Keeping your DSpace storage under control

Submitted by bram on Tue, 2013-06-11 10:03

After introducing an institutional repository in your organization, you hope and expect the repository to grow over time. Instead of just allocating more space to volumes approaching their limits, it can be worth your while to investigate if it is really the new and valuable repository content occupying your hard drives. We provide two hints to reduce the DSpace storage footprint and to distribute storage over multiple volumes in order to keep an oversight.

Assessing directory size in Linux

If you are not managing Linux servers full time, it may not be that obvious to view disk usage per folder. One of the standard commands you can use for this purpose is the du (disk usage) instruction. This command can be used to view the disk usage for an individual file and can also add up the usage for all files and folders contained in a particular folder. Tryout the du command by entering this instruction in one line, while you are in the dspace installation directory on your server:

user@yourserver:~/dspace$ *for each in $(ls) ; do du -hs "$each" ; done*

This command should produce output similar to the following, where we just added a small description to the larger directories

1.8G	assetstore - uploaded bitstreams (files)
132K	batchedit
60K	bin
22M	config
3.0M	documentation
324K	etc
8.0K	exports
4.0K	handle-server
82M	lib
692M	log - detailed system log files for troubleshooting
4.0K	reports
6.6M	search
474M	solr - indexes for oai, statistics and discovery
4.0K	upload
8.0K	var
286M	webapps - compiled web applications. It's possible that these are stored directly in your Tomcat directory

If you want to have the list sorted by folder size without the human readable G, M, K notations for gigabyte, megabyte and kilobyte, this is an alternative:

user@yourserver:~/dspace$ *for each in $(ls) ; do du -s "$each" ; done | sort -g -r*

Reducing your DSpace storage footprint

Now that we know which directories are generally taking up a lot of space, let's look at techniques to dramatically reduce the size for some of them.

Even for a small DSpace repository that has only been running for a couple of months, the size of the logfile directory can steadily increase. This folder contains system generated reports on what happens inside of DSpace. The individual log files can contain key information to trace back the origin of a particular problem. As a rule of thumb, it is good to keep these logfiles around, especially if you are experiencing strange downtimes or other problems that need to be analyzed. The date of events in a particular logfile can easily be determined from the name of the file. This allows you to delete older log files or to archive them on another type of storage.

You can also configure how much information is registered in the logfiles. In order to understand how this works, you need to know that DSpace can generate messages with different severity. The more serious the problem, the more reason you have to keep a record of it in a log file. The names of the different log levels, in order of severity are DEBUG - INFO - WARN - ERROR - FATAL. DSpace ships with INFO as the default level. This can be changed in the log4j.properties file within your config directory. If your DSpace runs well, you can experiment by decreasing the log level to WARN or even ERROR. This will make the newly generated files substantially smaller by omitting all information contained in the INFO statements. When your DSpace is plagued by a particular bug, you can also temporary increase the logging level to DEBUG, which will include even more information in the log files compared to the INFO level. We often come across seeing bloated logfile directories caused by putting the level to DEBUG, lateron forgetting to reinstate it to INFO or WARN after solving the problem.

In the aforementioned example, the 692MB logfile directory accounted for over 20% of the dspace disk usage. This particular deployment of DSpace used the default INFO log level.

The largest directory, in the example and on a typical installation of DSpace, is the assetstore. This directory is supposed to grow because it contains the actual bitstreams (files) uploaded to DSpace. The assetstore directory keeps deleted files around until a specific cleanup script is executed on the server. That's right: even if you manually delete a file from the DSpace user interface it is still retained, as is a reference to the file in the DSpace database. In the context of a backup strategy in which the entire assetstore folder is regularly placed in backup, some institutions include the cleanup script in their cron schedule. This ensures that the script gets executed at a regular interval, for example monthly.

Storage distribution

Instead of keeping all your files on a single volume, storage can be distributed to keep a better overview and to optimally use the available volumes.

As we saw earlier, the assetstore is a single folder within the dspace directory after a default installation. Moving this directory to a different volume or using multiple assetstore folders at the same time is easy. It only requires you to tell dspace where the different assetstore folders can be found and which one to use for newly incoming submissions in dspace.cfg. Avoid moving files manually from one assetstore folder to another one: DSpace stores the assetstore number for an uploaded file in the database. After manually moving a file, DSpace won't be able to retrieve the file in your other assetstore directory.

You can decide where and for how long, you want to preserve backups. In most backup strategies, the backup data will be kept on different volumes as a means to mitigate the risk of hardware and software failures on the actual production server. It is possible that you are unintentionally keeping backups in your main DSpace volume. The ant update command, an important part of the processes to rebuild DSpace, will automatically create local backups for the lib, bin, etc and webapps folders. So if you are seeing several copies of these directories, go to the folder that contains your build.xml file and execute *ant clean_backups*, sit back and enjoy the free space you just created.

Again, looking at the aforementioned example, the lib and webapps directories together account for over 10% of the entire dspace disk usage. So in this example, running a single ant update without cleaning your backups afterwards can result in a 10% disk usage increase.

Conclusion

If your institution aims for a high number of deposits and uploaded files, running out of disk space could actually be an indicator of success. Before celebrating, use these tips and tricks to make sure that you are not keeping unnecessary files around.

Image credits

hard disk: http://www.flickr.com/photos/mattnicklas/
ship log: http://www.flickr.com/photos/rrenomeron/
buckets: http://www.flickr.com/photos/linneberg/