

Continuous integration-

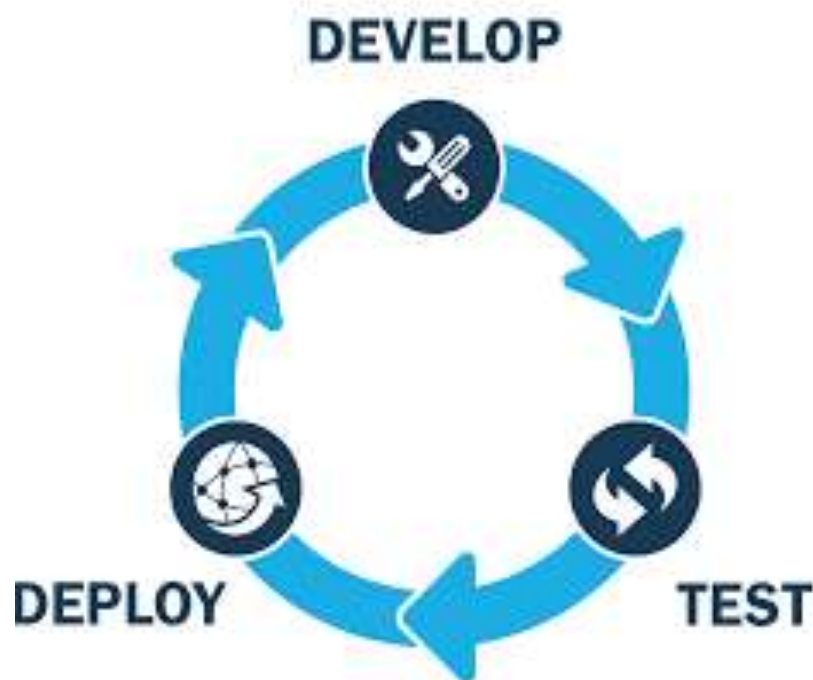
Continuous Integration (CI) is a development practice where developers integrate code into a shared repository frequently, preferably several times a day.

Each integration can then be verified by an automated build and automated tests.

One of the key benefits of integrating regularly is that you can detect errors quickly and locate them more easily.

As each change introduced is typically small, pinpointing the specific change that introduced a defect can be done quickly.

In recent years CI has become a best practice for software development and is guided by a set of key principles. Among them are revision control, build automation and automated testing.



Jenkins-

Jenkins is an open source automation server written in Java.

Jenkins helps to automate the non-human part of software development process, with continuous integration and facilitating technical aspects of continuous delivery.

It is a server-based system that runs in servlet containers such as Apache Tomcat. It supports version control tools, including AccuRev, CVS, Subversion, Git, Mercurial, Perforce, ClearCase and RTC, and can execute Apache Ant, Apache Maven and sbt based projects as well as arbitrary shell scripts and Windows batch commands.

Jenkins is under the MIT License, Jenkins is a free software.

Builds can be triggered by various means, for example by commit in a version control system, by scheduling via a cron-like mechanism and by requesting a specific build URL. It can also be triggered after the other builds in the queue have completed.

Jenkins functionality can be extended with plugins.

<https://wiki.jenkins.io/display/JENKINS/Home>

System Requirements

JDK	JDK 1.5 or above
Memory	2 GB RAM (recommended)
Disk Space	No minimum requirement. Note that since all builds will be stored on the Jenkins machines, it has to be ensured that sufficient disk space is available for build storage.
Operating System Version	Jenkins can be installed on Windows, Ubuntu/Debian, Red Hat/Fedora/CentOS, Mac OS X, openSUSE, FreeBSD, OpenBSD, Gentoo.
Java Container	The WAR file can be run in any container that supports Servlet 2.4/JSP 2.0 or later.(An example is Tomcat 5).

Build jobs

Build jobs are at the heart of the Jenkins build process.

Simply put, you can think of a Jenkins build job as a particular task or step in your build process.

This may involve simply compiling your source code and running your unit tests. Or you might want a build job to do other related tasks, such as running your integration tests, measuring code coverage or code quality metrics, generating technical documentation, or even deploying your application to a web server.

A real project usually requires many separate but related build jobs.

Jenkins first setup

1. Download Jenkins.war
<https://updates.jenkins-ci.org/download/war/>
2. Open CMD/terminal change directory to one containing Jenkins.war file (**cd c://users...**)
3. Type "**java -jar jenkins.war**"

Wait for "Jenkins is fully running"..

4. Open your browser on localhost:8080 (this is the default port by used by Jenkins).
5. Follow the instructions to use your secret key.
6. Follow the instructions and install default configurations and plugins.

First Build Job

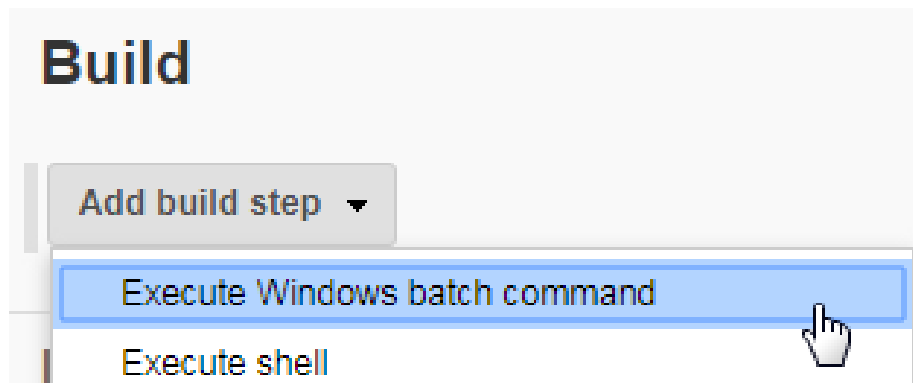
1. In main screen choose new:



2. Give it a name and choose "Freestyle" project:

The image shows the "Enter an item name" dialog box in Jenkins. The title bar says "Enter an item name". Below the title bar is a text input field containing the text "new test", which is highlighted with a red-bordered box. Below the input field is a small text label that says "Required field". Below the input field is a list of project types, each with an icon and a description. The first item, "Freestyle project", is highlighted with a red-bordered box. The other items are "Pipeline", "Multi-configuration project", "Folder", "GitHub Organization", and "Multibranch Pipeline". Below the list of project types is a section titled "if you want to create a new item from other exists". Below this section is a "Copy from" button and a text input field labeled "Type to autocomplete:". At the bottom of the dialog box is a blue "OK" button, which is also highlighted with a red-bordered box.

3. For now we are going to ignore "**Source code management**", "**Build triggers**" and "**Build environment**".
4. Go to Build and choose "**Add build step**"



Choose "**Execute windows command**" option

5. Inside the box type: **time /T**
6. Press save.
7. Your build job is now ready.
8. Press "**Build Now**" Button.



9. Once the build will be done, you will see your run under builds history:

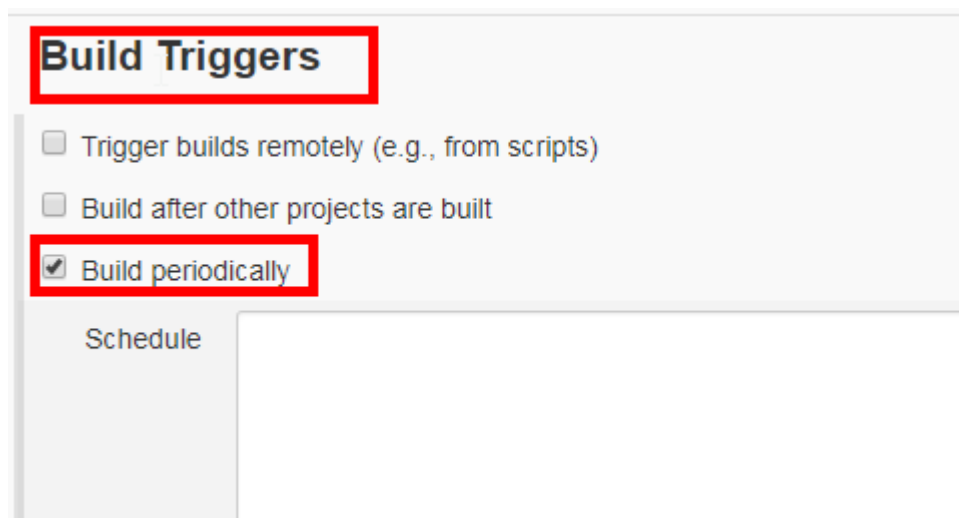


Scheduling

1. In order to schedule your build job:
 - Enter your build job
 - Press on settings



2. Under "**Build triggers**" choose "**Build periodically**"



3. To schedule your build job, Jenkins is using a "**CRON**" expressions

For example to run it every hour write:

H * * * *

CRON expression

A CRON expression is a string comprising five or six fields separated by white space that represents a set of times, normally as a schedule to execute some routine.

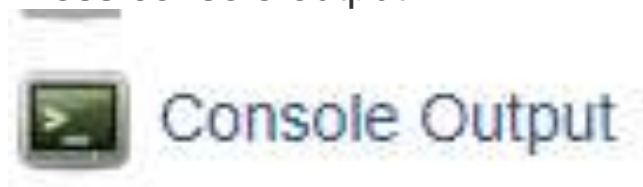
Comments begin with a comment mark #, and must be on a line by themselves.

Field	Required	Allowed values	Allowed special characters	Remarks
Minutes	Yes	0-59	* , -	
Hours	Yes	0-23	* , -	
Day of month	Yes	1-31	* , - ? L W	? L W only in some implementations
Month	Yes	1-12 or JAN-DEC	* , -	

Day of week	Yes	0-6 or SUN-SAT	* , - ? L #	? L W only in some implementations
Year	No	1970- 2099	* , -	This field is not supported in standard/default implementations.

Logs

1. Enter your recent run by pressing on it (clause 9)
2. Press console output:



Now you can see your result and output

```

Started by user Daniel
Building in workspace C:\Users\daniel.gotlieb\.jenkins\workspace\new test
[new test] $ cmd /c call C:\Users\DANIEL~1.GOT\AppData\Local\Temp\jenkins3238524300451769344.bat

C:\Users\daniel.gotlieb\.jenkins\workspace\new test>time /T
05:08 PM

C:\Users\daniel.gotlieb\.jenkins\workspace\new test>exit 0
Finished: SUCCESS

```

Reporting

1. To get job failed/success reporting we will need to use a plugin named "**Email Extension Plugin**".
2. Go into "**Manage Jenkins**" → "**Configure system**"



Configure System

Configure global settings and paths.

3. Scroll down until you see "E-mail Notification" and press on **advanced** button, fill up the information

E-mail Notification

SMTP server: smtp.gmail.com

Default user e-mail suffix:

☒ Use SMTP Authentication

User Name: m@gmail.com

Password:

☒ Use SSL

SMTP Port: 465

Reply-To Address:

Charset: UTF-8

- User name is the email address
- Password (of email)

For **Gmail** use:

1. SMTPserver: smtp.gmail.com
2. Mark "**Use SSL**" checkbox
3. **Smtp port: 465**

**** IMPORTANT NOTE:**

This will only work if "**Less secured apps**" is configured in your Gmail account:

<https://myaccount.google.com/u/1/security>

Allow less secure apps: ON



Master slave

- **Overview**

- This single Jenkins server was not enough to meet certain requirements
- Sometimes you might need several different environments to test your builds. This cannot be done by a single Jenkins server.
-

- **Jenkins Distributed Architecture**

- Jenkins uses a Master-Slave architecture to manage distributed builds. In this architecture, Master and Slave communicate through TCP/IP protocol.

- **Jenkins Master**

- Your main Jenkins server is the Master. The Master's job is to handle:
- Scheduling build jobs.
- Dispatching builds to the slaves for the actual execution.
- Monitor the slaves (possibly taking them online and offline as required).
- Recording and presenting the build results.
- A Master instance of Jenkins can also execute build jobs directly.

- **Jenkins Slave (build machine)**

- A Slave is a Java executable that runs on a remote machine. Following are the characteristics of Jenkins Slaves:
- It hears requests from the Jenkins Master instance.
- Slaves can run on a variety of operating systems.

- The job of a Slave is to do as they are told to, which involves executing build jobs dispatched by the Master.
- On your **master** machine go to **Manage Jenkins > Manage Nodes**.



Manage Nodes

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

- **New Node**
 - **Enter Node Name**
 - **Select Permanent Agent**
 - Press **OK**.

Node name


- ☐ **Permanent Agent**
- Fill out the following:
- Set a **number of executors**
 - (one or more) as needed.
- Set a **Remote FS (file system) Root**
 - a home directory for the master on the slave machine.
 - For a *Windows slave*, use something like:
"C:\Jenkins\"
- Select the appropriate **Usage** setting:
 - For an additional worker: *Utilize this slave as much as possible*
 - For specialized jobs: *Leave this machine for tied jobs only*
- **Launch Method:**

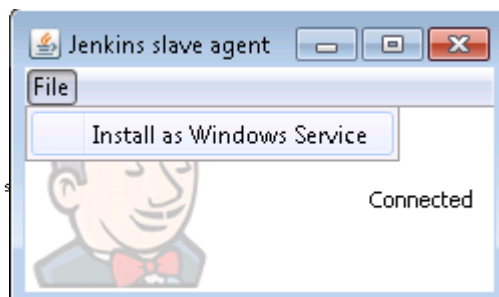
- An easy way to control a Windows slave is by using *Launch slave agents via Java Web Start* (Recommended for Windows)
- **Availability**
 - *Keep this slave online as much as possible*
- Press **OK**.
- Now you need to connect your slave machine to the master using the following steps.
- Open a browser on the **slave machine** and go to the **Jenkins master server** url (<http://yourjenkinsmaster:8080>).
- **In your slave machine** Go to **Manage Jenkins > Manage Nodes**,
 - Click on the newly created slave machine. You will need to login as someone that has the "Connect" Slave permission if you have configured global security.
- Click on the **Launch** button to launch agent from browser on slave, run the program and wait for it to show connected.



Slave Jenkins Slave

Connect slave to Jenkins one of these ways:

-  **Launch** Launch agent from browser on s
- Run from slave command line



- Now your build machine is ready for work
- To run your tests on your slave (build) machine enter any build job settings → check “**Restrict where this project can be run**” and enter your slave name and save
- You can test it by pressing “**Build now**” on your build job.

<https://wiki.jenkins.io/display/JENKINS/Step+by+step+guide+to+set+up+master+and+slave+machines+on+Windows>

Pipeline

Overview

- Jenkins Pipeline is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.
- A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers.
- Every change to your goes through a complex process on its way to being released.
- This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment.
- While a JOB is a defined process, and a BUILD is the result of that JOB being carried out, a PIPELINE is a defined series of JOBS that can be interrupted in between processes by different events such as failed tests, approval, et. al.
- Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code".
- Pipelines are Jenkins jobs enabled by the Pipeline plugin and built with simple text scripts that use a Pipeline DSL (domain-specific language) based on the **Groovy** programming language.
- 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 more.

- Pipeline can be written using two types of syntax - Declarative and Scripted.
- Declarative and Scripted Pipelines are constructed differently.
- Declarative Pipeline is a more recent feature of Jenkins which provides richer syntactical features over Scripted Pipeline syntax, and is designed to make writing and reading Pipeline code easier.
- In Declarative Pipeline syntax, the pipeline block defines all the work done throughout your entire Pipeline.
- In Scripted Pipeline syntax, one or more node blocks does the core work throughout the entire Pipeline. Although this is not a mandatory requirement of Scripted Pipeline syntax, confining your Pipeline's work inside of a node block does two things:
 - Schedules the steps contained within the block to run by adding an item to the Jenkins queue.
 - As soon as an executor is free on a node, the steps will run.
- Creates a workspace (a directory specific to that particular Pipeline) where work can be done on files checked out from source control

```
node {
    stage('Build') {
        //
    }
    stage('Test') {
        //
    }
    stage('Deploy') {
        //
    }
}
```

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                //
            }
        }
        stage('Test') {
            steps {
                //
            }
        }
        stage('Deploy') {
            steps {
                //
            }
        }
    }
}
```


Methods

- **echo** – used to print anything inside

echo 'hello world'

- Text has to be inside Apostrophe

- **sh / bat** - used to perform shell (OS X / Ubuntu) / bat (Windows) scripts:

Print **sh 'date'** the **OR bat 'date'** - which will current date.

- This will produce the same result as opening Terminal / CMD and typing **date**
- Script has to be inside Apostrophe
- **git** – used to perform a clone from the specified repository.

git 'https://github.com/jenkinsci/jenkins.git'

Syntax

- We can use declare variables in our pipeline so those can be used anywhere inside our pipeline.
- To use it in a scripted we will use the word **def** where in declarative we will wrap the variables using **environment**

```
{ } node {  
  def x = "456"  
  stage('Build') {  
    echo x  
  }  
}
```

```
node {  
  stage('Build') {  
    try{  
      echo 1/0  
    }catch(e){  
      echo 'divison by 0'  
    }  
  }  
}
```

```
pipeline {  
  agent any  
  environment {  
    MY_VAR = '123'  
  }  
  stages {  
    stage('Build') {  
      steps {  
        echo MY_VAR  
      }  
    }  
  }  
}
```

- try-catch mechanism can be used when we try to perform an action, and in case of a failure, we will have a “fallback” inside the **catch** block: