

# JENKINS

*An open-source CI/CD software*



e-Portfolio

Software Engineering, Berkling,  
TINF18B2, DHBW-KA

*Sebastian Momann*

## 1. Installation

The installation of Jenkins is straight forward and quite simple. Visit the [Jenkins download section](#) and download the appropriate installer. After that, open **localhost** in your browser and follow the installation wizard. Your Jenkins is now ready to use.

Alternatively, Jenkins is installable as Docker container. Read this [Blog](#) for more information about this process.

**NOTE: Jenkins requires JAVA 8 to function**

## 2. Initial problem

In the “ancient” way of programming and deployment, there was a fixed date or period, when all parts of a program come together to form the complete application. This required programmers to hand in their work at a specific time and date. When the time arrived, a process was started to connect all individual parts to form the final application. This process is also known as “building” (an application).

The problem with this technique was, that after the build, almost every time the programmers needed to fix some parts of the application, because some components do not work in parallel or cannot properly connect to the other modules. This caused huge amount of trouble, because the programmers of a module needed, to exchange information with the creators of another module, with which they wanted to connect. This cycle of building, communicating and fixing needed to be iterated very often until the final application worked as expected. As you can imagine, this is extremely time intensive, and so an alternative needed to form.

This is where the so called “nightly builds” came in place. Instead of trying to build the application e.g. four weeks before every release, the build gets started every single night. This means, that after each day of development, every working piece of code gets committed to e.g. git, in order to allow the nightly build with the latest code changes. On the next day, the results are on the table, allowing each responsible programmer to fix their part of the code, if the build failed due to malicious piece of code.

The advantage of this variation of building a software is obvious. Upon changing a code piece, on the next day, the programmer knows for sure, if it works within the application or not. With this process, rescheduling of release dates is almost not present — at least the rescheduling due to faulty builds.

But now, there is an even more advanced type of building an application. Basically, the application is built all the time. It is built “continuously” — hence the name “Continuous Integration (CI)”. With CI, the programmer can commit his code changes, a build starts, and sometimes when he is coming back from a coffee break he can already see, whether his changes are working as expected in the application context.

## 3. Jenkins as problem solver

This is where Jenkins comes to play. Jenkins allows programmers to build their application whenever they want. This means, that a user can manually run a build to see if everything looks fine, but can also rely on the automatic build start, after each change in the remote code repository (e.g. github). This is a crucial step to the secrets of CI/CD. CD stands for Continuous Delivery and is also an important feature that Jenkins offers.

Like mentioned, the way of building an application changed from fixed dates — e.g. four times a year — to having the ability to release on every successful build. That means daily! Of course, then you also need some sort of mechanism to ensure, that the freshly built application gets publicly available. Jenkins is also good for this issue. Instead of having a low paid worker to release the project after each

successful build, Jenkins offers the possibility to automatically deploy the latest build. In extreme cases, due to continuously building, this can happen several times a day.

#### 4. Why Jenkins?

There are many types of applications out there, that would profit from being continuously built and deployed. But this also means, that a software dedicated to ensuring proper CI/CD, needs to be capable of handling various types of applications and languages.

The cool thing about Jenkins is, that is easily extensible. This means, that for the most requirements not built into Jenkins directly, a proper extension can be added to the instance. Like that, almost every type of application can be managed via Jenkins.

And – guess what – it is web based!

Its installation process is fast and easy, it covers all your needs, is easy to use and its free.

#### 5. Managing websites via CI/CD

To give you an example on what types of applications benefit from CI/CD, we will look at web pages. When building an application that needs to be installed on your machine, an update to the software code requires to download the files and updating it on the machine, or in the worst case you need to completely install the new application.

With web pages, the part of updating the application on the user's machine itself is not necessary, due to the server client architecture. Then the developers decide to update the application, it can instantly be built and deployed to the web server, without any other steps. And just like that, the latest software version is accessible to the user, sometimes without them even realizing. In most extreme cases, a new version of websites is deployed 50 times a day. Remember the old concept of releasing a piece of software four times a year?

#### 6. CI but please not CD

Well, of course some applications cannot be deployed multiple times a day, especially if the application is an installable program (Or would you like to update e.g. your game after every played match?). But this is also no problem for Jenkins. Jenkins can be configured to build your application, maybe create a snapshot, and then kind of “forgetting” what happened. This is used if you still want to keep a type of release cycle. Build the application if something changed, test if all is working correctly, maybe keep the build for versioning, but do not release it.

When it is time to release the new software start the build manually with an extra checkbox checked that is called “release”. Only if this checkbox is checked, after building and testing, the new software gets released to the public.

#### 7. More Advantages

Before diving into the Jenkins-Software itself, another advantage of CI/CD management with Jenkins is, that you can add custom steps to your pipeline. Remember extensions? That is a way to use them. Upon having a large project, several people will handle the code. This means, that every user is writing in their own code style, resulting in confusion, which in some cases increases the time spend on a task. To encounter this problem, an extension for code formatting can be included into the pipeline. Also, code analytics and many more. Just imagine the craziest thing you want to do within your pipeline. I bet there is an extension for that.

## LIVE – DOMEONSTRATION (This are only the most important things I will show)

### 1. Dashboard

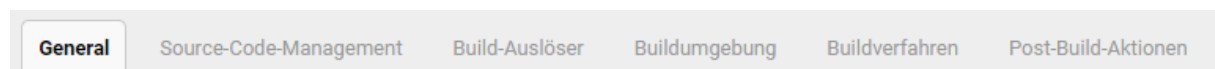


S	W	Name ↓	Letzter Erfolg	Letzter Fehlschlag	Letzte Dauer	Fav
✓	🔧	anmeldesystem-backend	24 Tage - #10	24 Tage - #9	1 Minute 34 Sekunden	🔍 ☆
📄	🔧	anmeldesystem-backend_pull_request	3 Monate 4 Tage - log	Unbekannt	2.7 Sekunden	🔍 ☆
!	🔧	anmeldesystem-frontend	Nicht anwendbar	1 Monat 4 Tage - #3	19 Sekunden	🔍 ☆
✓	⚙️	EmailPush	25 Tage - #38	6 Monate 11 Tage - #31	20 Sekunden	🔍 ☆
✓	🔧	EmailPushDocker	24 Tage - #30	24 Tage - #29	20 Sekunden	🔍 ☆
✓	⚙️	Respawn	3 Monate 12 Tage - #116	Unbekannt	2.6 Sekunden	🔍 ☆
!	🔧	test	Nicht anwendbar	24 Tage - #7 example	0.64 Sekunden	🔍 ☆

On the dashboard you can see every project running on this instance. It also displays important information like the last successful build or build stability.

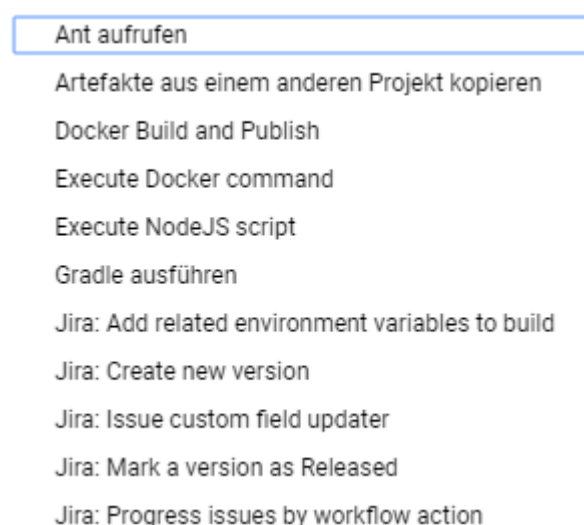
### 2. Freestyle Project

Upon creating a new freestyle project, there are various configurations you can change.



A basic example for a pipeline is to check out the source code from e.g. github.com, run some tests, build the application and then push it to a new version branch or deploy it to a server.

The core of such a build can be created by adding build steps. This allows to do various tasks like running a docker command, building a node.js application, printing information, running a test framework and so on. With this you can also deploy the built application.



After each build, there are steps like publishing test results, update JIRA and other stuff. Those steps are called "Post-Actions".

Artefakte archivieren  
Fingerabdrücke von Dateien aufzeichnen, um deren Verwendung zu verfolgen  
Javadoc veröffentlichen  
Jira: Update relevant issues  
Nachgelagerte Testergebnisse zusammenfassen  
Publish HTML reports  
Stop and remove Docker container  
Veröffentliche JUnit-Testergebnisse.  
Weitere Projekte bauen  
Git Publisher  
E-Mail-Benachrichtigung

#### Editable Email Notification

Jira: Create issue  
Jira: Create new version  
Jira: Mark a version as Released  
Jira: Move issues matching JQL to the specified version  
Set GitHub commit status (universal)  
Set build status on GitHub commit [deprecated]  
Arbeitsbereich nach dem Build löschen

Note that the listed options are mixed. This means that they not just include Jenkins built-in options, but also options provided by extensions. If you want to have an option that is not listed, install the appropriate plugin and you are good to go.

### 3. Pipeline

Instead of configuring the Jenkins job on the web app, Jenkins allows the usage of a **Jenkinsfile**. This file is a structure of the pipeline that Jenkins should execute.

See <https://github.com/sebamomann/anmeldesystem-backend/blob/tests/Jenkinsfile> for a functioning **Jenkinsfile** of my private project.

The advantage of Jenkins files is, that when all libraries are installed on the Jenkins-Server, no further requirements are needed to run a build with this configuration. Create a pipeline project, tell Jenkins to check out the code and to run the **Jenkinsfile** included in the code. It is that simple.

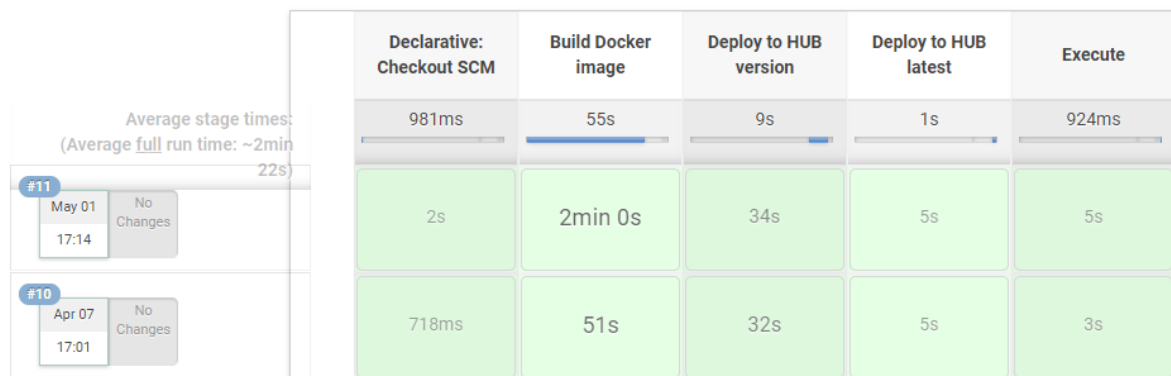
The obvious advantage is, that on some branches, the **Jenkinsfile** might differ. But by using the pipeline project, you do not need to create an own project for every branch, but one project, that uses the appropriate **Jenkinsfile** from the provided code.

Now you also do not need to give special permissions to the Jenkins user, because he does not need to change the project. In most cases he just needs to start builds and read the results.

### 4. Information output

The cool thing about pipelines is that you have a nice and structured overview about what is happening during the build.

## Stage View



In my project you can see that the Jenkins job consists of 5 steps. Getting the code, building a Docker image, deploy the set version to a Docker-Repository (also as latest version if checked) and spinning up the docker container. In this case, the Jenkins server is the same server I am hosting this project on.

For advanced output, Jenkins provides a console.



## ✓ Konsolenausgabe

```
Started by user Sebastian Momann
Obtained Jenkinsfile from git https://github.com/sebamomann/anmeldesystem-backend.git
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/anmeldesystem-backend
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
using credential sebamomann-git
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/sebamomann/anmeldesystem-backend.git # timeout=10
Fetching upstream changes from https://github.com/sebamomann/anmeldesystem-backend.git
> git --version # timeout=10
using GIT_ASKPASS to set credentials git
> git fetch --tags --progress -- https://github.com/sebamomann/anmeldesystem-backend.git +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 98fc3e63ec6cf9f4c257d13dea4274055433e162 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f 98fc3e63ec6cf9f4c257d13dea4274055433e162 # timeout=10
Commit message: "npm install nodemailer"
> git rev-list --no-walk 98fc3e63ec6cf9f4c257d13dea4274055433e162 # timeout=10
```

At the end, if all worked correctly, there should be this piece of information.

```
[Pipeline] End of Pipeline
Finished: SUCCESS
```


## Tutorial – A simple pipeline

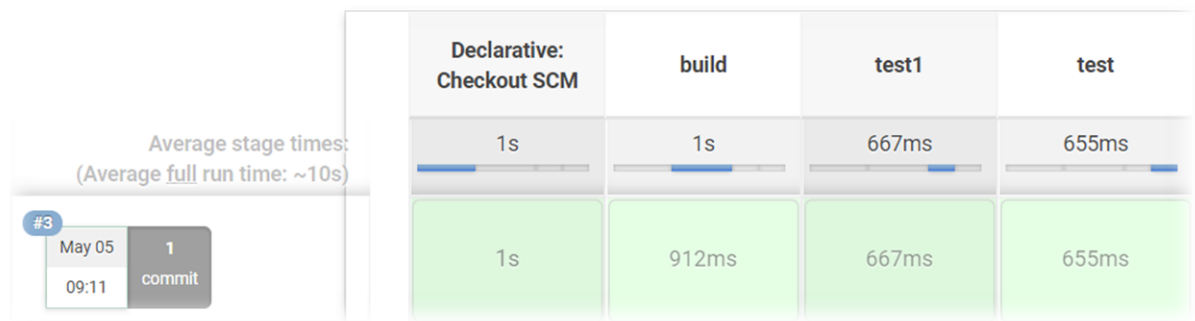
1. Head to  Element anlegen and create a new pipeline Project 

2. Goto the **PIPELINE** at your Jenkins Job, and include the script provided in <https://github.com/sebamomann/se/blob/master/Jenkinsfile>

Alternatively change the **Definition** from “Pipeline script” to “Pipeline script from SCM”

- Select git as **SCM** and paste <https://github.com/sebamomann/se.git> into the **Repository URL** field. As it is a public repository, you do not need to include any credentials
- Leave the rest as it is, so the Jenkinsfile from the root path will be executed upon running the build

3. Hit  Jetzt bauen and you should see



**Your first pipeline has been successfully created**