



White paper
Version 3.10

Table of Contents

About LizardFS	2
Architecture	3
Use Cases of LizardFS	4
Scalability	4
Hardware recommendation	6
Features	7
Snapshots	7
QoS	8
Data replication	8
Replication	9
Standard replica	9
XOR replica	9
Erasure Coding	10
Georeplication	11
Metadata replication	12
LTO library support	13
Quotas	13
POSIX compliance	13
Automated Trash bin	14
Native Windows™ client	14
ActiveMonitoring	15
Portability	16
Use cases	16
Contact	18

About LizardFS

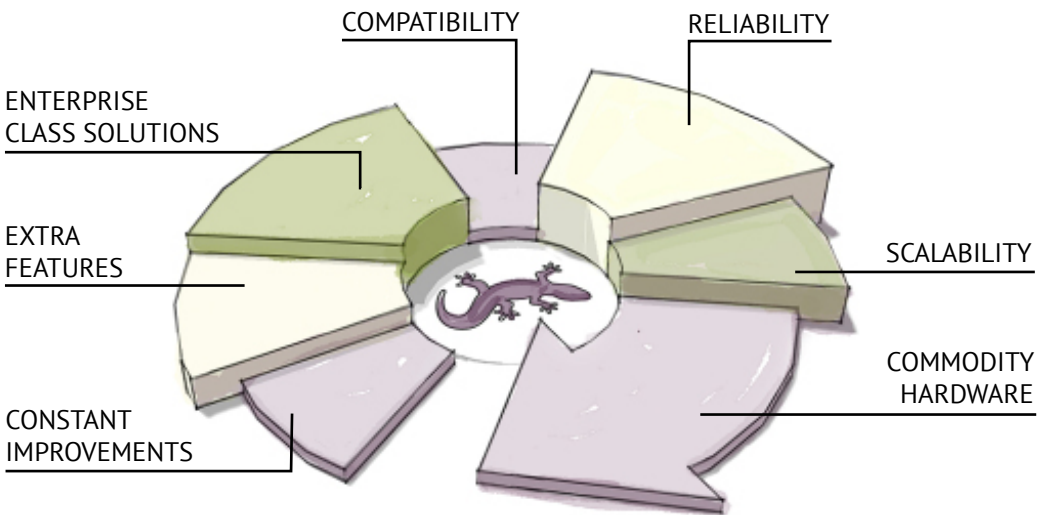
LizardFS – Software Defined Storage is a distributed, scalable, fault-tolerant and highly available file system. It allows users to combine disk space located on several servers into a single name space which is visible on Unix-like and Windows systems in the same way as other file systems.

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 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 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.

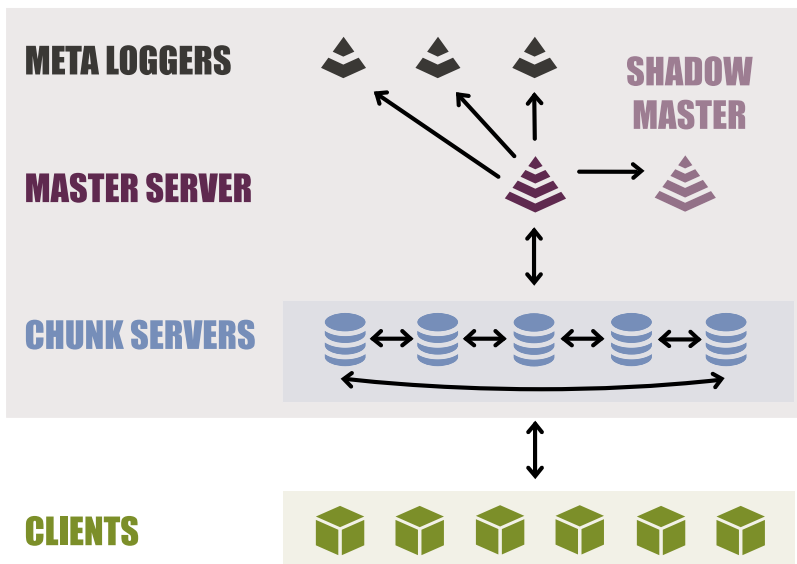


Figure 1: Architecture of LizardFS

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

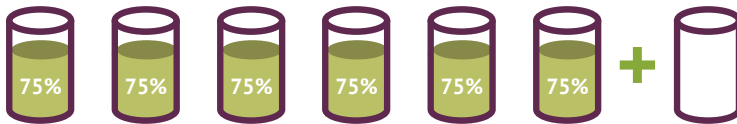
Consider the following installation:

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 disk usage is almost even, e.g.:

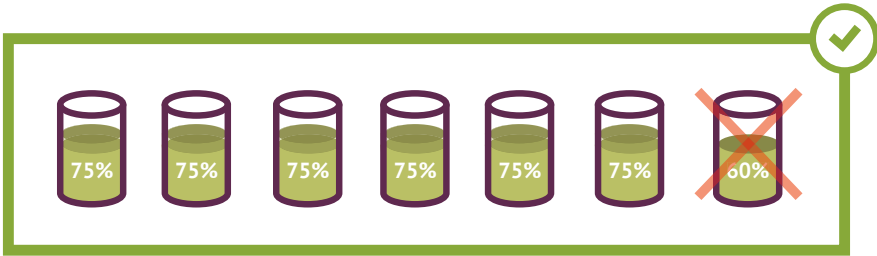
server	total space	used space
1	20 TB	12 TB (60%)
2	16 TB	9.6 TB (60%)
3	16 TB	9.6 TB (60%)
4	16 TB	9.6 TB (60%)
5	8 TB	4.8 TB (60%)
6	8 TB	4.8 TB (60%)
7	21 TB	12.6 TB (60%)
TOTAL	105 TB	63 TB (60%)

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 the 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: minimum 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

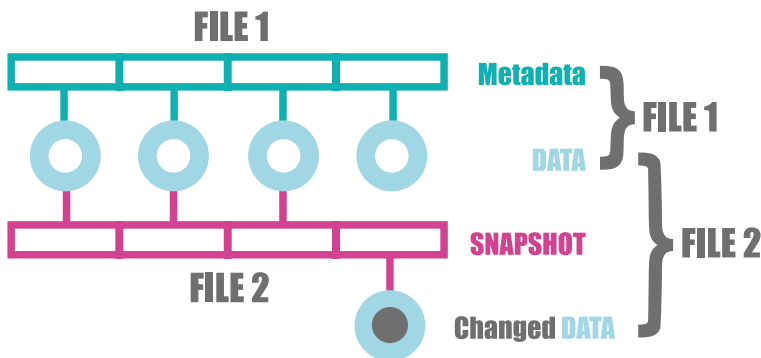
Features

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. 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 using the ``cp -r huge_vm huge_vm_backup`` command would take a long time and consume an additional 512GB of disk space.

We recommend instead to create a copy using the ``mfmakesnapshot 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 ``mfmakesnapshot`` 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 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 accidentally disrupt the production system. Any processes that was 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.

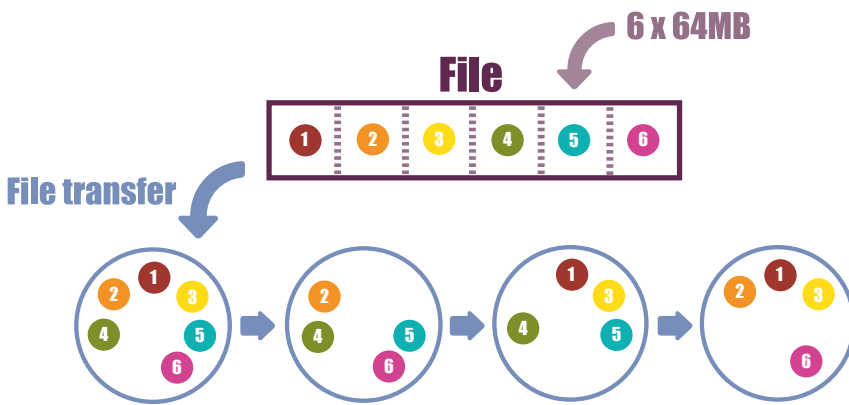
Data replication

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

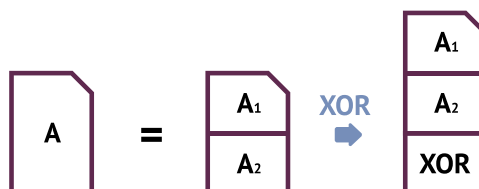
This kind of replication allows for setting the number of copies every chunk should have. Note that it is not possible to store 2 of the same chunks on 1 chunkserver, meaning standard goals should comply with the number of chunkservers and their labels.

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.

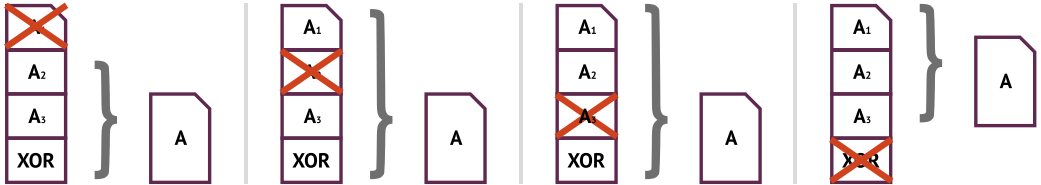
XOR replica

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 chunkserver fails, it is still possible to reconstruct the file from the remaining 3 XOR chunks.

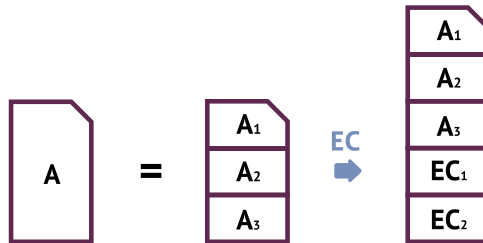


EC replica - Erasure Coding replica (since version 3.10.0)

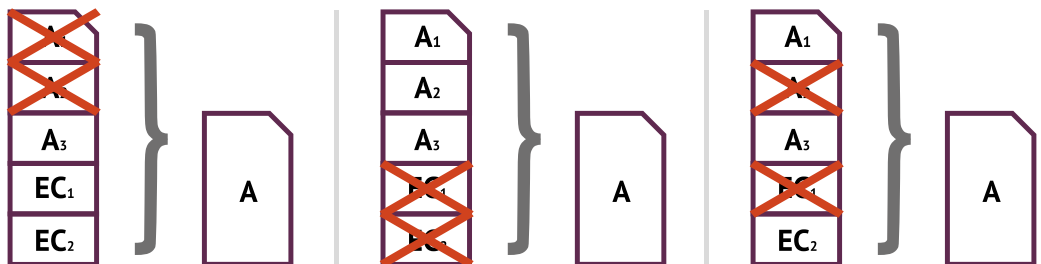
For each file using an $ec(K+M)$ goal, the system will split the file into K parts and generate M parity parts. The K parts are required for reproducing the file content. If labels are specified, labelled parts will be kept on chunkservers. Otherwise, default wildcard labels are used.

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 labelling 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” labelled server, and one on a “paris” labelled 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 `“save_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 balances disk usage across all servers, as defined by the 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 need for maintenance work on the master server.

Moreover, you can increase 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 file systems. It offers an option to set soft and hard limits for 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:

- 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

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 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 Windows™ applications.

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 an 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 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 built 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 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 Contact:

info@lizardfs.com

Poland tel. +48 22 100 32 48

USA tel. +1 (646) 655 0693

visit our website:

www.lizardfs.com

Company contact:

Skytechnology sp. z o.o.

ul. Głogowa 21

02-639 Warsaw

NIP: 5213633370

KRS: 0000426381

tel. +48 22 100 32 48

fax. +48 22 250 29 13

info@skytechnology.pl

visit our website:

www.skytechnology.pl



WWW.LIZARDFS.COM



INFO@LIZARDFS.COM



[LIZARDFS](https://www.linkedin.com/company/lizardfs)



[@LIZARDFS](https://twitter.com/LIZARDFS)

