Better backups with the daemon-based Bacula backup system

BACKUP POWER

When backup jobs become too challenging for a script, the daemon-based free backup tool Bacula may be the answer.

BY JENS-CHRISTOPH BRENDEL AND MARC SCHÖCHLIN

Backup policies come in all shapes and sizes. Cheap policies use simple scripts and cater for the worst case by calling the operating system's native tools (tar, dd, cpio). This approach is fine for low-volume local backup or for environments with just a few clients.

Mid-priced backup policies use more sophisticated techniques. Tools such as rsync and Amanda are effective for many environments, but these tools often require advanced scripting skills and have some hidden limitations regarding time, volume, and hardware support.

Enterprise-level tools remove many of these restrictions but typically come at a high price. An exception to this rule is Bacula [1], a free backup utility that offers a variety of features more typically associated with high-priced commercial products.

Bacula is not monolithic but is, instead, a set of various daemons and a user interface. The daemons have fixed responsibilities and use the network to communicate. This design distributes the work load, with control centered on the admin workstation, accounts handled by a database server, and the hard work – that is, reading and writing data – handled by a team of client-side file daemons and storage daemons on the backup servers. Of course, you can also use a single machine for multiple functions, which leads to a flexible and easily scalable architecture (Figure 1).

Central Leadership

The boss in charge of the team of daemons is appropriately known as the director. The director knows what to store where and can locate the required files if a user needs to recover lost data. The director also knows the schedules, clients, storage locations, and details of planned jobs, although the actual backup is performed by subordinate daemons. Bacula's director daemon also has the distinction of being the only daemon in the Bacula system that gets to talk to a human user.

The director stores the configuration details in an ASCII file (*bacula-dir.conf*) as hierarchically structured resource descriptions. The top notch in the hierarchy is taken by the job resource, which collates the settings for a specific job. These job settings include the job type (backup, restore, verify or admin), the execution time, or the level (for a backup: full, incremental, or differential).

To make things easier, most details are grouped in sub-resources, so-called directives. Common features of similar jobs can also be grouped as JobDefs resources to form a job class, which other job descriptions can then reference.

Dennis Elema

This approach makes for a leaner configuration file and saves typing.

For example, the Schedule resource type defines schedules that run jobs at specific intervals and support almost any kind of schedule. The FileSet resource lists the directories and files you are planning to back up. Directories are handled recursively, and this means that / will give you the simplest type of full backup, although you might prefer to exclude a few directories, such as */tmp*, or hidden files such as *.journal* or *.fsck*.

The backup will only traverse filesystem boundaries when explicitly told to do so. The default setting stays in the current filesystem to avoid the danger of entering infinite loops or inadvertently saving file servers. If you want to keep this security measure, you need to enumerate every single local filesystem the client has mounted in order to create a complete backup.

Of course, Bacula also supports more complex jobs. For example, you can reference an external file list, shell expressions, or scripts that produce backup lists at runtime. As inline shell commands mean escaping non-standard characters and blanks, scripts are typically the easier option.

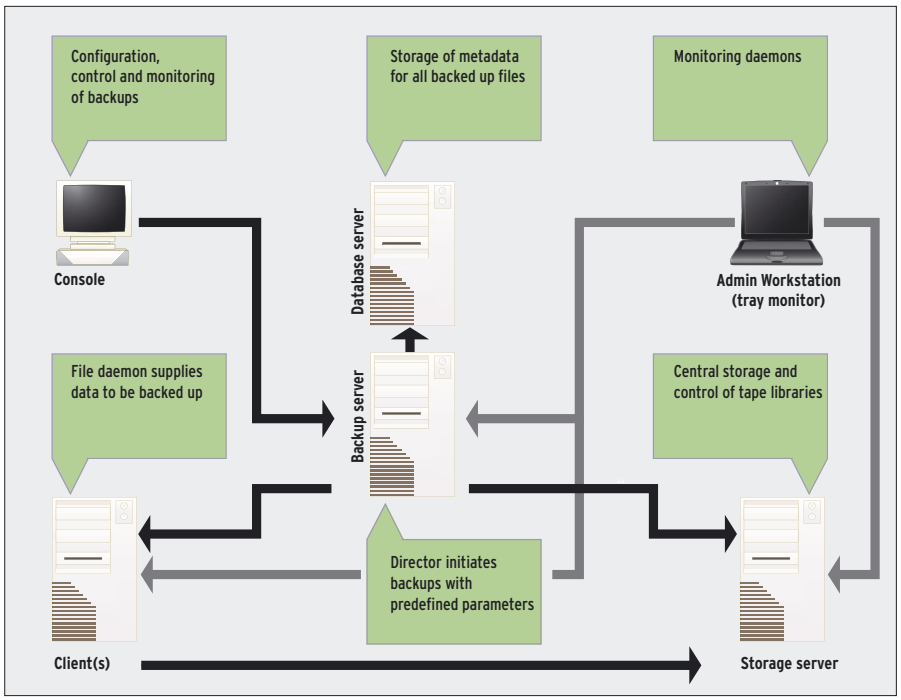

Figure 1: Divide and conquer – Bacula distributes backup functionality across the network, but storage is central.

Imagine you want to back up all the configuration files in */etc* and all the hidden files and directories in the user *jcb*'s home directory. The following mini-script would take care of this backup task:

```
#!/bin/sh
find /home/jcb ➥
-maxdepth 1 -name ".*"
find /etc -name "*.conf
```

In the preceding example, the FileSet to

Future

Bacula is definitely the Open Source backup system that comes closest to catering to professional needs in large-scale environments. The backup tool is undoubtedly suitable for production use in many cases, but there are still a few items on the wish list for future versions:

- **Security:** At present, encrypted backups are not supported by the daemon. In other words, an attacker could sniff the traffic on the local network to access the backup data. This is a genuine concern in environments with sensitive data, or wherever external providers offer backup services. As a workaround you can set up an SSH tunnel to encrypt the communication between the file and storage daemons and between the file daemon and the director. In Windows environments, at least, it would also make sense to integrate a virus checker. Solutions for this issue are in planning at present.
- **Large libraries:** Although multiple backup jobs can run simultaneously, there is still a need for more efficient parallel processing. For example, a file daemon cannot use multiplexing to provide data to multiple storage daemons, although this configuration would improve performance for higher volumes of data. Drive pools capable of statically assigning a number of drives to a specific job, and allowing the job to select any free drive in the pool, are not supported at present. There is also no support for dynamically assigning idle drives to pending jobs. This makes it difficult to put a library with multiple tape drives to optimum use.
- **GUI:** There is currently almost no graphical interface. Although some solutions have been attempted, they do not extend beyond simple text-based menus. For example, a file browser for GUI-based selection of files, or a calendar to help setting up schedules, would be useful. There is no configuration assistant to help administrators. Experienced Unix gurus might not mind this, but today's command line challenged users will tend to opt for products that give them top to tail point&click support and online help.
- **Online backup:** There are no modules for online database backups. There is also no means for backing up applications that use open files and locking these files to prevent access by others. The director partly compensates for this by allowing you to run client and server-side scripts prior to and following any job, which in turn allows you to stop and restart the applications in question. As both backup and restore jobs can use FIFOs as their data source or target, it is possible to handle data from running applications without taking a detour via a file. This is an interesting alternative, although it can't replace a full online backup.
- **Extras:** Commercial backup software gives users a number of useful extras that Bacula does not have. For instance, commercial systems often provide media cloning to mitigate the effect of irrecoverable read errors, as well as tools for managing the resumption of interrupted sessions.

match would be as shown in the following:

```
FileSet {
  name = "ConfigSet"
   include {
     Options {
       signature = MD5
     }
     File = 
   "|/etc/bacula/confbackup.sh"
    }
}
```

Besides using files, lists, or scripts, administrators can also specify raw devices as data sources (although these raw devices can only be mounted read-only). And finally, the backup can even read data from FIFOs, which link an active application with the backup. This unusual level of flexibility has its price: selecting sources is a lot less intuitive than simply letting an admin select the files in a GUI-based interface. A combination of both approaches would be ideal.

Includes Pool

Another configuration directive defines volume pools and thus sets itself apart from simple solutions. A pool groups a number of tapes logically, and thus allows a backup to extend beyond the physical capacity of a single tape. When the backup job reaches the end of the tape, Bacula continues the job on the next available tape in the same pool. This approach allows you to recycle older tapes in the group after a configurable period has elapsed.

Pool resources are controlled by a number of settings – for example, the media re-use wait period or the maximum number of lifecycles. These settings apply to all the tapes in the pool, which is a good thing; administrators do not need to set preferences for each medium in a group, although the option is available.

Assigning tapes to different pools also helps organize tapes by type of usage, thus avoiding a mix, or even an overwriting, of tapes used for incremental and full backups. You can also define pools for individual clients, weekdays, and so on.

Automatic tape changing assumes you have a tape library. Bacula supports a number of tape robots, also known as autochangers or autoloaders with DAT, VXA2, DLT, LTO, and AIT drives.

The Mtx [2] tool that Bacula uses to control tape libraries even supports barcode labels, which allow a robot to identify a tape without loading it in a drive. In some cases – for example when tapes have been manually resorted within a library – tapes need to be realigned with their previous locations. If this happens, you will definitely appreciate barcode support.

Catalogs

Whenever Bacula puts a file on a tape, it also stores details such as the file size, attributes, signature, last change date, or the time and location of the backup in a database known as the catalog. This directory is the third major unique selling point that sets Bacula apart from home-grown scripts, as it allows targeted recovery of individual files without the need for reading a complete archive. The files you wish to recover can be selected simply by referencing the meta-data, which includes the position of the required files on the tape. There is no need to read the tape sequentially from top to tail; instead, Bacula can position the tape (at the start of the job at least). Additionally, the catalog stores a history of all backup jobs.

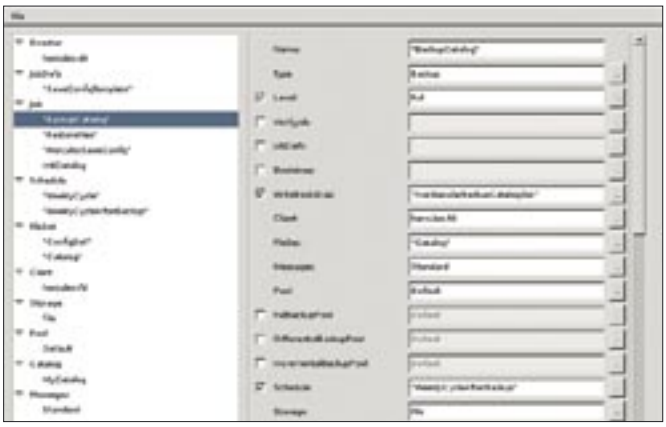

Figure 2: JBacula, an independently developed project, helps you configure the directory daemon.

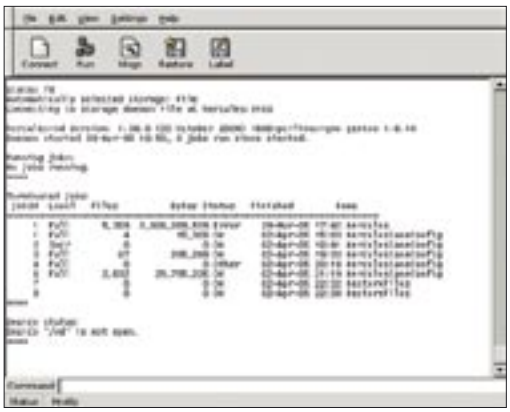

Figure 3: GConsole does not give you a GUI, but it does at least give you a graphic console with a few menus that does not need a terminal window.

Bacula can use any popular SQL database for management tasks. The package includes setup scripts for PostgreSQL, MySQL, and SQLite. Support for these popular SQL variants allows administrators to backup the database and supports manual access if worst comes to worst. A lost or inconsistent catalog is one of the most critical problems that can affect a backup set. To mitigate the effect of a lost catalog, the Bacula package includes scripts that store the catalog in an ASCII file while a job is running. If something goes wrong, at least the previous version is easily restored.

Incidentally, you can use Bacula's directory of stored files to perform a simple kind of intrusion detection a la Tripwire or Aide. Two integrated functions, which you can run independently of the backup or recovery features, are designed to collect meta-data for comparison with the filesystem. You might discover unauthorized modifications this way.

Teamwork

Of course, a director is nothing without a staff. In Bacula, the director rules over two groups of subordinates: one or multiple storage daemons and a number of file daemons. The latter run client-side and use the network to supply the data to the storage server. This is where the storage daemon runs, supporting the tape drive or library. If necessary, the storage daemon can also back up to disk, and this could be a useful short- to mid-term solution for storing the latest

backup in the light of plummeting hard disk prices.

File daemons are available for Linux, most Unix-style operating systems (for example Solaris, AIX, HPUX, FreeBSD, and even MacOS X), and all Windows versions. This more or less removes the need for detours via Samba or NFS, although both are supported.

Backward March!

Data recovery reverses the backup process. When told to do so by the director, the storage daemon sends the files you wish to restore to a file daemon, which then stores those files on the client. Files are not normally restored to their original location; instead a complete file-system tree is restored below a special directory. The restore job configuration can specify which directory this is; of course, the filesystem needs to have enough free space to accept the restored files. The default is */tmp/bacula-restores*.

You can change this behavior by specifying the root directory as the recovery target. This restores the rescued files to their original locations. A word of caution at this point: Bacula does not support conflict resolution policies. If a recovered file exists at the target location, the file is not protected or renamed but simply overwritten, and that may not be what you intended.

There are a number of approaches to selecting files to restore. All of them lead to a virtual directory tree that shows all files placed on tape. You can navigate the directory tree using Unix-style commands (*cd*, *ls*, *pwd*, and so on). And you will need to issue commands to tag files and directories for recovery (again a GUI-based selection interface would be a welcome alternative).

As a special service, Bacula lets you combine the last full backup for a client and all subsequent incremental backups in this view. You can also restrict the selection to all files backed up before or after a specific date and time.

Current Knoppix versions [3] include a Bacula file daemon and console, which makes Bacula useful as a simple disaster recovery solution, assuming you make a note of the partitioning for any disks you back up and also store Bacula's bootstrap files at a separate location. A Bacula recovery CD, intended to reanimate a system after a complete failure, will not work with later Linux kernels (2.6.x), but a remake is under discussion.

Designating Responsibilities

Access to the Bacula console is governed by a user's execute permissions; the application does not ask users to authenticate, and thus does not support different levels of privileges for its users. However, you can configure variants of the console application that only support specific jobs or command subsets, FileSets, media pools, or devices. This gives administrators a useful workaround that serves as a form of user management. The workaround is not granular enough to allow any users to restore their own files without asking the administrator, but it does support delegation of tasks within an administrative group.

In many cases, asking a user to restore data would prove too much of a challenge, as Bacula does not have a point&click interface. Tools such as Wxconsole and Gconsole provide a few menus to remove the need to memorize and type some commands, but they still have a command line option for commands you can't execute in any other way. The Java-based JBacula [4] tool, which is a separate project, provides templates and tooltips that facilitate the directory daemon configuration (Figure 2).

Conclusions

Administrators who are not afraid of the command line will find Bacula a very useful, extremely flexible backup system with many professional features. Bacula is also well-documented and integrates easily with a heterogeneous system environment. ■

Bacula – A Practical Application

At the beginning of 2004, an Internet Service Provider (ISP) based in Stuttgart, Germany was looking for a replacement for its slightly ancient backup system. Bacula was one of the major contenders, along with a number of commercial solutions.

What convinced the provider, besides the fact that Bacula would mean big savings on licensing fees, was that Bacula was independent of any manufacturer's product policy. The ISP was also looking for a solution that would allow them to reference their internal billing systems and support centralized configuration.

The ISP decided to set up a pilot installation to put the system through its paces. In the pilot phase, 32 FreeBSD production systems were backed up over a period of three months with the free Bacula backup software running parallel to existing backup solutions.

After successfully completing initial testing with a tape robot, the ISP opted for a combination with an LTO 1 drive and multiple hard disks as backup media. A 7-day backup cycle (one full backup, 6 incremental backups) and a retention period of 4 weeks was established; the administrators later fine-tuned the cycle after evaluating the catalog database.

The 32 systems in the test used a multiplex approach with 10 to 20 parallel data streams to back up their data. The *Maximum Concurrent Jobs* configuration parameter had to be tweaked to support this. Doing so had a positive effect on differential and incremental backups with respect to time required for completion and individual system load. Under production conditions, the system took an average of 19 hours to complete a full backup (of about 450GByte), 90 minutes for a differential backup, and just 40 minutes for an incremental backup.

The MySQL database originally used showed evidence of performance issues in long-term tests with increasing volumes of data, which led to MySQL being replaced by PostgreSQL later; this vastly improved the performance for restore jobs and tape recycling.

The optimized configuration was tested in daily operations over a period of several weeks. After completing this final test, the conclusion was that Bacula was easily capable of handling the requirements placed on it, and that there was nothing to prevent the ISP from installing Bacula throughout the data center.

INFO

[1] Bacula homepage: *http://www.bacula.org*

[2] Mtx for library control: *http://mtx.badtux.net*

[3] Knoppix with Bacula software: *http://www.knopper.net/knoppix/index-en.html*

[4] JBacula: *http://jbacula.sourceforge.net*