**How to use Jenkins pipeline to launch Selenium tests**

Jenkins is an open source Java software system designed to aid in the continuous software integration process. It is a server-based system that runs in servlet containers such as Apache Tomcat. Jenkins supports version control tools (for example – Git) and can execute your projects based on Apache Ant, Apache Maven, and sbt as well as arbitrary shell/Windows batch scripts and commands. During the development of large systems, there is a need for a continuous process of deployment and testing new versions of your product. To solve this problem, we will use the Jenkins Pipeline.

Jenkins Pipeline is a suite of plugins which support continuous delivery integration into Jenkins. In other words, it is a script that has several jobs in it and can be executed each time a certain condition is met. Let’s have a look at an example of working with Pipeline.

Our task is to create a Pipeline, which will be automatically launched with each push to our Maven public test repository. The repository will store a Maven project with a UI autotest, written using Selenium Webdriver.

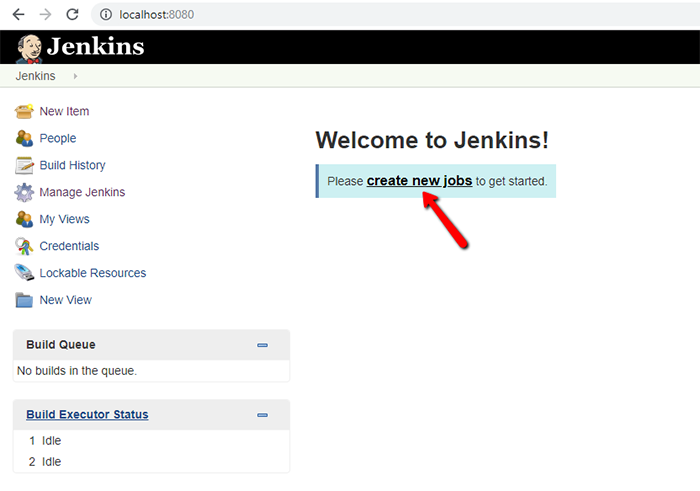
**Our Jenkins Pipeline will receive code changes, build a project, and run a simple autotest.**

In steps:

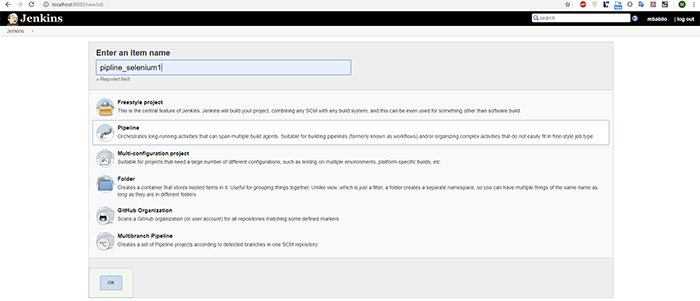
1) [Download](https://jenkins.io/download/) Jenkins.

2) Install with the recommended parameters following the installer. (Important note: it is recommended to install Jenkins on a separate machine with Internet access, it is necessary to connect your instance with the repository. However, for the demonstration, we can (and will in this article) use a local Jenkins project with access through the Ngrok tool).

3) When the installation is complete, click on the “create new jobs” button.

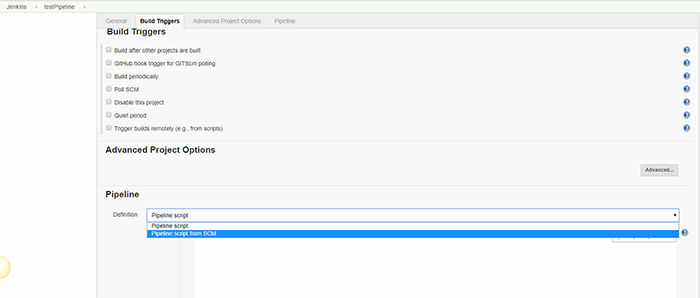


4) Then, select the Pipeline section and enter the name of your job. Click OK.

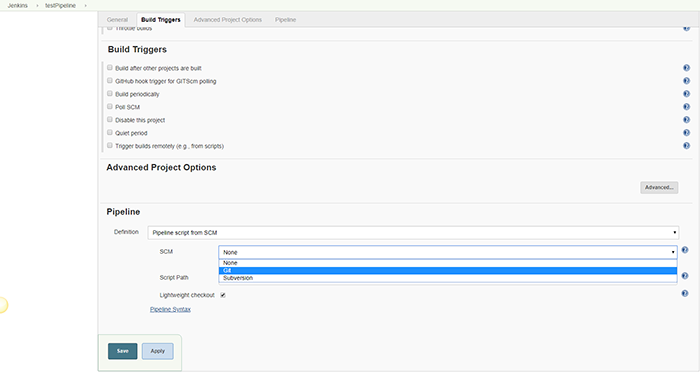


5) After that, go to the Pipeline section, and set the following parameters:

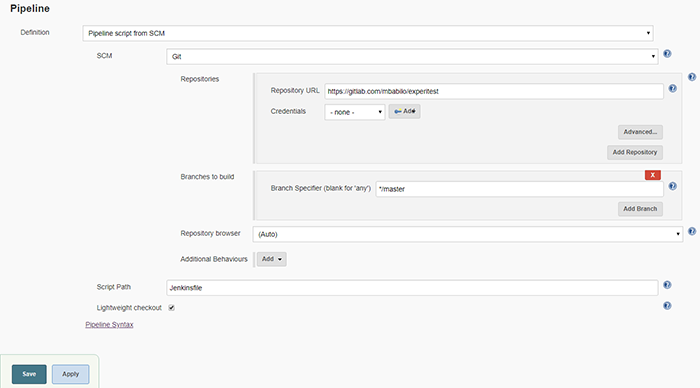
a) Definition = “pipeline script from SCM”



b) SCM = “Git”



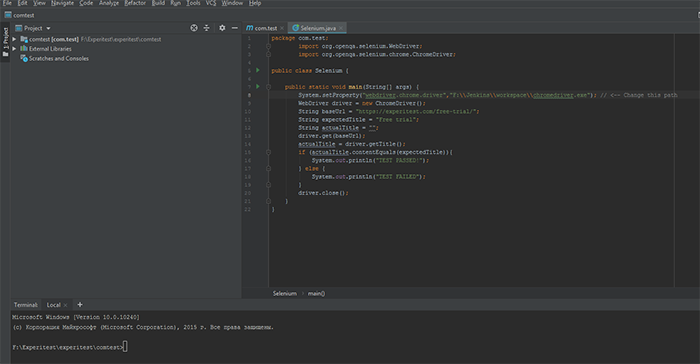
6) Set a repository. Here the Script Path field defaults to “Jenkinsfile” – this is the name of the file-script, which contains a description of all the tasks that our Jenkins Pipeline will perform. This file should be in the root of the repository by default. We’ll get back to creating and configuring Jenkinsfile later, so now we will prepare a repository and a project of our Selenium test. For now, just click the Save button.



7) The script of our autotest will visit the page <https://experitest.com/free-trial/> and check the title of this page, and then we will see “TEST PASSED” or “TEST FAILED” messages as a result. So, we will use [Chromedriver](http://chromedriver.chromium.org/downloads) and [IntelliJ IDEA](https://www.jetbrains.com/idea/download/#section=windows) IDE, please download and install them if needed.

8) Create a standard Maven project in your IDE, and place the chromedriver.exe where your Jenkins instance is installed. In my case, the code will look as follows then:

package com.test;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.chrome.ChromeDriver;  
public class Selenium {  
public static void main(String[] args) {  
System.setProperty(“webdriver.chrome.driver”,”F:\\Jenkins\\workspace\\chromedriver.exe”); // <– Change this path  
WebDriver driver = new ChromeDriver();  
String baseUrl = “https://experitest.com/free-trial/”;  
String expectedTitle = “Free trial”;  
String actualTitle = “”;  
driver.get(baseUrl);  
actualTitle = driver.getTitle();  
if (actualTitle.contentEquals(expectedTitle)){  
System.out.println(“TEST PASSED!”);  
} else {  
System.out.println(“TEST FAILED”);  
}  
driver.close();  
}  
}

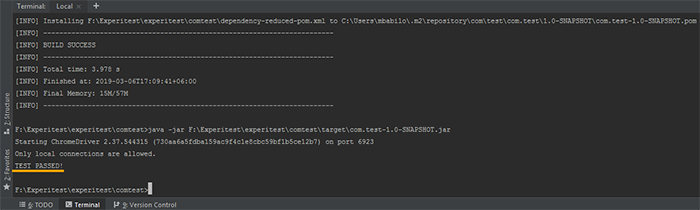


9) In order to make sure that our test works, we will launch it locally. Open the built-in terminal (you can find it in the lower right corner of the IDEA interface) and enter the following commands:

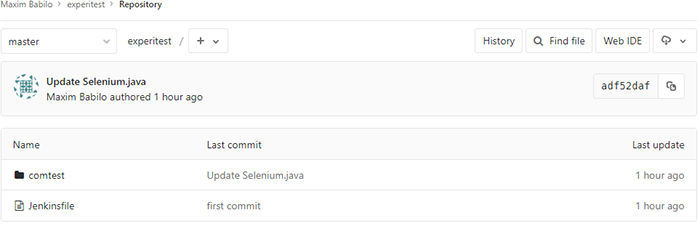
a) mvn clean install – As a result of this command, a folder /target will be created, and in it a file com.test-1.0-SNAPSHOT.jar.

b) java -jar {path to local project}\comtest\target\com.test-1.0-SNAPSHOT.jar – this command will launch our test.

See the results in the screenshot:



10) Now that our project is ready, we can put it in the root of our test repository. In the same place, we will create a Jenkinsfile. Like this:



11) Now we can discuss the contents of our Jenkinsfile:

node {  
stage (‘SCM checkout’){  
git “https://gitlab.com/mbabilo/experitest”  
}  
stage (‘Build’){  
dir(“comtest”) {  
sh “mvn clean install”  
}  
dir(“comtest/target”) {  
sh “java -jar com.test-1.0-SNAPSHOT.jar”  
}  
}  
}

Here:

a) node { } – some kind of wrapper for our script. Inside it, the script is divided into stages.

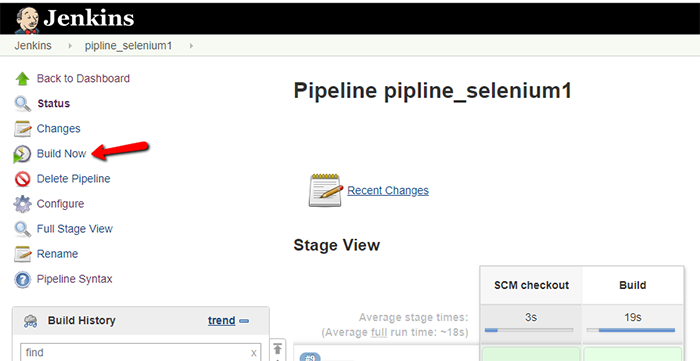
b) stage (‘SCM checkout’) – preparatory stage, during which we receive updates from our repository.

c) stage (‘Build’) – the main stage, during which Jenkins will perform the same steps that we performed locally from the console above.

d) dir (“comtest”) – works in a specific folder.

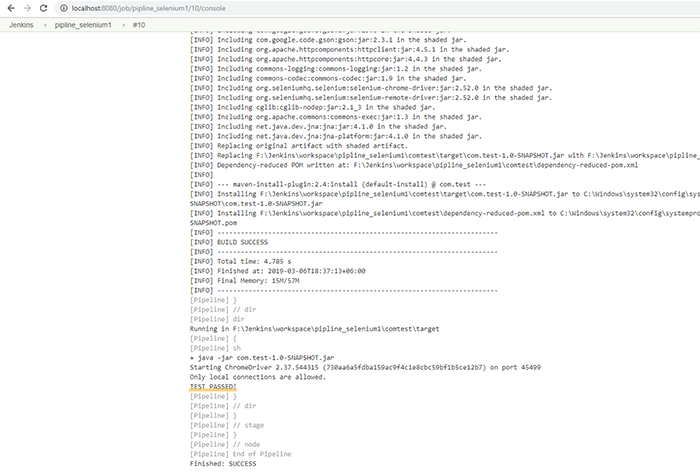
e) sh “mvn clean install” – executes shell commands.

12) At this step we can manually launch our Jenkins Pipeline, to make sure it works.

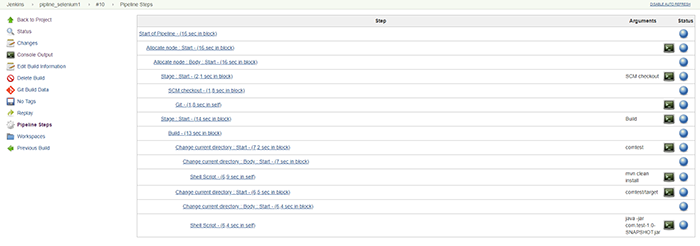


You can see the results of the work of the pipeline in the form of a solid text, or step by step.

“Console Output” option:

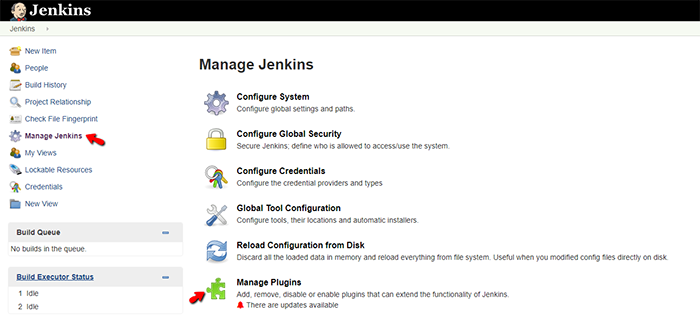


“Pipeline steps” option:

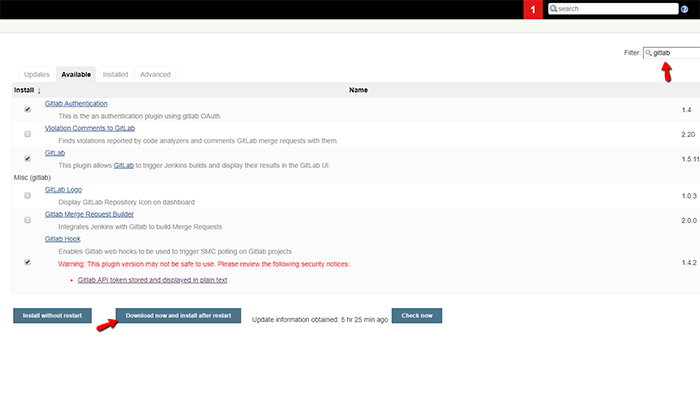


But our goal is not just to make a Jenkins Pipeline for autotests, but also to make its launch automatic. Let’s do it.

13) Install the plugins for integration with the GitLab. To do this, go to the next section: Jenkins=>Manage Jenkins=>Manage Plugins

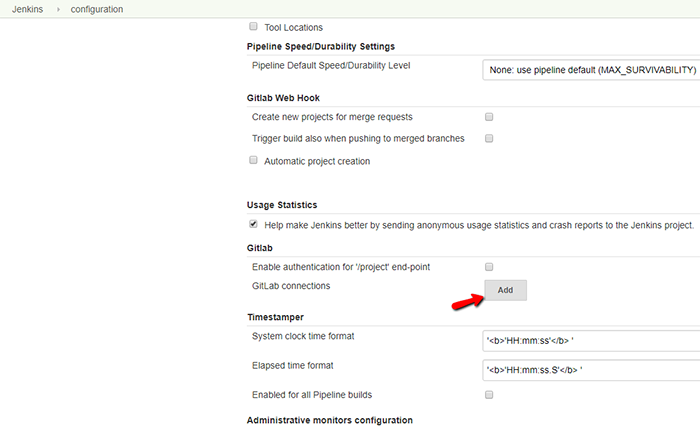


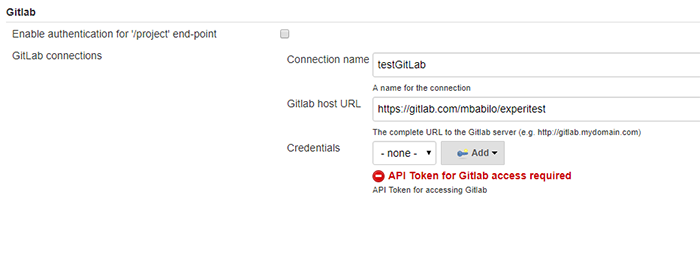
Then select the Available tab, and enter “gitlab” in the search field. Install the necessary plugins and restart Jenkins.



14) When the update is done, we return to the configuring. Go to Jenkins=>Configure System.

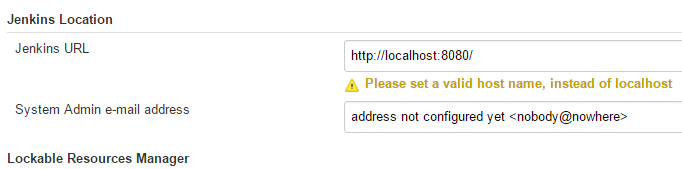
a) Add Gitlab repository to the plugin:





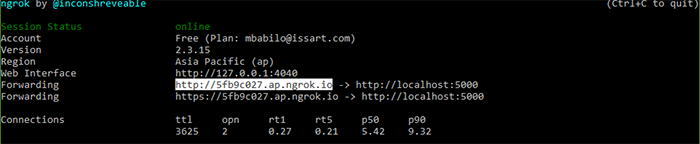
**Note: For our purposes, it is necessary to use a public repository, so we shouldn’t specify any credentials.**

b) Check if we have a valid Jenkins URL. If you installed your instance of Jenkins Pipeline on a separate machine, then this step should be skipped. If you used a local machine as I did, then you will see the inscription localhost in the “Jenkins URL” field.

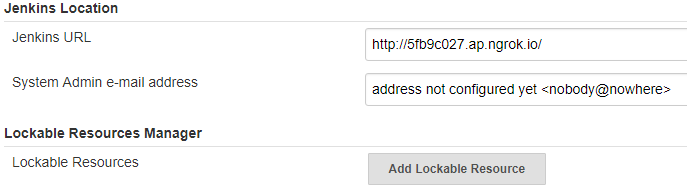


In this case, we need to get a URL, which can be accessed by GitLab via the Internet. As I said earlier, there is an easy way to do this – [ngrok](https://ngrok.com/download).

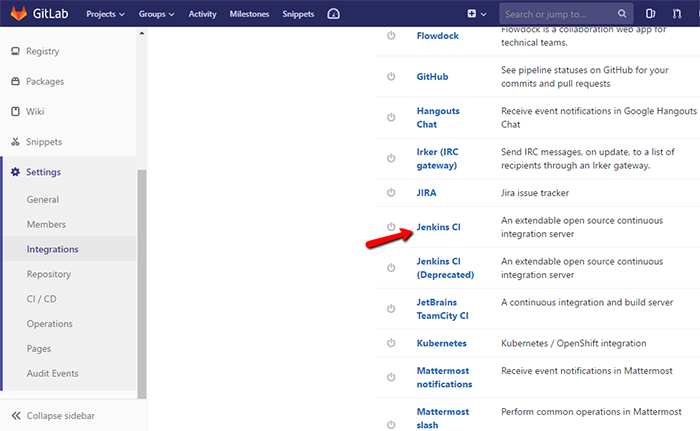
Following the simple instructions of this tool, get the URL on which our test Jenkins is located:



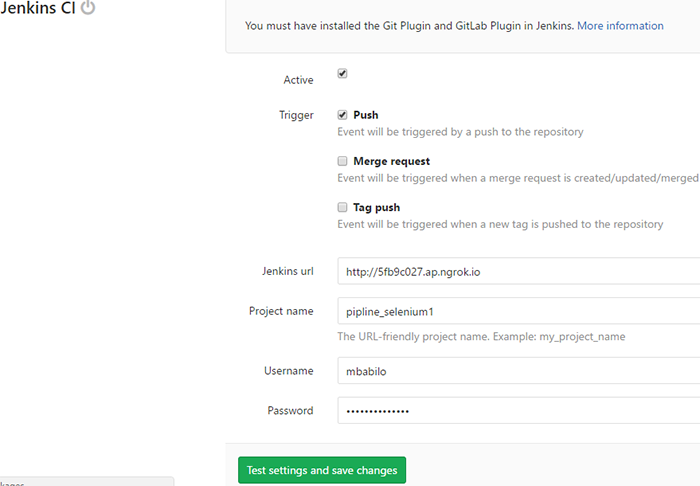
We will put this address in the field Jenkins URL, and save all the changes.



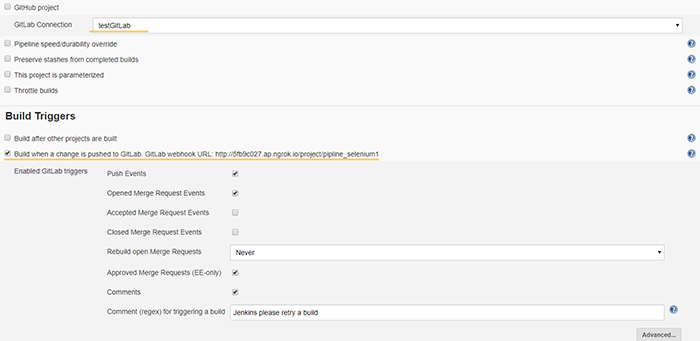
15) Now, go to the GitLab settings: GitLab=>project=>Settings=>Integrations. So we need to configure Jenkins CI webhook. Click the next link:



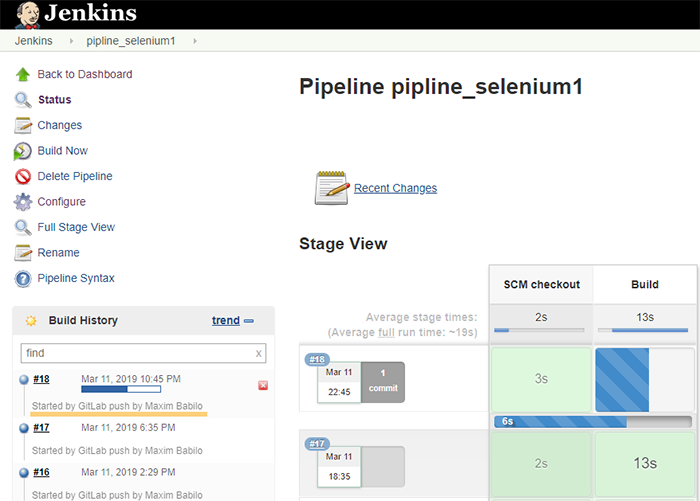
The integration settings with Jenkins CI will look as follows or similar:



16) Update the settings of our Jenkins Pipeline (pipeline\_selenium1) to use a GitLab webhook.



17) That’s all! Now let’s push something to our repository, and see what happens in Jenkins.



**Wrapping up the Jenkins pipeline launch**

As you can see, we have an easy way to create the Jenkins Pipeline, which will run Selenium autotests without human intervention. The Pipeline also allows us to perform any preparatory actions automatically. To learn more about the Jenkinsfile syntax, you can follow the link of Pipeline Syntax (it can be seen on the last screenshot).