CEPH-DISK

DEVICE-MAPPER CRYPT

SETTINGS

`osd_dmcrypt_type`

- **Description:** this option specifies the mode in which `cryptsetup` works. It can be `luks` or `plain`. It kicks in only if the `--dmcrypt` option is passed to `ceph-disk`. See also cryptsetup document for more details.
- **Type:** String
- **Default:** `luks`

`osd_dmcrypt_key_size`

- **Description:** the size of the random string in bytes used as the LUKS key. The string is read from `/dev/urandom` and then encoded using base64. It will be stored with the key of `dm-crypt/osd/$uuid/luks` using config-key.
- **Type:** String
- **Default:** 1024 if `osd_dmcrypt_type` is `luks`, 256 otherwise.

LOCKBOX

`ceph-disk` supports dmcrypt (device-mapper crypt). If dmcrypt is enabled, the partitions will be encrypted using this machinary. For each OSD device, a lockbox is introduced for holding the information regarding how the dmcrypt key is stored. To prepare a lockbox, `ceph-disk`

1. creates a dedicated lockbox partition on device, and
2. populates it with a tiny filesystem, then
3. automounts it at `/var/lib/ceph/osd-lockbox/$uuid`, read-only. where the `uuid` is the lockbox's uuid.

under which, settings are stored using plain files:

- key-management-mode: `ceph-mon v1`
- osd-uuid: the OSD's uuid
- ceph_fsid: the fsid of the cluster
- keyring: the lockbox's allowing one to fetch the LUKS key
- block_uuid: the partition uuid for the block device
- journal_uuid: the partition uuid for the journal device
- block.db_uuid: the partition uuid for the block.db device
- block.wal_uuid: the partition uuid for the block.wal device
- magic: a magic string indicating that this partition is a lockbox. It's not used currently.
- `${space_uuid}`: symbolic links named after the uuid of space partitions pointing to `/var/lib/ceph/osd-lockbox/$uuid`. in the case of FileStore, the space partitions are `data` and `journal` partitions, for BlueStore, they are `data`, `block.db` and `block.wal`.

Currently, `ceph-mon v1` is the only supported key-management-mode. In that case, the LUKS key is stored using the config-key in the monitor store with the key of `dm-crypt/osd/$uuid/luks`.

PARTITIONS

`ceph-disk` creates partitions for preparing a device for OSD deployment. Their partition numbers are hardcoded. For instance, data partition's partition number is always *1* :

1. data partition
2. journal partition, if co-located with data
3. block.db for BlueStore, if co-located with data
4. block.wal for BlueStore, if co-located with data
5. lockbox

PREPARE CLASS HIERARCHY

The `ceph-disk` prepare class hierarchy can be challenging to read and this guide is designed to explain how it is structured.

The `Prepare` class roughly replaces the `prepare_main` function but also handles the `prepare` subcommand argument parsing. It creates the data and journal objects and delegate the actual work to them via the `prepare()` method.

The `Prepare` class assumes that preparing an OSD consists of the following phases:

- optionally prepare auxiliary devices, such as the journal
- prepare a data directory or device
- populate the data directory with fsid etc. and optionally symbolic links to the auxiliary devices

The `PrepareFilestore` class is derived from `Prepare` and implements the current model where there only is one auxiliary device, the journal. It utilizes `PrepareJournal` and `PrepareFilestoredata` for preparing its journal and its data directory respectively. The latter is a derived class of `PrepareData`.

The `PrepareJournal` class implements the *journal* functions and is based on a generic class `PrepareSpace`, which handles the allocation of an auxiliary device. The only journal specific feature is left to the `PrepareJournal` class: querying the OSD to figure out if a journal is wanted or not.

The OSD data directory is prepared via the `PrepareData` class. It creates a file system if necessary (i.e. if a device) and populates the data directory. Further preparation is then delegated to the auxiliary devices (i.e. adding a symlink to the device for a journal).

There was some code paths related dmcrypt / multipath devices in the prepare functions, although it is orthogonal. A class tree for `Devices` was created to isolate that.

Although that was the primary reason for adding a new class tree, two other aspects have also been moved there: `ptypes` (i.e. partition types) and partition creation. The `ptypes` are organized into a data structure with a few helpers in the hope that it will be easier to maintain. All references to the `*_UUID` variables have been updated.

The creation of a partition is delegated to sgdisk and a wrapper helps reduce the code redundancy.

The `ptype` of a given partition depends on the type of the device (is it dmcrypt'ed or a multipath device ?). It is best implemented by derivation so the prepare function does not need to be concerned about how the `ptype` of a partition is determined.

Many functions could be refactored into a `Device` class and its derivatives, but that was not done to minimize the size of the refactor.

- `Device` knows how to create a partition and figure out the `ptype` `tobe`
- `DevicePartition` a regular device partition
- `DevicePartitionMultipath` a partition of a multipath device
- `DevicePartitionCrypt` base class for luks/plain dmcrypt, can map/unmap
- `DevicePartitionCryptPlain` knows how to setup dmcrypt plain
- `DevicePartitionCryptLuks` knows how to setup dmcrypt luks

The `CryptHelpers` class is introduced to factorize the code snippets that were duplicated in various places but that do not really belong because they are convenience wrappers to figure out:

- if dcmrypt should be used
- the keysize
- the dmcrypt type (plain or luks)

STATE TRANSITION OF PARTITIONS