### **Introduction**

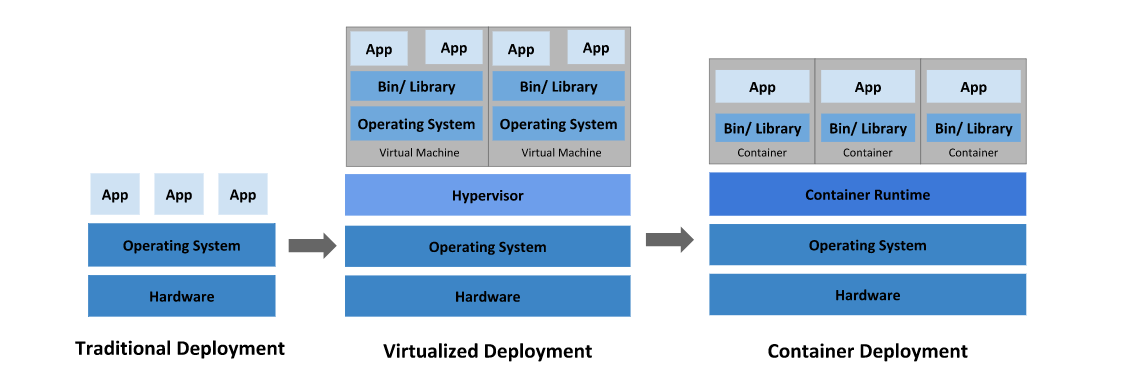
[Continuous integration, delivery, and deployment](https://www.digitalocean.com/community/tutorials/an-introduction-to-continuous-integration-delivery-and-deployment) are strategies designed to help increase the velocity of development and the release of well-tested, usable products. Continuous integration encourages development teams to test and integrate their changes to a shared codebase early to minimize integration conflicts. Continuous delivery builds off of this foundation by removing barriers on the way to deployment or release. Continuous deployment goes one step further by deploying every build that passes the test suite automatically.

* **Continuous Integration Server** (Jenkins, Bamboo, CruiseControl, TeamCity, and others)
* **Source Control Tool** (e.g., CVS, SVN, GIT, Mercurial, Perforce, ClearCase and others)
* **Build tool** (Make, ANT, Maven, Ivy, Gradle, and others)
* **Automation testing framework** (Selenium, Appium, TestComplete, UFT, and others)

**Continuous integration** focuses on integrating work from individual developers into a main repository multiple times a day to catch integration bugs early and accelerate collaborative development.

**Continuous delivery** is concerned with reducing friction in the deployment or release process, automating the steps required to deploy a build so that code can be released safely at any time.

**Continuous deployment** takes this one step further by automatically deploying each time a code change is made.



**Traditional deployment era:** Early on, organizations ran applications on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues. For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform. A solution for this would be to run each application on a different physical server. But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers.

**Virtualized deployment era:** As a solution, virtualization was introduced. It allows you to run multiple Virtual Machines (VMs) on a single physical server’s CPU. Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application.

Virtualization allows better utilization of resources in a physical server and allows better scalability because an application can be added or updated easily, reduces hardware costs, and much more. With virtualization you can present a set of physical resources as a cluster of disposable virtual machines.

Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.

**Container deployment era:** Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight. Similar to a VM, a container has its own filesystem, CPU, memory, process space, and more. As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.

Containers have become popular because they provide extra benefits, such as:

* Agile application creation and deployment: increased ease and efficiency of container image creation compared to VM image use.
* Continuous development, integration, and deployment: provides for reliable and frequent container image build and deployment with quick and easy rollbacks (due to image immutability).
* Dev and Ops separation of concerns: create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.
* Observability not only surfaces OS-level information and metrics, but also application health and other signals.
* Environmental consistency across development, testing, and production: Runs the same on a laptop as it does in the cloud.
* Cloud and OS distribution portability: Runs on Ubuntu, RHEL, CoreOS, on-prem, Google Kubernetes Engine, and anywhere else.
* Application-centric management: Raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources.
* Loosely coupled, distributed, elastic, liberated micro-services: applications are broken into smaller, independent pieces and can be deployed and managed dynamically – not a monolithic stack running on one big single-purpose machine.
* Resource isolation: predictable application performance.
* Resource utilization: high efficiency and density

[**Docker**](https://github.com/docker/docker) is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

**Docker** Swarm is a clustering and scheduling tool for [Docker](https://searchitoperations.techtarget.com/definition/Docker) containers. With Swarm, IT administrators and developers can establish and manage a [cluster](https://whatis.techtarget.com/definition/cluster) of Docker nodes as a single virtual system.

**Kubernetes** is a portable, extensible, open-source platform for managing containerized workloads and services. Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn’t it be easier if this behavior was handled by a system?

That’s how Kubernetes comes to the rescue! Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. For example, Kubernetes can easily manage a canary deployment for your system.

|  |  |  |
| --- | --- | --- |
| **Features** | **Kubernetes** | **Docker Swarm** |
| **Installation & Cluster Configuration** | Installation is complicated; but once setup, the cluster is very strong | Installation is very simple; but cluster is not very strong |
| **GUI** | GUI is the Kubernetes Dashboard | There is no GUI |
| **Scalability** | Highly scalable & scales fast | Highly scalable & scales 5x faster than Kubernetes |
| **Auto-Scaling** | Kubernetes can do auto-scaling | Docker Swarm cannot do auto-scaling |
| **Load Balancing** | Manual intervention needed for load balancing traffic between different containers in different Pods | Docker Swarm does auto load balancing of traffic between containers in the cluster |
| **Rolling Updates & Rollbacks** | Can deploy Rolling updates & does automatic Rollbacks | Can deploy Rolling updates, but not automatic Rollbacks |
| **Data Volumes** | Can share storage volumes only with other containers in same Pod | Can share storage volumes with any other container |
| **Logging & Monitoring** | In-built tools for logging & monitoring | 3rd party tools like ELK should be used for logging & monitoring |

**Puppet , Chef, Ansible, Saltstack**

Chef, Puppet, Ansible, and SaltStack are industry-wide used DevOps tools, included in [DevOps Certification](https://www.edureka.co/devops)*.*They are all “configuration management” tools, which means they are designed to deploy, configure and manage servers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Metrics** | **Chef** | **Puppet** | **Ansible** | **Saltstack** |
| Availability | ✔ | ✔ | ✔ | ✔ |
| Ease of Setup | Not very easy | Not very easy | Easy | Not very easy |
| Management | Not very easy | Not very easy | Easy | Easy |
| Scalability | Highly Scalable | Highly Scalable | Highly Scalable | Highly Scalable |
| Configuration language | DSL(Ruby) | DSL(PuppetDSL) | YAML(Python) | YAML(Python) |
| Interoperability | High | High | High | High |
| Pricing (upto 100 nodes) | $13700 | $11200-$19900 | $10,000 | $15,000(approx.) |

**Continuous integration** is a practice that encourages developers to integrate their code into a main branch of a shared repository early and often. Instead of building out features in isolation and integrating them at the end of a development cycle, code is integrated with the shared repository by each developer multiple times throughout the day.

The idea is to minimize the cost of integration by making it an early consideration. Developers can discover conflicts at the boundaries between new and existing code early, while conflicts are still relatively easy to reconcile. Once the conflict is resolved, work can continue with confidence that the new code honors the requirements of the existing codebase.

***Integrating code frequently*** does not, by itself, offer any guarantees about the quality of the new code or functionality. In many organizations, integration is costly because manual processes are used to ensure that the code meets standards, does not introduce bugs, and does not break existing functionality. Frequent integration can create friction when the level of automation does not match the amount quality assurance measures in place.

To address this friction within the integration process, in practice, ***continuous integration relies on robust test suites and an automated system to run those tests***. When a developer merges code into the main repository, automated processes kick off a build of the new code. Afterwards, test suites are run against the new build to check whether any integration problems were introduced. If either the build or the test phase fails, the team is alerted so that they can work to fix the build.

The end goal of continuous integration is to make integration a simple, repeatable process that is part of the everyday development workflow in order to reduce integration costs and respond to defects early. Working to make sure the system is robust, automated, and fast while cultivating a team culture that encourages frequent iteration and responsiveness to build issues is fundamental to the success of the strategy.

**Continuous delivery** is an extension of continuous integration. It focuses on automating the software delivery process so that teams can easily and confidently deploy their code to production at any time. By ensuring that the codebase is always in a deployable state, releasing software becomes an unremarkable event without complicated ritual. Teams can be confident that they can release whenever they need to without complex coordination or late-stage testing. As with continuous integration, continuous delivery is a practice that requires a mixture of technical and organizational improvements to be effective.

On the technology side, continuous delivery leans heavily on deployment pipelines to automate the testing and deployment processes. A **deployment pipeline** is an automated system that runs increasingly rigorous test suites against a build as a series of sequential stages. This picks up where continuous integration leaves off, so a reliable continuous integration setup is a prerequisite to implementing continuous delivery.

At each stage, the build either fails the tests, which alerts the team, or passes the tests, which results in automatic promotion to the next stage. As the build moves through the pipeline, later stages deploy the build to environments that mirror the production environment as closely as possible. This way the build, the deployment process, and the environment can be tested in tandem. The pipeline ends with a build that can be deployed to production at any time in a single step.

**Continuous deployment** is an extension of continuous delivery that automatically deploys each build that passes the full test cycle. Instead of waiting for a human gatekeeper to decide what and when to deploy to production, a continuous deployment system deploys everything that has successfully traversed the deployment pipeline. Keep in mind that while new code is automatically deployed, techniques exist to activate new features at a later time or for a subset of users. Deploying automatically pushes features and fixes to customers quickly, encourages smaller changes with limited scope, and helps avoid confusion over what is currently deployed to production.

**Key Concepts and Practices for Continuous Processes**

### Small, Iterative Changes

One of the most important practices when adopting continuous integration is to encourage small changes. Developers should practice breaking up larger work into small pieces and committing those early.

### Trunk-Based Development

With trunk-based development, work is done in the main branch of the repository or merged back into the shared repository at frequent intervals. Short-lived feature branches are permissible as long as they represent small changes and are merged back as soon as possible.

Releases are performed from the main branch or from a release branch created from the trunk specifically for that purpose.

### Keep the Building and Testing Phases Fast

Each of the processes relies on automated building and testing to validate correctness. Because the build and test steps must be performed frequently, it is essential that these processes be streamlined to minimize the time spent on these steps.

### Consistency Throughout the Deployment Pipeline

Because a continuous delivery or deployment implementations is supposed to be testing release worthiness, it is essential to maintain consistency during each step of the process—the build itself, the deployment environments, and the deployment process itself:

* **Code should be built once at the beginning of the pipeline**: The resulting software should be stored and accessible to later processes without rebuilding. By using the exact same artifact in each phase, you can be certain that you are not introducing inconsistencies as a result of different build tools.
* **Deployment environments should be consistent**: A configuration management system can control the various environments, and environmental changes can be put through the deployment pipeline itself to ensure correctness and consistency. Clean deployment environments should be provisioned each test cycle to prevent legacy conditions from compromising the integrity of the tests. The staging environments should match the production environment as closely as possible to reduce unknown factors present when the build is promoted.
* **Consistent processes should be used to deploy the build in each environment**: Each deployment should be automated and each deployment should use the same centralized tools and procedures. Ad-hoc deployments should be eliminated in favor of deploying only with the pipeline tools.

# **CI/CD Tools Comparison: Jenkins, GitLab CI, Buildbot, Drone, and Concourse**

[**Jenkins**](https://jenkins.io/) is one of the earliest open-source continuous integration servers and remains the most common option in use today.

Over the years, Jenkins has evolved into a powerful and flexible system of automating software-related tasks. Jenkins itself serves mainly as an automation framework with much of the important logic implemented through a library of plugins. Everything from listening for web hooks or watching repositories to building environments and language support is handled by plugins.

While this provides a great deal of flexibility, your CI process may come to rely on numerous third-party plugins, which can be fragile.

Jenkins is written in Java and released under an MIT license

[**GitLab CI**](https://about.gitlab.com/features/gitlab-ci-cd/) is a continuous integration tool built into [GitLab](https://about.gitlab.com/), a git repository hosting and development tools platform. Originally released as a standalone project, GitLab CI was integrated into the main GitLab software with the release of GitLab 8.0 in September, 2015.

The CI/CD process in GitLab CI is defined within a file in the code repository itself using a YAML configuration syntax. The work is then dispatched to machines called runners, which are easy to set up and can be provisioned on many different operating systems. When configuring runners, you can choose between different executors like Docker, shell, VirtualBox, or Kubernetes to determine how the tasks are carried out.

The tight coupling of GitLab CI with the GitLab repository platform has definite implications on the how the software can be used. GitLab CI is not an option for developers who use other repository hosting platforms. On the positive side, the integrated functionality allows GitLab users to set up a CI/CD environment without installing and learning an additional tool. Automated testing can begin by enabling a few options in the web interface, registering a runner machine, and adding a pipeline definition file into the repository. The close relationship also allows you to share runners between projects, see the current build status within the repository automatically, and keep build artifacts with the code that produced them.

GitLab and GitLab CI are written in Ruby and Go and released under an MIT license.

[Buildbot](https://buildbot.net/) is a continuous integration framework that offers tremendous amounts of flexibility. First released in 2003 as an alternative to Mozilla’s Tinderbox project, **Buildbot** was designed primarily as a way to automate build testing across a wide array of platforms.

Buildbot is released with GPL licensing and written in Python using the Twisted library. Rather than abstracting away the underlying language for the sake of simplified configuration, Buildbot’s configuration is written entirely in Python. This means that the configuration tends to be significantly more complex than other systems but administrators have more scope to design their ideal workflow and process. Each stage of the build is clearly separated and programmable. Buildbot positions itself as a framework with tools to build your own custom processes, comparable to how web frameworks allow you to build custom sites.

Buildbot’s history as a build testing platform means that it has support for many different operating systems and version control systems. Likewise, because it was designed with open-source testing in mind, its architecture allows users to easily submit workers with their preferred platforms to projects to expand the available test base. The user only needs to install a few Python packages on the system and then provide the credentials to the project.

[**Drone**](https://drone.io/) is a modern CI/CD platform built with a containers-first architecture. While the tools discussed above all include the option of running builds with Docker, a container-based workflow is at the core of Drone’s design. Drone is written in Go and was first released in 2014 under an Apache license.

Drone acts as a middle coordinating layer between Docker and a repository provider. Rather than starting up the CI/CD server and then hooking into a version control system hosting service afterwards, Drone requires the repository account information upfront to bootstrap its own authentication, user, and permissions models. As with all of its CI processes, Drone itself is run as a container. It supports multiple database backends and repository providers and has builtin support for setting up TLS/SSL certificates with [Let’s Encrypt](https://letsencrypt.org/) for transport encryption.

[**Concourse**](https://concourse.ci/) is a relatively new continuous integration platform initially released in 2014. Concourse’s approach to the CI/CD space is significantly different from the other tools we’ve looked at in that it attempts to take itself out of the equation as much as possible, minimizing state and abstracting every external factor into something it calls “resources”. The goal of this philosophy is to make the integration server entirely disposable so that the same processes can easily be run on any Concourse server.

Every part of the continuous integration process is composed from basic primitives that model different elements of the system. Each part of the process defines its dependencies explicitly. For example, the first task may require the latest commit to a VCS repository while later parts of the process may require the latest commit that passed previous stages. This method of constructing pipelines by mapping the exact dependencies of each step leads to strictly-defined behavior.

To further remove incidental state from the process, Concourse does not implicitly pass anything between jobs and does not provide any internal way of storing build artifacts. All information needed by the next stage must be explicitly defined, and potentially pushed to an external store to be pulled into the next step. By requiring explicit definitions, Concourse hopes to minimize the number of assumptions and unknown variables that the system has to account for.

Concourse is written in Go and released under an Apache license

## Conclusion

Continuous integration, delivery, and deployment software are complex automation systems designed to make your processes dependable and repeatable. As you can gather from the descriptions above, there are many different ideas about how automated testing and release is best accomplished, with emphasis placed on different parts of the equation. No single tool will satisfy the needs of every project, but with so many high quality open source solutions available, there’s a good chance you will be able to find a system that meets your team’s needs.

## JENKINS

## Jenkin is the most popular, open source Continuous Integration tool. It has tons of plugins that enhance its functionality.

## Jenkins is an open source Continuous Integration server capable of orchestrating a chain of actions that help to achieve the Continuous Integration process (and not only) in an automated fashion.

## In Continuous Integration after a code commit, the software is built and tested immediately. In a large project with many developers, commits are made many times during a day. With each commit code is built and tested. If the test is passed, build is tested for deployment. If deployment is a success, the code is pushed to production. This commit, build, test, and deploy is a continuous process and hence the name continuous integration/deployment.

## https://www.guru99.com/images/1/063018_1012_WhatisJenki1.jpg

### Jenkins Plugins

By default, Jenkins comes with a limited set of features. If you want to integrate your Jenkins installation with version control tools like Git, then you need to install plugins related to Git. In fact, for integration with tools like Maven, Amazon EC2, you need to install respective plugins in your Jenkins.

## Advantages of using Jenkins

* Jenkins is being managed by the community which is very open. Every month, they hold public meetings and take inputs from the public for the development of Jenkins project.
* So far around 280 tickets are closed, and the project publishes stable release every three months.
* As technology grows, so does Jenkins. So far Jenkins has around 320 plugins published in its plugins database. With plugins, Jenkins becomes even more powerful and feature rich.
* Jenkins also supports cloud-based architecture so that you can deploy Jenkins in cloud-based platforms.
* The reason why Jenkins became popular is that it was created by a developer for developers.

## Disadvantages of using Jenkins

Though Jenkins is a very powerful tool, it has its flaws.

* Its interface is out dated and not user friendly compared to current UI trends.
* Though Jenkins is loved by many developers, it's not that easy to maintain it because Jenkins runs on a server and requires some skills as server administrator to monitor its activity.
* One of the reasons why many people don't implement Jenkins is due to its difficulty in installing and configuring Jenkins.
* Continuous integrations regularly break due to some small setting changes. Continuous integration will be paused and therefore requires some developer attention.

# How to Download & Install Jenkins on Windows

Jenkins may be installed on either **Windows or Unix platforms.**

**Hardware requirements:**

* You need minimum 256 MB of RAM in your computer or laptop to install Jenkins
* You need at least 1 GB of space in your hard drive for Jenkins.

**Software Requirements:**

* Since Jenkins runs on Java, you need either latest version of Java Development Kit (JDK) or Java Runtime Environment (JRE).

**Step 1)** Got to <https://jenkins.io/download/> and select the platform

**Step 2)** Go to download location from local computer and unzip the downloaded package. Double-click on unzipped **jenkins.msi**.

(or)

Download the jenkins.war.

you can start Jenkins directly via the command line with java -jar jenkins\*.war.

**Step 3)** In the Jenkin Setup screen, click Next.

**Step 4)** Choose the location where you want to have the Jenkins instance installed (default location is C:\Program Files (x86)\Jenkins), then click on **Next** button.

**Step 5)**Click on the Install button.

**Step 6)** Once install is complete, click Finish.

**Step 7)** During the installation process an info panel may pop-up to inform the user that for a complete setup, the system should be rebooted at the end of the current installation. Click on OK button when the Info panel is popping-up:

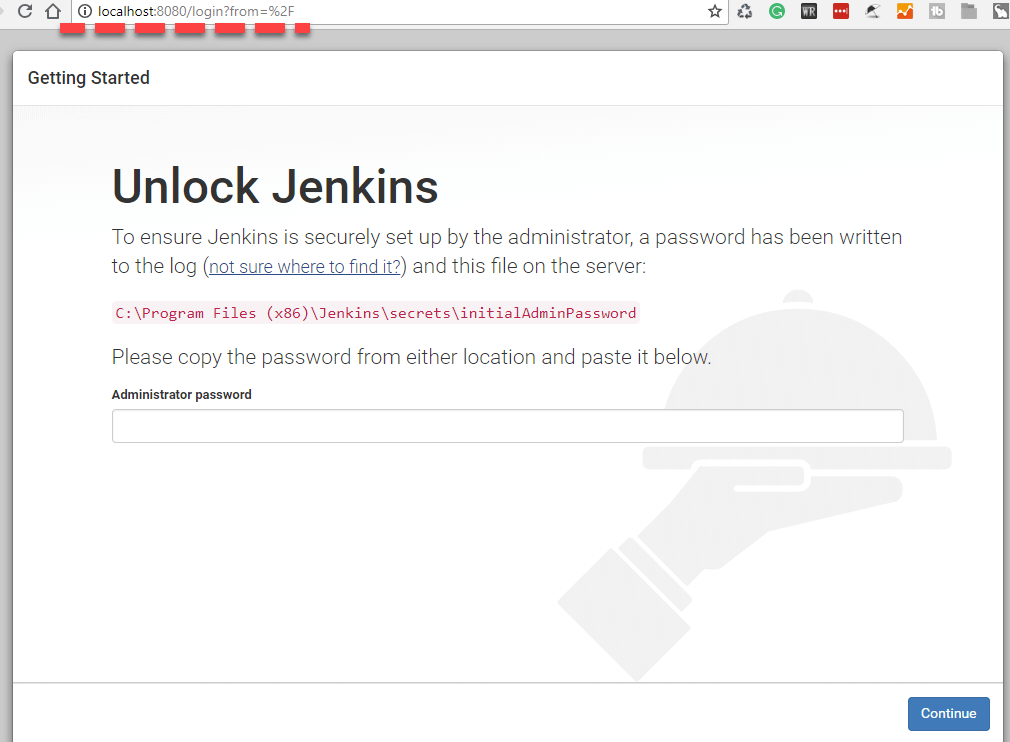
## How to Unblock Jenkins?

After completing the Jenkins installation phase, you should proceed further and start its configuration. Next steps will guide you how you can unblock Jenkins application:

**Step 1)** After completing the Jenkins installation process, a browser tab will pop-up asking for the initial Administrator password. To access Jenkins, you need to go to browse the following path in your web browser.

http://localhost:8080

If you can access the above URL, then it confirms that Jenkins is successfully installed in your system.



**Step 2)** The initial Administrator password should be found under the Jenkins installation path (set at Step 4 in Jenkins Installation).

For default installation location to C:\Program Files (x86)\Jenkins, a file called **initialAdminPassword** can be found under C:\Program Files (x86)\Jenkins\secrets.

However, If a custom path for Jenkins installation was selected, then you should check that location for **initialAdminPassword** file.

## **Step 3)**  copy the content of the **initialAdminPassword**file.

## **Step 4)** Paste the password it into browser's pop-up tab (<http://localhost:8080/login?form=%2F>) and click on Continue button.

## Customize Jenkins

You can also customize your Jenkins environment by below-given steps:

**Step 1)** Click on the "Install suggested plugins button" so Jenkins will retrieve and install the essential plugins

## https://www.guru99.com/images/1/063018_1023_HowtoDownlo12.jpg

## Jenkins will start to download and install all the necessary plugins needed to create new Jenkins Jobs.

## Note: You can choose the Option "Select Plugins to Install" and select the plugins you want to install

## Step 2) After all suggested plugins were installed, the "Create First Admin User" panel will show up. Fill all the fields with desired account details and hit the "Save and Finish" button.

## Step 3) Once you have filled the above data, finally it will ask for URL information where you can configure the default instance path for Jenkins. Leave it as it is to avoid any confusions later. However, if another application is already using 8080 port, you can use another port for Jenkins and finally save the settings, and you are done with installation of Jenkins. Hit the "Save and Continue" button:

## Step 3) Once you have filled the above data, finally it will ask for URL information where you can configure the default instance path for Jenkins. Leave it as it is to avoid any confusions later. However, if another application is already using 8080 port, you can use another port for Jenkins and finally save the settings, and you are done with installation of Jenkins. Hit the "Save and Continue" button:

## To know if Jenkins supports the third-party applications you have in mind, check their plugins directory at <https://wiki.jenkins-ci.org/display/JENKINS/Plugins>.

## Jenkins Standalone Architecture - What is Jenkins - Edureka

### [Configure the default port of the Jenkins build server](https://www.vogella.com/tutorials/Jenkins/article.html#configure-the-default-port-of-the-jenkins-build-server)

The default port number can be changed in the config file at

sudo vim /etc/default/jenkins

Here you can set a different port number, e.g. HTTP\_PORT=8082

Now you need to restart Jenkins with

service jenkins restart

or by adding /restart at the end of the Jenkins URL, e.g. [*https://yourjenkinsurl/ci/restart*](https://yourjenkinsurl/ci/restart).

# **How to Create Users & Manage Permissions: Role Strategy Plugin**

## By default, Jenkins comes with very basic user creation options. You can create multiple users but can only assign the same global roles and privileges to them. This not ideal, especially for a large organization.

## The ****Role Strategy Plugin****enable you to assign different roles and privileges to different users****.**** You will first need to install the plugin in your Jenkins mange environment.

## Install Role Strategy Plugin

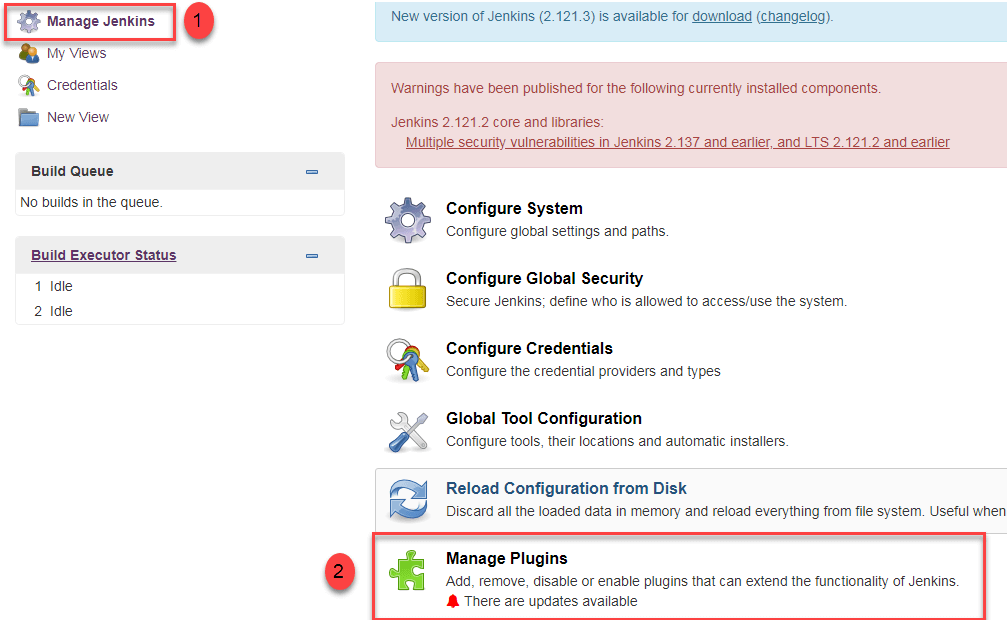
There are two methods for installing plugins in Jenkins:

1. Installing it through your Jenkins dashboard
2. Downloading the plugin from Jenkins website and installing it manually.

**Step 1)**

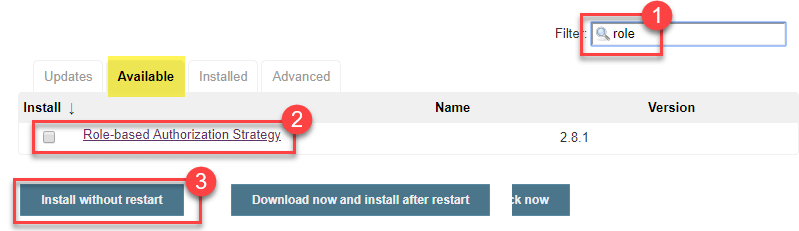
1. Go to **Manage Jenkins**

2. Click on the Manage Plugins option



**Step 2)**

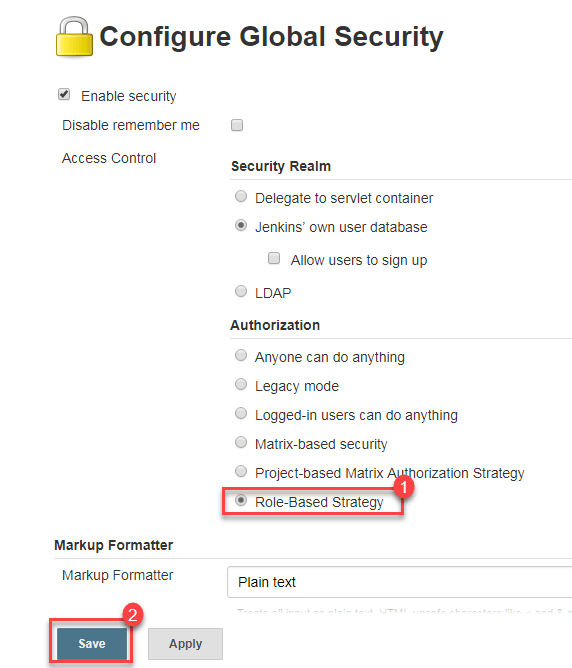
1. In available section, screenSearch for "role".
2. Select Role**-based Authorization Strategy**plugin
3. Click on "**Install without restart**" (make sure you have an active internet connection)



**Step 3)**

Once the plugin is installed, a "success" status will be displayed.

**Step 4)**Go to **Manage Jenkins ->**Configure Global Security -> Under **Authorization,**select **Role Based Strategy**.Click on**Save**



## How to Create/Add a User

* Under Manage Jenkins, Click Create User
* Enter User details like password, name, email etc.
* Click Create User

## https://www.guru99.com/images/1/091318_0444_HowtoCreate2.png

## Create Roles

In this step, we shall learn to certain roles to a single user or a group of users.

**Step 1)**

1. Go to **Manage Jenkins**

2. Select **Manage and Assign Roles**

## https://www.guru99.com/images/1/091318_0444_HowtoCreate8.png

**Note:**that the **Manage and Assign Roles**option will only be visible if you've installed the role strategy plugin.

**Step 2)**Click on **Manage Roles**to add new roles based on your organization.

## https://www.guru99.com/images/1/091318_0444_HowtoCreate9.png

**Step 3)**To create a new role called "developer",

1. Type "developer" under "role".
2. Click on "Add" to create a new role.
3. Now, select the permissions you want to assign to the "Developer" role.
4. Click Save

## https://www.guru99.com/images/1/091318_0444_HowtoCreate10.png

## Assign a Role

**Step 1)**Now that you have created roles, let us assign them to specific users.

1. Go to **Manage Jenkins**
2. Select Manage and Assign Roles

## https://www.guru99.com/images/1/091318_0444_HowtoCreate11.png

## ****Step 2)****We shall add the new role "developer" to user

1. Selector developer role checkbox
2. Click Save

## https://www.guru99.com/images/1/091318_0444_HowtoCreate12.png

# **What are the SCM tools supported by jenkins?**

below are Source code management tools supported by Jenkins:

* AccuRev: <https://wiki.jenkins-ci.org/display/JENKINS/AccuRev+Plugin>
* CVS: <https://wiki.jenkins-ci.org/display/JENKINS/CVS+Plugin>
* Subversion: <https://wiki.jenkins-ci.org/display/JENKINS/Subversion+Plugin>
* Git: <https://wiki.jenkins-ci.org/display/JENKINS/Git+Plugin>
* Mercurial: <https://wiki.jenkins-ci.org/display/JENKINS/Mercurial+Plugin>
* Perforce: <https://wiki.jenkins-ci.org/display/JENKINS/Perforce+Plugin>
* Clearcase: <https://wiki.jenkins-ci.org/display/JENKINS/ClearCase+Plugin>
* RTC: <https://wiki.jenkins-ci.org/display/JENKINS/Team+Concert+Plugin>

## Install GIT Plugin

## Jenkins needs to have GitHub plugin installed to be able to pull code from the GitHub repository.

**Step 1:**Click on the **Manage Jenkins** button on your Jenkins dashboard:

## https://www.guru99.com/images/1/091318_0440_JenkinsGitH2.png

## **Step 2:**Click on **Manage Plugins**:

## https://www.guru99.com/images/1/091318_0440_JenkinsGitH3.png

**Step 3:**In the Plugins Page

1. Select the GIT Plugin
2. Click on **Install without restart.**The plugin will take a few moments to finish downloading depending on your internet connection, and will be installed automatically.
3. You can also select the option **Download now and Install after restart**button. In which plugin is installed after restart
4. You will be shown a "No updates available" message if you already have the Git plugin installed.

## https://www.guru99.com/images/1/091318_0440_JenkinsGitH4.png

## ****Step 4:**** Once the plugins have been installed, go to ****Manage Jenkins****on your Jenkins dashboard. You will see your plugins listed among the rest.

# **How to Create Builds with the Jenkins Freestyle Project**

## ****Step 1)****To create a Jenkins freestyle job, log on to your Jenkins dashboard

## ****Step 2)****Click on "****New Item****" at the top left-hand side of your dashboard.

## https://www.guru99.com/images/1/091318_0458_HowtoCreate3.png

**Step 3)**In the next screen,

1. Enter the name of the item you want to create. We shall use the "Hello world" for this demo.
2. Select Freestyle project
3. Click Okay

## https://www.guru99.com/images/1/091318_0458_HowtoCreate4.png

## ****Step 4)****Enter the details of the project you want to test.

## https://www.guru99.com/images/1/091318_0458_HowtoCreate5.png

## ****Step 5)****Under Source Code Management, Enter your repository URL.

## https://www.guru99.com/images/1/091318_0440_JenkinsGitH10.png

## ****Step 6)****You might get an error message the first time you enter the repository URL. For example:

## https://www.guru99.com/images/1/091318_0440_JenkinsGitH11.png

## This happens if you do not have Gitinstalled in your local machine. To install Git in your local machine, go to <https://git-scm.com/downloads>

## Download the appropriate Git file for your Operating System

## ****Step 7)****You can execute Git repositories in your Jenkins once Git has been installed on your machine. To check ifithas been successfully installed onto your system, open your ****command prompt,**** type "Git"and press enter. You should see different options come up for Git:

## https://www.guru99.com/images/1/091318_0440_JenkinsGitH14.png

## Note: If you have GIT already installed in your system, just add git.exe path in Global Tool Configuration.

## ****Step 8)****Once you have everything in place, try adding the Git URL into Jenkins. You will not see any error messages:

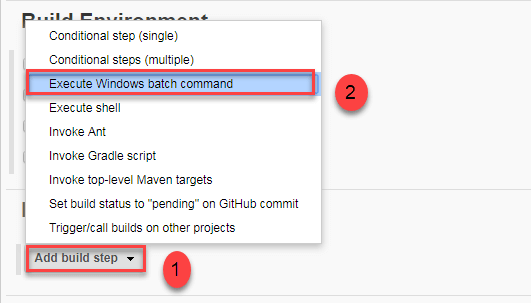
## If your GitHub repository is private, Jenkins will first validate your login credentials with GitHub and only then pull the source code from your GitHub repository.

**Step 9)**Now that you have provided all the details, it's time to build the code. Tweak the settings under the **build** section to build the code at the time you want. You can even schedule the build to happen periodically, at set times.

Under **build**,

1. Click on "**Add build step**"

2. Click on "**Execute Windows batch command**" and add the commands you want to execute during the build process.

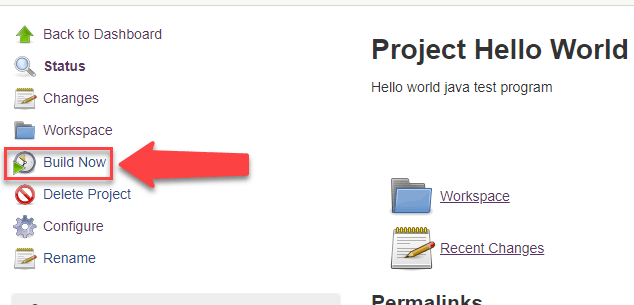




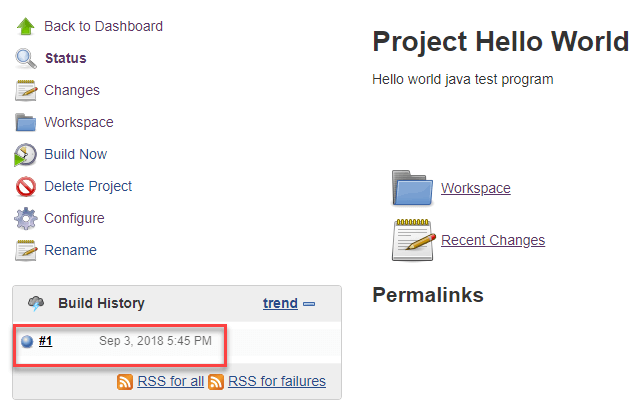
**Step 10)** When you have entered all the data,

1. Click **Apply**
2. **Save**the project.

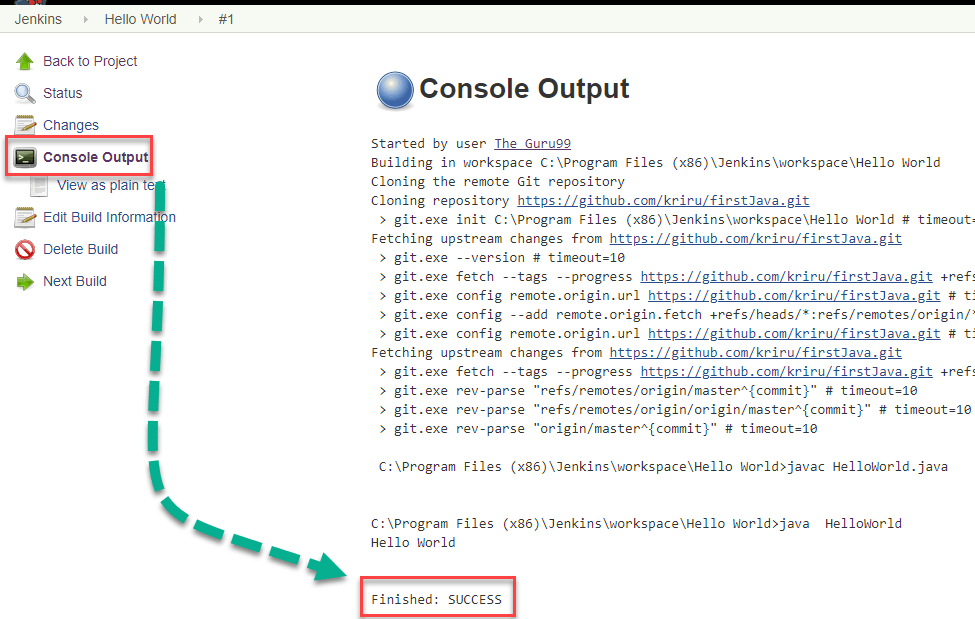
**Step 11)**Now, in the main screen, Click the **Build Now**button on the left-hand side to build the source code.



**Step 12)**After clicking on **Build now,**you can see the status of the build you run under **Build History**.



**Step 13)**Click on the **build number**andthenClick on **console output**to see the status of the build you run. It should show you a success message, provided you have followed the setup properly.



### **Creating Jenkins Builds:**

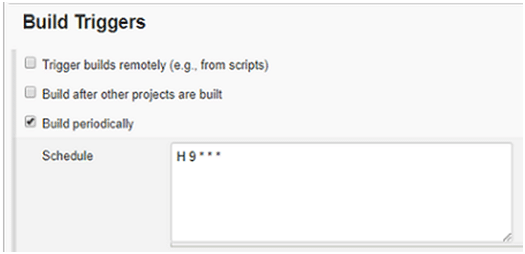
A build is most often called when the source code is converted to a usable and runnable form. It includes compiling the code into an executable form. The process of building is typically handled by the build tool. Builds are usually done when we reach a critical standpoint such as integration of a feature or so on. As Jenkins is CI based, we have a powerful feature where we can automate the Build process to happen at a particular time or event. This is called as “Scheduled Builds”

To schedule a build, follow the steps below:

* Find the “build triggers” section and check the build periodically box.
* In the box enter the scheduling parameters such as by date, day and time.

The general syntax is MINUTE (0-59), HOUR (0-23), DAY (1-31), MONTH (1-12), DAY OF THE WEEK (0-7)

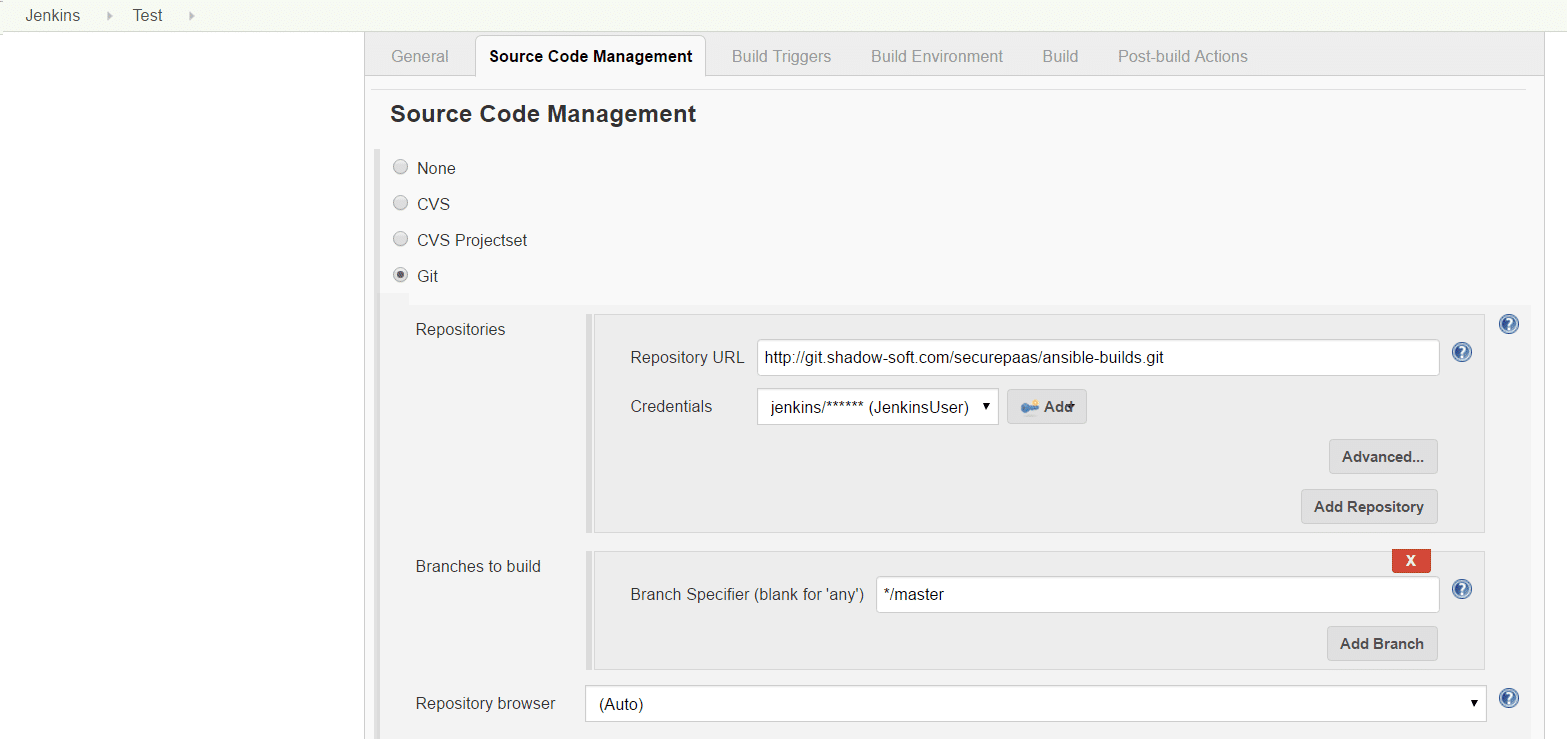
* If we would want to start a build every day at 8:30 form Monday to Friday, then we would have to give the parameters like: 30 08 \* \* 1-6.
* To start building daily in the afternoon from 4 to 4:59pm depending on the projects hash, we would have to give parameters like: H 16 \* \* 1-5.
* If we would like it to start it at midnight, then we would have to give the parameters like: @midnight or   we can also give it as 59 23 \* \* 6.
* To build every hour we would have to give it as:H \* \* \* \*



## GIT Webhook:

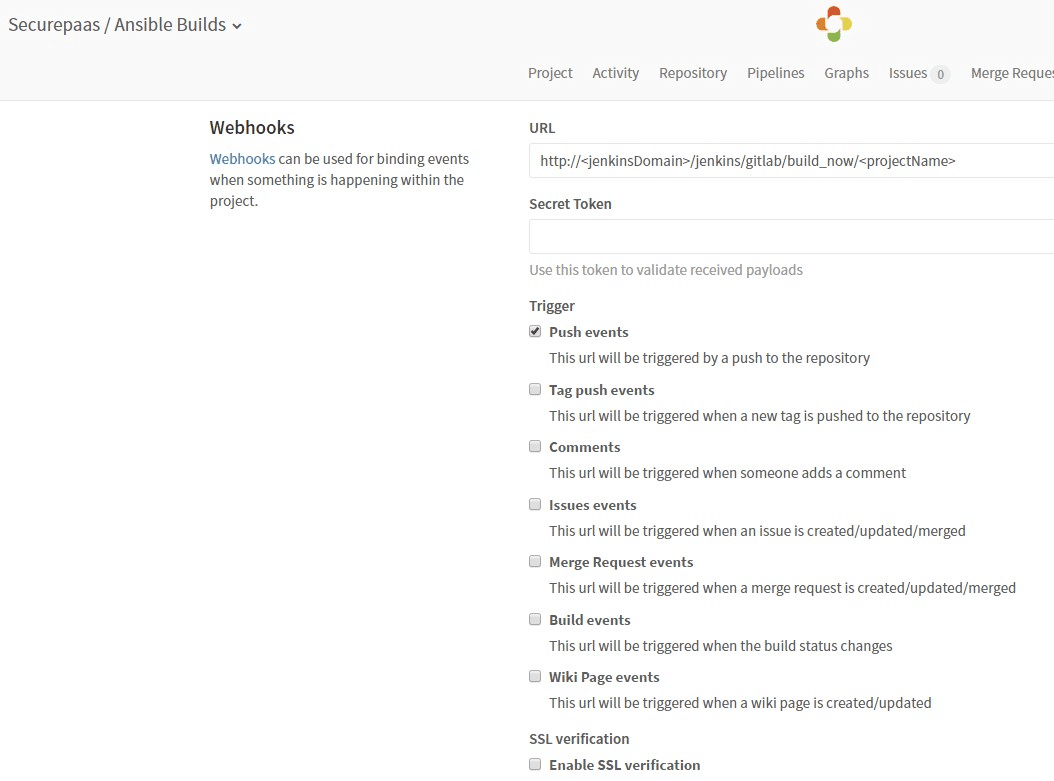
We have now seen how to automate builds in Jenkins with respect to a preset time or date. Now what if we could have it build every time we push a code in? To do exactly this we would need to install the “GitLab Hook Plugin” just as we have seen in the sections above.

scroll down to source control management and add the URL of the Git repository along with any credentials needed. Now add the steps that you would want to be performed during the build. Go to the add built setup and add build step. Note that this step will depend on what you want to do and the basis of your environment.



Now we would have to add a webhook for our repository in GitLab. Now navigate to the instance and select the cog icon and chose the webhook. Now fill in the URL fields with http://<jenkinsDomain>/jenkins    /gitlab/build\_now/<projectName> .

Now click on the “Add webhook” button at the end of the page.



## ****Jenkins Pipelines****

## What is CI CD

We’re all aware that Jenkins has proven to be an expert in implementing continuous integration, continuous testing and continuous deployment to produce good quality software. When it comes to **continuous delivery**, Jenkins uses a feature called Jenkins pipeline.

A pipeline is a collection of jobs that brings the software from version control into the hands of the end users by using automation tools. It is a feature used to **incorporate continuous delivery** in our software development workflow.

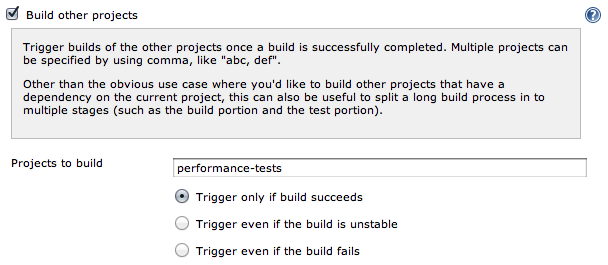
Over the years, there have been multiple Jenkins pipeline releases including, **Jenkins Build flow, Jenkins Build Pipeline plugin, Jenkins Workflow,** etc. What are the key features of these plugins?

* They represent multiple Jenkins jobs as one whole workflow in the form of a pipeline.
* What do these pipelines do? These pipelines are a **collection of Jenkins jobs** which trigger each other in a specified sequence.

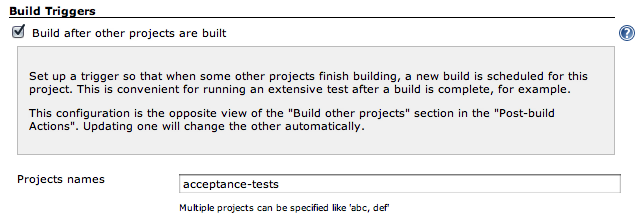
**Jenkins build pipeline**

Suppose I’m developing a small application on Jenkins and I want to build, test and deploy it. To do this, I will allot 3 jobs to perform each process. So, job1 would be for build, job2 would perform tests and job3 for deployment. I can use the Jenkins **build pipeline plugin** to perform this task. After creating three jobs and chaining them in a sequence, the build plugin will run these jobs as a pipeline

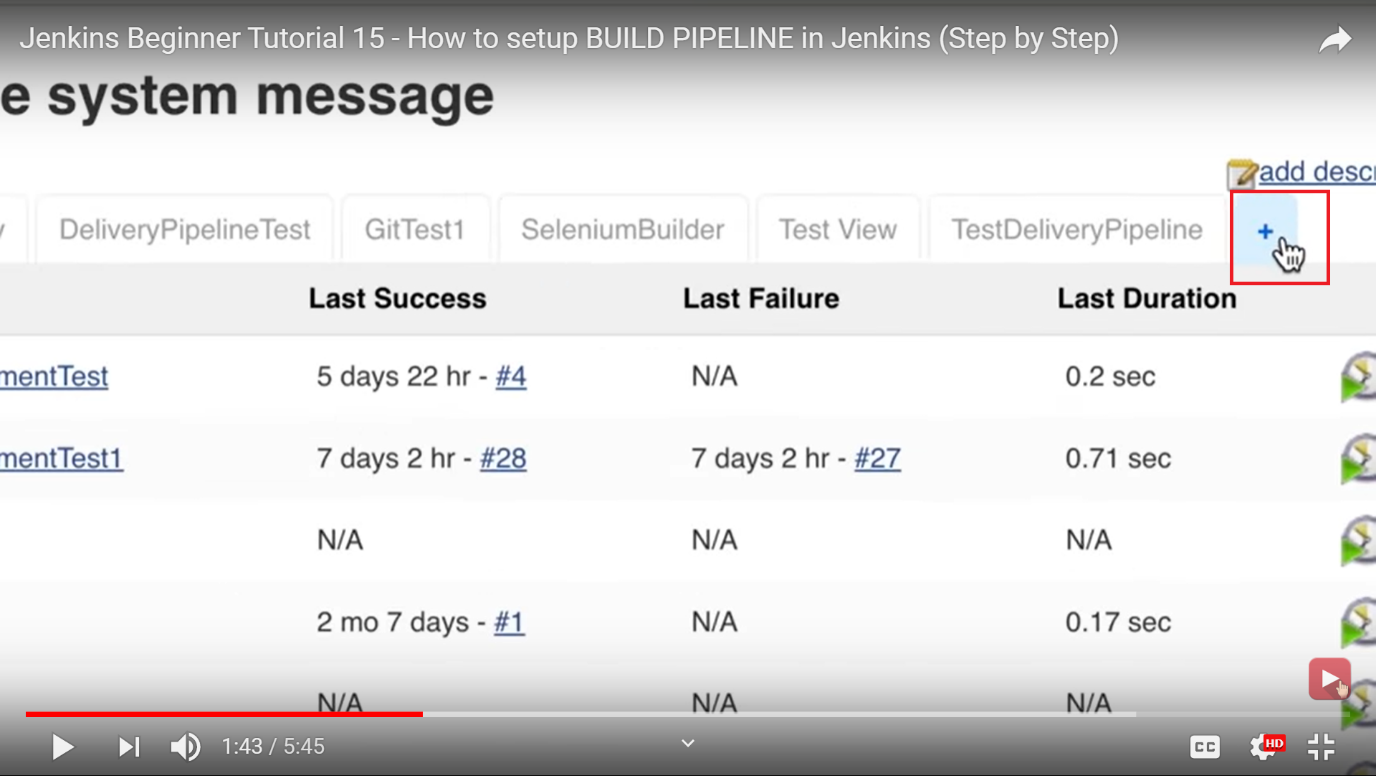
Jenkins has a built-in feature to build other projects. It is in the **Post-build Actions section**. You can specify the projects that you want to build after this project is built (you can trigger more than one). So whenever project A is built you will trigger the building of project B. You can also specify the conditions when the other jobs are built. Most often you are interested in continuing with the build pipeline only if the job is successful but your mileage might vary.

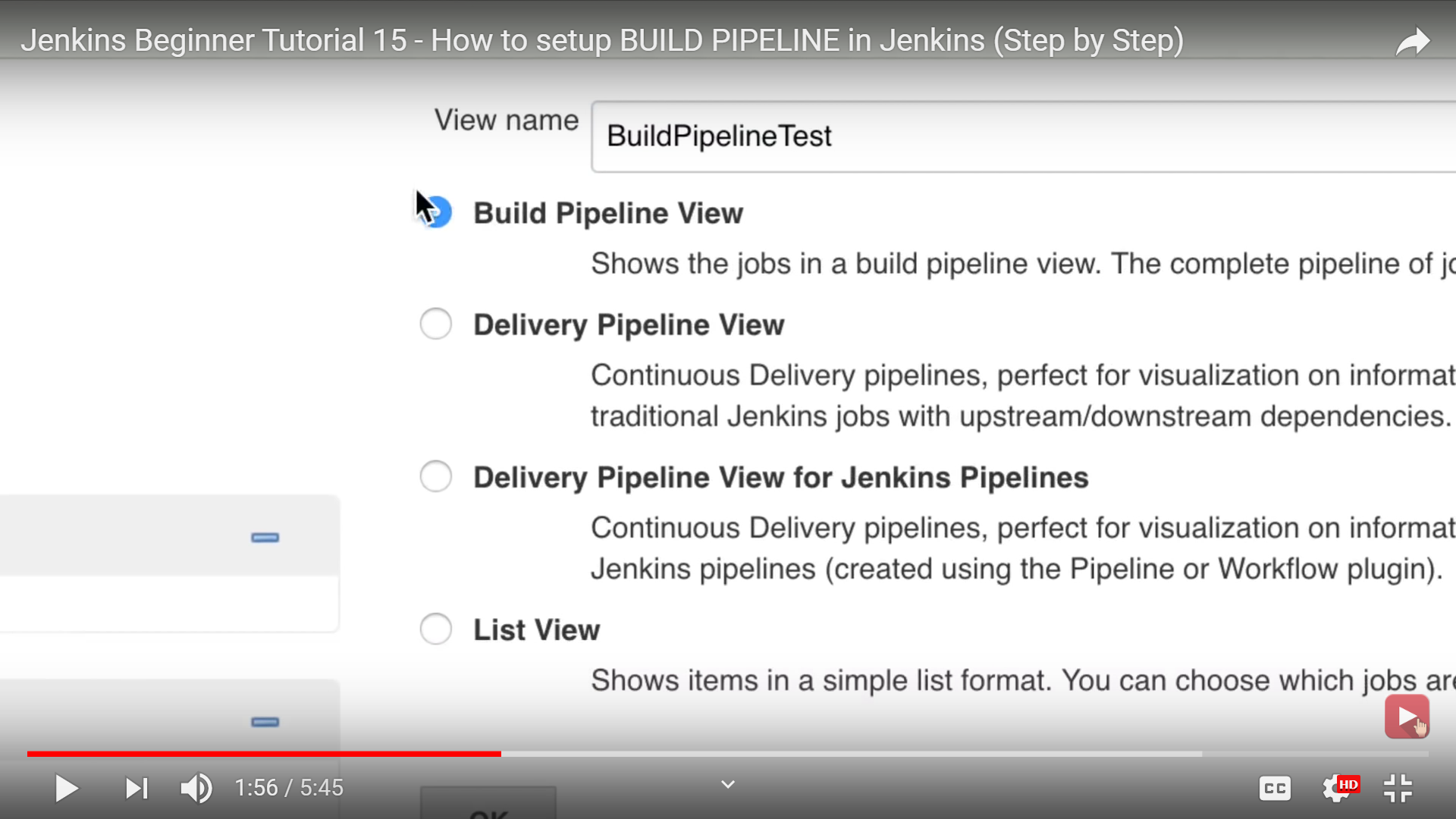


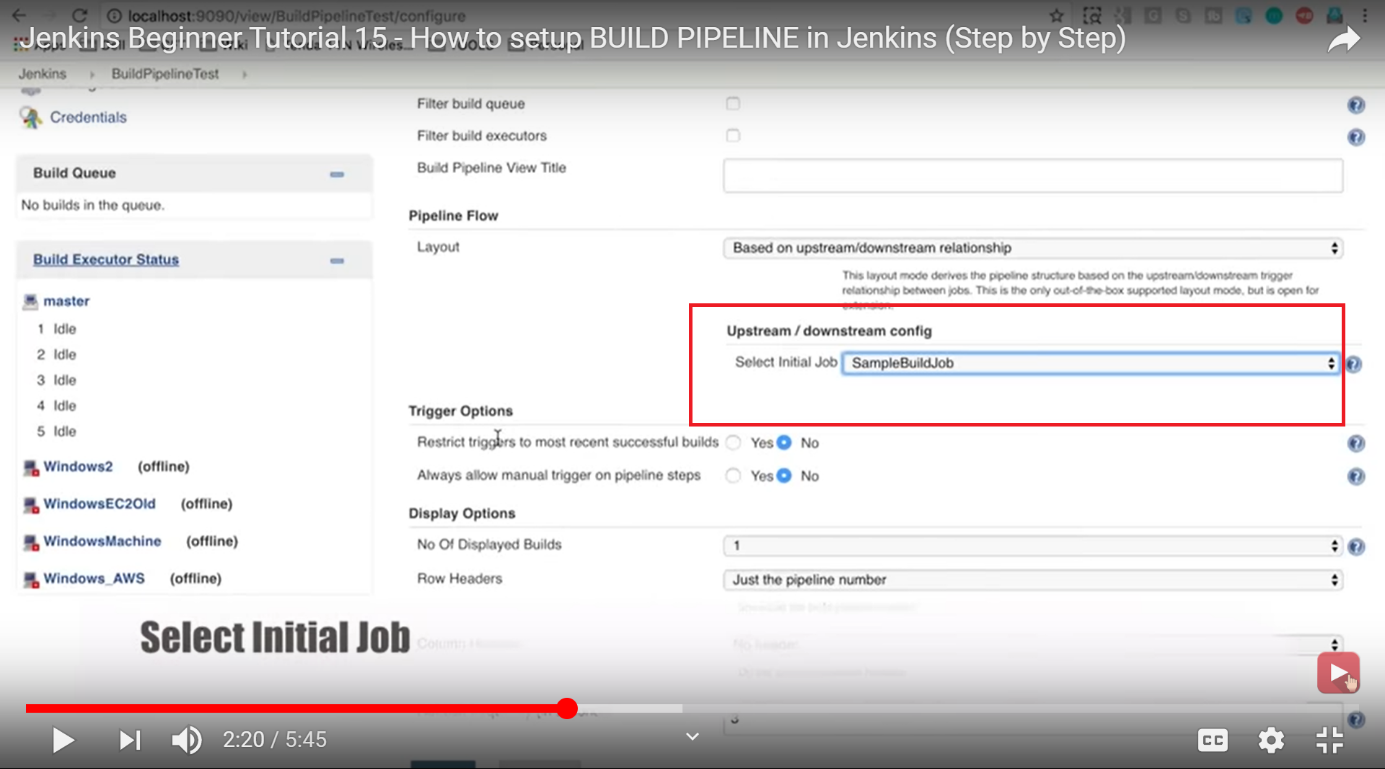
Another option is to configure this from project **B** and say “build this project **B** only after project **A** is built”. You don’t have to fill out both, just change one and the other is updated

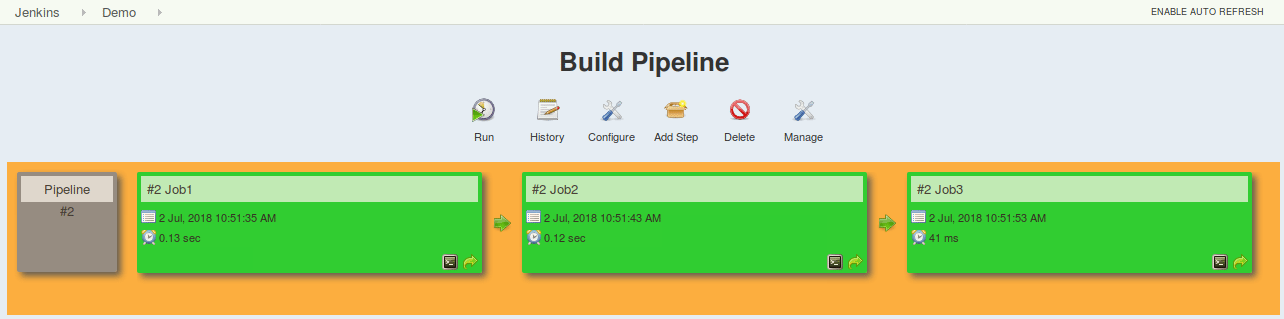


The Jenkins Build Pipeline plugin offers a very good visualization of the build pipeline. By configuring a new Jenkins view and choosing which job is the first job in the Jenkins pipeline you can get a visualization of the whole build pipeline.







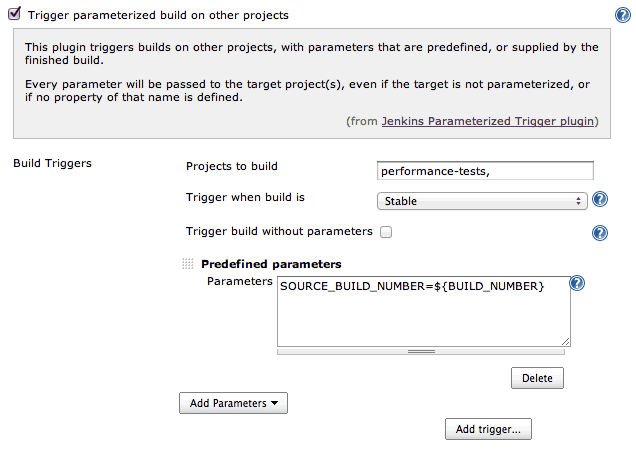


This approach is effective for deploying small applications. But what happens when there are complex pipelines with several processes (build, test, unit test, integration test, pre-deploy, deploy, monitor) running 100’s of jobs?

The maintenance cost for such a complex pipeline is huge and increases with the number of processes. It also becomes tedious to build and manage such a vast number of jobs. To overcome this issue, a new feature called **Jenkins Pipeline Project** was introduced.

### Jenkins Parameterized Trigger Plugin

This plugin lets you configure more aspects of the Jenkins triggering logic. It covers the basic [Out of the Box Solution](https://jrebel.com/rebellabs/how-to-use-jenkins-for-job-chaining-and-visualizations/#outofthebox) features and adds many more. The most important one is the option to trigger the next build with Jenkins pipeline parameters. For example, by defining SOURCE\_BUILD\_NUMBER=${BUILD\_NUMBER} you are able to use the variable $SOURCE\_BUILD\_NUMBER in project B. This way you can, for example, use the artifact built in the previous job to be fetched from your central artifact repository using the ${BUILD\_NUMBER}.



## Jenkin's Pipeline