Jenkins

Raúl Estrada

Octubre 2020

Installing without Docker

For the reasons mentioned earlier, the Docker installation is recommended. However, if it's not an option or there are other reasons to proceed otherwise, then the installation process is just as simple. As an example, in case of Ubuntu, it's enough to run:

```
$ wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key
add -
$ sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'
$ sudo apt-get update
$ sudo apt-get install jenkins
```

All installation guides (Ubuntu, Mac, Windows, and others) can be found on the official Jenkins page `https://jenkins.io/doc/book/getting-started/installing/`.

Initial configuration

No matter which installation you choose, the first start of Jenkins requires a few configuration steps. Let's walk through them step by step:

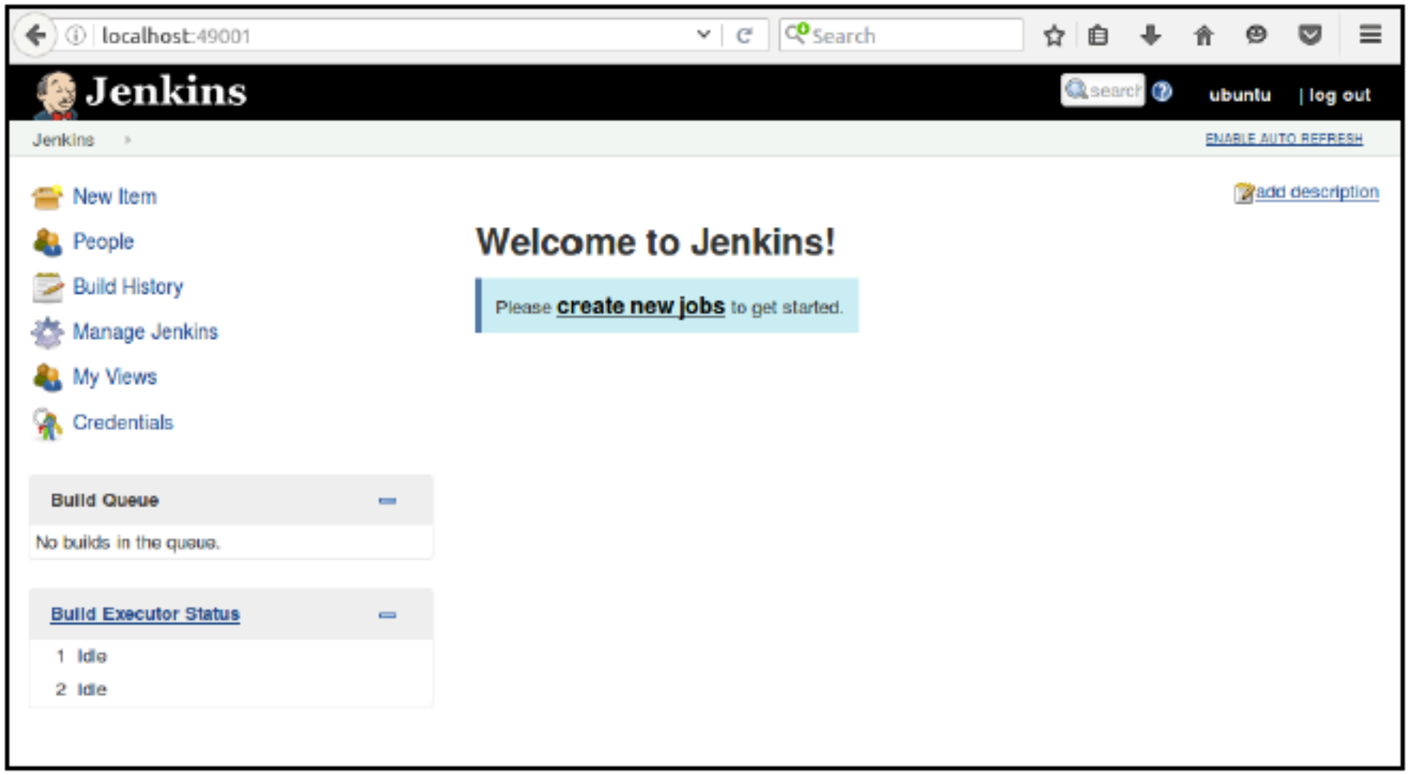
1. Open Jenkins in the browser: `http://localhost:49001` (for binary installations, the default port is `8080`).
2. Jenkins should ask for the administrator password. It can be found in the Jenkins logs:

```
$ docker logs jenkins
...
Jenkins initial setup is required. An admin user has been created
and a password generated.
Please use the following password to proceed to installation:

c50508effc6843a1a7b06f6491ed0ca6
```

3. After accepting the initial password, Jenkins asks whether to install the suggested plugins, which are adjusted to the most common use cases. Your answer depends, of course, on your needs. However, as the first Jenkins installation, it's reasonable to let Jenkins install all the recommended plugins.
4. After the plugin installation, Jenkins asks to set up username, password, and other basic information. If you skip it, the token from step 2 will be used as the admin password.

The installation is complete and you should see the Jenkins dashboard:



Welcome to Jenkins!

Please **create new jobs** to get started.

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

2 Idle

We are ready to use Jenkins and create the first pipeline.

Jenkins hello world

Everything in the entire IT world starts from the Hello World example.

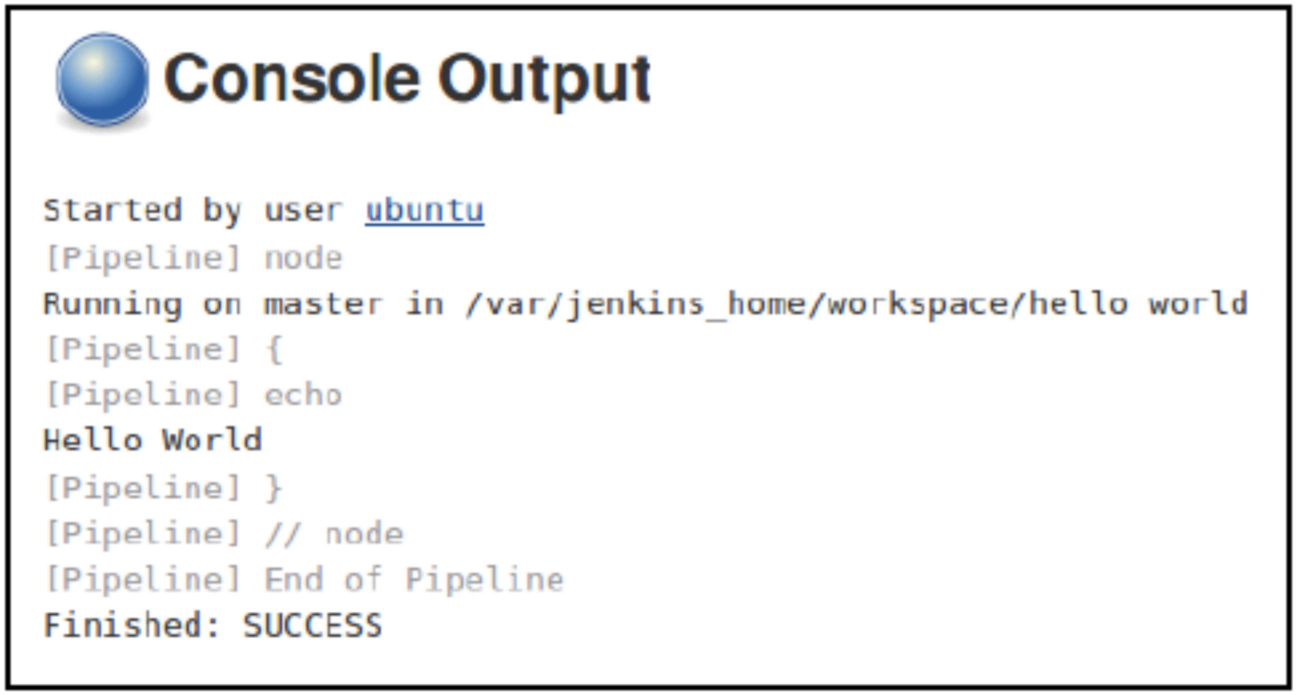
Let's follow this rule and see the steps to create the first Jenkins pipeline:

1. Click on **New Item**.
2. Enter `hello world` as the item name, choose **Pipeline**, and click on **OK**.
3. There are a lot of options. We will skip them for now and go directly to the **Pipeline** section.
4. There, in the **Script** textbox, we can enter the pipeline script:

```
pipeline {
     agent any
     stages {
          stage("Hello") {
               steps {
                    echo 'Hello World'
               }
          }
     }
}
```

5. Click on **Save**.
6. Click on **Build Now**.

We should see #1 under the **Build History**. If we click on it, and then on **Console Output**, we will see the log from the Pipeline build.



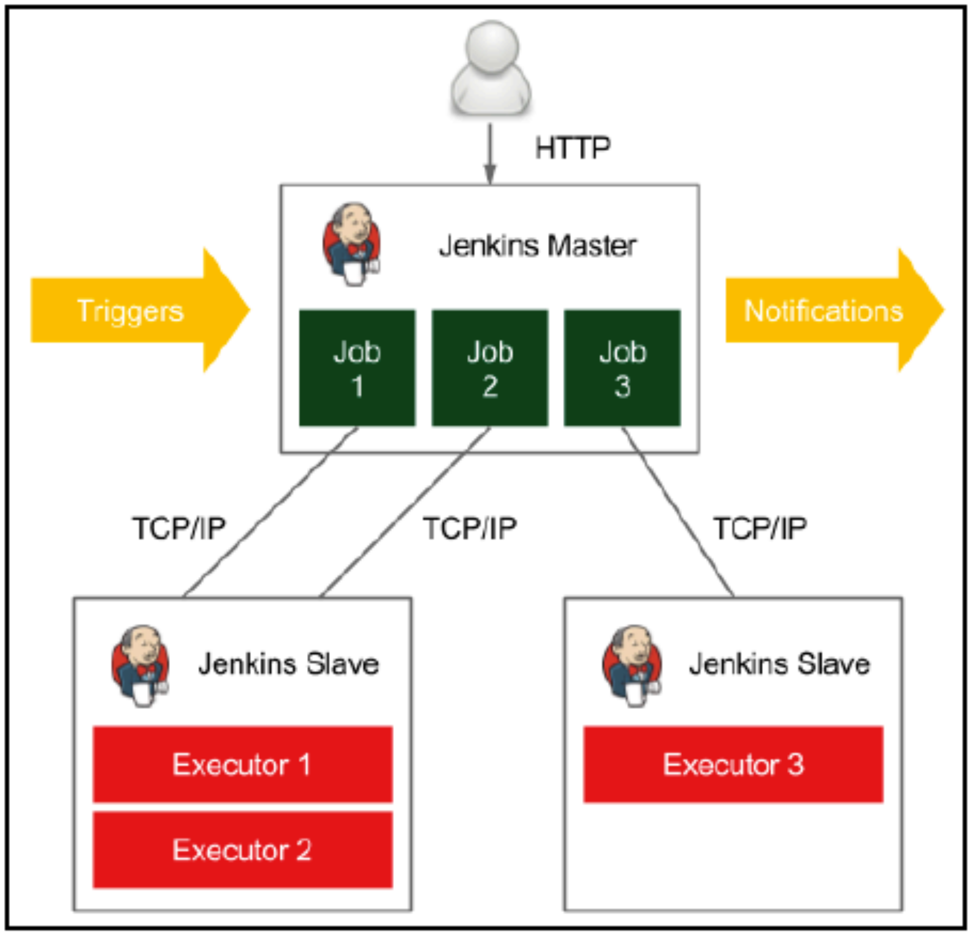
```
Started by user ubuntu
[Pipeline] node
Running on master in /var/jenkins_home/workspace/hello world
[Pipeline] {
[Pipeline] echo
Hello World
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Master and slaves

Jenkins becomes overloaded sooner than it seems. Even in case of a small (micro) service, the build can take a few minutes. That means that one team committing frequently can easily kill the Jenkins instance.

For that reason, unless the project is really small, Jenkins should not execute builds at all, but delegate them to the slave (agent) instances. To be precise, the Jenkins we're currently running is called the Jenkins master and it can delegate to the Jenkins agents.

Let's look at the diagram presenting the master-slave interaction:



In a distributed builds environment, the Jenkins master is responsible for:

- Receiving build triggers (for example, after a commit to GitHub)
- Sending notifications (for example, email or HipChat message sent after the build failure)
- Handling HTTP requests (interaction with clients)
- Managing the build environment (orchestrating the job executions on slaves)

The build agent is a machine that take care of everything that happens after the build is started.

Since the responsibilities of the master and the slaves are different, they have different environmental requirements:

- **Master**: This is usually (unless the project is really small) a dedicated machine with RAM ranging from 200 MB for small projects to 70GB plus for huge single-master projects.
- **Slave**: There are no general requirements (other than the fact that it should be capable of executing a single build, for example, if the project is a huge monolith that requires 100 GB of RAM, then the slave machine needs to satisfy these needs).

Agents should also be as generic as possible. For instance, if we have different projects: one in Java, one in Python, and one in Ruby, then it would be perfect if each agent could build any of these projects. In such a case, the agents can be interchanged, which helps to optimize the usage of resources.

If agents cannot be generic enough to match all projects, then it's possible to label (tag) agents and projects, so that the given build would be executed on a given type of agent.

Scalability

We can use Jenkins slaves to balance the load and scale up the Jenkins infrastructure. Such a process is called the horizontal scaling. The other possibility would be to use only one master node and increase resources of its machine. That process is called the vertical scaling. Let's look closer at these two concepts.

Vertical scaling

Vertical scaling means that, when the master's load grows, then more resources are applied to the master's machine. So, when new projects appear in our organization, we buy more RAM, add CPU cores, and extend the HDD drives. This may sound like a no-go solution; however, it's often used, even by well-known organizations. Having a single Jenkins master set on the ultra-efficient hardware has one very strong advantage: maintenance. Any upgrades, scripts, security settings, roles assignment, or plugin installations have to be done in one place only.

Horizontal scaling

Horizontal scaling means that, when an organization grows, then more master instances are launched. This requires a smart allocation of instances to teams and, in the extreme case, each team can have its own Jenkins master. In that case, it might even happen that no slaves are needed.

The drawbacks are that it may be difficult to automate cross-project integrations and that a part of the team's development time is spent on the Jenkins maintenance. However, the horizontal scaling has some significant advantages:

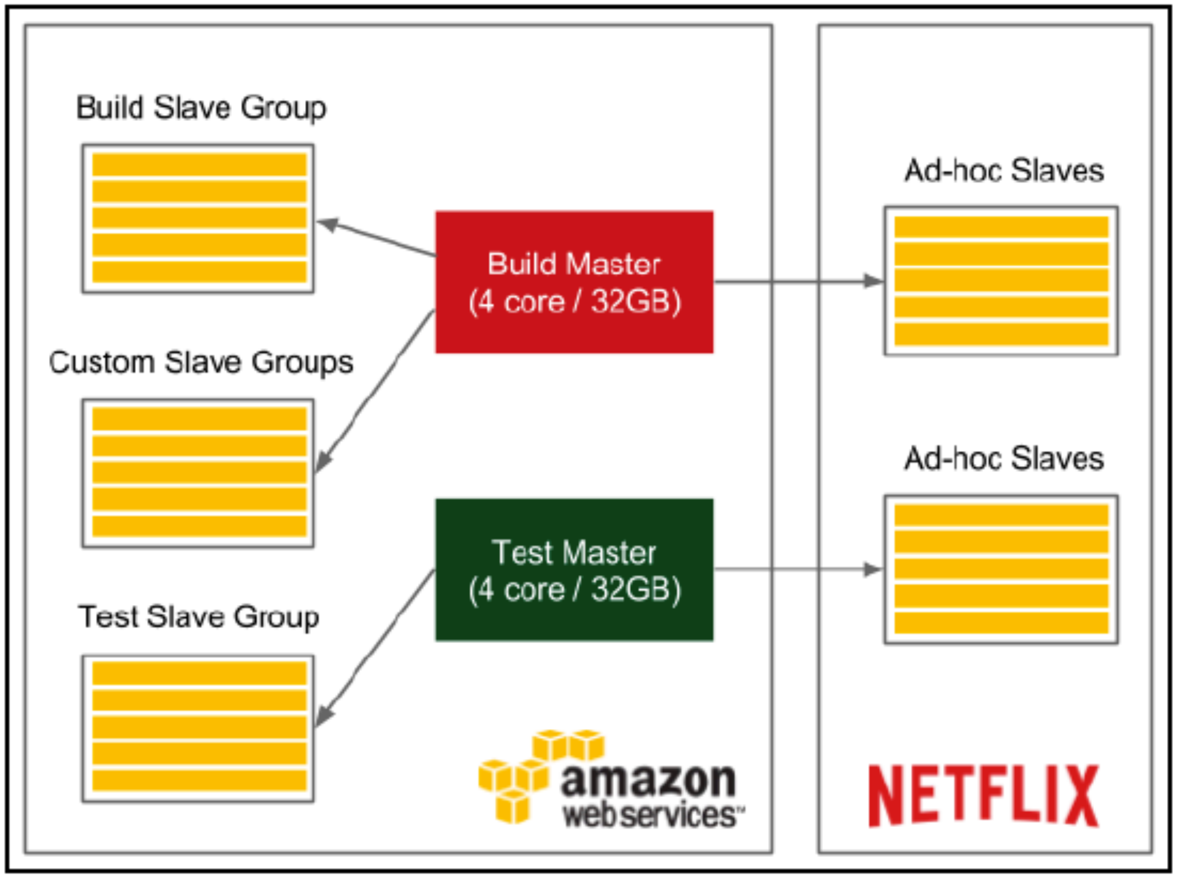
- Master machines don't need to be special in terms of hardware
- Different teams can have different Jenkins settings (for example, different set of plugins)
- Teams usually feel better and work with Jenkins more efficiently if the instance is their own
- If one master instance is down, it does not impact the whole organization
- The infrastructure can be segregated into standard and mission-critical
- Some maintenance aspects can be simplified, for example, the team of five could reuse the same Jenkins password, so we may skip the roles and security settings (surely that is possible only if the corporate network is well firewalled)

Test and production instances

Apart from the scaling approach, there is one more issue: how to test the Jenkins upgrades, new plugins, or pipeline definitions? Jenkins is critical to the whole company. It guarantees the quality of the software and (in case of Continuous Delivery) deploys to the production servers. That is why it needs to be highly available, so it is definitely not for the purpose of testing. It means there should always be two instances of the same Jenkins infrastructure: test and production.

Test environment should always be as similar as possible to the production, so it also requires the similar number of agents attached.

Let's look at the example of Netflix to have a complete picture of the Jenkins infrastructure (they shared it as the **planned infrastructure** at Jenkins User Conference San Francisco 2012):



Communication protocols

In order for the master and the agent to communicate, the bi-directional connection has to be established.

There are different options how it can be initiated:

- **SSH**: Master connects to slave using the standard SSH protocol. Jenkins has an SSH-client built-in, so the only requirement is the SSHD server configured on slaves. This is the most convenient and stable method because it uses standard Unix mechanisms.
- **Java Web Start**: Java application is started on each agent machine and the TCP connection is established between the Jenkins slave application and the master Java application. This method is often used if the agents are inside the firewalled network and the master cannot initiate the connection.
- **Windows service**: The master registers an agent on the remote machine as a Windows service. This method is discouraged since the setup is tricky and there are limitations on the graphical interfaces usage.