

JENKINS

Table of Contents

What is Jenkins?.....	2
What is Continuous Integration?.....	2
Jenkins History.....	3
Why use Continuous Integration with Jenkins?.....	3
Real-world case study of Continuous Integration.....	4
Jenkins Plugins.....	5
Advantages of using Jenkins.....	6
Disadvantages of using Jenkins.....	7
Conclusion:.....	7
Construire imagine Jenkins.....	8
Lansare container myjenk cu volum persistent.....	8
rezultat consola.....	10

What is Jenkins?

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.

Jenkins is free and is entirely written in Java. Jenkins is a widely used application around the world that has around 300k installations and growing day by day.

It is a server-based application and requires a web server like Apache Tomcat. The reason Jenkins became so popular is that of its monitoring of repeated tasks which arise during the development of a project. For example, if your team is developing a project, Jenkins will continuously test your project builds and show you the errors in early stages of your development.

By using Jenkins, software companies can accelerate their software development process, as Jenkins can automate build and test at a rapid rate. Jenkins supports the complete development lifecycle of software from building, testing, documenting the software, deploying and other stages of a software development lifecycle.

In this tutorial, you will learn

- [What is Jenkins?](#)
- [What is Continuous Integration?](#)
- [Jenkin History](#)
- [Why use Continuous Integration with Jenkins?](#)
- [Real-world case study of Continuous Integration](#)
- [Advantages of using Jenkins](#)
- [Disadvantages of using Jenkins](#)

What is Continuous Integration?

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.

A Continuous Integration Pipeline is a powerful instrument that consists of a set of tools designed to **host**, **monitor**, **compile** and **test** code, or code changes, like:

- **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)

Jenkins History

- Kohsuke Kawaguchi, a Java developer, working at SUN Microsystems, was tired of building the code and fixing errors repetitively. In 2004, created an automation server called Hudson that automates build and test task.
- In 2011, Oracle who owned Sun Microsystems had a dispute with Hudson open source community, so they forked Hudson and renamed it as Jenkins.
- Both Hudson and Jenkins continued to operate independently. But in short span of time, Jenkins acquired a lot of projects and contributors while Hudson remained with only 32 projects. With time, Jenkins became more popular, and Hudson is not maintained anymore.

Why use Continuous Integration with Jenkins?

Some people might think that the old-fashioned way of developing the software is the better way. Let's understand the advantages of CI with Jenkins with the following example

Let us imagine, that there are around 10 developers who are working on a shared repository. Some developer completes their task in 25 days while others take 30 days to complete.

Before Jenkins

Once all Developers had completed their assigned coding tasks, they used to commit their code all at once.

After Jenkins

The code is built and test as soon as Developer commits code. Jenkin will build and test code

same time. Later, Build is tested and deployed.

Code commit built, and test cycle was very infrequent, and a single build was done after many days.

Since the code was built all at once, some developers would need to wait until other developers finish coding to check their build

It is not an easy task to isolate, detect, and fix errors for multiple commits.

Code build and test process are entirely manual, so there are a lot of chances for failure.

The code is deployed once all the errors are fixed and tested.

Development Cycle is slow

many times during the day

If the build is successful, then Jenkins will deploy the source into the test server and notifies the deployment team.

If the build fails, then Jenkins will notify the errors to the developer team.

The code is built immediately after any of the Developer commits.

Since the code is built after each commit of a single developer, it's easy to detect whose code caused the build to fail

Automated build and test process saving timing and reducing defects.

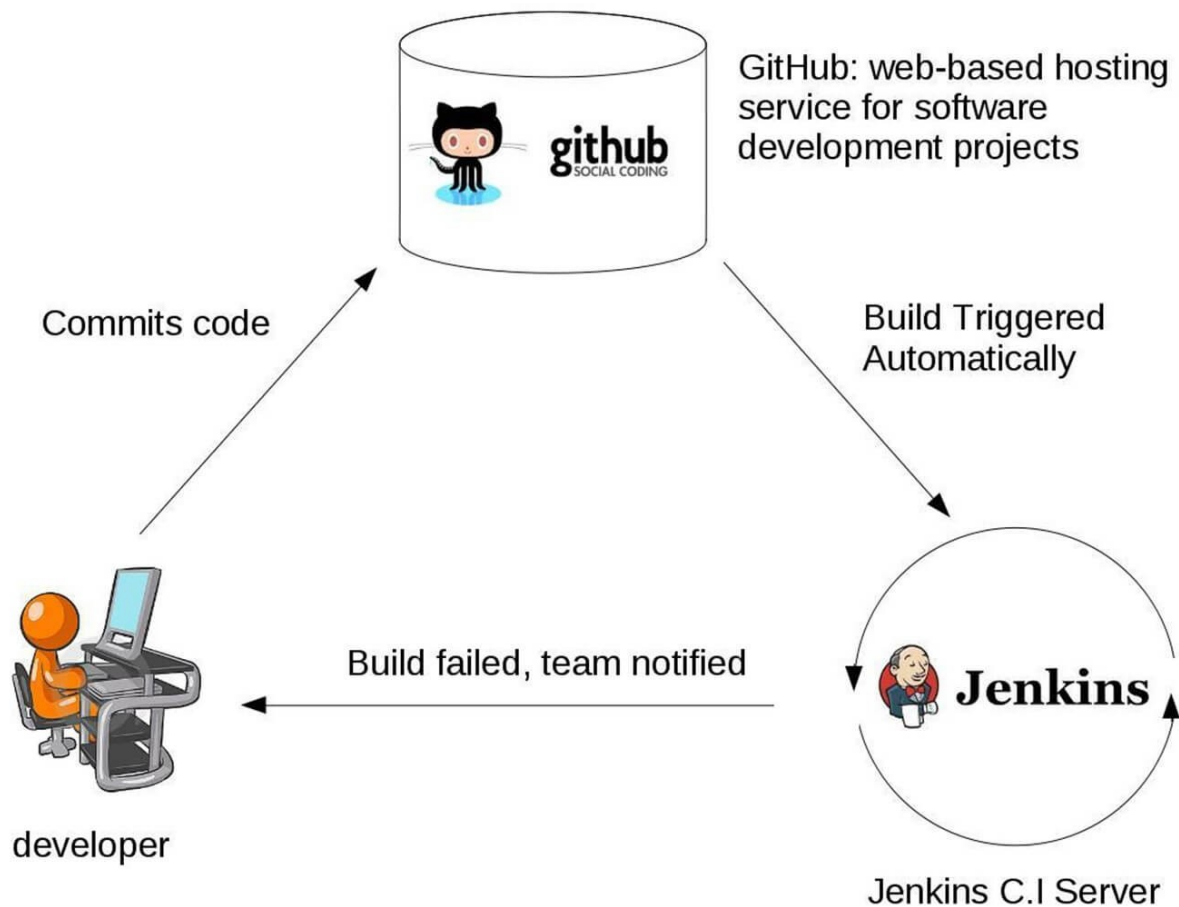
The code is deployed after every successful build and test.

The development cycle is fast. New features are more readily available to users. Increases profits.

Real-world case study of Continuous Integration

I am sure all of you aware of old phone Nokia. Nokia used to implement a procedure called nightly build. After multiple commits from diverse developers during the day, the software built every night. Since the software was built only once in a day, it's a huge pain to isolate, identify, and fix the errors in a large code base.

Later, they adopted Continuous Integration approach. The software was built and tested as soon as a developer committed code. If any error is detected, the respective developer can quickly fix the defect.



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.

Jenkins

[Jenkins](#)

[New Job](#)

[Manage Jenkins](#)

[People](#)

[Build History](#)

Build Queue

No builds in the queue.

Build Executor Status

#	Status
1	Idle
2	Idle

Manage Jenkins

[Configure System](#)
 Configure global settings and paths.

[Reload Configuration from Disk](#)
 Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files di

[Manage Plugins](#)
 Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

[System Information](#)
 Displays various environmental information to assist trouble-shooting.

[System Log](#)
 System log captures output from java.util.logging output related to Jenkins.

[Load Statistics](#)
 Check your resource utilization and see if you need more computers for your builds.

[Jenkins CLI](#)
 Access/manage Jenkins from your shell, or from your script.

[Script Console](#)
 Executes arbitrary script for administration/trouble-shooting/diagnostics.

[Manage Nodes](#)
 Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

[Install as Windows Service](#)
 Installs Jenkins as a Windows service to this system, so that Jenkins starts automatically when the machine boots.

[Prepare for Shutdown](#)
 Stops executing new builds, so that the system can be eventually shut down safely.

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.

Conclusion:

- In Continuous Integration, after a code commit, the software is built and tested immediately
- Jenkins is an open source Continuous Integration server capable of orchestrating a chain of actions
- Before Jenkins when all Developers had completed their assigned coding tasks, they used to commit their code all at same time. Later, Build is tested and deployed.
- After Jenkins the code is built and test as soon as Developer commits code. Jenkin will build and test code many times during the day
- 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
- The biggest pros of Jenkins is that it is managed by the community which holds public meetings and take inputs from the public for the development of Jenkins projects
- The biggest con of Jenkins is that Its interface is out dated and not user friendly compared to current UI trends.

Construire imagine Jenkins

Fișierul Dockerfile

```
FROM jenkins/jenkins:lts
```

```
USER root
```

```
RUN apt-get update
```

```
RUN apt-get install sudo
```

```
RUN rm -rf /var/lib/apt/lists/*
```

```
RUN echo "jenkins ALL=NOPASSWD: ALL" >> /etc/sudoers
```

```
RUN groupadd -g 999 docker
```

```
RUN usermod -a -G root jenkins
```

```
RUN usermod -a -G docker jenkins
```

```
USER jenkins
```

```
COPY plugins.txt /usr/share/jenkins/plugins.txt
```

```
RUN /usr/local/bin/install-plugins.sh < /usr/share/jenkins/plugins.txt
```

```
# RUN /usr/local/bin/plugins.sh /usr/share/jenkins/plugins.txt
```


`docker build -t myjenk .`

```
v@v-Latitude-E6430s:~/exJenkins/DinDocker$ docker build -t myjenk .
Sending build context to Docker daemon 5.632kB
Step 1/12 : FROM jenkins/jenkins:lts
--> 7e250da768ed
Step 2/12 : USER root
--> Using cache
--> 5b047d5b8463
Step 3/12 : RUN apt-get update
--> Using cache
--> 15aadb33afd6
Step 4/12 : RUN apt-get install sudo
--> Using cache
--> 65fcd21e78f4
```

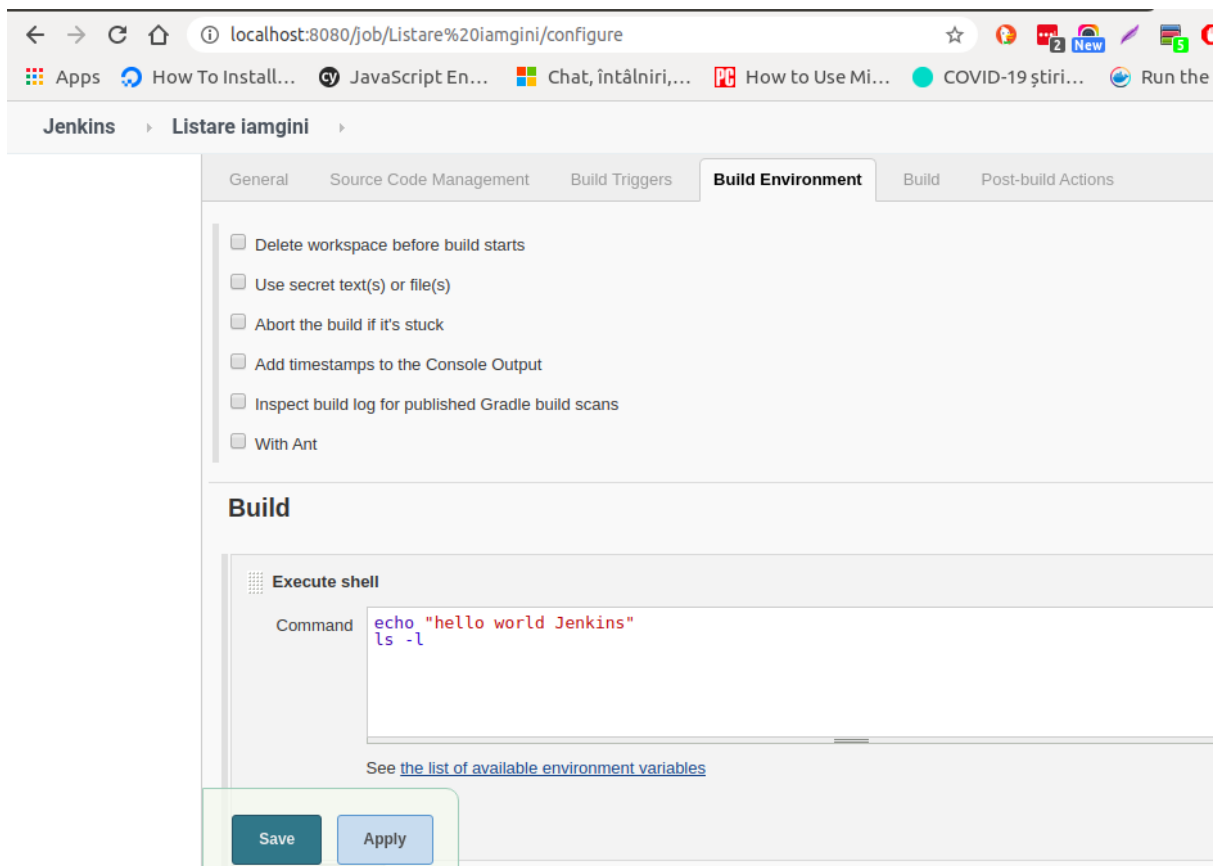
Lansare container myjenk cu volum persistent

`docker run -d -v /var/run/docker.sock:/var/run/docker.sock \`
`-v $(which docker):/usr/bin/docker -v /home/v/exJenkins/temp:/var/jenkins_home -p 8080:8080`
`myjenk`

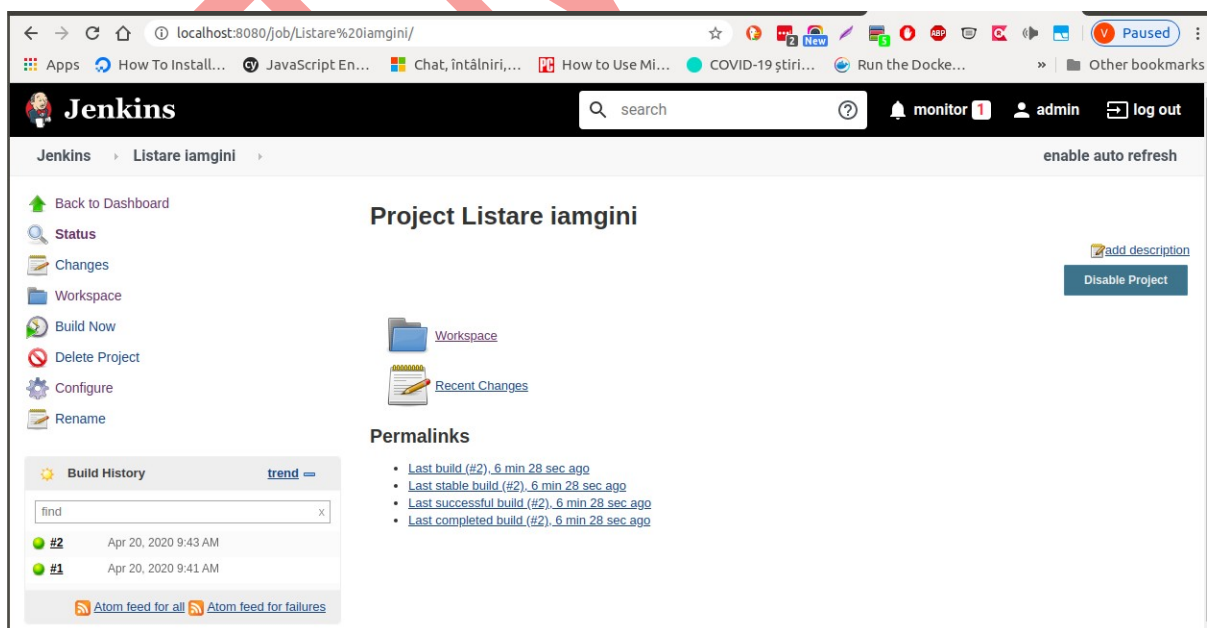
`docker run -d -v /var/run/docker.sock:/var/run/docker.sock -v $(which`
`docker):/usr/bin/docker -v /home/v/exJenkins/temp:/var/jenkins_home -p 8080:8080`
`jenkins/jenkins:lts`

user admin și parola 1234 (modificate înainte)

construire new job hello-world



se executa build



rezultat consola

The screenshot shows the Jenkins web interface for build #3 of the 'Listare iamgini' job. The console output displays the build process, including the user 'admin', the system 'SYSTEM', and the workspace path. It shows the execution of a shell script that prints 'hello world Jenkins' and lists Docker images. A table of Docker images is displayed, showing repository names, tags, image IDs, creation times, and sizes.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
myjenk	latest	0c3726654662	About an hour ago	650MB
machine_snapshot_8lvpk329b0ajvoew	latest	20059be85bee	4 days ago	1.12GB
machine_snapshot_9vxy7pja7wekrtw	latest	8d8d7b407047	4 days ago	899MB
machine_snapshot_9cng84xpu19gf4vk	latest	851b160c642f	4 days ago	792MB
openjdk	15-jdk-slim	cb1b790ca4d3	10 days ago	411MB
openjdk	8-jdk-slim	24c99a4e95d3	2 weeks ago	284MB
openjdk	11-jdk-slim	f74c1fc08a73	2 weeks ago	401MB
jenkins/jenkins	lts	7e250da768ed	3 weeks ago	619MB
hairyhenderson/figlet	latest	279c09cb45df	4 weeks ago	6.26MB
portainer/portainer	latest	10383f5b5720	2 months ago	78.6MB
alpine	latest	e7d92cdc71fe	3 months ago	5.59MB
openjdk	9-jdk-slim	339dc16e8d08	23 months ago	374MB
codenvy/ubuntu_python	latest	f8ae496bc78e	3 years ago	890MB
eclipse/che-server	5.0.0-latest	942570307823	3 years ago	274MB
codenvy/ubuntu_jdk8	latest	4074bfc5705b	3 years ago	668MB

Fisiere necesare

```
v@v-latitude-e6430s:~/exJenkins/DinDocker$ tree
.
├── dindocker.txt
├── Dockerfile
├── Jenkinsfile
└── plugins.txt

0 directories, 4 files
```

The screenshot shows a file manager window with the 'DinDocker' directory selected. The directory contains four files: dindocker.txt, Dockerfile, Jenkinsfile, and plugins.txt. The table below lists these files with their names, sizes, and modification dates.

Name	Size	Modified
dindocker.txt	284 bytes	20 iul 2019
Dockerfile	383 bytes	20 iul 2019
Jenkinsfile	752 bytes	17 iul 2019
plugins.txt	62 bytes	16 iul 2019

Dockerfile

```
FROM jenkins/jenkins:lts
```

```
USER root
```

```
RUN apt-get update
```

```
RUN apt-get install sudo
```

```
RUN rm -rf /var/lib/apt/lists/*
```

```
RUN echo "jenkins ALL=NOPASSWD: ALL" >> /etc/sudoers
```

```
RUN groupadd -g 999 docker
```

```
RUN usermod -a -G root jenkins
```

```
RUN usermod -a -G docker jenkins
```

```
USER jenkins
```

```
COPY plugins.txt /usr/share/jenkins/plugins.txt
```

```
RUN /usr/local/bin/install-plugins.sh < /usr/share/jenkins/plugins.txt
```

UTM2022

Jenkinsfile

```
node {
    def app

    stage('Clone repository') {
        /* Cloning the Repository to our Workspace */

        checkout scm
    }

    stage('Build image') {
        /* This builds the actual image */

        app = docker.build("vi111/myjenk")
    }

    stage('Test image') {

        app.inside {
            echo "Tests passed"
        }
    }

    stage('Push image') {
        /*
            You would need to first register with DockerHub before you can push images
            to your account
        */
        docker.withRegistry('https://registry.hub.docker.com', 'docker-hub') {
            app.push("${env.BUILD_NUMBER}")
            app.push("latest")
        }
        echo "Trying to Push Docker Build to DockerHub"
    }

}
}
```

plugins.txt

scm-api:latest
git-client:latest
git:latest
greenballs:latest

UTM2022

UTM2022