

Table of Contents

[Jenkins Introduction](#)

[Workflow](#)

[Advantages](#)

[Jenkins Alternatives](#)

[Master-slave Concept](#)

[Jenkins Setup & Stages of CI-CD](#)

[Git Integration](#)

[Jobs in Jenkins](#)

[Types of Parameters](#)

[Branch Building](#)

[Schedule Project](#)

[User Administration in Jenkins](#)

[Upstream & Downstream](#)

[Linked Projects](#)

Jenkins Introduction

Jenkins offers a simple way to set up a continuous integration or continuous delivery (CI/CD) environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks. While Jenkins doesn't eliminate the need to create scripts for individual steps, it does give you a faster and more robust way to integrate your entire chain of build, test, and deployment tools than you can easily build yourself.

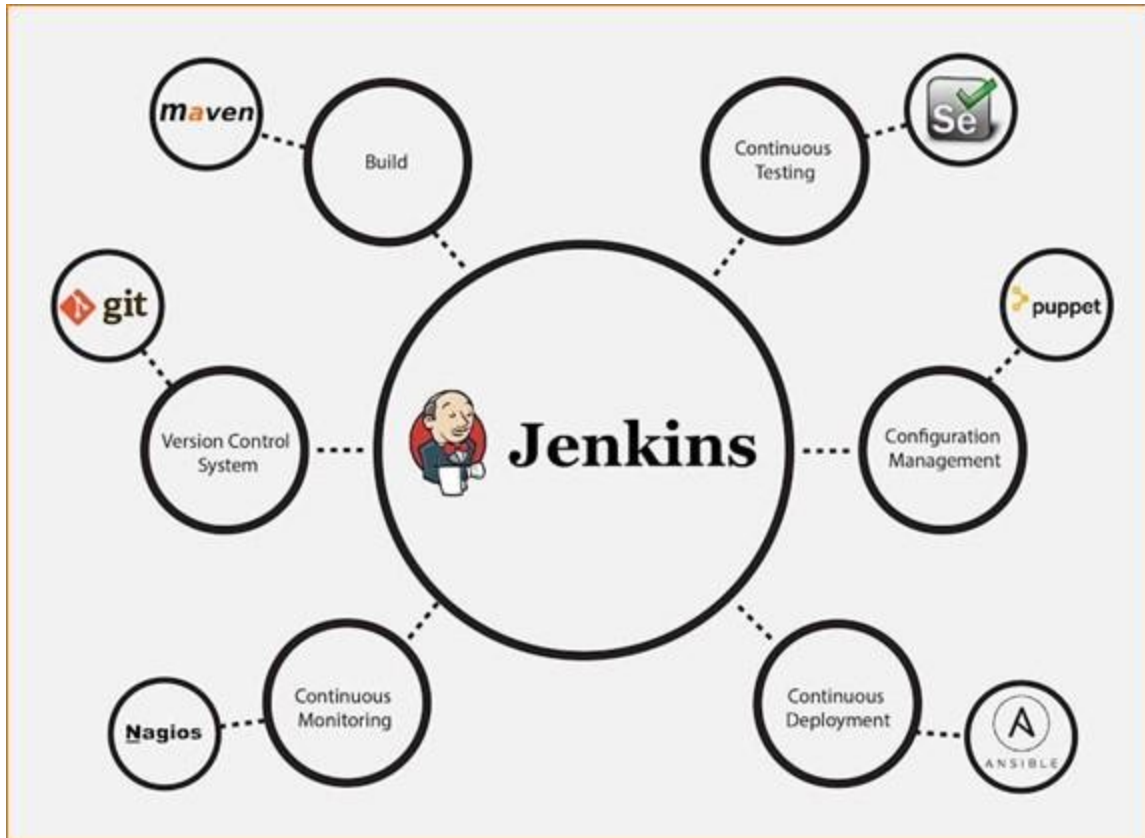
What is Jenkins and why do we use it?

Jenkins is an open-source automation tool written in Java with plugins built for Continuous Integration purposes. Jenkins is used to build and test your software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. It also allows you to continuously deliver your software by integrating with a large number of testing and deployment technologies.

With Jenkins, organizations can accelerate the software development process through automation. Jenkins integrates development life-cycle processes of all kinds, including build, document, test, package, stage, deploy, static analysis, and much more.

Jenkins achieves Continuous Integration with the help of plugins. Plugins allow the integration of Various DevOps stages. If you want to integrate a particular tool, you need to install the plugins for that tool. For example: Git, Maven 2 project, Amazon EC2, HTML publisher etc.

The image below depicts that Jenkins is integrating various DevOps stages:



Jenkins Features

The following are some facts about Jenkins that makes it better than other Continuous Integration tools:

Adoption: Jenkins is widespread, with more than 147,000 active installations and over 1 million users around the world.

Plugins: Jenkins is interconnected with well over 1,000 plugins that allow it to integrate with most of the development, testing and deployment tools.

It is evident from the above points that Jenkins has a very high demand globally.

Workflow

Jenkins Workflow is a plugin for Jenkins. Once installed, a new item type becomes available: a "Workflow". Workflow projects can be used for the same purposes as regular "Freestyle" Jenkins projects, but they also have the ability to orchestrate much larger tasks that can span multiple projects, and even create and manage multiple workspaces in a single Workflow. What's more, all of this management can be organized into a single script, rather than spread out across a collection of configurations, projects, and steps.

Advantages

- It is an open-source tool with great community support.
- It is easy to install.
- It has 1000+ plugins to ease your work. If a plugin does not exist, you can code it and share it with the community.
- It is free of cost.
- It is built with Java and hence, it is portable to all the major platforms.



Jenkins Alternatives

Following are some of the best alternatives to Jenkins:

- GitHub Actions.
- GitLab.
- Atlassian Bamboo.
- JFrog Pipelines.
- Spinnaker.
- JetBrains TeamCity.
- AWS CodePipeline.
- Azure DevOps Server (formerly Microsoft Team Foundation Server)
- Buddy
- FinalBuilder
- GoCD
- IBM Urbancode
- TeamCity

Jenkins installation on docker:

1. Install Docker on your ubuntu server by running the following command:

```
sudo apt update  
sudo apt install docker.io
```

```
root@jenkins: ~  
* Using username "pythonlife123".  
* Authenticating with public key "pythonlife123"  
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 5.4.0-1100-gcp x86_64)  
  
 * Documentation:  https://help.ubuntu.com  
 * Management:    https://landscape.canonical.com  
 * Support:       https://ubuntu.com/advantage  
  
System information as of Sat Feb 25 17:42:11 UTC 2023  
  
System load: 0.31          Processes: 118  
Usage of /: 15.2% of 9.51GB  Users logged in: 0  
Memory usage: 21%         IP address for eno4: 10.128.0.21  
Swap usage: 0%  
  
Expanded Security Maintenance for Applications is not enabled.  
0 updates can be applied immediately.  
  
Enable ESM Apps to receive additional future security updates.  
See https://ubuntu.com/esm or run: sudo pro status  
  
The programs included with the Ubuntu system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/*copyright.  
  
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by  
applicable law.  
  
pythonlife123@jenkins:~$ sudo -i  
root@jenkins:~#
```

```
root@jenkins: ~  
Get:1 http://us-central1-gcp.archive.ubuntu.com/ubuntu bionic/main amd64 bridge-utils amd64 1.5-15ubuntu1 [30.1 kB]  
Get:2 http://us-central1-gcp.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 runc amd64 1.1.0-0ubuntu1-18.04.1 [3896 kB]  
Get:3 http://us-central1-gcp.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 containerd amd64 1.5.9-0ubuntu1-18.04.2 [33.0 MB]  
Get:4 http://us-central1-gcp.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 docker.io amd64 20.10.12-0ubuntu2-18.04.1 [31.8 MB]  
Get:5 http://us-central1-gcp.archive.ubuntu.com/ubuntu bionic/main amd64 ubuntu-fan all 0.12.10 [34.7 kB]  
Fetched 68.5 MB in 7s (95.5 MB/s)  
Preconfiguring packages ...  
Selecting previously unselected package pigz.  
(Reading database ... 6668 files and directories currently installed.)  
Preparing to unpack .../0-pigz_2.4-1_amd64.deb ...  
Unpacking pigz (2.4-1) ...  
Selecting previously unselected package bridge-utils.  
Preparing to unpack .../1-bridge-utils_1.5-15ubuntu1_amd64.deb ...  
Unpacking bridge-utils (1.5-15ubuntu1) ...  
Selecting previously unselected package runc.  
Preparing to unpack .../2-runc_1.1.0-0ubuntu1-18.04.1_amd64.deb ...  
Unpacking runc (1.1.0-0ubuntu1-18.04.1) ...  
Selecting previously unselected package containerd.  
Preparing to unpack .../3-containerd_1.5.9-0ubuntu1-18.04.2_amd64.deb ...  
Unpacking containerd (1.5.9-0ubuntu1-18.04.2) ...  
Selecting previously unselected package docker.io.  
Preparing to unpack .../4-docker.io_20.10.12-0ubuntu2-18.04.1_amd64.deb ...  
Unpacking docker.io (20.10.12-0ubuntu2-18.04.1) ...  
Selecting previously unselected package ubuntu-fan.  
Preparing to unpack .../5-ubuntu-fan_0.12.10_all.deb ...  
Unpacking ubuntu-fan (0.12.10) ...  
Setting up pigz (2.4-1) ...  
Setting up runc (1.1.0-0ubuntu1-18.04.1) ...  
Setting up containerd (1.5.9-0ubuntu1-18.04.2) ...  
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.service → /lib/systemd/system/containerd.service.  
Setting up bridge-utils (1.5-15ubuntu1) ...  
Setting up ubuntu-fan (0.12.10) ...  
Created symlink /etc/systemd/system/multi-user.target.wants/ubuntu-fan.service → /lib/systemd/system/ubuntu-fan.service.  
Setting up docker.io (20.10.12-0ubuntu2-18.04.1) ...  
Setting up pigz (2.4-1) ...  
Adding group 'docker' (GID 116) ...  
Done.  
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /lib/systemd/system/docker.service.  
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /lib/systemd/system/docker.socket.  
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...  
Processing triggers for udev (0.100.0-21) ...  
root@jenkins:~#
```

2. Start docker service by running the following command:

`sudo systemctl start docker`

3. Create a Docker network by running the following command:

`sudo docker network create jenkins`

```
root@jenkins: ~  
Get:1 http://us-central1-gce.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 containerd amd64 1.5.9-0ubuntu1-18.04.2 [33.0 MB]  
Get:2 http://us-central1-gce.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 docker.io amd64 20.10.12-0ubuntu2-18.04.1 [31.8 MB]  
Fetched 68.9 MB in 2s (35.5 MB/s)  
Preconfiguring packages ...  
Selecting previously unselected package pigz.  
(Reading database ... 6686 files and directories currently installed.)  
Preparing to unpack .../0-pigz_2.4-1_amd64.deb ...  
Unpacking pigz (2.4-1) ...  
Selecting previously unselected package bridge-utils.  
Preparing to unpack .../1-bridge-utils_1.5-15ubuntu1_amd64.deb ...  
Unpacking bridge-utils (1.5-15ubuntu1) ...  
Selecting previously unselected package runc.  
Preparing to unpack .../2-runc_1.1.0-0ubuntu1-18.04.1_amd64.deb ...  
Unpacking runc (1.1.0-0ubuntu1-18.04.1) ...  
Selecting previously unselected package containerd.  
Preparing to unpack .../3-containerd_1.5.9-0ubuntu1-18.04.2_amd64.deb ...  
Unpacking containerd (1.5.9-0ubuntu1-18.04.2) ...  
Selecting previously unselected package docker.io.  
Preparing to unpack .../4-docker.io_20.10.12-0ubuntu2-18.04.1_amd64.deb ...  
Unpacking docker.io (20.10.12-0ubuntu2-18.04.1) ...  
Selecting previously unselected package ubuntu-fan.  
Preparing to unpack .../5-ubuntu-fan_0.12.10_all.deb ...  
Unpacking ubuntu-fan (0.12.10) ...  
Setting up runc (1.1.0-0ubuntu1-18.04.1) ...  
Setting up containerd (1.5.9-0ubuntu1-18.04.2) ...  
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.service → /lib/systemd/system/containerd.service.  
Setting up bridge-utils (1.5-15ubuntu1) ...  
Setting up ubuntu-fan (0.12.10) ...  
Created symlink /etc/systemd/system/multi-user.target.wants/ubuntu-fan.service → /lib/systemd/system/ubuntu-fan.service.  
Setting up pigz (2.4-1) ...  
Setting up docker.io (20.10.12-0ubuntu2-18.04.1) ...  
Adding group 'docker' (GID 116) ...  
Done.  
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /lib/systemd/system/docker.service.  
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /lib/systemd/system/docker.socket.  
Processing triggers for systemd (237-3ubuntu0.56) ...  
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...  
Processing triggers for ureadahead (0.100.0-21) ...  
root@jenkins:~# sudo systemctl start docker  
root@jenkins:~# sudo docker network create jenkins  
41a38cb38e5429607b91e78a52c6e87092cca46b3a5de7ea4663aff16bb6e87  
root@jenkins:~#
```

4. Pull the latest Jenkins image from from Docker Hub by running the following command:

`sudo docker pull jenkins/jenkins:its`

```
root@jenkins: ~  
root@jenkins:~# sudo docker pull jenkins/jenkins:its  
its: Pulling from jenkins/jenkins  
698c8a76747e: Pull complete  
656037b0c63c3: Pull complete  
81ba85001557: Pull complete  
f365c6b160fe: Pull complete  
7f0db80857bb: Pull complete  
b51e09c8a0bd: Pull complete  
a02b1ab95401: Pull complete  
b113a3f0acff6: Pull complete  
90c616f07a2d: Pull complete  
22b926230283: Pull complete  
32rdd8ea7030: Pull complete  
e04af8a1a00: Pull complete  
3938fe646b55: Pull complete  
Digest: sha256:8656eb80548f7d9c7be5d1f4c367ef432f2dd62f01efa6e795c9155258010d99  
Status: Downloaded newer image for jenkins/jenkins:its  
docker.io/jenkins/jenkins:its  
root@jenkins:~#
```

5. Run the Jenkins container with the following command:

`sudo docker run -d --name jenkins -p 80:8080 -p 50000:50000 --network jenkins -v jenkins_home:/var/jenkins_home jenkins/jenkins:its`

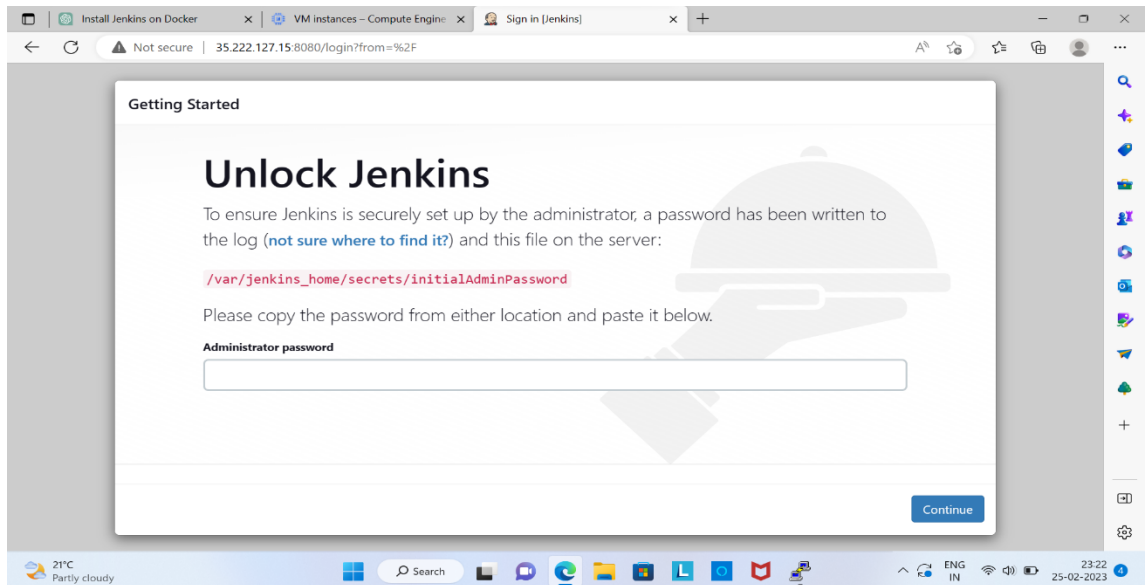
This command will start the Jenkins container in detached mode(-d) with the name “jenkins”. It maps the container’s port 8080 to the host’s port 8080(-p 8080:8080) and port 50000 to the host’s port 50000(-p 50000:50000).

```
root@jenkins:~# sudo docker pull jenkins/jenkins:lts
lts: Pulling from jenkins/jenkins
699c8a97647f: Pull complete
656837bc63c3: Pull complete
81ba89001557: Pull complete
f565c8b160fe: Pull complete
7f0db80857b0: Pull complete
b51e09c8a0bd: Pull complete
a02b1ab95401: Pull complete
b113c3f8acf6: Pull complete
90c61f07a2d: Pull complete
22b926230283: Pull complete
32fdd8eaf030: Pull complete
e04af8a1a0b: Pull complete
3938fe646b55: Pull complete
Digest: sha256:8656eb80548f7d9c7be5d1f4c367ef432f2dd62f81efa86795c9155258010d99
Status: Downloaded newer image for jenkins/jenkins:lts
docker.io/jenkins/jenkins:lts
root@jenkins:~# sudo docker run -d --name jenkins -p 8080:8080 -p 50000:50000 --network jenkins -v jenkins_home:/var/jenkins_home jenkins/jenkins:lts
f45814949746falc54bb3087b5cf80b43e00089dced4ced9ddb268ee4366f301
root@jenkins:~#
```

6. To access Jenkins, open a web browser and enter the following URL.

<http://<Server IP Address or Hostname>:8080>

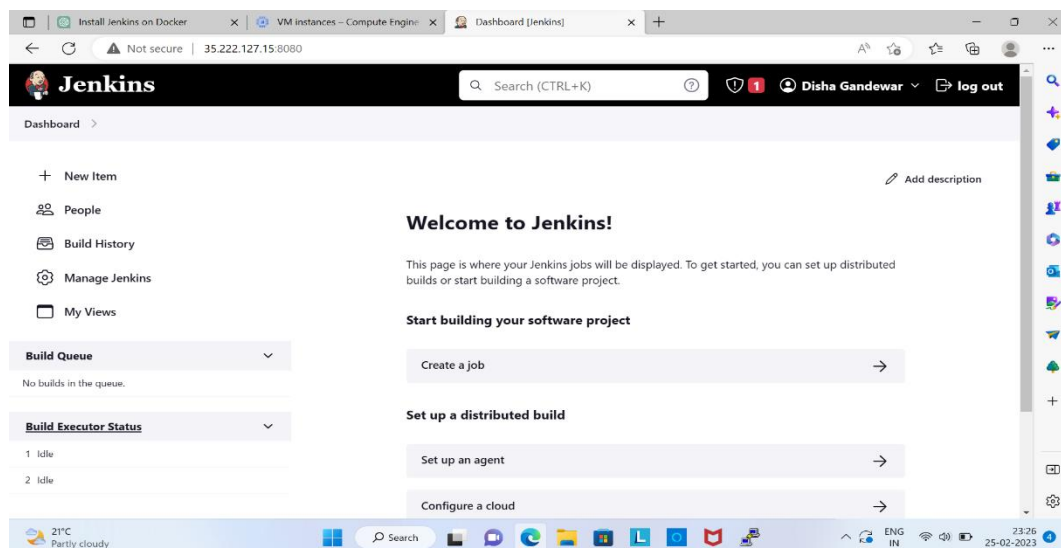
This will open the Jenkins login page. Follow the on-screen instructions to setup Jenkins.



7. After installing Jenkins on Docker, the Jenkins administration password can be obtained from the logs of the Jenkins container.

`sudo docker logs Jenkins` (OR)

`sudo docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword`



Stages of CI-CD

Stages in CI-CD of Jenkins.

- Stage 1 (Continuous Download)

Whenever developers upload some code into the Git repository Jenkins will receive a notification and it will download all that code. This is called Continuous Download.

- Stage 2 (Continuous Build)

The code downloaded in the previous stage had to be converted into a setup file commonly known as artifact. To create this artifact jenkins uses certain build tools like ANT, Maven etc. The artifact can be in the format of a .jar/.war/.ear file etc. This stage is called Continuous Build.

- Stage 3 (Continuous Deployment)

The artifact created in the previous stage has to be deployed into the QA Servers where a team of testers can start accessing it. This QA environment can be running on some application servers like tomcat, Weblogic etc. Jenkins deploys the artifact into these application servers and this is called Continuous Deployment.

- Stage 4 (Continuous Testing)

Testers create automation test scripts using tools like selenium, UFT etc. Jenkins runs these automation test scripts and checks if the

application is working according to client's requirement or not, if testing fails Jenkins will send automated email notifications to the corresponding team members and developers will fix the defects and upload the modified code into Git, Jenkins will again start from stage 1.

- Stage 5 (Continuous Delivery)

Once the application is found to be defect free Jenkins will deploy it into the Prod servers where the end user or client can start accessing it. This is called continuous delivery.

Git Integration

How does Jenkins integrate with Git? Go to Jenkins dashboard, click on "Manage Jenkins." Now follow these steps- Manage Plugins -> 'Available' tab -> Enter Git in search bar and filter -> Install required plugin. After the installation, all you need to do is click on "Configure System" and go to the 'GitHub' section.

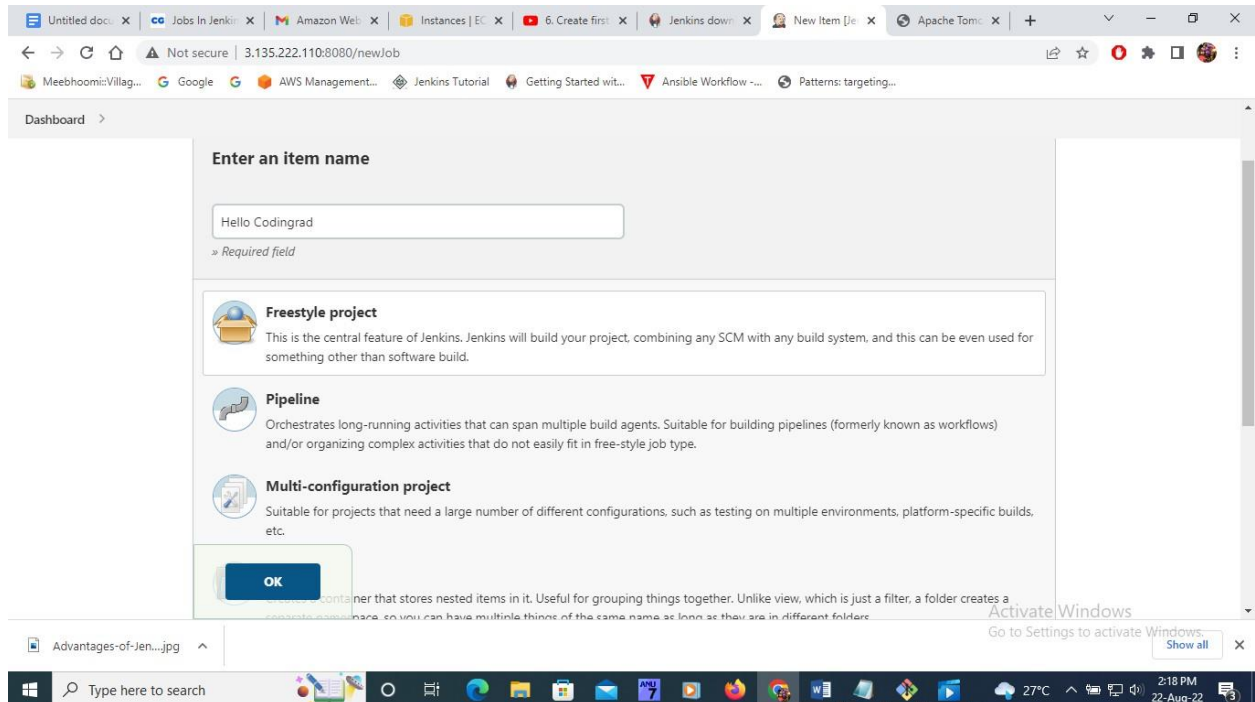
Jobs in Jenkins

Jenkins jobs are done in various stages and concepts like pipeline jobs. The major difference between any Jenkins job and a Jenkins Pipeline Job is that the Pipeline Scripted job runs on the Jenkins master. This uses a lightweight executor which uses only some resources to translate in the master to atomic commands that execute or send to the agents.

The following steps are the example for Jenkins jobs.

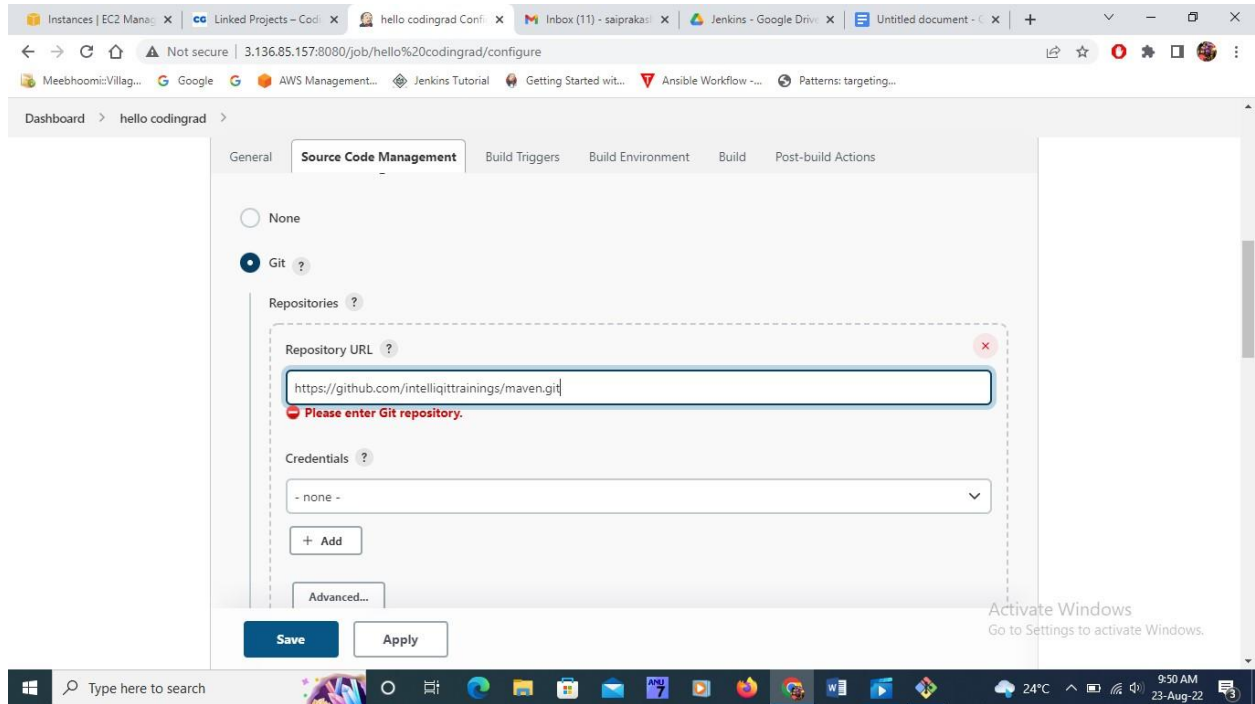
□ Stage 1 (Continuous Download)

- 1 Open the dashboard of Jenkins.
- 2 Click on New item---->enter item name as "HelloPythonlife".
- 3 Select Free style project---->OK



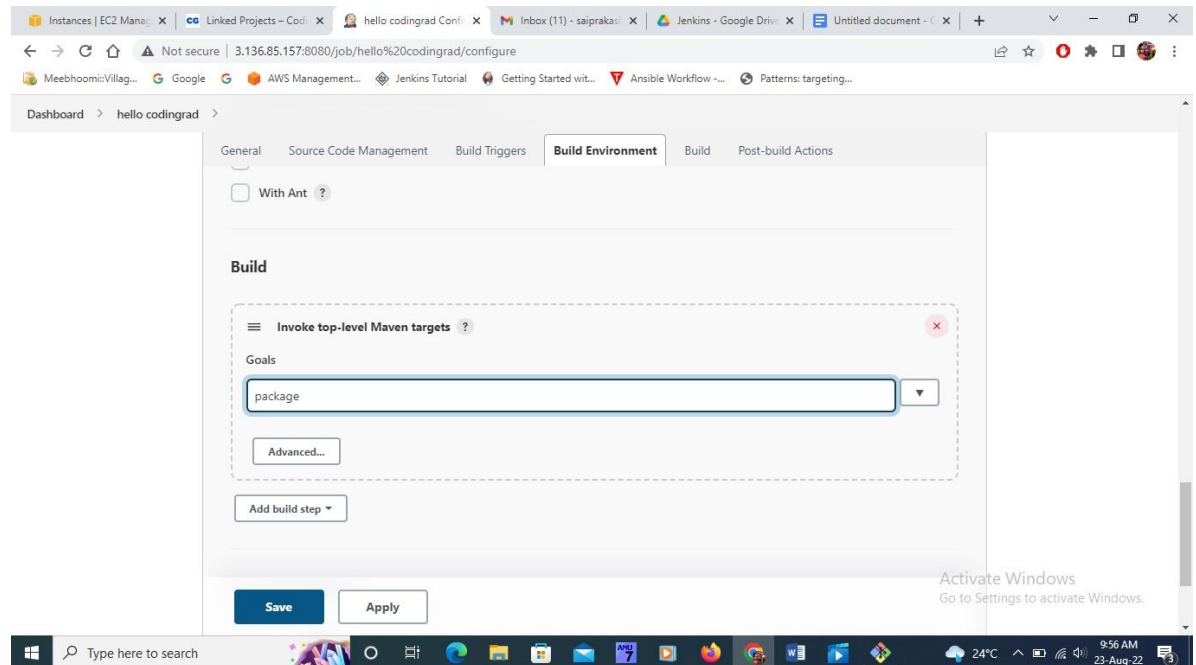
- 4 Go to Source code management
- 5 Select Git
- 8 Enter github url where developers have uploaded the code
<https://github.com/intelliqittrainings/maven.git>
- 9 Click on Apply--->Save
- 10 Go to the dashboard of Jenkins
- 11 Click on Build icon

This job will download all the code from github repository into the Jenkins server GCP instance.



☐ Stage 2 (Continuous Build)

- 1 Open the dashboard of Jenkins
- 2 Go to the hello pythonlife job--->Click on Configure
- 3 Go to Build section
- 4 Click on "Add build step"
- 5 Click on "Invoke top level maven target"



6 Enter Goal as: package

7 Apply--->Save

8 Go to the dashboard of Jenkins

9 Go to the hello pythonlife job and click on Build

iconThis job will convert the code into an artifact and this artifact comes in the format of a war file.



```
[1;34mINFO[m] Packaging webapp
[1;34mINFO[m] Assembling webapp [webapp] in [/home/ubuntu/.jenkins/workspace/hello_codingrad/webapp/target/webapp]
[1;34mINFO[m] Processing war project
[1;34mINFO[m] Copying webapp resources [/home/ubuntu/.jenkins/workspace/hello_codingrad/webapp/src/main/webapp]
[1;34mINFO[m] Webapp assembled in [54 msecs]
[1;34mINFO[m] Building war: /home/ubuntu/.jenkins/workspace/hello_codingrad/webapp/target/webapp.war
[1;34mINFO[m] WEB-INF/web.xml already added, skipping
[1;34mINFO[m] [1m-----[m
[1;34mINFO[m] [1mReactor Summary for Maven Project 1.0-SNAPSHOT:[m
[1;34mINFO[m]
[1;34mINFO[m] Maven Project ..... [1;32mSUCCESS[m [ 0.009 s]
[1;34mINFO[m] Server ..... [1;32mSUCCESS[m [ 11.070 s]
[1;34mINFO[m] Webapp ..... [1;32mSUCCESS[m [ 1.504 s]
[1;34mINFO[m] [1m-----[m
[1;34mINFO[m] [1;32mBUILD SUCCESS[m
[1;34mINFO[m] [1m-----[m
[1;34mINFO[m] Total time: 12.877 s
[1;34mINFO[m] Finished at: 2022-08-23T04:26:30Z
[1;34mINFO[m] [1m-----[m
Finished: SUCCESS
```

☐ Stage 3 (Continuous Deployment)

- 1 Open the dashboard of Jenkins
- 2 Click on Manage Jenkins
- 3 Click on Manage Plugins
- 4 Go to Available section
- 5 Search for "Deploy to container" plugin
- 6 Click on Install without restart
- 7 Go to the dashboard of jenkins
- 8 Go to the hello pythonlife job
- 9 Click on Configure
- 10 Go to Post build actions
- 11 Click on Add post build action
- 12 Click on Deploy war/ear to containerswar/ear file:
 ***.war Context path: testapp
 Click on Add container
 Select tomcat9
 Enter tomcat credentials
 Tomcat url: private_ip_qaserver:8080
- 13 Apply--->Save
- 14 Go to the dashboard of jenkins
- 15 Go to the hello pythonlife job---->Click on buildThis job will deploy the artifact into the QAServers and to access

the application from the level of browser
public_ip_qaserver:8080/testapp.

□ Stage 4 (Continuous Testing)

- 1 Open the dashboard of Jenkins
- 2 Click on New item--->Enter some item name(Testing)
- 3 Go to Source Code management---->Select Git4 Enter the github url where testers have uploaded the selenium code

<https://github.com/intelligittrainings/FunctionalTesting.git>

- 5 Go to Build section
- 6 Click on Add build step
- 7 Click on Execute shelljava -jar path_of_testing.jar
- 8 Click on Apply--->Save
- 9 Go to the dashboard of Jenkins
- 10 Go to the Testing job--->Click on Build icon

This job will download all the selenium test scripts and execute them.
Linking the "hello pythonlife job" with the "Testing job".

- 1 Open the dashboard of Jenkins
- 2 Go to the hello pythonlife job--->Click on Configure
- 3 Go to Post build actions
- 4 Click on Add Post Build action
- 5 Click on Build other projects
- 6 Enter project name as Testing
- 7 Click on Save

Now if we run the hello pythonlife job,it will finish the first 3 stages of CI and then call the Testing job.

Copy artifacts from hello pythonlife job to Testing job.

- 1 Open the dashboard of Jenkins
- 2 Click on Manage Jenkins
- 3 Click on Manage Plugins

- 4 Go to Available section
- 5 Search for Copy artifact plugin
- 6 Click on Install without restart
- 7 Go to the dashboard of Jenkins
- 8 Go to hello pythonlife job--->Click on configure
- 9 Go to Post build actions--->Click on Add post build actions
- 10 Click on Archive the artifacts
- 11 Enter files to be archived as `**/*.war`
- 12 Apply--->Save
- 13 Go to dashboard of Jenkins
- 14 Go to Testing job--->Configure
- 15 Go to Build section--->Click on Add build step
- 16 Click on Copy artifacts from other projects
- 17 Enter project name as "hello pythonlife"
- 18 Apply--->Save

☐ Stage 5 (Continuous Delivery)

- 1 Go to the dashboard of Jenkins
- 2 Go to the Testing job--->Configure
- 3 Go to Post build actions
- 4 Click on add post build actions
- 5 Click on Deploy war/ear to containerwar/ear file: `**/*.war`
context path: `prodapp`
Click on Add container---->Select tomcat9
- Enter username and password of tomcat9
- Tomcat url: `private_ip_prodserver:8080`
- 6 Apply--->Save

Pipeline as Code:

This is the process of implementing all the stages of CI-CD from the level of a Groovy script file called Jenkinsfile.

Advantages:-

- Since this is a code it can be uploaded into git and all the team members can review and edit the code and still git will maintain multiple versions and we can decide what version to use.
- Jenkinsfiles can withstand planned and unplanned restart of the Jenkins master.
- They can perform all stages of ci-cd with minimum no of plugins so they are faster and more secure.
- We can handle real world challenges like if conditions, loops, exception handling etc. ie if a stage in ci-cd passes we want to execute some steps and if it fails we want to execute some other steps.

Pipeline as code can be implemented in 2 ways.

1 Scripted Pipeline

2 Declarative Pipeline

Syntax of Scripted Pipeline:-

```
node('built-in')
{
    stage('Stage name in ci-cd')
    {
        Groovy code to implement this stage
    }
}
```

Syntax of Declarative Pipeline:-

```
pipeline
{
  agent
  any
  stages
  {
    stage('Stage name in CI-CD')
    {
      step
      s
      {
        Groovy code to implement this stage
      }
    }
  }
}
```

Types of Parameters

Jenkins supports several parameter types. Below is a list of the most common ones, but keep in mind that different plugins may add new parameter types:

- Choice: a predefined set of strings from which a user can pick a value.
- Multi-line String: same as *String*, but allows newline characters.
- Credentials: a predefined Jenkins credential.
- String: any combination of characters and numbers.

- File: the full path to a file on the filesystem.
- Password: similar to the *Credentials* type, but allows us to pass a plain text parameter specific to the job or pipeline.
- Run: an absolute URL to a single run of another job.

Branch Building

When the pipeline build starts, Jenkins uses the Jenkinsfile in that branch for build stages. SCM (Source Control) can be Github, Bitbucket, or a Gitlab repo. You can choose to exclude selected branches if you don't want them to be in the automated pipeline with Java regular expressions.

Schedule Project

Scheduling the job for a particular date and time.

- 1 Open the dashboard of Jenkins
- 2 Go to the configuration page of the job
- 3 Go to Build triggers
- 4 Click on Build periodically
- 5 Schedule the date and time
- 6 Click on Save

Jenkins Job Scheduling Syntax:

First, let's look at the Jenkins job scheduling configuration. It looks a lot like Linux's cron syntax, but you don't have to be familiar with command line Linux to figure it out.

A scheduling entry consists of five whitespace-separated fields. You can schedule a job for more than one time by adding more than one entry.

Minute	Hour	Day of Month	Month	Day of Week
--------	------	--------------	-------	-------------

Each field can contain an exact value or use a set of special expressions:

- The familiar asterisk "*" indicates all valid values. So, a job that runs every day has a "*" in the third field.
- A dash separates ranges of values. For example, a job that runs every hour from 9:00 a.m. to 5:00 p.m. would have 9-17 in the second field.
- Intervals are specified with a slash /. A job that runs every 15 minutes has H/15 in the first field. Note that the H in the first field has a special meaning. If you wanted a job to run every 15 minutes, you could configure it as 0/15, which would make it run at the start of every hour. However, if you configure too many jobs this way, you can overload your Jenkins controller. Ultimately, the H tells Jenkins to pick a minute based on a hash of the job name.
- Finally, you can specify multiple values with a comma. So, a job that runs Monday, Wednesday, and Friday would have 1,3,5 in the fifth field.

Jenkins provides a few examples in their help section for scheduling.

```
# Every fifteen minutes (perhaps at :07, :22, :37, :52):H/15 * * * *
# Every ten minutes in the first half of every hour (three times,
perhaps at :04, :14, :24):H(0-29)/10 * * * *
# Once every two hours at 45 minutes past the hour starting at 9:45
AM and finishing at 3:45 PM every weekday:45 9-16/2 * * 1-5
# Once in every two hour slot between 8 AM and 4 PM every weekday
(perhaps at 9:38 AM, 11:38 AM, 1:38 PM, 3:38 PM):H H(8-15)/2 * *
1-5
# Once a day on the 1st and 15th of every month except December:
H H 1,15 1-11 *
```

Jenkins also has a set of aliases that makes using common intervals easier.

Macro	Equivalent	Explanation
@hourly	H * * * *	any time during the hour
@daily	H H * * *	sometime during the day
@midnight	H H(0-2) * * *	sometime between 12:00 a.m. and 2:59 a.m.
@weekly	H H * * H	sometime during the week
@monthly	H H H * *	sometime during the month

Now that we've seen some examples, let's create a job and set it up with a schedule.

User Administration in Jenkins

Creating users in Jenkins

- 1) Open the dashboard of jenkins
- 2) click on manage jenkins
- 3) click on manage users
- 4) click on create users
- 5) enter user credentials

Creating roles and assigning

- 1) Open the dashboard of jenkins
- 2) click on manage jenkins
- 3) click on manage plugins
- 4) click on role based authorization strategy plugin
- 5) install it
- 6) go to dashboard-->manage jenkins
- 7) click on configure global security
- 8) check enable security checkbox
- 9) go to authorization section-->click on role based strategyradio button
- 10) apply-->save

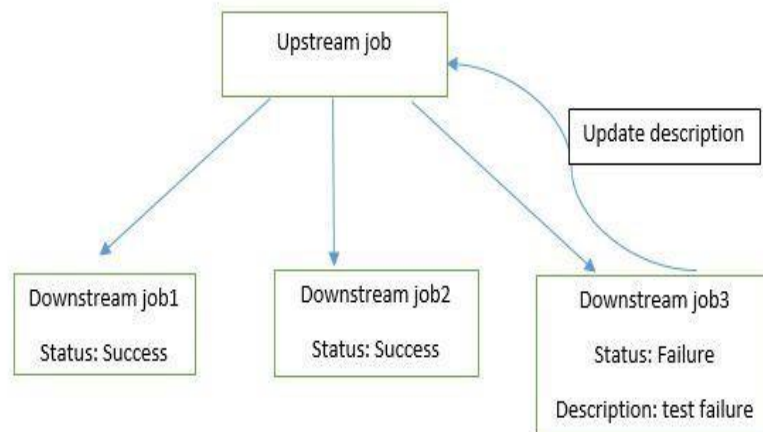
- 11) go to dashboard of jenkins
- 12) click on manage jenkins
- 13) click on manage and assign roles
- 14) click on manage roles
- 15) go to global roles and create a role "employee"
- 16) for this employee in overall give read access and in view section give all access
- 17) go to project roles-->Give the role as developer and pattern as Dev.* (ie developer role can access only those jobs whose name start with Dev)
- 18) similarly create another role as tester and assign the pattern as "Test.*"
- 19) give all permissions to developers and tester
- 20) apply--save
- 21) click on assign roles
- 22) go to global roles and add user1 and user2
- 23) check user1 and user2 as employees
- 24) go to item roles
- 25) add user1 and user2
- 26) check user1 as developer and user2 as tester
- 27) apply-->save

If we login into jenkins as user1 we can access only the development related jobs and user2 can access only the testing related jobs.

Upstream & Downstream

An upstream job is a configured project that triggers a project as part of its execution.

A downstream job is a configured project that is triggered as part of a pipeline execution.



Upstream and downstream jobs help you to configure the sequence of execution for different operations and hence you can orchestrate the flow of execution. We can configure one or more projects as downstream jobs in Jenkins.

Linked Projects

Linking different jobs to create pipelines in Jenkins. With the emergence of trends such as continuous deployment and delivery, the continuous integration server is not limited to integrating your products, but has become a central piece of infrastructure. Many of these jobs depend on the other jobs to be successfully completed. So you would want to proceed with a certain action only if the previous action is completed or do a different action. So you can either create a massive job or create small jobs which are linked together. If you go with creating many small jobs, you can re-execute that job(read as redo that action), and not do the previous related jobs to it. In other words, you can start from a failing step rather than from the very initial stage. So it's always

advisable to create multiple small jobs and link them together to create a pipeline.

In this Scenario, we'll learn how to create linking between different jobs or projects in Jenkins. For our learning purposes, we have created two small jobs named 'Job 01' and 'Job 02' in Jenkins server:



The screenshot shows the Jenkins job list interface. At the top right is a link 'add description'. Below it is a table with columns: S, W, Name, Last Success, Last Failure, and Last Duration. There are two jobs listed: 'Job 01' and 'Job 02'. 'Job 01' has a last success of '2 hr 12 min - #1' and a last duration of '3.2 sec'. 'Job 02' has a last success of '2 hr 11 min - #1' and a last duration of '0.38 sec'. Below the table, there is a link 'Icon: S M L' and a 'Legend' section with three RSS links: 'RSS for all', 'RSS for failures', and 'RSS for just latest builds'.

S	W	Name	Last Success	Last Failure	Last Duration
		Job 01	2 hr 12 min - #1	N/A	3.2 sec
		Job 02	2 hr 11 min - #1	N/A	0.38 sec

Icon: [S](#) [M](#) [L](#)

Legend [RSS for all](#) [RSS for failures](#) [RSS for just latest builds](#)

Two Small Jobs for creating linking

These two jobs contain nothing but a simple echo statement echoing the name of the job. One to One Relationship between Jobs. The first approach is pretty simple: one job triggers another one after successful completion. To achieve it, you can either configure a post-build action starting the next job (downstream) or configure the trigger of the current build (upstream). Both ways are totally equivalent. In order to this, let's go to job 02 and select configure from dropdown:



Select Configure from black triangle next to job name.

Now go to 'Build triggers' and select 'Build after other projects are built'. Under 'Projects to Watch' to select 'job 01':

Build Triggers

☐ Trigger builds remotely (e.g., from scripts)

☒ Build after other projects are built

Projects to watch

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

☐ Build periodically

☐ Poll SCM

Let's select 'Trigger only if build is stable' so that 'Job 02' is executed only when 'Job 01' is successfully completed. After this, select save and apply. Now when you build 'Job 01' and see the log output, it should be something like this:

Console Output

```
Started by user admin
Building remotely on Slave 01 in workspace /home/jenkins/workspace/Job 01
[Job 01] $ /bin/sh -xe /tmp/hudson7253560760777707550.sh
+ echo 'this is project 01'
this is project 01
Warning: you have no plugins providing access control for builds, so falling back to legacy behavior of permitting any
downstream builds to be triggered
Triggering a new build of Job 02
Finished: SUCCESS
```

Job 01 build output after configuring upstream for job 02

If you look at the last 2nd line, you can see that it started a new build of 'Job 02'. Since it contains a link, instead of going to the dashboard, you can directly click on the link to go to job. It contains a new section named 'Upstream Projects' and would have listed 'job 01' as the build. Again from the build history pane, you can see the recent build of this 'Job 02'.

Using this one-to-one dependency approach is probably the simplest way to orchestrate jobs on Jenkins.

One to Many Relationship between Jobs

You can also create a master job and associate various child jobs to it. So let's say we have 'Job 01' as a master job and 'Job 02', 'Job 03', 'Job 04' as child jobs. Again, we'll use the same method to create links between jobs as above and initiate build.



Console Output

Success

```
Started by user admin
Building remotely on Slave 01 in workspace /home/jenkins/workspace/Job 01
[Job 01] $ /bin/sh -xe /tmp/hudson1391137972907823814.sh
+ echo 'this is job 01'
this is job 01
Warning: you have no plugins providing access control for builds, so falling back to legacy behavior of permitting any
downstream builds to be triggered
Triggering a new build of Job 02
Triggering a new build of Job 03
Triggering a new build of Job 04
Finished: SUCCESS
```

Job 01 build output after configuring upstream for job 02, job 03 and job 04.

Kindly keep below points in mind, before going for one-to-many relationship:

1. If one of the child jobs fails, other child jobs will still be triggered. They may or may not have some inter-dependency later in the flow. Of Course, you can use the success of multiple jobs as the trigger as well. So you need to plan it carefully.
2. There is no order defined for calling the child jobs. This is because Jenkins threads are asynchronous in nature.

We can use the 'Join Plugin' for the same. This plugin allows a job to be run after all the immediate downstream jobs have completed. In this way, the execution can branch out and perform many steps in parallel, and then run a final aggregation step just once after all the parallel work is finished. You can go to Manage Jenkins -> Manage Plugins -> Available tab and search for this plugin:

Filter:

Updates Available Installed Advanced

Install	Name	Version
<input type="checkbox"/>	Self-Organizing Swarm Plug-in Modules This plugin enables slaves to auto-discover nearby Jenkins master and join it automatically, thereby forming an ad-hoc cluster.	3.3
<input type="checkbox"/>	bootstrapped-multi-test-results-report Allows you to have a pretty bootstrapped HTML report of the test results generated by the automated tests for Cucumber / JUnit / RSpec / TestNG. Join chat for more details about the plugin, or visit repo or page .	2.0.2
<input type="checkbox"/>	JobFanIn This plugin provides a watch on upstream projects to trigger downstream projects once all the upstream projects are build & have stable status. Easier to join multiple project to trigger single downstream project. This plugin can be used with Build Pipeline, Delivery Pipeline etc. It solves complexity of merging pipeline flows from multiple branches to single.	1.1.3
<input type="checkbox"/>	Join plugin This plugin allows a job to be run after all the immediate downstream jobs have completed.	1.21

Update information obtained: 2 hr 15 min ago

Install Join Plugin from Manage Plugins.JPG Select

and install it.

Now Let's go to 'Job 01' and go to the post-build section. In order to build other projects, add 'Job 02' and 'Job 03'. Then add a join trigger from post build steps and enter 'Job 04'. This will ensure that 'Job 04' is executed only when both 'Job 02' and 'Job 03' are completed. Again, you can use this plugin to create multiple diamond shape dependencies like above. So this helps in creating complex pipelines.