Anatomy of a Jenkins Job
Continuous Integration with Jenkins and Artifactory
© Copyright SELA Software & Education Labs Ltd. | 14-18 Baruch Hirsch St Bnei Brak, 51202 Israel | www.selagroup.com | DevOps Course | Leon Jalfon – leonj@sela.co.il

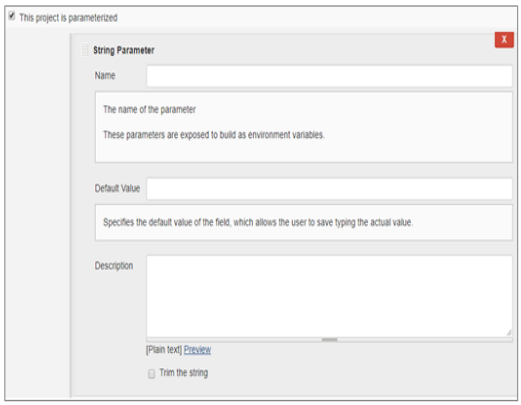
Agenda

- 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

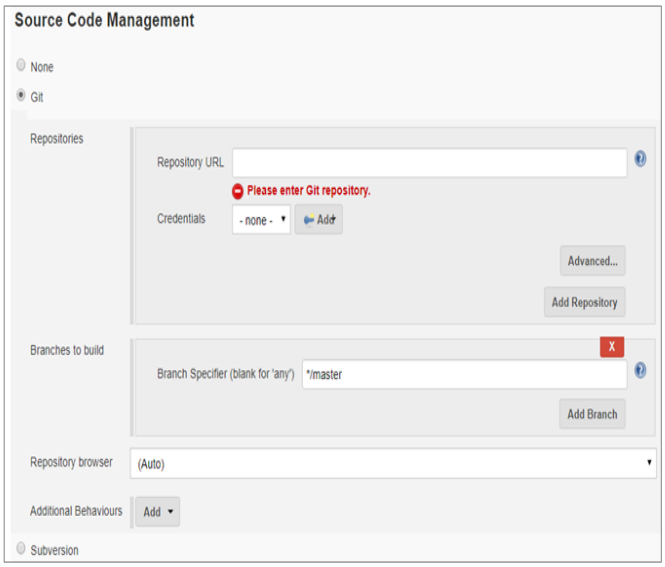
- Contains general information about the project, such as a unique name and description, and other information about how and where the build job should be executed.





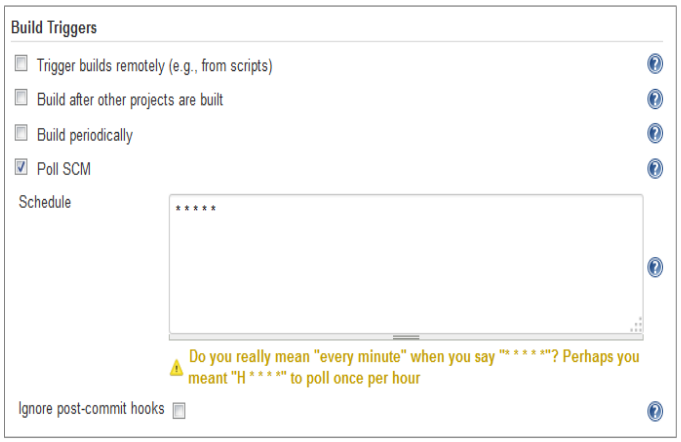
Source Code Management

- 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.



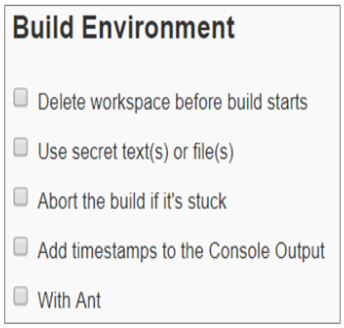
Build Triggers

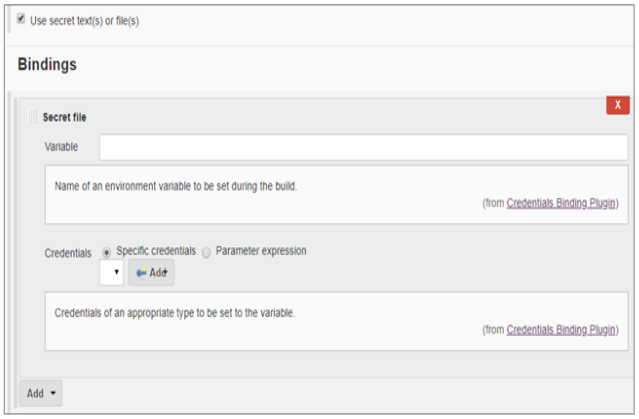
- Once you have configured your version control system, you need to tell Jenkins when to kick off a build. You set this up in the Build Triggers section.



Build Environment

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



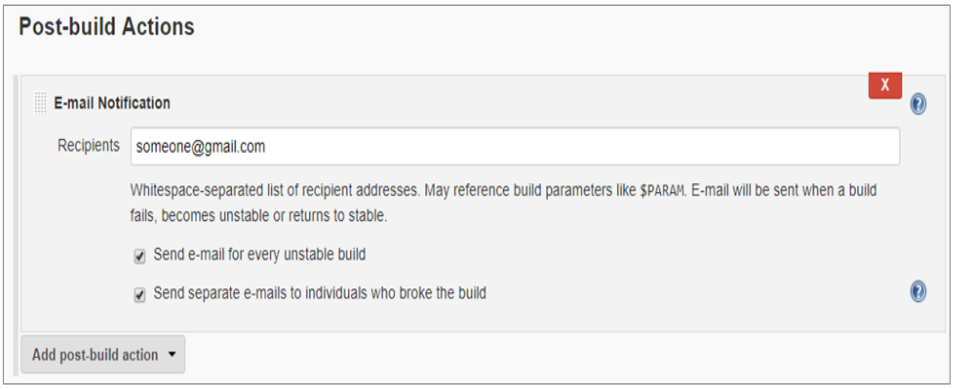


Build Steps

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

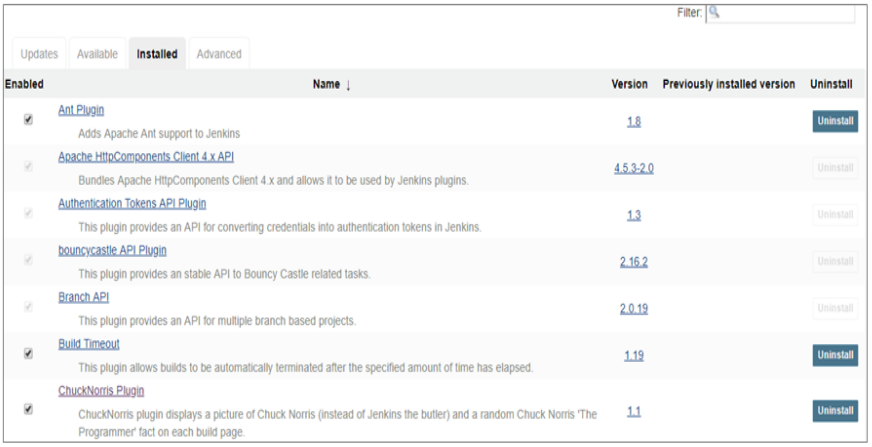
Post-Build steps

- 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.



Using plugins

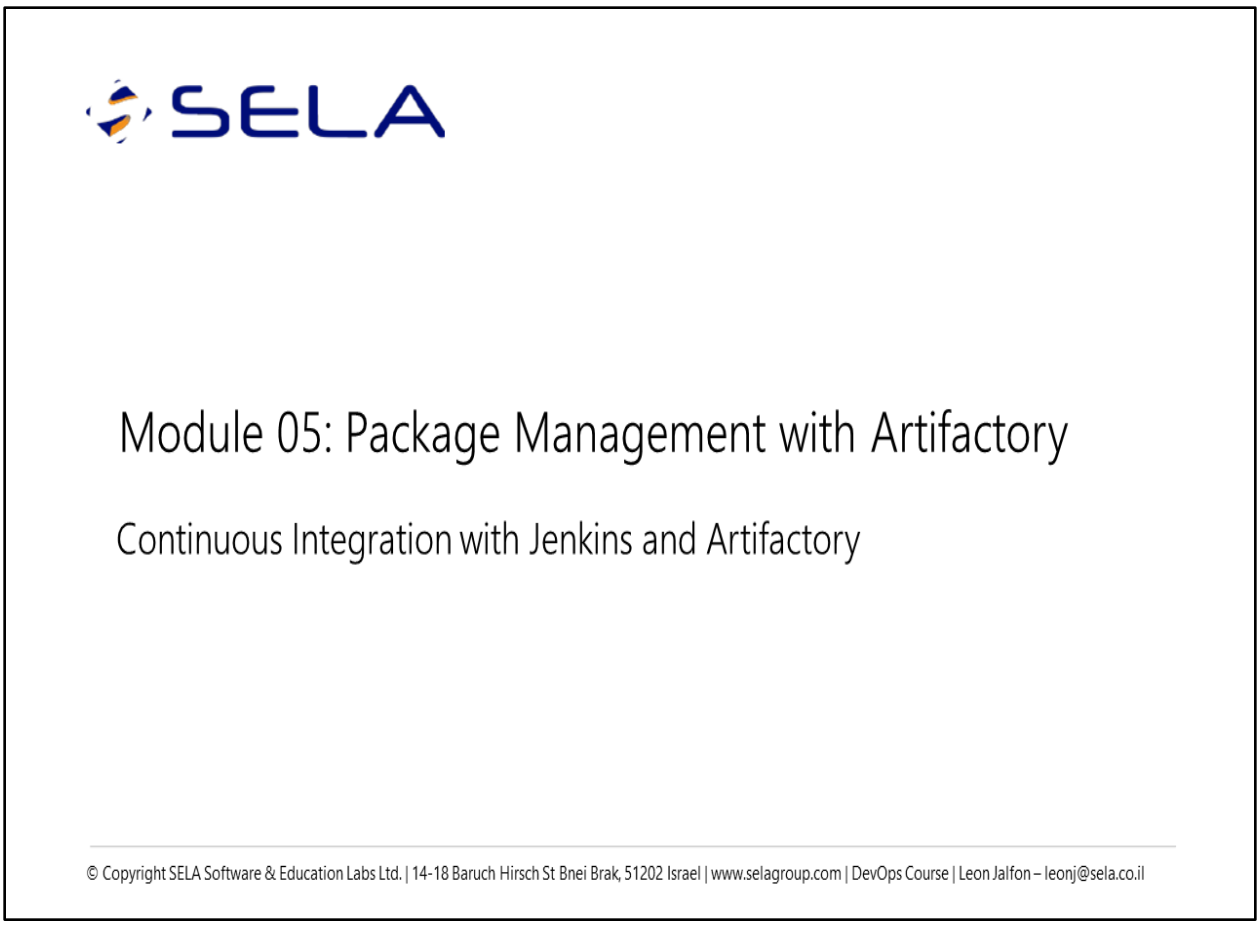
- Plugins make Jenkins what it is, very adaptable and customizable.
- To enable a plugin first make sure it is selected in the plug in manager.



Lab: Lab 02: Creating the first freestyle Jenkins job

Module 05 - Package Management with Artifactory

Contents:


SELA
Module 05: Package Management with Artifactory
Continuous Integration with Jenkins and Artifactory
© Copyright SELA Software & Education Labs Ltd. | 14-18 Baruch Hirsch St Bnei Brak, 51202 Israel | www.selagroup.com | DevOps Course | Leon Jalfon – leonj@sela.co.il

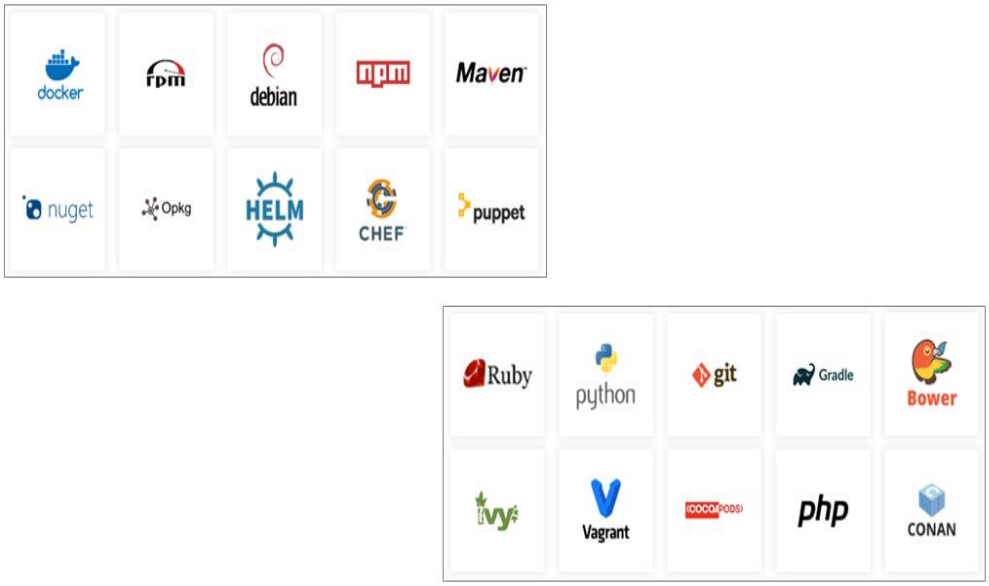
Agenda

- 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

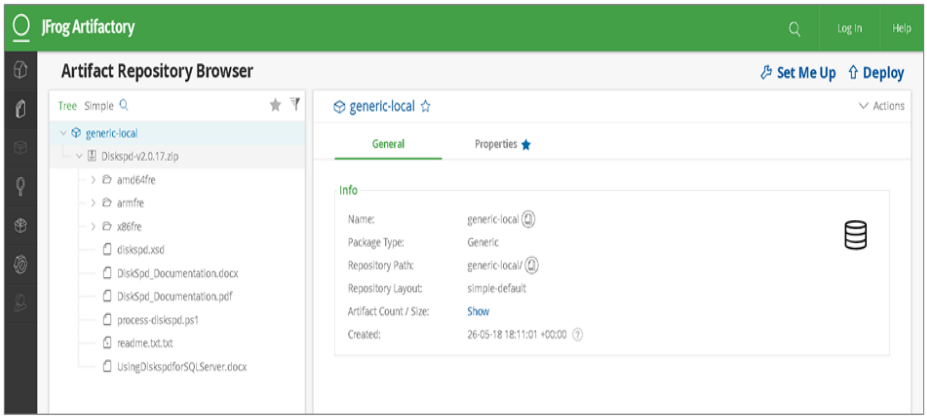
- Artifactory OSS
- Artifactory Pro
- Artifactory SaaS
- Artifactory SaaS (dedicated server)
- Artifactory Enterprise

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

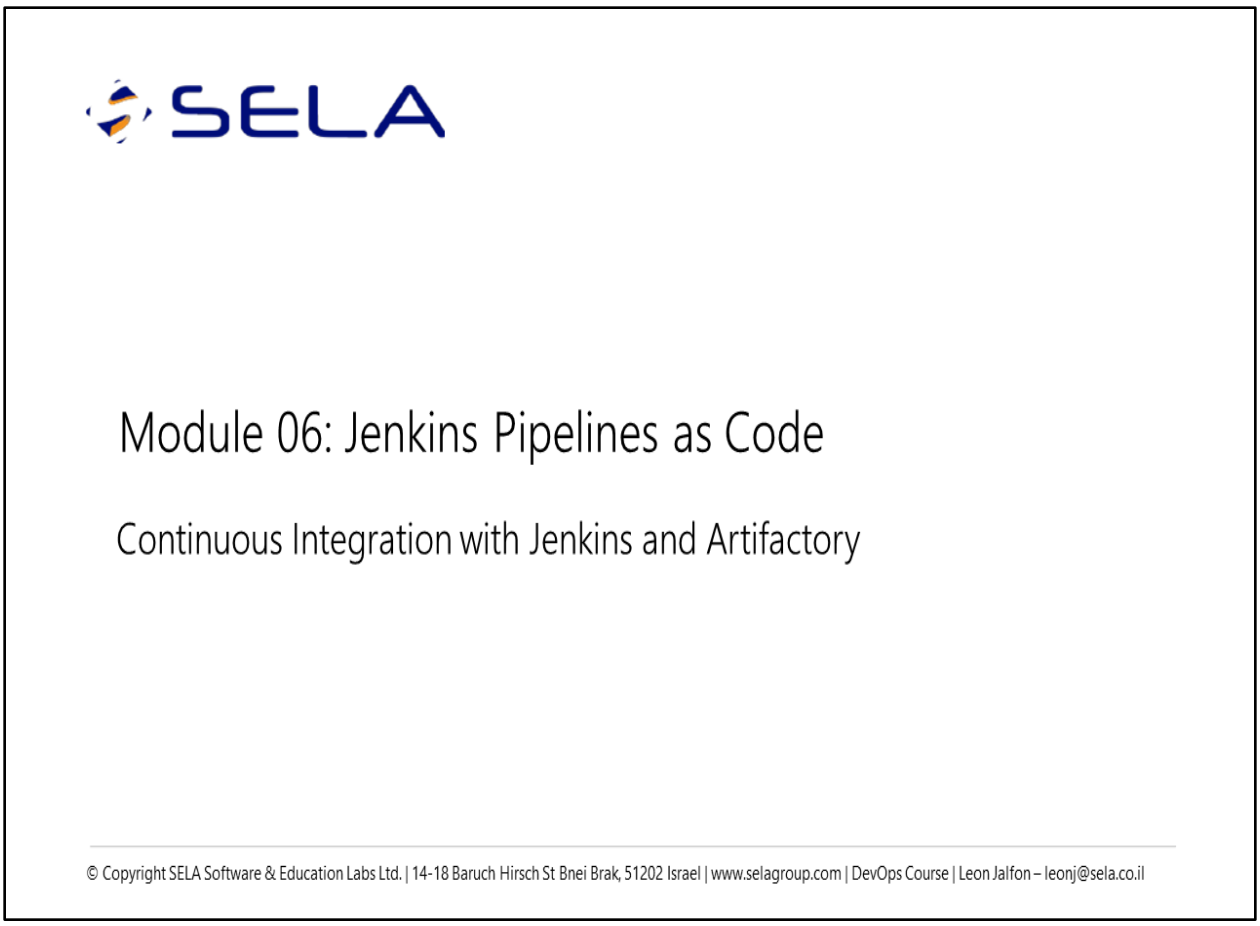
Generic Repositories

Questions

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

Module 06 - Jenkins pipelines as code

Contents:

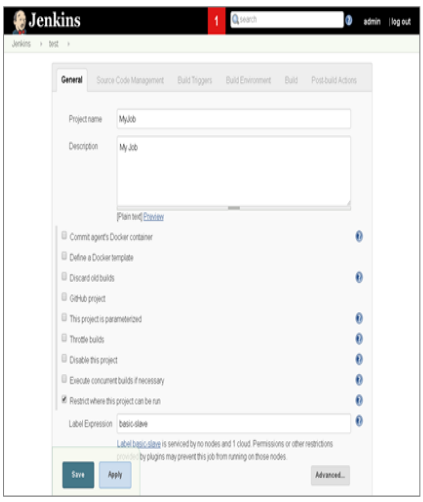

SELA
Module 06: Jenkins Pipelines as Code
Continuous Integration with Jenkins and Artifactory
© Copyright SELA Software & Education Labs Ltd. | 14-18 Baruch Hirsch St Bnei Brak, 51202 Israel | www.selagroup.com | DevOps Course | Leon Jalfon – leonj@sela.co.il

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

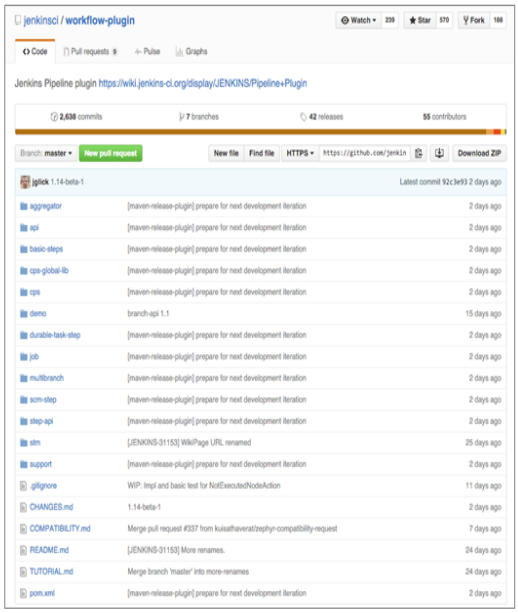
Pipeline as Code

- 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

```
stage('Integration Testing') {
    steps {
        echo 'Running Integration Tests..'
        sh 'cypress run --record --key xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx -c baseUrl=https://<url.com>/'

        echo 'Running Visual Regression tests..'
        sh 'npm run image-compare-ci'
    }
}
```

- 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.

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                sh 'npm --version'
            }
        }
    }
}
```

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.

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                sh 'echo "Hello World"'
                sh '''
                    echo "Multiline shell steps"
                    ls -lah
                '''
            }
        }
    }
}
```

Declarative Pipelines

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

```
pipeline {
    agent any
        stages {
            stage('Build') {
                when {
                    expression {
                        "foo" == "bar"
                    }
                }
                steps {
                    echo 'Building'
                }
            }
        stage('Test') {
            when {
                environment name: 'JOB_NAME', value: 'foo'
            }
            steps {
                echo 'Testing'
            }
        }
        stage('Deploy') {
            when {
                branch 'master'
            }
            steps {
                echo 'Deploying'
            }
        }
    }
}
```

Declarative Pipelines

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

```
pipeline {
    agent any
    stages {
        stage('Browser Tests') {
            parallel {
                stage('Chrome') {
                    steps {
                        echo "Chrome Tests"
                    }
                }
                stage('Firefox') {
                    steps {
                        echo "Firefox Tests"
                    }
                }
            }
        }
    }
}
```

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

```
pipeline {
    agent {
        node { label 'my-agent' }
    }
}
```

```
pipeline {
    agent {
        node {
            label 'my-defined-label'
            customWorkspace '/some/other/path'
        }
    }
}
```

Declarative Pipelines

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

```
pipeline {
    agent {
        node { label 'my-docker' }
    }
    stages {
        stage("Build") {
            agent {
                docker {
                reuseNode true
                image 'maven:3.5.0-jdk-8'
                }
            }
            steps {
                sh 'mvn install'
            }
        }
    }
}
```

Declarative Pipelines

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

```
pipeline {
    agent any
    environment {
        DISABLE_AUTH = 'true'
        DB_ENGINE    = 'sqlite'
    }
    stages {
        stage('Build') {
            steps {
                sh 'printenv'
            }
        }
    }
}
```

```
environment {
    AWS_ACCESS = credentials('AWS_ KEY')
    AWS_SECRET = credentials('AWS_SECRET')
}
```

Declarative Pipelines

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

```
pipeline {
    agent any
    stages {
        stage('No-op') {
            steps {
                sh 'ls'
            }
        }
    }
    post {
        always {
            echo 'I have finished'
            deleteDir() // clean up workspace
        }
        success {
            echo 'I succeeded!'
        }
        unstable {
            echo 'I am unstable :/'
        }
        failure {
            echo 'I failed :('
        }
        changed {
            echo 'Things are different...'
        }
    }
}
```

Declarative Pipelines

- Email/Slack Notification

```
post {
    failure {
        mail to: 'team@example.com',
            subject: 'Failed Pipeline',
            body: "Something is wrong"
    }
}
```

```
post {
    success {
        slackSend channel:'#ops-room',
            color: 'good',
            message: 'Completed successfully.'
    }
}
```

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

```
stage('Sanity check') {
  steps {
      input "Does the staging environment look ok?"
  }
}
```

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)
}
```

Lab: Lab 4: CI jobs with Jenkins pipelines

Module 07 - Jenkins with Blue Ocean

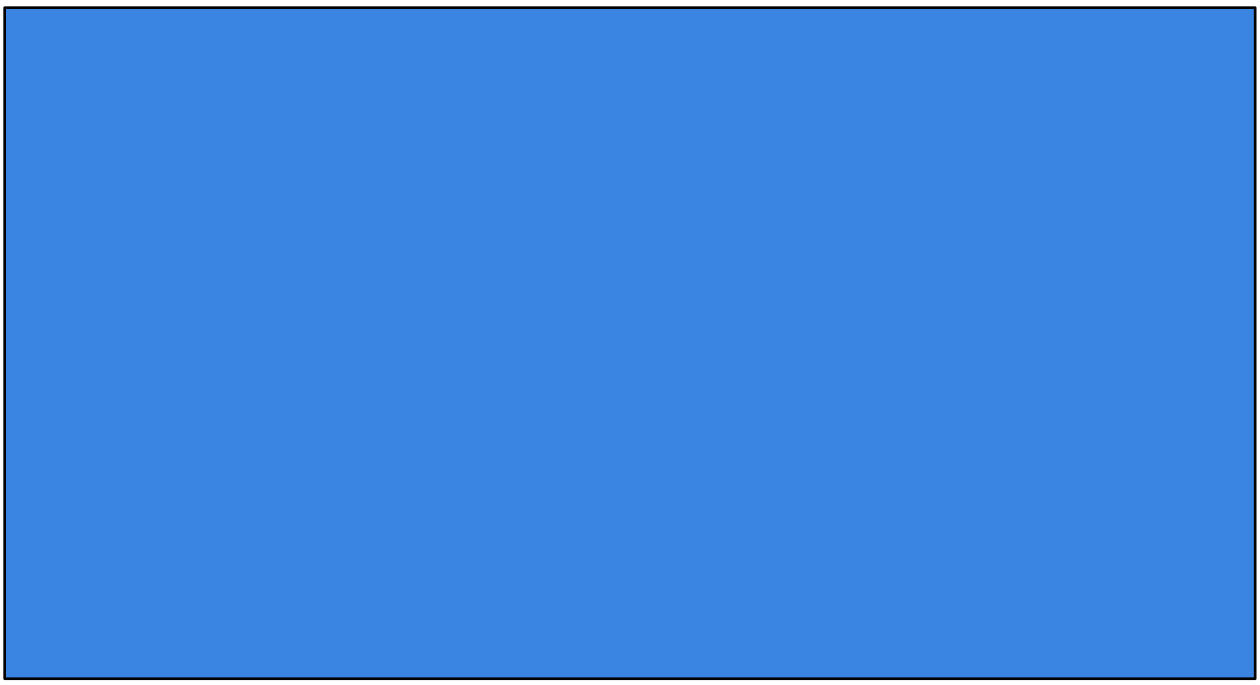
Contents:

Module 07: Jenkins with Blue Ocean

Continuous Integration with Jenkins and Artifactory

Agenda

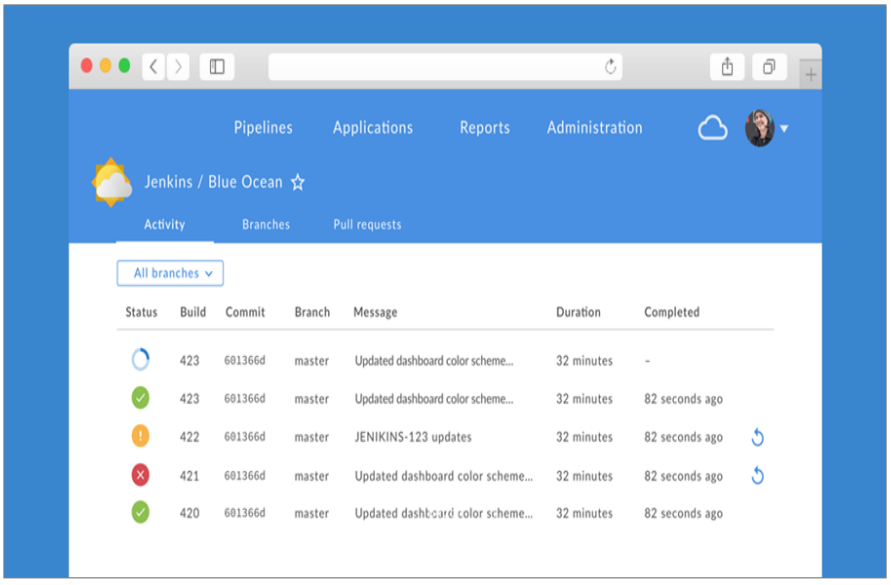
- What is Jenkins Blue Ocean?
- Main Features
- Installation
- Overview
- Lab 5: Creating a Jenkins job using Blue Ocean



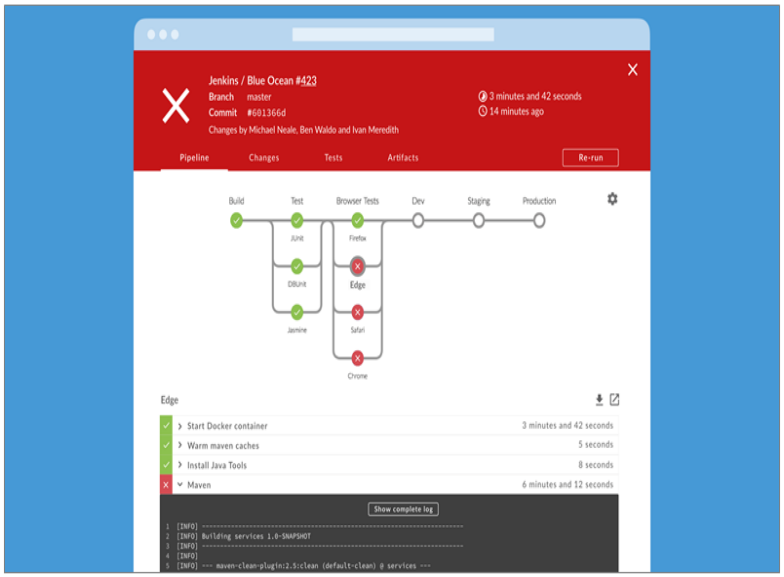
What is Jenkins Blue Ocean

- 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

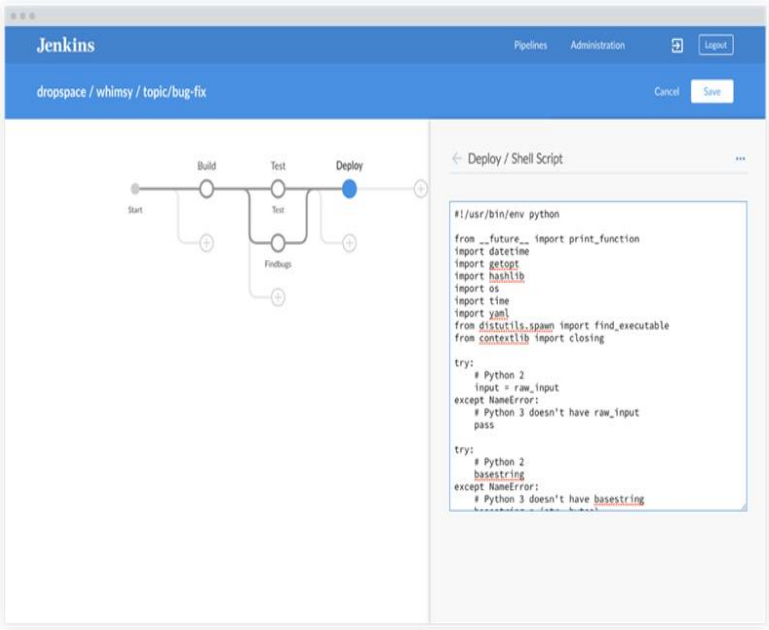
Main Features



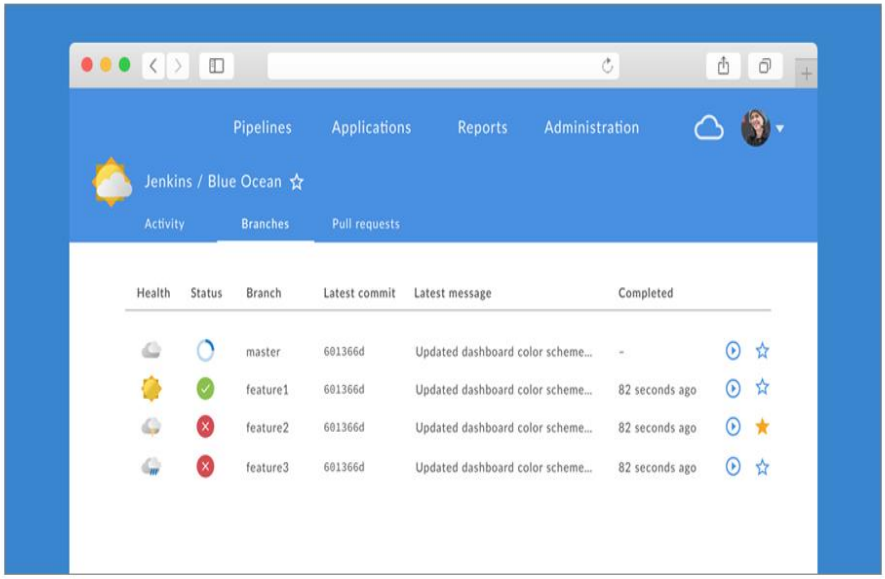
Main Features



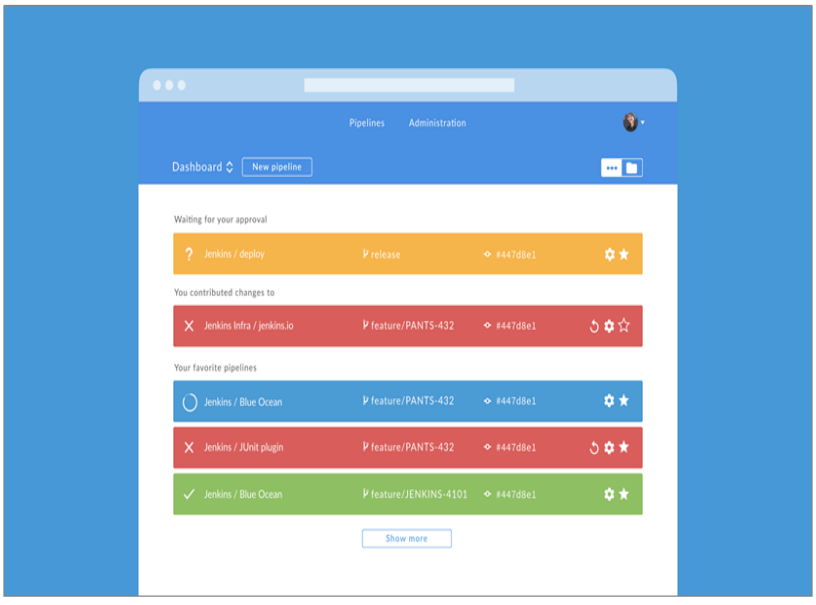
Main Features



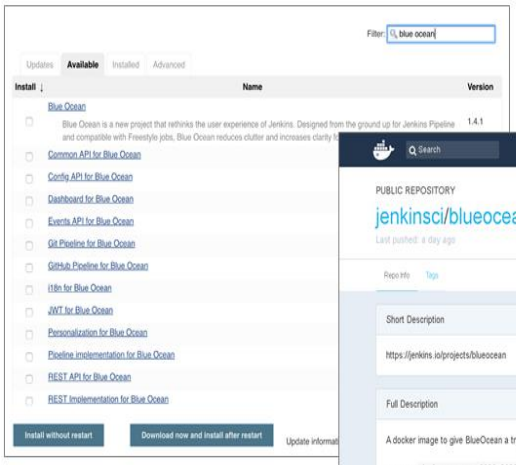
Main Features



Main Features

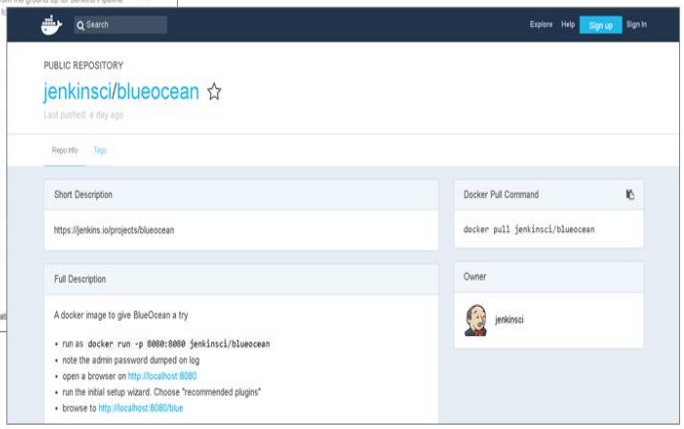


Blue Ocean installation



(Plugin)

(New instance as docker container)



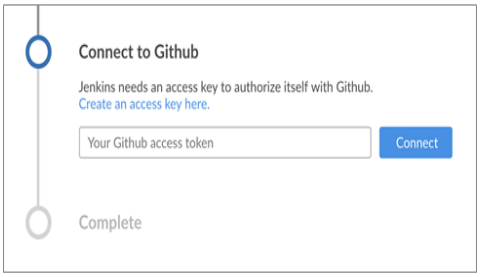
Blue Ocean Overview

- Creating a pipeline

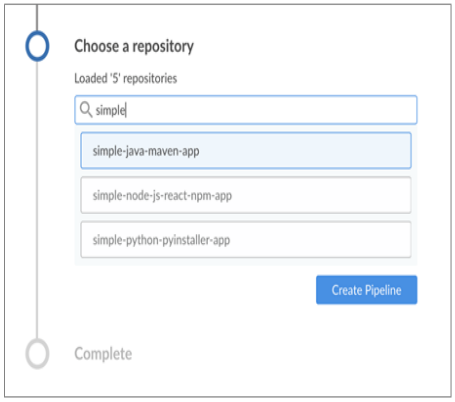
- Getting Sources

Blue Ocean Overview

- Connecting to the repository

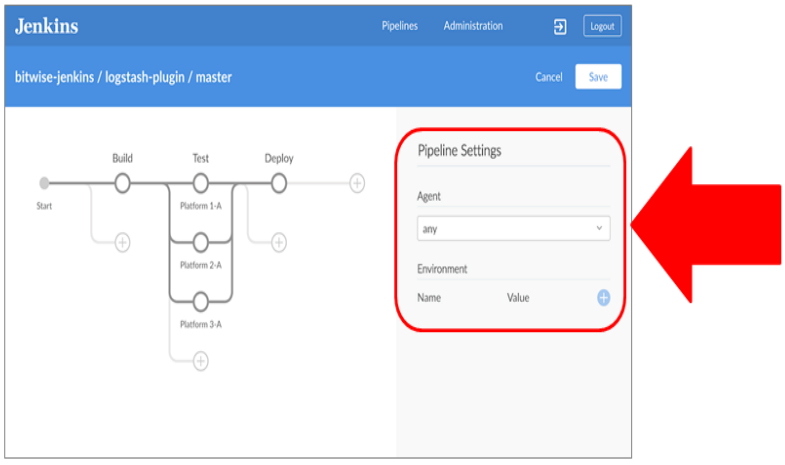


- Choosing the repository

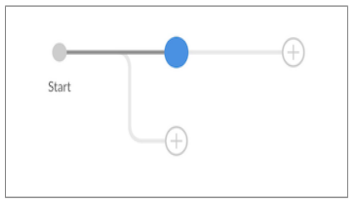


Blue Ocean Overview

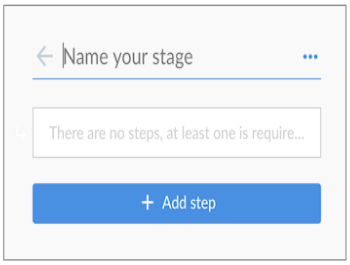
- Pipeline settings



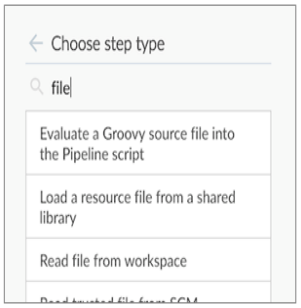
Blue Ocean Overview



Stage editor



Stage configuration

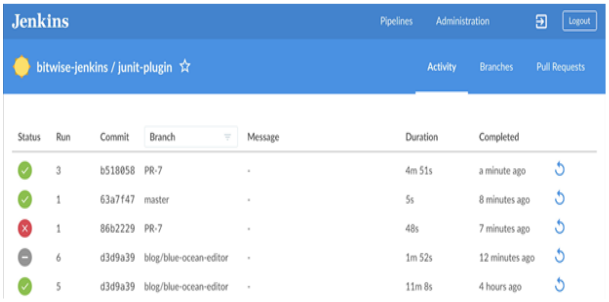


Step configuration

Blue Ocean Overview

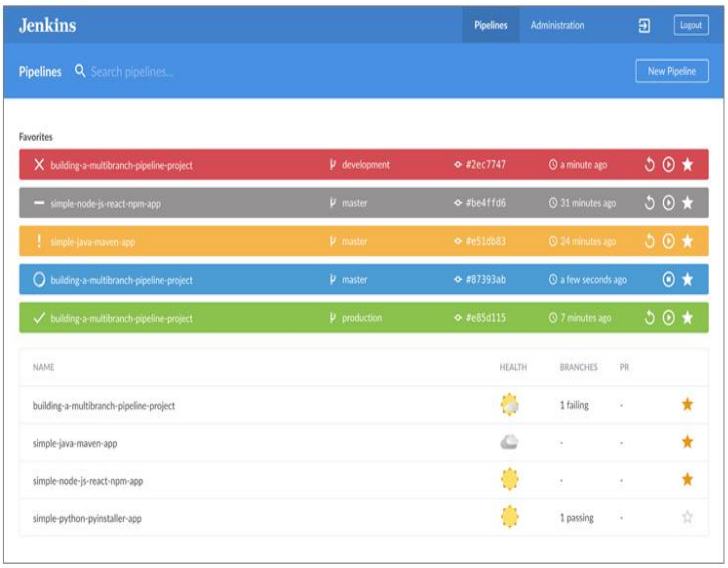
Save pipeline

Activity View



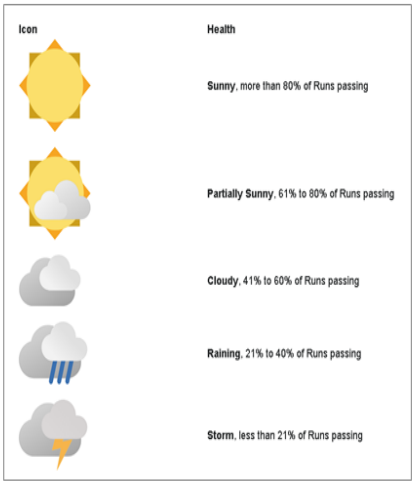
Blue Ocean Overview

- Dashboard



Blue Ocean Overview

- Dashboard - Health Icons

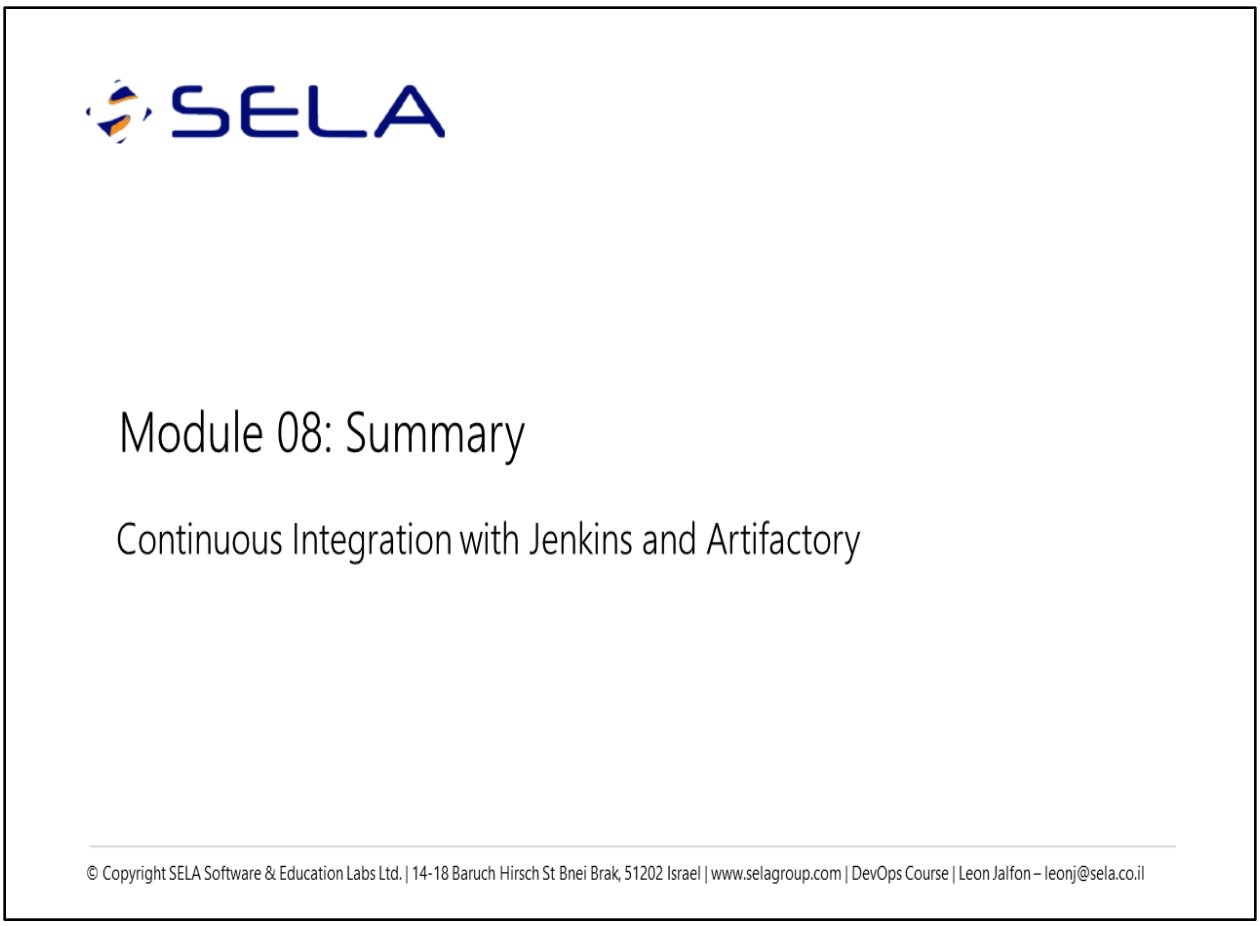


- Dashboard - Run Status

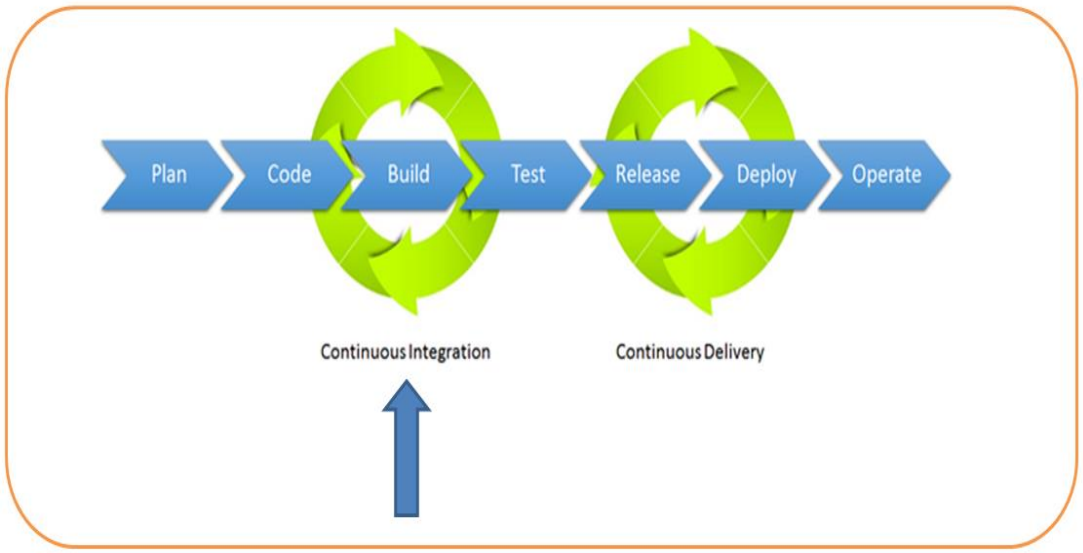
Lab: Lab 5: Creating a Jenkins job using Blue Ocean

Module 08 - Summary

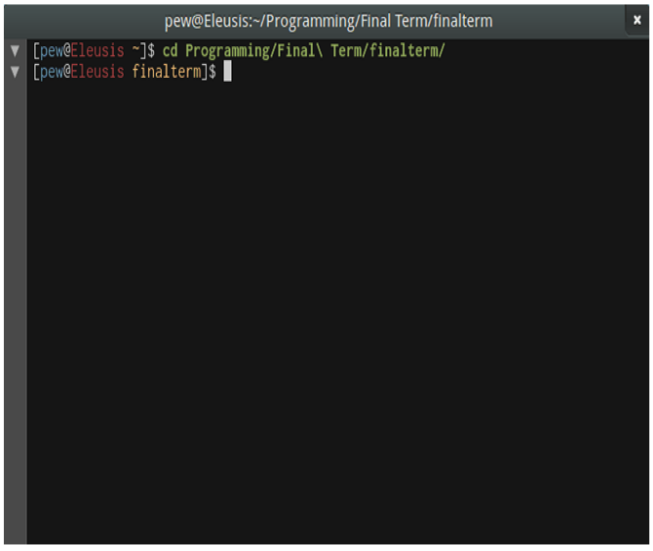
Contents:


SELA
Module 08: Summary
Continuous Integration with Jenkins and Artifactory
© Copyright SELA Software & Education Labs Ltd. | 14-18 Baruch Hirsch St Bnei Brak, 51202 Israel | www.selagroup.com | DevOps Course | Leon Jalfon – leonj@sela.co.il

DevOps Pipelines

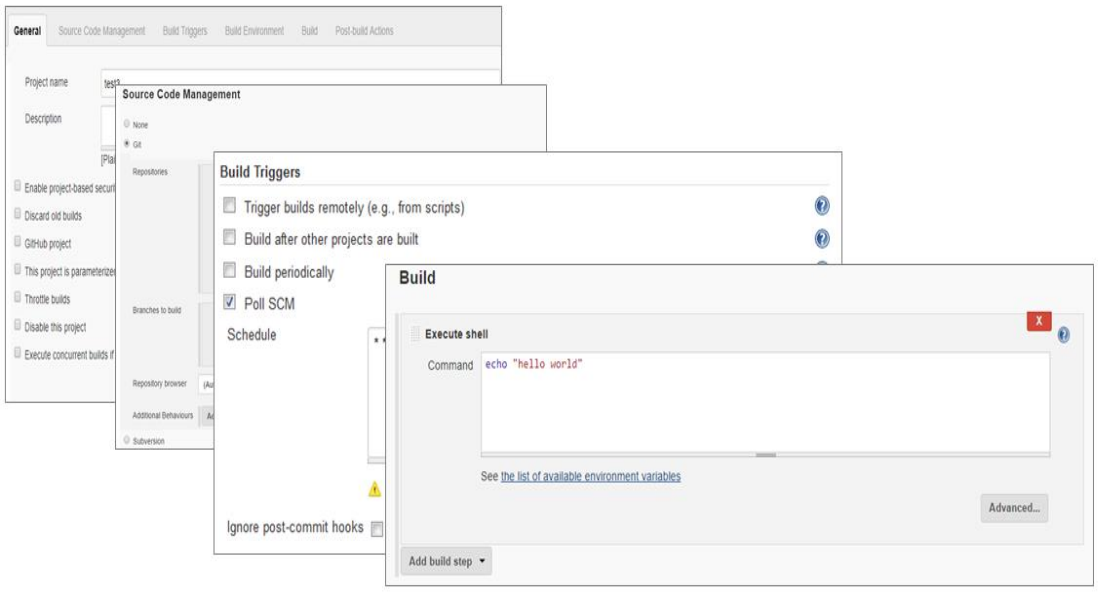


Linux Overview

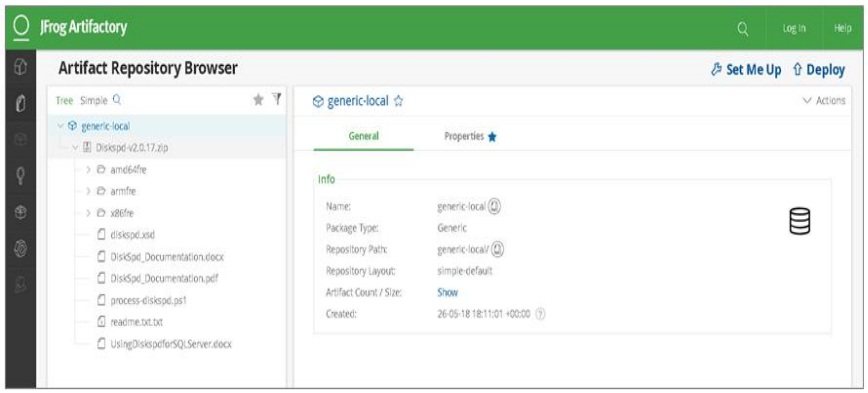


Jenkins Overview

Jenkins Freestyle Jobs



Package Management with Artifactory



Jenkins Pipelines as Code

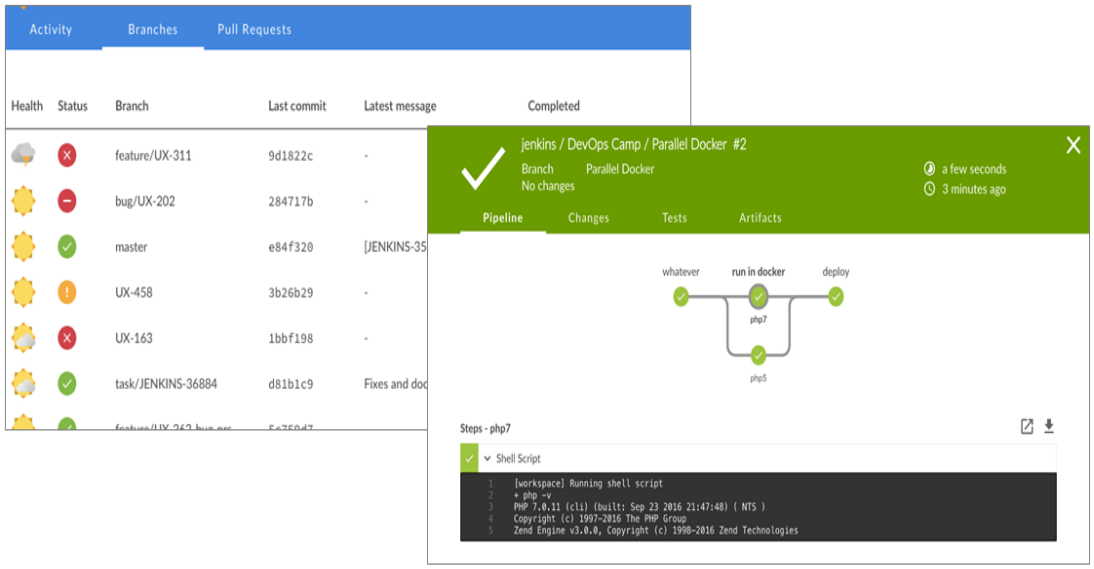
```
pipeline {
    agent any
    stages {
        stage('Browser Tests') {
            parallel {
                stage('Chrome') {
                    steps {
                        echo "Chrome Tests"
                    }
                }
                stage('Firefox') {
                    steps {
                        echo "Firefox Tests"
                    }
                }
            }
        }
    }
}
```

```
node {

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

}
```

Jenkins Blue Ocean



Itshak Eli


SELA
Thank you for your attention!
Continuous Integration with Jenkins and Artifactory
Itshak Eli
itshak.eli@gmail.com
© Copyright SELA Software & Education Labs Ltd. | 14-18 Baruch Hirsch St Bnei Brak, 51202 Israel | www.selagroup.com | Leon Jalfon - Blog: http://blogs.microsoft.co.il/leonj