Jenkins

**Official Documentation**

[**https://jenkins.io/**](https://jenkins.io/)

<https://jenkins.io/doc/pipeline/tour/getting-started/>

<https://plugins.jenkins.io/>

**Introduction**

* Jenkins is the java application
* Is a free and opensource web application for Continuous build, integration, testing and deployment over a web server
* It is installed on a server where the central build will take place
* **Workflow:**
  1. Developers check their source code
  2. Jenkins will pick up the changed source code, trigger a build and run any tests if required
  3. The build output will be available in the Jenkins dashboard and automatic notifications can also be sent back to the users (developers).
* **System Requirements:**
  + JDK - JDK 1.5 or above
  + Memory - 2GB RAM (Recommended)
  + Disk Space - No minimum requirement. Note that since all builds will be stored on the Jenkins machines, it must be ensured that sufficient disk space is available for build storage.
  + OS - It’s a platform independent. Jenkins can be installed on Windows, Ubuntu/Debian, RedHat/Fedora/CentOS, MacOS 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).

**Download:**

* <https://jenkins.io/>
* Download Jenkins from LTS (Long Term Support) for the required OS

**Installation on Windows**

There are two ways in which you can set up Jenkins on Windows. One is by using the Jenkins native package for Windows, and the other is through the **jenkins.war** file.

Using native package for windows:

* Download Jenkins.zip file
* Extract it and install using “.msi” file
* Once the installation is done, the Jenkins console will get opened in the browser using <http://localhost:8080/>
* Install suggested plugins
* Create First Admin User
* Get started with Jenkins 😊

Using Jenkins.war: (Stand-alone installation)

* Download “Generic Java package (.war) file from - <http://mirrors.jenkins.io/war-stable/latest/jenkins.war>
* Place the war file into any location on the system
* Open command prompt and run the below command:

cd <location to Jenkins war file>

Java -jar jenkins.war

* Open up browser and access Jenkins using <http://localhost:8080>
* Install suggested plugins
* Create First Admin User
* Get started with Jenkins

**Jenkins Home Directory**

Jenkins home directory contains:

1. All Configurations
2. Plugins
3. Jobs details
4. Logs

Why do we change home directory?

* To move Jenkins home directory to a location that has enough disk space
* Project requirements

How to change Home Directory

1. Check the Jenkins current home directory
   1. Login to the Jenkins console
   2. Go to Manage Jenkins 🡪 Configure System
      1. Here we can find “Home directory”
2. Create a new folder (which will be the new home directory)
3. Copy all the data from old directory to new directory
4. Change the environment variable i.e., JENKINS\_HOME = new home directory path
5. Restart Jenkins

**Installation on AWS**

1. Launch an EC2 Instance (Amazon Linux)
2. <https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-16-04> for Ubuntu
3. <http://bhargavamin.com/how-to-do/install-jenkins-on-amazon-linux-aws/> for Amazon linux

**Setup Jenkins on Application Servers**

Why do we deploy Jenkins on Application Servers?

* We can run Jenkins server in one of two ways:
  1. Stand-alone application
  2. Within Application Server or Java Servlet Container

1. **Stand-alone Application:**
   * Jenkins comes bundled as a WAR file that you can run directly using an embedded servlet container
   * Jenkins uses the built-in java servlet container/application server (Jetty/Winstone)
   * Here we don’t need to configure any webserver
   * It is also a very flexible option, and provides some extra features unavailable if you deploy Jenkins to a conventional application server.
   * In particular, if you are running Jenkins as a stand-alone server, you will be able to install plugins and upgrades on the fly, and restart Jenkins directly from the administration screens.
2. **Within Application Server or Java Servlet Container**
   * Jenkins is available on the following servlet containers
     1. Tomcat
     2. Apache Geronimo 3.0
     3. Glassfish
     4. IBM WebSphere
     5. JBoss
     6. Jetty
     7. WebLogic
     8. Jonas
     9. Liberty Profile

**Jenkins – Tomcat Setup:**

**Java setup:**

* Verify the Java installation using the below command:

\>java -version

java version "1.8.0\_111"

Java(TM) SE Runtime Environment (build 1.8.0\_111-b14)

Java HotSpot(TM) Client VM (build 25.111-b14, mixed mode)

* Set the JAVA\_HOME environment variable to point to the base directory location where Java is installed on your machine.

Example: JAVA\_HOME = C:\Program Files (x86)\Java\jdk1.8.0\_111

* Append the full path of Java compiler location to the system path:

Example: C:\Program Files (x86)\Java\jdk1.8.0\_111\bin

**Download Tomcat**

* Download Tomcat from - <http://www-us.apache.org/dist/tomcat/tomcat-9/v9.0.7/bin/apache-tomcat-9.0.7-windows-x64.zip>
* Extract the zip file apache-tomcat-9.0.7-windows-x64.zip
* Copy the extracted file to the path C:\
* Copy the Jenkins.war file to C:\apache-tomcat-9.0.7\webapps\
* Open the command prompt and start the tomcat server using the below commands:

\> cd C:\apache-tomcat-9.0.7\bin\

C:\apache-tomcat-9.0.7\bin\> startup.bat

C:\apache-tomcat-9.0.7\bin\> shutdown.bat (To stop the tomcat server)

* Once the process is completed we could see the server startup info message in the console
* Access the Jenkins from the browser using – <http://localhost:8080/jenkins>
* Do the initial setup and get started with Jenkins 😊

**Jenkins – Git Setup**

* Go to Jenkins Dashboard
* Click the “Manage Jenkins” in the left-hand side
* Click the “Manage Plugins” from the list
* Click the “Available” tab and filter with the keyword “Git Plugin”
* Check the “Git Plugin” option from the list and click “Install without restart”
* Now, the GIT is available in the Source code management section for the job creation

**Jenkins Command Line Interface (CLI)**

The reasons for using Jenkins command line are:

* Easier
* Faster
* Memory management
* Continuous Integration

How to use

1. Start Jenkins
2. Go to Jenkins console 🡪 Manage Jenkins 🡪 Configure Global Security
   1. Enable Security checkbox if it’s not enabled
   2. Save
3. Go to Manage Jenkins 🡪 Jenkins CLI
   1. Download Jenkins-cli.jar and place at any location in the system
   2. Go to the folder where Jenkins CLI file is placed and execute the following:
      1. java -jar jenkins-cli.jar -s http://localhost:8080/ help

**Jenkins User Management**

1. **Create Users:**
   1. Start Jenkins on default port (8080)
   2. Login to the Jenkins console
   3. Go to Manage Jenkins 🡪 Manage User
      1. Create User and fill the required details
      2. Logout from the current user
      3. Login with the new created user credentials
2. **Manage Users:**
   1. Login to the Jenkins console
   2. Click on the user from the top right and select “configure”
      1. Add/Modify the details which are required
3. **Create and Manage Roles:**
   1. Login to the Jenkins console as admin user
   2. Download and install Role Strategy Plugin
   3. Restart the Jenkins
   4. Go to Manage Jenkins 🡪 Configure Global Security
      1. Make sure that “Enable Security” is checked
      2. Under Authorization section, select Role-Based Strategy
      3. Apply and Save
   5. Now try to login with the non-admin user
      1. It should give “Access Denied” error
   6. Login as admin user to the Jenkins console
   7. Go to Manage Jenkins 🡪 Manage and Assign Roles 🡪 Manage Roles
      1. There are 3 different roles will be listed. They are:
         1. Global Roles - Global access
         2. Project Roles – Project Specific
         3. Slave Roles
      2. Under Global Roles, create a role and give required access
         1. Ex: Employee – Has Read Access
      3. Under Project Roles, Create a role and Pattern
         1. Ex: Role to add = Developer
         2. Pattern = Dev.\*
      4. Apply and Save
   8. Go to Manage Jenkins 🡪 Manage and Assign Roles 🡪 Assign Roles
      1. Assign required roles to the non-admin users (Note that the project roles also should be added to the users to give control on the project)
      2. Apply and Save
4. **Test the Roles:**
   1. Login to the Jenkins console
   2. Add two different projects as follows:
      1. Click on “New Item” in the left on Jenkins console
         1. DevelopProject1
         2. TestProject1
      2. The newly created projects will be displayed in the dashboard
   3. Logout from the Jenkins console as admin
   4. Login as non-admin users to the Jenkins console and check the access to the projects that are provided.

**Jenkins Configuration:**

Login to the Jenkins console, go to Manage Jenkins 🡪 Configure System

1. Home Directory = Explained in the above sections
2. System Message = This message will be displayed at the top of the [Jenkins main page](http://localhost:8080/). This can be useful for posting notifications to the users. HTML can be used by changing “Markup Formatter” value to “safe HTML” in Configure Global Security section.
3. # of executors = The total number of concurrent job executions that can take place on the Jenkins machine
4. Labels = Labels of the Nodes for the distributed builds
5. Environment Variables = This is used to add custom environment variables to all the jobs. These are key-value pairs and can be accessed and used in Builds wherever required.
6. Jenkins URL = By default, the Jenkins URL points to localhost. If you have a domain name setup for your machine, set this to the domain name else overwrite localhost with IP of machine.
7. Email Notifications = In the email Notification area, you can configure the SMTP settings for sending out emails. This is required for Jenkins to connect to the SMTP mail server and send out emails to the recipient list.

**Set Up Build Jobs**

Login to the Jenkins console and click on “New Item” from the left side.

1. Enter an item name = The name of the project or job.
2. Select “Free style project” from the list
3. Click on “OK’, It will navigate to job configuration window. Fill in the following information:
   1. General:
      1. Project Name = job/project name
      2. Description = Description of the project/job
   2. Source Code Management:
      1. None
      2. Git = Git repository details need to be added
      3. Subversion = Subversion repository details need to be added
   3. Build Triggers:
      1. Trigger builds remotely = Enable this option if you would like to trigger new builds by accessing a special predefined URL (convenient for scripts).
         1. Example: JENKINS\_URL/job/sample2/build?token=TOKEN\_NAME

Here, JENKINS\_URL = The URL to access your Jenkins server (localhost:8080)

And TOKEN\_NAME = The Authentication Token that is given from the UI.

* + 1. Build after other projects are built = Set up a trigger so that when some other projects finish building, a new build is scheduled for this project. This is convenient for running an extensive test after a build is complete
    2. Build Periodically = Provides a [cron](http://en.wikipedia.org/wiki/cron)-like feature to periodically execute this project. Based on the provided expression, the job will gets executed at some interval. (Ex: \* \* \* \* - to execute the job for every minute)
    3. GitHub hook trigger for GIT SCM polling = If Jenkins will receive PUSH GitHub hook from repo defined in Git SCM section it will trigger Git SCM polling logic.
    4. Poll SCM = Configure Jenkins to poll changes in SCM
  1. Build Environment:
     1. Delete workspace before build starts = By default, the job’s entire workspace will be deleted
     2. Use secret text(s) or file(s) = Allows you to take credentials of various sorts and use them from shell build steps and the like. Each binding will define an environment variable.
     3. Provide Configuration files = Make [globally configured](http://192.168.56.105:8082/configfiles) files available in your local workspace. All files configured via the [config-file-provider plugin](https://wiki.jenkins-ci.org/display/JENKINS/Config+File+Provider+Plugin) are available and can be referenced.
     4. Abort the build if it's stuck = Abort build based on a time out strategy
     5. Add timestamps to the Console Output = Execution timestamp will be added to the console output
     6. Inject environment variables to the build process = Add or override environment variables to the whole build process. The variables will be injected after a SCM checkout (workspace elements are available).
     7. inject passwords to the build as environment variables = Inject global passwords to the job provided by Jenkins configuration.   
        Password values are encrypted.
  2. Build: Build information will be declared
  3. Post-build Actions: Here we can give post build actions

1. Click on “Apply” and then “Save”

The above-mentioned procedure explains the basic project in Jenkins.

**How to trigger the job remotely**

* Select the project from the list on dashboard that needs to be triggered remotely
* click on configure from left side menu’
* select “Build Triggers” tab and do the following:
  + check the trigger builds remotely option
  + Give Authentication Token value (ex. 1234)
  + Use the URL - <JENKINS\_URL>/job/sample%20project/build?token= TOKEN\_NAME
    - **Ex:** http://192.168.56.105:8082/job/sample%20project/build?token=1234
* Check the last build time on the workflow dashboard

**How to chain job executions**

* Create a sample project (ex: test1)
* Select “Build” tab and give command under the execute shell (ex: pwd)
* Click on “Apply” and “Save”
* Create a sample project (ex: test2)
* Select “Build” tab and give command under the execute shell (ex: ls -ltra)
* Click on “Apply” and “Save”
* Create a sample project (ex: test3)
* Select “Build” tab and give command under the execute shell (ex: date)
* Click on “Apply” and “Save”
* Select “test2” project and configure as follows:
  + go to “Build Trigger” section and check “Build after other projects are build”
  + Give the values “test1” and select one of the three options
  + Go to “Post-build Actions” section and select “Build other projects”
  + Give the value “test3” in projects to build
  + Click on “Apply” and “Save”
* Now, as soon as the “test1” build gets executed the remaining “test2 & test3” also get executed. Check the status on dashboard

**Jenkins Integration with GIT (SCM)**

To achieve this, we need to do the following

1. Create a java program and run it through command line
2. Create a Jenkins job to run the java program
3. Add the program/project to GIT
4. Jenkins – add git plugin
5. Configure Jenkins job to trigger the execution when a change is pushed to the GitHub
   * Create a free style project
   * Under “Source Code Management” , select “Git” and provide the Repository URL
   * Under “Build Triggers”, select ”Poll SCM” and provide Schedule value as “\* \* \* \* \* ”
   * Save

**Automated Deployment**

Main stages in continuous delivery and deployment pipeline are:

1. Build
2. Deploy
3. Test
4. Release

Above all the stages will have corresponding jobs that are chained. That means, Whenever Build jobs are successful then only the Deployment jobs will get triggered and when the Deployment jobs get successful then only the testing jobs get triggered and so on.

*Real-world project setup:*

*Build Stage:*

* Developers will make the changes in the application and commit the code to version control system (Git, SVN, CVS, etc.,)
* A Jenkins job (Build Job) will be triggered which will be either poll for the changes in the version control system and get triggered whenever there is a change, or we can configure it to run at some particular time interval

*Deploy Stage:*

* The outcome of the build job will be the project artifacts (Ex. WAR/EAR) which will be deployed to the servers
* A Jenkins job (Deploy Job) will be triggered which will take the outcomes of the build i.e., WAR/EAR file and deploy it to the corresponding testing environment
* Here we can have one or multiple test environments based on the organization structure.

*Test Stage:*

* Once the deployment jobs are successfully completed, then the testing jobs will get triggered which will run different kind of automated tests
* Finally, when all kinds of tests (Functional and non-functional tests) are successfully completed then the application will gets deployed into Production environment

*Release Stage*

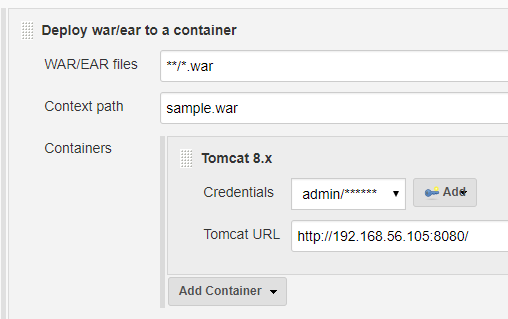
* The application deployment in Production environment is nothing but “Release”

***What is Automated Deployments:***

The Automated deployment is the process of automating the deployment process in a continuous delivery system which includes Build – Deploy – Test – Release

***How to do Automated Deployments:***

1. Start Jenkins and login to the console
2. Install Plugin – [*Deploy to container Plugin*](http://wiki.jenkins-ci.org/display/JENKINS/Deploy+Plugin)
3. Create a build job in Jenkins
4. Download a sample war file or can use any existing/available one
5. Add Post-build actions
   1. Select “Deploy war/ear to container” and add the fields as follows:



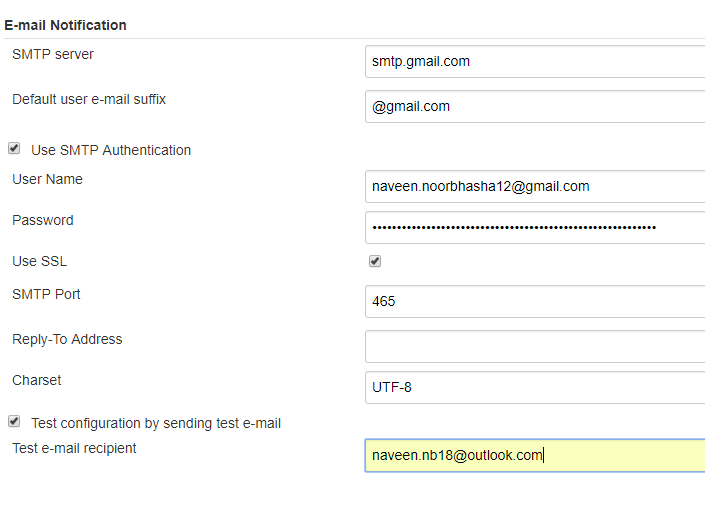
* 1. Click on “Apply” and “Save”

1. Build Now
2. Check the manage-app in tomcat webpage

Note: Deploy Maven application for hands on

**How to send Email from Jenkins**

* Get the SMTP server details from - <https://domar.com/smtp_pop3_server>
* Go to Jenkins console > Manage Jenkins > Configure System
* Find “E-mail Notification” and fill the details as follows: (Example)



*Notification Plugin:*

Using this plugin, we can send notifications in the JSON and XML formats.

Ref: <https://plugins.jenkins.io/notification>

*Extreme Notification Plugin:*

This plugin can notify some endpoints about events that occur in Jenkins

Ref: <https://plugins.jenkins.io/extreme-notification>

*Email Extension Plugin:*

This plugin allows you to configure every aspect of email notification. You can customize when an email is sent, who should receive it, and what the email says.

Ref: <https://plugins.jenkins.io/email-ext>

**Pipeline in Jenkins:**

Pipeline is a workflow with group of events or jobs that are chained and integrated with each other in sequence. Every job in pipeline has some dependency on one or more other jobs. Every section of pipeline has jobs that do some processing or action.

**Setup DELIVERY PIPELINE in Jenkins;**

1. Chain required jobs in sequence
   1. SampleBuildJob
   2. SampleDeployJob
   3. SampleTestJob

Chain these jobs using Build after other projects are built section under Build Triggers

1. Install “delivery pipeline” plugin
2. Add delivery pipeline view in the Jenkins dashboard
   1. Add required settings in the “Edit View”
3. Run and validate

Note: Update the options in Delivery Pipeline View as per your need

**Setup BUILD PIPELINE in Jenkins;**

1. Chain required jobs in sequence
   1. SampleBuildJob
   2. SampleDeployJob
   3. SampleTestJob

Chain these jobs using Build after other projects are built section under Build Triggers

1. Install “build pipeline” plugin
2. Add build pipeline view in the Jenkins dashboard
   1. Add required settings in the “Configure”
3. Run and validate

**Jenkins Blue Ocean:**

Blue ocean is a new user interface for Jenkins and provides an interactive view for Jenkins pipeline and jobs

Prerequisites – Jenkins 2.7 or above

1. Install “Blue Ocean” plugin
2. Switch to Blue Ocean view by clicking “Open Blue Ocean” from the dashboard left hand side menu

Users can switch to Jenkins Classic view or Blue Ocean view any time

**Trigger a job with Email:**

1. Install “Poll Mailbox Trigger Plugin”
2. Create a job and enable “Poll Mailbox Trigger” option and add the required details
   1. Host
   2. Username
   3. Password
   4. Schedule
3. Test the connection
4. Click on Apply and Save