Distributed builds

It is pretty common when starting with Jenkins to have a single server which runs the master and all builds, however Jenkins architecture is fundamentally "Master+Agent". The master is designed to do co-ordination and provide the GUI and API endpoints, and the Agents are designed to perform the work. The reason being that workloads are often best "farmed out" to distributed servers. This may be for scale, or to provide different tools, or build on different target platforms. Another common reason for remote agents is to enact deployments into secured environments (without the master having direct access).

Many people today use Jenkins in cloud environments, and there are plugins and extensions to support the various environments and clouds. These may involve Virtual Machines, Docker Containers, Kubernetes (for example see Jenkins-X), EC2, Azure, Google Cloud, VMWare and more. In these cases the agents are managed for you typically (and in many cases on demand, as needed), so you may not need to read the content of this document for those cases.

This document describes this distributed mode of Jenkins and some of the ways in which you can configure it, should you need to take control (or maybe you are curious)

Contents

How does this work?

A "master" operating by itself is the basic installation of Jenkins and in this configuration the master handles all tasks for your build system. In most cases installing an agent doesn't change the behavior of the master. It will serve all HTTP requests, and it can still build projects on its own. Once you install a few agents you might find yourself removing the executors on the master in order to free up master resources (allowing it to concentrate resources on managing your build environment) but this is not a necessary step. If you start to use Jenkins a lot with just a master you will most likely find that you will run out of resources (memory, CPU, etc.). At this point you can either upgrade your master or you can setup agents to pick up the load. As mentioned above you might also need several different environments to test your builds. In this case using an agent to represent each of your required environments is almost a must.

An agent is a computer that is set up to offload build projects from the master and once setup this distribution of tasks is fairly automatic. The exact delegation behavior depends on the configuration of each project; some projects may choose to "stick" to a particular machine for a build, while others may choose to roam freely between agents. For people accessing your Jenkins system via the integrated website (http://yourjenkinsmaster:8080), things work mostly transparently. You can still browse javadoc, see test results, download build results from a master, without ever noticing that builds were done by agents. In other words, the master becomes a sort of "portal" to the entire build farm.

Since each agent runs a separate program called an "agent" there is no need to install the full Jenkins (package or compiled binaries) on an agent. There are various ways to start agents, but in the end the agent and Jenkins master need to establish a bi-directional communication link (for example a TCP/IP socket) in order to operate.

Follow the Step by step guide to set up master and agent machines on Windows to quickly start using distributed builds.

Master to agent connections

The most popular ways agents are configured are via connections that are initiated from the master. This allows agents to be minimally configured and the control lives with the master. This does require that the master have network access (ingress) to the agent (typically this is via ssh). In some cases this is not desirable due to security network rules, in which case you can use Agent to master connections via "JNLP".

Agent to master connections

In some cases the agent server will not be visible to the master, so the master can not initiate the agent process. You can use a different type of agent configuration in this case called "JNLP". This means that the master does not need network "ingress" to the agent (but the agent will need to be able to connect back to the master). Handy for if the agents are behind a firewall, or perhaps in some more secure environment to do trusted deploys (as an example). See the sections below to choose the type of agent that is most appropriate for your needs.

Choosing which agent pipelines and steps run on

As you will see below, agents can be labelled. This means different part of your build, or pipeline, can be allocated to run in specific agents (based on their label). This can be useful for tools, operating systems or perhaps for security purposes (it is possible to set quite detailed access rules of what can run where, based on agent configurations). A server that runs an agent is often referred to as a "Node" in Jenkins terminology.

Different ways of starting agents

Pick the right method depending on your environment and OS that master/agents run, or if you want the connection initiated from the master or from the agent end.

Have master launch agent via ssh

Jenkins has a built-in SSH client implementation that it can use to talk to remote sshd and start an agent. This is the most convenient and preferred method for Unix agents, which normally has sshd out-of-the-box. Click Manage Jenkins, then Manage Nodes, then click "New Node." In this set up, you'll supply the connection information (the agent host name, user name, and ssh credential). Note that the agent will need the master's public ssh key copied to ~/.ssh/authorized_keys. (This is a decent howto if you need ssh help). Jenkins will do the rest of the work by itself, including copying the binary needed for an agent, and starting/stopping agents. If your project has external dependencies (like a special ~/.m2/settings.xml, or a special version of java), you'll need to set that up yourself, though. The Slave Setup Plugin may be of help.

This is the most convenient set up on Unix. However, if you are on Windows and you don't have ssh commands with cygwin for example, you can use a tool like PuTTY and PuTTYgen to generate your private and public pair of keys.

For connecting to Windows agents through cygwin sshd, see SSH agents and Cygwin for more details.

Have master launch agent on Windows

For Windows agents, Jenkins can use the remote management facility built into Windows 2000 or later (WMI+DCOM, to be more specific.) In this set up, you'll supply the username and the password of the user who has the administrative access to the system, and Jenkins will use that remotely create a Windows service and remotely start/stop them.

This is the most convenient set up on Windows, but does not allow you to run programs that require display interaction (such as GUI tests).

Note : Unlike other Node's configuration type, the Node's name is very important as it is taken as the node's address where to create the service !

Write your own script to launch Jenkins agents

If the above turn-key solutions do not provide flexibility necessary, you can write your own script to start an agent. You place this script on the master, and tell Jenkins to run this script whenever it needs to connect to an agent.

Typically, your script uses a remote program execution mechanism like SSH, or other similar means (on Windows, this could be done by the same protocols through cygwin or tools like psexec), but Jenkins doesn't really assume any specific method of connectivity.

What Jenkins expects from your script is that, in the end, it has to execute the agent program like `java -jar agent.jar`, on the right computer, and have its stdin/stdout connect to your script's stdin/stdout. For example, a script that does "`ssh` *`mynode`* `java -jar ~/bin/agent.jar`" would satisfy this.
(The point is that you let Jenkins run this command, as Jenkins uses this stdin/stdout as the communication channel to the agent. Because of this, running this manually from your shell will do you no good).

A copy of `agent.jar` can be downloaded from `http://yourserver:port/jnlpJars/agent.jar` . Many people write scripts in such a way that this 160K jar is downloaded during the running of said script, to ensure that a consistent version of `agent.jar` is always used. Such an approach eliminates the agent.jar updating issue discussed below. Note that the SSH Slaves plugin does this automatically, so agents configured using this plugin always use the correct `agent.jar`.

> **Updating slave.jar**
>
> Technically speaking, in this set up you should update `agent.jar` every time you upgrade Jenkins to a new version. However, in practice `agent.jar` changes infrequently enough that it's also practical not to update until you see a fatal problem in start-up.

Launching agents this way often requires an additional initial set up on agents (especially on Windows, where remote login mechanism is not available out of box), but the benefits of this approach is that when the connection goes bad, you can use Jenkins's web interface to re-establish the connection.

Launch agent via "JNLP" from agent back to master in a browser

Another way of doing this is to start an agent through Java Web Start (JNLP).

It requires the server to be configured to appear in first place. So, before attempting to create the build agent, head into manage *Jenkins->Global Security->TCP port for JNLP agents*.

In this approach, you'll interactively logon to the agent node, open a browser, and open the agent page. You'll be then presented with the JNLP launch icon. Upon clicking it, Java Web Start will kick in, and it launches an agent on the computer where the browser was running.

This mode is convenient when the master cannot initiate a connection to agents, such as when it runs outside a firewall while the rest of the agents are in the firewall. OTOH, if the machine with an agent goes down, the master has no way of re-launching it on its own.

On Windows, you can do this manually once, then from the launched JNLP agent, you can install it as a Windows service so that you don't need to interactively start the agent from then on.

If you need display interaction (e.g. for GUI tests) on Windows and you have a dedicated (virtual) test machine, this is a suitable option. Create a jenkins user account, enable auto-login, and put a shortcut to the JNLP file in the Startup items (after having trusted the agent's certificate). This allows one to run tests as a restricted user as well.

Note: If the master is running behind a reverse proxy or similar, you might need to configure "Tunnel connection through" in the "Advanced" section of the JNLP start method on the agent configuration page to make JNLP work.

Launch agent headlessly from agent back to master on command line

This launch mode uses a mechanism very similar to JNLP as described above, except that it runs without using GUI, making it convenient for an execution as a daemon on Unix. To do this, configure this agent to be a JNLP agent, take `agent.jar` as discussed above, and then from the agent, run a command like this:

```
$ java -jar agent.jar -jnlpUrl http://yourserver:port/computer/agent-name/slave-agent.jnlp
```

Make sure to replace "agent-name" with the name of your agent.

Other Requirements

Also note that the agents are a kind of a cluster, and operating a cluster (especially a large one or heterogeneous one) is always a non-trivial task. For example, you need to make sure that all agents have JDKs, Ant, CVS, and/or any other tools you need for builds. You need to make sure that agents are up and running, etc. Jenkins is not a clustering middleware, and therefore it doesn't make this any easier. Nevertheless, one can use a server provisioning tool and a configuration management software to facilitate both aspects.

Node labels for agents

Labels are tags one can give an agent which allows it to differentiate itself from other nodes in Jenkins.

A few reasons why node labels are important:

- Nodes might have certain tools associated with it. Labels could include different tools a given node supports.
- Nodes may be in a multi-operating system build environment (e.g. Windows, Mac, and Linux agents within one Jenkins build system). There can be a label for the operating system of the node.
- Nodes may be in geographically different locations which can be the case for multi-datacenter deployments. Jenkins can have agents in different datacenters when inter-datacenter communication is strictly regulated with edge firewalls. In this case, you might have a label for the datacenter or cloudstack in which the agent resides.

Defining labels

Labels are defined in the settings of static agents and for agent clouds. They must be space separated words which define that agent. Sticking to standard ASCII characters is recommended. Here's a few label suggestions one can use for agent agents:

- For toolchains: `jdk`, `node_js`, `ruby`, etc
- For operating systems: `linux`, `windows`, `osx`; or you can be more detailed like `ubuntu16.04`
- For geographic locations: `us-east`, `japan`, `eu-central` etc
- For platforms: `docker`, `openstack`, etc.

Using labels

Jobs and pipelines can be pinned to specific agents or groups of agents if multiple agents have similar sets of labels. In jobs, visit advanced settings and choose restrict where the job can run. In pipelines, you would restrict it with the `node` block. You can restrict jobs by specifying a single label or use a label expression. Here's two examples:

- Single label: `us-east`

- Label expression: `openstack && us-east && linux`

The above label expression means that a given agent must have all of those labels.

Example: Configuration on Unix

This section describes Kohsuke Kawaguchi's set up of Jenkins agents that he used to use inside Sun for his day job. His master Jenkins node ran on a SPARC Solaris box, and he had many SPARC Solaris agents, Opteron Linux agents, and a few Windows agents.

- Each computer has an user called `jenkins` and a group called `jenkins`. All computers use the same UID and GID. (If you have access to NIS, this can be done more easily.) This is not a Jenkins requirement, but it makes the agent management easier.
- On each computer, `/var/jenkins` directory is set as the home directory of user `jenkins`. Again, this is not a hard requirement, but having the same directory layout makes things easier to maintain.
- All machines run `sshd`. Windows agents run `cygwin sshd`.
- All machines have `/usr/sbin/ntpdate` installed, and synchronize clock regularly with the same NTP server.
- Master's `/var/jenkins` have all the build tools beneath it --- a few versions of Ant, Maven, and JDKs. JDKs are native programs, so I have JDK copies for all the architectures I need. The directory structure looks like this:

```
/var/jenkins
  +- .ssh
  +- bin
  |   +- agent  (more about this below)
  +- workspace (jenkins creates this file and store all data files inside)
  +- tools
      +- ant-1.5
      +- ant-1.6
      +- maven-1.0.2
      +- maven-2.0
      +- java-1.4 -> native/java-1.4 (symlink)
      +- java-1.5 -> native/java-1.5 (symlink)
      +- java-1.8 -> native/java-1.8 (symlink)
      +- native -> solaris-sparcv9 (symlink; different on each computer)
      +- solaris-sparcv9
      |   +- java-1.4
      |   +- java-1.5
      |   +- java-1.8
      +- linux-amd64
          +- java-1.4
          +- java-1.5
          +- java-1.8
```

- Master's `/var/jenkins/.ssh` has private/public key and `authorized_keys` so that a master can execute programs on agents through `ssh`, by using public key authentication.
- On master, I have a little shell script that uses rsync to synchronize master's `/var/jenkins` to agents (except `/var/jenkins/workspace`). I also use the script to replicate tools on all agents.
- `/var/jenkins/bin/launch-agent` is a shell script that Jenkins uses to execute jobs remotely. This shell script sets up `PATH` and a few other things before launching `agent.jar`. Below is a very simple example script.

```
#!/bin/bash

JAVA_HOME=/opt/SUN/jdk1.8.0_152
PATH=$PATH:$JAVA_HOME/bin
export PATH
java -jar /var/jenkins/bin/agent.jar
```

- Finally all computers have other standard build tools like `svn` and `cvs` installed and available in PATH.

Note that in the more recent Jenkins packages, the default JENKINS_HOME (aka home directory for the 'jenkins' user on Linux machines, e.g. Red Hat, CentOS, Ubuntu) is set to /var/lib/jenkins.

Scheduling strategy

Some agents are faster, while others are slow. Some agents are closer (network wise) to a master, others are far away. So doing a good build distribution is a challenge. Currently, Jenkins employs the following strategy:

1. If a project is configured to stick to one computer, that's always honored.
2. Jenkins tries to build a project on the same computer that it was previously built.

If you have interesting ideas (or better yet, implementations), please let me know.

Node monitoring

Jenkins has a notion of a “node monitor” which can check the status of an agent for various conditions, displaying the results and optionally marking the agent offline accordingly. Jenkins bundles several, checking disk space in the workspace; disk space in the temporary partition; swap space; clock skew (compared to the master); and response time.

Plugins can add other monitors.

Offline status and retention strategy

Administrators can manually mark agents offline (with an optional published reason) or reconnect them.

Groovy scripts such as Monitor and Restart Offline Slaves can perform batch operations like this. There is also a CLI command to reconnect.

Then there is a background task which automatically reconnects agents that are thought to be back up. The behavior is configurable per agent (or per cloud, if using cloudy provisioning for agents) via a “retention strategy”, of which Jenkins bundles several (plugins can contribute others): always keep online if possible; drop offline when not in use; use a schedule; behave according to cloud’s notion of load.

Transition from master-only to master/agent

Typically, you start with a master-only installation and then much later you add agents as your projects grow. When you enable the master/agent mode, Jenkins automatically configures all your existing projects to stick to the master node. This is a precaution to avoid disturbing existing projects, since most likely you won't be able to configure agents correctly without trial and error. After you configure agents successfully, you need to individually configure projects to let them roam freely. This is tedious, but it allows you to work on one project at a time.

Projects that are newly created on master/agent-enabled Jenkins will be by default configured to roam freely.

Access an Internal CI Build Farm (Master + Agents) from the Public Internet

One might consider make the Jenkins master accessible on the public network (so that people can see it), while leaving the build agents within the firewall (typical reasons: cost and security) There are several ways to make it work:

- Equip the master node with a network interface that's exposed to the public Internet (simple to do, but not recommended in general)
- Allow port-forwarding from the master to your agents within the firewall. The port-forwarding should be restricted so that only the master with its known IP can connect to agents. With this set up in the firewall, as far as Jenkins is concerned it's as if the firewall doesn't exist. If multiple hops are involved, you may wish to investigate how to do ssh "jump host" transparently using the ProxyCommand construct. In fact, with a properly configured "jump host" setup, even the master doesn't need to expose itself to the public Internet at all - as long as the organization's firewall allows port 22 traffic.

- Use JNLP agents and have agents connect to the master, not the other way around. In this case it's the agents that initiates the connection, so it works correctly with the NAT firewall.

Note that in both cases, once the master is compromised, all your agents can be easily compromised (IOW, malicious master can execute arbitrary program on agents), so both set-up leaves much to be desired in terms of isolating security breach. Build Publisher Plugin provides another way of doing this, in more secure fashion.

Running Multiple Agents on the Same Machine

Using a well established virtualization infrastructure such as Kernel-based Virtual Machine (KVM), it is quite easy to run multiple agent instances on a single physical node. Such instances can be running various Linux, *BSD UNIX, Solaris, Windows. For Windows, one can have them installed as separate Windows services so they can start up on system startup. While the correct use of executors largely obviates the need for multiple agent instances on the same machine, there are some unique use cases to consider:

- You want more configurability between the configured nodes. Say you have one node set to be used as much as possible, and the other node to be used only when needed.
- You may have multiple Jenkins master installations building different things, and so this configuration would allow you to have agents for more than one master on the same box. That's right, with Jenkins you really can serve two masters.
- You may wish to leverage the easiness of starting/stopping/replacing virtual machines, perhaps in conjunction with Jenkins plugins such as the Lib virt Slaves Plugin.
- You wish to maximize your hardware investment and utilization, at the same time minimizing operating cost (e.g. utility expenses for running idling agents).

Follow these steps to get multiple agents working on the same Windows box:

- Add the first agent node in Jenkins and give it its own working dir (e.g. jenkins-agent-a).
- Go to the agent page from the agent box and launch by JNLP, then use the menu to install it as a service instead.
- Once the service is running, you'll get jenkins-slave.exe and jenkins-slave.xml in your agent's work dir.

- Bring up windows services and stop the Jenkins Slave service.
- Open a shell prompt, cd into the agent work dir.
- First run "jenkins-slave.exe uninstall" to uninstall the one that the jnlp-launched app installed. This should remove it from the service list.
- Now edit jenkins-slave.xml. Modify the id and name values so that your multiple agents are distinct. I called mine jenkins-agent-a and Jenkins Agent A.
- Run jenkins-slave.exe install and then check the Windows service list to ensure it is there. Start it up, and watch Jenkins to see if the agent instance becomes active.
- Now repeat this process for a second agent, beginning with configuring the new node in the master config.

When you go to create the second node, it is nice to be able to copy an existing node, and copy the first node you setup. Then you just tweak the Remote FS Root and a couple other settings to make it distinct. When you are done you should have two (or more) Jenkins slave services in the list of Windows services.

Troubleshooting tips

Some interesting pages on issues (and resolutions) occurring when using Windows agents:

- Windows agents fail to start via DCOM
- Windows slaves fail to start via ssh
- Windows slaves fail to start via JNLP

Some more general troubleshooting tips:

1. Every time Jenkins launches a program locally/remotely, it prints out the command line to the log file. So when a remote execution fails, login to the computer that runs the master by using the same user account, and try to run the command from your shell. You tend to solve problems quickly in this way.
2. Each agent has a log page showing the communication between the master and the agent agent. This log often shows error reports.
3. If you use binary-unsafe remoting mechanism like telnet to launch an agent, add the `-text` option to `agent.jar` so that Jenkins avoids sending binary data over the network.
4. When the same command runs outside Jenkins just fine, make sure you are testing it with the same user account as Jenkins runs under. In particular, if you run Jenkins master on Windows, consult How to get command prompt as the SYSTEM user.
5. Feel free to send your trouble to one of our mailing lists

Windows agent service upgrades

If a newer version of the Jenkins windows service wrapper (jenkins-slave.exe) is available it will be replaced and used on the next start of the service. On very rare occasions the service wrapper may change its behaviour that would require a change in configuration of the service. This can not be done automatically as the service configuration may not be the default and as such could break an installation.

A quick fix of this is to uninstall the jenkins service then verify the service xml is up-to-date (and contains any site configuration such as the user credentials) and then re-install the service.

Other manual task that may fix the issue:

- Jenkins > 1.565.1 - a message similar to `Restart failure. 'C:\jenkins\jenkins-slave.exe restart' completed with 0 but I'm still alive` in the agent error logs. In the windows service manager edit the service configuration to restart the service on failure and add `-noReconnect` to the agent arguments in the service xml configuration.

Other readings

- Jenkins Build Farm Experience Volume I, Volume 2, Volume 3 and Volume 4
- HudsonWindowsSlavesSetup
  http://community.jboss.org/wiki/HudsonWindowsSlavesSetup