**Distributed Builds**

Jenkins supports the "master/slave" mode, where the workload of building projects are delegated to multiple "slave" nodes, allowing a single Jenkins installation to host a large number of projects, or to provide different environments needed for builds/tests. This document describes this mode and how to use it.

**Contents**

* [Contents](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Contents)
* [How does this work?](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Howdoesthiswork%3F)
* [Different ways of starting slave agents](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Differentwaysofstartingslaveagents)
  + [Have master launch slave agent via ssh](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Havemasterlaunchslaveagentviassh)
  + [Have master launch slave agent on Windows](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-HavemasterlaunchslaveagentonWindows)
  + [Write your own script to launch Jenkins slaves](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-WriteyourownscripttolaunchJenkinsslaves)
  + [Launch slave agent via Java Web Start](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-LaunchslaveagentviaJavaWebStart)
  + [Launch slave agent headlessly](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Launchslaveagentheadlessly)
  + [Other Requirements](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-OtherRequirements)
* [Example: Configuration on Unix](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Example%3AConfigurationonUnix)
* [Scheduling strategy](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Schedulingstrategy)
* [Node monitoring](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Nodemonitoring)
* [Offline status and retention strategy](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Offlinestatusandretentionstrategy)
* [Transition from master-only to master/slave](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Transitionfrommasteronlytomaster%2Fslave)
* [Access an Internal CI Build Farm (Master + Slaves) from the Public Internet](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-AccessanInternalCIBuildFarm%28MasterSlaves%29fromthePublicInternet)
* [Running Multiple Slaves on the Same Machine](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-RunningMultipleSlavesontheSameMachine)
* [Troubleshooting tips](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Troubleshootingtips)
  + [Windows slave service upgrades](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Windowsslaveserviceupgrades)
* [Other readings](https://wiki.jenkins-ci.org/display/JENKINS/Distributed+builds#Distributedbuilds-Otherreadings)

**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 a slave 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 slaves 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 slaves to pick up the load. As mentioned above you might also need several different environments to test your builds. In this case using a slave to represent each of your required environments is almost a must.

A slave 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 slaves. 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 slaves.  In other words, the master becomes a sort of "portal" to the entire build farm.

Since each slave runs a separate program called a "slave agent" there is no need to install the full Jenkins (package or compiled binaries) on a slave. There are various ways to start slave agents, but in the end the slave agent and Jenkins master needs 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 slave machines](https://wiki.jenkins-ci.org/display/JENKINS/Step+by+step+guide+to+set+up+master+and+slave+machines) to quickly start using distributed builds.

**Different ways of starting slave agents**

Pick the right method depending on your environment and OS that master/slaves run.

**Have master launch slave agent via ssh**

Jenkins has a built-in SSH client implementation that it can use to talk to remote sshd and start a slave agent. This is the most convenient and preferred method for Unix slaves, 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 slave host name, user name, and ssh credential). Note that the slave will need the master's public ssh key copied to ~/.ssh/authorized\_keys. ([This is a decent howto](http://www.linuxproblem.org/art_9.html) if you need ssh help). Jenkins will do the rest of the work by itself, including copying the binary needed for a slave agent, and starting/stopping slaves. 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](https://wiki.jenkins-ci.org/display/JENKINS/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 slaves through cygwin sshd, see [SSH slaves and Cygwin](https://wiki.jenkins-ci.org/display/JENKINS/SSH+slaves+and+Cygwin) for more details.

**Have master launch slave agent on Windows**

For Windows slaves, Jenkins can use the remote management facility built into Windows 2000 or later ([WMI](http://en.wikipedia.org/wiki/Windows_Management_Instrumentation)+[DCOM](http://en.wikipedia.org/wiki/Distributed_Component_Object_Model), 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 slaves**

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

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](http://www.cygwin.com/) or tools like [psexec](http://technet.microsoft.com/en-ca/sysinternals/bb897553.aspx)), 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 slave agent program like java -jar slave.jar, on the right computer, and have its stdin/stdout connect to your script's stdin/stdout. For example, a script that does "ssh *myslave* java -jar ~/bin/slave.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 slave agent. Because of this, running this manually from your shell [will do you no good](https://wiki.jenkins-ci.org/display/JENKINS/Launching+slave.jar+from+from+console)).

A copy of slave.jar can be downloaded from <http://yourserver:port/jnlpJars/slave.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 slave.jar is always used. Such an approach eliminates the slave.jar updating issue discussed below. Note that the [SSH Slaves](https://wiki.jenkins-ci.org/display/JENKINS/SSH+Slaves+plugin) plugin does this automatically, so slaves configured using this plugin always use the correct slave.jar.

|  |  |
| --- | --- |
| https://wiki.jenkins-ci.org/images/icons/emoticons/information.gif | **Updating slave.jar** Technically speaking, in this set up you should update slave.jar every time you upgrade Jenkins to a new version. However, in practice slave.jar changes infrequently enough that it's also practical not to update until you see a fatal problem in start-up. |

Launching slaves this way often requires an additional initial set up on slaves (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 slave agent via Java Web Start**

Another way of doing this is to start a slave agent through Java Web Start (JNLP). In this approach, you'll interactively logon to the slave node, open a browser, and open the slave page. You'll be then presented with the JNLP launch icon. Upon clicking it, Java Web Start will kick in, and it launches a slave agent on the computer where the browser was running.

This mode is convenient when the master cannot initiate a connection to slaves, such as when it runs outside a firewall while the rest of the slaves are in the firewall. OTOH, if the machine with a slave 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 slave agent, you can [install it as a Windows service](https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins+as+a+Windows+service) so that you don't need to interactively start the slave 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](http://support.microsoft.com/default.aspx?scid=kb;en-us;324737), and put a shortcut to the JNLP file in the Startup items (after having trusted the slave agent's certificate). This allows one to run tests as a restricted user as well.

**Launch slave agent headlessly**

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

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

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

**Other Requirements**

Also note that the slaves 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 slaves have JDKs, Ant, CVS, and/or any other tools you need for builds. You need to make sure that slaves 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](http://en.wikipedia.org/wiki/Provisioning#Server_provisioning) and [a configuration management software](http://en.wikipedia.org/wiki/Comparison_of_open_source_configuration_management_software) to facilitate both aspects.

**Example: Configuration on Unix**

This section describes Kohsuke Kawaguchi's set up of Jenkins slaves 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 slaves, Opteron Linux slaves, and a few Windows slaves.

* 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 slave 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 slaves 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
* | +- slave (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)
* +- native -> solaris-sparcv9 (symlink; different on each computer)
* +- solaris-sparcv9
* | +- java-1.4
* | +- java-1.5
* +- linux-amd64
* +- java-1.4
* +- java-1.5
* Master's /var/jenkins/.ssh has private/public key and authorized\_keys so that a master can execute programs on slaves through ssh, by using [public key authentication](http://www.google.com/search?q=ssh+keygen).
* On master, I have a little shell script that uses rsync to synchronize master's /var/jenkins to slaves (except /var/jenkins/workspace). I also use the script to replicate tools on all slaves.
* /var/jenkins/bin/launch-slave is a shell script that Jenkins uses to execute jobs remotely. This shell script sets up PATH and a few other things before launching slave.jar. Below is a very simple example script.
* #!/bin/bash
* JAVA\_HOME=/opt/SUN/jdk1.6.0\_04
* PATH=$PATH:$JAVA\_HOME/bin
* export PATH
* java -jar /var/jenkins/bin/slave.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 slaves are faster, while others are slow. Some slaves 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.
3. Jenkins tries to move long builds to slaves, because the amount of network interaction between a master and a slave tends to be logarithmic to the duration of a build (IOW, even if project A takes twice as long to build as project B, it won't require double network transfer.) So this strategy reduces the network overhead.

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

**Node monitoring**

Jenkins has a notion of a [“node monitor”](http://javadoc.jenkins-ci.org/hudson/node_monitors/NodeMonitor.html) which can check the status of a slave for various conditions, displaying the results and optionally marking the slave 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 slaves offline (with an optional published reason) or reconnect them.

Groovy scripts such as [Monitor and Restart Offline Slaves](https://wiki.jenkins-ci.org/display/JENKINS/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 slaves that are thought to be back up. The behavior is configurable per slave (or per cloud, if using cloudy provisioning for slaves) via a [“retention strategy”](http://javadoc.jenkins-ci.org/hudson/slaves/RetentionStrategy.html), 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/slave**

Typically, you start with a master-only installation and then much later you add slaves as your projects grow. When you enable the master/slave 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 slaves correctly without trial and error. After you configure slaves 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/slave-enabled Jenkins will be by default configured to roam freely.

**Access an Internal CI Build Farm (Master + Slaves) 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 slaves 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 slaves within the firewall. The port-forwarding should be restricted so that only the master with its known IP can connect to slaves. 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 slaves and have slaves connect to the master, not the other way around. In this case it's the slaves that initiates the connection, so it works correctly with the NAT firewall.

Note that in both cases, once the master is compromised, all your slaves can be easily compromised (IOW, malicious master can execute arbitrary program on slaves), so both set-up leaves much to be desired in terms of isolating security breach. [Build Publisher Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Build+Publisher+Plugin) provides another way of doing this, in more secure fashion.

**Running Multiple Slaves on the Same Machine**

Using a well established virtualization infrastructure such as [Kernel-based Virtual Machine (KVM)](http://en.wikipedia.org/wiki/Kernel-based_Virtual_Machine), it is quite easy to run multiple slave 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 slave 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 slaves 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 [Libvirt Slaves Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Libvirt+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 slaves).

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

* Add the first slave node in Jenkins and give it its own working dir (e.g. jenkins-slave-a).
* Go to the slave page from the slave 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 slave's work dir.
* Bring up windows services and stop the Jenkins Slave service.
* Open a shell prompt, cd into the slave 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 slaves are distinct. I called mine jenkins-slave-a and Jenkins Slave 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 slave instance becomes active.
* Now repeat this process for a second slave, 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 slaves:

* [Windows slaves fail to start via DCOM](https://wiki.jenkins-ci.org/display/JENKINS/Windows+slaves+fail+to+start+via+DCOM)
* [Windows slaves fail to start via ssh](https://wiki.jenkins-ci.org/display/JENKINS/Windows+slaves+fail+to+start+via+ssh)
* [Windows slaves fail to start via JNLP](https://wiki.jenkins-ci.org/display/JENKINS/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 slave has a log page showing the communication between the master and the slave agent. This log often shows error reports.
3. If you use binary-unsafe remoting mechanism like telnet to launch a slave, add the -text option to slave.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](https://wiki.jenkins-ci.org/display/JENKINS/How+to+get+command+prompt+as+the+SYSTEM+user).
5. Feel free to send your trouble to one of our mailing lists|http://jenkins-ci.org/content/mailing-lists

**Windows slave 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.

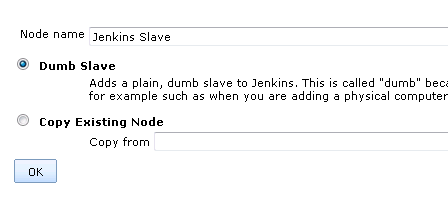
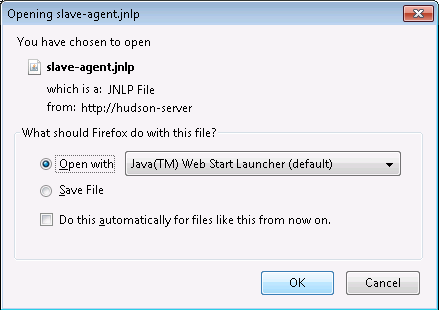
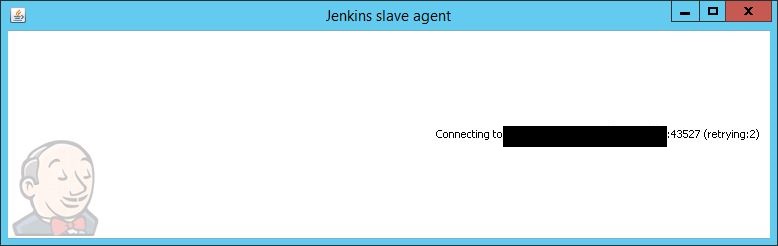
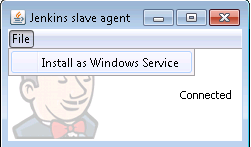
Other manual task that may fix the issue:

* 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 slave error logs. In the windows service manager edit the service configuration to restart the service on failure and add -noReconnect to the slave arguments in the service xml configuration.

|  |  |
| --- | --- |
|  |  |

# [Step by step guide to set up master and slave machines](https://wiki.jenkins-ci.org/display/JENKINS/Step+by+step+guide+to+set+up+master+and+slave+machines)

# There is no need to install Jenkins on the slave machine.

1. On your master machine go to **Manage Jenkins** > **Manage Nodes**.
2. **New Node** --> **Enter Node Name**.
3. **Select Dumb Slave** --> Press **OK**.  
   
4. Fill out the following:
   1. Set a **number of executors** (one or more) as needed.
   2. Set a **Remote FS Root**, a home directory for the master on the slave machine.
      1. For a *Windows slave*, use something like: "C:\Jenkins\"
      2. ***TODO***: add details.
   3. Select the appropriate **Usage** setting:
      1. For an additional worker: *Utilize this slave as much as possible*
      2. For specialized jobs: *Leave this machine for tied jobs only*
   4. **Launch Method**:
      1. An easy way to control a Windows slave is by using *Launch slave agents via Java Web Start*  (Recommended for Windows)
      2. ***TODO***: add steps for other methods.
   5. **Availability** --> *Keep this slave online as much as possible*
      1. ***TODO:*** add details for each option.
   6. Press **OK**.  
      
5. Now you need to connect your slave machine to the master using the following steps.
   1. Open a browser on the **slave machine** and go to the **Jenkins master server** url (<http://yourjenkinsmaster:8080>).
   2. Go to **Manage Jenkins** > **Manage Nodes**, Click on the newly created slave machine. You will need to login as someone that has the "Connect" Slave permission if you have configured global security.
   3. Click on the **Launch** button to launch agent from browser on slave.  
      
   4. Run the program.  
      
      1. If you encounter connection issue, then you could enlarge the popup windows to see the master **port used** and check your network configuration (firewall, port forward, ...)
         1. 
   5. Now you should see the Slave machine connected under **Nodes**.
6. If you want the service to run on start-up of the slave machine do the following (Windows only directions):
   1. In the Slave agent program running on your slave machine,
   2. click **File** --> **Install as Windows Service.**  
        
      Note that this feature requires ".Net Framework 3.5"  
      https://wiki.jenkins-ci.org/download/attachments/41878276/dotNet3_5Features.JPG?version=2&modificationDate=1426780066000
   3. **Start**, type Services and Select the **Services** program.
   4. Find **Jenkins Slave** in the list, Double click to open.
   5. Select **Startup type** --> **Automatic**.
   6. Go to the **Log On** tab, change the **Log on as** to a user of your choice (Special user account Jenkins recommended).
   7. Make sure that auto login is set for the slave machine for the user account, then the VM (or physical computer) should connect and be available when needed.

## Jenkins Enterprise HA

## Introduction

Jenkins Enterprise comes with the capability to run in a high-availability setup, where two or more JVMs form a so-called "HA singleton" cluster, to ensure that the Jenkins master is always running somewhere. This improves the availability of the service against unexpected problems in the Jenkins master JVM, the hardware that it runs on, etc. When a Jenkins master JVM becomes unavailable (for example, when it stops responding, or when it dies), other nodes in the cluster automatically take over the role of the master, thereby restoring service with minimum interruption.

It is also important for users to understand what this feature does not do in its current form. Namely, it is not a symmetric cluster, where participating nodes will share workloads together. At any given point only one of the nodes is performing the master role (hence "HA singleton"). Because of this, when a fail-over takes place, users will see a brief down time, comparable to someone rebooting a Jenkins master in a non-HA setup. Builds that were in progress would be lost, too (but see below).

With that said, care has been taken to simplify the deployment as much as possible without sacrificing the flexibility of the deployment, and we think we have built the right foundation to enable more sophisticated clustering down the road.

## Components of Jenkins Enterprise HA

This section describes 3 building blocks of Jenkins Enterprise by CloudBees that you will combine into a deployment.

### Jenkins Enterprise by CloudBees WAR file

As you saw in the tutorial, Jenkins Enterprise WAR file contains the necessary code to run Jenkins in a highly-available setup. The extra control logic in this WAR file cooperatively elect the "primary" JVM, and depending on the outcome of this election, members of a cluster starts/stops the Jenkins master in the same JVM. From the view point of the code inside Jenkins, this is as if Jenkins is started/stopped programmatically.

Jenkins Enterprise relies on [JGroups](http://jgroups.org/) for the underlying group membership service.

#### Network configuration

By default, Jenkins Enterprise uses TCP to communicate between members, with IP addresses and ports registered in a directory $JENKINS\_HOME/jgroups (which all members must be able to write to). This can be changed by creating $JENKINS\_HOME/jgroups.xml that describes the JGroups protocol stack configuration XML. See the [JBoss Clustering documentation](http://docs.jboss.org/jbossclustering/cluster_guide/5.1/html/jgroups.chapt.html#jgroups-transport) for the format of this file, as well as typical configuration tips and trouble-shooting.

### Jenkins Enterprise HA proxy WAR

To allow you to run OSS Jenkins WAR in the HA configuration, a separate "proxy WAR" is provided. This optional file can be downloaded from [the ha-proxy section](http://nectar-downloads.cloudbees.com/ha-proxy/) of the download site. The proxy WAR is a proxy to real Jenkins WAR file; you drop it in place of jenkins.war, and instead of acting as Jenkins by itself, it expects you to have the real thing in $JENKINS\_HOME/jenkins.war. The proxy WAR only controls the cluster control logic, and when it decides that it needs to act as the primary JVM, it’ll load $JENKINS\_HOME/jenkins.war into memory and runs that program.

This setup allows you to run newer versions of mainline Jenkins releases (for example to benefit from the latest and greatest features), or your custom build of Jenkins.

With the rpm/deb packages of Jenkins or Jenkins Enterprise, the easiest way to use this feature is to modify /etc/init.d/jenkins (for RPM) and /etc/default/jenkins (for DEB) and update JENKINS\_WAR variable to point to the proxy war. For other deployments, move the real jenkins.war into $JENKINS\_HOME, then put ha-proxy.war as jenkins.war.

The proxy WAR uses $JENKINS\_HOME in the same ways as Jenkins Enterprise WAR to pick up the network configuration and other parameters.

### Jenkins Enterprise HA monitor tool

Depending on the way you deploy the high-availability setup, some system level setup/teardown activities need to take place when the primary JVM changes. In the tutorial, we needed to run ifconfig to move the floating IP address. Other examples include DRBD mode change, updates to the front-end reverse proxy, or sending out an e-mail.

Because this needs to be executed when the primary Jenkins JVM becomes unresponsive, such setup/teardown scripts can only be reliably triggered from outside Jenkins. Those scripts also normally require root privileges to run. The Jenkins Enterprise HA monitor tool is a small background application that provides this role. It is an optional component that should be installed only when necessary. It can be downloaded from [the jenkins-ha-monitor section](http://nectar-downloads.cloudbees.com/jenkins-ha-monitor/) of the download site.

The monitor tool is packaged into a single jar file that can be executed as java -jar jenkins-ha-monitor.jar. It is also packaged as the jenkins-ha-monitor RPM/DEB packages for an easier installation. Its 3 defining options are as follows:

* The -home option that specifies the location of $JENKINS\_HOME. The monitor tool picks up network configuration and other important parameters from here, so it needs to know this location.
* The -host-promotion option that specifies the location of the promotion script, which gets executed when the primary Jenkins JVM moves in from another system into this system as a result of election. In native packages, this file is placed at /etc/jenkins-ha-monitor/promotion.sh.
* The -host-demotion option that specifies the demotion script, which is the opposite of the promotion script and gets executed when the primary Jenkins JVM moves from this system into another system. In native packages, this file is placed at /etc/jenkins-ha-monitor/demotion.sh.

Promotion and demotion scripts need to be idempotent, in the sense that the monitor tool may run the promotion script on an already promoted node, and the demotion script on an already demoted node. This can happen, for example, when a power outage hits a stand-by node and when it comes back up. The monitor tool runs the demotion script again on this node, since it cannot be certain about the state of the node before the power outage.

Run the tool with the -help option to see the complete list of available options.

## HA Tutorial

In this tutorial, we’ll describe the simplest HA Jenkins setup, which creates a baseline when we later discuss other modes of deployment. Here, we deploy HA Jenkins in the following configuration:

* An NFS server that hosts $JENKINS\_HOME. We assume this is available already, and we will not discuss how one would set this up. (We’ll call this machine sierra.)
* Two linux systems that form a Jenkins HA cluster, by running one JVM each (we’ll call them 'alpha' and 'bravo'). In this tutorial, those two machines need to be on the same local network.
* One floating IP address (we’ll call this 1.2.3.4). Alpha and bravo take this IP address while each is acting as the primary, thereby ensuring that users can always access Jenkins through the DNS name that’s associated with this IP address.

First, install Jenkins Enterprise packages to alpha and bravo. You need to install both the jenkins package (for the Jenkins Enterprise by CloudBees itself), and the jenkins-ha-monitor package, which is the Jenkins Enterprise by CloudBees HA monitor tool.

### Tip

Linux packages for jenkins-ha-monitor are not included in all versions of the Jenkins Enterprise repository. If yours is missing this package, or you are using a non-Linux system, see [the section called “Jenkins Enterprise HA monitor tool”](http://jenkins-enterprise.cloudbees.com/docs/user-guide-docs/ha-sect-components.html#ha-sect-components-monitor) for information on direct download.

Choose the appropriate debian/redhat/openSUSE package format depending on the type of your distribution. Upon installation, both instances of Jenkins will start running. Stop them by issuing /etc/init.d/jenkins stop, while we work on the HA setup.

Let’s say sierra exports the /jenkins directory that hosts $JENKINS\_HOME. Log on to alpha and mount this directory.

$ mount -t nfs -o rw,hard,intr sierra:/jenkins /var/lib/jenkins

/var/lib/jenkins is chosen to match what the Jenkins Enterprise packages use as $JENKINS\_HOME. If you change them, update /etc/default/jenkins (on Debian) or /etc/sysconfig/jenkins (on RedHat and SUSE) to have $JENKINS\_HOME point to the correct directory.

To make this mount automatically happen, update your /etc/fstab by adding the following entry:

sierra:/jenkins nfs rw,hard,intr 0 2

Repeat this mount setup on bravo, and ensure that both alpha and bravo see the same data. (For example, touch a from alpha, and make sure ls from bravo will see it. Make sure the uid and the gid appear the same on alpha and bravo.)

Boot Jenkins on alpha and bravo by issuing /etc/init.d/jenkins start. Now Jenkins Enterprise boots up in a two-node HA cluster. Access <http://alpha:8080/> and <http://bravo:8080/>. One will serve the familiar Jenkins UI, and the other will tell you that it’s acting as a stand-by node. In the Jenkins UI, go to "Manage Jenkins" then click "High Availability Status" and make sure two nodes are listed as members. You can kill the primary JVM (for example by kill -9 PID) while watching the log file via tail -f /var/log/jenkins/jenkins.log, and you’ll see the stand-by node take over the primary role.

Finally, we set up a monitoring service to ensure that the floating IP address gets assigned to the system that’s hosting the primary. To do this, log on to alpha and install the jenkins-ha-monitor package.

This monitoring program watches Jenkins as root, and when the role transition occurs, it’ll execute the promotion script or the demotion script. In this tutorial, we’ll make these scripts assign/release the floating IP address.

In /etc/jenkins-ha-monitor/promotion.sh, write the following script:

#!/bin/sh

# assign the floating IP address 1.2.3.4 as an alias of eth1

ifconfig eth1:100 1.2.3.4

Similarly, in /etc/jenkins-ha-monitor/demotion.sh, write the following script:

#!/bin/sh

# release the floating IP address

ifconfig eth1:100 down

eth1:100 needs to be your network interface name followed by an unique alias ID (see "Linux IP aliasing" for more details.) Now that the configuration file is updated, restart the JA monitoring service by running /etc/init.d/jenkins-ha-monitor restart. Access the "High Availability Status" in the "Manage Jenkins" section from the web UI to verify that the monitoring service is recognized as a part of the cluster. Run ifconfig to verify that the virtual IP address is assigned to the system that’s hosting the primary JVM.

Congratulations, now you have a highly-available Jenkins!

### Using haproxy as a reverse proxy

Let’s expand on this setup further by introducing an external load balancer / reverse proxy that receives traffic from users, then direct them to the active primary JVM. Compared to IP aliasing, this is more complex, but it allows two nodes that aren’t in the same subnet to form a cluster, and you can set this up without having a root access.

haproxy can be installed on most Linux systems via native packages, such as apt-get install haproxy or yum install haproxy. For Jenkins Enterprise HA, the configuration file (normally /etc/haproxy/haproxy.cfg) should look like the following:

# this section is a stock setting

global

log 127.0.0.1 local0

log 127.0.0.1 local1 notice

maxconn 4096

user haproxy

group haproxy

defaults

log global

# The following log settings are useful for debugging

# Tune these for production use

option logasap

option http-server-close

option redispatch

option abortonclose

option log-health-checks

mode http

option dontlognull

retries 3

maxconn 2000

timeout http-request 10s

timeout queue 1m

timeout connect 10s

timeout client 1m

timeout server 1m

timeout http-keep-alive 10s

timeout check 500

default-server inter 5s downinter 500 rise 1 fall 1

# this block specifies a Jenkins Enterprise HA cluster

listen application 0.0.0.0:80

balance roundrobin

reqadd X-Forwarded-Proto:\ http

option forwardfor except 127.0.0.0/8

option httplog

option httpchk HEAD /ha/health-check

server alpha alpha:8080 check

server bravo bravo:8080 check

listen jnlp 0.0.0.0:10001

mode tcp

option tcplog

timeout server 15m

timeout client 15m

# Jenkins by default runs a ping every 10 minutes and waits 4

# minutes for a timeout before killing the connection, thus we

# need to keep these TCP raw sockets open for at least that

# long.

option httpchk HEAD /ha/health-check

server alpha alpha:10001 check port 8080

server bravo bravo:10001 check port 8080

listen ssh 0.0.0.0:2022

mode tcp

option tcplog

option httpchk HEAD /ha/health-check

server alpha alpha:2022 check port 8080

server bravo bravo:2022 check port 8080

# monitor port

listen status 0.0.0.0:8081

stats enable

stats uri /

The global section is stock settings. defaults has been modified to include additional logging for debugging. It is advisable to review and tune the log settings before production use. defaults has also been configured with typical timeout settings. Again these should be tuned in production.

The part that you will need to add and configure specifically for your environment is the listen blocks for application, jnlp and ssh. These tell haproxy to forward traffic to two servers alpha and bravo, and periodically check their health by sending a GET request to /ha/health-check. Stand-by nodes do not respond positively to this health check, while real Jenkins will, and that’s how haproxy will know which of the two machines to send the traffic to.

Note the specific timeout overrides for the jnlp service based on Jenkins internal behaviour.

For the jnlp and ssh services haproxy is configured to forward tcp requests. For these services we have specifically told haproxy to use the same health check on the application port (8080). This ensures that all services fail over together when the health check fails.

The second listen status section allows you to monitor haproxy by accessing port 8081. This is handy when you want to understand how haproxy is behaving.

**Fail-over behavior in more details**

In this section, we’ll look at exactly what happens during a fail over.

In this release of Jenkins Enterprise, a fail-over is effectively (1) shutting down the current Jenkins master, followed by (2) starting it up in another location. Sometimes the step 1 doesn’t happen, for example when the current master becomes unresponsive. Because these masters work with the same $JENKINS\_HOME, this fail over has the following characteristics:

* Jenkins global settings, configuration of jobs/users, fingerprints, record of completed builds (including archived artifacts, test reports, etc.), will all survive a fail-over.
* Builds that were in progress will normally not survive a fail-over, although their records will survive. Normally no attempt will be made to re-execute those builds, though [Restart Aborted Builds](http://jenkins-enterprise.cloudbees.com/docs/user-guide-docs/aborted-builds.html#aborted-builds-sect-intro) can do so. [Long-running builds](http://jenkins-enterprise.cloudbees.com/docs/user-guide-docs/long-running-build.html#long-running-build-sect-intro) may also be used in case aborting builds would be problematic.
* User sessions are lost. If your Jenkins installation requires users to log in, they’ll be asked to log in again.
* During the startup phase of the fail-over, Jenkins will not be able to serve inbound requests or builds. Therefore, a fail-over typically takes a few minutes, not a few seconds.

|  |  |
| --- | --- |
|  | **cloudbees-enterprise-plugins** |
|  |  |

**for Jenkisn Enterprise**

# Jenkins Enterprise by CloudBees plugin

This plugin converts an OSS installation to a Jenkins Enterprise by CloudBees (JEBC) installation. JEBC has 20+ plugins that address issues in enterprise installations. Plugins include folders, validate merges, templates, roles-based access control, backup plugins and others ([complete list](http://www.cloudbees.com/jenkins-enterprise-by-cloudbees-available-plugins.cb)). Additionally, you can use the HA component to make Jenkins highly available.  
**Note:** As part of the installation process, you will require a valid license to use the CloudBees Enterprise plugins.

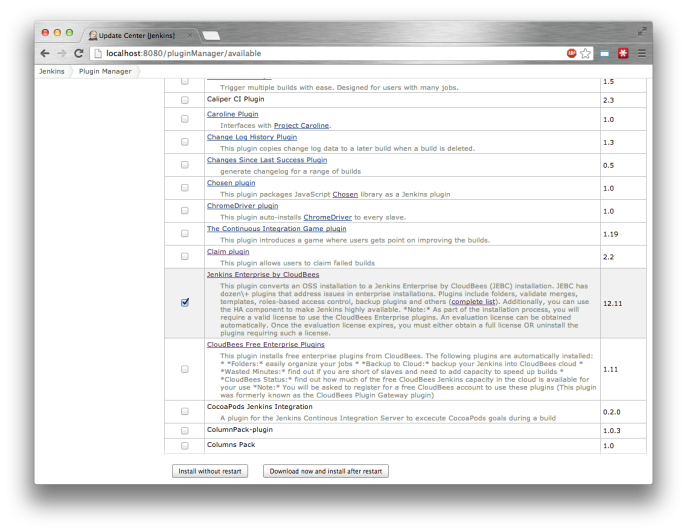
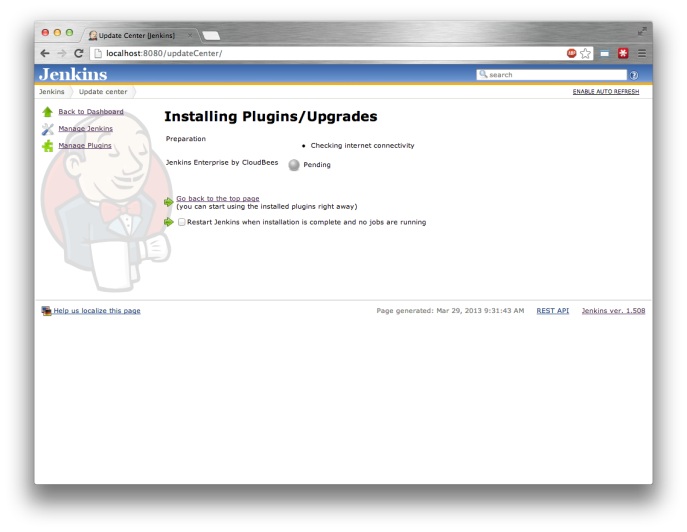
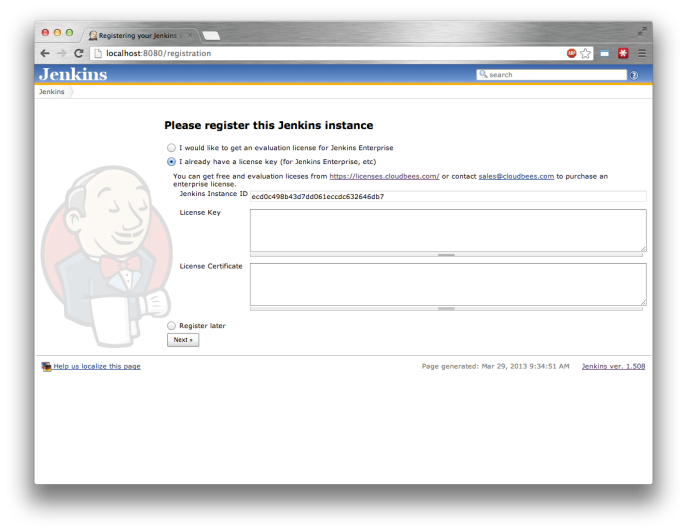
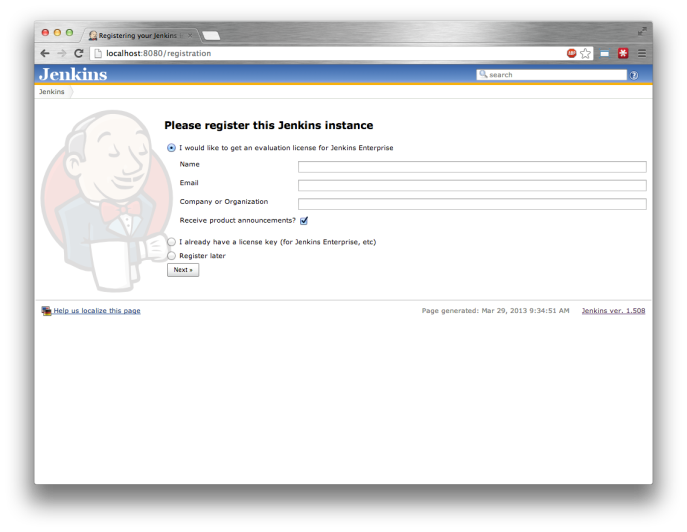
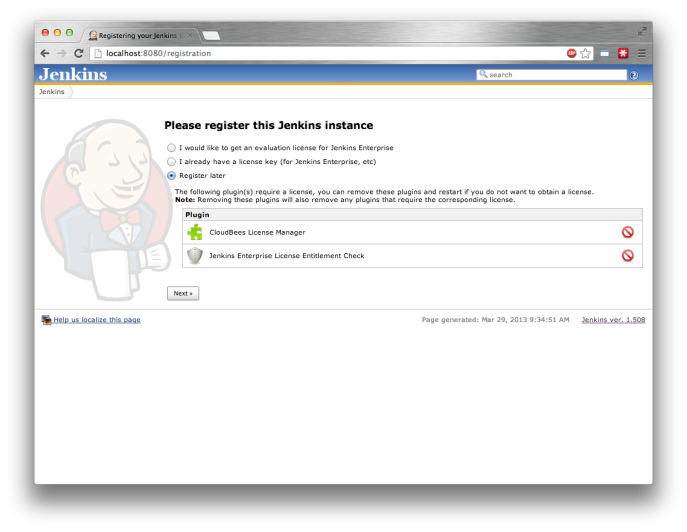
## Installation/Uninstallation

### Prerequisites

* You must be running a sufficiently new version of Jenkins: either the current LTS release, or the newest available weekly build.
* Your Jenkins installation must be able to download and install plugins from the OSS update centre.  
  In other words, if you cannot use the plugin update center to install this plugin then it will not work.

### Installation

This plugin can be installed without restart, but will require a restart to complete installation.

1. Install the Jenkins Enterprise by CloudBees plugin.  
   
2. Install the plugin without restart.  
   
3. Once the plugin is installed, go to Manage Jenkins » Install Jenkins Enterprise by CloudBees and select your preferred installation option. It will add the CloudBees proprietary update center to the list of update centers used by your Jenkins instance as well as the certificate used to validate that update center's certificate.
4. When all the plugins have been downloaded, a Safe Restart will be triggered, or if the Servlet Container that your Jenkins instance is running in does not support safe restarts, a notice bar will be displayed to tell you to restart your Jenkins instance.
5. When your Jenkins instance restarts, you will be asked to either Request an evaluation license or provide an Enterprise license. A valid license is a necessary condition for the Jenkins Enterprise by CloudBees plugins.  
     
   If you do not want to register your system with CloudBees there is an option for removing the CloudBees plugins that require a license for use.  
     
   You click on the red delete icons to remove the license enforcement check or the license manager plugin. A restart will be required if you remove any of the license enforcement checks or the license manager plugin.
6. Once you have completed the registration process your Jenkins instance will complete startup as normal.

### Uninstallation

To uninstall, simply disable or delete the Jenkins Enterprise by CloudBees plugin and the Jenkins Enterprise License Entitlement Check plugins. Once these plugins are disabled, all the Jenkins Enterprise by CloudBees plugins will be disabled automatically, though it is better to remove them from your instance to improve startup performance.

I will use the same .war file for my Production System. Will it have it’s own unique instance?

A. A Jenkins instance ID is generated first time the Jenkins war is started, and it is stored in your JENKINS\_HOME. So you can use the same war on another host and will get a distinct ID, that will require a different dedicated license key.

Can I install Jenkins Enterprise on top of my existing Jenkins community installation?

A. Yes. If the core version of Jenkins Enterprise (e.g. 1.480.3) is at least as new the core version of your existing installation (e.g. 1.480.3, 1.466.2) then you can just replace your jenkins.war with the Jenkins Enterprise download and start it up. If you are running a newer community version of Jenkins (e.g. 1.490) you will be better off using the [Jenkins Enterprise metaplugin](https://wiki.jenkins-ci.org/display/JENKINS/Jenkins+Enterprise+by+CloudBees) to add the Jenkins Enterprise value-add plugins to your installation while still using the same core version.

an I convert existing Jenkins Enterprise installation to an OSS community Jenkins installation while keeping all the plugins?

A. Yes. Download jenkins.war from <http://jenkins-ci.org/> (make sure to pick up the version newer than your current Jenkins Enterprise version), and overwrite jenkins.war with it. Next time you’ll start Jenkins, it’ll come up with OSS Jenkins + all the plugins you had before.

What do I need to do for uninstallation?

A. From a license perspective - nothing. You will need to delete the CloudBees plugins from your JENKINS\_HOME (answered [here](https://developer.cloudbees.com/bin/view/Jenkins+Enterprise/Jenkins+Enterprise+Installation%2C+Uninstallation+and+Upgrade+FAQ))

### ****How do I uninstall the trial version of Jenkins Enterprise?****

**A.** The instructions for uninstallation vary depending on how you installed Jenkins Enterprise, and what functionality you wish to have remaining after uninstallation.

* If you used the [“meta plugin”](https://wiki.jenkins-ci.org/display/JENKINS/Jenkins+Enterprise+by+CloudBees) to upgrade a Jenkins OSS installation to Jenkins Enterprise then you do not need to change your jenkins.war.
* On the other hand, if you downloaded a Jenkins Enterprise WAR, you will need to replace that jenkins.war with the nearest equivalent Jenkins OSS WAR file.
* Finally, if you used one of the Jenkins Enterprise native package installers (e.g. Windows, RPM, DEB, OS-X, etc), you will need to switch to the corresponding Jenkins OSS package installer and potentially change the update sources from which the native package installer pulls updates (primarily for the RPM and DEB installers)

The reason for the above, is that the Jenkins Enterprise version of jenkins.war includes all the bundled plugins, and so if you remove any of the bundled plugins, they will be re-extracted every time Jenkins is installed, so you will find yourself unable to uninstall the licensed components.

When you start Jenkins Enterprise with an expired license you will see the following screen:

# Jenkins Enterprise Plugins

#### For demanding environments

In addition to professional support, **CloudBees Jenkins Enterprise** provides five categories of plugins to address core challenges typically found in enterprise Jenkins implementations:

#### Continuous Delivery

Continuous Delivery plugins enable you to more easily manage complex software delivery pipelines

#### Security

These plugins help administrators secure their projects and/or installations

#### High Availability

Eliminate downtime due to Jenkins master failures

#### Large Implementations

This group of features helps administrators manage large numbers of Jenkins jobs, projects and teams

#### Optimized Utilization

With these plugins, Jenkins administrators can make better use of existing resources, greatly improving build times and reducing costs

## HA Tutorial

In this tutorial, we’ll describe the simplest HA Jenkins setup, which creates a baseline when we later discuss other modes of deployment. Here, we deploy HA Jenkins in the following configuration:

* An NFS server that hosts $JENKINS\_HOME. We assume this is available already, and we will not discuss how one would set this up. (We’ll call this machine sierra.)
* Two linux systems that form a Jenkins HA cluster, by running one JVM each (we’ll call them 'alpha' and 'bravo'). In this tutorial, those two machines need to be on the same local network.
* One floating IP address (we’ll call this 1.2.3.4). Alpha and bravo take this IP address while each is acting as the primary, thereby ensuring that users can always access Jenkins through the DNS name that’s associated with this IP address.

First, install Jenkins Enterprise packages to alpha and bravo. You need to install both the jenkins package (for the Jenkins Enterprise by CloudBees itself), and the jenkins-ha-monitor package, which is the Jenkins Enterprise by CloudBees HA monitor tool.

### Tip

Linux packages for jenkins-ha-monitor are not included in all versions of the Jenkins Enterprise repository. If yours is missing this package, or you are using a non-Linux system, see [the section called “Jenkins Enterprise HA monitor tool”](http://jenkins-enterprise.cloudbees.com/docs/user-guide-docs/ha-sect-components.html#ha-sect-components-monitor) for information on direct download.

Choose the appropriate debian/redhat/openSUSE package format depending on the type of your distribution. Upon installation, both instances of Jenkins will start running. Stop them by issuing /etc/init.d/jenkins stop, while we work on the HA setup.

Let’s say sierra exports the /jenkins directory that hosts $JENKINS\_HOME. Log on to alpha and mount this directory.

$ mount -t nfs -o rw,hard,intr sierra:/jenkins /var/lib/jenkins

/var/lib/jenkins is chosen to match what the Jenkins Enterprise packages use as $JENKINS\_HOME. If you change them, update /etc/default/jenkins (on Debian) or /etc/sysconfig/jenkins (on RedHat and SUSE) to have $JENKINS\_HOME point to the correct directory.

To make this mount automatically happen, update your /etc/fstab by adding the following entry:

sierra:/jenkins nfs rw,hard,intr 0 2

Repeat this mount setup on bravo, and ensure that both alpha and bravo see the same data. (For example, touch a from alpha, and make sure ls from bravo will see it. Make sure the uid and the gid appear the same on alpha and bravo.)

Boot Jenkins on alpha and bravo by issuing /etc/init.d/jenkins start. Now Jenkins Enterprise boots up in a two-node HA cluster. Access <http://alpha:8080/> and <http://bravo:8080/>. One will serve the familiar Jenkins UI, and the other will tell you that it’s acting as a stand-by node. In the Jenkins UI, go to "Manage Jenkins" then click "High Availability Status" and make sure two nodes are listed as members. You can kill the primary JVM (for example by kill -9 PID) while watching the log file via tail -f /var/log/jenkins/jenkins.log, and you’ll see the stand-by node take over the primary role.

Finally, we set up a monitoring service to ensure that the floating IP address gets assigned to the system that’s hosting the primary. To do this, log on to alpha and install the jenkins-ha-monitor package.

This monitoring program watches Jenkins as root, and when the role transition occurs, it’ll execute the promotion script or the demotion script. In this tutorial, we’ll make these scripts assign/release the floating IP address.

In /etc/jenkins-ha-monitor/promotion.sh, write the following script:

#!/bin/sh

# assign the floating IP address 1.2.3.4 as an alias of eth1

ifconfig eth1:100 1.2.3.4

Similarly, in /etc/jenkins-ha-monitor/demotion.sh, write the following script:

#!/bin/sh

# release the floating IP address

ifconfig eth1:100 down

eth1:100 needs to be your network interface name followed by an unique alias ID (see "Linux IP aliasing" for more details.) Now that the configuration file is updated, restart the JA monitoring service by running /etc/init.d/jenkins-ha-monitor restart. Access the "High Availability Status" in the "Manage Jenkins" section from the web UI to verify that the monitoring service is recognized as a part of the cluster. Run ifconfig to verify that the virtual IP address is assigned to the system that’s hosting the primary JVM.

Congratulations, now you have a highly-available Jenkins!

### Using haproxy as a reverse proxy

Let’s expand on this setup further by introducing an external load balancer / reverse proxy that receives traffic from users, then direct them to the active primary JVM. Compared to IP aliasing, this is more complex, but it allows two nodes that aren’t in the same subnet to form a cluster, and you can set this up without having a root access.

haproxy can be installed on most Linux systems via native packages, such as apt-get install haproxy or yum install haproxy. For Jenkins Enterprise HA, the configuration file (normally /etc/haproxy/haproxy.cfg) should look like the following:

# this section is a stock setting

global

log 127.0.0.1 local0

log 127.0.0.1 local1 notice

maxconn 4096

user haproxy

group haproxy

defaults

log global

# The following log settings are useful for debugging

# Tune these for production use

option logasap

option http-server-close

option redispatch

option abortonclose

option log-health-checks

mode http

option dontlognull

retries 3

maxconn 2000

timeout http-request 10s

timeout queue 1m

timeout connect 10s

timeout client 1m

timeout server 1m

timeout http-keep-alive 10s

timeout check 500

default-server inter 5s downinter 500 rise 1 fall 1

# this block specifies a Jenkins Enterprise HA cluster

listen application 0.0.0.0:80

balance roundrobin

reqadd X-Forwarded-Proto:\ http

option forwardfor except 127.0.0.0/8

option httplog

option httpchk HEAD /ha/health-check

server alpha alpha:8080 check

server bravo bravo:8080 check

listen jnlp 0.0.0.0:10001

mode tcp

option tcplog

timeout server 15m

timeout client 15m

# Jenkins by default runs a ping every 10 minutes and waits 4

# minutes for a timeout before killing the connection, thus we

# need to keep these TCP raw sockets open for at least that

# long.

option httpchk HEAD /ha/health-check

server alpha alpha:10001 check port 8080

server bravo bravo:10001 check port 8080

listen ssh 0.0.0.0:2022

mode tcp

option tcplog

option httpchk HEAD /ha/health-check

server alpha alpha:2022 check port 8080

server bravo bravo:2022 check port 8080

# monitor port

listen status 0.0.0.0:8081

stats enable

stats uri /

The global section is stock settings. defaults has been modified to include additional logging for debugging. It is advisable to review and tune the log settings before production use. defaults has also been configured with typical timeout settings. Again these should be tuned in production.

The part that you will need to add and configure specifically for your environment is the listen blocks for application, jnlp and ssh. These tell haproxy to forward traffic to two servers alpha and bravo, and periodically check their health by sending a GET request to /ha/health-check. Stand-by nodes do not respond positively to this health check, while real Jenkins will, and that’s how haproxy will know which of the two machines to send the traffic to.

Note the specific timeout overrides for the jnlp service based on Jenkins internal behaviour.

For the jnlp and ssh services haproxy is configured to forward tcp requests. For these services we have specifically told haproxy to use the same health check on the application port (8080). This ensures that all services fail over together when the health check fails.

The second listen status section allows you to monitor haproxy by accessing port 8081. This is handy when you want to understand how haproxy is behaving.

All CloudBees Jenkins Enterprise plugins can be used on any Long Term Support (LTS) release, Jenkins open source trunk or the CloudBees Platform. CloudBees Jenkins Enterprise plugins can be used with any version of Jenkins.

# High Availability Plugin

Challenge:

When a Jenkins master fails (due to software or hardware crashes), it results in extended downtime for the entire product team. Administrators detect these failures manually or through homegrown scripts. Once a failure is noticed, administrators scamper to get the master up as quickly as possible; this process is often manual and easily takes more than a few hours. On larger projects, the downtime experienced from a failure can be the equivalent of several days of lost project time.

#### Solution:

The High Availability plugin, offered within [CloudBees Jenkins Enterprise,](https://www.cloudbees.com/products/jenkins-enterprise) eliminates downtime due to master failures. Multiple Jenkins masters act as backups waiting for a primary master failure. Once a failure is detected, a backup master automatically boots up and acts as a failover.The best part? Project work continues.

With this feature, failure detection and recovery is automatic and requires no intervention from administrators or their home-grown scripts. Additionally, this feature can be used to do rolling upgrades of Jenkins.

## Components of Jenkins Enterprise HA

This section describes 3 building blocks of Jenkins Enterprise by CloudBees that you will combine into a deployment.

### Jenkins Enterprise by CloudBees WAR file

As you saw in the tutorial, Jenkins Enterprise WAR file contains the necessary code to run Jenkins in a highly-available setup. The extra control logic in this WAR file cooperatively elect the "primary" JVM, and depending on the outcome of this election, members of a cluster starts/stops the Jenkins master in the same JVM. From the view point of the code inside Jenkins, this is as if Jenkins is started/stopped programmatically.

Jenkins Enterprise relies on [JGroups](http://jgroups.org/) for the underlying group membership service.

#### Network configuration

By default, Jenkins Enterprise uses TCP to communicate between members, with IP addresses and ports registered in a directory $JENKINS\_HOME/jgroups (which all members must be able to write to). This can be changed by creating $JENKINS\_HOME/jgroups.xml that describes the JGroups protocol stack configuration XML. See the [JBoss Clustering documentation](http://docs.jboss.org/jbossclustering/cluster_guide/5.1/html/jgroups.chapt.html#jgroups-transport) for the format of this file, as well as typical configuration tips and trouble-shooting.

### Jenkins Enterprise HA proxy WAR

To allow you to run OSS Jenkins WAR in the HA configuration, a separate "proxy WAR" is provided. This optional file can be downloaded from [the ha-proxy section](http://nectar-downloads.cloudbees.com/ha-proxy/) of the download site. The proxy WAR is a proxy to real Jenkins WAR file; you drop it in place of jenkins.war, and instead of acting as Jenkins by itself, it expects you to have the real thing in $JENKINS\_HOME/jenkins.war. The proxy WAR only controls the cluster control logic, and when it decides that it needs to act as the primary JVM, it’ll load $JENKINS\_HOME/jenkins.war into memory and runs that program.

This setup allows you to run newer versions of mainline Jenkins releases (for example to benefit from the latest and greatest features), or your custom build of Jenkins.

With the rpm/deb packages of Jenkins or Jenkins Enterprise, the easiest way to use this feature is to modify /etc/init.d/jenkins (for RPM) and /etc/default/jenkins (for DEB) and update JENKINS\_WAR variable to point to the proxy war. For other deployments, move the real jenkins.war into $JENKINS\_HOME, then put ha-proxy.war as jenkins.war.

The proxy WAR uses $JENKINS\_HOME in the same ways as Jenkins Enterprise WAR to pick up the network configuration and other parameters.

### Jenkins Enterprise HA monitor tool

Depending on the way you deploy the high-availability setup, some system level setup/teardown activities need to take place when the primary JVM changes. In the tutorial, we needed to run ifconfig to move the floating IP address. Other examples include DRBD mode change, updates to the front-end reverse proxy, or sending out an e-mail.

Because this needs to be executed when the primary Jenkins JVM becomes unresponsive, such setup/teardown scripts can only be reliably triggered from outside Jenkins. Those scripts also normally require root privileges to run. The Jenkins Enterprise HA monitor tool is a small background application that provides this role. It is an optional component that should be installed only when necessary. It can be downloaded from [the jenkins-ha-monitor section](http://nectar-downloads.cloudbees.com/jenkins-ha-monitor/) of the download site.

The monitor tool is packaged into a single jar file that can be executed as java -jar jenkins-ha-monitor.jar. It is also packaged as the jenkins-ha-monitor RPM/DEB packages for an easier installation. Its 3 defining options are as follows:

* The -home option that specifies the location of $JENKINS\_HOME. The monitor tool picks up network configuration and other important parameters from here, so it needs to know this location.
* The -host-promotion option that specifies the location of the promotion script, which gets executed when the primary Jenkins JVM moves in from another system into this system as a result of election. In native packages, this file is placed at /etc/jenkins-ha-monitor/promotion.sh.
* The -host-demotion option that specifies the demotion script, which is the opposite of the promotion script and gets executed when the primary Jenkins JVM moves from this system into another system. In native packages, this file is placed at /etc/jenkins-ha-monitor/demotion.sh.

Promotion and demotion scripts need to be idempotent, in the sense that the monitor tool may run the promotion script on an already promoted node, and the demotion script on an already demoted node. This can happen, for example, when a power outage hits a stand-by node and when it comes back up. The monitor tool runs the demotion script again on this node, since it cannot be certain about the state of the node before the power outage.

Run the tool with the -help option to see the complete list of available options.

For Clustering in OSS Jenkins add the plugin : cloudbees-enterprise-plugins