WHITE PAPER

7 Ways to Optimize Jenkins

By: Kohsuke Kawaguchi, Chief Technology Officer, CloudBees, and Jenkins Founder

cloudbees

Contents

INTRODUCTION

Jenkins is the world's most popular open source continuous integration software, with more than 65,000 sites using it to deliver superior-quality products. Jenkins provides two critical functions that help teams improve code quality:

- Continuously build and test
- Monitor jobs

Jenkins is widely recognized as a core tool for developers, QA engineers, project managers, release engineers, operations and managers alike. You can set it up to watch for any code changes in places like SVN and Git; automatically do a build with tools like Ant and Maven; initiate tests; perform actions like roll back; and set up alerts to notify you of any issues that arise along the way. Over the last few years, Jenkins has become the hub of the development lifecycle and has proven to be an efficient tool to deliver superior-quality code, increase productivity and reduce costs. Teams using agile methodology, in which continuous integration is a fundamental practice, will find Jenkins particularly useful.

This paper gives you seven easy techniques you can apply immediately to ensure a smoothly running Jenkins production server. While Jenkins is not difficult to set up and configure, you will get better results, support more projects and save administration time if you know the tips, tricks and optimal settings that can make your installation function most effectively. Even if you are already running Jenkins, it's not too late to implement these simple best practices that ensure reliable, optimized production operation.

ABOUT THE AUTHOR

Kohsuke Kawaguchi is the creator and community lead of the Jenkins continuous integration server, as well as the CTO of CloudBees®. He wrote the majority of the Jenkins core single-handedly and has been involved in JAXB, Metro web services stack, GlassFish v3, and RELAX NG at Sun Microsystems. Kohsuke is also known for a large number of open-source projects, such as args4j, YouDebug, com4j, Animal Sniffer, Sorcerer, wagon-svn, MSV and Parallel JUnit extension.

Backup and Restore

PROBLEM

Teams often procrastinate on performing backups. When disaster hits, they are left scrambling.

BACKGROUND

If you're like typical folks out there, you've probably been postponing backups because you have more important things to worry about. But as you surely know, it's very important to have a backup, and better late than never!

In addition to disaster recovery, Jenkins backups are useful insurance against accidental configuration changes, which might be discovered long after they were made. A regular backup system lets you go back in time to find the correct settings.

SOLUTION: JUST DO IT!

So don't wait, just do it! Fortunately, it's easy…

BACKUP PLANNING

Jenkins stores everything under the Jenkins Home directory, `$JENKINS_HOME`[1], so the easiest way to back it up is to simply backup the entire `$JENKINS_HOME` directory. Even if you have a distributed Jenkins setup, you do not need to backup anything on the build agent side.

Another backup planning issue is whether to do backups on live instances without taking Jenkins offline. Fortunately, Jenkins is designed so that doing a live backup works fine – configuration changes are atomic, so backups can be done without affecting a running instance.

TIPS AT A GLANCE

1. Make sure you have backups – better late than never
2. Plan disk usage – make sure it's expandable
3. For easier installation and migration, use native packages if possible
4. Do distributed builds
5. Use labels to optimize resource utilization and improve manageability
6. Make your Jenkins URL short and memorable
7. Discard old build records to keep your Jenkins instance healthy

DON'T BACK THESE UP

The following directories contain bits that can be easily recreated, so you don't need to include these in the backup:

- `/war` (exploded war)
- `/cache` (downloaded tools)
- `/tools` (extracted tools)

[1] To find the $JENKINS_HOME location, go to the Configure System menu.

OPTIMIZING BACKUPS

Optimization 1: Backup a Subset of `$JENKINS_HOME`

Although `$JENKINS_HOME` is the only directory you need to backup, there's a catch: this directory can become rather large. To save space, consider what parts of this directory you really need to backup and back them up selectively.

The bulk of your data, including your job configuration and past filed records, lives in the /jobs directory. The /jobs directory holds information pertaining to all the jobs you create in Jenkins. Its directory structure looks like this:

```
/jobs/*
- builds                    (build records)
- builds/*/archive          (archived artifacts)
- workspace                 (checked out workspace)
```

The `/builds` directory stores past build records. So if you're interested in configuration only, don't backup the builds. Or perhaps you need to keep build records but can afford to throw away archived artifacts (which are actually produced binaries). You can do this excluding `builds/*/archive`; note that these artifacts can be pretty big, excluding them may introduce a substantial savings.

Finally, the workspace directory contains the files that you check out for the version control systems. Normally these directories can be safely thrown away. If you need to recover, Jenkins can always perform a clean checkout, so there's usually no need to backup your workspace.

Optimization 2: Use OS-level Snapshots

If you want maximum consistency in your backups, use the snapshot capability in your file system. Although you can perform live backups, they take a long time to run, so you run the risk of taking different data at different time points…which may or may not be a real concern. Snapshots solve this problem.

Many file systems let you take snapshots, including Linux Logical Volume Manager (LVM) and Solaris ZFS (which also lets you take incremental backups). Some separate storage devices also let you create snapshots at the storage level.

CloudBees' enhanced Jenkins product, CloudBees Jenkins Enterprise™, can also help you with snapshots and backups. Look for more information about CloudBees Jenkins Enterprise in the Appendix.

TEST YOUR RESTORE!

Nothing is worse than thinking you have a backup and then when disaster hits, finding out you can't actually recover. So it's worth testing to make sure you have a proper backup.

The `JENKINS_HOME` directory is "relocate-able" – meaning you can extract it anywhere and it still works. Here's the easiest way to test a restoration:

1. Copy the backup Home directory somewhere on your machine, such as `~/backup_test`
2. Set JENKINS_HOME as an environment property and point to backup_test; for example, export `JENKINS_HOME=~/backup_test`
3. Run `java -jar jenkins.war --httpPort=9999`

This sequence of commands will pick up the new `JENKINS_HOME` with the `backup_test` directory. You can use this instance of Jenkins to make sure your backup works. Be sure to specify a random HTTP port so you don't collide with the real one – otherwise the server won't start!

HANDS-ON TRAINING:

Mastering Continuous Integration with Jenkins

Become a Jenkins expert, check out training courses offered by CloudBees:

https://www.cloudbees.com/jenkins/training

Plan for Disk Usage Growth Up Front

PROBLEM

Running out of disk space as your Jenkins installation grows, consuming more resources.

BACKGROUND

When you set up Jenkins, the most important disk planning consideration is to prepare for inevitable growth. Jenkins disk usage can grow quickly, particularly when you start hosting multiple jobs.

SOLUTION: PREPARE FOR DISK USAGE GROWTH

So don't wait, just do it! Fortunately, it's easy…

Spanned Volume on Windows

On NTFS devices (e.g. on Windows), you can create a spanned volume: take an existing volume, add a new one at the end, and then make the combined volume behave as a single volume. Then it's simple to add new disks.

In this case the only planning you need to do is to put Jenkins in a separate partition when you install it, so you can convert to a spanned volume later.

LVM Volume Manager

On Linux, the LVM volume manager does something similar, but unlike Windows, you have to configure the LVM up front: set up LVM, create the Volume Group and Logical Volume, and set up a volume in your manager. One you have the first volume in LVM, you can expand the files system and add more disks later[2].

Solaris ZFS

Solaris ZFS is much more flexible and thus very easy to prepare for disk expansion. On ZFS, `$JENKINS_HOME` should be on its own file system, which allows you to easily create snapshots, backups and other nice things.

Use Symlinks

If you already have Jenkins running and cannot use any of the solutions above, use symbolic links (symlinks) to ease your pain. Simply identify the jobs that take up a lot of disk space, copy those directories into separate volumes and then symlink to those directories.

Use Native Packages

PROBLEM

Two problems, actually: you want to be able to simplify migration of instances to different machines, as well as start Jenkins upon machine startup.

BACKGROUND

Java developers aren't always aware of some of the powerful features in their underlying operating systems, so I want to mention some highlights that will help with Jenkins. For example, developers tend to ignore the OS-specific installation packages as they start building their Jenkins environment. As the system grows, these packages help ease migrations to other machines and are also useful to start Jenkins as the machine comes up.

---

[2] Refer to Linux docs for detailed steps.

SOLUTION: USE OS-SPECIFIC INSTALLATION PACKAGES OVER THE DEFAULT WAR-BASED INSTALLATION.

In addition to the Jenkins war, Jenkins is available as OS-installable packages for Debian, RedHat/CentOS and Suse[3]. Both installing and upgrading Jenkins is much easier with these packages. So unless you have to choose a specific application server or have some special requirements on your application server, package-based installation is highly recommended.

It's also more reproducible, so if you need to move your Jenkins service into another machine, this native package installation is much easier than going through all the steps of setting up your corresponding application server.

The native packages come with an `init` script, where a daemon starts up Jenkins after you boot the machine and runs the Jenkins process.

Configuration files follow the OS level conventions, and boot-up parameters for Jenkins are located in these files:

- `etc/default/Hudson` for Debian
- `etc/sysconfig/Hudson` for RPM

Take Advantage of Distributed Builds

PROBLEM

You will eventually outgrow the ability to run builds on just one machine. Additionally, single systems do not take advantage of the full power of Jenkins.

BACKGROUND

I am always surprised at how people make do with a single-system Jenkins. I can promise you will grow beyond a single system…the real question is when.

SOLUTION: DO DISTRIBUTED BUILDS!

One best practice I can't recommend enough is to try out Jenkins' distributed builds. Dealing with the amount of compression load is not the only reason to use distributed builds. You also need isolation between builds. For example, when your tests depend on local resources like a local database or particular TCP/IP port, you can't use the same machine to run tests that access those resources. And while you can

[3] Windows installer is in the works and hopefully will be available soon.

always work around these problems by tweaking your build script and tests and such, it's much easier if you have boxes that provide natural isolation.

Another driving factor for distributed builds is that if you're testing against multiple platforms, you often want to have more diversity in the environment. This situation inherently calls for multiple systems.

To get started with distributed builds and create Jenkins build agents to connect to your master, do the following:

1. Choose *Manage Jenkins*, then *Manage Nodes* (or just click *Build Executor Status*).

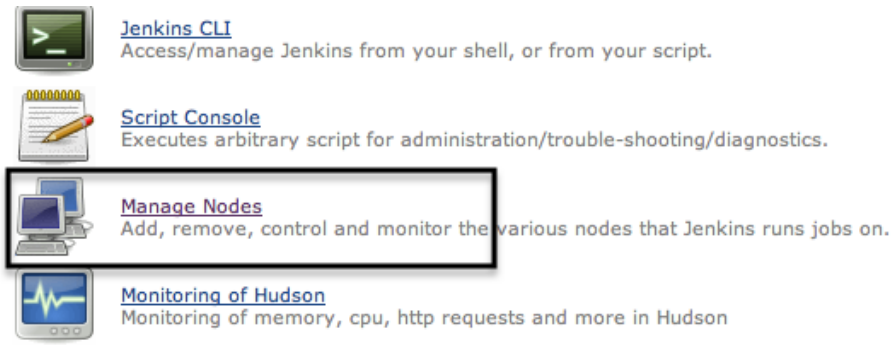

Figure 1: Managing nodes

2. Create a new node and name it (e.g. NewBuildAgent, as shown in Figure 2).
    - » The screen in Figure 3 appears. Here you can enter configuration information for the Jenkins master to connect to the build agent.
    - » The Master will bring up the build agent and then start allocating jobs to it.

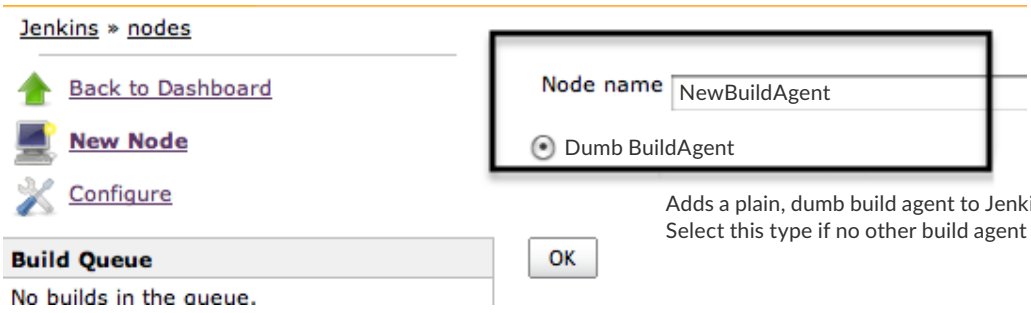

Figure 2: Creating a new node

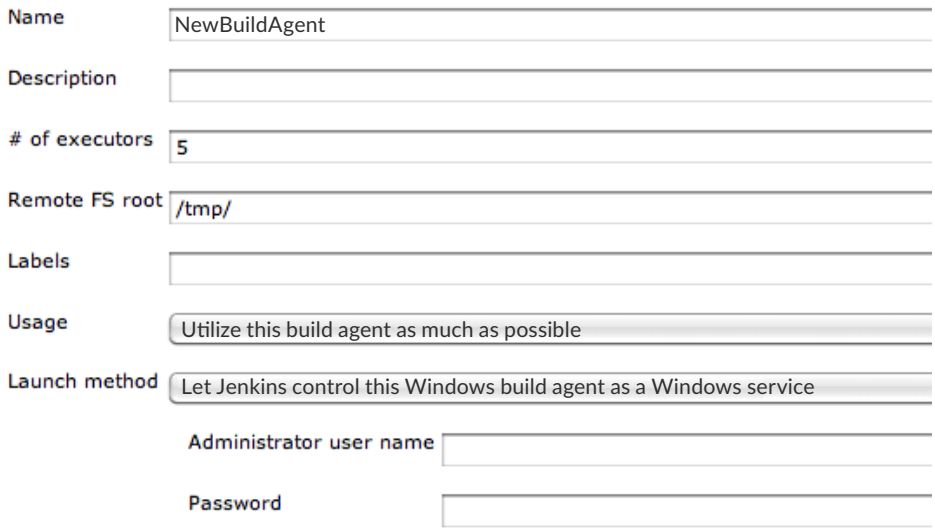

Figure 3: Configuring a new node

And that's it! Now you have a cluster of machines onto which Jenkins can delegate the load.

BONUS TIP: BUILD AGENT RECONNECTIVITY

Once you're doing distributed builds, let the masters launch the build agents if you can. Allow the Jenkins master to proactively talk to the build agent and then bring the instance up. This way there is no intervention from the client side; the master will bring up the build agents as it allocates jobs or bring the build agent online if the build agent machine goes down. Jenkins also supports cases where the build agent should initiate connection back to the master.

To set connectivity, specify the "Launch method" option as you configure settings for a new build agent (as shown in Figure 3). You have four options:

- Let Jenkins control this Windows build agent as a Windows service: Used on Windows to start Windows services remotely. Jenkins uses DCOM to start the build agent services.
- Launch build agents on Unix via SSH: Jenkins remotely logs into the build agent machine from the master and starts the build agent.
- Launch build agent via an execution of command on the Master: User needs to write a script that Jenkins can use after it logs into the build agent (typical expectation is that the user will provide access to slave.jar). This option is used when the master has access to the build agent using SSH or RSH.

- Launch build agents via JNLP: Used when build agents have to initiate contact with the master. Jenkins uses the JNLP protocol to launch.

Use Labels

PROBLEM

Managing a diverse set of platforms and machines easily, and making machines interchangeable.

BACKGROUND

A continuous integration environment is a mixed bag of machines, platforms and operating systems. You need the utmost flexibility in managing these machines, you want your build machines to be interchangeable, and in general you don't want to tie builds to a specific build machine. But sometimes you need more diversity in the build cluster. Your machines might not be entirely homogeneous and you still need a way to identify some subset of them; for example, you might need a certain job to run on Linux instead of Windows.

SOLUTION: MAKE USE OF LABELS

A very useful but under-used Jenkins feature is labels. Labels are simply tags you can assign to nodes to describe their capacities. Some typical useful labels includ:

- Operating system
- 32 vs. 64-bit
- Additional infrastructure that exists only on certain machines (for example, WebSphere)
- Machine's geographical location

Assign labels on the build machines themselves; then on the job side, specify that the job needs to run in a certain place based on label criteria. Instead of tying jobs to individual build machines, labels give Jenkins flexibility to choose where to run the builds, which results in better resource utilization and promotes manageability.

SAVE TIME WITH PUBLIC KEY AUTHENTICATION

I also recommend taking advantage of the SSH public key authentication in Jenkins. By setting up the public key mechanism, you can log in from one system to another without ever typing your password. This time-saver is useful for Jenkins and all kinds of administration or automated tasks. For example, if you want to write a script in these multiple systems to copy files, it's imperative you are able to do that without requiring a password from the console. Setting up SSH public key authentication is very easy and only takes about five minutes, so do it if you haven't already.

Using labels is very easy:

1. Select a build agent machine and choose Configure.
2. Specify a label in the Labels field.
3. Create a new job, name it and fill out any other necessary parameters, and click OK.
4. Click the checkbox, Restrict where this project can be run (for example amd64, linux and sanfrancisco).
5. Fill in the Label Expression that matches the label on your build agent machine (or any label).

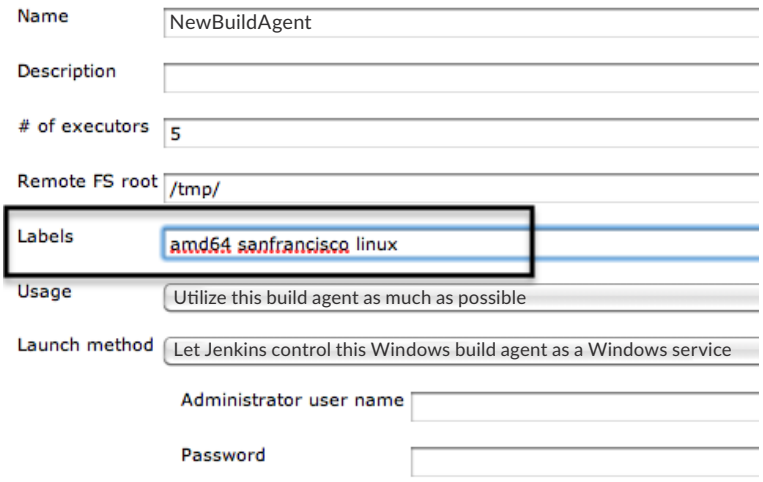

Figure 4: Configuring labels

One final reason to use labels: if a machine goes down, Jenkins has the flexibility to shift the load to another machine with a compatible label, which gives you time to diagnose and fix the problem. This way you can increase the level of service for your users without service disruption.

Use a Memorable URL for Jenkins

PROBLEM

Jenkins is most useful if development teams refer to it often. Using an IP address or mangled name for the running instance of Jenkins makes it hard to remember and inhibits team adoption.

BACKGROUND

If your users can't see Jenkins, much of the benefit is lost. In some places, Jenkins is referred to by IP address, but that's hard to remember.

A FEW SOLUTIONS:

Invest in an Easy-to-Remember URL

The point of the continuous integration server is to become visible, so if Jenkins has a long URL that is hard to remember, your team won't use it as much.

BAD: `http://sca12-3530-sca.cloudbees.com:8080/hudson`

GOOD: `http://jenkins.cloudbees.com`

Use the Service Name

Although your machine may already have hosting, you don't have to use it. In fact, it's often a bad idea to use the primary machine name to point to a particular service, because the service might move later to another system. So instead of using the primary machine name, use the host alias.

If your IT operations guys aren't helping you create a host alias, you can also use the external dynamic DNS services. Your hosting will be visible outside, but your machine won't be, so this method is still secure. Using the Service name makes Jenkins relocatable: if you later move Jenkins to a more powerful machine – or are running multiple services on a single system, and later decide that both services need to move to respective machines – you can execute the move without disrupting the services.

Run Jenkins on Port 80 with Other Apps

Another useful tip: it's worthwhile to run Jenkins on the default port (80) instead of a custom port (like 8080) – then you don't have to specify the port number. You can do this on Unix systems through Apache *reverse proxy*.

In reverse proxy, the browser talks to Apache and Apache forwards the HTTP request to Jenkins, which is running on a custom port like 8080. Another benefit here is you can run Jenkins as non-root user.

On Windows, it's harder to share port 80, but now you can install a free IIS7 module called *URL Rewrite + ARR*, which lets you achieve the same thing.

Don't Use the Appserver Context Root to Jenkins as a Way to Remember the Jenkins URL

By default in many application servers, if you deploy the Jenkins war, you get a default context-root `/Jenkins`. This is redundant – use a virtual host to distinguish multiple apps, not the `/Jenkins` context path.

Prevent Build Record Sprawl

PROBLEM

Build records can accumulate quickly. Keeping too many around increases memory usage and impacts startup times.

BACKGROUND

Keeping build records under control is related to the disk planning we discussed in Tip #1. If possible, it's best to discard all build records. Keeping too many negatively impacts the startup time of Jenkins and gradually increases memory usage over time. Some jobs in some environments really do require you to keep all the build records (for example, if you're doing a release of Jenkins, you don't want to lose those records). But for most jobs, you can throw all the records away. Since you're probably running serial continuous integration builds on Jenkins, it's not useful to have records of all the continuous integration builds since inception.

SOLUTION: KEEP ONLY A SUBSET OF BUILD RECORDS AROUND

You can set Jenkins up to clean house by checking Discard Old Builds in the Configure menu and filling in one or both of this option's two text boxes. You can choose to throw the old records away if they are more than 30 days old, or you can choose to keep only the last X records and discard everything else.

Figure 5: Configuring Jenkins to discard old builds

The whole point of controlling your build records is to avoid unbounded consumption. Sometimes you want to keep some records for a while so people can look at problems when there are failures…so don't make the number too low. You really just need to have a fixed cap; 50 or 100 is a good number. Note that you can configure build record housekeeping on a per-project basis.

Another way to keep specific records is to use the fingerprinting feature, which allows you to create an association between jobs. As you enable fingerprinting, you can enable Jenkins to keep a build log of dependencies – this captures all the upstream build log dependencies.

And there you have it – seven ways you can get the best performance from Jenkins continuous integration server! Of course, if you want to save time and money and optimize performance even further, CloudBees will be happy to manage Jenkins for you in the cloud with DEV@cloud™, or provide ongoing support and enhanced features through our CloudBees Jenkins Enterprise subscription service.

Jenkins and CloudBees

CloudBees is the only cloud company focused on servicing the complete develop-to-deploy lifecycle of Java web applications in the cloud – where customers do not have to worry about servers, virtual machines or IT staff. We are also the world's premier experts on Jenkins and are dedicated to helping teams make the most of their Jenkins continuous integration servers.

CloudBees Jenkins Enterprise, CloudBees' on-premise, fully-supported enterprise Jenkins package, gives you:

- Ongoing support from Jenkins experts
- VMware scaling for your Jenkins environment
- Enterprise features that extend Jenkins for large and mission-critical installations
- Seamless rollover to cloud-based Jenkins during peak usage times and easy transition to cloud-based production deployment, through integration with the CloudBees DEV@cloud service

DEV@cloud, affectionately also known as Jenkins as a Service, lets you:

- Scale your Jenkins environment with the power of the cloud – start building immediately in a fully-tested, robust environment and grow when you need to grow
- Ease Jenkins management overhead – spend your time writing code, not maintaining servers
- Speed your Jenkins builds – access to unlimited build agents whenever you need them
- Save money with on-demand Jenkins Service – only pay for what you use

Please get in touch with us if you'd like more information about CloudBees Jenkins Enterprise, DEV@cloud or Jenkins in general – we'd be delighted to help you! You can contact us at info@cloudbees.com.

APPENDIX A:

Additional Jenkins Resources

Jenkins Training

Mastering Continuous Integration with Jenkins training gets you off to a quick start with Jenkins.

All instructors are Jenkins experts: http://cloudbees.com/training.cb

Learn More About CloudBees Jenkins Enterprise

https://www.cloudbees.com/products

Learn More About DEV@cloud

Jenkins as a Service: https://www.cloudbees.com/products/jenkins-cloud

Subscribe to *Continuous Information*

Get All of the Latest Jenkins News: https://www.cloudbees.com/jenkins/newsletter

Videos/Recordings about the CloudBees Offerings

https://www.cloudbees.com/resources/webinar-recordings

Watch CloudBees Videos on YouTube

http://www.youtube.com/CloudBeesTV

Read the CloudBees Blog

Get industry insights and technical tips from CloudBees experts: https://www.cloudbees.com/blog/

APPENDIX B:

Easier Backups with CloudBees Jenkins Enterprise

In addition to providing professional support for open source Jenkins, CloudBees Jenkins Enterprise delivers additional plugins that are available only through CloudBees.

Jenkins backup functionality is one such feature. As mentioned in Tip 1, making backups is crucial. The Backup Scheduling plugin from CloudBees Jenkins Enterprise greatly simplifies the job of doing backups: just create a backup by creating a new type of job called Backup Jenkins.

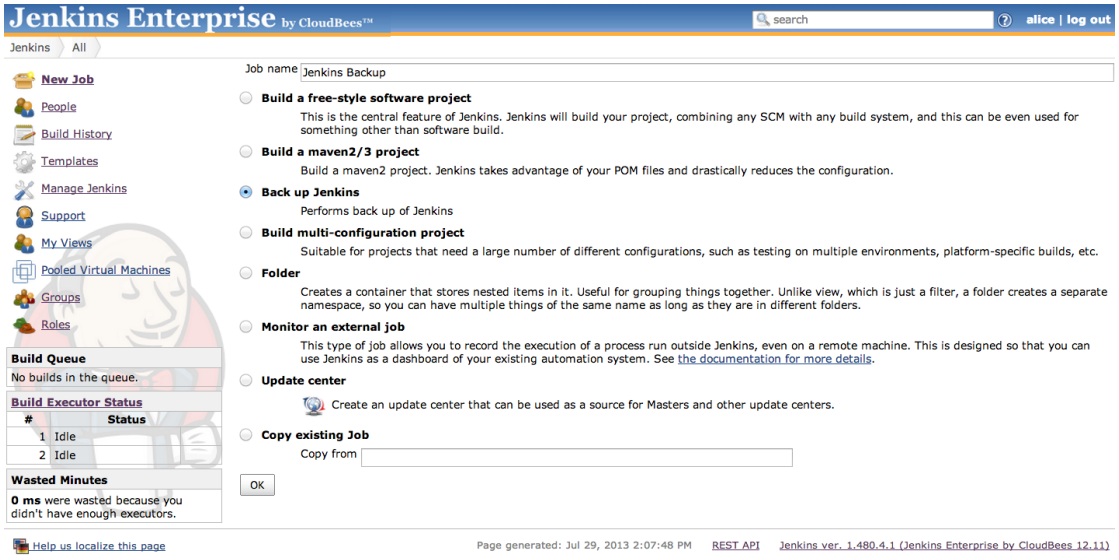

Figure A: CloudBees Jenkins Enterprise – Backup Jenkins job type

Selecting the Backup Jenkins option brings up additional configuration options on the Job Configuration page. Here you can choose to backup job configurations, build records, system configurations or any combination thereof. You don't have to write scripts or cron jobs to perform these backups, as you would if implementing Tip 1 without CloudBees Jenkins Enterprise.

In addition, since a backup job is a Jenkins job, you can easily relocate the configuration to a different system if required – no porting of shell scripts! Figure B below shows how to use the Jenkins Build Periodically feature to do a daily backup of all the configuration information, which includes Job Configurations, Build Records and System Configuration. We could also have easily chosen to backup a subset of information as outlined in Tip 1.

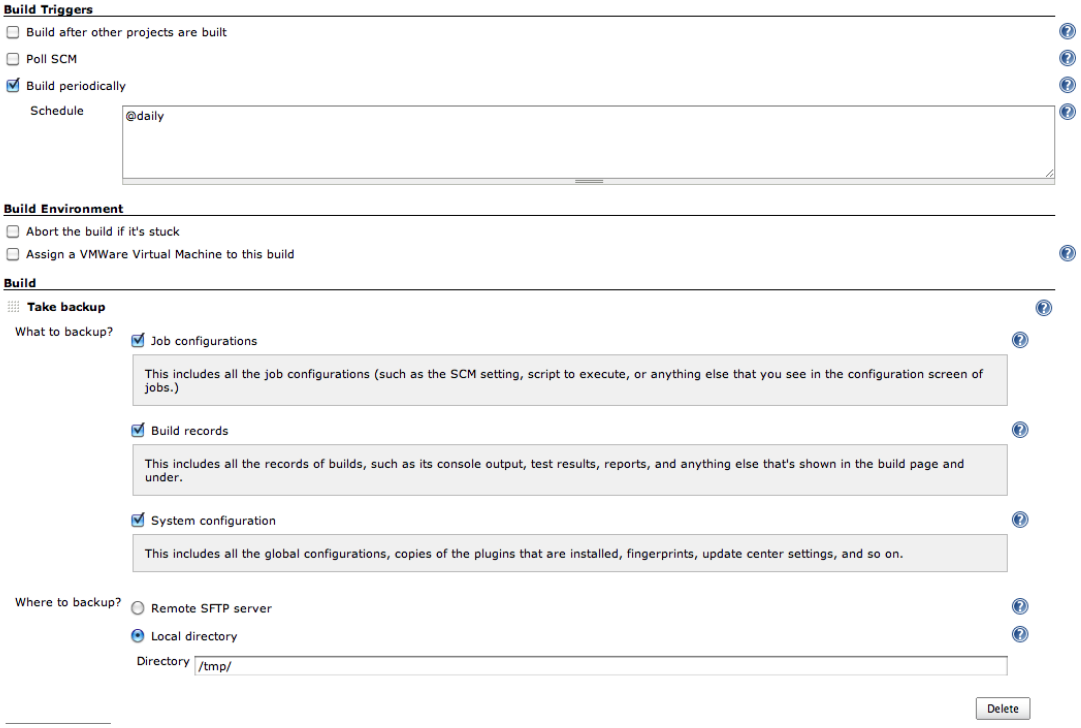

Figure B: Backup configuration

In addition to premium support and automated backup capability, CloudBees Jenkins Enterprise also includes auto-scaling for VMware and many other features.

CloudBees, Inc.
2001 Gateway Place, Suite 670W | San Jose, CA 95110 | United States
www.cloudbees.com | info@cloudbees.com

0816v01