**Chapter 13. Maintaining Jenkins**

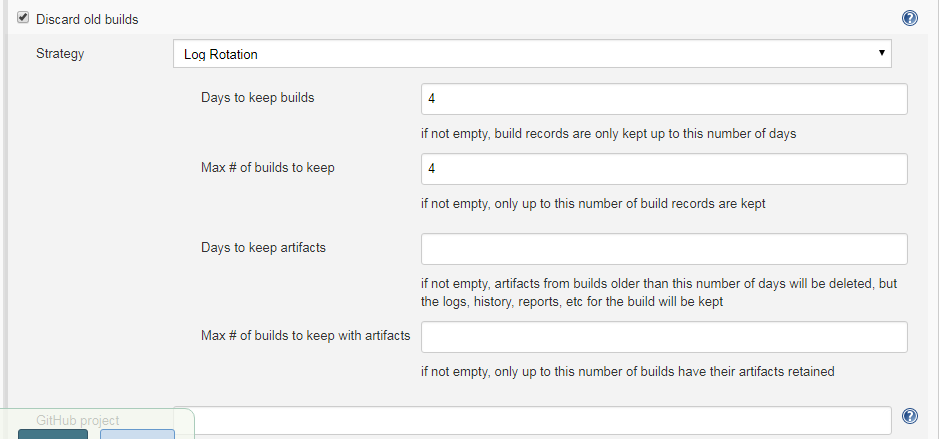
**Introduction**

In this chapter, we will be discussing a few tips and tricks that you might find useful when maintaining a large Jenkins instance. We will look at things like how to limit, and keep tabs on, disk usage, how to give Jenkins enough memory and how to archive build jobs or migrate them from one server to another. Some of these topics are discussed elsewhere in the book, but here we will be looking at things from the point of view of the system administrator.

**Monitoring Disk Space**

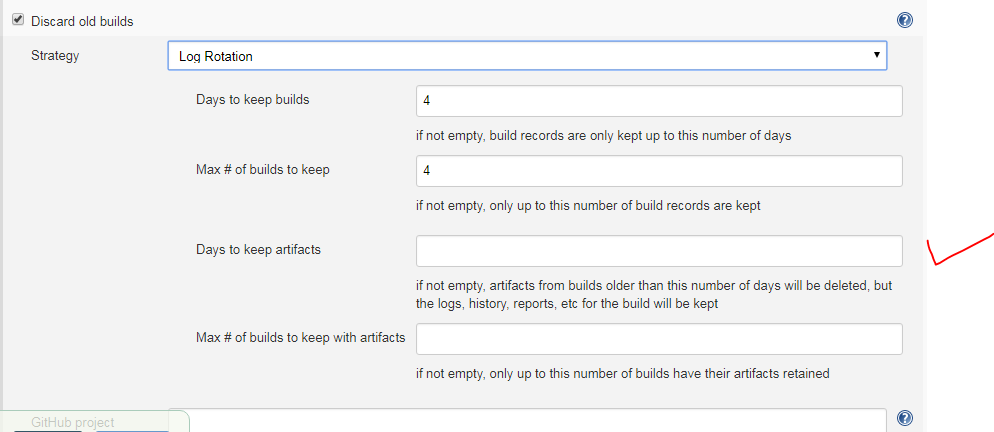
Build History takes disk space. In addition, Jenkins analyzes the build records when it loads a project configuration, so a build job with a thousand archived builds is going to take a lot longer to load than one with only fifty. If you have a large Jenkins server with tens or hundreds of build jobs, multiply this accordingly.

Probably the simplest way to keep a cap on disk usage is to limit the number of builds a project maintains in its history. You can configure this by ticking the Discard Old Builds checkbox at the top of the project configuration page. If you tell Jenkins to only keep the last 20 builds, it will start discarding (and deleting) older build jobs once it reaches this number. You can limit them by number (i.e., no more than 20 builds) or by date (i.e., builds no older than 30 days). It does this intelligently, though: if there has ever been a successful build, Jenkins will always keep at least the latest successful build as part of its build history, so you will never loose your last successful build.

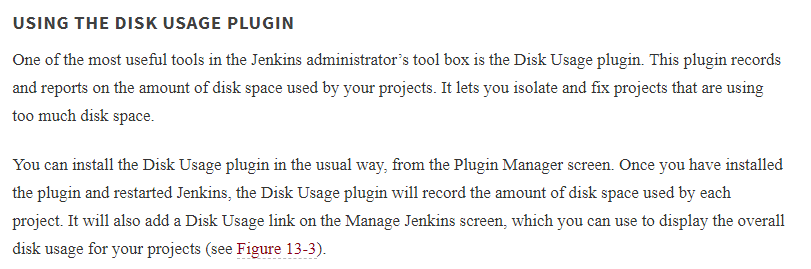


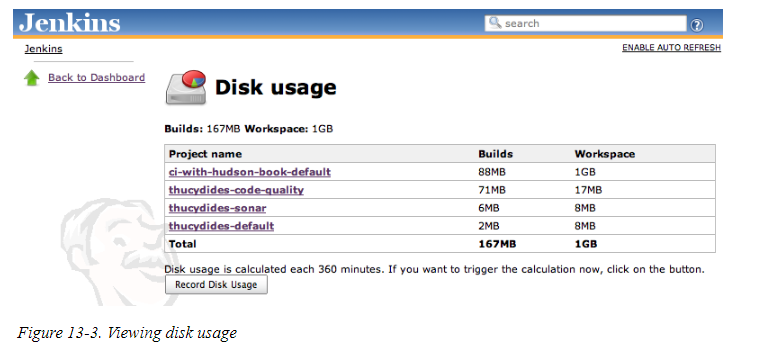
The problem with discarding old builds is that you loose the build history at the same time. Jenkins uses the build records to produce graphs of test results and build metrics. If you limit the number of builds to be kept to twenty, for example, Jenkins will only display graphs containing the last twenty data points, which can be a bit limited. This sort of information can be very useful to the developers, but it is often good to be able to see how the project metrics are doing throughout the whole life of the project, not just over the last week or two.

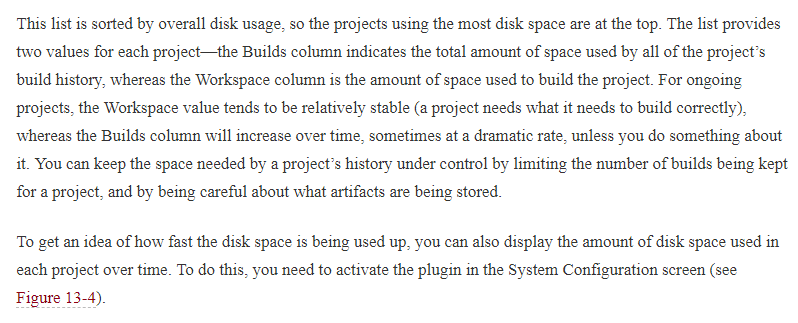
Fortunately, Jenkins has a work-around that can keep both developers and system administrators happy. In general, the items that take up the most disk space are the build artifacts: JAR files, WAR files, and so on. The build history itself is mostly XML log files, which don’t take up too much space. If you click on the“Advanced...” button, Jenkins will let you discard the artifacts, but not the build data. For example, we have configured Jenkins to keep artifacts for a maximum of 7 days. This is a great option if you need to put a cap on disk usage, but still want to provide a full scope of build metrics for the development teams.

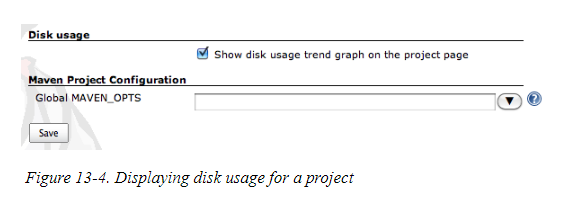


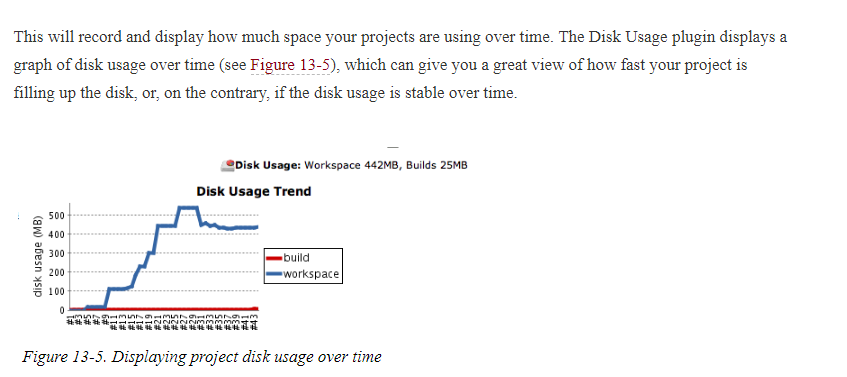
Don’t hesitate to be ruthless, keeping the maximum number of builds with artifacts quite low. Remember, Jenkins will always keep the last stable and the last successful builds, no matter what you tell it, so you will always have at least one working artifact (unless of course the project has yet to successfully build). Jenkins also lets you mark an individual build as “Keep this log forever”, to exclude certain important builds from being discarded automatically.

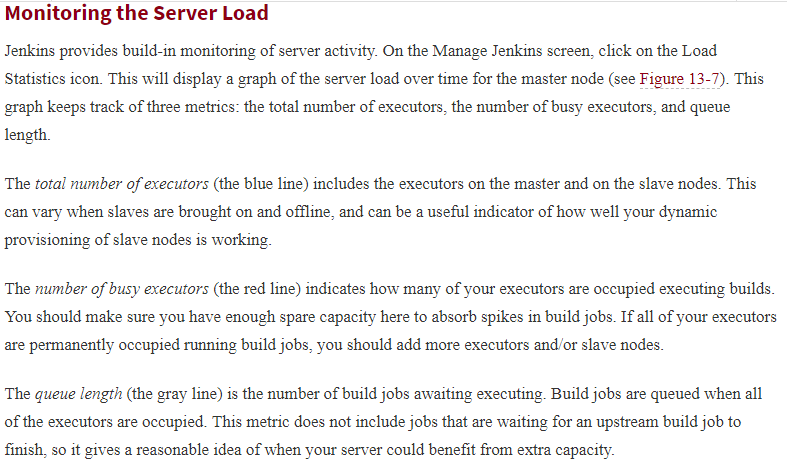


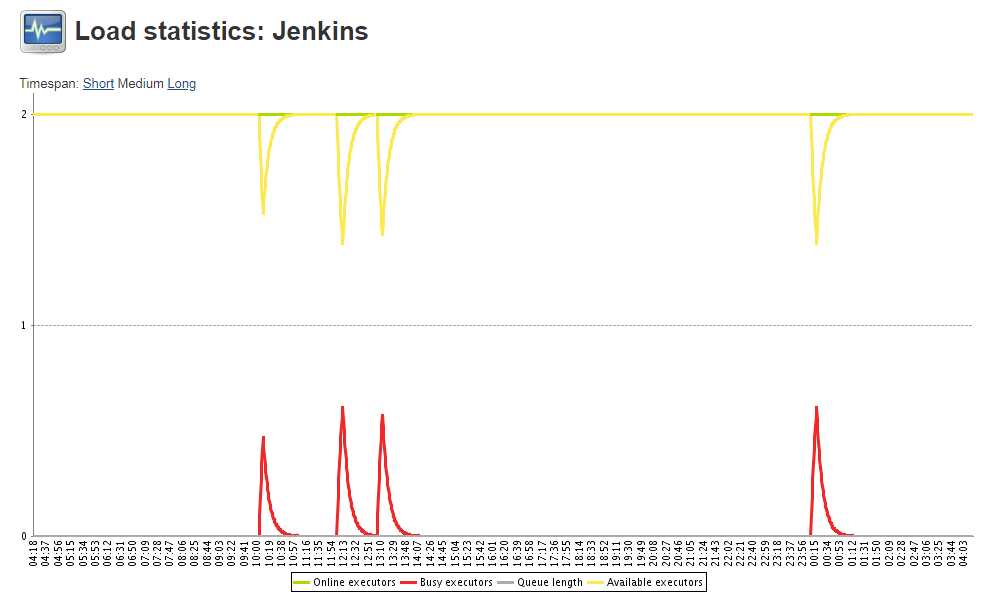


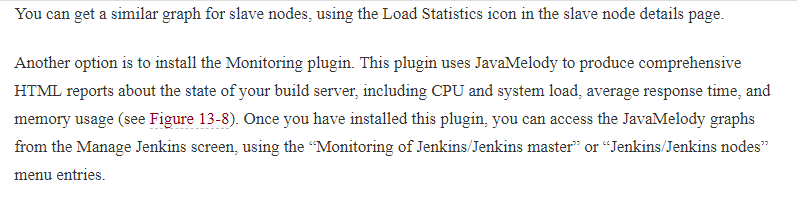


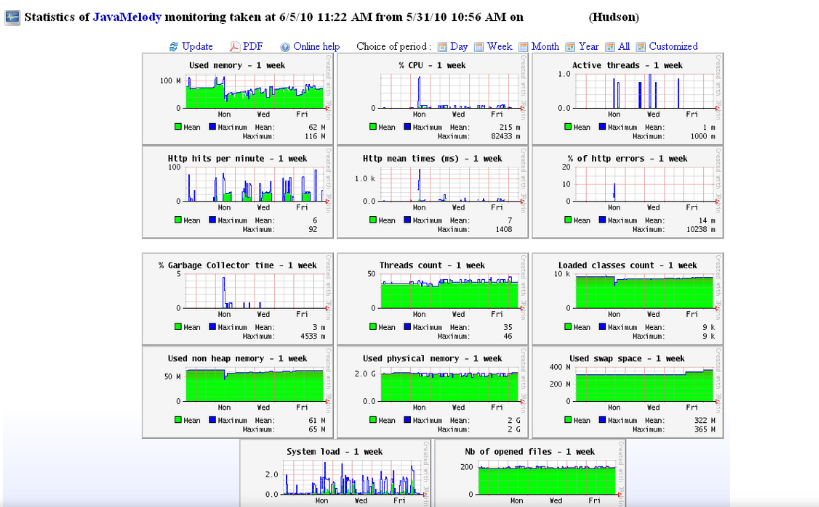






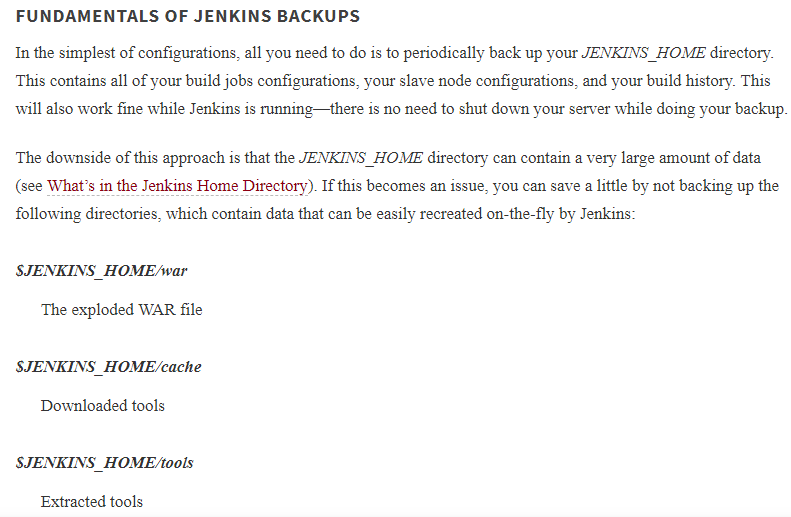


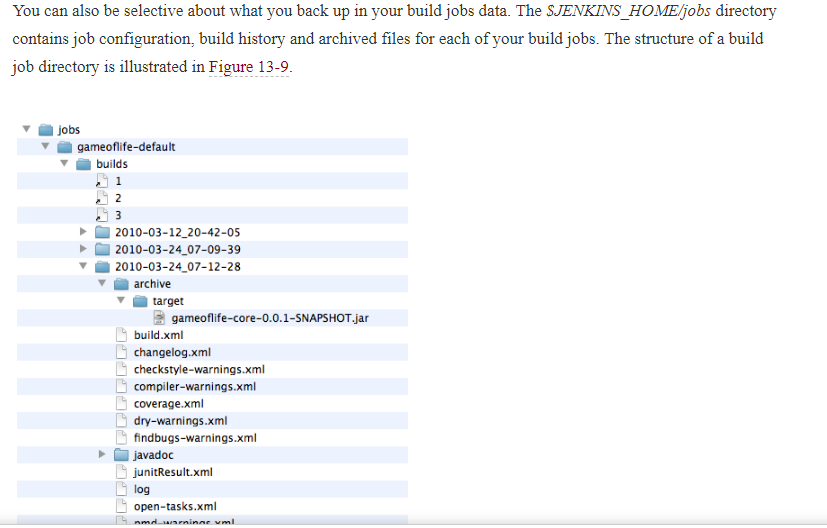


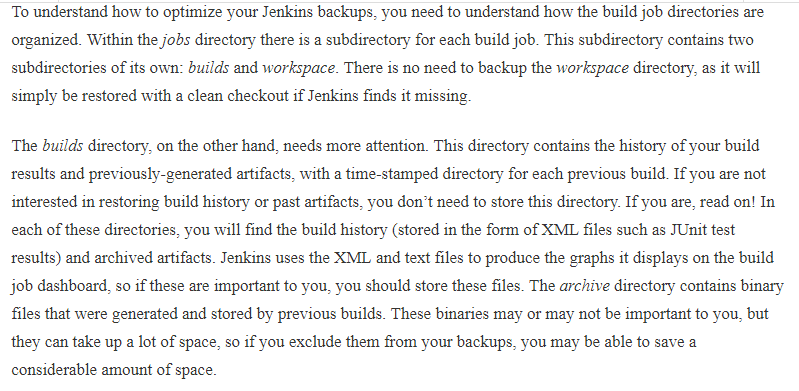


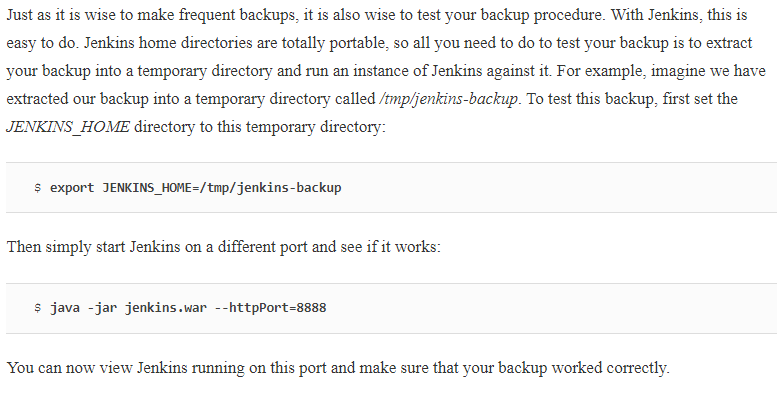
## Backing Up Your Configuration

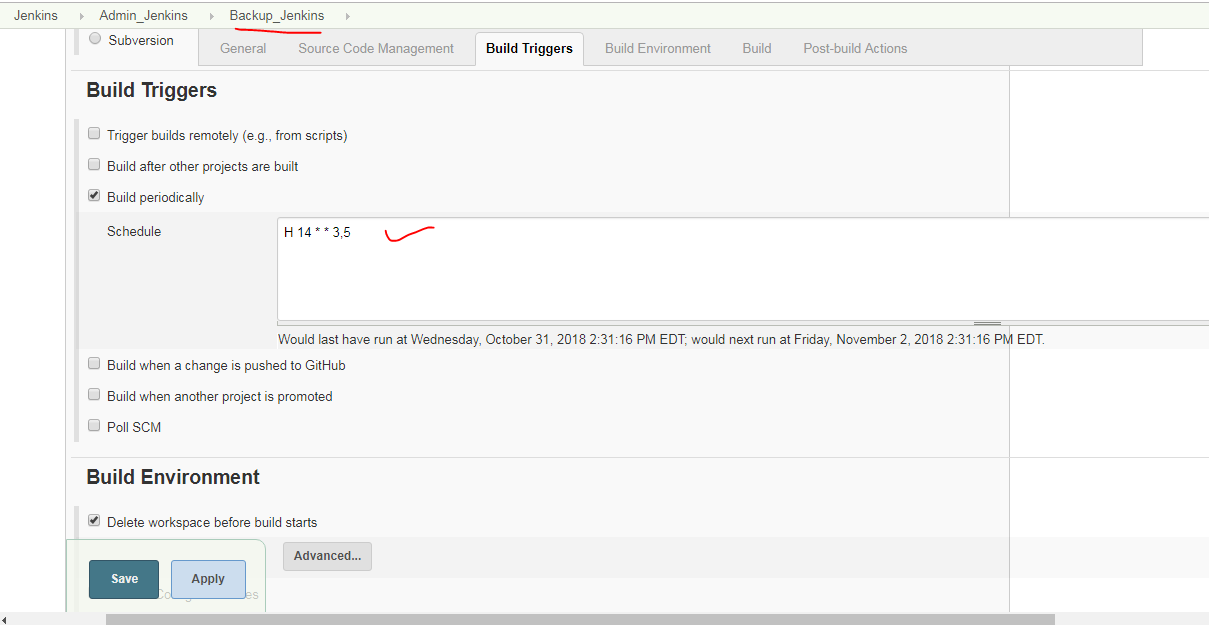
Backing up your data is a universally recommended practice, and your Jenkins server should be no exception. Fortunately, backing up Jenkins is relatively easy. In this section, we will look at a few ways to do this.



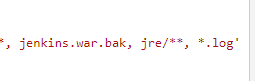












## Migrating Build Jobs

There are times when you need to move or copy Jenkins build jobs from one Jenkins instance to another, without copying the entire Jenkins configuration. For example, you might be migrating your build jobs to a Jenkins instance on a brand new box, with system configuration details that vary from the original machine. Or you might be restoring an old build job that you have archived.

As we have seen, Jenkins stores all of the data it needs for a project in a subdirectory of the jobs directory in your Jenkins home directory. This subdirectory is easy to identify—it has the same name as your project. Incidentally, this is one reason why your project names really shouldn’t contain spaces, particularly if Jenkins is running under Unix or Linux—it makes maintenance and admin tasks a lot easier if the project names are also well-behaved Unix filenames.

You can copy or move build jobs between instances of projects simply enough by copying or moving the build job directories to the new Jenkins instance. The project job directory is self-contained—it contains both the full project configuration and all the build history. It is even safe enough to copy build job directories to a running Jenkins instance, though if you are also deleting them from the original server, you should shut this one down first. You don’t even need to restart the new Jenkins instance to see the results of your import—just go to the Manage Jenkins screen and click on Reload Configuration From Disk. This will load the new jobs and make them immediately visible on the Jenkins dashboard.

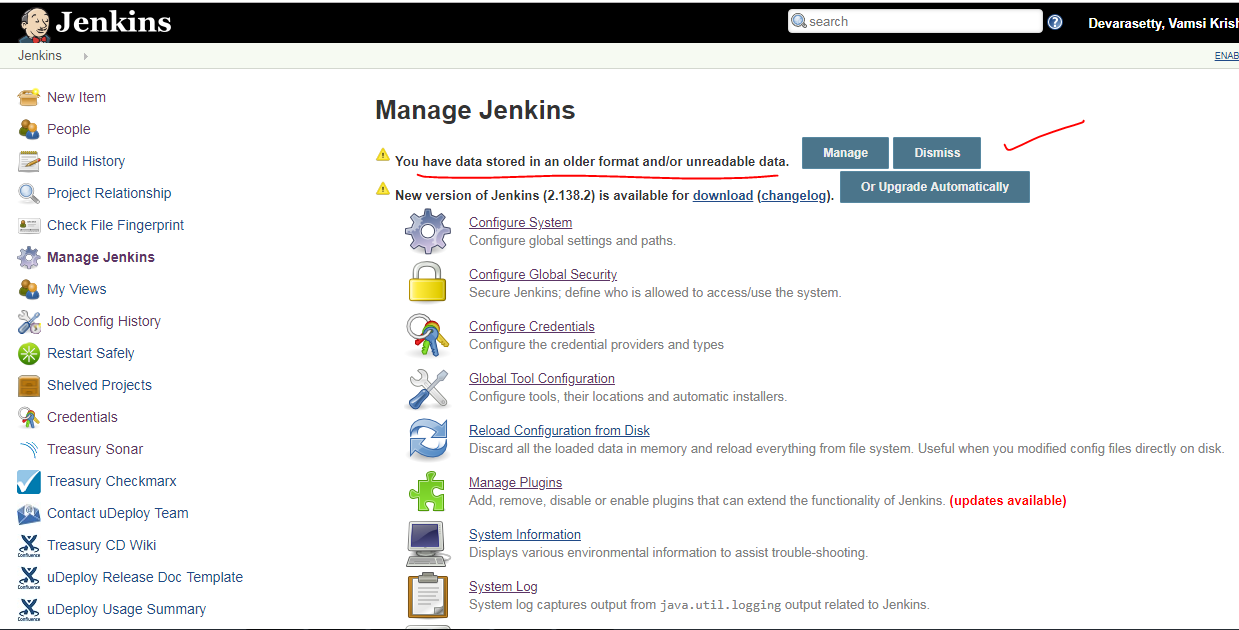
There are a few gotchas, however. If you are migrating your jobs to a brand new Jenkins configuration, remember to install, or migrate, the plugins from your original server. The plugins can be found in the pluginsdirectory, so you can simply copy everything from this directory to the corresponding directory in your new instance.

Of course, you might be migrating the build jobs to a new instance precisely because the plugin configuration on the original box is a mess. Some Jenkins plugins can be a bit buggy sometimes, and you may want to move to a clean installation with a well-known, well-defined set of vetted plugins. In this case, you may need to rework some of your project configurations once they have been imported.

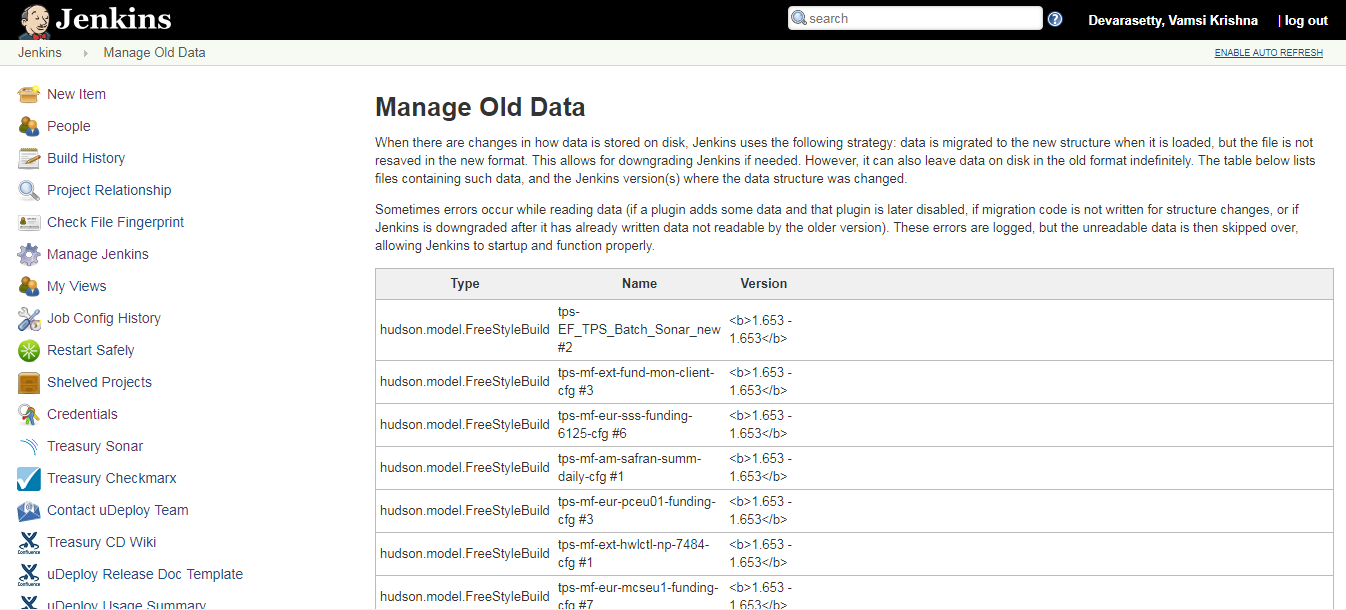
The reason for this is straightforward. When you use a plugin in a project, the project’s config.xml will be updated with plugin-specific configuration fields. If for some reason you need to migrate projects selectively to a Jenkins installation without these plugins installed, Jenkins will no longer understand these parts of the project configuration. The same thing can also sometimes happen if the plugin versions are very different on the machines, and the data format used by the plugin has changed.

If you are migrating jobs to a Jenkins instance with a different configuration, it also pays to keep an eye on the system logs. Invalid plugin configurations will usually let you know through warnings or exceptions. While not always fatal, these error messages often mean that the plugin will not work as expected, or at all.

Jenkins provides some useful features to help you migrate your project configurations. If Jenkins finds data that it thinks is out of date or invalid, it will tell you so. On the Manage Jenkins screen, you will get a message like the one in



From here, you can choose to either leave the configuration as it is (just in case you roll back to a previous version of your Jenkins instance, for example), or let Jenkins discard the fields it cannot read. If you choose this option, Jenkins will bring up a screen containing more details about the error, and can even help tidy up your project configuration files if you wish



This screen gives you more details about the project containing the dodgy data, as well as the exact error message. This gives you several options. If you are sure that you no longer need the plugin that originally created the data, you can safely remove the redundant fields by clicking on the Discard Unreadable Data button. Alternatively, you may decide that the fields belong to a useful plugin that hasn’t yet been installed on the new Jenkins instance. In this case, install the plugin and all should be well. Finally, you can always choose to leave the redundant data and live with the error message, at least until you are sure that you won’t need to migrate the job back to the old server some day.

However, Jenkins doesn’t always detect all of the errors or inconsistencies—it still pays to keep one eye on the system logs when you migrate your build jobs. For example, the following is a real example from a Jenkins log file showing what can happen during the migration process:

