

2. Further LINSTOR tasks

- [2.1. LINSTOR high availability](#)
- [2.2. DRBD clients](#)
- [2.3. LINSTOR – DRBD consistency group/multiple volumes](#)
- [2.4. Volumes of one resource to different Storage-Pools](#)
- [2.5. LINSTOR without DRBD](#)
- [2.6. Storage Providers](#)
- [2.7. Managing Network Interface Cards](#)
- [2.8. Encrypted volumes](#)
- [2.9. Checking the state of your cluster](#)
- [2.10. Managing snapshots](#)
- [2.11. Setting options for resources](#)
- [2.12. Scheduled Backup Shipping](#)
- [2.13. Adding and removing disks](#)
- [2.14. DRBD Proxy with LINSTOR](#)
- [2.15. External database](#)
- [2.16. LINSTOR REST-API](#)
- [2.17. Logging](#)
- [2.18. Monitoring](#)
- [2.19. Secure Satellite connections](#)
- [2.20. Automatisms for DRBD-Resources](#)
- [2.21. QoS Settings](#)
- [2.22. Getting help](#)

2.1. LINSTOR high availability

By default a LINSTOR cluster consists of exactly one LINSTOR controller. Making LINSTOR highly-available involves providing replicated storage for the controller database, multiple LINSTOR controllers where only one is active at a time, and a service manager that takes care of mounting/unmounting the highly-available storage and starting/stopping LINSTOR controllers.

2.1.1. Highly-available Storage

For configuring the highly-available storage we use LINSTOR itself. This has the advantage that the storage is under LINSTOR control and can for example be easily extended to new cluster nodes. Just create a new resource with 200MB in size. This could look like this, you certainly need to adapt the storage pool name:

```
# linstor resource-definition create linstor_db
# linstor resource-definition drbd-options --on-no-quorum=io-error linstor_db
# linstor resource-definition drbd-options --auto-promote=no linstor_db
# linstor volume-definition create linstor_db 200M
# linstor resource create linstor_db -s pool1 --auto-place 3
```

From now on we assume the resource's name is "linstor_db". It is crucial that your cluster qualifies for auto-quorum and uses the io-error policy (see Section [AutoQuorum Policies](#)), and that auto-promote is disabled.

After the resource is created, it is time to move the LINSTOR DB to the new storage and to create a systemd mount service. First we stop the current controller and disable it, as it will be managed by drbd-reactor later.

```
# systemctl disable --now linstor-controller

# cat << EOF > /etc/systemd/system/var-lib-linstor.mount
```

[Unit]

```
Description=Filesystem for the LINSTOR controller
```

[Mount]

```
# you can use the minor like /dev/drbdX or the udev symlink
What=/dev/drbd/by-res/linstor_db/0
```

```
Where=/var/lib/linstor
EOF

# mv /var/lib/linstor{,.orig}
# mkdir /var/lib/linstor
# chatter +i /var/lib/linstor # only if on LINSTOR >= 1.14.0
# drbdadm primary linstor_db
# mkfs.ext4 /dev/drbd/by-res/linstor_db/0
# systemctl start var-lib-linstor.mount
# cp -r /var/lib/linstor.orig/* /var/lib/linstor
# systemctl start linstor-controller
```

copy the `/etc/systemd/system/var-lib-linstor.mount` mount file to all the standby nodes for the linstor controller. Again, do not `systemctl` enable any of these services, they get managed by `drbd-reactor`.

2.1.2. Multiple LINSTOR controllers

The next step is to install LINSTOR controllers on all nodes that have access to the `linstor_db` DRBD resource (as they need to mount the DRBD volume) and which you want to become a possible LINSTOR controller. It is important that the controllers are managed by `drbd-reactor`, so make sure the `linstor-controller.service` is disabled on all nodes! To be sure, execute `systemctl disable linstor-controller` on all cluster nodes and `systemctl stop linstor-controller` on all nodes except the one it is currently running from the previous step. Also make sure to set `chattr +i /var/lib/linstor` on all potential controller nodes if you use LINSTOR version equal or greater to 1.14.0.

2.1.3. Managing the services

For starting and stopping the mount service and the `linstor-controller` service we use `drbd-reactor`. Install this component on all nodes that could become a LINSTOR controller and edit their `/etc/drbd-reactor.d/linstor_db.toml` configuration file. It should contain an enabled promoter plugin section like this:

```
[[promoter]]
id = "linstor_db"

[promoter.resources.linstor_db]
start = ["var-lib-linstor.mount", "linstor-controller.service"]
```

Depending on your requirements you might also want to set an on-stop-failure action and set `stop-services-on-exit`.

After that restart `drbd-reactor` and enable it on all the nodes you configured it.

```
# systemctl restart drbd-reactor
# systemctl enable drbd-reactor
```

Check that there are no warnings from drbd-reactor in the logs by running `systemctl status drbd-reactor`. As there is already an active LINSTOR controller things will just stay the way they are. Run `drbd-reactorctl status linstor_db` to check the health of the `linstor_db` target unit.

The last but never the less important step is to configure the LINSTOR satellite services to not delete (and then regenerate) the resource file for the LINSTOR controller DB at its startup. Do not edit the service files directly, but use `systemctl edit`. Edit the service file on all nodes that could become a LINSTOR controller and that are also LINSTOR satellites.

```
# systemctl edit linstor-satellite
```

[Service]

```
Environment=LS_KEEP_RES=linstor_db
```

After this change you should execute `systemctl restart linstor-satellite` on all satellite nodes.

Be sure to configure your LINSTOR client for use with multiple controllers as described in the section titled, [Using the LINSTOR client](#) and make sure you also configured your integration plugins (e.g., the Proxmox plugin) to be ready for multiple LINSTOR controllers.

2.2. DRBD clients

By using the `--drbd-diskless` option instead of `--storage-pool` you can have a permanently diskless DRBD device on a node. This means that the resource will appear as block device and can be mounted to the filesystem without an existing storage-device. The data of the resource is accessed over the network on another nodes with the same resource.

```
# linstor resource create delta backups --drbd-diskless
```

The option `--diskless` was deprecated. Please use `--drbd-diskless` or `--nvme-initiator` instead.

2.3. LINSTOR – DRBD consistency group/multiple volumes

The so called consistency group is a feature from DRBD. It is mentioned in this user-guide, due to the fact that one of LINSTOR's main functions is to manage storage-clusters with DRBD. Multiple volumes in one resource are a consistency group.

This means that changes on different volumes from one resource are getting replicated in the same chronological order on the other Satellites.

Therefore you don't have to worry about the timing if you have interdependent data on different volumes in a resource.

To deploy more than one volume in a LINSTOR-resource you have to create two volume-definitions with the same name.

```
# linstor volume-definition create backups 500G  
# linstor volume-definition create backups 100G
```

2.4. Volumes of one resource to different Storage-Pools

This can be achieved by setting the `StorPoolName` property to the volume definitions before the resource is deployed to the nodes:

```
# linstor resource-definition create backups
# linstor volume-definition create backups 500G
# linstor volume-definition create backups 100G
# linstor volume-definition set-property backups 0 StorPoolName pool_hdd
# linstor volume-definition set-property backups 1 StorPoolName pool_ssd
# linstor resource create alpha backups
# linstor resource create bravo backups
# linstor resource create charlie backups
```

Since the `volume-definition create` command is used without the `--vlmnr` option LINSTOR assigned the volume numbers starting at 0. In the following two lines the 0 and 1 refer to these automatically assigned volume numbers.

Here the ‘resource create’ commands do not need a `--storage-pool` option. In this case LINSTOR uses a ‘fallback’ storage pool. Finding that storage pool, LINSTOR queries the properties of the following objects in the following order:

- Volume definition
- Resource
- Resource definition
- Node

If none of those objects contain a `StorPoolName` property, the controller falls back to a hard-coded ‘`DfltStorPool`’ string as a storage pool.

This also means that if you forgot to define a storage pool prior deploying a resource, you will get an error message that LINSTOR could not find the storage pool named ‘`DfltStorPool`’.

2.5. LINSTOR without DRBD

LINSTOR can be used without DRBD as well. Without DRBD, LINSTOR is able to provision volumes from LVM and ZFS backed storage pools, and create those volumes on individual nodes in your LINSTOR cluster.

Currently LINSTOR supports the creation of LVM and ZFS volumes with the option of layering some combinations of LUKS, DRBD, and/or NVMe-oF/NVMe-TCP on top of those volumes.

For example, assume we have a Thin LVM backed storage pool defined in our LINSTOR cluster named, thin-lvm:

```
# linstor --no-utf8 storage-pool list
```

StoragePool	Node	Driver	PoolName	...
thin-lvm	linstor-a	LVM_THIN	drbdpool/thinpool	...
thin-lvm	linstor-b	LVM_THIN	drbdpool/thinpool	...
thin-lvm	linstor-c	LVM_THIN	drbdpool/thinpool	...
thin-lvm	linstor-d	LVM_THIN	drbdpool/thinpool	...

We could use LINSTOR to create a Thin LVM on linstor-d that's 100GiB in size using the following commands:

```
# linstor resource-definition create rsc-1
# linstor volume-definition create rsc-1 100GiB
# linstor resource create --layer-list storage \
    --storage-pool thin-lvm linstor-d rsc-1
```

You should then see you have a new Thin LVM on linstor-d. You can extract the device path from LINSTOR by listing your linstor resources with the --machine-readable flag set:

```
# linstor --machine-readable resource list | grep device_path
"device_path": "/dev/drbdpool/rsc-1_00000",
```

If you wanted to layer DRBD on top of this volume, which is the default --layer-list option in LINSTOR for ZFS or LVM backed volumes, you would use the following resource creation pattern instead:

```
# linstor resource-definition create rsc-1
# linstor volume-definition create rsc-1 100GiB
# linstor resource create --layer-list drbd,storage \
    --storage-pool thin-lvm linstor-d rsc-1
```

You would then see that you have a new Thin LVM backing a DRBD volume on linstor-d:

```
# linstor --machine-readable resource list | grep -e device_path -e backing_disk
    "device_path": "/dev/drbd1000",
    "backing_disk": "/dev/drbdpool/rsc-1_00000",
```

The following table shows which layer can be followed by which child-layer:

Layer	Child layer
DRBD	CACHE, WRITECACHE, NVME, LUKS, STORAGE
CACHE	WRITECACHE, NVME, LUKS, STORAGE
WRITECACHE	CACHE, NVME, LUKS, STORAGE
NVME	CACHE, WRITECACHE, LUKS, STORAGE
LUKS	STORAGE
STORAGE	-

One layer can only occur once in the layer-list

For information about the prerequisites for the LUKS layer, refer to the Encrypted Volumes section of this User's Guide.

2.5.1. NVMe-oF/NVMe-TCP LINSTOR Layer

NVMe-oF/NVMe-TCP allows LINSTOR to connect diskless resources to a node with the same resource where the data is stored over NVMe fabrics. This leads to the advantage that resources can be mounted without using local storage by accessing the data over the network. LINSTOR is not using DRBD in this case, and therefore NVMe resources provisioned by LINSTOR are not replicated, the data is stored on one node.

NVMe-oF only works on RDMA-capable networks and NVMe-TCP on every network that can carry IP traffic. If you want to know more about NVMe-oF/NVMe-TCP visit <https://www.linbit.com/en/nvme-linstor-swordfish/> for more information.

To use NVMe-oF/NVMe-TCP with LINSTOR the package nvme-cli needs to be installed on every Node which acts as a Satellite and will use NVMe-oF/NVMe-TCP for a resource:

If you are not using Ubuntu use the suitable command for installing packages on your OS –
SLES: zypper – CentOS: yum

```
# apt install nvme-cli
```

To make a resource which uses NVMe-oF/NVMe-TCP an additional parameter has to be given as you create the resource-definition:

```
# linstor resource-definition create nvmedata -l nvme,storage
```

As default the -l (layer-stack) parameter is set to drbd, storage when DRBD is used. If you want to create LINSTOR resources with neither NVMe nor DRBD you have to set the -l parameter to only storage.

In order to use NVMe-TCP instead of the default NVMe-oF the following property needs to be set:

```
# linstor resource-definition set-property nvmedata NVMe/TRType tcp
```

The property NVMe/TRType can alternatively be set on resource-group or controller level.

Next, create the volume-definition for our resource:

```
# linstor volume-definition create nvmedata 500G
```

Before you create the resource on your nodes you have to know where the data will be stored locally and which node accesses it over the network.

First we create the resource on the node where our data will be stored:

```
# linstor resource create alpha nvmedata --storage-pool pool_ssd
```

On the nodes where the resource-data will be accessed over the network, the resource has to be defined as diskless:

```
# linstor resource create beta nvmedata --nvme-initiator
```

Now you can mount the resource nvmedata on one of your nodes.

If your nodes have more than one NIC you should force the route between them for NVMe-oF/NVMe-TCP, otherwise multiple NICs could cause troubles.

2.5.2. OpenFlex™ Layer

Since version 1.5.0 the additional Layer openflex can be used in LINSTOR. From LINSTOR's perspective, the [OpenFlex Composable Infrastructure](#) takes the role of a combined layer acting as a storage layer (like LVM) and also providing the allocated space as an NVMe target. OpenFlex has a REST API which is also used by LINSTOR to operate with.

As OpenFlex combines concepts of LINSTOR's storage as well as NVMe-layer, LINSTOR was added both, a new storage driver for the storage pools as well as a dedicated openflex layer which uses the mentioned REST API.

In order for LINSTOR to communicate with the OpenFlex-API, LINSTOR needs some additional properties, which can be set once on controller level to take LINSTOR-cluster wide effect:

- StorDriver/Openflex/ApiHost specifies the host or IP of the API entry-point
- StorDriver/Openflex/ApiPort this property is glued with a colon to the previous to form the basic [http://ip:port](#) part used by the REST calls
- StorDriver/Openflex/UserName the REST username
- StorDriver/Openflex/UserPassword the password for the REST user

Once that is configured, we can now create LINSTOR objects to represent the OpenFlex architecture. The theoretical mapping of LINSTOR objects to OpenFlex objects are as follows: Obviously an OpenFlex storage pool is represented by a LINSTOR storage pool. As the next thing above a LINSTOR storage pool is already the node, a LINSTOR node represents an OpenFlex storage device. The OpenFlex objects above storage device are not mapped by LINSTOR.

When using NVMe, LINSTOR was designed to run on both sides, the NVMe target as well as on the NVMe initiator side. In the case of OpenFlex, LINSTOR cannot (or even should not) run on the NVMe target side as that is completely managed by OpenFlex. As LINSTOR still needs nodes and storage pools to represent the OpenFlex counterparts, the LINSTOR client was extended with special node create commands since 1.0.14. These commands not only accept additionally needed configuration data, but also starts a "special satellite" besides the already running controller instance. This special satellites are completely LINSTOR managed, they will shutdown when the controller shuts down and will be started again when the controller starts. The new client command for creating a "special satellite" representing an OpenFlex storage device is:

```
$ linstor node create-openflex-target ofNode1 192.168.166.7 000af795789d
```

The arguments are as follows:

- ofNode1 is the node name which is also used by the standard linstor node create command
- 192.168.166.7 is the address on which the provided NVMe devices can be accessed. As the NVMe devices are accessed by a dedicated network interface, this address differs from the address specified with the property StorDriver/Openflex/ApiHost. The latter is

used for the management / REST API.

- 000af795789d is the identifier for the OpenFlex storage device.

The last step of the configuration is the creation of LINSTOR storage pools:

```
$ linstor storage-pool create openflex ofNode1 sp0 0
```

- ofNode1 and sp0 are the node name and storage pool name, respectively, just as usual for the LINSTOR's create storage pool command
- The last 0 is the identifier of the OpenFlex storage pool within the previously defined storage device

Once all necessary storage pools are created in LINSTOR, the next steps are similar to the usage of using an NVMe resource with LINSTOR. Here is a complete example:

```
# set the properties once
linstor controller set-property StorDriver/Openflex/ApiHost 10.43.7.185
linstor controller set-property StorDriver/Openflex/ApiPort 80
linstor controller set-property StorDriver/Openflex/UserName myusername
linstor controller set-property StorDriver/Openflex/UserPassword mypassword
```

```
# create a node for openflex storage device "000af795789d"
linstor node create-openflex-target ofNode1 192.168.166.7 000af795789d
```

```
# create a usual linstor satellite. later used as nvme initiator
linstor node create bravo
```

```
# create a storage pool for openflex storage pool "0" within storage device "000af795789d"
linstor storage-pool create openflex ofNode1 sp0 0
```

```
# create resource- and volume-definition
linstor resource-definition create backupRsc
linstor volume-definition create backupRsc 10G
```

```
# create openflex-based nvme target
linstor resource create ofNode1 backupRsc --storage-pool sp0 --layer-list openflex
```

```
# create openflex-based nvme initiator
linstor resource create bravo backupRsc --nvme-initiator --layer-list openflex
```

In case a node should access the OpenFlex REST API through a different host than specified with `linstor controller set-property StorDriver/Openflex/ApiHost 10.43.7.185` you can always use LINSTOR's inheritance mechanism for properties. That means simply define the same property on the node-level you need it, i.e. `linstor node set-property ofNode1 StorDriver/Openflex/ApiHost 10.43.8.185`

2.5.3. Writecache Layer

A [DM-Writecache](#) device is composed by two devices, one storage device and one cache device. LINSTOR can setup such a writecache device, but needs some additional information, like the storage pool and the size of the cache device.

```
# linstor storage-pool create lvm node1 lvmpool drbdpool
# linstor storage-pool create lvm node1 pmempool pmempool
```

```
# linstor resource-definition create r1
# linstor volume-definition create r1 100G
```

```
# linstor volume-definition set-property r1 0 Writecache/PoolName pmempool
# linstor volume-definition set-property r1 0 Writecache/Size 1%
```

```
# linstor resource create node1 r1 --storage-pool lvmpool --layer-list WRITECACHE,STORAGE
```

The two properties set in the examples are mandatory, but can also be set on controller level which would act as a default for all resources with WRITECACHE in their `--layer-list`. However, please note that the `Writecache/PoolName` refers to the corresponding node. If the node does not have a storage-pool named `pmempool` you will get an error message.

The 4 mandatory parameters required by [DM-Writecache](#) are either configured via property or figured out by LINSTOR. The optional properties listed in the mentioned link can also be set via property. Please see `linstor controller set-property --help` for a list of `Writecache/*` property-keys.

Using `--layer-list DRBD,WRITECACHE,STORAGE` while having DRBD configured to use external metadata, only the backing device will use a writecache, not the device holding the external metadata.

2.5.4. Cache Layer

LINSTOR can also setup a [DM-Cache](#) device, which is very similar to the DM-Writecache from the previous section. The major difference is that a cache device is composed by three devices: one storage device, one cache device and one meta device. The LINSTOR properties are quite similar to those of the writecache but are located in the Cache namespace:

```
# linstor storage-pool create lvm node1 lvmpool drbdpool
# linstor storage-pool create lvm node1 pmempool pmempool
```

```
# linstor resource-definition create r1
# linstor volume-definition create r1 100G
```

```
# linstor volume-definition set-property r1 0 Cache/CachePool pmempool
# linstor volume-definition set-property r1 0 Cache/Cachesize 1%
```

```
# linstor resource create node1 r1 --storage-pool lvmpool --layer-list CACHE,STORAGE
```

Instead of Writecache/PoolName (as when configuring the Writecache layer) the Cache layer's only required property is called Cache/CachePool. The reason for this is that the Cache layer also has a Cache/MetaPool which can be configured separately or it defaults to the value of Cache/CachePool.

Please see `linstor controller set-property --help` for a list of Cache/* property-keys and default values for omitted properties.

Using `--layer-list DRBD,CACHE,STORAGE` while having DRBD configured to use external metadata, only the backing device will use a cache, not the device holding the external metadata.

2.5.5. Storage Layer

The storage layer will provide new devices from well known volume managers like LVM, ZFS or others. Every layer combination needs to be based on a storage layer, even if the resource should be diskless – for that type there is a dedicated diskless provider type.

For a list of providers with their properties please see [Storage Providers](#).

For some storage providers LINSTOR has special properties:

- `StorDriver/WaitTimeoutAfterCreate`: If LINSTOR expects a device to appear after creation (for example after calls of `lvcreate`, `zfs create`,...), LINSTOR waits per default 500ms for the device to appear. These 500ms can be overridden by this property.
- `StorDriver/dm_stats`: If set to true LINSTOR calls `dmstats create $device` after creation and `dmstats delete $device --allregions` after deletion of a volume. Currently only enabled for LVM and LVM_THIN storage providers.

2.6. Storage Providers

LINSTOR has a few storage providers. The most used ones are LVM and ZFS. But also for those two providers there are already sub-types for their thinly provisioned variants.

- **Diskless:** This provider type is mostly required to have a storage pool that can be configured with LINSTOR properties like PrefNic as described in [Managing Network Interface Cards](#).
- **LVM / LVM-Thin:** The administrator is expected to specify the LVM volume group or the thin-pool (in form of “LV/thinpool”) in order to use the corresponding storage type. These drivers support following properties for fine-tuning:
 - **StorDriver/LvcreateOptions:** The value of this property is appended to every lvcreate ... call LINSTOR executes.
- **ZFS / ZFS-Thin:** The administrator is expected to specify the ZPool that LINSTOR should use. These drivers support following properties for fine-tuning:
 - **StorDriver/ZfscreateOptions:** The value of this property is appended to every zfs create ... call LINSTOR executes.
- **File / FileThin:** Mostly used for demonstration / experiments. LINSTOR will basically reserve a file in a given directory and will configure a [loop device](#) on top of that file.
- **OpenFlex:** This special storage provider currently requires to be run on a “special satellite”. Please see [OpenFlex™ Layer](#) for more details.
- **EXOS:** This special storage provider currently requires to be run on a “special satellite”. Please see the [EXOS Integration](#) chapter
- **SPDK:** The administrator is expected to specify the logical volume store which LINSTOR should use. The usage of this storage provider implies the usage of the [NVME Layer](#).
 - **Remote-SPDK:** This special storage provider currently requires to be run on a “special satellite”. Please see [Remote SPDK Provider](#) for more details.

2.6.1. Remote SPDK Provider

A storage pool with the type remote SPDK can only be created on a “special satellite”. For this you first need to start a new satellite using the command:

```
$ linstor node create-remote-spdk-target nodeName 192.168.1.110
```

This will start a new satellite instance running on the same machine as the controller. This special satellite will do all the REST based RPC communication towards the remote SPDK proxy. As the help message of the LINSTOR command shows, the administrator might want to use additional settings when creating this special satellite:


```
$ linstor node create-remote-spdk-target -h
usage: linstor node create-remote-spdk-target [-h] [--api-port API_PORT]
                                           [--api-user API_USER]
                                           [--api-user-env API_USER_ENV]
                                           [--api-pw [API_PW]]
                                           [--api-pw-env API_PW_ENV]
                                           node_name api_host
```

The difference between the `--api-*` and their corresponding `--api-*-env` versions is that the version with the `-env` ending will look for an environment variable containing the actual value to use whereas the `--api-*` version directly take the value which is stored in the LINSTOR property. Administrators might not want to save the `--api-pw` in plaintext, which would be clearly visible using commands like `linstor node list-property` .

Once that special satellite is up and running the actual storage pool can be created:

```
$ linstor storage-pool create remotespdk -h
usage: linstor storage-pool create remotespdk [-h]
                                           [--shared-space SHARED_SPACE]
                                           [--external-locking]
                                           node_name name driver_pool_name
```

Whereas `node_name` is self-explanatory, `name` is the name of the LINSTOR storage pool and `driver_pool_name` refers to the SPDK logical volume store.

Once this `remotespdisk` storage pool is created the remaining procedure is quite similar as using NVMe: First the target has to be created by creating a simple “diskful” resource followed by a second resource having the `--nvme-initiator` option enabled.

2.7. Managing Network Interface Cards

LINSTOR can deal with multiple network interface cards (NICs) in a machine, they are called netif in LINSTOR speak.

When a satellite node is created a first netif gets created implicitly with the name default. Using the `--interface-name` option of the node create command you can give it a different name.

Additional NICs are created like this:

```
# linstor node interface create alpha 100G_nic 192.168.43.221
# linstor node interface create alpha 10G_nic 192.168.43.231
```

NICs are identified by the IP address only, the name is arbitrary and is **not** related to the interface name used by Linux. The NICs can be assigned to storage pools so that whenever a resource is created in such a storage pool, the DRBD traffic will be routed through the specified NIC.

```
# linstor storage-pool set-property alpha pool_hdd PrefNic 10G_nic
# linstor storage-pool set-property alpha pool_ssd PrefNic 100G_nic
```

FIXME describe how to route the controller <-> client communication through a specific netif.

2.8. Encrypted volumes

LINSTOR can handle transparent encryption of drbd volumes. dm-crypt is used to encrypt the provided storage from the storage device.

In order to use dm-crypt please make sure to have cryptsetup installed before you start the satellite

Basic steps to use encryption:

1. Disable user security on the controller (this will be obsolete once authentication works)
2. Create a master passphrase
3. Add luks to the layer-list. Note that all plugins (e.g., Proxmox) require a DRBD layer as the top most layer if they do not explicitly state otherwise.
4. Don't forget to re-enter the master passphrase after a controller restart.

2.8.1. Disable user security

Disabling the user security on the Linstor controller is a one time operation and is afterwards persisted.

1. Stop the running linstor-controller via systemd: `systemctl stop linstor-controller`
2. Start a linstor-controller in debug mode: `/usr/share/linstor-server/bin/Controller -c /etc/linstor -d`
3. In the debug console enter: `setSecLvl secLvl(NO_SECURITY)`
4. Stop linstor-controller with the debug shutdown command: `shutdown`
5. Start the controller again with systemd: `systemctl start linstor-controller`

2.8.2. Encrypt commands

Below are details about the commands.

Before LINSTOR can encrypt any volume a master passphrase needs to be created. This can be done with the `linstor-client`.

```
# linstor encryption create-passphrase
```

`crypt-create-passphrase` will wait for the user to input the initial master passphrase (as all other crypt commands will with no arguments).

If you ever want to change the master passphrase this can be done with:

```
# linstor encryption modify-passphrase
```

The luks layer can be added when creating the resource-definition or the resource itself, whereas the former method is recommended since it will be automatically applied to all resource created from that resource-definition.

```
# linstor resource-definition create crypt_rsc --layer-list luks,storage
```

To enter the master passphrase (after controller restart) use the following command:

```
# linstor encryption enter-passphrase
```

Whenever the linstor-controller is restarted, the user has to send the master passphrase to the controller, otherwise LINSTOR is unable to reopen or create encrypted volumes.

2.8.3. Automatic Passphrase

It is possible to automate the process of creating and re-entering the master passphrase.

To use this, either an environment variable called `MASTER_PASSPHRASE` or an entry in `/etc/linstor/linstor.toml` containing the master passphrase has to be created.

The required `linstor.toml` looks like this:

```
[encrypt]
```

```
passphrase="example"
```

If either one of these is set, then every time the controller starts it will check whether a master passphrase already exists. If there is none, it will create a new master passphrase as specified. Otherwise, the controller enters the passphrase.

If a master passphrase is already configured, and it is not the same one as specified in the environment variable or `linstor.toml`, the controller will be unable to re-enter the master passphrase and react as if the user had entered a wrong passphrase. This can only be resolved through manual input from the user, using the same commands as if the controller was started without the automatic passphrase.

In case the master passphrase is set in both an environment variable and the `linstor.toml`, only the master passphrase from the `linstor.toml` will be used.

2.9. Checking the state of your cluster

LINSTOR provides various commands to check the state of your cluster. These commands start with a 'list-' prefix and provide various filtering and sorting options. The '-groupby' option can be used to group and sort the output in multiple dimensions.

```
# linstor node list  
# linstor storage-pool list --groupby Size
```

2.10. Managing snapshots

Snapshots are supported with thin LVM and ZFS storage pools.

2.10.1. Creating a snapshot

Assuming a resource definition named 'resource1' which has been placed on some nodes, a snapshot can be created as follows:

```
# linstor snapshot create resource1 snap1
```

This will create snapshots on all nodes where the resource is present. LINSTOR will ensure that consistent snapshots are taken even when the resource is in active use.

Setting the resource-definition property AutoSnapshot/RunEvery LINSTOR will automatically create snapshots every X minute. The optional property AutoSnapshot/Keep can be used to clean-up old snapshots which were created automatically. No manually created snapshot will be cleaned-up / deleted. If AutoSnapshot/Keep is omitted (or ≤ 0), LINSTOR will keep the last 10 snapshots by default.

```
# linstor resource-definition set-property AutoSnapshot/RunEvery 15
# linstor resource-definition set-property AutoSnapshot/Keep 5
```

2.10.2. Restoring a snapshot

The following steps restore a snapshot to a new resource. This is possible even when the original resource has been removed from the nodes where the snapshots were taken.

First define the new resource with volumes matching those from the snapshot:

```
# linstor resource-definition create resource2
# linstor snapshot volume-definition restore --from-resource resource1 --from-snapshot snap1 --to-resource resource2
```

At this point, additional configuration can be applied if necessary. Then, when ready, create resources based on the snapshots:

```
# linstor snapshot resource restore --from-resource resource1 --from-snapshot snap1 --to-resource resource2
```

This will place the new resource on all nodes where the snapshot is present. The nodes on which to place the resource can also be selected explicitly; see the help (linstor snapshot resource restore -

h).

2.10.3. Rolling back to a snapshot

LINSTOR can roll a resource back to a snapshot state. The resource must not be in use. That is, it may not be mounted on any nodes. If the resource is in use, consider whether you can achieve your goal by [restoring the snapshot](#) instead.

Rollback is performed as follows:

```
# linstor snapshot rollback resource1 snap1
```

A resource can only be rolled back to the most recent snapshot. To roll back to an older snapshot, first delete the intermediate snapshots.

2.10.4. Removing a snapshot

An existing snapshot can be removed as follows:

```
# linstor snapshot delete resource1 snap1
```

2.10.5. Shipping a snapshot

Both, the source as well as the target node have to have the resource for snapshot shipping deployed. Additionally, the target resource has to be deactivated.

```
# linstor resource deactivate nodeTarget resource1
```

Deactivating a resource with DRBD in its layer-list can NOT be reactivated again. However, a successfully shipped snapshot of a DRBD resource can still be restored into a new resource.

To manually start the snapshot-shipping, use:

```
# linstor snapshot ship --from-node nodeSource --to-node nodeTarget --resource resource1
```

By default, the snapshot-shipping uses tcp ports from the range 12000-12999. To change this range, the property SnapshotShipping/TcpPortRange, which accepts a to-from range, can be set on the controller:

```
# linstor controller set-property SnapshotShipping/TcpPortRange 10000-12000
```

A resource can also be periodically shipped. To accomplish this, it is mandatory to set the properties SnapshotShipping/TargetNode as well as SnapshotShipping/RunEvery on the resource-

definition. SnapshotShipping/SourceNode can also be set, but if omitted LINSTOR will choose an active resource of the same resource-definition.

To allow incremental snapshot-shipping, LINSTOR has to keep at least the last shipped snapshot on the target node. The property SnapshotShipping/Keep can be used to specify how many snapshots LINSTOR should keep. If the property is not set (or ≤ 0) LINSTOR will keep the last 10 shipped snapshots by default.

```
# linstor resource-definition set-property resource1 SnapshotShipping/TargetNode nodeTarget
# linstor resource-definition set-property resource1 SnapshotShipping/SourceNode nodeSource
# linstor resource-definition set-property resource1 SnapshotShipping/RunEvery 15
# linstor resource-definition set-property resource1 SnapshotShipping/Keep 5
```


2.11. Setting options for resources

DRBD options are set using LINSTOR commands. Configuration in files such as `/etc/drbd.d/global_common.conf` that are not managed by LINSTOR will be ignored. The following commands show the usage and available options:

```
# linstor controller drbd-options -h
# linstor resource-definition drbd-options -h
# linstor volume-definition drbd-options -h
# linstor resource drbd-peer-options -h
```

For instance, it is easy to set the DRBD protocol for a resource named backups:

```
# linstor resource-definition drbd-options --protocol C backups
```

2.12. Scheduled Backup Shipping

Starting with LINSTOR Controller version 1.19.0 and working with LINSTOR client version 1.14.0 or above, you can configure scheduled backup shipping for deployed LINSTOR resources.

Scheduled backup shipping consists of three parts:

- A data set that consists of one or more deployed LINSTOR resources that you want to backup and ship
- A remote destination to ship backups to (another LINSTOR cluster or an S3 instance)
- A schedule that defines when the backups should ship

LINSTOR backup shipping only works for deployed LINSTOR resources that are backed by LVM and ZFS storage pools, because these are the storage pool types with snapshot support in LINSTOR.

2.12.1. Creating a Backup Shipping Schedule

You create a backup shipping schedule by using the LINSTOR client `schedule create` command and defining the frequency of backup shipping using `cron` syntax. You also need to set options that name the schedule and define various aspects of the backup shipping, such as on-failure actions, the number of local and remote backup copies to keep, and whether to also schedule incremental backup shipping.

At a minimum, the command needs a schedule name and a full backup cron schema to create a backup shipping schedule. An example command would look like this:

```
# linstor schedule create \  
--incremental-cron '* * * * *' \ (1)  
--keep-local 5 \ (2)  
--keep-remote 4 \ (3)  
--on-failure RETRY \ (4)  
--max-retries 10 \ (5)  
<schedule_name> \ (6)  
'* * * * *' # full backup cron schema (7)
```

Enclose cron schemas within single or double quotation marks.

1	If specified, the incremental cron schema describes how frequently to create and ship incremental backups. New incremental backups are based on the most recent full backup.
2	The --keep-local option allows you to specify how many snapshots that a full backup is based upon should be kept at the local backup source. If unspecified, all snapshots will be kept.[OPTIONAL]
3	The --keep-remote option allows you to specify how many full backups should be kept at the remote destination. This option only works with S3 remote backup destinations, because you would not want to allow a cluster node to delete backups from a node in another cluster. All incremental backups based on a deleted full backup will also be deleted at the remote destination. If unspecified, the --keep-remote option defaults to “all”. [OPTIONAL]
4	Specifies whether to “RETRY” or “SKIP” the scheduled backup shipping if it fails. If “SKIP” is specified, LINSTOR will ignore the failure and continue with the next scheduled backup shipping. If “RETRY” is specified, LINSTOR will wait 60 seconds and then try the backup shipping again. The LINSTOR schedule create command defaults to “SKIP” if no --on-failure option is given. [OPTIONAL]
5	The number of times to retry the backup shipping if a scheduled backup shipping fails and the --on-failure RETRY option has been given. Without this option, the LINSTOR controller will retry the scheduled backup shipping indefinitely, until it is successful. [OPTIONAL]
6	The name that you give the backup schedule so that you can reference it later with the schedule list, modify, delete, enable, or disable commands. [REQUIRED]
7	This cron schema describes how frequently LINSTOR creates snapshots and ships full backups.

If you specify an incremental cron schema that has overlap with the full cron schema that you specify, at the times when both types of backup shipping would occur simultaneously, LINSTOR will only make and ship a full backup. For example, if you specify that a full backup be made every three hours, and an incremental backup be made every hour, then every third hour, LINSTOR will only make and ship a full backup. For this reason, specifying the same cron schema for both your incremental and full backup shipping schedules would be useless, because incremental backups will never be made.

2.12.2. Modifying a Backup Shipping Schedule

You can modify a backup shipping schedule by using the LINSTOR client `schedule modify` command. The syntax for the command is the same as that for the `schedule create` command. The name that

you specify with the `schedule modify` command must be an already existing backup schedule. Any options to the command that you do not specify will retain their existing values. If you want to set the `keep-local` or `keep-remote` options back to their default values, you can set them to “all”. If you want to set the `max-retries` option to its default value, you can set it to “forever”.

2.12.3. Configuring the Number of Local Snapshots and Remote Backups to Keep

Your physical storage is not infinite and your remote storage has a cost, so you will likely want to set limits on the number of snapshots and backups you keep.

Both the `--keep-remote` and `--keep-local` options deserve special mention as they have implications beyond what may be obvious. Using these options, you specify how many snapshots or full backups should be kept, either on the local source or the remote destination.

Configuring the Keep-local Option

For example, if a `--keep-local=2` option is set, then the backup shipping schedule, on first run, will make a snapshot for a full backup. On the next scheduled full backup shipping, it will make a second snapshot for a full backup. On the next scheduled full backup shipping, it makes a third snapshot for a full backup. This time, however, after successful completion, LINSTOR deletes the first (oldest) full backup shipping snapshot. If snapshots were made for any incremental backups based on this full snapshot, they will also be deleted from the local source node. On the next successful full backup shipping, LINSTOR will delete the second full backup snapshot and any incremental snapshots based upon it, and so on, with each successive backup shipping.

If there are local snapshots remaining from failed shipments, these will be deleted first, even if they were created later.

If you have enabled a backup shipping schedule and then later manually delete a LINSTOR snapshot, LINSTOR may not be able to delete everything it was supposed to. For example, if you delete a full backup snapshot definition, on a later full backup scheduled shipping, there may be incremental snapshots based on the manually deleted full backup snapshot that will not be deleted.

Configuring the Keep-remote Option

As mentioned in the callouts for the example `linstor schedule create` command above, the `keep-remote` option only works for S3 remote destinations. Here is an example of how the option works. If a `--keep-remote=2` option is set, then the backup shipping schedule, on first run, will make a snapshot for a full backup and ship it to the remote destination. On the next scheduled full backup shipping, a second snapshot is made and a full backup shipped to the remote destination. On the next scheduled full backup shipping, a third snapshot is made and a full backup shipped to the remote destination. This time, additionally, after the third snapshot successfully ships, the first full

backup is deleted from the remote destination. If any incremental backups were scheduled and made between the full backups, any that were made from the first full backup would be deleted along with the full backup.

This option only deletes backups at the remote destination. It does not delete snapshots that the full backups were based upon at the local source node.

2.12.4. Listing a Backup Shipping Schedule

You can list your backup shipping schedules by using the `linstor schedule list` command.

For example:

```
# linstor schedule list
```

Name	Full	Incremental	KeepLocal	KeepRemote	OnFailure
my-bu-schedule	2 * * * *		3	2	SKIP

2.12.5. Deleting a Backup Shipping Schedule

The LINSTOR client `schedule delete` command completely deletes a backup shipping schedule LINSTOR object. The command's only argument is the schedule name that you want to delete. If the deleted schedule is currently creating or shipping a backup, the scheduled shipping process is stopped. Depending on at which point the process stops, a snapshot, or a backup, or both, might not be created and shipped.

This command does not affect previously created snapshots or successfully shipped backups. These will be retained until they are manually deleted.

2.12.6. Enabling Scheduled Backup Shipping

You can use the LINSTOR client `backup schedule enable` command to enable a previously created backup shipping schedule. The command has the following syntax:

```
# linstor backup schedule enable \  
  [--node source_node] \ (1)  
  [--rg resource_group_name | --rd resource_definition_name] \ (2)  
  remote_name \ (3)  
  schedule_name (4)
```

1	This is a special option that allows you to specify the controller node that will be used as a source for scheduled backup shipments, if possible. If you omit this option from the command, then LINSTOR will choose a source node at the time a scheduled shipping is made. [OPTIONAL]
2	You can set here either the resource group or the resource definition (but not both) that you want to enable the backup shipping schedule for. If you omit this option from the command, then the command enables scheduled backup shipping for all deployed LINSTOR resources that can make snapshots. [OPTIONAL]
3	The name of the remote destination that you want to ship backups to. [REQUIRED]
4	The name of a previously created backup shipping schedule. [REQUIRED]

2.12.7. Disabling a Backup Shipping Schedule

To disable a previously enabled backup shipping schedule, you use the LINSTOR client `backup schedule disable` command. The command has the following syntax:

```
# linstor backup schedule disable \
  [--rg resource_group_name | --rd resource_definition_name] \
  remote_name \ (3)
  schedule_name (4)
```

If you include the option specifying either a resource group or resource definition, as described in the `backup schedule enable` command example above, then you disable the schedule only for that resource group or resource definition.

For example, if you omitted specifying a resource group or resource definition in an earlier `backup schedule enable` command, LINSTOR would schedule backup shipping for all its deployed resources that can make snapshots. Your disable command would then only affect the resource group or resource definition that you specify with the command. The backup shipping schedule would still apply to any deployed LINSTOR resources besides the specified resource group or resource definition.

The same as for the `backup schedule enable` command, if you specify neither a resource group nor a resource definition, then LINSTOR disables the backup shipping schedule at the controller level for all deployed LINSTOR resources.

2.12.8. Deleting Aspects of a Backup Shipping Schedule

You can use the `linstor backup schedule delete` command to granularly delete either a specified resource definition or a resource group from a backup shipping schedule, without deleting the schedule itself. This command has the same syntax and arguments as the `backup schedule disable` command. If you specify neither a resource group nor a resource definition, the backup shipping schedule you specify will be deleted at the controller level.

It may be helpful to think about the `backup schedule delete` command as a way that you can **remove** a backup shipping schedule-remote pair from a specified LINSTOR object level, either a resource definition, a resource group, or at the controller level if neither is specified.

The `backup schedule delete` command does not affect previously created snapshots or successfully shipped backups. These will be retained until they are manually deleted, or until they are removed by the effects of a still applicable `keep-local` or `keep-remote` option.

You might want to use this command when you have disabled a backup schedule for multiple LINSTOR object levels and later want to affect a granular change, where a `backup schedule enable` command might have unintended consequences.

For example, consider a scenario where you have a backup schedule-remote pair that you enabled at a controller level. This controller has a resource group, `myresgroup` that has several resource definitions, `resdef1` through `resdef9`, under it. For maintenance reasons perhaps, you disable the schedule for two resource definitions, `resdef1` and `resdef2`. You then realize that further maintenance requires that you disable the backup shipping schedule at the resource group level, for your `myresgroup` resource group.

After completing some maintenance, you are able to enable the backup shipping schedule for `resdef3` through `resdef9`, but you are not yet ready to resume (enable) backup shipping for `resdef1` and `resdef2`. You can enable backup shipping for each resource definition individually, `resdef3` through `resdef9`, or you can use the `backup schedule delete` command to delete the backup shipping schedule from the resource group, `myresgroup`. If you use the `backup schedule delete` command, backups of `resdef3` through `resdef9` will ship again because the backup shipping schedule is enabled at the controller level, but `resdef1` and `resdef2` will not ship because the backup shipping schedule is still disabled for them at the resource definition level.

When you complete your maintenance and are again ready to ship backups for `resdef1` and `resdef2`, you can delete the backup shipping schedule for those two resource definitions to return to your starting state: backup shipping scheduled for all LINSTOR deployed resources at the controller level. To visualize this it may be helpful to refer to the decision tree diagram for how LINSTOR decides whether or not to ship a backup in the [How the LINSTOR Controller Determines Scheduled Backup Shipping](#) subsection.

In the example scenario above, you might have enabled backup shipping on the resource group, after completing some maintenance. In this case, backup shipping would resume for resource definitions `resdef3` through `resdef9` but continue not to ship for resource definitions `resdef1` and `resdef2` because backup shipping was still disabled for those resource

definitions. After you completed all maintenance, you could delete the backup shipping schedule on `resdef1` and `resdef2`. Then all of your resource definitions would be shipping backups, as they were prior to your maintenance, because the `schedule-remote` pair was enabled at the resource group level. However, this would remove your option to globally stop all scheduled shipping at some later point in time at the controller level because the enabled schedule at the resource group level would override any `schedule disable` command applied at the controller level.

2.12.9. Listing Backup Shipping Schedules by Resource

You can list backup schedules by resource, using the LINSTOR client `schedule list-by-resource` command. This command will show LINSTOR resources and how any backup shipping schedules apply and to which remotes they are being shipped. If resources are not being shipped then the command will show:

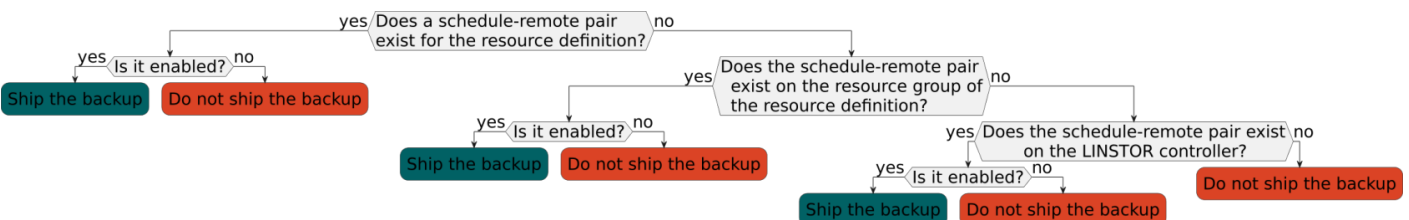
- Whether resources have no schedule-remote-pair entries (empty cells)
- Whether they have schedule-remote-pair entries but they are disabled (“disabled”)
- Whether they have no resources, so no backup shipments can be made, regardless of whether any schedule-remote-pair entries are enabled or not (“undeployed”)

If resources have schedule-remote-pairs and are being shipped, the command output will show when the last backup was shipped and when the next backup is scheduled to ship. It will also show whether the next and last backup shipments were full or incremental backups. Finally, the command will show when the next planned incremental (if any) and full backup shipping will occur.

You can use the `--active-only` flag with the `schedule list-by-resource` command to filter out all resources that are not being shipped.

2.12.10. How the LINSTOR Controller Determines Scheduled Backup Shipping

To determine if the LINSTOR Controller will ship a deployed LINSTOR resource with a certain backup schedule for a given remote destination, the LINSTOR Controller uses the following logic:



As the diagram shows, enabled or disabled backup shipping schedules have effect in the following order:

- ### 1. Resource definition level

2. Resource group level
3. Controller level

A backup shipping schedule-remote pair that is enabled or disabled at a preceding level will override the enabled or disabled status for the same schedule-remote pair at a later level.

2.12.11. Determining How Scheduled Backup Shipping Affects a Resource

To determine how a LINSTOR resource will be affected by scheduled backup shipping, you can use the LINSTOR client `schedule list-by-resource-details` command for a specified LINSTOR resource.

The command will output a table that shows on what LINSTOR object level a backup shipping schedule is either not set (empty cell), enabled, or disabled.

By using this command, you can determine on which level you need to make a change to enable, disable, or delete scheduled backup shipping for a resource.

Example output could look like this:

```
# linstor schedule list-by-resource-details my_linstor_resource_name
```

Remote	Schedule	Resource-Definition	Resource-Group	Controller
rem1	sch1	Disabled		Enabled
rem1	sch2		Enabled	
rem2	sch1	Enabled		
rem2	sch5		Enabled	
rem3	sch4		Disabled	Enabled

2.13. Adding and removing disks

LINSTOR can convert resources between diskless and having a disk. This is achieved with the resource toggle-disk command, which has syntax similar to resource create.

For instance, add a disk to the diskless resource backups on 'alpha':

```
# linstor resource toggle-disk alpha backups --storage-pool pool_ssd
```

Remove this disk again:

```
# linstor resource toggle-disk alpha backups --diskless
```

2.13.1. Migrating disks

In order to move a resource between nodes without reducing redundancy at any point, LINSTOR's disk migrate feature can be used. First create a diskless resource on the target node, and then add a disk using the --migrate-from option. This will wait until the data has been synced to the new disk and then remove the source disk.

For example, to migrate a resource backups from 'alpha' to 'bravo':

```
# linstor resource create bravo backups --drbd-diskless  
# linstor resource toggle-disk bravo backups --storage-pool pool_ssd --migrate-from alpha
```

2.14. DRBD Proxy with LINSTOR

LINSTOR expects DRBD Proxy to be running on the nodes which are involved in the relevant connections. It does not currently support connections via DRBD Proxy on a separate node.

Suppose our cluster consists of nodes 'alpha' and 'bravo' in a local network and 'charlie' at a remote site, with a resource definition named backups deployed to each of the nodes. Then DRBD Proxy can be enabled for the connections to 'charlie' as follows:

```
# linstor drbd-proxy enable alpha charlie backups
# linstor drbd-proxy enable bravo charlie backups
```

The DRBD Proxy configuration can be tailored with commands such as:

```
# linstor drbd-proxy options backups --memlimit 100000000
# linstor drbd-proxy compression zlib backups --level 9
```

LINSTOR does not automatically optimize the DRBD configuration for long-distance replication, so you will probably want to set some configuration options such as the protocol:

```
# linstor resource-connection drbd-options alpha charlie backups --protocol A
# linstor resource-connection drbd-options bravo charlie backups --protocol A
```

Please contact LINBIT for assistance optimizing your configuration.

2.14.1. Automatically enable DRBD Proxy

LINSTOR can also be configured to automatically enable the above mentioned Proxy connection between two nodes. For this automation, LINSTOR first needs to know on which site each node is.

```
# linstor node set-property alpha Site A
# linstor node set-property bravo Site A
# linstor node set-property charlie Site B
```

As the Site property might also be used for other site-based decisions in future features, the DrbdProxy/AutoEnable also has to be set to true:

```
# linstor controller set-property DrbdProxy/AutoEnable true
```

This property can also be set on node, resource-definition, resource and resource-connection level (from left to right in increasing priority, whereas the controller is the left-most, i.e. least prioritized

level)

Once this initialization steps are completed, every newly created resource will automatically check if it has to enable DRBD proxy to any of its peer-resources.

2.15. External database

It is possible to have LINSTOR working with an external database provider like PostgreSQL, MariaDB and since version 1.1.0 even ETCD key value store is supported.

To use an external database there are a few additional steps to configure. You have to create a DB/Schema and user to use for linstor, and configure this in the `/etc/linstor/linstor.toml`.

2.15.1. PostgreSQL

A sample PostgreSQL `linstor.toml` looks like this:

```
[db]
user = "linstor"
password = "linstor"
connection_url = "jdbc:postgresql://localhost/linstor"
```

2.15.2. MariaDB/MySQL

A sample MariaDB `linstor.toml` looks like this:

```
[db]
user = "linstor"
password = "linstor"
connection_url = "jdbc:mariadb://localhost/LINSTOR?createDatabaseIfNotExist=true"
```

The LINSTOR schema/database is created as LINSTOR so make sure the MariaDB connection string refers to the LINSTOR schema, as in the example above.

2.15.3. ETCD

ETCD is a distributed key-value store that makes it easy to keep your LINSTOR database distributed in a HA-setup. The ETCD driver is already included in the LINSTOR-controller package and only needs to be configured in the `linstor.toml`.

More information on how to install and configure ETCD can be found here: [ETCD docs](#)

And here is a sample `[db]` section from the `linstor.toml`:

[db]

only set user/password if you want to use authentication, only since LINSTOR 1.2.1

user = "linstor"

password = "linstor"

for etcd

do not set user field if no authentication required

connection_url = "etcd://etcdhost1:2379,etcdhost2:2379,etcdhost3:2379"

if you want to use TLS, only since LINSTOR 1.2.1

ca_certificate = "ca.pem"

client_certificate = "client.pem"

if you want to use client TLS authentication too, only since LINSTOR 1.2.1

client_key_pkcs8_pem = "client-key.pkcs8"

set client_key_password if private key has a password

client_key_password = "mysecret"

2.16. LINSTOR REST-API

To make LINSTOR's administrative tasks more accessible and also available for web-frontends a REST-API has been created. The REST-API is embedded in the linstor-controller and since LINSTOR 0.9.13 configured via the linstor.toml configuration file.

```
[http]
enabled = true
port = 3370
listen_addr = "127.0.0.1" # to disable remote access
```

If you want to use the REST-API the current documentation can be found on the following link:

<https://app.swaggerhub.com/apis-docs/Linstor/Linstor/>

2.16.1. LINSTOR REST-API HTTPS

The HTTP REST-API can also run secured by HTTPS and is highly recommended if you use any features that require authorization. To do so you have to create a java keystore file with a valid certificate that will be used to encrypt all HTTPS traffic.

Here is a simple example on how you can create a self signed certificate with the keytool that is included in the java runtime:

```
keytool -keyalg rsa -keysize 2048 -genkey -keystore ./keystore_linstor.jks\
-alias linstor_controller\
-dname "CN=localhost, OU=SecureUnit, O=ExampleOrg, L=Vienna, ST=Austria, C=AT"
```

keytool will ask for a password to secure the generated keystore file and is needed for the LINSTOR-controller configuration. In your linstor.toml file you have to add the following section:

```
[https]
keystore = "/path/to/keystore_linstor.jks"
keystore_password = "linstor"
```

Now (re)start the linstor-controller and the HTTPS REST-API should be available on port 3371.

More information on how to import other certificates can be found here:

<https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html>

When HTTPS is enabled, all requests to the HTTP /v1/ REST-API will be redirected to the HTTPS redirect.

LINSTOR REST-API HTTPS restricted client access

Client access can be restricted by using a SSL truststore on the Controller. Basically you create a certificate for your client and add it to your truststore and the client then uses this certificate for authentication.

First create a client certificate:

```
keytool -keyalg rsa -keysize 2048 -genkey -keystore client.jks\  
-storepass linstor -keypass linstor\  
-alias client1\  
-dname "CN=Client Cert, OU=client, O=Example, L=Vienna, ST=Austria, C=AT"
```

Then we import this certificate to our controller truststore:

```
keytool -importkeystore\  
-srcstorepass linstor -deststorepass linstor -keypass linstor\  
-srckeystore client.jks -destkeystore truststore_client.jks
```

And enable the truststore in the linstor.toml configuration file:

```
[https]  
keystore = "/path/to/keystore_linstor.jks"  
keystore_password = "linstor"  
truststore = "/path/to/truststore_client.jks"  
truststore_password = "linstor"
```

Now restart the Controller and it will no longer be possible to access the controller API without a correct certificate.

The LINSTOR client needs the certificate in PEM format, so before we can use it we have to convert the java keystore certificate to the PEM format.

```
# Convert to pkcs12  
keytool -importkeystore -srckeystore client.jks -destkeystore client.p12\  
-storepass linstor -keypass linstor\  
-srcalias client1 -srcstoretype jks -deststoretype pkcs12
```

```
# use openssl to convert to PEM  
openssl pkcs12 -in client.p12 -out client_with_pass.pem
```


To avoid entering the PEM file password all the time it might be convenient to remove the password.

```
openssl rsa -in client_with_pass.pem -out client1.pem  
openssl x509 -in client_with_pass.pem >> client1.pem
```

Now this PEM file can easily be used in the client:

```
linstor --certfile client1.pem node list
```

The `--certfile` parameter can also be added to the client configuration file, see Using [the LINSTOR client](#) for more details.

2.17. Logging

Linstor uses [SLF4J](#) with [Logback](#) as binding. This gives Linstor the possibility to distinguish between the log levels ERROR, WARN, INFO, DEBUG and TRACE (in order of increasing verbosity). In the current linstor version (1.1.2) the user has the following four methods to control the logging level, ordered by priority (first has highest priority):

1. TRACE mode can be enabled or disabled using the debug console:

```
Command ==> SetTrcMode MODE(enabled)
SetTrcMode      Set TRACE level logging mode
New TRACE level logging mode: ENABLED
```

2. When starting the controller or satellite a command line argument can be passed:

```
java ... com.linbit.linstor.core.Controller ... --log-level INFO
java ... com.linbit.linstor.core.Satellite ... --log-level INFO
```

3. The recommended place is the logging section in the configuration file. The default configuration file location is `/etc/linstor/linstor.toml` for the controller and `/etc/linstor/linstor_satellite.toml` for the satellite. Configure the logging level as follows:

```
[logging]
level="INFO"
```

4. As Linstor is using Logback as an implementation, `/usr/share/linstor-server/lib/logback.xml` can also be used. Currently only this approach supports different log levels for different components, like shown in the example below:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration scan="false" scanPeriod="60 seconds">
<!--
Values for scanPeriod can be specified in units of milliseconds, seconds, minutes or hours
https://logback.qos.ch/manual/configuration.html
-->
<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
<!-- encoders are assigned the type
      ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
<encoder>
  <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger - %msg%n</pattern>
```

```

</encoder>
</appender>
<appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${log.directory}/linstor-${log.module}.log</file>
  <append>true</append>
  <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
    <Pattern>%d{yyyy_MM_dd HH:mm:ss.SSS} [%thread] %-5level %logger - %msg%n</Pattern>
  </encoder>
  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
    <FileNamePattern>logs/linstor-${log.module}-%i.log.zip</FileNamePattern>
    <MinIndex>1</MinIndex>
    <MaxIndex>10</MaxIndex>
  </rollingPolicy>
  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
    <MaxFileSize>2MB</MaxFileSize>
  </triggeringPolicy>
</appender>
<logger name="LINSTOR/Controller" level="INFO" additivity="false">
  <appender-ref ref="STDOUT" />
  <!-- <appender-ref ref="FILE" /> -->
</logger>
<logger name="LINSTOR/Satellite" level="INFO" additivity="false">
  <appender-ref ref="STDOUT" />
  <!-- <appender-ref ref="FILE" /> -->
</logger>
<root level="WARN">
  <appender-ref ref="STDOUT" />
  <!-- <appender-ref ref="FILE" /> -->
</root>
</configuration>

```

See the [Logback Manual](#) to find more details about logback.xml.

When none of the configuration methods above is used Linstor will default to INFO log level.

2.18. Monitoring

Since LINSTOR 1.8.0, a [Prometheus](#) /metrics HTTP path is provided with LINSTOR and JVM specific exports.

The /metrics path also supports 3 GET arguments to reduce LINSTOR's reported data:

- resource
- storage_pools
- error_reports

These are all default true, to disabled e.g. error-report data:

http://localhost:3370/metrics?error_reports=false

2.18.1. Health check

The LINSTOR-Controller also provides a /health HTTP path that will simply return HTTP-Status 200 if the controller can access its database and all services are up and running. Otherwise it will return HTTP error status code 500 Internal Server Error.

2.19. Secure Satellite connections

It is possible to have the LINSTOR use SSL secure TCP connection between controller and satellite connections. Without going into further details on how java's SSL engine works we will give you command line snippets using the keytool from java's runtime environment on how to configure a 3 node setup using secure connections. The node setup looks like this:

Node alpha is the just the controller. Node bravo and node charlie are just satellites.

Here are the commands to generate such a keystore setup, values should of course be edited for your environment.

```
# create directories to hold the key files
```

```
mkdir -p /tmp/linstor-ssl
```

```
cd /tmp/linstor-ssl
```

```
mkdir alpha bravo charlie
```

```
# create private keys for all nodes
```

```
keytool -keyalg rsa -keysize 2048 -genkey -keystore alpha/keystore.jks\
```

```
-storepass linstor -keypass linstor\
```

```
-alias alpha\
```

```
-dname "CN=Max Mustermann, OU=alpha, O=Example, L=Vienna, ST=Austria, C=AT"
```

```
keytool -keyalg rsa -keysize 2048 -genkey -keystore bravo/keystore.jks\
```

```
-storepass linstor -keypass linstor\
```

```
-alias bravo\
```

```
-dname "CN=Max Mustermann, OU=bravo, O=Example, L=Vienna, ST=Austria, C=AT"
```

```
keytool -keyalg rsa -keysize 2048 -genkey -keystore charlie/keystore.jks\
```

```
-storepass linstor -keypass linstor\
```

```
-alias charlie\
```

```
-dname "CN=Max Mustermann, OU=charlie, O=Example, L=Vienna, ST=Austria, C=AT"
```

```
# import truststore certificates for alpha (needs all satellite certificates)
```

```
keytool -importkeystore\
```

```
-srcstorepass linstor -deststorepass linstor -keypass linstor\
```

```
-srckeystore bravo/keystore.jks -destkeystore alpha/certificates.jks
```

```
keytool -importkeystore\
```

```
-srcstorepass linstor -deststorepass linstor -keypass linstor\  
-srckeystore charlie/keystore.jks -destkeystore alpha/certificates.jks
```

```
# import controller certificate into satellite truststores  
keytool -importkeystore\  
-srcstorepass linstor -deststorepass linstor -keypass linstor\  
-srckeystore alpha/keystore.jks -destkeystore bravo/certificates.jks  
  
keytool -importkeystore\  
-srcstorepass linstor -deststorepass linstor -keypass linstor\  
-srckeystore alpha/keystore.jks -destkeystore charlie/certificates.jks
```

```
# now copy the keystore files to their host destinations  
ssh root@alpha mkdir /etc/linstor/ssl  
scp alpha/* root@alpha:/etc/linstor/ssl/  
ssh root@bravo mkdir /etc/linstor/ssl  
scp bravo/* root@bravo:/etc/linstor/ssl/  
ssh root@charlie mkdir /etc/linstor/ssl  
scp charlie/* root@charlie:/etc/linstor/ssl/
```

```
# generate the satellite ssl config entry  
echo '[netcom]  
  type="ssl"  
  port=3367  
  server_certificate="ssl/keystore.jks"  
  trusted_certificates="ssl/certificates.jks"  
  key_password="linstor"  
  keystore_password="linstor"  
  truststore_password="linstor"  
  ssl_protocol="TLSv1.2"  
' | ssh root@bravo "cat > /etc/linstor/linstor_satellite.toml"  
  
echo '[netcom]  
  type="ssl"  
  port=3367  
  server_certificate="ssl/keystore.jks"  
  trusted_certificates="ssl/certificates.jks"  
  key_password="linstor"
```

```
keystore_password="linstor"  
truststore_password="linstor"  
ssl_protocol="TLSv1.2"  
' | ssh root@charlie "cat > /etc/linstor/linstor_satellite.toml"
```

Now just start controller and satellites and add the nodes with --communication-type SSL.

2.20. Automatism for DRBD-Resources

2.20.1. AutoQuorum Policies

LINSTOR automatically configures quorum policies on resources **when quorum is achievable**. This means, whenever you have at least two diskful and one or more diskless resource assignments, or three or more diskful resource assignments, LINSTOR will enable quorum policies for your resources automatically.

Inversely, LINSTOR will automatically disable quorum policies whenever there are less than the minimum required resource assignments to achieve quorum.

This is controlled via the, `DrbdOptions/auto-quorum`, property which can be applied to the `linstor-controller`, `resource-group`, and `resource-definition`. Accepted values for the `DrbdOptions/auto-quorum` property are `disabled`, `suspend-io`, and `io-error`.

Setting the `DrbdOptions/auto-quorum` property to `disabled` will allow you to manually, or more granularly, control the quorum policies of your resources should you so desire.

The default policies for `DrbdOptions/auto-quorum` are `quorum majority`, and `on-no-quorum io-error`. For more information on DRBD's quorum features and their behavior, please refer to the quorum section of the DRBD user's guide.

The `DrbdOptions/auto-quorum` policies will override any manually configured properties if `DrbdOptions/auto-quorum` is not disabled.

For example, to manually set the quorum policies of a resource-group named `my_ssd_group`, you would use the following commands:

```
# linstor resource-group set-property my_ssd_group DrbdOptions/auto-quorum disabled
# linstor resource-group set-property my_ssd_group DrbdOptions/Resource/quorum majority
# linstor resource-group set-property my_ssd_group DrbdOptions/Resource/on-no-quorum suspend-io
```

You may wish to disable DRBD's quorum features completely. To do that, you would need to first disable `DrbdOptions/auto-quorum` on the appropriate LINSTOR object, and then set the DRBD quorum features accordingly. For example, use the following commands to disable quorum entirely on the `my_ssd_group` resource-group:

```
# linstor resource-group set-property my_ssd_group DrbdOptions/auto-quorum disabled
# linstor resource-group set-property my_ssd_group DrbdOptions/Resource/quorum off
```


Setting `DrbdOptions/Resource/on-no-quorum` to an empty value in the commands above deletes the property from the object entirely.

2.20.2. Auto-Evict

If a satellite is offline for a prolonged period of time, LINSTOR can be configured to declare that node as evicted. This triggers an automated reassignment of the affected DRBD-resources to other nodes to ensure a minimum replica count is kept.

This feature uses the following properties to adapt the behaviour.

- `DrbdOptions/AutoEvictMinReplicaCount` sets the number of replicas that should always be present. You can set this property on the controller to change a global default, or on a specific resource-definition or resource-group to change it only for that resource-definition or resource-group. If this property is left empty, the place-count set for the auto-placer of the corresponding resource-group will be used.
- `DrbdOptions/AutoEvictAfterTime` describes how long a node can be offline in minutes before the eviction is triggered. You can set this property on the controller to change a global default, or on a single node to give it a different behavior. The default value for this property is 60 minutes.
- `DrbdOptions/AutoEvictMaxDisconnectedNodes` sets the percentage of nodes that can be not reachable (for whatever reason) at the same time. If more than the given percent of nodes are offline at the same time, the auto-evict will not be triggered for any node, since in this case LINSTOR assumes connection problems from the controller. This property can only be set for the controller, and only accepts a value between 0 and 100. The default value is 34. If you wish to turn the auto-evict-feature off, simply set this property to 0. If you want to always trigger the auto-evict, regardless of how many satellites are unreachable, set it to 100.
- `DrbdOptions/AutoEvictAllowEviction` is an additional property that can stop a node from being evicted. This can be useful for various cases, for example if you need to shut down a node for maintenance. You can set this property on the controller to change a global default, or on a single node to give it a different behavior. It accepts true and false as values and per default is set to true on the controller. You can use this property to turn the auto-evict feature off by setting it to false on the controller, although this might not work completely if you already set different values for individual nodes, since those values take precedence over the global default.

After the linstor-controller loses the connection to a satellite, aside from trying to reconnect, it starts a timer for that satellite. As soon as that timer exceeds `DrbdOptions/AutoEvictAfterTime` and all of the DRBD-connections to the DRBD-resources on that satellite are broken, the controller will check whether or not `DrbdOptions/AutoEvictMaxDisconnectedNodes` has been met. If it hasn't, and `DrbdOptions/AutoEvictAllowEviction` is true for the node in question, the satellite will be marked as

EVICTED. At the same time, the controller will check for every DRBD-resource whether the number of resources is still above `DrbdOptions/AutoEvictMinReplicaCount`. If it is, the resource in question will be marked as DELETED. If it isn't, an auto-place with the settings from the corresponding resource-group will be started. Should the auto-place fail, the controller will try again later when changes that might allow a different result, such as adding a new node, have happened. Resources where an auto-place is necessary will only be marked as DELETED if the corresponding auto-place was successful.

The evicted satellite itself will not be able to reestablish connection with the controller. Even if the node is up and running, a manual reconnect will fail. It is also not possible to delete the satellite, even if it is working as it should be. The satellite can, however, be restored. This will remove the EVICTED-flag from the satellite and allow you to use it again. Previously configured network interfaces, storage pools, properties and similar entities as well as non-DRBD-related resources and resources that could not be autoplace somewhere else will still be on the satellite. To restore a satellite, use

```
# linstor node restore [nodename]
```

Should you wish to instead throw everything that once was on that node, including the node itself, away, you need to use the node lost command instead.

2.21. QoS Settings

2.21.1. Sysfs

LINSTOR is able to set the following Sysfs settings:

SysFs	Linstor property
/sys/fs/cgroup/blkio/blkio.throttle.read_bps_device	sys/fs/blkio_throttle_read
/sys/fs/cgroup/blkio/blkio.throttle.write_bps_device	sys/fs/blkio_throttle_write
/sys/fs/cgroup/blkio/blkio.throttle.read_iops_device	sys/fs/blkio_throttle_read_iops
/sys/fs/cgroup/blkio/blkio.throttle.write_iops_device	sys/fs/blkio_throttle_write_iops

If a LINSTOR volume is composed of multiple “stacked” volume (for example DRBD with external metadata will have 3 devices: backing (storage) device, metadata device and the resulting DRBD device), setting a `sys/fs/*` property for a Volume, only the bottom-most local “data”-device will receive the corresponding `/sys/fs/cgroup/...` setting. That means, in case of the example above only the backing device will receive the setting. In case a resource-definition has an `nvme-target` as well as an `nvme-initiator` resource, both bottom-most devices of each node will receive the setting. In case of the target the bottom-most device will be the volume of LVM or ZFS, whereas in case of the initiator the bottom-most device will be the connected `nvme-device`, regardless which other layers are stacked on top of that.

2.22. Getting help

2.22.1. From the command line

A quick way to list available commands on the command line is to type `linstor`.

Further information on sub-commands (e.g., `list-nodes`) can be retrieved in two ways:

```
# linstor node list -h
# linstor help node list
```

Using the ‘help’ sub-command is especially helpful when LINSTOR is executed in interactive mode (`linstor interactive`).

One of the most helpful features of LINSTOR is its rich tab-completion, which can be used to complete basically every object LINSTOR knows about (e.g., node names, IP addresses, resource names, ...). In the following examples, we show some possible completions, and their results:

```
# linstor node create alpha 1<tab> # completes the IP address if hostname can be resolved
# linstor resource create b<tab> c<tab> # linstor assign-resource backups charlie
```

If tab-completion does not work out of the box, please try to source the appropriate file:

```
# source /etc/bash_completion.d/linstor # or
# source /usr/share/bash_completion/completions/linstor
```

For zsh shell users `linstor-client` can generate a zsh compilation file, that has basic support for command and argument completion.

```
# linstor gen-zsh-completer > /usr/share/zsh/functions/Completion/Linux/_linstor
```

2.22.2. SOS-Report

If something goes wrong and you need help finding the cause of the issue, you can use

```
# linstor sos-report create
```

The command above will create a new sos-report in `/var/log/linstor/controller/` on the controller node. Alternatively you can use

```
# linstor sos-report download
```

which will create a new sos-report and additionally downloads that report to the local machine into your current working directory.

This sos-report contains logs and useful debug-information from several sources (Linstor-logs, dmesg, versions of external tools used by Linstor, ip a, database dump and many more). These information are stored for each node in plaintext in the resulting .tar.gz file.

2.22.3. From the community

For help from the community please subscribe to our mailing list located here:

<https://lists.linbit.com/listinfo/drbd-user>

2.22.4. GitHub

To file bug or feature request please check out our GitHub page <https://github.com/linbit>

2.22.5. Paid support and development

Alternatively, if you wish to purchase remote installation services, 24/7 support, access to certified repositories, or feature development please contact us: +1-877-454-6248 (1-877-4LINBIT) ,

International: +43-1-8178292-0 | sales@linbit.com