LIZARDFS

White paper
Version 3.12

Table of Contents

About LizardFS

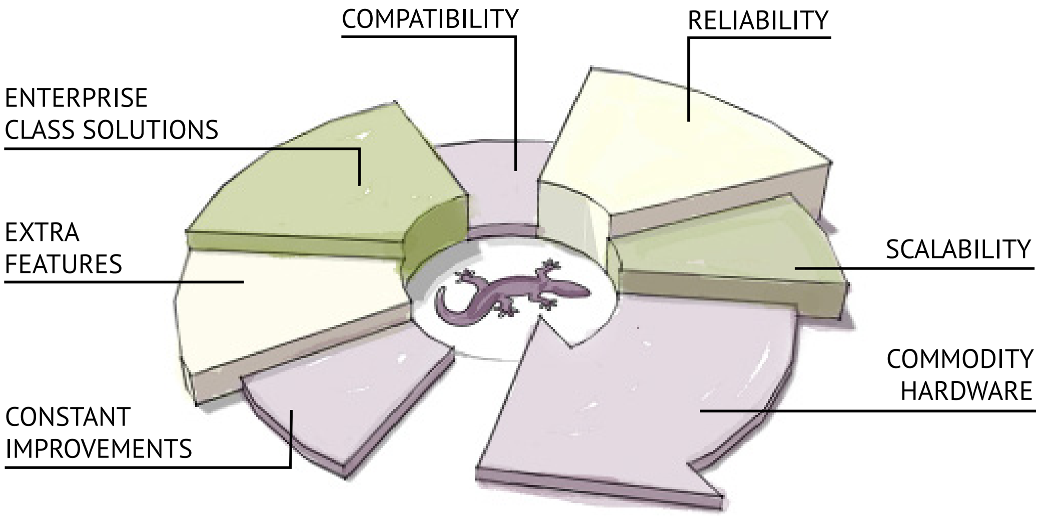
LizardFS - Software Defined Storage is a distributed, scalable, fault-tolerant and highly available file system. It allows users to combine disk space located on many servers into a single namespace which is visible on Unix-like and Windows systems in the same way as other file systems.

LizardFS makes files secure by keeping all the data in multiple replicas spread over the available servers. It can also be used to build space-efficient storage because it is designed to run on commodity hardware.

Disk and server failures are handled transparently, without any downtime or loss of data. If storage requirements grow, it is straightforward to scale an existing LizardFS installation by simply adding new servers - at any time and without any downtime. The system will automatically move data to the newly added servers, as it continuously manages the balancing of disk usage across all connected nodes. Removing a server is just as easy as adding a new one.

LizardFS offers unique features such as:

- support for many data centers and media types,
- fast snapshots,
- transparent trash bin,
- QoS mechanisms,
- quotas,
- a set of monitoring tools.



We have set ourselves a clear goal - to deliver the best Software Defined Storage solution. However, we know even when we reach our current objective, we will not stop innovating.

Architecture

LizardFS keeps metadata (e.g. file names, modification timestamps, directory trees) and the data separately. Metadata are kept on metadata servers, while data is kept on chunkservers. A typical installation consists of:

- At least two metadata servers, which work in the master-slave mode for failure recovery. Their role is to manage the whole installation, so the active metadata server is often called the master server. The role of other metadata servers is to keep in sync with the active master server, so they are often called shadow master servers. Any shadow master server is ready to take the role of the master server at any time. A suggested configuration of a metadata server is a machine with fast CPU, at least 32 GB of RAM and at least one drive (preferably SSD) to store several GB of metadata.
- A set of chunkservers which store the data. Each file is divided into blocks called chunks (each up to 64 MB) which are stored on the chunkservers. A suggested configuration of a chunkserver is a machine with large disk space available either in a JBOD or RAID configuration. CPU and RAM are not very important. You can have as little as 2 chunkservers or as many as hundreds of them.
- Clients which use the data stored on LizardFS. These machines use LizardFS mount to access files in the installation and process them just as those on their local hard drives. Files stored on LizardFS can be seen and accessed by as many clients as needed.

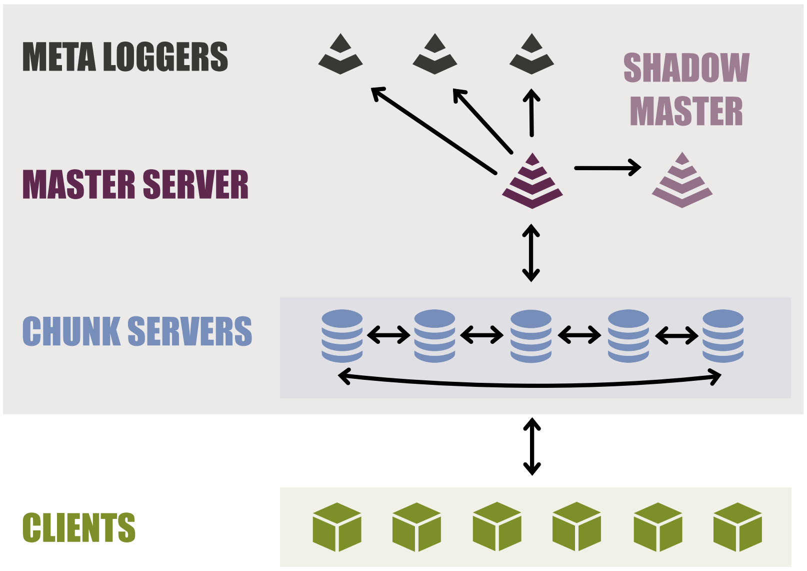


Figure 1: Architecture of LizardFS

Use Cases

Our customers use LizardFS successfully in the following scenarios:

- archive - with LTO Library/Drive support
- storage for virtual machines (as an OpenNebula backend or similar)
- storage for media / CCTV etc
- storage for backups
- as a network share drive for Windows™ servers
- DRC (Disaster Recovery Center)
- HPC (High Performance Computing)

Scalability

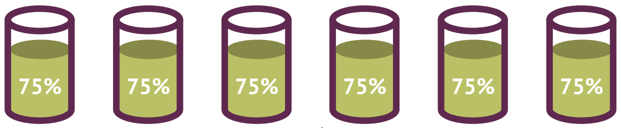
If storage requirements grow, an existing LizardFS installation can be scaled by adding new chunkservers. Adding a new server is possible without any downtime and can be performed at any time, i.e., it is transparent to clients. There are no restrictions about disk space of newly added chunkservers, e.g., adding a 24 TB chunkserver to an installation consisting of 8 TB chunkservers will work without any problems.

Performance of the system scales linearly with the number of disks, so adding a new chunkserver will not only increase available storage capacity but also the overall performance of the storage. LizardFS automatically arranges data within all chunkservers including the newly added chunkserver, as it balances disk usage across all connected nodes. Removing servers (e.g. for longer maintenance) is as easy as adding one.

Example

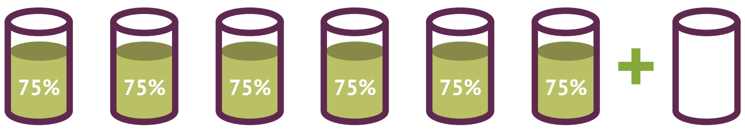
Consider the following installation:

server	total space	used space
1	20 TB	15 TB (75%)
2	16 TB	12 TB (75%)
3	16 TB	12 TB (75%)
4	16 TB	12 TB (75%)
5	8 TB	6 TB (75%)
6	8 TB	6 TB (75%)
TOTAL	**84 TB**	**63 TB (75%)**



After adding a new server the disk usage statistics will look as follows:

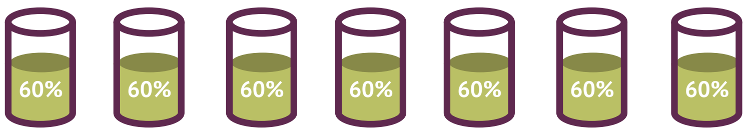
server	total space	used space
1	20 TB	15 TB (75%)
2	16 TB	12 TB (75%)
3	16 TB	12 TB (75%)
4	16 TB	12 TB (75%)
5	8 TB	6 TB (75%)
6	8 TB	6 TB (75%)
7	21 TB	0 TB (0%)
TOTAL	**105 TB**	**63 TB (60%)**



The data will be rebalanced so that the dish usage is almost even, e.g.:

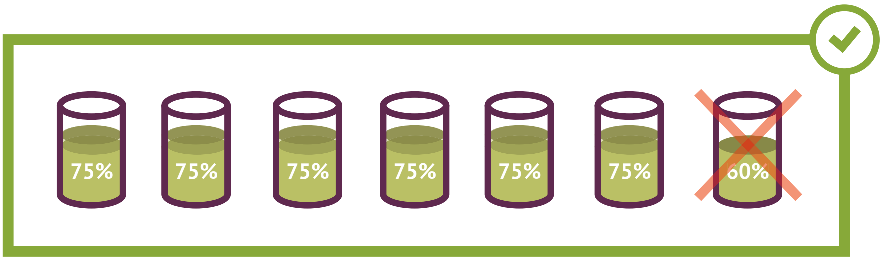
server	total space	used space
1	20 TB	12 TB (60%)
2	16 TB	9.6 TB (60%)
3	16 TB	9.6 TB (60%)
4	16 TB	9.6 TB (60%)
5	8 TB	4.8 TB (60%)
6	8 TB	4.8 TB (60%)
7	21 TB	12.6 TB (60%)
TOTAL	**105 TB**	**63 TB (60%)**

Administrators stay in full control of this process as they can decide how long this process should take.



Should you decide to take the 7th server out, you can mark it for removal. LizardFS then automatically copies all the data from the 7th server to other servers.

server	total space	used space
1	20 TB	15 TB (75%)
2	16 TB	12 TB (75%)
3	16 TB	12 TB (75%)
4	16 TB	6 TB (75%)
5	8 TB	6 TB (75%)
6	8 TB	6 TB (75%)
7	21 TB	12.6 TB (60%)
TOTAL	**105 TB**	**75.6 TB (72%)**



Hardware recommendation

LizardFS is fully hardware agnostic. Commodity hardware can be utilized for cost efficiency. The minimum requirements are two dedicated nodes with a number of disks, but to obtain a proper HA installation, you should provision at least 3 nodes. This will also enable you to use erasure coding.

We recommend that each node has at least two 1Gbps network interface controllers (NICs). Since most commodity hard disk drives have a throughput of 100MB/s, your NICs should be able to handle the traffic for the chunkservers on your host.

- Master / Shadow - at least 2 GHz CPU, 64bit
- RAM: depends on the number of files (e.g. 4GB often sufficient for a small installation)
- Disk - 128G, HDD is sufficient, SSD improves performance
- Chunkserver: recommended minimum 2GB RAM
- Metalogger: recommended minimum 2GB RAM

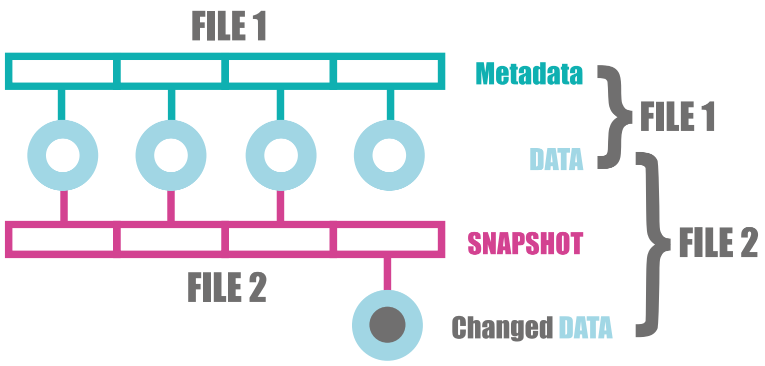
Features

It is the variety of practical and stable features offered by LizardFS that makes the system a mature enterprise solution. You can use LizardFS for Hierarchical Storage Management (HSM), in Disaster Recovery Centers with asynchronous replication, to reduce the disk space required for replication, to effectively manage storage pools (QoS, Quotas) and much more.

LizardFS is extremely flexible. If you have another use case that would require additional functionalities, these can be developed to cater to your particular needs.

Snapshots

Copying large files and directories (eg. virtual machines) can be done extremely efficiently by using the snapshot feature. When creating a snapshot, only the metadata of a target file is copied, speeding up the operation. Chunks of the original and the duplicated file are now shared until one of them is modified.



Example

Consider a directory /mnt/lizardfs/huge_vm, which contains several large files, totalling 512GB. Making a backup copy with `cp -r huge_vm huge_vm_backup` command would take a long time and consume additional 512GB of disk space.

We recommend instead to create a copy using the `mfsmakesnapshot huge_vm huge_vm_backup` command. This command, in contrast, will be executed instantaneously and consume a minimum amount of disk space.

Now, consider the case of modifying one of the files in the /mnt/lizardfs/huge_vm directory in a way that affects only 2 of its chunks. When using the `mfsmakesnapshot` option, only the modified chunks would be copied, an operation that uses only 128MB, which is negligible when compared to the 512GB consumed by the `cp -r` alternative.

QoS

LizardFS offers mechanisms that allow administrators to set read/write bandwidth limits for all the traffic generated by a given mount point, as well as for a specific group of processes spread over multiple client machines and mountpoints. This makes LizardFS suitable for critical applications, where the proper quality of service guarantees are required.

Example

Consider an installation with 300 hard drives and provisioned according to our recommendations. One can expect it to serve about 20 GB/s to all clients. Imagine a production system consisting of many virtual machines using LizardFS as their storage. If a group of administrators has access to the host machines and use it for maintenance work, it is possible to use bandwidth limits to ensure the production system runs smoothly.

After configuring the host machines to place all ssh processes (and all processes spawned by them) in a cgroup named, for example, "interactive" and defining a bandwidth limit of 100 MB/s for the "interactive" group, administrators will not be able to accidentaly disrupt the production system. Any processes that were started manually from an ssh session will be limited to 100 MB/s in total, even though they may be spread over tens of machines.

In the same installation, it is also possible to add a bandwidth limit for a set of virtual machines used by a subsystem (or a customer) with limited performance requirements, to make sure that these machines will not disrupt the operation of other virtual machines. After setting the bandwidth limit to 1 GB/s administrators can be sure that all the traffic generated by these machines (both reads and writes) will sum to a total of 1 GB/s or less.

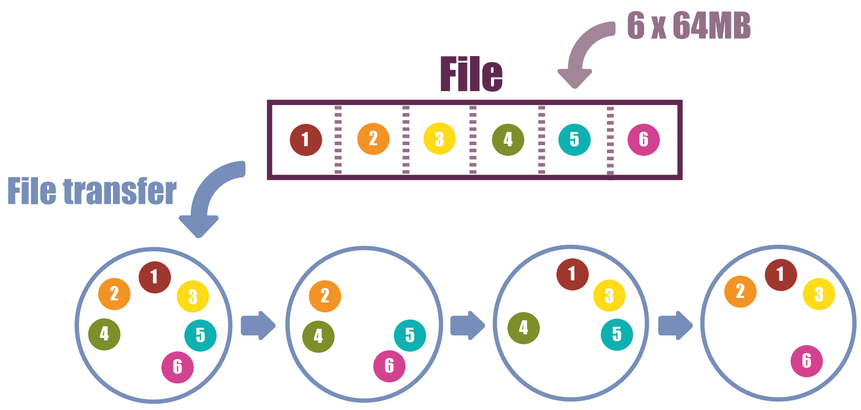
Data replication

Files stored in LizardFS are divided into blocks called chunks, each up to 64 MB in size. Each chunk is kept on chunkservers and administrators can choose how many copies of each file are maintained. For example, choosing to keep 3 copies (configuration goal=3), all the of the data will survive a failure of any two disks or chunkservers, because LizardFS will never keep 2 copies of the same chunk on the same node.

Security levels (the number of copies) can be set independently for each file and directory tree. Whenever a chunk is lost, e.g. due to a disk failure, the system will replicate it using the remaining copies of the chunk to maintain the requested number of copies.

Data is spread evenly across all the nodes. There are no dedicated mirror servers, which means that in the case when goal=3 each chunk has its three copies on three randomly chosen servers.

This unique design makes failure recovery very fast. If one disk is lost, backup copies of its data are evenly spread over all other disks, so all disks in the installation can take part in the process of replication. Consequently, the speed of recovering lost data increases with the number of disks in the system.



Replication mechanism

LizardFS implements two replication mechanisms: standard and XOR copies.

Standard replica

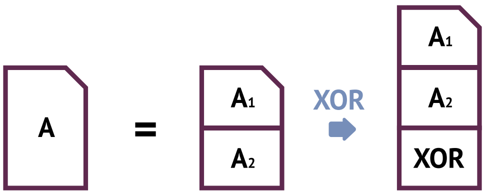
This kind of replication allows for setting the number of copies every chunk should have. Note that it is not possible to store 2 of the same chunks on 1 chunkserver, meaning standard goals should comply with the number of chunkservers and their labels.

Example

Standard goal "3" indicates that each chunk should be placed on 3 different chunkservers. The specific location of the chunks will be determined by the master server, which balances space usage evenly.

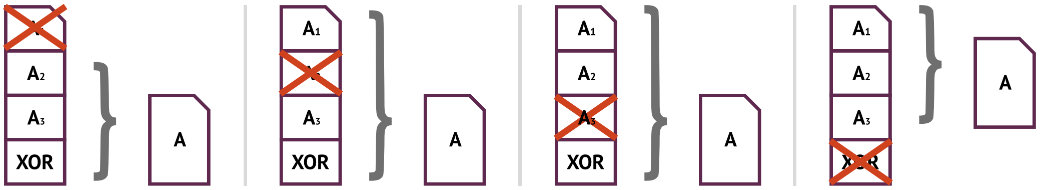
XOR replica

This kind of replication divides chunks into smaller parts according to the "xor level" parameter. A file with XOR level N is kept as N+1 chunks (N for data, 1 for parity check). As a result, the disk space required to store the file is (N+1)/N, while retaining data integrity despite one of the chunkservers failing. The XOR level can range from 2 to 10.



Example

Goal "xor3" means that each chunk of a file will be divided into 4 parts and distributed among the chunkservers. Now, if any single chunkservers fails, it is possible to reconstruct the file from the remaining 3 XOR chunks.

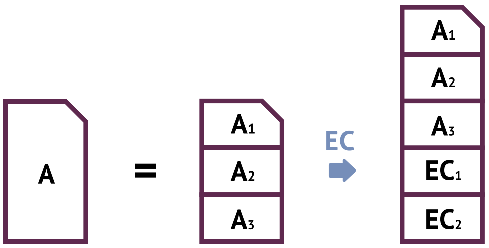


EC replica - Erasure Coding replica (since version 3.10.0)

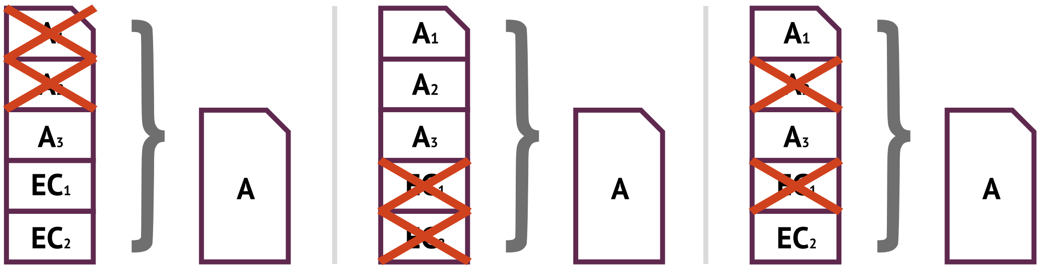
For each file using an ec(K+M) goal, the system will split the file into K parts and generate M parity parts. The K parts are required for reproducing the file content. If labels are specified, labeled parts will be kept on chunkservers. Otherwise, default wildcard labels are used.

This kind of goal allows M of the K+M copies to be lost and the file still remains accessible. Using the erasure code goal occupies M/K additional space, e.g. goal 5+1 requires 20% additional space, while 3+2 requires 66.6% additional space.

Please note that the minimal K value is 2 and the minimal M value is 1. Erasure code goals with M value equal to 1 are equivalent to XOR replication of type K.



Example EC(3,2)



Georeplication (aka custom goals)

With Georeplication you can decide where the chunks are stored. The topology feature allows for suggesting which copy should be read by a client in the case when more than one copy is available. The custom goals feature makes LizardFS more useful when working with many locations and media types (e.g., SSD and HDD drives).

Administrators are able to use the feature by labeling chunkservers to decide how the system spreads chunks across available servers. This enables LizardFS to withstand the failure of a whole data center (Disaster Recovery), as shown in the example below.

Example

When LizardFS is deployed across two data centers, e.g. one located in London and one in Paris, it is possible to assign the label "london" to each server in the London location and "paris" to each server in the Paris location. Additionally, one can create a custom goal named 2_locations, defined as "2_locations: paris london". Each file created in a directory with the 2_locations goal will have its chunks stored in two copies, one on a "london" labeled server, and one on a "paris" labeled server. This ensures that even when one entire location is unavailable, the files at the other location are all still available for reading and writing.

One can use the topology mechanism to make mounts prefer reading Custom goals (as described in this example) rather than simple goals, which specify only a number of copies (e.g., goal=3). It is possible to set and change the goals at any time for any file or any directory tree, and the system will move data accordingly when required.

There are more possibilities, e.g.:

- To define a goal "safe_2_locations: paris london _" which is interpreted as: "3 copies, at least one in London and one in Paris" – now we can lose any two servers or a whole data center without any downtime. The system will place the third copy in a way that evens out disk usage on all servers to a point defined by the system administrator.

- To define a goal "fast: ssd ssd" – if some servers are equipped only with ssd drives and others with hdd drives (and properly labelled), this ensures that files labelled in this way are stored in two copies only on chunkservers labelled ssd, which in most cases will be much faster than the non-SSD ones.

- If there is only one location with a configuration as in the previous example (SSD and HDD servers) one can use a custom goal such as "one_fast_copy: ssd hdd" for some files to force access from the specified medium.

Metadata replication

Metadata in LizardFS consists of:

- names of files and directories
- POSIX attributes of files and directories, (e.g. owner, owning group, access mask, last modification and access timestamps, size of files, etc.)
- extended attributes
- access control lists
- information about each file's goal and trash time
- information about which chunks form each file

Metadata is stored on metadata servers. At any time, one of the metadata servers also manages the whole installation and is called the master server. Other metadata servers remain in sync with it and are shadow master servers.

Each metadata server stores all the metadata in its RAM as well as on a local hard drive. All the changes in the metadata made by the master server are instantly applied by all connected shadow master servers and also saved on their hard drives. At any time, it is possible to add a new shadow master server or to remove an existing one. These operations are completely transparent to the clients.

Shadow master servers provide LizardFS with High Availability. If there is at least one shadow master server running and the active master server is lost, one of the shadow master servers takes over. All other shadow master servers, chunkservers, clients and metaloggers will continue to work with the new master server. This process can also be triggered manually when there is a need for maintenance work on the master server.

Moreover, you can increase the security of your metadata by adding metaloggers. Each metalogger is a lightweight daemon (low or no RAM/CPU requirements) which connects to the active metadata server, continuously receives the same information that all shadow masters receive and continuously saves it to its local hard drive. In this way, an additional copy of metadata is created and if all the metadata servers are lost, the whole system can be easily recovered from such a cold copy.

As metaloggers use almost no CPU nor RAM, the cost of adding many of them is very low. In this way, one can have any number of independent and constantly updated copies of all the metadata.

Example

For a file system which consists of 50 million files which sum up to 1 PB of data, the metadata will occupy about 13 GB of hard drive space. A single metalogger which keeps a couple of copies of the metadata (the most recent version as well as a couple of older backups) will require less than 100 GB of hard drive space.

After setting up metaloggers on 10 machines which work as chunkservers there will be 12 constantly updated copies of the system's metadata (including one copy kept on the active master server and one on a shadow master server) making it virtually indestructible.

LTO library support

LizardFS offers native support for LTO libraries. Storing archival backups may consume a lot of memory, even though such files are almost never read, which means that this kind of data can be efficiently stored on a tape.

LizardFS offers a comfortable way of working with backend LTO storage. Files can be chosen to have a backup copy on a tape by setting a tape goal. Setting a tape goal to a file makes it read-only for obvious reasons - tape storage does not support random writes. Reading from tape storage is a timely process, so data stored in there should be read very rarely.

If a regular copy of a file is still available, it will be used for reading. If a file exists only on tape, it has to be restored to LizardFS first. To achieve that, one must use lizardfs-restore-tape-copy utility: $ lizardfs-restore-tape-copy file_path. After running this command, all needed data will be read from tape storage and loaded to the file system, making the file accessible to clients.

Quotas

LizardFS support disk quota mechanism known from other POSIX le systems. It offers an option to set soft and hard limits for a number of files and their total size for a specific user or a group of users. A user whose hard limit is exceeded cannot write new data to LizardFS.

From version 3.10.0 LizardFS supports also quotas per directory.

POSIX compliance

LizardFS offers all the features of POSIX file systems, for example:

- a hierarchical structure of files and directories
- attributes of files like uid, gid, access mask, access time, modification time, change time
- support for symlinks, hard links, and other special files (unix sockets, devices)
- access control lists
- extended attributes

The semantics of access rights in LizardFS are precisely the same as in other POSIX file systems (like ext or xfs). This makes it compatible with all applications that can use a local hard drive as their storage.

Trash

Another feature of LizardFS is a transparent and fully automatic trash bin. After removing any file, it is moved to a trash bin, which is visible only to the administrator. Any file in the trash bin can be restored or deleted permanently. Thus, data stored on LizardFS is more secure than data stored on a hardware RAID.

The trash is automatically cleaned up: each file is removed permanently from the trash by the system after some period of time (trash time). At any time, the trash time can be adjusted for any file or directory tree.

Example

If trash time for the whole installation is set to two hours, every file will be available within two hours after deletion. This should be sufficient for any accidents caused by users. Additionally, trash time can be adjusted for any file or directory tree, e.g.:

- set to 0 for a directory with temporary files, so they would be removed immediately and free up space
- set to 0 for files before removing when we need to free space and we know that this file is no longer needed
- set to 1 month for very important data

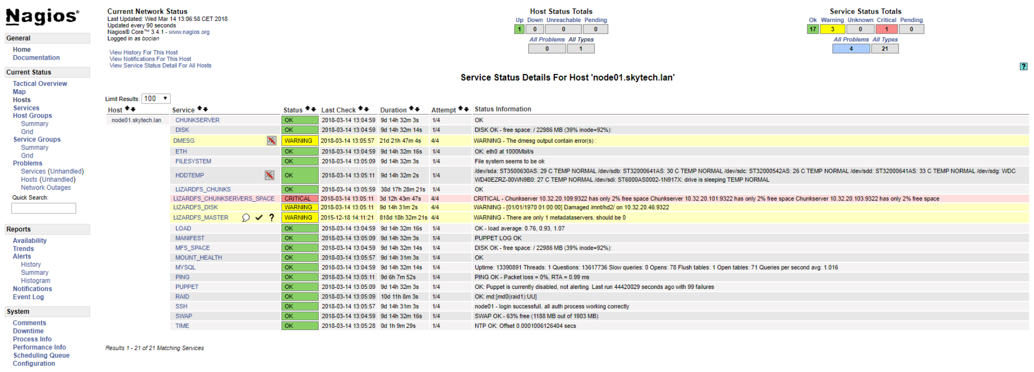
Native Windows™ client

LizardFS offers a native Windows Client which allows users to build a software solution free from a single point of failures like SMB servers which are typically used to export Unix file systems to Windows applications.

LizardFS Windows Client can be installed on both workstations and servers. It provides access to files stored on LizardFS via a virtual drive, for example L:, which makes it compatible with almost all existing WindowsTM applications.

Monitoring

LizardFS offers two monitoring interfaces. First of all, there is a command line tool useful for systems like Nagios, Zabbix, Icinga, which are typically used for proactive monitoring. It offers a comprehensive set of information about connected servers and disks, client machines, storage space, errors, chunk copies lost due to disk failures, and much more.



Current Network Status
Last Updated: Wed Mar 14 13:06:58 CET 2018
Updated every 90 seconds
Nagios® Core™ 3.4.1 - www.nagios.org
Logged in as *bocian*

Host Status Totals

Up	Down	Unreachable	Pending
1	0	0	0

All Problems	All Types
0	1

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
17	3	0	1	0

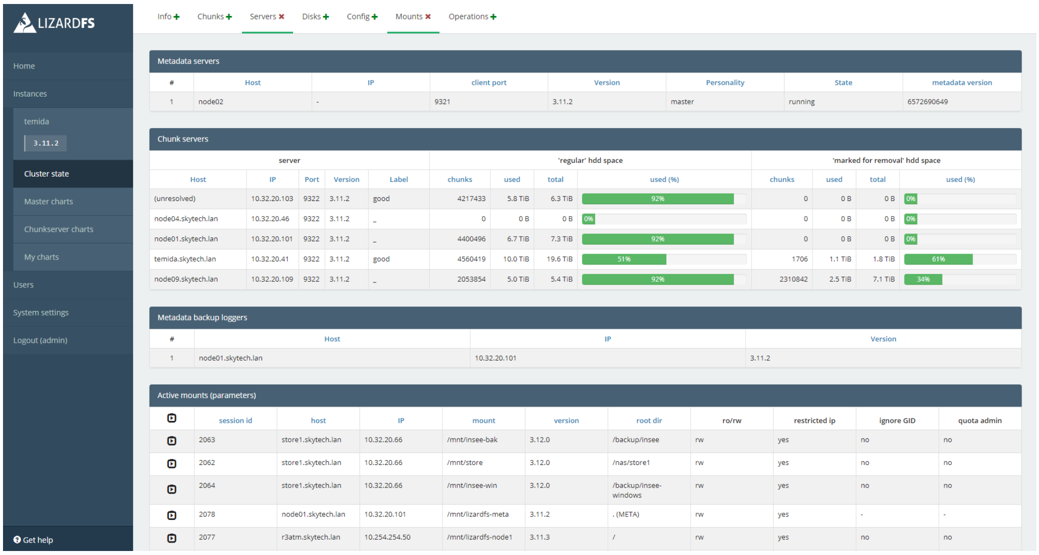
All Problems	All Types
4	21

Service Status Details For Host 'node01.skytech.lan'

Host	Service	Status	Last Check	Duration	Attempt	Status Information
node01.skytech.lan	CHUNKSERVER	OK	2018-03-14 13:04:59	9d 14h 32m 3s	1/4	OK
	DISK	OK	2018-03-14 13:04:59	9d 14h 32m 14s	1/4	DISK OK - free space: / 22986 MB (39% inode=92%):
	DMESG	WARNING	2018-03-14 13:05:57	21d 21h 47m 4s	4/4	WARNING - The dmesg output contain error(s) :
	ETH	OK	2018-03-14 13:04:59	9d 14h 32m 16s	1/4	OK: eth0 at 1000Mbit/s
	FILESYSTEM	OK	2018-03-14 13:05:09	9d 14h 32m 3s	1/4	File system seems to be ok
	HDDTEMP	OK	2018-03-14 13:05:11	9d 14h 32m 2s	1/4	/dev/sda: ST3500630AS: 29 C TEMP NORMAL /dev/sdb: ST32000641AS: 30 C TEMP NORMAL /dev/sdc: ST32000542AS: 26 C TEMP NORMAL /dev/sdd: ST32000641AS: 33 C TEMP NORMAL /dev/sdg: WDC WD40EZRZ-00WN9B0: 27 C TEMP NORMAL /dev/sdi: ST6000AS0002-1N917X: drive is sleeping TEMP NORMAL
	LIZARDFS_CHUNKS	OK	2018-03-14 13:05:59	38d 17h 28m 21s	1/4	OK
	LIZARDFS_CHUNKSERVERS_SPACE	CRITICAL	2018-03-14 13:05:11	3d 12h 43m 47s	4/4	CRITICAL - Chunkserver 10.32.20.109:9322 has only 2% free space Chunkserver 10.32.20.101:9322 has only 2% free space Chunkserver 10.32.20.103:9322 has only 2% free space
	LIZARDFS_DISK	WARNING	2018-03-14 13:05:11	9d 14h 31m 2s	4/4	WARNING - [01/01/1970 01:00:00] Damaged /mnt/hd2/ on 10.32.20.46:9322
	LIZARDFS_MASTER	WARNING	2015-12-18 14:11:21	818d 18h 32m 21s	4/4	WARNING - There are only 1 metadataservers, should be 0
	LOAD	OK	2018-03-14 13:04:59	9d 14h 32m 16s	1/4	OK - load average: 0.76, 0.93, 1.07
	MANIFEST	OK	2018-03-14 13:05:09	9d 14h 32m 3s	1/4	PUPPET LOG OK
	MFS_SPACE	OK	2018-03-14 13:04:59	9d 14h 32m 14s	1/4	DISK OK - free space: / 22986 MB (39% inode=92%):
	MOUNT_HEALTH	OK	2018-03-14 13:05:57	9d 14h 31m 3s	1/4	OK
	MYSQL	OK	2018-03-14 13:04:59	9d 14h 32m 14s	1/4	Uptime: 13390891 Threads: 1 Questions: 13617736 Slow queries: 0 Opens: 78 Flush tables: 1 Open tables: 71 Queries per second avg: 1.016
	PING	OK	2018-03-14 13:05:11	9d 6h 7m 52s	1/4	PING OK - Packet loss = 0%, RTA = 0.99 ms
	PUPPET	OK	2018-03-14 13:05:09	9d 14h 32m 3s	1/4	OK: Puppet is currently disabled, not alerting. Last run 44420029 seconds ago with 99 failures
	RAID	OK	2018-03-14 13:05:09	10d 11h 8m 3s	1/4	OK: md:[md0(raid1):UU]
	SSH	OK	2018-03-14 13:05:57	9d 14h 31m 3s	1/4	node01 - login successfull, all auth process working correctly
	SWAP	OK	2018-03-14 13:04:59	9d 14h 32m 16s	1/4	SWAP OK - 63% free (1188 MB out of 1903 MB)
	TIME	OK	2018-03-14 13:05:28	0d 1h 9m 29s	1/4	NTP OK: Offset 0.0001006126404 secs

Results 1 - 21 of 21 Matching Services

Moreover, there is a graphical web-based monitoring interface available for administrators, which allows tracking almost all aspects of a system.



Metadata servers

#	Host	IP	client port	Version	Personality	State	metadata version
1	node02	-	9321	3.11.2	master	running	6572690649

Chunk servers

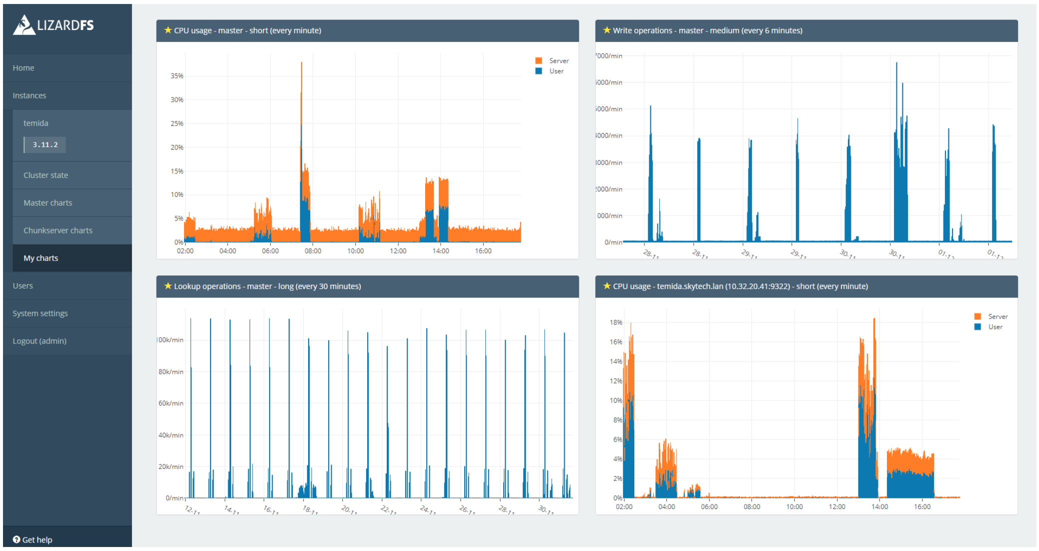
server					'regular' hdd space				'marked for removal' hdd space			
Host	IP	Port	Version	Label	chunks	used	total	used (%)	chunks	used	total	used (%)
(unresolved)	10.32.20.103	9322	3.11.2	good	4217433	5.8 TiB	6.3 TiB	92%	0	0 B	0 B	0%
node04.skytech.lan	10.32.20.46	9322	3.11.2	_	0	0 B	0 B	0%	0	0 B	0 B	0%
node01.skytech.lan	10.32.20.101	9322	3.11.2	_	4400496	6.7 TiB	7.3 TiB	92%	0	0 B	0 B	0%
temida.skytech.lan	10.32.20.41	9322	3.11.2	good	4560419	10.0 TiB	19.6 TiB	51%	1706	1.1 TiB	1.8 TiB	61%
node09.skytech.lan	10.32.20.109	9322	3.11.2	_	2053854	5.0 TiB	5.4 TiB	92%	2310842	2.5 TiB	7.1 TiB	34%

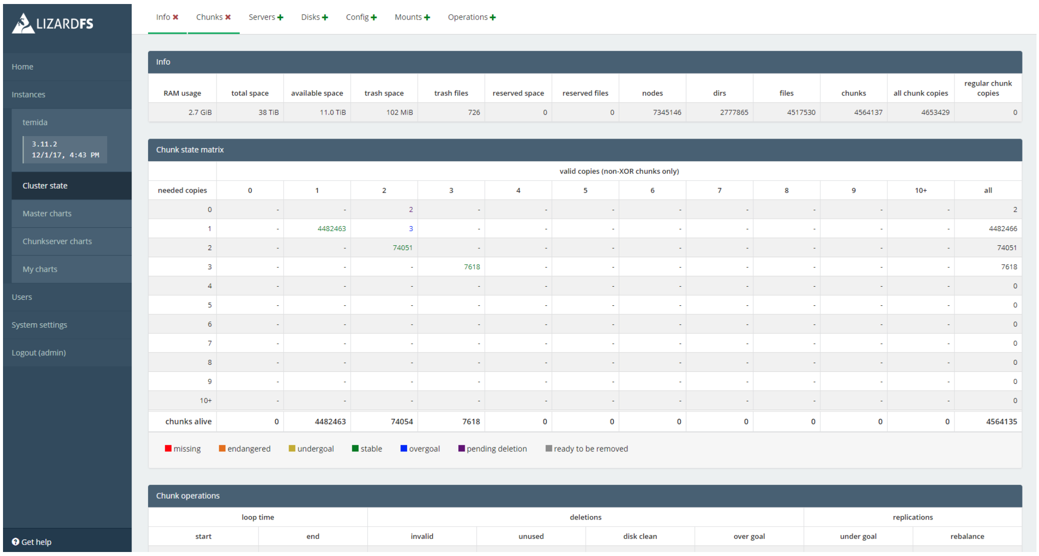
Metadata backup loggers

#	Host	IP	Version
1	node01.skytech.lan	10.32.20.101	3.11.2

Active mounts (parameters)

	session id	host	IP	mount	version	root dir	ro/rw	restricted ip	ignore GID	quota admin
	2063	store1.skytech.lan	10.32.20.66	/mnt/insee-bak	3.12.0	/backup/insee	rw	yes	no	no
	2062	store1.skytech.lan	10.32.20.66	/mnt/store	3.12.0	/nas/store1	rw	yes	no	no
	2064	store1.skytech.lan	10.32.20.66	/mnt/insee-win	3.12.0	/backup/insee-windows	rw	yes	no	no
	2078	node01.skytech.lan	10.32.20.101	/mnt/lizardfs-meta	3.11.2	. (META)	rw	yes	-	-
	2077	r3atm.skytech.lan	10.254.254.50	/mnt/lizardfs-node1	3.11.3	/	rw	yes	no	no


LIZARDFS
Home
Instances
temida
3.11.2
Cluster state
Master charts
Chunkserver charts
My charts
Users
System settings
Logout (admin)
Get help
★ CPU usage - master - short (every minute)
★ Write operations - master - medium (every 6 minutes)
★ Lookup operations - master - long (every 30 minutes)
★ CPU usage - temida.skytech.lan (10.32.20.41:9322) - short (every minute)
Server
User



Info

RAM usage	total space	available space	trash space	trash files	reserved space	reserved files	nodes	dirs	files	chunks	all chunk copies	regular chunk copies
2.7 GiB	38 TiB	11.0 TiB	102 MiB	726	0	0	7345146	2777865	4517530	4564137	4653429	0

Chunk state matrix

	valid copies (non-XOR chunks only)											
needed copies	0	1	2	3	4	5	6	7	8	9	10+	all
0	-	-	2	-	-	-	-	-	-	-	-	2
1	-	4482463	3	-	-	-	-	-	-	-	-	4482466
2	-	-	74051	-	-	-	-	-	-	-	-	74051
3	-	-	-	7618	-	-	-	-	-	-	-	7618
4	-	-	-	-	-	-	-	-	-	-	-	0
5	-	-	-	-	-	-	-	-	-	-	-	0
6	-	-	-	-	-	-	-	-	-	-	-	0
7	-	-	-	-	-	-	-	-	-	-	-	0
8	-	-	-	-	-	-	-	-	-	-	-	0
9	-	-	-	-	-	-	-	-	-	-	-	0
10+	-	-	-	-	-	-	-	-	-	-	-	0
chunks alive	0	4482463	74054	7618	0	0	0	0	0	0	0	4564135

🟥 missing 🟧 endangered 🟨 undergoal 🟩 stable 🟦 overgoal 🟪 pending deletion ⬜ ready to be removed

Chunk operations

loop time		deletions				replications	
start	end	invalid	unused	disk clean	over goal	under goal	rebalance

Hadoop

This is a java based solution allowing Hadoop to use LizardFS storage, implementing an HDFS interface to LizardFS. It functions as kind of a File System Abstraction Layer. It enables you to use Hadoop jobs to directly access the data on a LizardFS cluster. The plugin translates LizardFS protocol and makes the metadata readable for Yarn and Map Reduce. For performance, Hadoop nodes should run on the same machines as LizardFS chunk servers.

LizardFS mount gives direct access to stored files, from the OS level. This allows you to use it as a shared storage in your company and a computation storage for HADOOP at the same time. It is not required to use HADOOP tools to put/get files from your storage in comparison to HDFS. We can also take advantage of Erasure Coding and save a lot of disk space (HDFS recommends to store 3 copies).

The function: public BlockLocation[] getFileBlockLocations(FileStatus file, long start, long len) returns information where data blocks are held in your LizardFS installation. If Hadoop is run on the same machines, it can take advantage of data locality.

To install Hadoop with LizardFS:

1. Install and setup LizardFS cluster.

2. Install HADOOP – but don't start.

3. Install LizardFS-HADOOP plugin on all HADOOP nodes.

4. Configure LizardFS-Plugin in HADOOP (alongside HDFS or replace it).

5. Start HADOOP.

RichACL

ACLs (Access Control Lists) define type and level of access that users have. LizardFS supports ACLs on Linux, MacOSX, Windows and through NFS (version 4 and 4.1).

Because there are a lot of ACLs that are the same, LizardFS deduplicates them in memory, so the impact on performance is negligible.

NFS and pNFS

How is NFS Client mounted to master server?

LizardFS uses NFS-ganesha server to create NFS shares, so technically NFS client connects not with master server, but with a ganesha file server that talks directly with LizardFS components. From the user point of view, it works just like an ordinary NFS server.

The pNFS communication process overview

The pNFS cluster consists of MDS (Meta-Data-Server) and DS (Data-Server(s)). The client sends all the read/write requests directly to DS and all other operations are handled by the MDS. Both MDS and DS are handled by nfs-ganesha, MDS typically just needs an entry ‚PNFS_MDS = true;' in its configuration file.

When installing LizardFS with nfs-ganesha the LizardFS FSAL (File System Abstraction Layer) is installed as well. In short, FSAL sees LizardFS stored data and presents it to nfs-ganesha. With these components, one can mirror the native LizardFS set-up and install MDS on the same machine(s) where the LizardFS Master Server is located and DS on each chunkserver. To access such a cluster with an NFS client one just needs MDS address:port.

pNFS client contacts MDS, MDS talks via FSAL to LizardFS Master Server to get file metadata which is passed to MDS and further to the client together with a list of DS the NFS server can access. Then pNFS client connects (multiple) DS. Each DS contacts LizardFS chunkserver(s) via FSAL and retrieves/stores the data. In a case of a DS node malfunction pNFS switches to another DS. FSAL operations are the place where the pNFS protocol is translated to LizardFS and back.

In the case when an older, non-parallel (pre 4.1) NFS, MDS is used as a NFS gateway - nfs-ganesha on MDS contacts LizardFS via FSAL, receives data from multiple chunkservers and sends it back to the NFS client.

Portability

LizardFS packages are available for most operating systems, including:

- Ubuntu LTS
- Debian
- Red Hat Enterprise Linux
- CentOS

For these systems, installation is performed by the package manager which is a part of these Linux distributions. The software is highly portable and there is a possibility to use a tarball to build LizardFS for almost any operating system from the Unix family.

Additionally, a client driver is available for Windows machines and can be installed using a simple installation wizard.

Examples of configuration

Example 1

- 2 metadata servers, each equipped with 64 GB RAM and RAID-1 built on top of two 128 GB SSD drives
- 20 chunkservers, each equipped with RAID-6 controller with 8 SAS drives, 3 TB each
- 10 Gbit network
- replication level 2
- a metalogger running on 3 out of 20 chunkservers

Such a configuration offers a possibility to store 180 TB of data in goal=2. It offers high performance as well as a high durability.

There will be **no downtime** in case of:

- losing any 5 disks
- losing any single server (including the master server)

No loss of data in case of:

- losing any 5 disks
- losing any single chunkserver
- losing both metadata servers

Example 2

- 2 metadata servers, each equipped with 64 GB of RAM and a RAID-1 storage build on top of two 128 GB SSD drives
- 30 chunkservers, each equipped with 12 hard drives (in JBOD configuration)
- 3 TB each
- 1 Gbit network
- replication level 3
- a metalogger running on 3 out of 30 chunkservers

This configuration offers a possibility to store 360 TB of data in goal=3.

No downtime in case of:

- failure of any 2 chunkservers or any 2 disks
- failure of any single metadata server

No loss of data in case of:

- failure of any 2 chunkservers or any 2 disks
- failure of both metadata servers

Example 3

- 2 data centers, primary and secondary
- 3 metadata servers, two in the primary data center one in the backup data center,
- each equipped with 64 GB of RAM and a RAID-1 storage built on top of two 128 GB
- SSD drives
- 15 chunkservers in each of 2 data centers, each equipped with a RAID-6 controller with 16 SAS drives, 3 TB each
- 10 Gbit link between data centers
- replication level which enforces one copy to be kept in each data center
- 2 metaloggers (on selected chunkserver machines) in each data center

This configuration offers a possibility to store 630 TB of data.

No downtime and no loss of data in case of:

- failure of any single server (including the master server)
- failure of any 5 disks
- failure of the connection between data centers
- failure of all servers in the secondary data center (e.g. power failure, fire or flooding)
- failure of all chunkservers in the primary data center

No data will be lost in case of failure all servers in the primary location, but there would be a need to start the master server manually (no quorum in the secondary data center prevents this automatically), thus some downtime may occur.

Contact

LIZARDFS

LIZARDFS, INC.	contact@lizardfs.com	visit our website:
1691 Kettering Street	+1 (646) 655 0693	www.lizardfs.com
Irvine, CA, 92614	+48 22 100 32 48	

www.lizardfs.com