Jenkins Agents

Raúl Estrada

Octubre 2020

Setting agents

At the low level, agents communicate with the Jenkins master always using one of the protocols described above. However, at the higher level, we can attach slaves to the master in various ways. The differences concern two aspects:

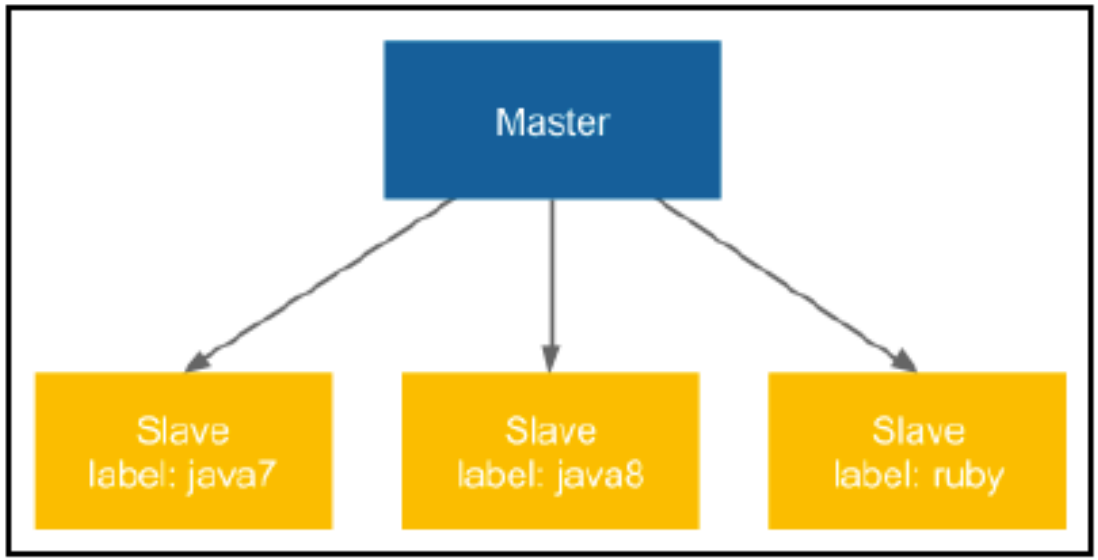
- **static versus dynamic**: The simplest option is to add slaves permanently in the Jenkins master. The drawback of such solution is that we always need to manually change something if we need more (or less) slave nodes. A better option is to dynamically provision slaves as they are needed.
- **specific versus general-purpose**: Agents can be specific (for example, different agents for the projects based on Java 7 and different agents for Java 8) or general-purpose (an agent acts as a Docker host and a pipeline is built inside a Docker container).

These differences resulted in four common strategies how agents are configured:

- Permanent agents
- Permanent Docker agents
- Jenkins Swarm agents
- Dynamically provisioned Docker agents

Understanding permanent agents

As already mentioned, the drawback of such a solution is that we need to maintain multiple slave types (labels) for different project types. Such a situation is presented in the following diagram:



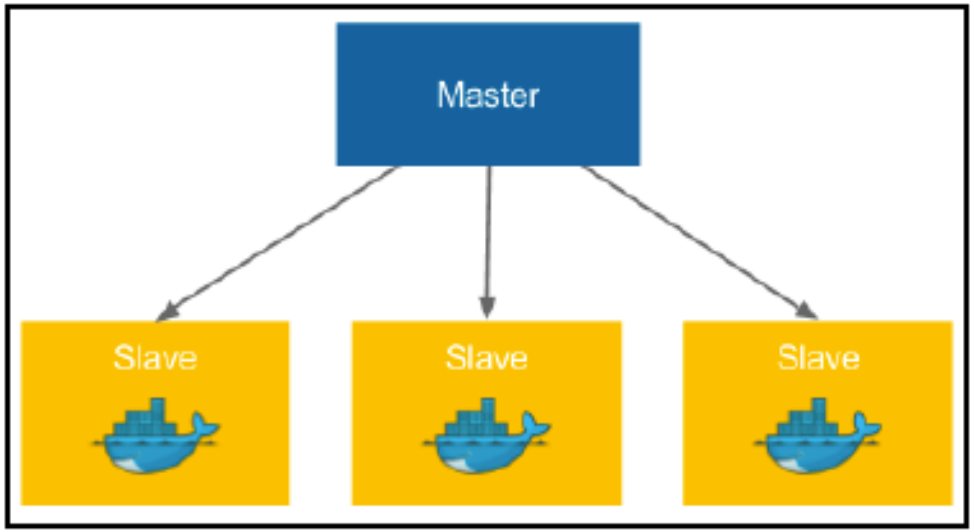
Permanent Docker agents

The idea behind this solution is to permanently add general-purpose slaves. Each slave is identically configured (Docker Engine installed) and each build is defined together with the Docker image inside which the build is run.

When the build is started, the Jenkins slave starts a container from the Docker image `openjdk:8-jdk-alpine` and then executes all pipeline steps inside that container. This way, we always know the execution environment and don't have to configure each slave separately depending on the particular project type.

Understanding permanent Docker agents

Looking at the same scenario we took for the permanent agents, the diagram looks like this:



Jenkins Swarm agents

So far, we always had to permanently define each of the agents in the Jenkins master. Such a solution, even though good enough in many cases, can be a burden if we need to frequently scale the number of slave machines. Jenkins Swarm allows you to dynamically add slaves without the need to configure them in the Jenkins master.

Configuring Jenkins Swarm agents

The first step to use Jenkins Swarm is to install the **Self-Organizing Swarm Plug-in Modules** plugin in Jenkins. We can do it via the Jenkins web UI under **Manage Jenkins** and **Manage Plugins**. After this step, the Jenkins master is prepared for Jenkins slaves to be dynamically attached.

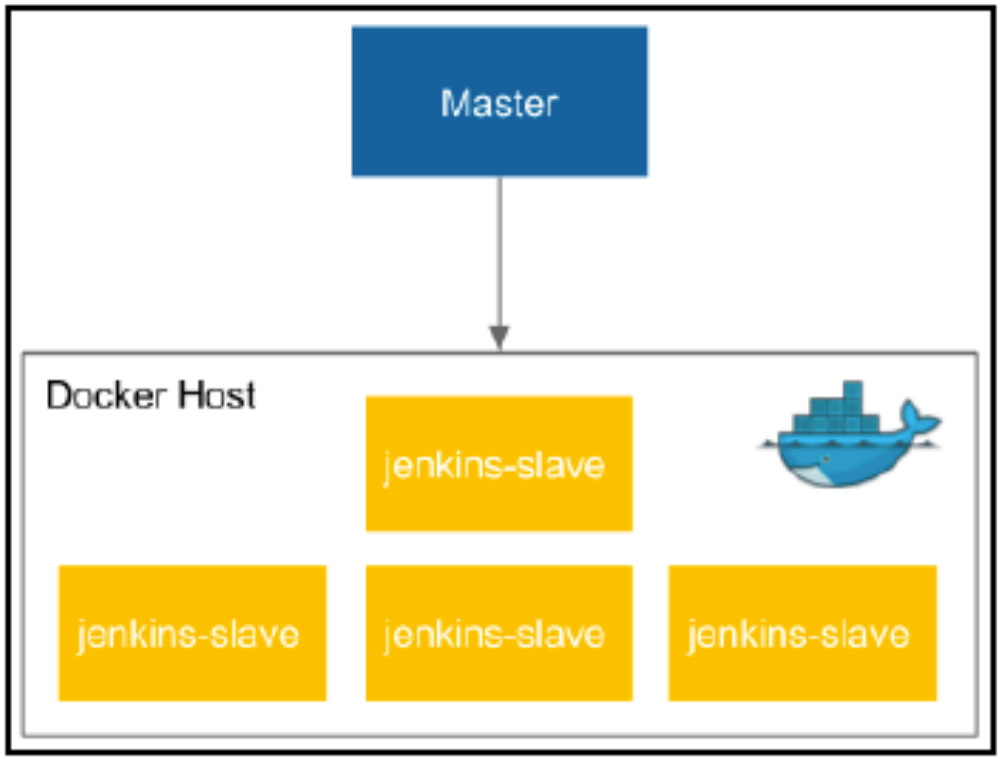
Dynamically provisioned Docker agents

Another option is to set up Jenkins to dynamically create a new agent each time a build is started. Such a solution is obviously the most flexible one since the number of slaves dynamically adjust to the number of builds. Let's have a look at how to configure Jenkins this way.

Understanding dynamically provisioned Docker agents

Dynamically provisioned Docker agents can be treated as a layer over the standard agent mechanism. It changes neither the communication protocol nor how the agent is created. So, what does Jenkins do with the Docker agent configuration we provided?

The following diagram presents the Docker master-slave architecture we've configured:



Let's describe step by step how the Docker agent mechanism is used:

1. When the Jenkins job is started, the master runs a new container from the `jenkins-slave` image on the slave Docker host.
2. The jenkins-slave container is, actually, the ubuntu image with the SSHD server installed.
3. The Jenkins master automatically adds the created agent to the agent list (same as what we did manually in the *Setting agents* section).
4. The agent is accessed using the SSH communication protocol to perform the build.
5. After the build, the master stops and removes the slave container.

Therefore, it has two great advantages as follows:

- **Automatic agent lifecycle**: The process of creating, adding, and removing the agent is automated.
- **Scalability**: Actually, the slave Docker host could be not a single machine, but a cluster composed of multiple machines (we'll cover clustering using Docker Swarm in `Chapter 8`, *Clustering with Docker Swarm*). In that case, adding more resources is as simple as adding a new machine to the cluster and does not require any changes in Jenkins.

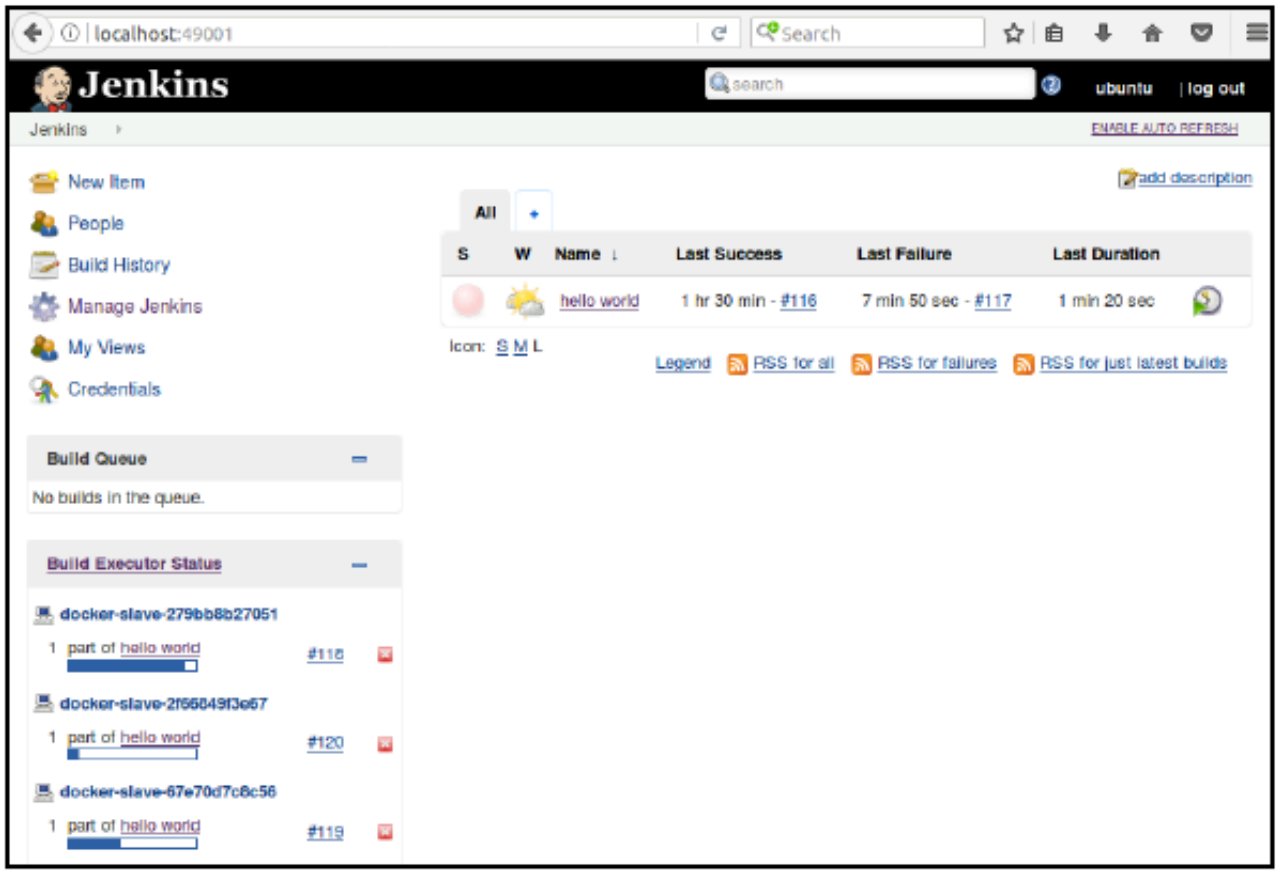
Testing agents

No matter which agent configuration you chose, we should now check if it works correctly.

Let's go back to the hello world pipeline. Usually, the builds last longer than the hello-world example, so we can simulate it by adding sleeping to the pipeline script:

```
pipeline {
     agent any
     stages {
          stage("Hello") {
               steps {
                    sleep 300 // 5 minutes
                    echo 'Hello World'
               }
          }
     }
}
```

After clicking on **Build Now** and going to the Jenkins main page, we should see that the build is executed on an agent. Now, if we click on build many times, then different agents should be executing different builds (as shown in the following screenshot):



Custom Jenkins images

So far, we have used the Jenkins images pulled from the internet. We used `jenkins` for the master container and `evarga/jenkins-slave` for the slave container. However, we may want to build our own images to satisfy the specific build environment requirements. In this section, we cover how to do it.

There are three steps to build and use the custom image:

1. Create a Dockerfile.
2. Build the image.
3. Change the agent configuration on master.

1. **Dockerfile**: Let's create a new directory inside the Dockerfile with the following content:

```
FROM evarga/jenkins-slave
RUN apt-get update && \
    apt-get install -y python
```

2. **Build the image**: We can build the image by executing the following command:

```
$ docker build -t jenkins-slave-python .
```

3. **Configure the master**: The last step, of course, is to set `jenkins-slave-python` instead of `evarga/jenkins-slave` in the Jenkins master's configuration (as described in the *Setting Docker agent* section).