

---

# 3-Tier System Administration

---

by Kurt Keller  
Hilti Aktiengesellschaft

**SUMMIT**

JBoss  
WORLD

PRESENTED BY RED HAT



last compilation time: Wednesday 26<sup>th</sup> May, 2010 11:19 

# Contents

<b>1</b>	<b>What is wrong?</b>	<b>2</b>
<b>2</b>	<b>What is better?</b>	<b>3</b>
2.1	The Software Tier	4
2.2	The Configuration Tier	4
2.3	The Data Tier	5
<b>3</b>	<b>Classifying the Files</b>	<b>6</b>
<b>4</b>	<b>Rules for Tiers</b>	<b>7</b>
<b>5</b>	<b>Why should you care?</b>	<b>9</b>

## 1 What is wrong?

Not only for Unix, but also for Linux, the paradigm “*Everything is a File*” holds true. So by oversimplifying a little, we could say the job of a system administrator is to take care of all the files on the servers. Sure, there is software to install, there are services to run and to configure, users to manage and logs to analyze. But in the very end, each of these tasks is done by managing the applicable files. Installing software means placing a bunch of files in the proper place on a system. Configuring and running a service is to manipulate certain files in the proper way. Logins for users are created by adding their credentials to the appropriate files and all the settings for how a user account is setup end up in files too. Finally, log entries are usually written to files as well.

Ask a number of Unix/Linux system administrators to classify the files on the servers they manage. To help them understand your expectation, give them a table like the one in figure 1 to fill in.

Attribute Item	Item Value	Change Frequency	Restore Method	Deployment Method

Figure 1: Empty Table

What will this table look like when it is filled in and returned to you? Actually, you might give it a shot yourself and see what you come up with. Better even, have each of your fellow administrators also fill in the table and then consolidate it.

Of course, the outcome will be different for each group of system administrators, but often it will resemble something which could be dubbed “1.5-Tier System Administration”, meaning 3-Tier System Administration done half-heartedly. A typical 1.5-Tier table might look similar to the one in figure 2 on the next page.

So what is wrong with this? Actually several things are amiss.

Both of the items “Software” and “Data” slightly stretch into the “Configuration” layer, which itself reaches into all three layers. So there is no clear line. The rationale behind this is not surprising, though. Clearly there are configuration files which are rather static and rarely change, if ever, while other configuration files may need rather frequent tinkering with. Most of the software on a system, hopefully, is installed automatically but some custom scripts might be deployed by manually copying them to the system. Similarly, while scripts commonly are perceived as being software, it is not uncommon to have scripts with embedded

Attribute Item	Item Value	Change Frequency	Restore Method	Deployment Method
Software	\$\$\$	with every patch	restore from backup	partially automatic: kickstart, rpm, setup
Configuration	no idea	it depends	restore or recreate	usually manually: copy, create, restore
Data	high	changes constantly	restore from backup	usually manually: external, restore

Figure 2: Typical “1.5-Tier” Table

configuration which periodically may need customization. This all makes the separation somewhat fuzzy.

Another problem with the presented categorization is that the values of some attributes are not comparable. The dollar signs in the “Item Value” column signify a monetary value, which is incomparable to a value such as high in the data layer.

There are more problems which we will not specifically point out here. Rather than finding mistakes, we should concentrate on how to do this better, in a consistent and meaningful way.

## 2 What is better?

Even though “Everything is a File”, not every file is the same and not every file can be handled the same way. Having to deal with a large number of different file types poses a problem of its own, but working with a small, predefined set can help a lot. In a very broad sense, there are three main types of files:

- Software
- Configuration
- Data

So in analogy to the 3-Tier model used in software development, we are going to build a 3-Tier model for system administration.

Each of these three types of files, or tiers, differs from the others in a couple of ways and each type needs to be handled differently, at least in some situations. But before being able to define rules on how to handle the items in a specific tier, a definition first has to be created which can be used to classify the files and assign them to the correct tier.

## 2.1 The Software Tier

These days, the use of “Free and Open Source Software” (FOSS) is widespread. This software itself does not cost anything. Proprietary software solutions, often deployed together with free software on the same system, may cost a huge amount of money. So, how to specify the general value of software if it comes in prices ranging from free to very expensive? Well, the price for purchasing some software has nothing in common with the value of the software. Even if the software you use originally cost hundreds of thousands of dollars, it is only of low value. Software is a tool and as such readily replaceable.

The frequency with which software changes usually also is very low. There may be new and enhanced versions from time to time and patches which fix bugs, but neither big and critical changes nor radical changes are too frequent.

If you accidentally delete a piece of software from your server, you can simply redeploy or reinstall it, copy it over from another system or, if it was your only copy and your installation media broke as well, you ask your software vendor for another copy. This also means that there is only little to no need for daily backups of your files. If these files can simply be reinstalled in case of loss or corruption, why should daily copies be made and 7, 30 or even more identical copies kept in your backup archive? After all, would you make multiple photocopies of all your printed books on your bookshelf just in case one got lost? Hardly! If indeed one got lost or destroyed and you need it again, you would simply order another copy from your bookshop.

Software usually comes, or at least should come, readily packaged for easy, automated hands-free installation. So deploying the software to your systems in the first place can and should be a fully automated and thus repeatable process.

## 2.2 The Configuration Tier

Generally, you don’t pay any extra money for configuration files. But nevertheless, they are more valuable than the software for which you may have paid. Hidden in the configuration files, there is all the effort and time you spent to create a working and well performing system, which indirectly makes them somewhat valuable. By spending some time and effort again, the same configuration, or at least an approximation of it, could be recreated. So all considered, configuration is of medium value.

Configuration files are not static. There are various reasons why their content may change; new software versions, changing environments, new services to integrate, etc. There is a modest but constant change to configuration in general, so configuration files can be said to have a medium frequency of change.

For configuration files, it is not possible to apply the same approach used for software when restoring an accidentally deleted file. While some configuration items might be identical on all systems and thus could be copied over from another system, there are others which differ from server to server. Still, a backup may not be the best approach as a safety net for accidents and disaster. After

all, a backup is only a snapshot of the state at a certain point in time. With configuration changes, however, it often is useful being able to go back to an earlier than the immediately previous state or finding the exact difference between two earlier states. So rather than taking a regular backup, recording all the different states of a configuration ever created would be a more advisable approach. This can easily be achieved by keeping configuration files in a repository. Using this approach offers not only the possibility of restoring a corrupted or deleted configuration file, but also to go back to any previously good state, no matter how many changes have been made since.

Using a repository for all configuration files has another advantage. Either by using the repository as backend data store for a configuration management system or by directly integrating it into the deployment process, all the configuration can automatically be deployed at installation time without further manual intervention.

### 2.3 The Data Tier

There are many different types of data, depending on the business of an enterprise and depending on the services offered. But common to all data is that it is of high value. If software files are the tools used to create profit and configuration files are the instructions on how to use these tools, then data is the raw material processed and turned into profit.

For many businesses, their data is highly volatile, changing constantly. For example, with an online retail store, whenever a customer makes a purchase, a dataset is updated and changed and the more successful the online store is, the more often such updating happens. So generally, there is a high frequency of change in the data tier.

The contents of this tier usually can neither be repurchased, nor recreated. If something bad happens to your data files, you will have to restore them from backup. So it is in your utmost interest to have a good backup solution in place for your data. There may not be one backup solution for all types of data. Depending not only on the type, but also on the container holding your data, one or another backup solution may be favourable. It may even be advisable to use several different backup tools on one single system if there are several different data types and containers, which can not efficiently and safely be handled by a single tool.

Deploying new systems often does not directly involve installation of data. It is common practice to keep data externally, on separate data servers, and then attach application servers to them for processing the data. If this is not the case, or if a data server needs to be installed, then the data will either have to be migrated from an existing data source or restored from backup.

### 3 Classifying the Files

The question still remains, how to identify and classify the tens of thousands of files typically found on a system. By simply saying that all compiled and interpreted program code is software, all configuration files are configuration and any data files are data, you will not come to an easy classification method. What about all the documentation files? Do you need a documentation category? Are font files data or configuration or yet another, separate entity? Another separate category for log files? This approach would quickly lead to dozens of tiers, chaos and much overhead.

Here, too, software developers already have solved the problem for us: *duck typing*. Duck typing is very easy to apply: *“If it walks like a duck, quacks like a duck and looks like a duck, it must be a duck.”* In essence, this means that once the attributes for defining the three tiers are specified and the attribute values for each tier defined, then it is simply a matter of matching each file to these values to determine whether it is part of the software, configuration or data tier. Following the last section, our properly filled in sample table looks like the one in figure 3.

Attribute Item	Item Value	Change Frequency	Restore Method	Deployment Method
Software	low	low	redploy	automatic: kickstart, rpm, ...
Configuration	medium	medium	redploy	automatic: deploy from repo
Data	high	high	restore from backup	manually: external, restore

Figure 3: Sample 3-Tier Table

Fortunately, we do not need to go through each of our systems and label every single of the tens of thousands of files. Sticking to the three layers, done properly, methodically and consistently, classification does not even result in much extra work.

Configuration files are very easy to identify: any file which must be edited by an administrator, be it one time or multiple times, is a configuration file. The first time a file is edited manually, it must be classified as configuration and thus added to the configuration repository. Even if the file to edit is, in fact, a shell script but with embedded configuration settings which must be adapted, then the whole file is part of the configuration tier, despite the fact that a large part of the file content is actually software code.

Data files are more difficult to find, because they are very often, but not exclusively, created automatically by processes and usually are not directly handled by an administrator. Good software comes with documentation detailing where it creates data files, how to backup the data and how to restore it. Also, if there

are any log files with content that must be archived or at least kept for some time, then this content is data too and the applicable log files are part of the data tier. Additionally, as data is the most important asset on any system, it may also be wise to define what business unit actually owns the data. A single server may easily contain multiply different types of data belonging to different business units.

All the remaining files can simply be treated as part of the software tier. Even, for example, if you know exactly that a certain file contains the configuration for some software, there is no need to classify it in the configuration tier, as long as the file is automatically installed when the software or system is deployed and never needs to be touched. Since you never needed to touch it in any way, it can readily be reinstalled in its default state should it ever be accidentally removed from the system. So it is handled just like all the other software.

While classifying files with this approach saves a lot of time and effort, it is important to realize that such a classification can not be static. It may well happen that a certain configuration file has been in the software tier for a long time, but then the need arises to manually adapt it, which at that time makes it part of the configuration tier. Equally important is the requirement to do the classification separately for every single system. For example, configuration files are recommended to be kept in a repository and automatically deployed by some configuration management system. Thus, what resides inside of the repository is part of the configuration tier as soon as it is deployed to some target system, but on the server housing the repository the contents of the repository are data. Or another example: if your business is to create software, the code you produce and sell is software for your customers and of low value to them, but for yourself it actually is your data and thus very valuable to you.

## 4 Rules for Tiers

Some of the rules for handling files are implicitly defined by the attributes of the tiers, such as restore and deployment methods. Others, however, may need to be specified separately. For example, whether and how frequently to take a backup and what tools to use for backup, what process to follow to bring new files onto the system or to change existing files, etc.

Setting the rules may be the toughest part in the transition to conscious 3-Tier System Administration. It can take a lot of work up-front and requires not only some thought, but also knowledge of the procedures and tools already in place. There is not one single solution which fits every environment; this part must specifically be tailored to each entity. Figure 4 on the next page may serve as a basic starting point.

In the simplistic sample implementation shown in figure 4, both configuration and software is automatically deployed using kickstart. Kickstart itself interfaces with the same tools used for system administration after initial deployment, namely yum/rpm for package management and puppet for configuration management. Every change to configuration items during the lifecycle of a system

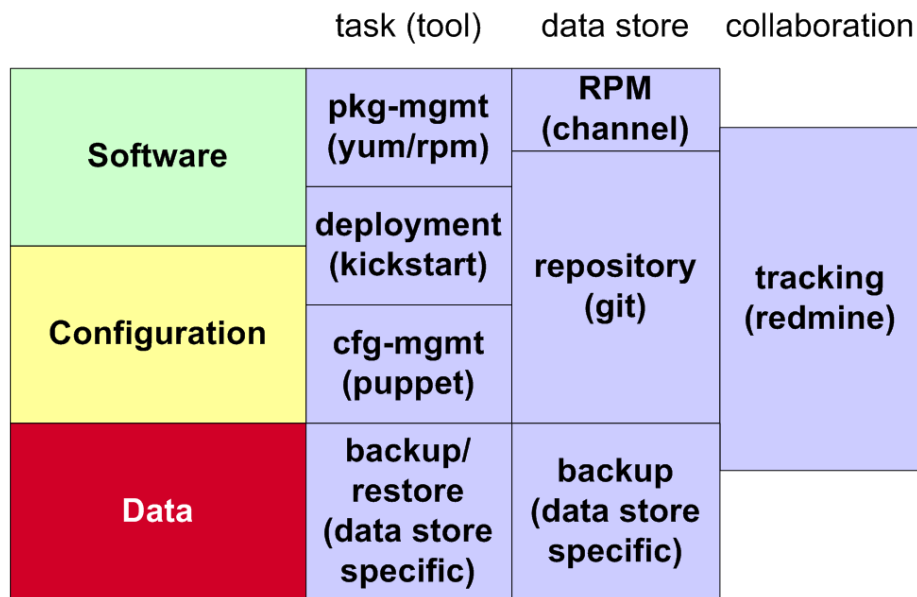


Figure 4: Sample Implementation

is performed through the configuration management layer, using puppet, while every change in installed software, such as additional software deployments or de-installations, is performed through the package management layer with yum and rpm. There are neither backups performed for the software tier, nor for the configuration tier. Any necessary restores in these two tiers are handled by redeploying the items. Backup is, however, used in the data tier. Both, initial deployment of existing data as well as restore of accidentally deleted or corrupted data is performed by doing a restore from a previous backup. There is not one single backup tool for data in use. Rather, the tool is chosen and defined separately for each type and container of backup. There may be database-specific procedures for data held in databases, normal file-level backups for any data in flat files, etc.

Not only the tools to use for the various tasks are listed, but also the related backend data stores. Software packages which come from the OS vendor or are otherwise distributed in readily usable RPM packages, are added to software channels and used directly out of these channels, which are easily accessed by rpm and yum. For other software, which may be custom built or can not be obtained pre-packaged, RPM packages are built by the administrators themselves. The required packaging source data, such as SPEC files etc. are kept in a central repository based on git. The resulting RPM packages, of course, are added to the software channels. Also all the kickstart files used for initial system setup are kept in a separate branch of the git repository. Yet another branch of the same git repository is used for all the configuration data, so every changed version of a configuration file ever used on a system can easily be retrieved at any time. With the repository, auditing requirements can be met as well, as it is possible to exactly verify who made what change to which configuration at what

time. And if useful commit messages are used when checking in changes into the repository, then this can also serve as a change log. As there are different backup tools used in the data tier, the backend data store for these backups is specific to each tool and thus to the individual data stores.

Finally, for tracking changes, problems, requests, etc. a tracking tool is being used. Redmine integrates nicely with git and other repositories and it allows direct and easy linking of commits in the repository to tickets. While for every configuration change, a redmine ticket is requested and the resulting change committed to the git repository is linked to the applicable ticket, this integration is less tight for the software and data tiers. Changes in the software tier which are performed by system administrators do have a redmine ticket too, but most of the changes in this tier, such as bug fixes, feature enhancements etc, happen on the side of the OS or software vendor and thus are not directly tracked by the system administrators. These external changes do not appear in the tracking tool. A similar situation exists for the data tier. In this sample scenario, data owners do not only own their data, but they also manage the data. Thus, initial deployment, backup, restore etc. are all handled by the data owners directly and therefore do not result in redmine tickets. Where involvement of system administrators in handling data issues is required, however, redmine tickets are created for tracking purposes.

## 5 Why should you care?

In IT, there is hardly ever a decrease of installed systems. While, due to evolving virtualization technologies, the number of physically deployed machines may eventually decline, this is not at all the case for the number of installed system images; this number is steadily growing. In order to handle a growing number of systems, the way to manage them must be as efficient as possible. Classification into a small number of distinct types, using the tools and procedures best suited to each type and having a predefined set of rules for each type helps to increase efficiency.

As the number of installed systems increases, the effort spent per system to manage them should actually decrease. So the approach for doing system administration needs to be scalable. Rules for each tier which can be applied to all installations and procedures which are applicable to all types of systems and thus can be automated, not increasing the required effort linearly, will enhance scalability.

Automation is the key factor in becoming more efficient and scalable. Only when the environment is defined, known and consistent throughout, can automation be applied successfully and with maximum benefit. Systems managed in the same defined way by every administrator, using the same set of tools for all installations and ideally using data from a central data store, offer the best consistency.