JENKINS 2 & CONTINUOUS INTEGRATION

JOHN BRYCE
Leading in IT Education
a matrix company

Jenkins Introduction

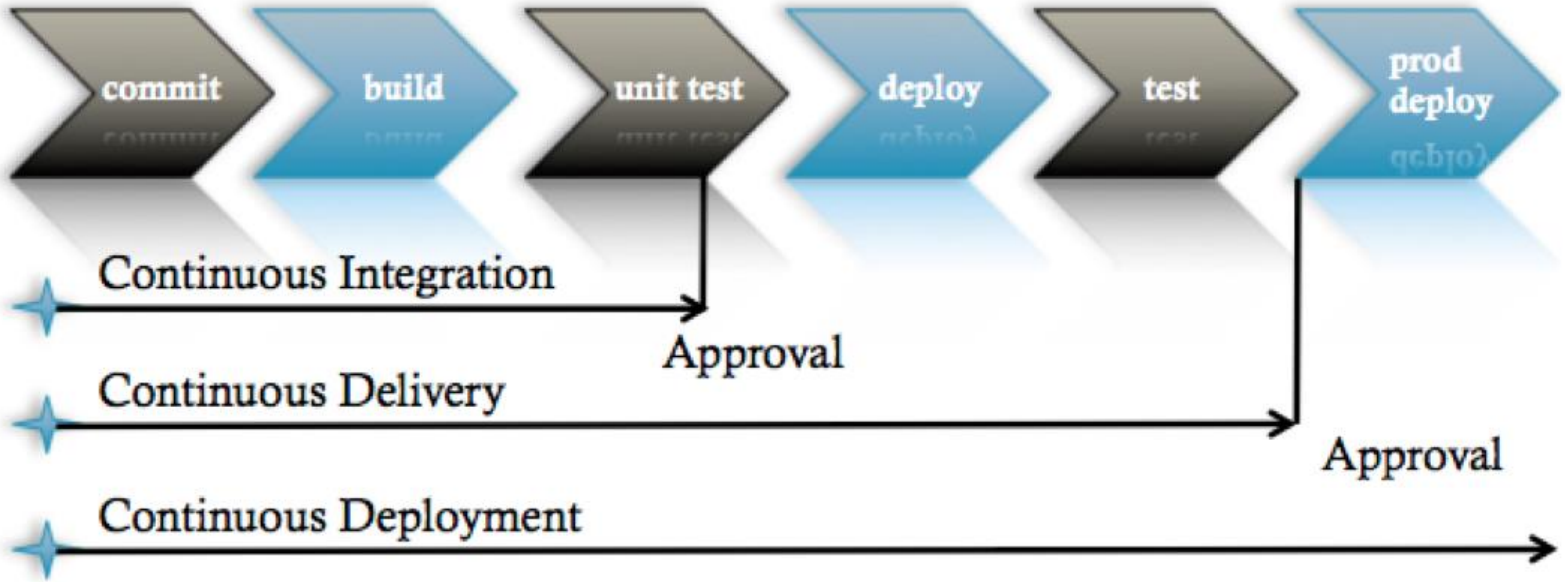
JOHN BRYCE
Leading in IT Education
a matrix company

What is Jenkins?

Jenkins is a self-contained, open source automation service which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

The tool offers a simple way to set up a **continuous integration** or **continuous delivery** environment for almost any combination of languages and source code repositories using pipelines (scripts), as well as automating other routine **development** tasks.

Jenkins Introduction

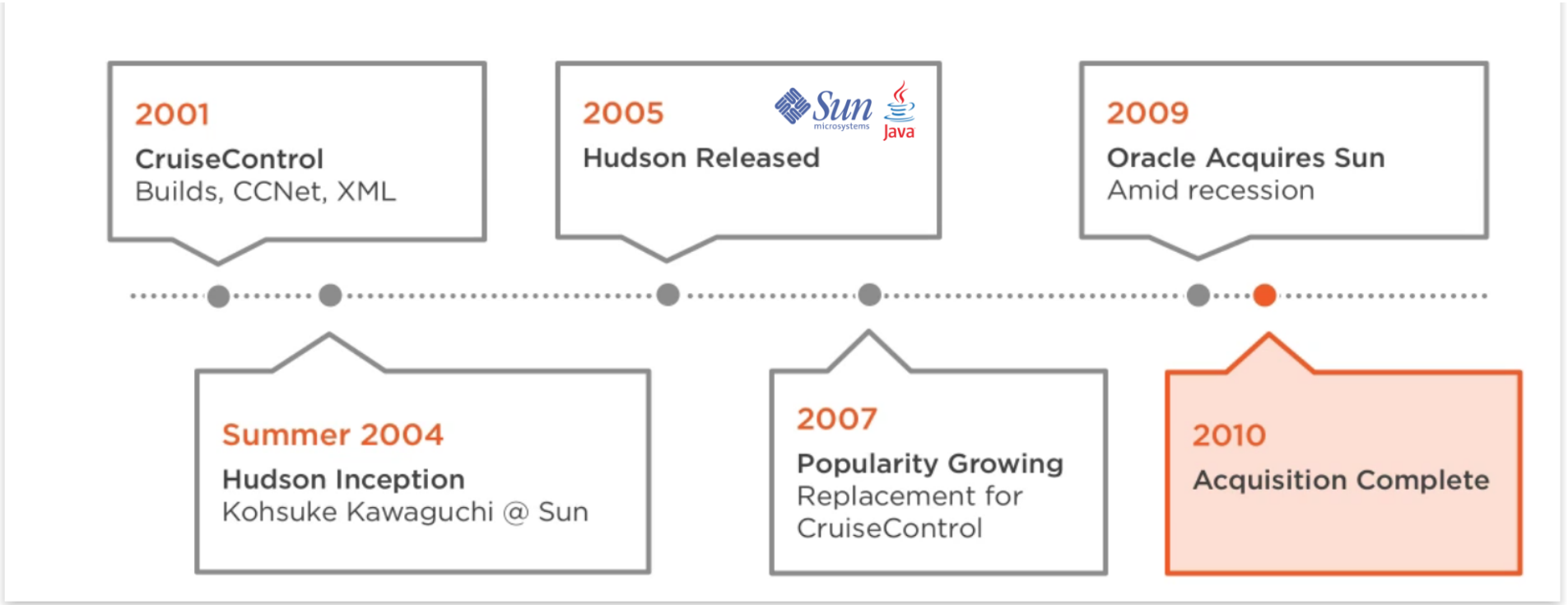


Jenkins Introduction

- Branched from Hudson
- Java based Continuous Build System
- Runs in servlet container Tomcat
- Supported by over **1600+** plugins and the **1000+** community contributed Jenkins (https://plugins.jenkins.io/)
- SCM,Testing, Notifications, Reporting, Artifact Saving, Triggers,
- External Integration
- Under development since 2005

History of Jenkins

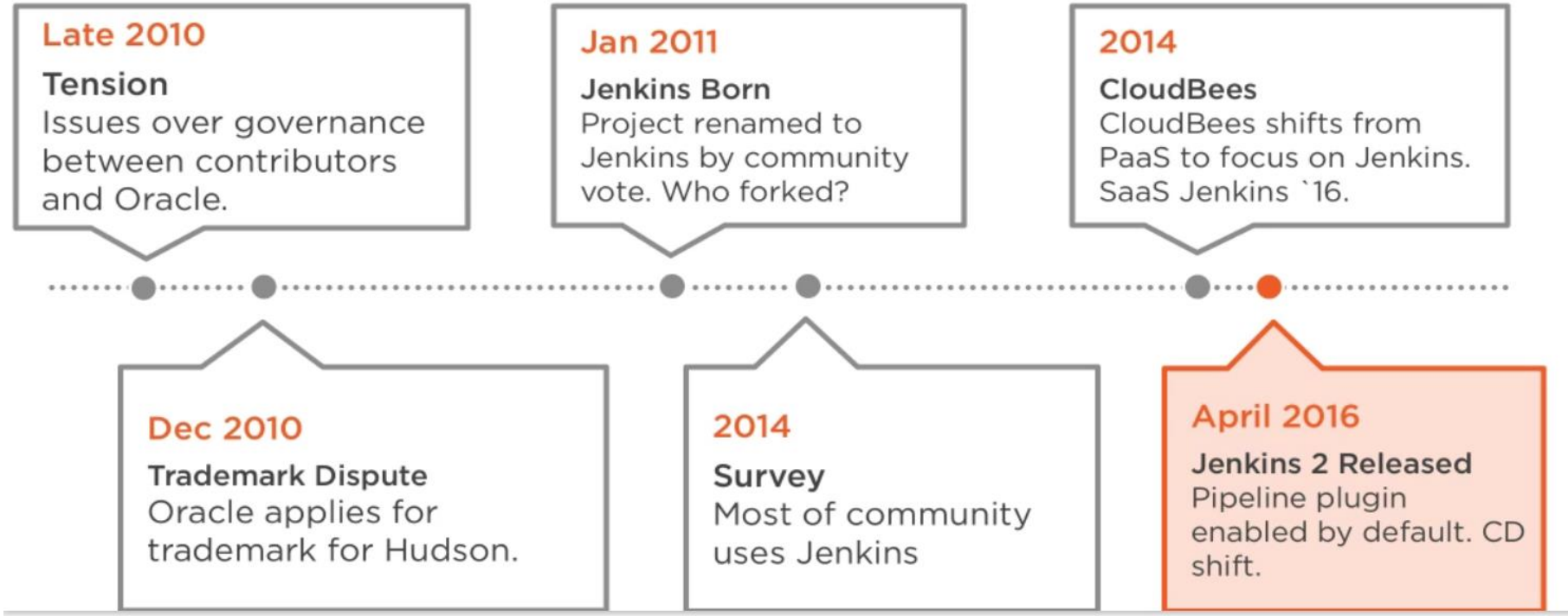
In 2004, **Kohsuke Kawaguchi** (Sun corporation) built an automation server in and for Java called **Hudson**. Fast-forward to 2011, and a dispute between Oracle (which had acquired Sun) and the independent Hudson open source community led to a fork with a name change, **Jenkins**.



History of Jenkins

JOHN BRYCE
Leading in IT Education
a matrix company

Both forks Jenkins and Hudson continue to exist, although Jenkins is much more active. In 2014 Kawaguchi became CTO of CloudBees, which offers Jenkins-based continuous delivery products.



Jenkins Installation

Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed:

- Master direct Installation
- JAR/WAR BASE
- Docker BASE

Jenkins Installation: Tomcat-Servlet

JOHN BRYCE
Leading in IT Education
a matrix company

- Jenkins is packaged as a WAR, so you can drop it into whichever servlet container you prefer to use (Tomcat , Apache etc)
- Jenkins comes pre-packaged with a servlet if you just want a lightweight implementation
- Native/Supported packages exist for:

  Windows Ubuntu/Debian Redhat/Fedora/CentOS Mac OSX openSUSE FreeBSD OpenBSD Solaris
- Cloudbees - http://www.cloudbees.com/

Core Features

For this current course we have already created the Jenkins CI server on Ubuntu for you, please ask mentor to provide the credentials.

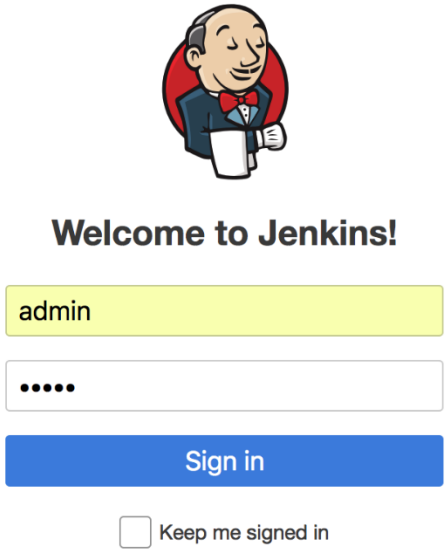
 *- by this sign we would mark the practice part*

Get to the course Jenkins environment using the following credentials:

http://<your course PC IP>:8080/

User: ***admin***

Password: ***admin***



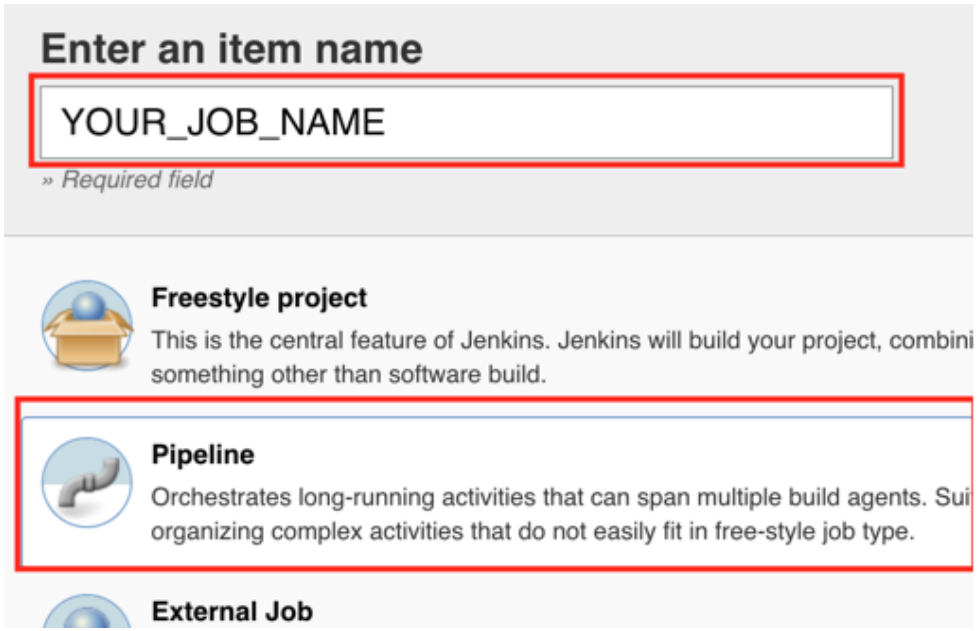
Core Features

When setting up a project in Jenkins, out of the box you have the following general options:

- Associating with a version control server
- Triggering builds
- Polling, Periodic, Building based on other projects
- Execution of shell scripts, bash scripts, Ant targets, and Maven targets Artifact archival
- Publish
- JUnit test results and Javadocs
- Email notifications
- Plugins expand the functionality even further

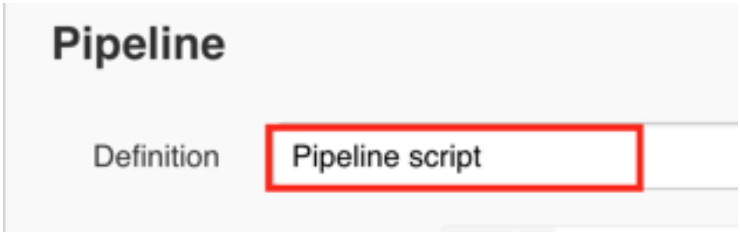
Core Features

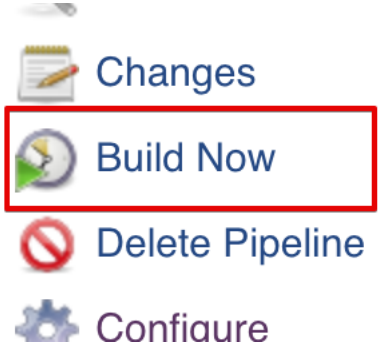
Go to the Jenkins portal and create the first test job:



Core Features

Set the pipeline definition as "Pipeline Script" and start creating the code:





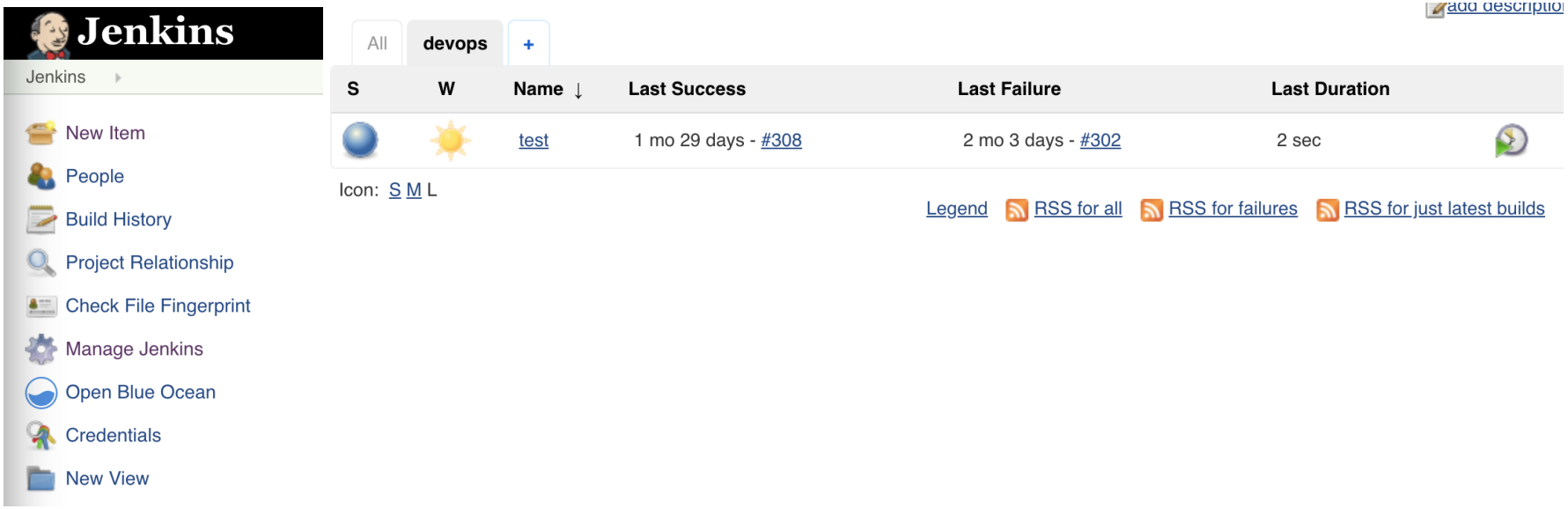
The code could be like this

```
node {
//My first test pipeline
echo 'Thanks John Bryce and Automat-IT for giving us a hand :-)'
}
```

*Run the code and see the output. The "**echo**" command prints the output in ' '.*

Core Features

- The main page provides a summary of the projects
- Quick view of What's building ("No builds in the queue")
- Build Executor Status (both "Idle")
- Status of the projects



Groovy

The language we used for our first pipeline was **Groovy**.

Apache Groovy is a **Java-syntax-compatible** object-oriented programming language for the Java platform. It is both a **static** and **dynamic** language with features similar to those of **Python**, **Ruby, Perl**, and **Smalltalk**.

Groovy

It can be used as both a **programming** language and a **scripting** language for the Java Platform, is compiled to Java virtual machine (JVM) bytecode, and interoperates **seamlessly** with other Java code and libraries. Groovy uses a **curly-bracket** syntax similar to Java's. Groovy supports closures, multiline strings, and expressions embedded in strings. Much of Groovy's power lies in its AST transformations, triggered through annotations.

Groovy VS Java

Apart of being syntax-compatible, the Groove the following odds:

1. ***Default imports*** *(some of the classes are included by the default, no need to import them initially)*
2. ***Multi-methods*** *(methods are chosen based on the types of the arguments at runtime not during the compilation)*
3. ***Array initializers*** *(int[] array = [1,2,3], not {})*
4. ***Package scope visibility*** *(In Groovy, omitting a modifier on a field doesn't result in a package-private field like in Java)*
5. ***Automatic Resource Management blocks*** *(not supported in Groovy)*

Groovy VS Java

JOHN BRYCE
Leading in IT Education
a matrix company

6. ***Inner classes*** *(If you absolutely need an inner class, you should make it a static one.)*

7. ***GStrings*** *(As double-quoted string literals are interpreted as GString values, Groovy may fail with compile error or produce subtly different code if a class with String literal containing a dollar character is compiled with Groovy and Java compiler.)*

8. ***String and Character literals*** *(Singly-quoted literals in Groovy are used for String, and double-quoted result in String or GString, depending whether there is interpolation in the literal.)*

9. ***Primitives and wrappers*** *(Because Groovy uses Objects for everything, it autowraps references to primitives.)*

Groovy VS Java

JOHN BRYCE
Leading in IT Education
a matrix company

*10. **Behaviour of ==** (In Java == means equality of primitive types or identity for objects. In Groovy == translates to a.compareTo(b)==0, if they are Comparable, and a.equals(b) otherwise. To check for identity, there is is. E.g. a.is(b).)*

*11. **Conversions** (Groovy expands greatly on automatic widening and narrowing conversions of Java)*

*12. **Extra keywords** (There are a few more keywords in Groovy than in Java. Don't use them for variable names etc. **as, def, in, trait**)*

Why Jenkins? Plugins flexibility!

- Jenkins is a highly configurable system by itself
- The additional community developed plugins provide even more flexibility
- By combining Jenkins with Ant, Gradle, or other Build Automation tools, the possibilities are limitless

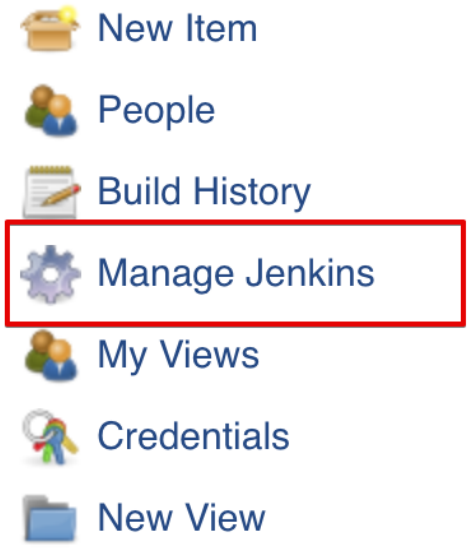
Manage Plugins

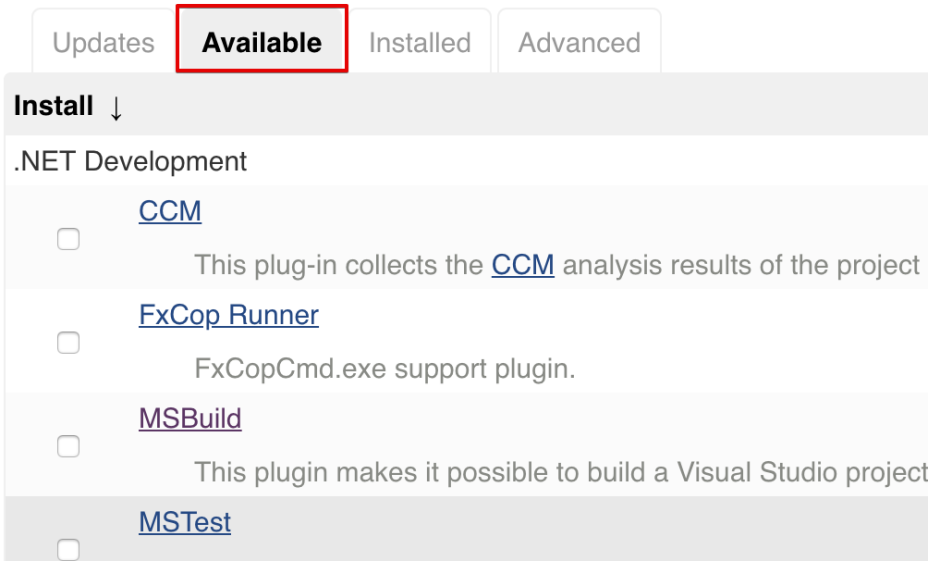
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

There are updates available

Why Jenkins? Plugins flexibility!

*Go to the Jenkins portal and opt for the "**Manage Jenkins**" → "**Manage Plugins**":"**Available**":*



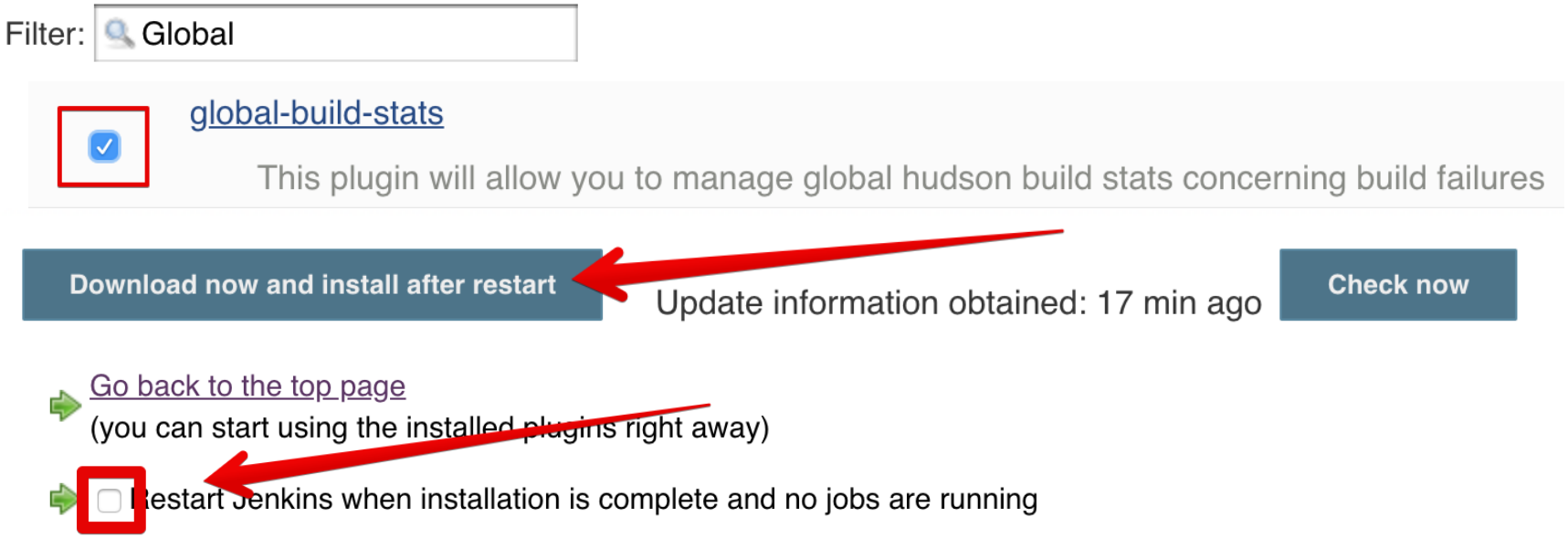


See over 1600+ plugins within the descriptions to install

Why Jenkins? Plugins flexibility!

JOHN BRYCE
Leading in IT Education
a matrix company

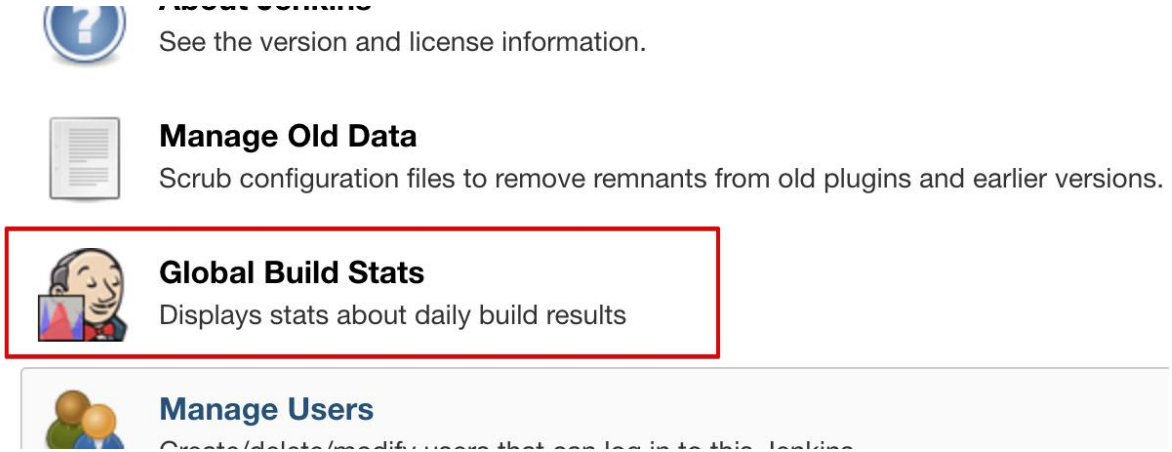
Find the "global-build-stats" plugin and install it:



Wait till the Jenkins is restarted

Why Jenkins? Plugins flexibility!

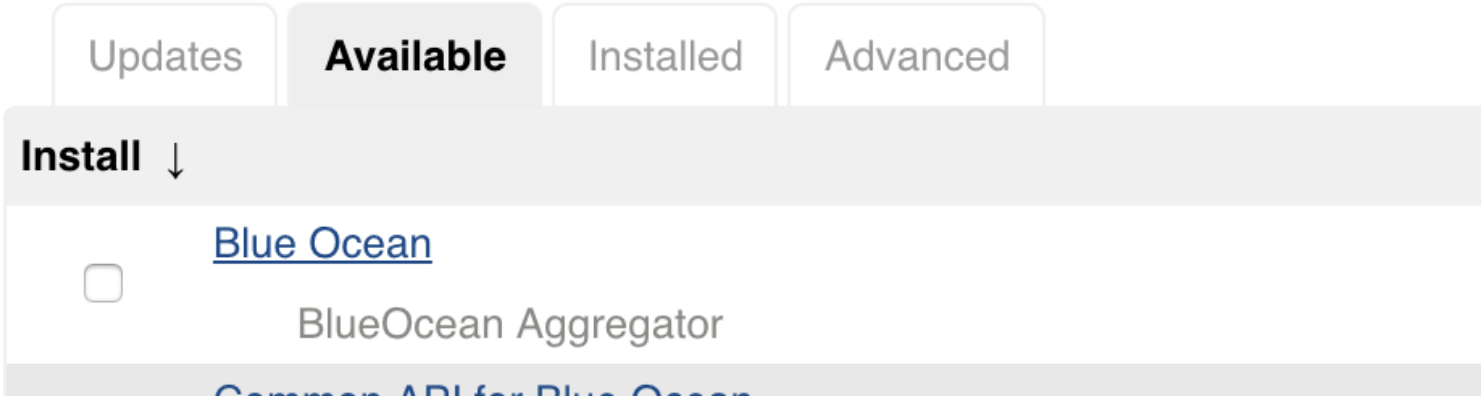
*Go to the Jenkins portal and opt for the "**Manage Jenkins**" and check the new feature installed:*



The same easy way we could adapt the Jenkins to cope with any demands we may have desired. Flexibility!

Why Jenkins? Plugins flexibility!

*Now please install the **"BlueOcean UI Plugin"**:*



You would see a lot of the plugins to be installed. No worries, those are dependency plugins.

Plugins: Blue Ocean UI plugin

JOHN BRYCE
Leading in IT Education
a matrix company

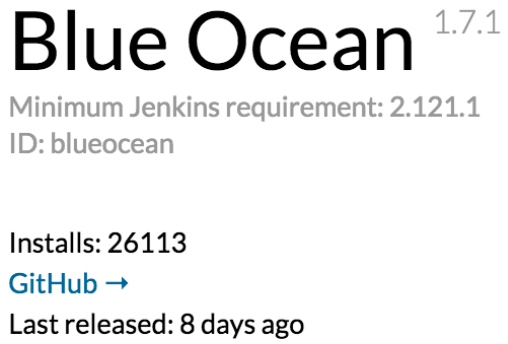
Sophisticated visualizations of CD pipelines, allowing for fast and intuitive comprehension of software pipeline status.

Pipeline editor that makes automating CD pipelines approachable by guiding the user through an intuitive and visual process to create a pipeline.

Personalization of the Jenkins UI to suit the role-based needs of each member of the DevOps team.

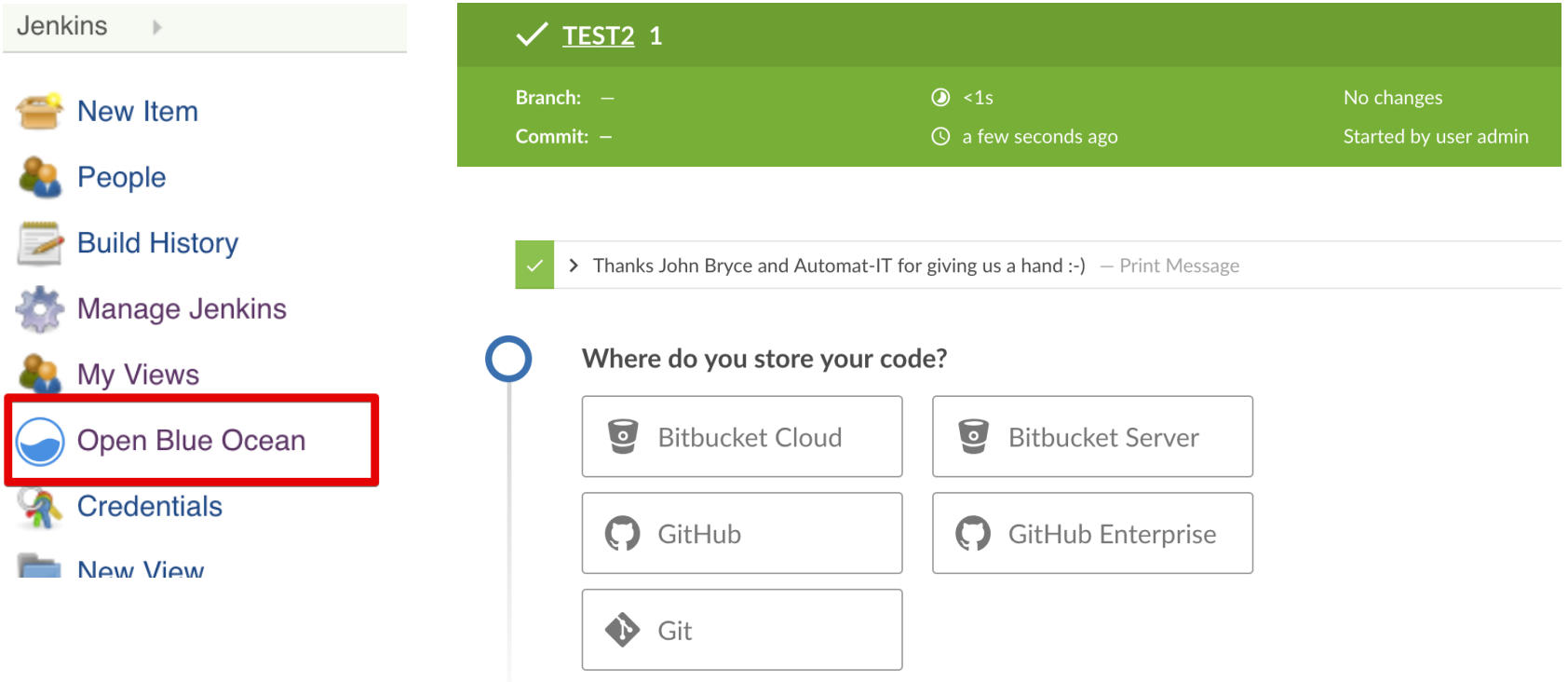
Pinpoint precision when intervention is needed and/or issues arise. The Blue Ocean UI shows where in the pipeline attention is needed, facilitating exception handling and increasing productivity.

Native integration for branch and pull requests enables maximum developer productivity when collaborating on code with others in GitHub and Bitbucket.



Plugins: Blue Ocean UI plugin

We could see the new Blue Ocean item on the main menu and the previous job output:



Have it a try and see the new interface and pipeline creation/launch wizard

Plugins: Code Repository - Git

JOHN BRYCE
Leading in IT Education
a matrix company

Git is a version control system created by Linus Torvalds in 2005 for development of the Linux kernel, having multiple advantages over the other systems available, it stores file changes more efficiently and ensures file integrity better.

- Local projects directory- files.
- Single ".git" directory.
- To review the basic thigs, please follow our GIT course

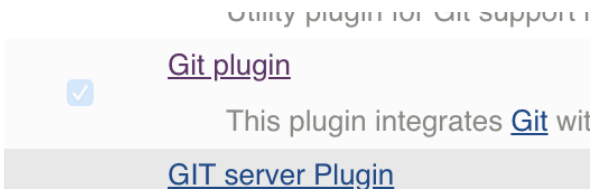
Plugins: GitHub plugin

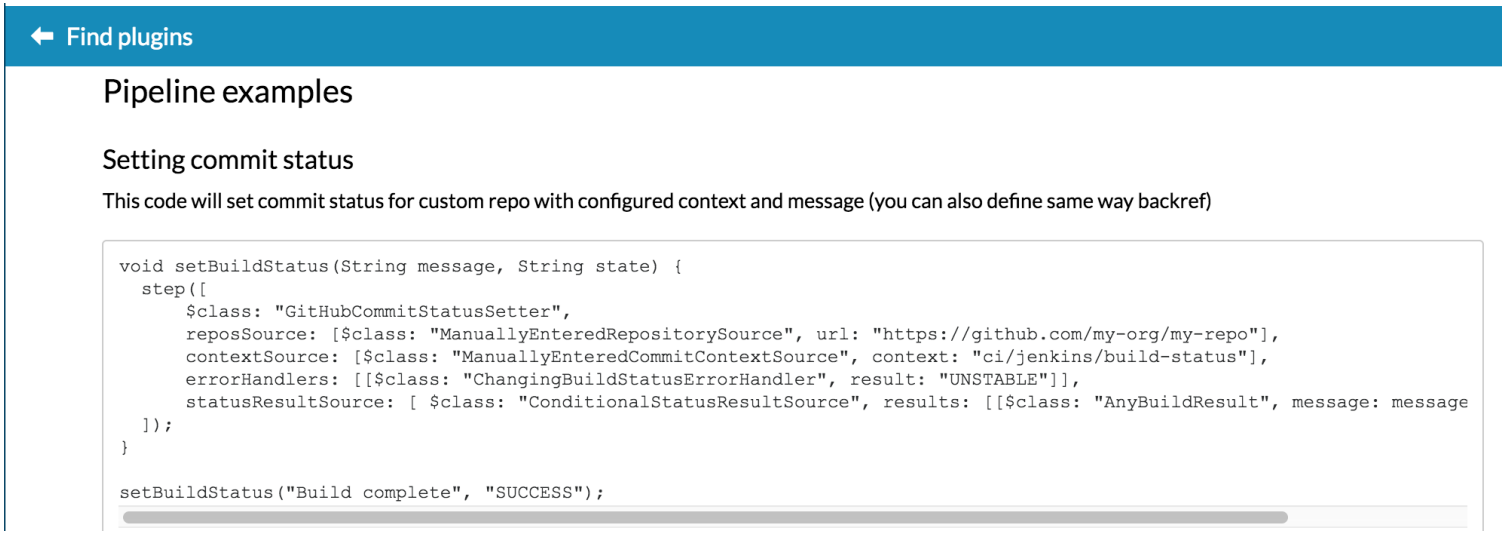
JOHN BRYCE
Leading in IT Education
a matrix company

- Create hyperlinks between your Jenkins projects and GitHub
- Trigger a job when you push to the repository by froking HTTP POSTs from post-receive hook and optionally auto-managing the hook setup.
- Report build status result back to github as Commit Status (documented on SO)
- Base features for other plugins

Plugins: GitHub plugin

The crucial thing about the plugins is that you could drill down the plugin details out of the Jenkins and see the full description with usage:





Plugins: Builds

JOHN BRYCE
Leading in IT Education
a matrix company

- Once a project is successfully created in Jenkins, all future builds are automatic
- Jenkins executes the build in an executer
- By default, Jenkins gives one executor per core on the build server
- Jenkins also has the concept of slave build servers
  - Useful for building on different architectures
  - Distribution of load

Manage Nodes

Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Plugins: MSBuild plugin

- This plugin allows you to use MSBuild to build .NET projects.

Usage

- To use this plugin, specify the location directory of MSBuild.exe on Jenkin's configuration page.
- Then, on your project configuration page, specify the name of the build file (.proj or .sln) and any command line arguments you want to pass in.
- The files are compiled to the directory where Visual Studio would put them as well.

Plugins: Maven Integration plugin

This plugin provides an advanced integration for Maven 2/3 projects.

- Automatic configuration of reporting plugins (Junit, Findbugs, ...)
- Automatic triggering across jobs based on SNAPSHOTs published/consumed
- Incremental build - only build changed modules
- Build modules in parallel on multiple executors/nodes
- Post build deployment of binaries only if the project succeeded and all tests passed

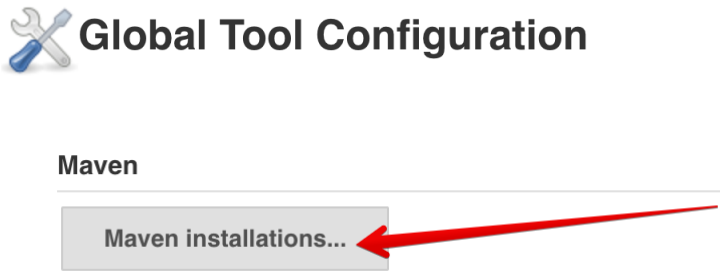
Maven Integration 3.1.2

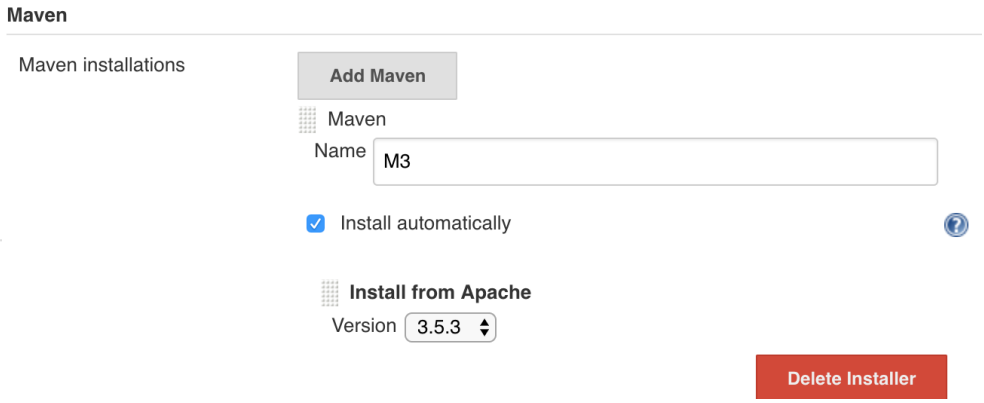
Minimum Jenkins requirement: 1.625.3
ID: maven-plugin

Plugins: Maven Integration plugin

*Apache Maven is a software project management and comprehension tool. Based on the concept of a **project object model** (**POM**), Maven can manage a project's build, reporting and documentation from a central piece of information.*

*The maven tool is already configured on our test Jenkins. To check that, please go to the Jenkins portal and opt for the "**Manage Jenkins**" → "**Global Tool Configuration**":*





***Name** – alias for the Jenkins Pipeline*

***Version** – the version of the maven to be installed once demanded*

Plugins: Maven Integration plugin

The pipeline stage to build the maven project should look like this:

```
...
stage('Maven build')
      {
       // Get the Maven tool alias
       mvnHome = tool 'M3'
       // Run the maven package
       echo sh (returnStdout: true, script: "'${mvnHome}/bin/mvn' package")
      }
...
```

In case of having any maven project in the workspace directory (ls –ltr ~/workspace/<YOUR_JOB_NAME>) it would build the artifacts for the deployment and testing.

Plugins: Simple Theme Plugin

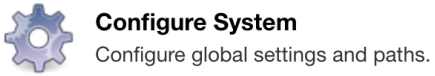
JOHN BRYCE
Leading in IT Education
a matrix company

This plugin allows to customize Jenkin's appearance with custom CSS and JavaScript. It also allows to replace the Favicon:

- Use your own CSS.
- Use your own JavaScript.
- Replace the Jenkins Favicon by your companies' one.

Plugins: Simple Theme Plugin

- *Go to the Jenkins configuration and try to apply one of the given themes:*



"Atlassian"

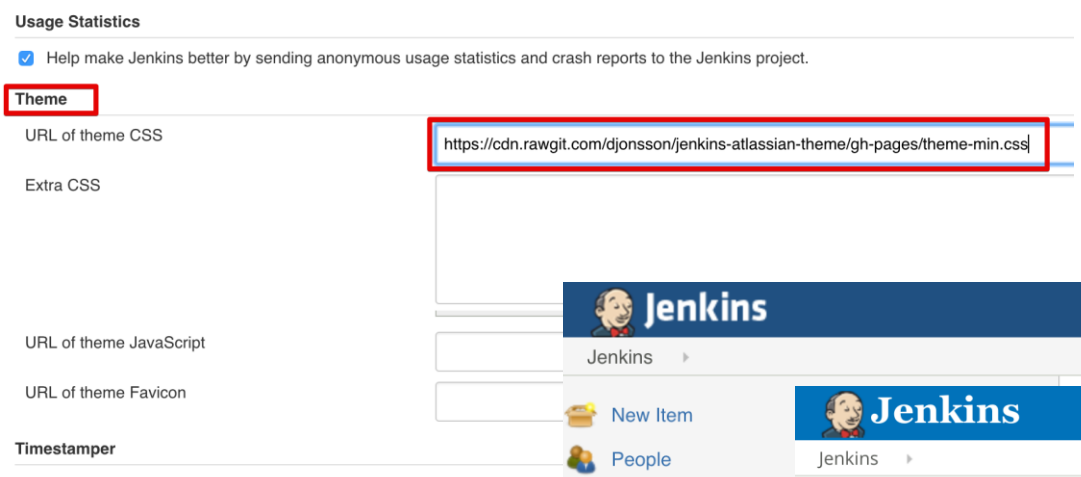
Doony

Material

Neo2

Rackspace Canon

Plugins: Simple Theme Plugin



Testing

- Unit testing - Does my small part of code working properly in my system, my class?
- Integration Testing Does our code work correctly against code we can't change?
- Acceptance testing - Does whole system working properly?

Plugins: Selenium Plugin

- Selenium automates browsers. That's it!
- This plugin turns your Jenkins cluster into a Selenium2 Grid
- This plugin sets up Selenium Grid in the following way:

  - On master, Selenium Grid Hub is started on port 4444, unless configured otherwise in Jenkins global configurations. This is where all your tests should connect to.
  - For each slave, necessary binaries are copied and Selenium RCs are started.
  - RCs and the Selenium Grid Hub are hooked up together automatically

Plugins: Docker plugin

Docker plugin allows to use a docker host to dynamically provision build agents, run a single build, then tear-down agent.

Optionally, the container can be committed, so that (for example) manual QA could be performed by the container being imported into a local docker provider, and run from there.

Reporting & notifications

JOHN BRYCE
Leading in IT Education
a matrix company

Jenkins comes with basic reporting features

- Keeping track of build status

	Build	Time Since ↑	
	test #308	1 mo 29 days	stable
	test #307	1 mo 29 days	stable
	test #306	2 mo 3 days	stable
	test #305	2 mo 3 days	stable

- Last success and failure

Last Success	Last Failure
1 mo 29 days - #308	2 mo 3 days - #302

- "Weather" – Build trend

S	W

Reporting & notifications

The basial reporting can be greatly enhanced with the use of pre-build plugins

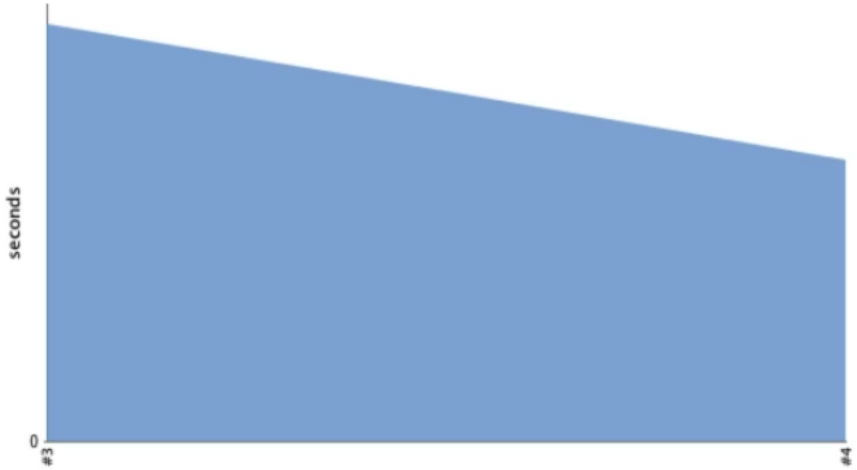
- Unit test coverage
- Test result trending
- Findbugs, Checkstyle, PMD

Reporting & notifications

JOHN BRYCE
Leading in IT Education
a matrix company

- History
- Git Build Data
- No Tags
- Test Result
- Replay
- Pipeline Steps
- Previous Build
- Next Build

History for Test Results



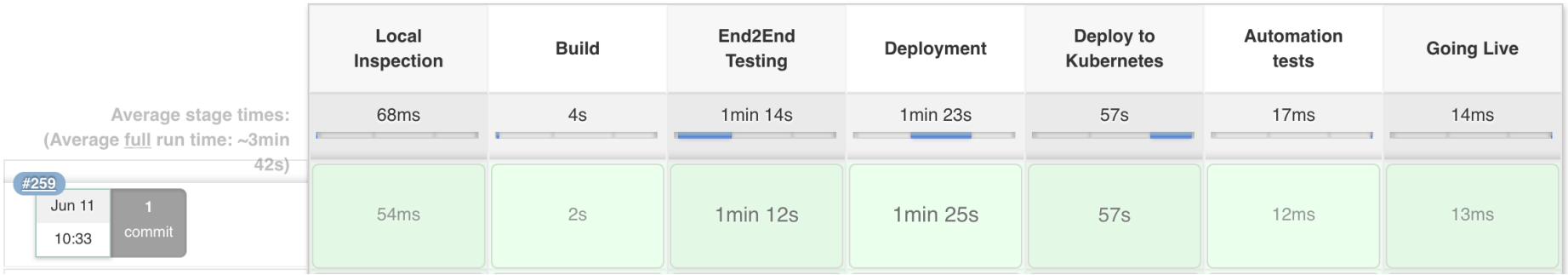
show count

Build	Description	Duration	Fail	Skip	Total
petclinic pipeline #4		6.1 sec	0	0	62
petclinic pipeline #3		9.1 sec	0	0	62

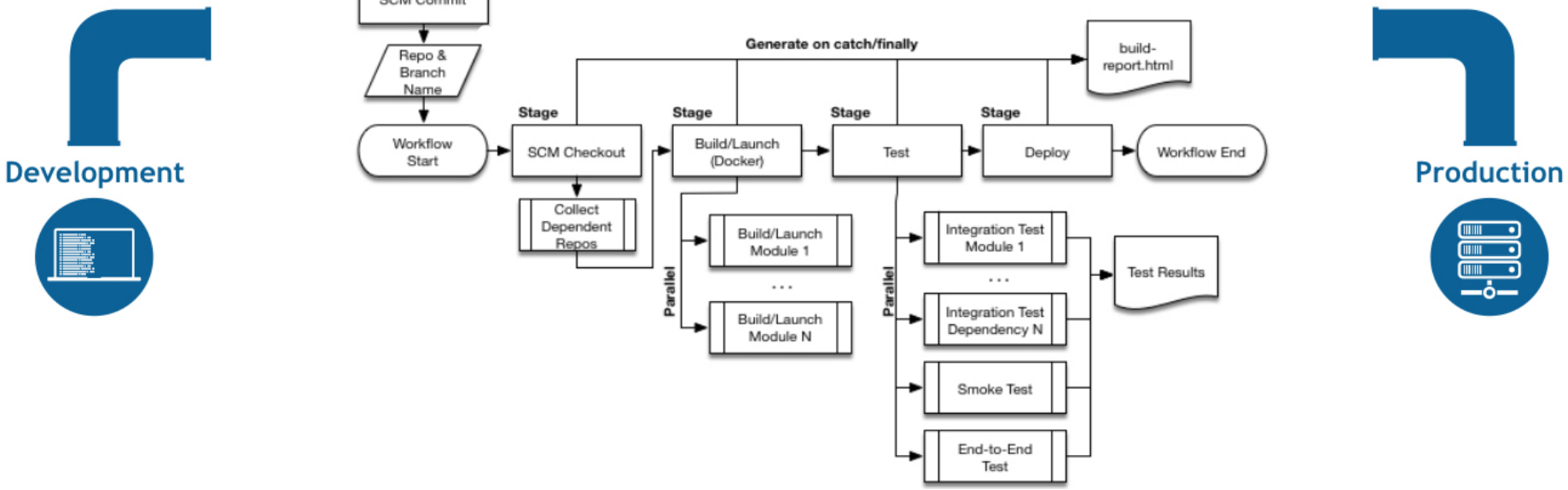
Pipeline

Key Features and Benefits of Pipelines:

- Pipeline as Code. You could supervise and keep the changes stored in GIT
- Easily define simple and complex pipelines through the DSL in a Jenkinsfile.
- Pipeline as code provides a common language to help teams (e.g. Dev and Ops) work together.
- Easily share pipelines between teams by storing common "steps" in shared repositories.



Pipeline



Pipeline

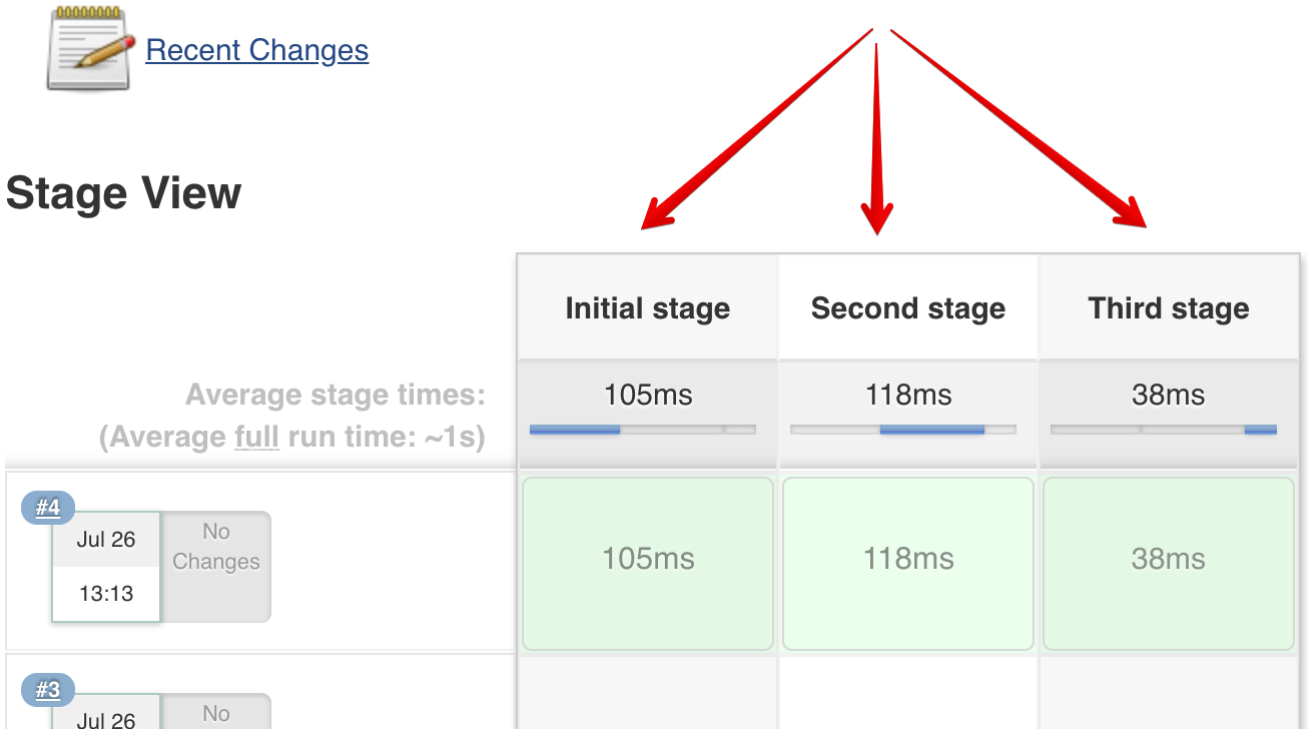
Alter our first pipeline to have stages. The code could be like this:

```
node {
//My first test pipeline
   stage('Initial stage')
      {
         echo 'Thanks John Bryce and Automat-IT for giving us a hand :-)'
      }
   stage('Second stage')
      {
         def text = 'John'
         def text2 = 'Bryce'
         echo "${text} ${text2} has reached the second stage"
      }
   stage('Third stage')
      {
         println 'Thanks for your patience'
      }
}
```

*Run the code and see the output. The "**def**" command defines a variable.*

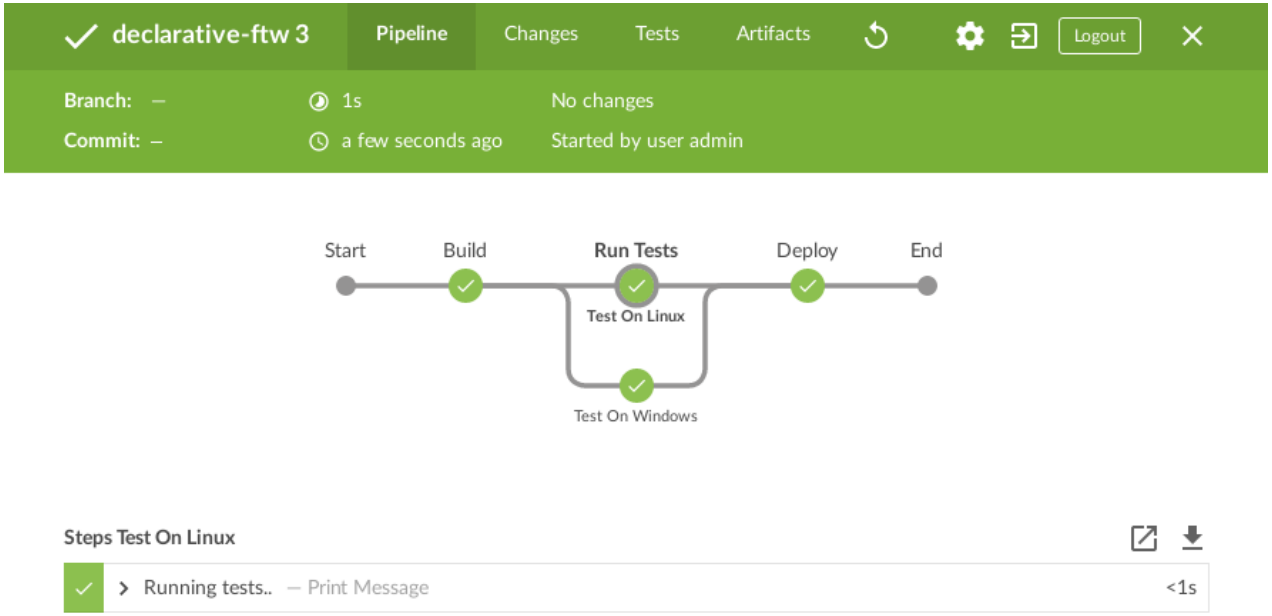
Pipeline

As the result we would see the following pipeline with the 3 stages:



Parallel pipeline

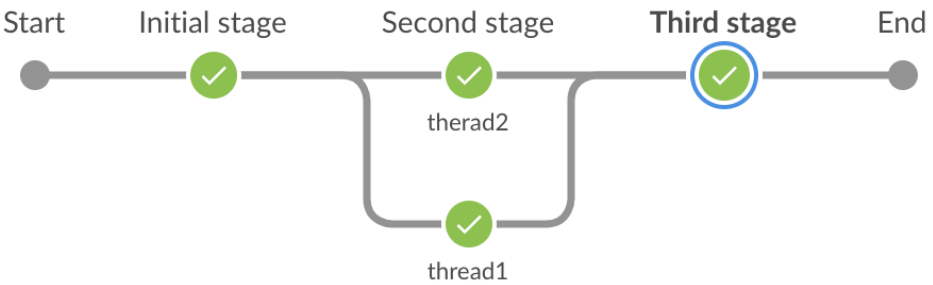
In some cases there is a need to run the scripts in parallel. For instance we have a test step to run on the different nodes with different OS and in a sake of saving the time resource we could write the parallel commands to be triggered simultaneously:



Parallel pipeline

Let's modify our pipeline's second stage to run some commands in parallel:

```
...
stage('Second stage')
      {
        def text = 'John'
        def text2 = 'Bryce'
        echo "${text} ${text2} has reached the second stage"
        parallel(
            thread1: {
              echo "Running thread1"
              },
            therad2: {
              echo "Running thread2"
              }
           )
      }
...
```



Methods in pipeline

JOHN BRYCE
Leading in IT Education
a matrix company

There are cases when we do need to run the same logic on the pipeline more than one time or a massive code setup needs to be structured and optimized The good practice for that is to create a **method**.

A method in Groovy is defined with a return type or with the **def** keyword. Methods can receive any number of arguments:

```
def methodName() {
    //Method code
}
```

Modifiers such as public, private and protected can be added. By default, if no visibility modifier is provided, the method is **public**.

Methods in pipeline

*Let's modify our pipeline's and add our first method **after** the **node** section:*

```
...
    stage('Third stage')
        {
            println 'Thanks for your patience'
        }
}
def getTextMessage()
{
    a1 = "Thanks"
    a2 = "for your patience"
    return "${a1} ${a2}"
}
```

Methods in pipeline

Let's modify our third stage to use the method:

```
...
    stage('Third stage')
        {
            println getTextMessage()
        }
}
def getTextMessage()
{
    a1 = "Thanks"
    a2 = "for your patience"
    return "${a1} ${a2}"
}
```

```
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Third stage)
[Pipeline] echo
Thanks for your patience
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Methods in pipeline

JOHN BRYCE
Leading in IT Education
a matrix company

Methods could be parametrized as well:

```
def methodName(inner_meth_par_1, inner_meth_par_2) {
        //Method code
}
```

It is usually opted for creating the logic sub-blocks. For instance, here we are creating the branch in the repository:

```
def createNewBranch(source, destination){
withCredentials([[$class: 'UsernamePasswordMultiBinding', credentialsId: 'bitbucket',
        usernameVariable: 'USERNAME', passwordVariable: 'PASSWORD']]){
        def apiUrl = "https://api.bitbucket.org/2.0/repositories/${getRepoSlug()}/src/"
        echo sh(returnStdout: true, script: "curl -u ${env.USERNAME}:${env.PASSWORD} -F \"parents=${source}\" -F \"branch=${destination}\" ${apiUrl}")
        } }
```

Methods in pipeline

Let's add our parametrized method:

```
...
a2 = "for your patience"
    return "${a1} ${a2}"
}
def getTextMessageParam(parameter_1, parameter_2)
{
    a1 = parameter_1.toUpperCase()
    a2 = parameter_2.toLowerCase()
    echo "Running parametrized method"
    return "${a1} ${a2}"
}
```

Methods in pipeline

Modify our third step to use our second method :

```
...
stage('Third stage')
{
 println getTextMessage()
 println getTextMessageParam("shOuld be iN upper cAse,", "LOWERcAse is HERe")
 }
}
...
```

```
[Pipeline] echo
Thanks for your patience
[Pipeline] echo
Running parametrized method
[Pipeline] echo
SHOULD BE IN UPPER CASE, lowercase is here
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Shared Libraries

As Pipeline is adopted for more and more projects in an organization, common patterns are likely to emerge. Oftentimes it is useful to share parts of Pipelines between various projects to reduce redundancies and keep code "DRY" and easily modify the block chain.

Pipeline has support for creating "**Shared Libraries**" which can be defined in external source control repositories and loaded into existing Pipelines.

Shared Libraries

JOHN BRYCE
Leading in IT Education
a matrix company

The directory structure of a Shared Library repository is as follows:

```
(root)
+- src                # Groovy source files
|   +- org
|       +- foo
|           +- Bar.groovy  # for org.foo.Bar class
+- vars
|   +- foo.groovy        # for global 'foo' variable
|   +- foo.txt           # help for 'foo' variable
+- resources             # resource files (external libraries only)
|   +- org
|       +- foo
|           +- bar.json    # static helper data for org.foo.Bar
```

Shared Libraries

JOHN BRYCE
Leading in IT Education
a matrix company

The **src** directory should look like standard Java source directory structure. This directory is added to the classpath when executing Pipelines.

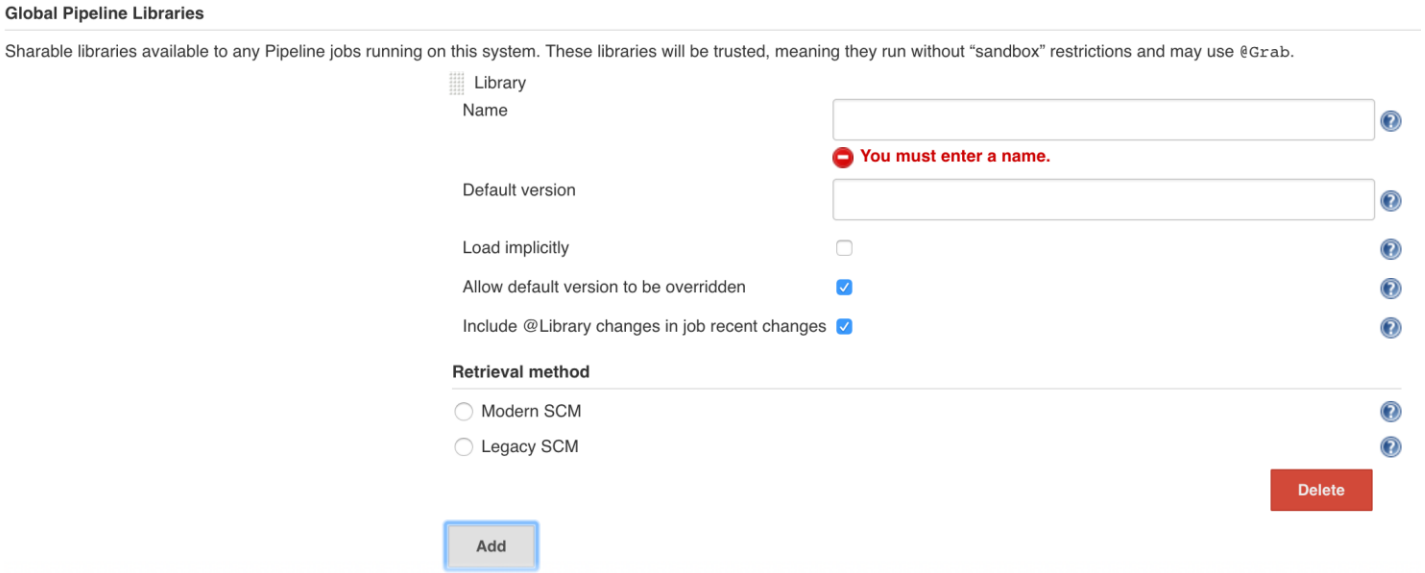
The **vars** directory hosts scripts that define global variables accessible from Pipeline. The basename of each **.groovy file should be a Groovy (~ Java) identifier, conventionally camelCased.**

The matching .txt, if present, can contain documentation, processed through the system's configured markup formatter (so may really be HTML, Markdown, etc., though the txt extension is required).

Shared Libraries

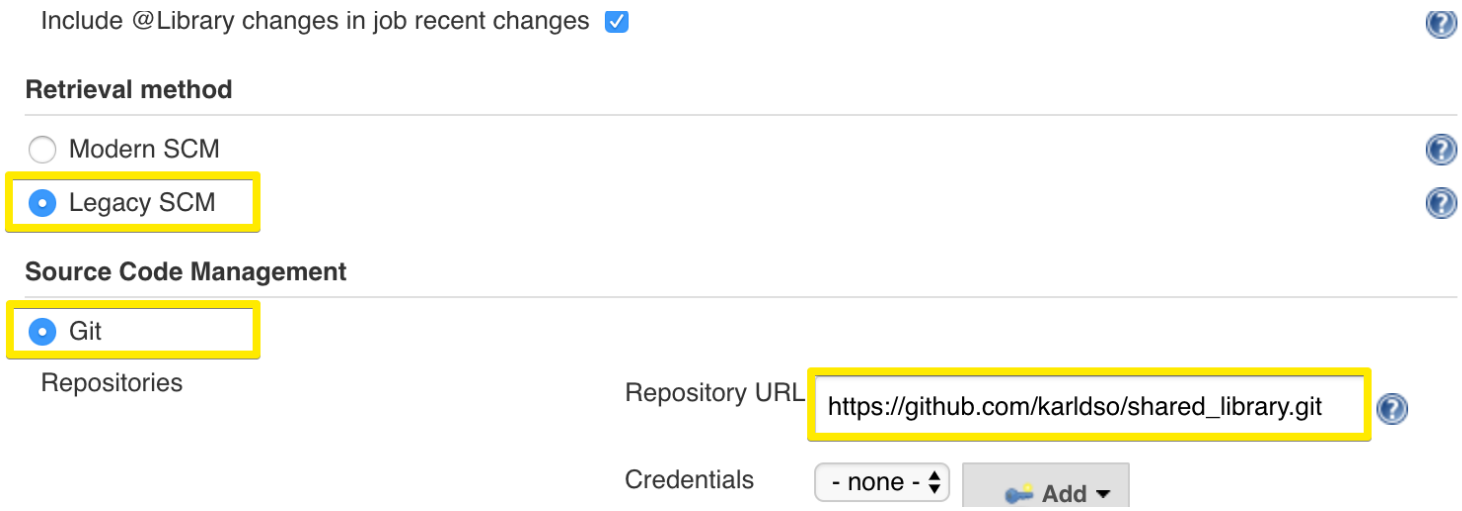
JOHN BRYCE
Leading in IT Education
a matrix company

Basically, the shared libraries are the predefined pipelines and methods which could be evoked out of the initial pipeline. To plug in the library, please go to the "**Configure system**" →"**Global Pipeline Libraries**" section and add the relevant pipeline:



Shared Libraries

Test shared library: ***https://github.com/karldso/shared_library.git***



Methods in pipeline

Add the library to the pipeline and create the fourth step to use it. Add to the top:

```
@Library ('JB_shared@1.0') _
...
stage('Fours stage')
   {
      testStep{}
   }
}
def getTextMessage()
{
   a1 = "Thanks”
...
```

eline script

Script

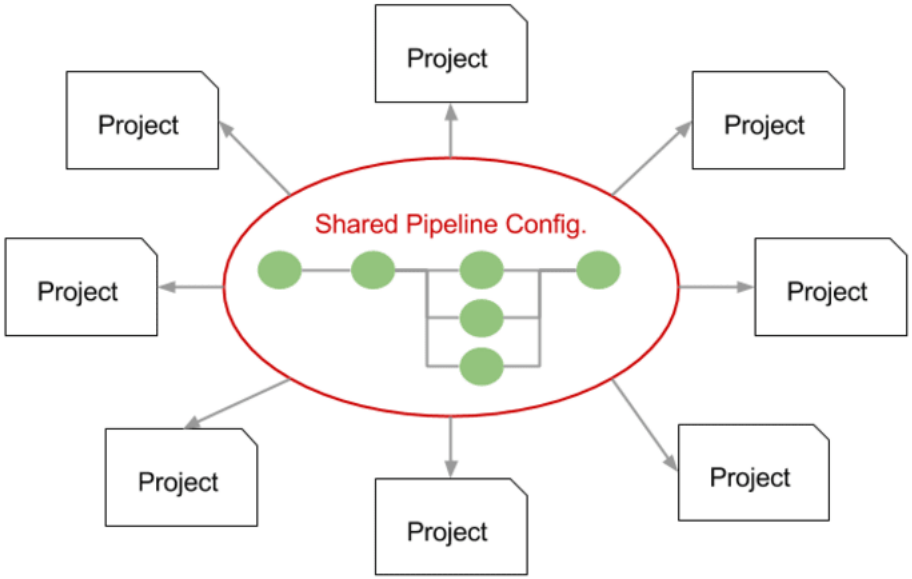
```
1  @Library ('JB_shared@1.0') _
2
3  node {
4  //My first test pipeline
5  stage('Initial stage')
```

```
[Pipeline] { (Fours stage)
[Pipeline] sh
[test] Running shell script
+ echo Running the JB test shared library
[Pipeline] }
[Pipeline] // stage
```

Shared Libraries

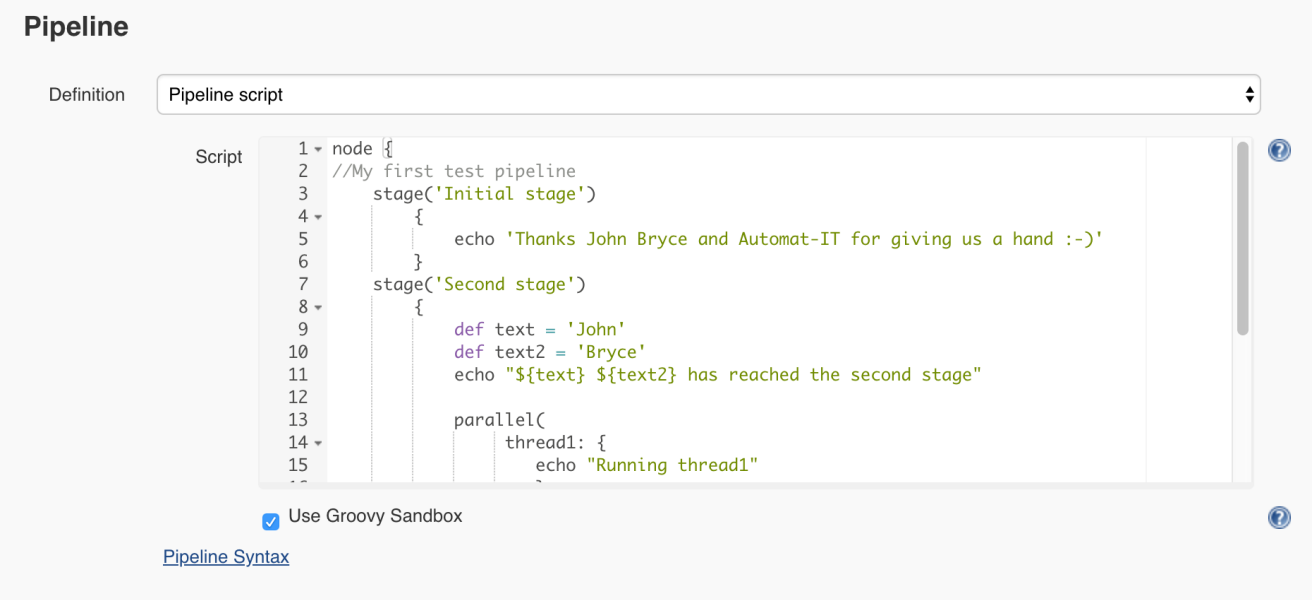
JOHN BRYCE
Leading in IT Education
a matrix company

This is a very basic illustration of how using shared libraries work. There is much more detail and functionality surrounding shared libraries, and extending your pipeline in general, than we can cover here.



JenkinsFile

Before this stage, we altered the pipeline directly in the Groovy Sandbox:

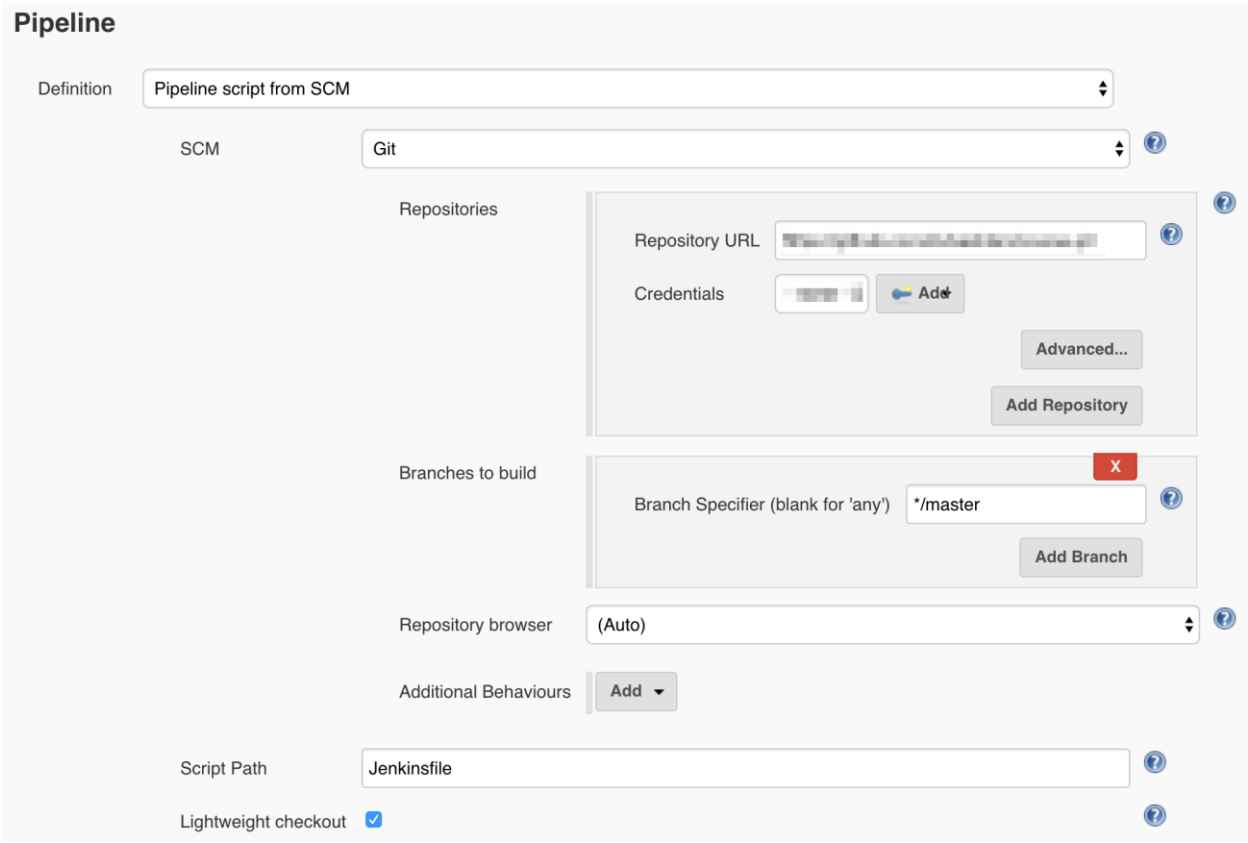


To track, supervise and allow all the changes we could keep and evoke the code right out of the repository.

JenkinsFile

JOHN BRYCE
Leading in IT Education
a matrix company

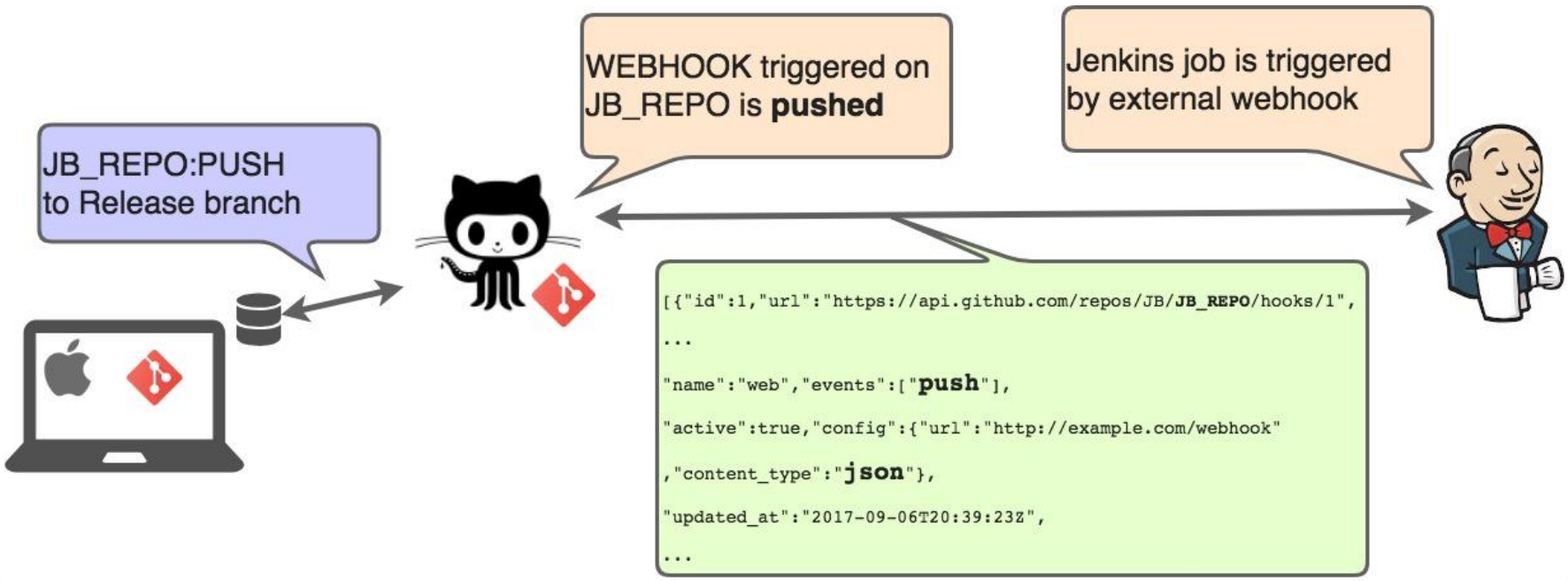
To define the Jenkinsfile usage we need to shift under the "Pipeline script from SCM":



Webhook

JOHN BRYCE
Leading in IT Education
a matrix company

As we do remember from our GIT course, the **Webhooks** provide a way for notifications to be delivered to the integrated external system (Jenkins in our case) whenever certain actions occur on a repository or organization. Previously, we triggered the job manually, let's see how to automate this a bit.



Webhook

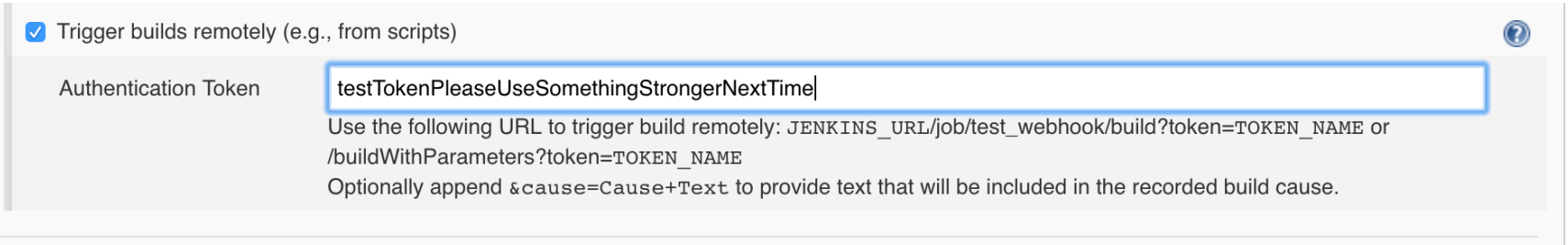
Let's modify our job to be triggered by GitHub:

- *Please install the "**Generic Webhook Trigger**" plugin:*

- *Opt for the Generic Webhook Trigger item inside the job configuration:*

Webhook

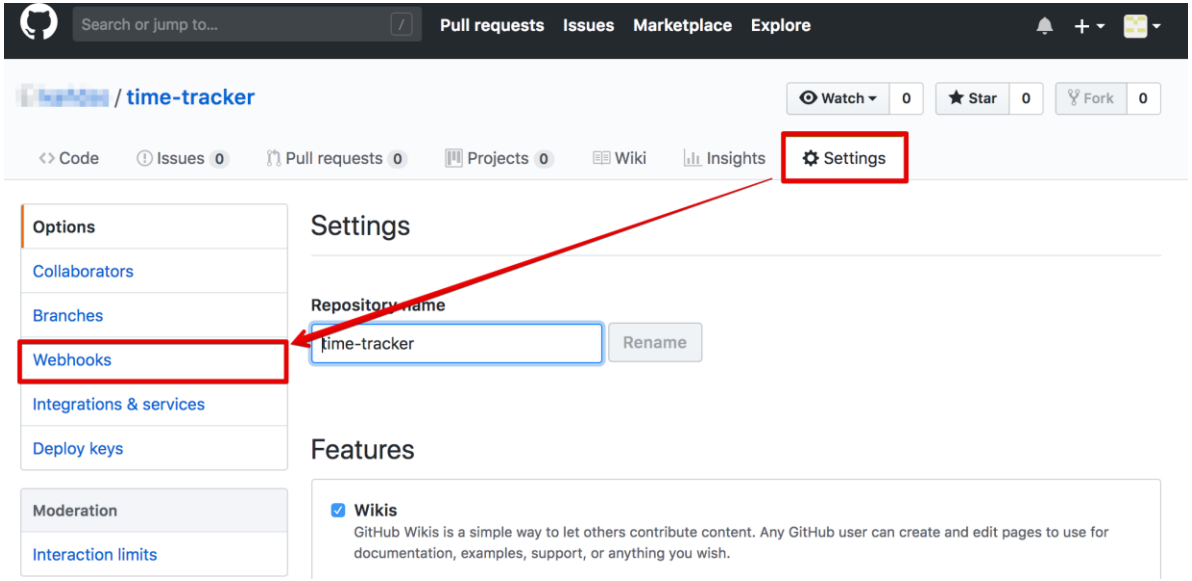
To protect our Jenkins from the webhook invokes, please setup a ***Authentication Token*** *in the job configuration:*



Save the job

Webhook

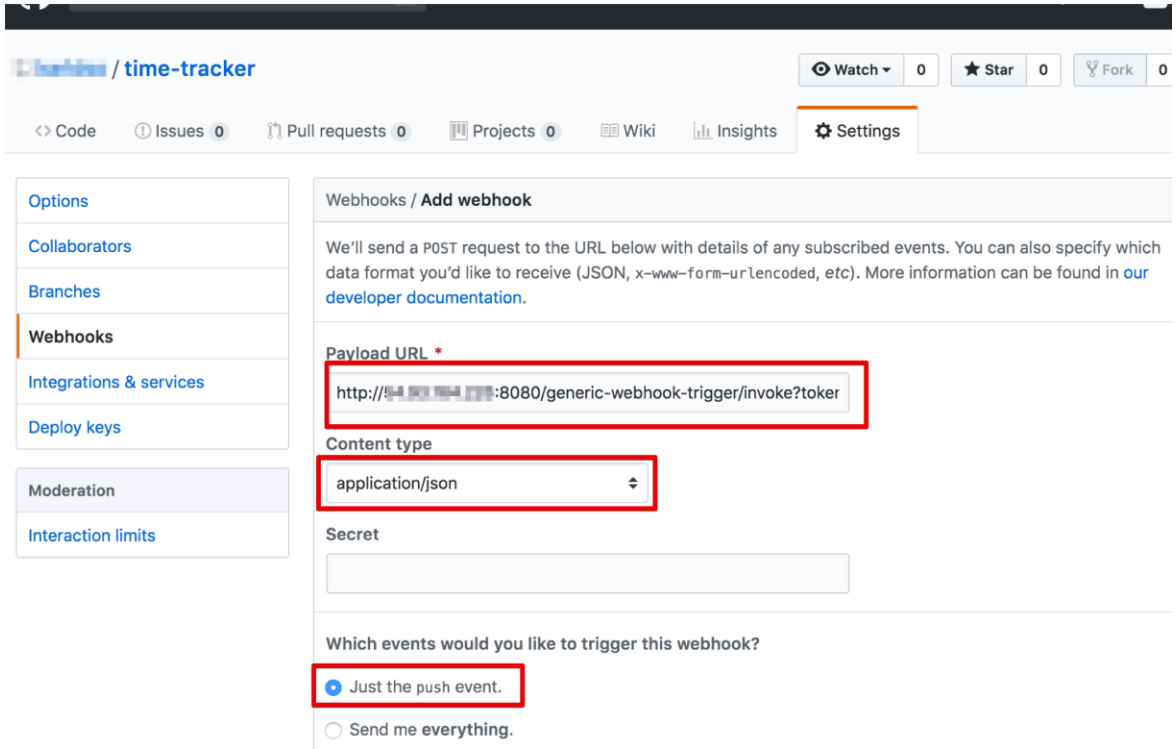
- *Go to the GitHub (https://github.com) and configure the repository to send a hook on push to our Jenkins:*



Webhook

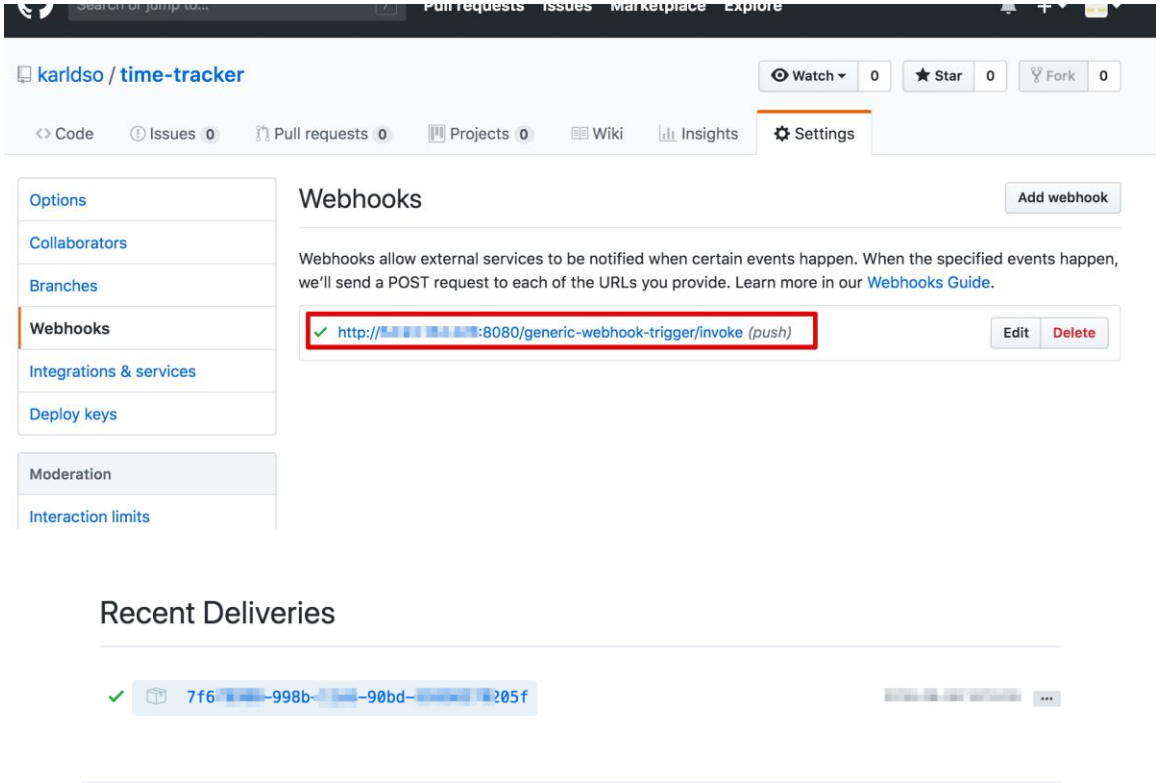
- *As the Payload URL, please use:*

http://<Jenkins IP>:8080/generic-webhook-trigger/invoke?token=testTokenPleaseUseSomethingStrongerNextTime

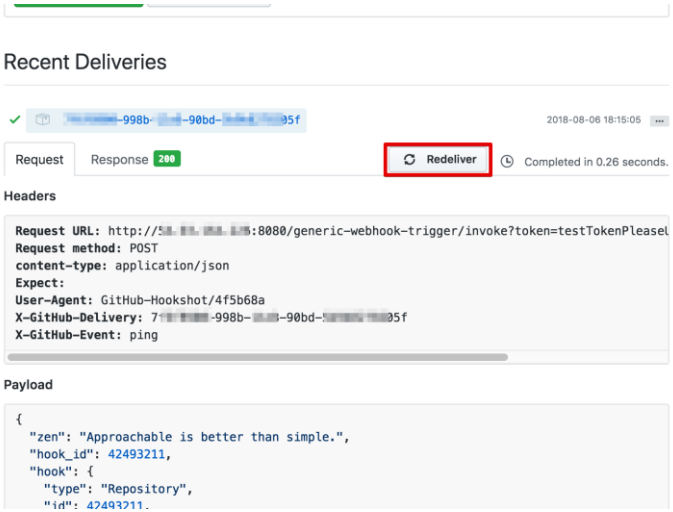


Webhook

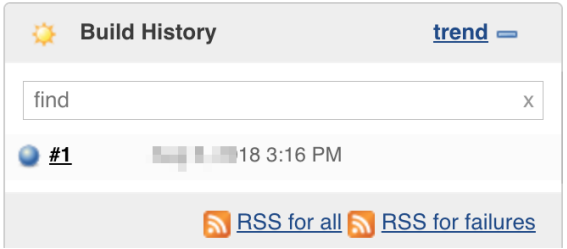
- *After the webhook activation, please drill down to the parameters and "Redeliver" the Payload to trigger our Jenkins job:*



Webhook

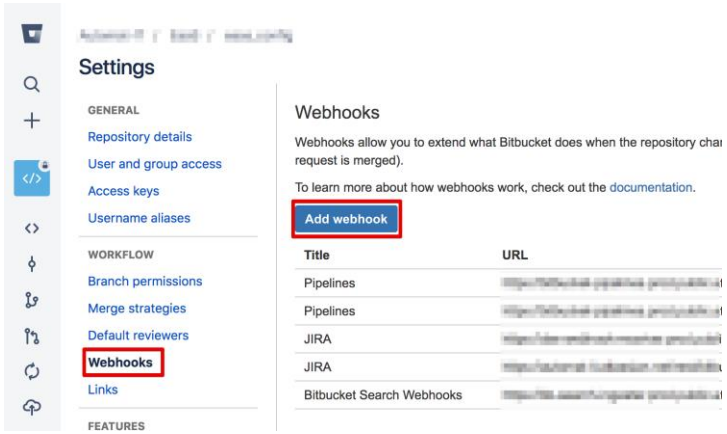


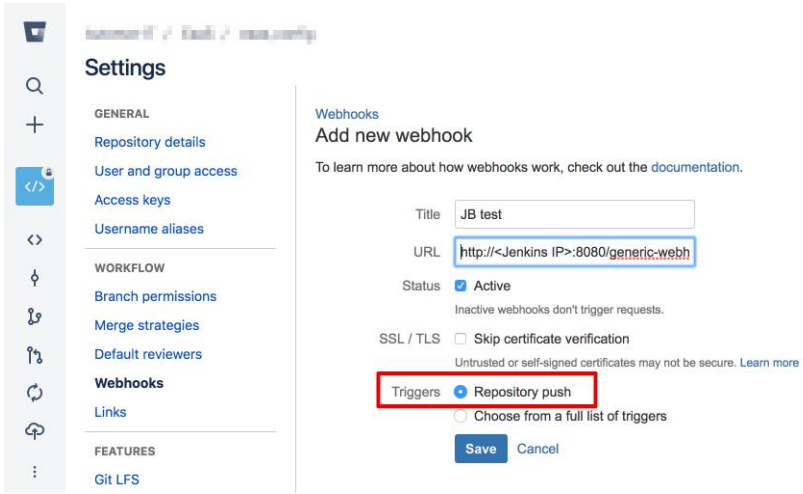
- *See the Jenkins job triggered and run:*



Webhook

- *In the very same way the **Bitbucket** repository could be configured as well:*





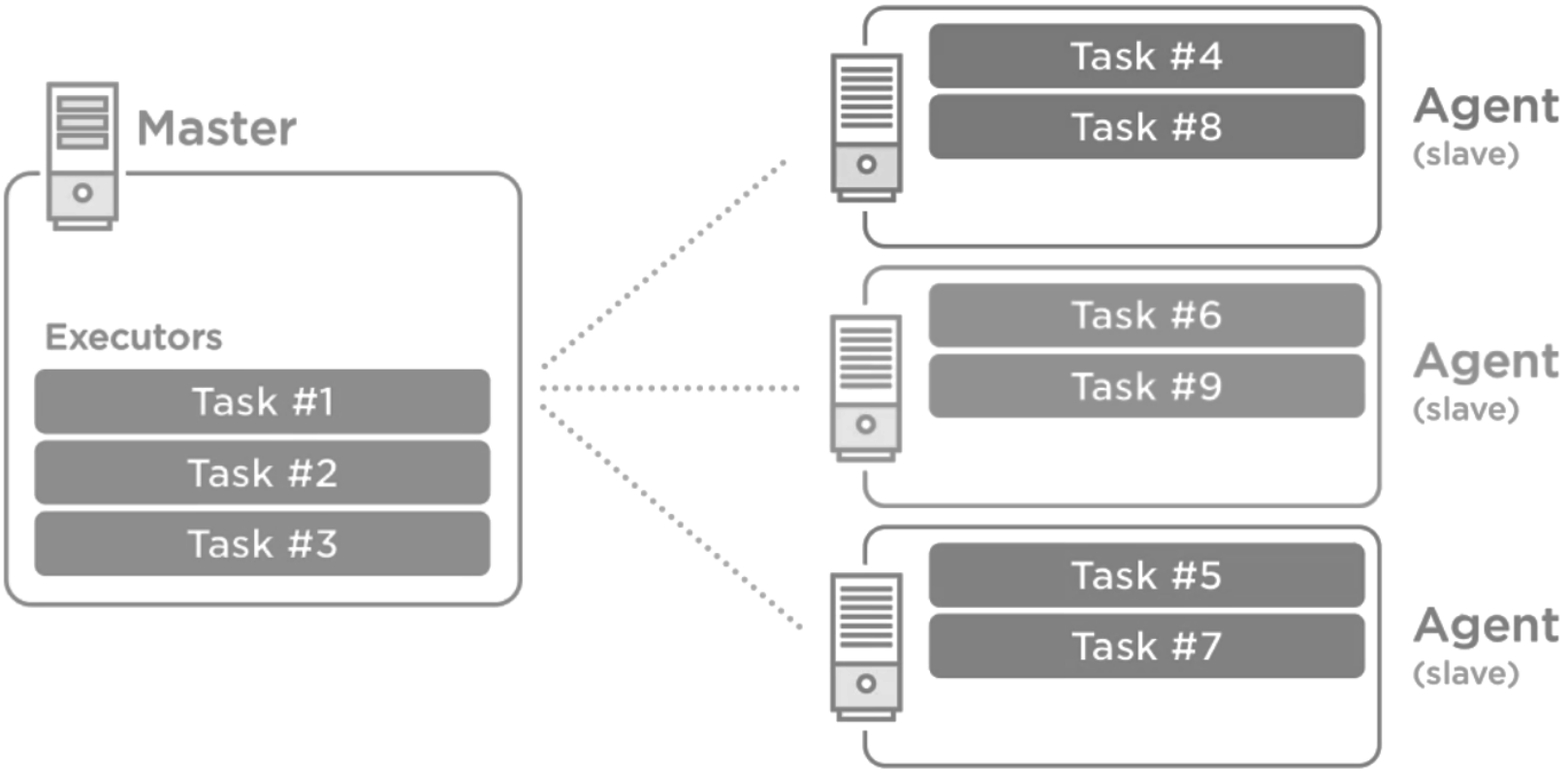
Jenkins. Master-Agent model

As the matter of practice part we have not mentioned the node to run, since we do have only master to proceed. For the real DevOps battle the one master node could not be enough.

Jenkins Master/Slave Mode

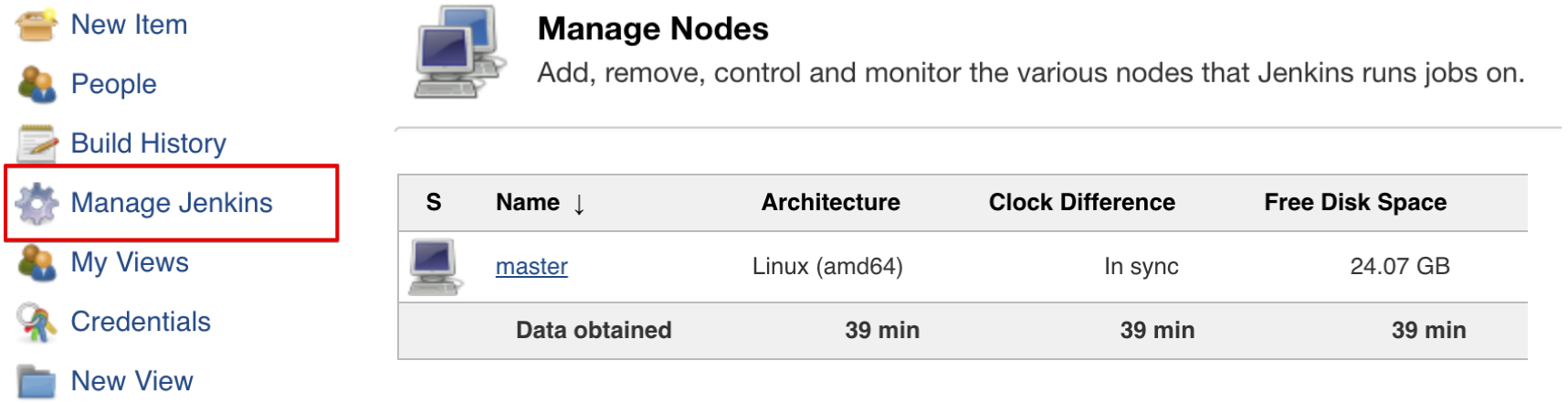
- Distribute the workload
- Provide different environments needed for builds/tests/deployments

Jenkins. Master-Agent model



Jenkins. Master-Agent model

To check the current test environment executers, go to the "Manage Jenkins" → "Manage Nodes":



Here we could configure the additional nodes to execute the pipeline on or only a part of the pipeline. Use case: the artifacts are get from the repository and for the test purposes deployed to run the test cases on the two different nodes under two different OS: MacOS and RedHeat.

Jenkins. Master-Agent model

JOHN BRYCE
Leading in IT Education
a matrix company

Agents Launch Methods:

Launch agent via Java Web Start

Allows an agent to be launched using Java Web Start. In this case, a JNLP file must be opened on the agent machine, which will establish a TCP connection to the Jenkins master. This means that the agent need not be reachable from the master; the agent just needs to be able to reach the master. If you have enabled security via the *Configure Global Security* page, you can customize the port on which the Jenkins master will listen for incoming JNLP agent connections. By default, the JNLP agent will launch a GUI, but it's also possible to run a JNLP agent without a GUI, e.g. as a Window service.

Jenkins. Master-Agent model

JOHN BRYCE
Leading in IT Education
a matrix company

Agents Launch Methods

Jenkins. Master-Agent model

JOHN BRYCE
Leading in IT Education
a matrix company

Agents Launch Methods:

Launch agent via execution of command on the master

Starts an agent by having Jenkins execute a command from the master. Use this when the master is capable of remotely executing a process on another machine, e.g. via SSH or RSH.

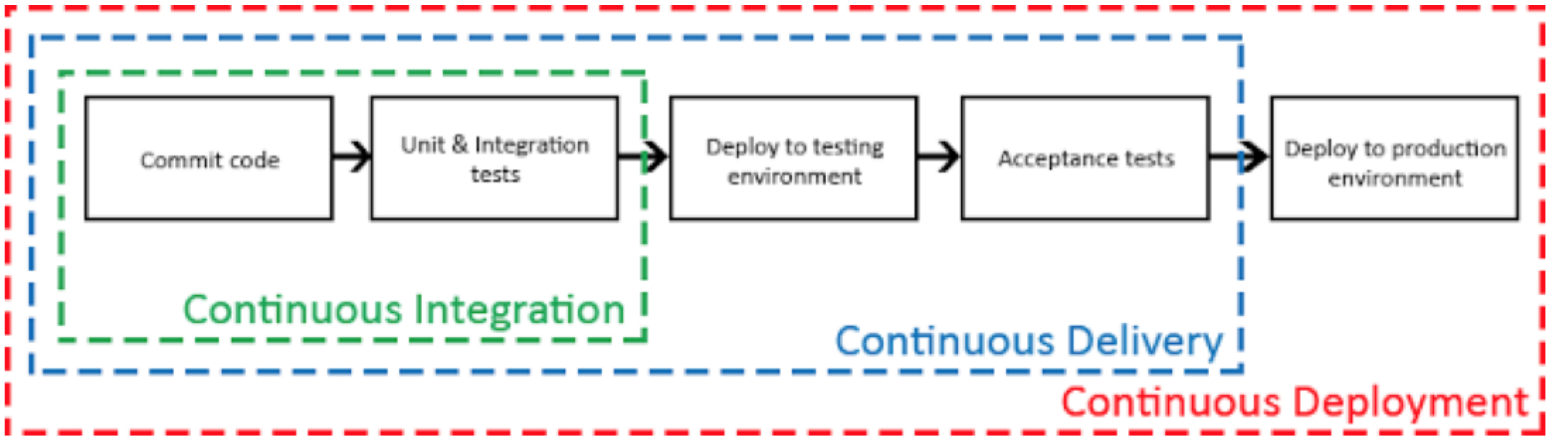
Launch slave agents via SSH

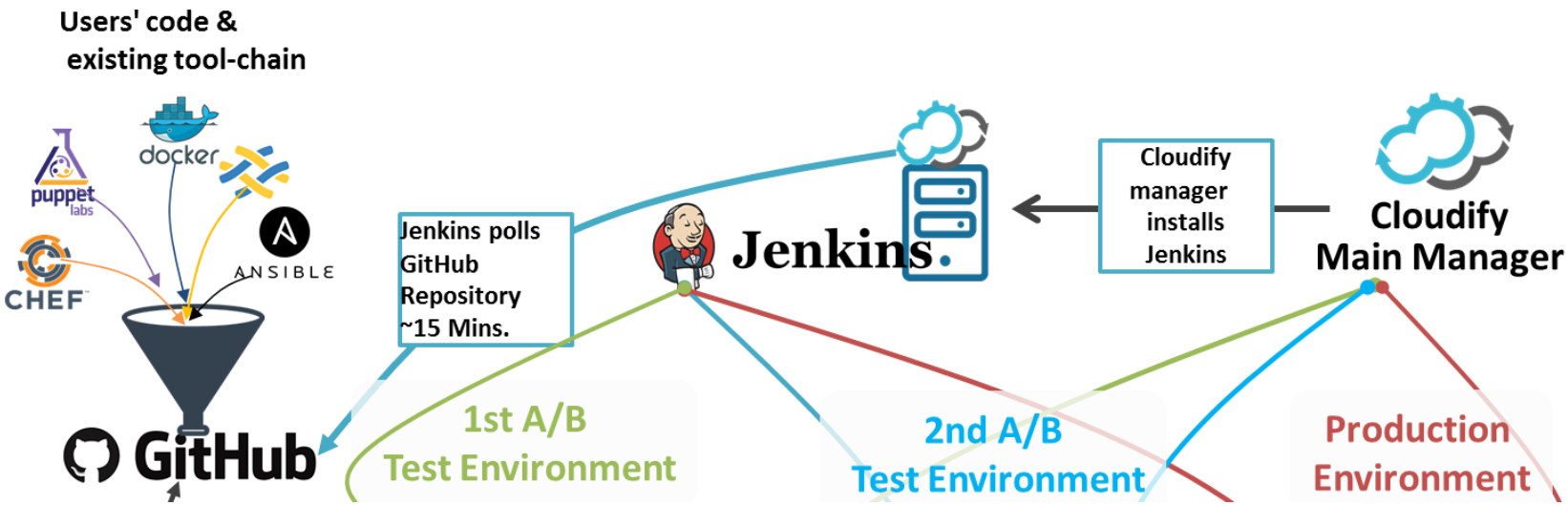
Starts a slave by sending commands over a secure SSH connection. The slave needs to be reachable from the master, and you will have to supply an account that can log in on the target machine. No root privileges are required.

Let Jenkins control this Windows slave as a Windows service

Starts a Windows slave by a remote management facility built into Windows. Suitable for managing Windows slaves. Slaves need to be IP reachable from the master.

Jenkins Continuous Delivery



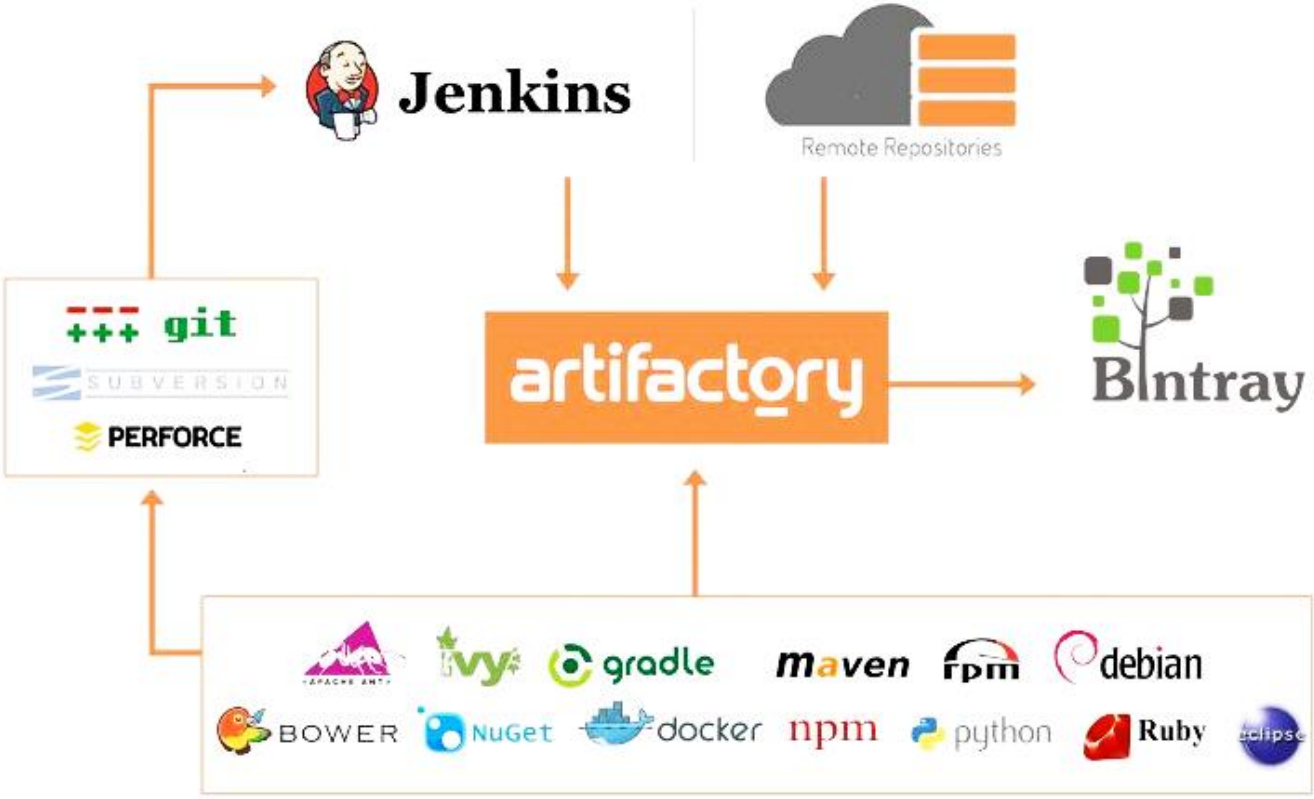


Steps to CD in Jenkins

- Ensuring reproducible builds
- Sharing build artifacts throughout the pipeline
- Choosing the right granularity for each job
- Parallelizing and joining jobs
- Gates and approvals
- Visualizing the pipeline
- Organizing and securing jobs
- Good practice: versioning your Jenkins configuration

Artifact repository

CI/CD stages supposed to have a storage where all the versioned product versions would be stored for the deployment ether whilst the “Deploy and testing environment” stage or “Deploy to production environment” stage:



Nexus artifact repository

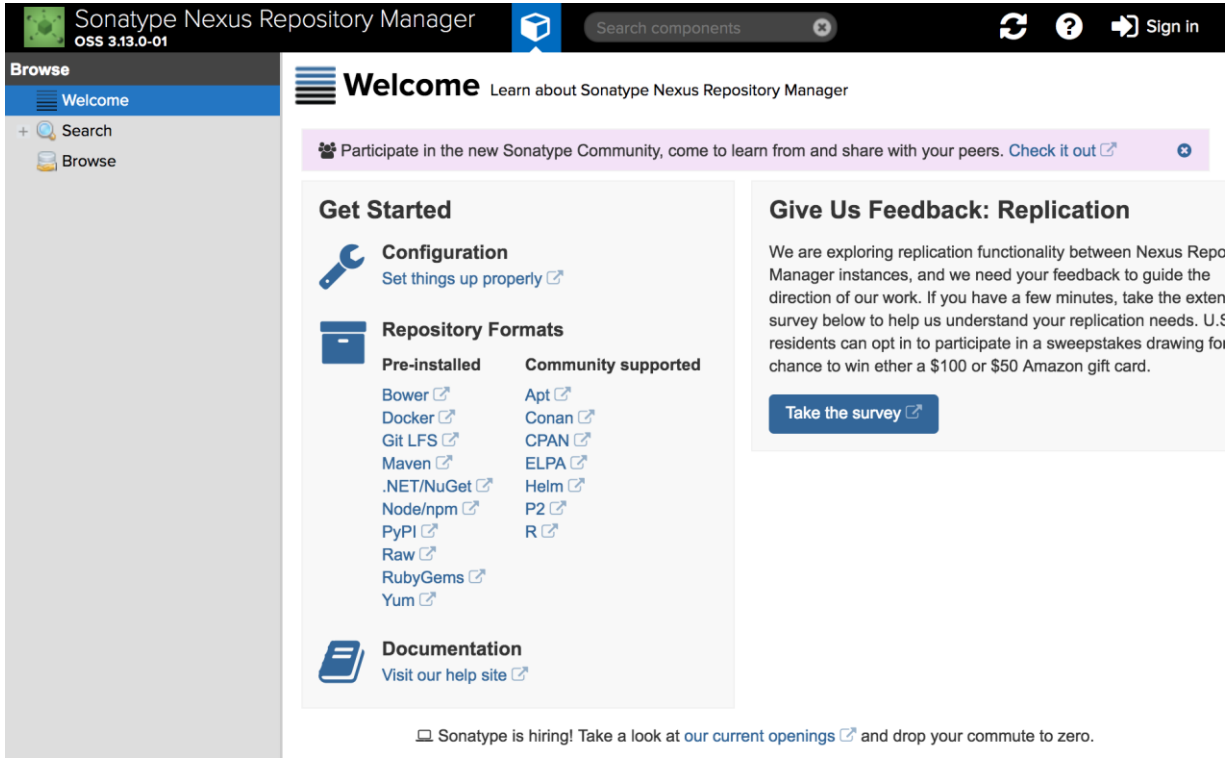
There are cases when a company couldn't afford to keep the artifacts for the delivery or deployment on the external cloud services. The security policies tend them to move the storages inside the secured network. Using a private antifactory managers may solve the issue.

Sonatype Nexus Repository is the installable repository manager with the wide list of features and sophisticated API to be integrated into your infrastructure:

- Manage components, build artifacts, and release candidates in one central location.
- Understand component security, license, and quality issues.
- Modernize software development with intelligent staging and release functionality.
- Scale DevOps delivery with high availability and active/active clustering.
- Sleep comfortably with world-class support and training.
- Store and distribute **Maven/Java, npm, NuGet, RubyGems, Docker, P2, OBR, APT** and **YUM** and more.
- Manage components from dev through delivery: binaries, containers, assemblies, and finished goods.
- Awesome support for the Java Virtual Machine (JVM) ecosystem, including Gradle, Ant, Maven, and Ivy.
- Integrated with popular tools like Eclipse, IntelliJ, Hudson, Jenkins, Puppet, Chef, Docker, and more.

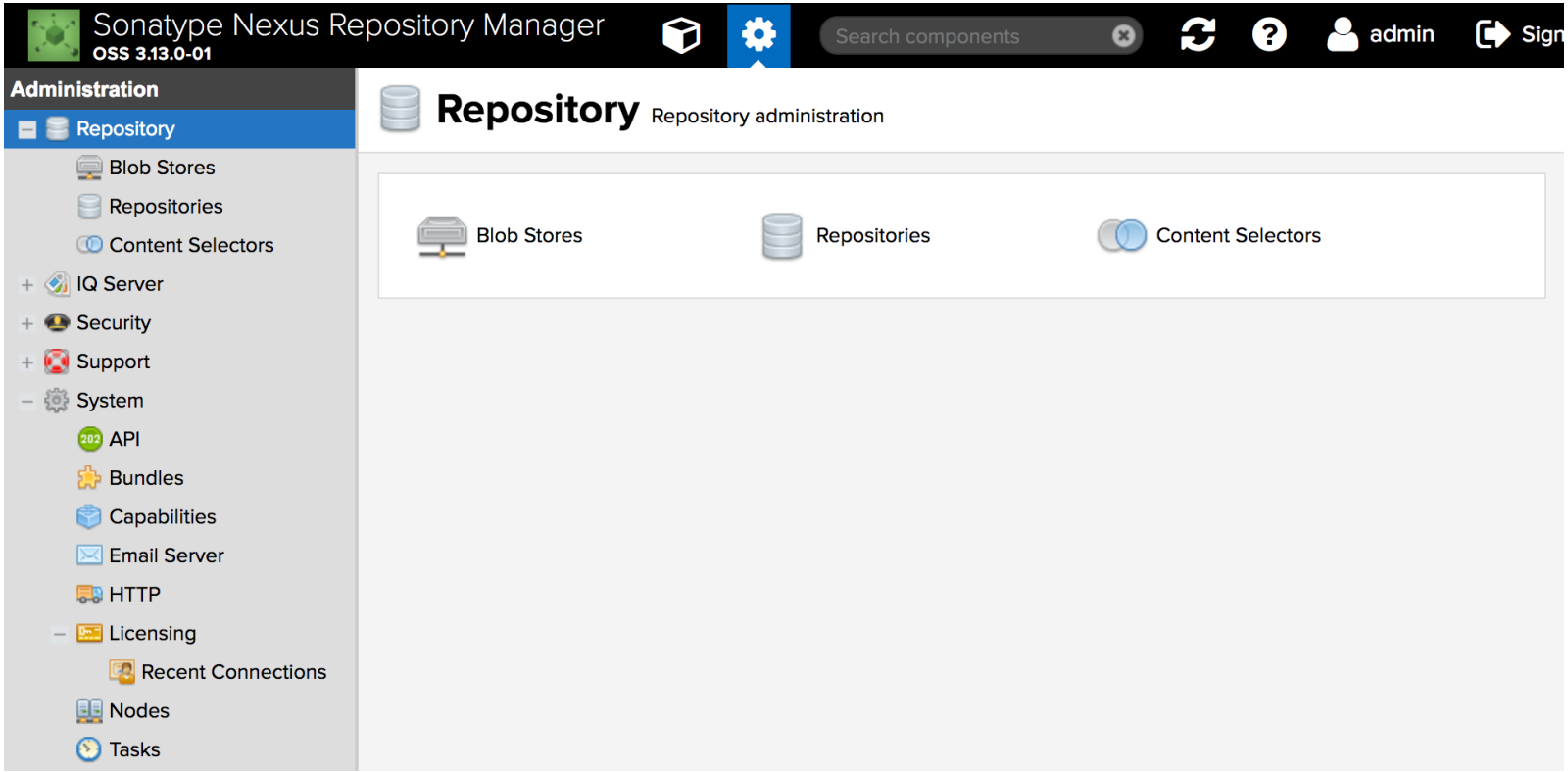
Nexus artifact repository

We have already installed the Sonatype Nexus to your course instance, to reach it please go to: http://<your course PC IP>:8081/



Nexus artifact repository

During the main pipeline exercise we would create the Docker hosted repository in the persisted Nexus for storing our artifacts.



JFrog artifactory

JFrog Artifactory is an artifact repository manager, which is entirely technology agnostic and fully supports software created in any language or using any tool.

JFrog gives you freedom of choice. You can run Artifactory on Amazon Web Service, Google Cloud Platform, or Microsoft Azure.

Select your cloud provider:

JFrog artifactory

The tool is designed to integrate with the majority of continuous integration and delivery tools, so as to provide an end to end automated solution for tracking artifacts from development to production.

JFrog Artifactory is intended for use both by developers and DevOps teams as a whole. It serves as a singles access point that organizes all of the resources and removes the associated complications. Simultaneously it enables the operations staff to efficiently manage the continual flow of code from each developer's machine to the organization's production environment.

Thanks!!!

JOHN BRYCE
Leading in IT Education
a matrix company

Next 10 Years

- User Experience
- Scalability and Availability
- Continuous Delivery