RuBackup

Система резервного копирования
и восстановления данных

СЦЕНАРИИ РАБОТЫ

ВЕРСИЯ 2.9.0.0.0

Содержание

В разделе описаны отдельные вопросы, с которыми встречаются пользователи СРК.

Резервное копирование и восстановление метаданных блочного дедуплицированного пула

Метаданные блочного дедуплицированного пула хранятся в служебной БД СРК RuBackup в таблицах:

- `pool_list`,
- `pool_block_device_extention`,
- `storage_block_devices`,
- `deduplicated_block_device_<signature>`.

Резервное копирование

Резервное копирование метаданных блочного дедуплицированного пула возможно с помощью скрипта `script_block_device_metadata.sh` и с помощью утилиты `pg_dump`.

Резервное копирование с помощью скрипта

Выполните резервное копирование таблиц с метаданными блочного дедуплицированного пула с помощью скрипта `script_block_device_metadata.sh`.

1. Включите сервисный режим с помощью утилиты `rb_global_config`.

   ```
   rb_global_config -s service_mode=yes
   ```

2. Выдайте права на выполнение скрипта `script_block_device_metadata.sh`.

   ```
   chmod 700 script_block_device_metadata.sh
   ```

3. Укажите в скрипте `script_block_device_metadata.sh` значения обязательных параметров:
   - `HOST` — адрес хоста с базой данных,
   - `DBNAME` — имя базы данных,
   - `USER` — имя пользователя базы данных,
   - `PASS` — пароль пользователя базы данных,
   - `BACKUP_FILENAME` — имя файла резервной копии выбранных таблиц.
4. Запустите скрипт `script_block_device_metadata.sh` с параметром `dump`.

```
./script_block_device_metadata.sh dump
```

В текущем каталоге будет создан файл `rb_block_device_metadata_backup.sql` с таблицами.

5. Выключите сервисный режим с помощью утилиты `rb_global_config`.

```
rb_global_config -s service_mode=no
```

Сохраните файл `rb_block_device_metadata_backup.sql` в надежном месте.

Резервное копирование с помощью утилиты

Выполните резервное копирование таблиц с метаданными блочного дедуплицированного пула с помощью утилиты `pg_dump`.

1. Включите сервисный режим с помощью утилиты `rb_global_config`.

```
rb_global_config -s service_mode=yes
```

2. Передайте утилите `pg_dump` названия таблиц, в которых хранятся метаданные.

Пример 1. Резервное копирование метаданных из таблиц в файл `rb_block_device_metadata_backup.sql`

```
pg_dump -h localhost \
    -d rubackup \ (1)
    -U rubackup \ (2)
    -t pool_list \ (3)
    -t pool_block_device_extention \
    -t storage_block_devices \
    -t deduplicated_block_device_<signature> \
    > rb_block_device_metadata_backup.sql (4)
```

(1) Имя служебной БД.

(2) Имя пользователя служебной БД.

(3) Названия таблиц.

Для таблицы `deduplicated_block_device_<signature>` дополнительно укажите `signature` — уникальную подпись для каждого блочного устройства (например, `deduplicated_block_device_7fb0b3bac69d4fa0`).

Подпись блочного устройства можно просмотреть с помощью утилиты `rb_block_devices` (`rb_block_devices -v`) или с помощью приложений Tucana или Менеджер администратора RuBackup (RBM) в разделе **Хранилища** → **Блочные устройства** в столбце **Подпись**.

4. Имя файла.

В текущем каталоге будет создан файл `rb_block_device_metadata_backup.sql` с таблицами.

3. Выключите сервисный режим с помощью утилиты `rb_global_config`.

```
rb_global_config -s service_mode=no
```

Сохраните файл `rb_block_device_metadata_backup.sql` в надежном месте.

Восстановление

Восстановление метаданных блочного дедуплицированного пула возможно с помощью скрипта `script_block_device_metadata.sh` и с помощью утилиты `psql`.

Восстановление с помощью скрипта

Выполните восстановление таблиц с метаданными блочного дедуплицированного пула с помощью скрипта `script_block_device_metadata.sh`.

1. Включите сервисный режим с помощью утилиты `rb_global_config`.

```
rb_global_config -s service_mode=yes
```

2. Восстановите таблицы из файла `rb_block_device_metadata_backup.sql`.

```
./script_block_device_metadata.sh restore
```

3. Выключите сервисный режим с помощью утилиты `rb_global_config`.

```
rb_global_config -s service_mode=no
```

4. Перезапустите сервер СРК RuBackup.

```
systemctl restart rubackup_server
```

Метаданные блочного дедуплицированного пула будут восстановлены.

Восстановление с помощью утилиты

Выполните восстановление таблиц с метаданными блочного дедуплицированного пула с помощью утилиты `psql`.

1. Включите сервисный режим с помощью утилиты `rb_global_config`.

```
rb_global_config -s service_mode=yes
```

2. Восстановите таблицы из файла `rb_block_device_metadata_backup.sql`.

```
psql -h localhost -d rubackup -U rubackup -f
rb_block_device_metadata_backup.sql
```

3. Выключите сервисный режим с помощью утилиты `rb_global_config`.

```
rb_global_config -s service_mode=no
```

4. Перезапустите сервер СРК RuBackup.

```
systemctl restart rubackup_server
```

Метаданные блочного дедуплицированного пула будут восстановлены.

Листинг скрипта `script_block_device_metadata.sh`

```
#!/bin/bash

set -e  # Выход при любой ошибке

# Параметры подключения к базе данных
HOST="localhost"
DBNAME="rubackup"
USER="rubackup"
PASS="12345"
BACKUP_FILENAME="rb_block_device_metadata_backup.sql"
PORT=5432

# Список таблиц для резервного копирования
```

```
TABLE_LIST="-t pool_list -t pool_block_device_extention -t
storage_block_devices"
# Запрос на получение дополнительных таблиц для резервного копирования
QUERY_TO_GET_ADDITIONAL_TABLES="SELECT DISTINCT table_name
                                FROM information_schema.tables
                                WHERE table_name LIKE 'deduplicated_%'"


if [ "$#" -eq 1 ]; then
    if [ "$1" = "dump" ]; then
        echo "RuBackup script handler saving Dedup pool metadata started"

        ADDITIONAL_TABLES=$(PGPASSWORD="$PASS" psql \
            -h "$HOST" \
            -d "$DBNAME" -U "$USER" \
            -p "$PORT" -qt -c "$QUERY_TO_GET_ADDITIONAL_TABLES")

        for table in $ADDITIONAL_TABLES; do
            TABLE_LIST="$TABLE_LIST -t $table"
        done

        # Создание резервной копии всех таблиц в одном файле
        PGPASSWORD="$PASS" pg_dump \
            -h "$HOST" -d "$DBNAME" -U "$USER" $TABLE_LIST \
            -p "$PORT" > "$BACKUP_FILENAME"

        echo "A backup copy of the table is saved in a file $BACKUP_FILENAME"
        exit 0
    fi

    if [ "$1" = "restore" ]; then
        echo "RuBackup script handler restores Dedup pool metadata started"
        # Восстановление
        PGPASSWORD="$PASS" psql \
            -h "$HOST" -d "$DBNAME" -U "$USER" -f "$BACKUP_FILENAME" \
            -p "$PORT"
        echo "RuBackup script handler restores Dedup pool metadata finished"
        exit 0
    fi

    echo "Incorrect argument. Type 'dump' or 'restore'"
    exit 1
fi

echo "Argument required. Type 'dump' or 'restore'"
```

```
exit 1
```

Настройка хранилища резервных копий

Если в процессе конфигурирования клиента РК или сервера СРК RuBackup при помощи утилиты `rb_init` или `rb_init_gui` не был назначен каталог для хранения резервных копий для пула `Default`, то после конфигурирования сервера RuBackup в журнальном файле `/opt/rubackup/log/RuBackup.log` появятся записи о том, что в пуле `Default` нет ни одной файловой системы для хранения резервных копий:

```
Thu Sep 19 12:40:30 2019: Warning: Pool: Default has no any file system
```

Необходимо назначить для пула `Default` хотя бы один каталог для хранения резервных копий. Это можно сделать при помощи утилиты командной строки или Менеджера администратора RuBackup (RBM):

1. Настройка хранилища с помощью `rb_local_filesystem`:

   Пользователи, от имени которых будет осуществляться запуск утилит командной строки RuBackup, должны входить в группу `rubackup`. Чтобы добавить пользователей в группу, внесите изменения в файл `/etc/group`.

   Чтобы назначить локальный каталог в качестве хранилища резервных копий, следует выполнить команду:

   ```
   rb_local_filesystems -a /rubackup1 -p 1
   ```

   В этом примере в качестве хранилища добавляется каталог `/rubackup1`.

Настройка хранилища с помощью RBM производится, если хранилища не настроены утилитой `rb_init` в процессе первоначальной настройки.

Порядок настройки хранилища изложен в документе Менеджер администратора RuBackup (RBM).

Модуль ядра Linux `dattobd`

`dattobd` — это модуль ядра Linux, который используется для создания снимков блочных устройств.

Модуль применяется в СРК RuBackup для резервного копирования некоторых ресурсов без остановки их работы.

В Linux существуют встроенные инструменты для создания мгновенных копий (снимков) файловой системы, из которых наиболее известен LVM. Однако, у них есть ограничения, которые делают их неудобными в работе с постоянно работающими серверами. Резервирование тома «на горячую» требует отмонтировать том, сделать его снимок, примонтировать снимок и отправить том на резервное хранение. Промышленный сервер редко может быть отключен на это время.

Блочный драйвер Datto (`dattobd`) предоставляет фунциональность, похожую на теневое копирование тома (Volume Shadow Copy, VSS) в Windows и позволяет делать мгновенные снимки файловых систем. Драйвер `dattobd` может быть установлен без перезагрузки машины. `dattobd` создает снимок любого блочного устройства, после чего отслеживает инкрементальные изменения на блочном устройстве и обновляет его резервные копии, копируя только измененные блоки.

`dattobd` работает на уровне слоя блоков, и поддерживает большинство актуальных файловых систем (ext2, ext3, ext4 и xfs). Файловые системы с собственной реализацией управления блоками (ZFS, BTRFS) не поддерживаются.

Установка

Для установки модуля ядра Linux `dattobd` на ОС Astra Linux 1.7 или Astra Linux 1.8:

1. Добавьте GPG-ключ репозитория Datto:

```
sudo apt-key adv --fetch-keys https://cpkg.datto.com/DATTO-PKGS-GPG-KEY
```

2. Добавьте репозиторий Datto:

Пример 2. Добавление репозитория для ОС Astra Linux 1.7

```
echo "deb [arch=amd64] https://cpkg.datto.com/datto-deb/public/buster
buster main" | sudo tee /etc/apt/sources.list.d/datto-linux-agent.list
```

Пример 3. Добавление репозитория для ОС Astra Linux 1.8

```
echo "deb [arch=amd64] https://cpkg.datto.com/datto-deb/public/bookworm
bookworm main" | sudo tee /etc/apt/sources.list.d/datto-linux-
agent.list
```

3. Обновите список пакетов:

```
sudo apt update
```

4. Установите пакеты:

```
sudo apt install dattobd-dkms dattobd-utils
```

5. Загрузите модуль в ядро:

```
sudo modprobe dattobd
```

Дедупликация

Система резервного копирования *RuBackup* позволяет использовать режим дедупликации при создании резервных копий данных.

В режиме дедупликации данные, которые должны попасть в резервную копию, разделяются на блоки равного размера, и для каждого блока вычисляется хеш-сумма по алгоритму `sha1`, `sha2`, `blake2b`, `skein` или `streebog`. Перед выполнением резервного копирования сервер передаёт клиенту хеш-таблицу блоков, уже расположенных в дедуплицированном хранилище и которые с высокой степенью вероятности могут содержаться в источнике данных, резервное копирование которых будет выполняться. Серверу передаются только уникальные блоки резервной копии, которые размещаются в дедуплицированном хранилище резервных копий, представляющее собой блочное устройство в операционной системе (это может быть одиночный диск, `RAID` массив или `LUN` система хранения данных).

Таким образом, при первом резервном копировании источника данных серверу резервного копирования будет передан полный уникальный набор блоков. При повторном резервном копировании будут переданы только изменившиеся блоки данных. Это позволяет уменьшить окно резервного копирования, снизить нагрузку на сеть передачи данных и сэкономить место в хранилище резервных копий.

При восстановлении сервер передаёт клиенту метафайл, содержащий всю необходимую информацию о резервной копии и целевом ресурсе, который требует восстановления. Если восстановление информации происходит непосредственно в источник данных, где были утеряны или изменены какие-либо блоки данных, и требуется восстановить целостность источника данных, то сервер передаст клиенту только те блоки данных, которые были изменены и требуют восстановления. Это позволяет значительно уменьшить время восстановления.

Система резервного копирования *RuBackup* позволяет объединять дедуплицированные блочные устройства в пулы типа **Блочное устройство**. Любой сервер в серверной группировке *RuBackup* может управлять несколькими пулами типа **Блочное устройство**. Это может быть полезно для использования пула только для определённых данных. Например, вы можете использовать один пул для хранения резервных копий виртуальных машин с гостевой операционной системой *MS Windows*, и другой пул для резервных копий ВМ с ОС *Astra Linux*. Параметры пула определяют размер блока дедупликации, алгоритм хеш-функции длину хеша.

Принципы дедупликации

При выполнении дедупликации происходит вычисление хеша для всех блоков данных, которые должны попасть в резервную копию. Хеш-алгоритмы, поддерживаемые *RuBackup*, приведены в таблице.

Таблица 1. Алгоритмы хеш-функций, поддерживаемые RuBackup

Алгоритм	Длина хэш, бит	Ссылка на описание
sha1	160	https://en.wikipedia.org/wiki/SHA-1
sha2	256, 512	https://en.wikipedia.org/wiki/SHA-2
skein	256, 512	https://en.wikipedia.org/wiki/Skein_%28hash_function%29
blake2b	256, 512	https://en.wikipedia.org/wiki/BLAKE_%28hash_function%29#BLAKE2
streebog	256, 512	https://en.wikipedia.org/wiki/Streebog

Вы можете определить параметры дедупликации при создании пула типа **Блочное устройство**. К ним относятся:

- Размер блока дедупликации (от `16 КБ` до `1 МБ`),
- Хеш-алгоритм,
- Длина хеш (где поддерживается).

Следует учитывать, что чем больше длина хеш-функции и чем меньше размер блока дедупликации, тем больше процессорных ресурсов и времени будет затрачено на выполнение процесса дедупликации. Но чем меньше длина хеш-функции, тем больше вероятность возникновения коллизии. И чем меньше размер блока дедупликации, тем более эффективен процесс дедупликации, т.к. вероятность нахождения одинаковых блоков возрастает.

Использование дедупликации целесообразно для тех источников данных, которые могут содержать в себе повторяющиеся блоки данных. Это файловые системы, блочные устройства (например, тома `LVM`), виртуальные машины и т.п. Некоторые источники данных в ходе своего функционирования могут значительно изменить своё содержимое, например, СУБД после переиндексации таблиц. Использование дедупликации для таких ресурсов может быть значительно менее эффективно.

Создание резервной копии

Система осуществляет создание резервной копии с применением дедупликации следующим образом:

1. Сервер резервного копирования:

   Запускает задачу резервного копирования, принимает от клиента дедуплицированные данные, размещает их в соответствующее хранилище и создаёт необходимые записи в базе данных

2. Клиент резервного копирования:

   Запускает соответствующий модуль и ожидает передачу дедуплицированных

блоков от утилиты `rbfd`.

3. Модуль *RuBackup*:

   Подготавливает источник данных к резервному копированию и запускает утилиту `rbfd`.

4. Утилита `rbfd`:

   Выполняет дедупликацию источника данных и передаёт дедуплицированные блоки клиенту резервного копирования.

Восстановление резервной копии

Система осуществляет восстановление резервной копии, созданной с применением дедупликации, следующим образом:

1. Сервер резервного копирования:

   Передаёт клиенту необходимые для восстановления блоки данных.

2. Клиент резервного копирования:

   Запускает соответствующий модуль и принимает блоки данных от сервера

3. Модуль *RuBackup*:

   Запускает утилиту `rbfd` и, после получения всех необходимых данных, при необходимости, развёртывает резервную копию в информационной системе.

4. Утилита `rbfd`:

   Выполняет сборку данных резервной копии из дедуплицированных блоков.

Настройка

Настройка дедупликации включает в себя следующие действия:

1. На сервере *RuBackup* выделить блочное устройство для хранения.
2. На сервере *RuBackup* создать пул типа **Блочное устройство**.
3. Добавить выделенное блочное устройство в созданный пул.
4. Добавить созданный пул к правилу или стратегии резервного копирования.

Для использования дедупликации необходимо, чтобы модуль резервного копирования соответствующего типа ресурса поддерживал дедупликацию.

Блочное устройство

Чтобы использовать дедупликацию в системе резервного копирования *RuBackup*, необходимо на сервере резервного копирования выделить блочное устройство (одно или несколько) для хранения дедуплицированных резервных копий. Блочным устройством может быть обычный жёсткий диск, `RAID` массив или `LUN` система хранения данных.

В ОС *Linux* получить информацию о доступных блочных устройствах можно с помощью команды `lsblk`, например:

```
# lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda           8:0    0 931,5G  0 disk
  sda1        8:1    0 931,5G  0 part /rubackup1
sdb           8:16   0 931,5G  0 disk
  sdb1        8:17   0 931,5G  0 part /rubackup2
sdc           8:32   0   1,8T  0 disk
  sdc1        8:33   0   1,8T  0 part /rubackup3
sdd           8:48   0   3,6T  0 disk
nvme0n1     259:0    0 953,9G  0 disk
  nvme0n1p1 259:1    0   512M  0 part /boot/efi
  nvme0n1p2 259:2    0 953,4G  0 part /
```

В этом примере на сервере резервного копирования в качестве устройства для хранения дедуплицированных резервных копий может быть использован диск `/dev/sdd`.

Пул хранения данных

Чтобы использовать дедупликацию на сервере резервного копирования необходимо создать пул типа **Блочное устройство**. Это можно сделать при помощи утилиты командной строки `rb_pools` или при помощи менеджера администратора *RBM*, следующим образом:

1. В главном меню RBM открыть пункт **Конфигурация** → **Хранилища** → **Пулы** (Рисунок 1).

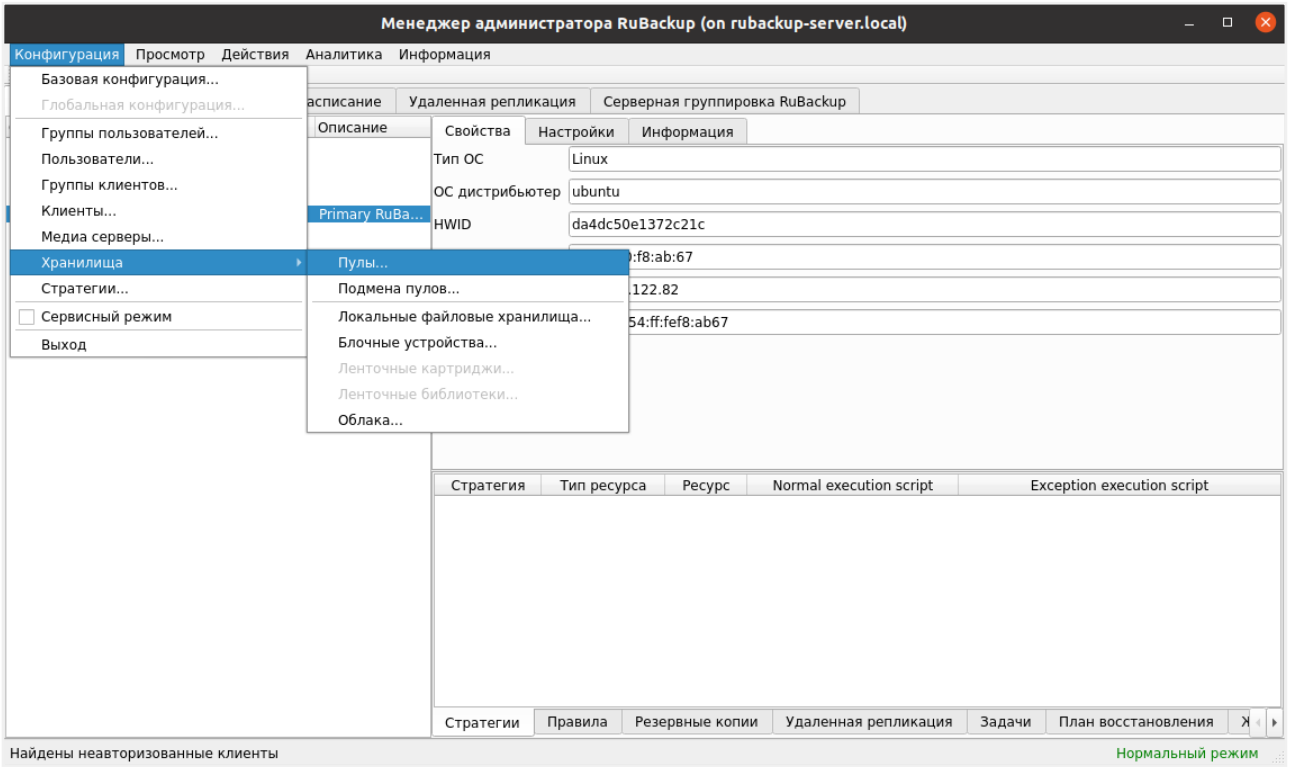

Рисунок 1. Пункт **Пулы** главного меню RBM

2. В окне **Пулы** нажать кнопку **Добавить** (Рисунок 2).

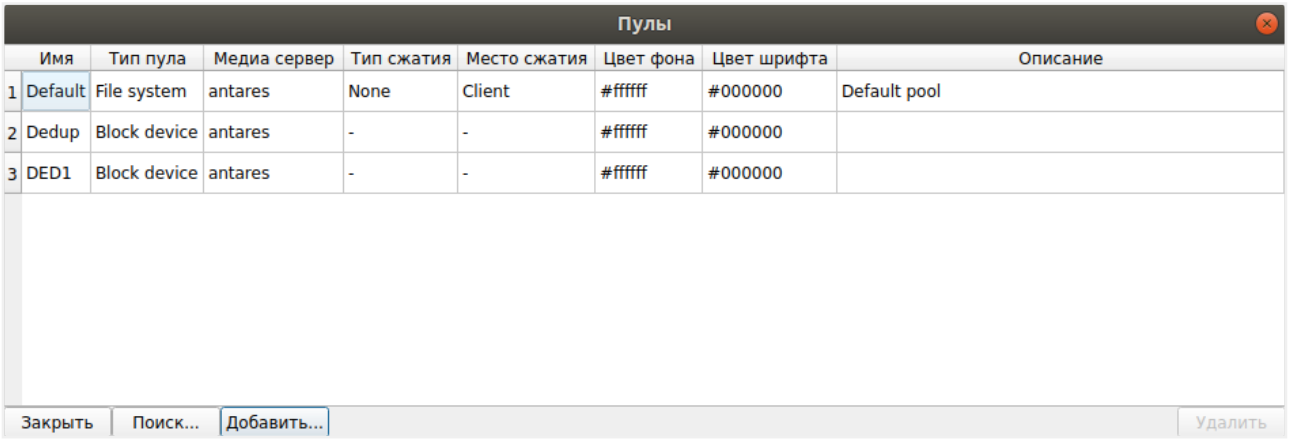

Рисунок 2. Окно **Пулы** в RBM

3. Для нового пула указать имя и тип пула **Блочное устройство**, выбрать размер блока дедупликации, алгоритм хеш-функции и длину хеш-функции (если доступно), а также указать медиасервер, которому будет принадлежать создаваемый пул (если серверная группировка *RuBackup* содержит несколько серверов) (Рисунок 3).

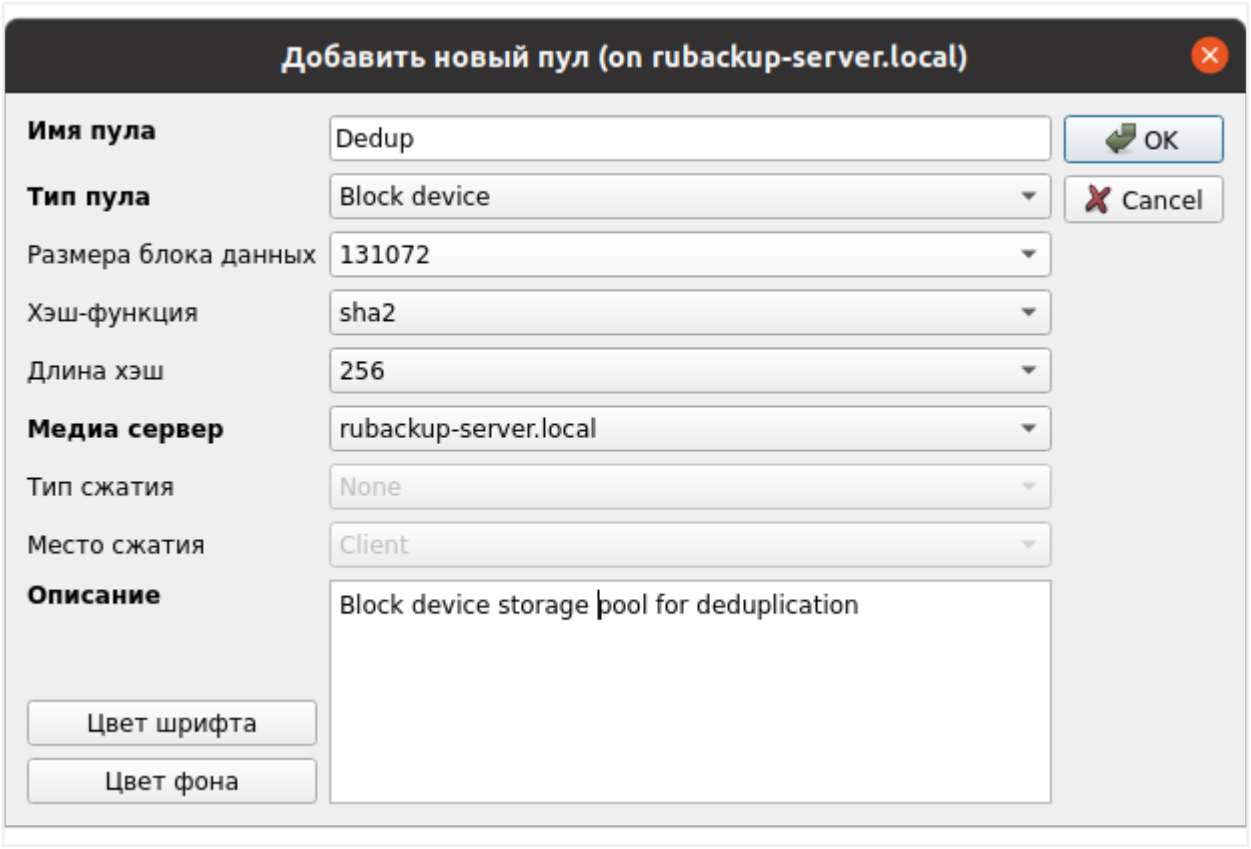

Рисунок 3. Добавление нового пула в RBM

Добавление блочного устройства в пул

В созданный пул типа **Блочное устройство** необходимо добавить одно или несколько выделенных блочных устройств. Это можно сделать при помощи утилиты командной строки `rb_block_devices` или при помощи менеджера администратора *RBM*, следующим образом:

1. В главном меню *RBM* открыть пункт **Конфигурация** → **Хранилища** → **Блочные устройства** (Рисунок 4).

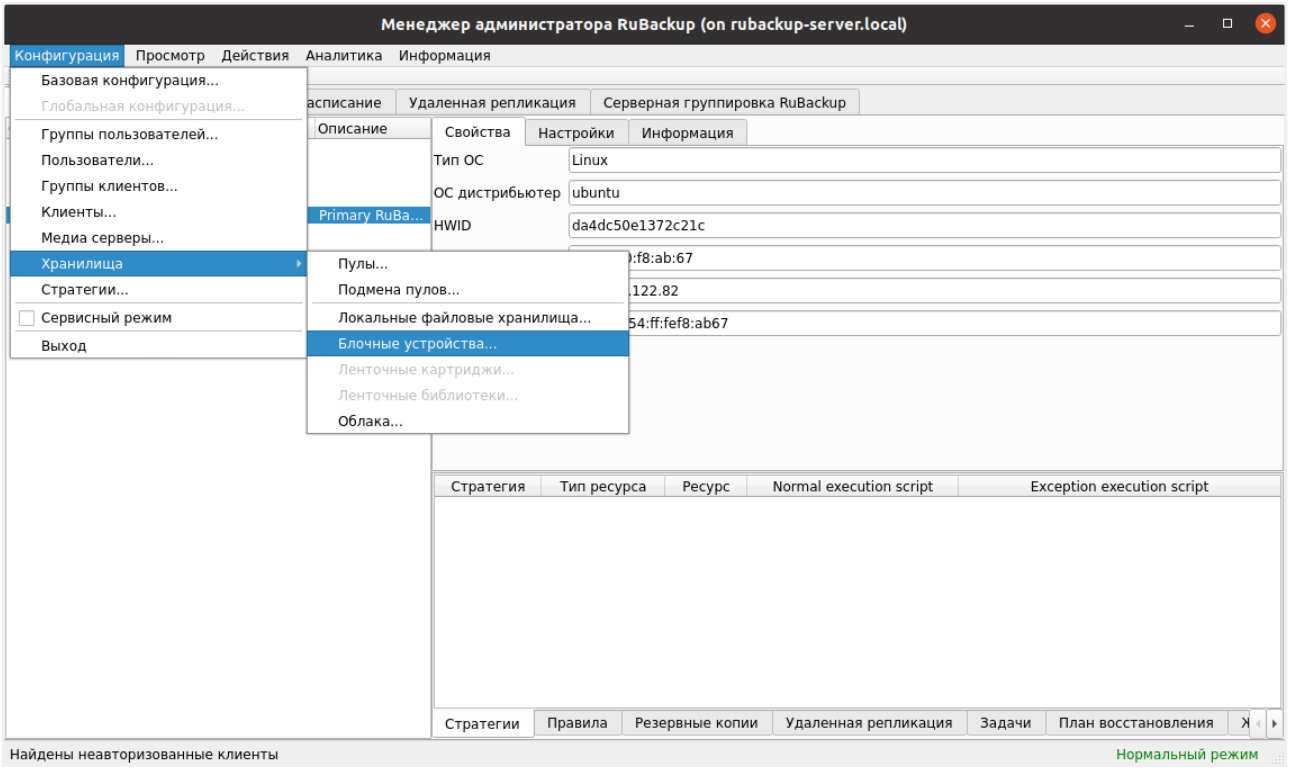

Рисунок 4. Пункт «Блочные устройства» главного меню RBM

2. В окне **Пулы** нажать кнопку **Добавить** (Рисунок 5).

Рисунок 5. Окно «Блочные устройства» в RBM

3. Выбрать созданный пул и выделенное блочное устройство хранения. Если на выбранном блочном устройстве уже существует файловая система, то, чтобы использовать его для хранения дедуплицированных резервных копий, следует перезаписать существующую файловую систему, включив переключатель **Перезаписать сущ. ФС** (Рисунок 6).

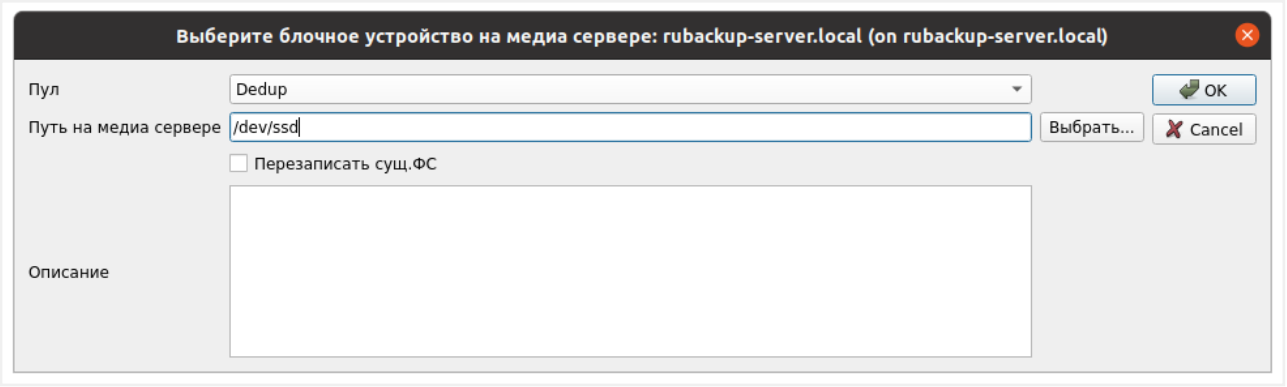

Рисунок 6. Добавление блочного устройства в пул хранения данных

После добавления блочного устройства в систему резервного копирования, оно появится в конфигурации RuBackup. При этом в системный журнальный файл на сервере резервного копирования будет записана информация о добавлении блочного устройства, например:

```
Request to add block device as storage: /dev/sda2 in pool: 'Dedup' media
server: antares
RuBackup block device signature not found on the device: /dev/sda2. Try to
create it: ffc64b63aeef891C
Block device size: 14268435456000
without signature: 14268435451904
Total usable blocks: 82047999
Create table name: deduplicated_block_device_ffc64b63aeef891C for local block
device: /dev/sdd
Local block device: /dev/sdd was included in the pool: Dedup
Load meta data of deduplicated block device: /dev/sdd in memory...
Hash table of: /dev/sda2 loaded
```

Чтобы выполнять резервное копирование с использованием дедупликации, для соответствующего правила или стратегии должен быть выбран пул типа **Блочное устройство** с назначенным в качестве хранилища резервных копий блочным устройством. Также необходимо, чтобы модуль резервного копирования соответствующего типа ресурса поддерживал дедупликацию. Если модуль не поддерживает дедупликацию, то резервное копирование будет завершено с ошибкой.

Для получения дополнительной информации об утилитах командной строки см. Утилиты командной строки.

Параметры системы

Настройка глобальных параметров дедупликации осуществляется в окне настроек глобальной конфигурации системы.

Для получения доступа к меню **Глобальная конфигурация** нужно перевести

систему в сервисный режим. Для этого включите переключатель в меню **Конфигурация** → **Сервисный режим** (Рисунок 7).

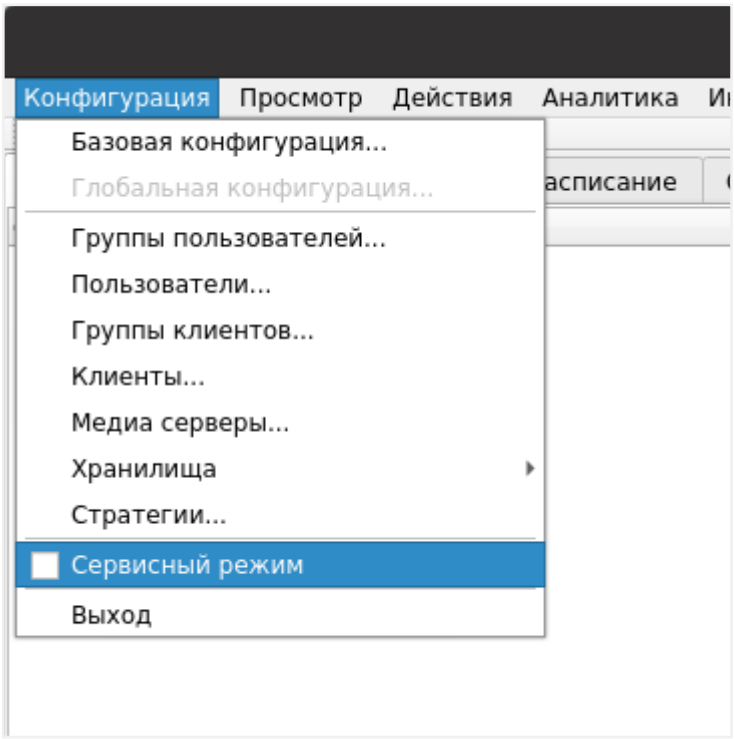

Рисунок 7. Переключение сервисного режима

По завершении работы с окном «Глобальная конфигурация» следует отключить сервисный режим.

Настройки глобальной конфигурации доступны в меню **Конфигурация** → **Глобальная конфигурация** на вкладке **Дедупликация** (Рисунок 8). Там вы можете настроить следующие параметры:

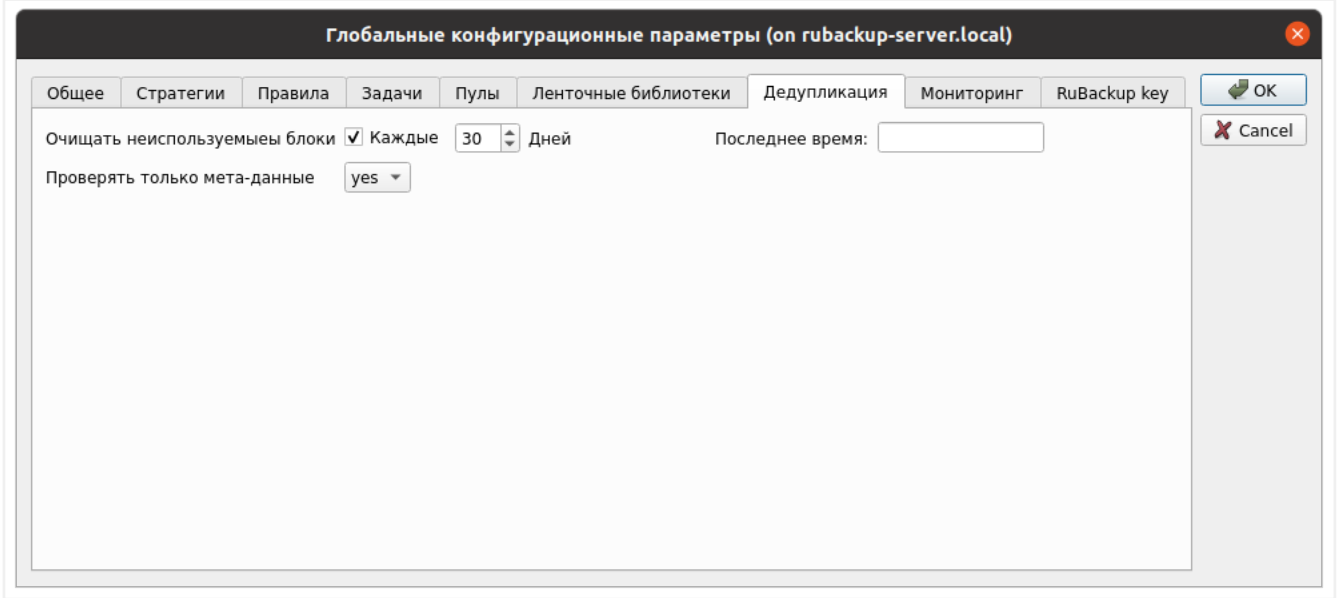

Рисунок 8. Параметры дедупликации в настройках глобальной конфигурации RuBackup

- Какие данные будут проверены на соответствие хеша при проверке резервной копии. Если параметр **Проверять только метаданные** имеет значение `yes` (по умолчанию), то будут проверены только метаданные. При значении этого параметра `no` будут проверены все используемые резервной копией блоки данных в блочном устройстве.
- Возможность периодической очистки блочных устройств. Очистка блочных устройств будет проводиться только в установленное сервисное окно, которое настраивается на вкладке **Общее** при помощи параметров **Начало сервисного окна** и **Окончание сервисного окна** (Рисунок 9).

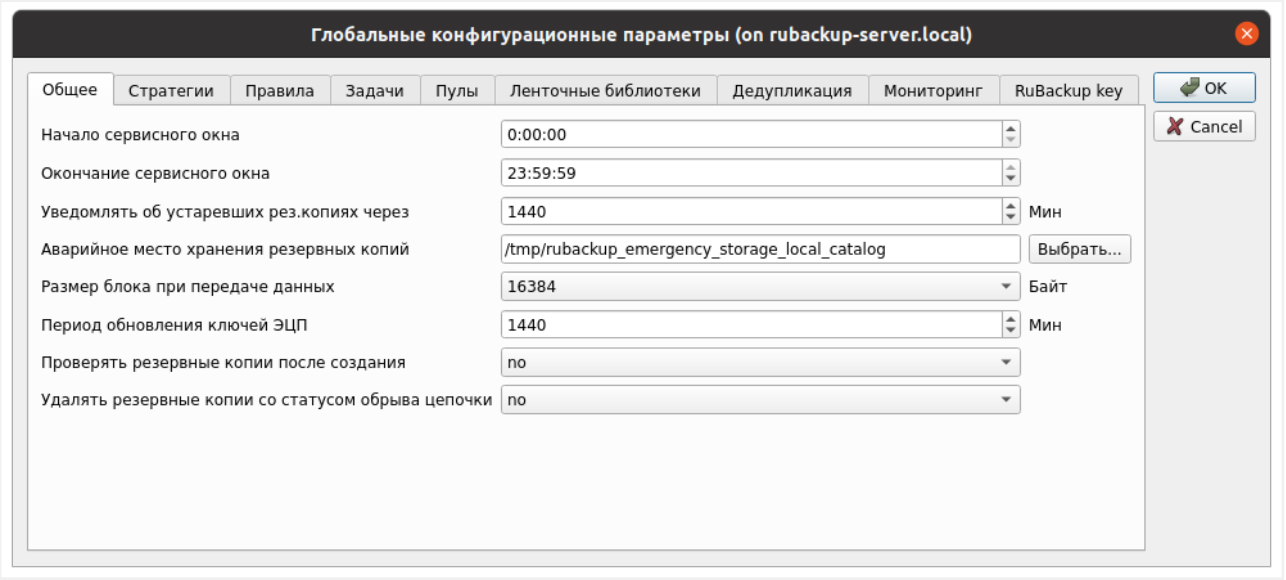

Рисунок 9. Общие параметры в настройках глобальной конфигурации RuBackup

По завершении работы с окном **Глобальная конфигурация** следует отключить сервисный режим.

Особенности

При использовании дедупликации следует учитывать следующие нюансы:

- Для использования дедупликации при выполнении резервного копирования каких-либо данных, необходимо убедиться, что модуль резервного копирования этих данных поддерживает дедупликацию. Показателем этого является поддержка модулем параметра вызова `-D`. При его вызове с этим параметром будет возвращён `0`, например:

```
/opt/rubackup/modules/rb_module_filesystem -D
echo $?
0
```

- Перемещение и копирование резервных копий, созданных с применением

дедупликации, возможно только в пулы типа **Блочное устройство**. При этом параметры пула назначения (размер блока дедупликации, алгоритм хеш-функции и длина хеш-функции) должны совпадать с параметрами пула хранения резервной копии.

- При создании дедуплицированной резервной копии создаётся метафайл, который размещается в пуле типа **Файловая система** сервера резервного копирования. В репозитории RuBackup этот файл указывается одновременно как `archive` и `snapshot` резервной копии. При этом сами данные резервной копии располагаются в блочном устройстве.
- При удалении резервной копии из репозитория происходит удаление только метафайла резервной копии и записи в базе данных RuBackup. Непосредственно блоки данных из хранилища не удаляются. Для освобождения хранилища от неиспользуемых блоков можно периодически выполнять операцию очистки. Настройка этой операции осуществляется в окне настроек глобальной конфигурации системы на вкладке **Дедупликация**.
- При выполнении операции электронной подписи резервной копии будет подписан только метафайл резервной копии, но не сами дедуплицированные блоки данных. При проверке резервной копии будет проверен метафайл. В окне настроек глобальной конфигурации системы на вкладке **Дедупликация** вы можете установить для параметра **Проверять только метаданные** значение `no`. В таком случае на соответствие хеша будут проверены все используемые резервной копией блоки данных в блочном устройстве.
- Если в пул добавлено несколько блочных устройств, то хеш-таблица уникальных блоков будет создана для каждого из устройств. Это означает, что дедупликация работает в рамках одного блочного устройства. Разные устройства могут содержать одинаковые блоки данных.
- Хеш-таблица блочного устройства загружается в оперативную память сервера резервного копирования. Это означает, что при большом объёме блочного устройства потребуется учесть необходимость в большем объёме оперативной памяти.
- Максимально возможный объём памяти для отдельной операции резервного копирования или восстановления определяется в конфигурационном файле `/opt/rubackup/etc/config.file` значением параметра `deduplication-task-memory`. Если на сервере резервного копирования предполагается выполнение большого количества одновременных операций с использованием дедупликации, необходимо учесть это в требованиях к объёму оперативной памяти сервера.
- В репозитории резервного копирования в качестве объёма дедуплицированной резервной копии указывается объём её метафайла.
- При выполнении дедуплицированного резервного копирования файловой системы с файлами разного размера, файл размером больше, чем размер дедуплицированного блока данных, займёт несколько блоков в блочном

устройстве (по возможности, последовательно). Файл размером меньше, чем размер дедуплицированного блока данных, займёт один блок.

- В случае выполнения полной резервной копии на сервер передаются только те блоки данных, которых нет в дедуплицированном хранилище. Это фактически означает, что исчезает практический смысл выполнения инкрементального и дифференциального резервного копирования, и вместо разностного резервного копирования можно всегда выполнять полное резервное копирование. Несмотря на это, модули резервного копирования могут поддерживать разностное резервное копирование и для дедупликационного режима работы.

Работа с сертификатами и ключами SSL

В этом разделе описан процесс создания собственных ключей и сертификатов вместо тех, которые входят в стандартную поставку RuBackup. В комплекте поставки RuBackup есть необходимые для работы SSL-сертификаты клиента и сервера.

Сертификаты, необходимые для работы RuBackup, располагаются в каталоге `/opt/rubackup/keys` и предоставляются в составе пакета `rubackup-common`.

В процессе подключения к серверу клиент отправляет свой сертификат `/opt/rubackup/keys/client/clientCert.crt` для проверки подлинности клиента сервером. Также клиент принимает от сервера его сертификат `/opt/rubackup/keys/server/serverCert.crt` и проверяет его подлинность с использованием серверного корневого сертификата `/opt/rubackup/keys/rootCA/serverRootCACert.crt`. Сервер проверяет подлинность полученного клиентского сертификата с помощью клиентского корневого сертификата `/opt/rubackup/keys/rootCA/clientRootCACert.crt`.

При подключении к серверу оконный менеджер отправляет свой сертификат `/opt/rubackup/keys/rbm/rbmCert.crt` на проверку. Также он принимает от сервера его сертификат `/opt/rubackup/keys/server/serverCert.crt` и проверяет его подлинность с использованием серверного корневого сертификата `/opt/rubackup/keys/rootCA/serverRootCACert.crt`. Сервер проверяет подлинность полученного сертификата оконного менеджера с помощью клиентского корневого сертификата `/opt/rubackup/keys/rootCA/clientRootCACert.crt`.

Для взаимодействия с сервером лицензий и проверки его на подлинность используется корневой сертификат сервера лицензий `/opt/rubackup/keys/rootCA/licenseServerRootCACert.crt`.

Размещение сертификатов и ключей

Файлы приватных ключей следует хранить в надёжном месте, недоступном ни с сервера, ни с клиента RuBackup.

При замене сертификатов на собственные необходимо убедиться, что все сертификаты обновлены на всех узлах, где установлены компоненты RuBackup: клиент, сервер, медиасервер, резервный сервер, оконный менеджер, REST API сервис и другие.

Использование цепочки сертификатов

Иногда клиентский или серверный сертификат подписывается не корневым клиентским или серверным сертификатом, а промежуточным сертификатом, кото-

рый, в свою очередь, подписан корневым или следующим промежуточным сертификатом. Это называется цепочкой сертификатов.

Чтобы RuBackup мог работать с такой цепочкой сертификатов, необходимо объединить все промежуточные и корневой сертификаты в единый корневой клиентский или серверный сертификат.

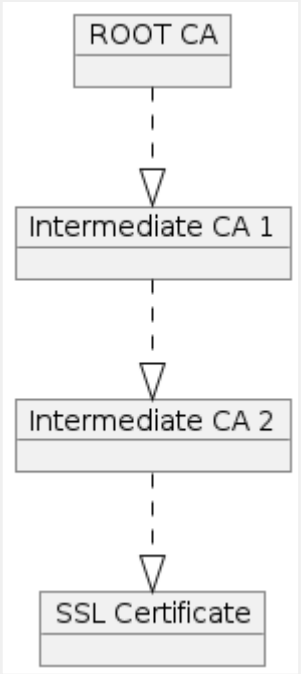

Рисунок 10. Цепочка сертификатов

Серверная часть

Создание сертификата

Чтобы создать серверный сертификат, выполните следующие шаги:

1. Создайте приватный ключ для серверного корневого сертификата командой:

```
openssl genrsa -out serverRootCAKey.key 2048
```

Храните этот ключ в надежном месте!

2. Создайте серверный корневой сертификат. В представленном примере сертификат действует 20000 дней:

```
openssl req -x509 -new -nodes -key serverRootCAKey.key -days 20000 -out
/opt/rubackup/keys/rootCA/serverRootCACert.crt
```

3. В интерактивном меню введите двухбуквенный `код страны`, `провинцию`, `город`,

`организацию`, `подразделение`, `Common Name` и `e-mail адрес`.

4. Создайте приватный ключ сервера:

```
openssl genrsa -out /opt/rubackup/keys/server/serverKey.key 2048
```

5. Создайте запрос на подпись:

```
openssl req -new -key /opt/rubackup/keys/server/serverKey.key -out
/opt/rubackup/keys/server/serverCert.csr
```

6. В интерактивном меню впишите ответ на те же вопросы, что и при создании корневого сертификата. Введенный `Common Name` должен отличаться от `Common Name` у корневого сертификата.
7. Создайте серверный сертификат и подпишите его серверным корневым сертификатом. В представленном примере сертификат действует 20000 дней:

```
openssl x509 -req -in /opt/rubackup/keys/server/serverCert.csr -CA
/opt/rubackup/keys/rootCA/serverRootCACert.crt -CAkey serverRootCAKey.key
-CAcreateserial -out /opt/rubackup/keys/server/serverCert.crt -days 20000
```

8. При необходимости пересоздайте файл, используемый в алгоритме Диффи-Хеллмана, для обмена сессионными ключами с клиентом:

```
openssl dhparam -out /opt/rubackup/keys/server/dh_2048.pem 2048
```

Подготовка сертификатов для сервера

Чтобы подготовить сертификаты для сервера, выполните следующие шаги:

1. Разместите в отдельной папке промежуточные сертификаты и корневой сертификат.
2. Если некоторые из промежуточных или корневой сертификат имеют расширение `.cer` или `.pem`, конвертируйте их в формат `.crt` с помощью одной из следующих команд:

```
openssl x509 -in '<имя сертификата>.pem' -out '<имя сертификата>.crt'
-outform DER

openssl x509 -inform PEM -in '<имя сертификата>.cer' -out '<имя
```

```
сертификата>.crt'
```

3. Объедините промежуточные сертификаты и корневой сертификаты в единый корневой серверный сертификат:

```
cat <путь к промежуточному сертификату 1> <путь к промежуточному
сертификату 2> <путь к корневому сертификату>
/opt/rubackup/keys/rootCA/serverRootCACert.crt
```

Проверка созданных ключей и сертификатов

```
openssl verify -no-CApath -CAfile
/opt/rubackup/keys/rootCA/serverRootCACert.crt
/opt/rubackup/keys/server/serverCert.crt
```

Вывод команды должен содержать: **OK**.

Клиентская часть

Создание сертификата

Чтобы создать клиентский сертификат, выполните следующие шаги:

1. Создайте приватный ключ для клиентского корневого сертификата командой:

```
openssl genrsa -out clientRootCAKey.key 2048
```

Храните этот ключ в надежном месте!

2. Создайте клиентский корневой сертификат. В представленном примере сертификат действует 20000 дней:

```
openssl req -x509 -new -nodes -key serverRootCAKey.key -days 20000 -out
/opt/rubackup/keys/rootCA/clientRootCACert.crt
```

3. В интерактивном меню введите двухбуквенный `код страны`, `провинцию`, `город`, `организацию`, `подразделение`, `Common Name` и `e-mail адрес`.
4. Создайте приватный ключ клиента:

```
openssl genrsa -out /opt/rubackup/keys/client/clientKey.key 2048
```

5. Создайте запрос на подпись:

```
openssl req -new -key /opt/rubackup/keys/client/clientKey.key -out
/opt/rubackup/keys/client/clientCert.csr
```

6. В интерактивном меню впишите ответ на те же вопросы, что и при создании корневого сертификата. Введенный `Common Name` должен отличаться от `Common Name` у корневого сертификата.
7. Создайте клиентский сертификат и подписать его клиентским корневым сертификатом. В представленном примере сертификат действует 20000 дней:

```
openssl x509 -req -in /opt/rubackup/keys/client/clientCert.csr -CA
/opt/rubackup/keys/rootCA/clientRootCACert.crt -CAkey clientRootCAKey.key
-CAcreateserial -out /opt/rubackup/keys/client/clientCert.crt -days 20000
```

Подготовка сертификатов для клиента

```
Чтобы подготовить сертификаты для клиента, выполните следующие шаги:
```

1. Разместите в отдельной папке промежуточные сертификаты и корневой сертификат.
2. Если некоторые из промежуточных или корневой сертификат имеют расширение `.cer` или `.pem`, конвертируйте их в формат `.crt` с помощью одной из следующих команд:

```
openssl x509 -in '<имя сертификата>.pem' -out '<имя сертификата>.crt'
-outform DER

openssl x509 -inform PEM -in '<имя сертификата>.cer' -out '<имя
сертификата>.crt'
```

3. Объедините промежуточные сертификаты и корневой сертификат в единый корневой клиентский сертификат:

```
cat <путь к промежуточному сертификату 1> <путь к промежуточному
сертификату 2> <путь к корневому сертификату>
```

```
/opt/rubackup/keys/rootCA/clientRootCACert.crt
```

Проверка созданных ключей и сертификатов

```
openssl verify -no-CApath -CAfile
/opt/rubackup/keys/rootCA/clientRootCACert.crt
/opt/rubackup/keys/client/clientCert.crt
```

Вывод команды должен содержать: **OK**.

Менеджер администратора RuBackup

Создание сертификата

Чтобы создать сертификат, выполните следующие шаги:

1. Создайте приватный ключ оконного менеджера:

   ```
   openssl genrsa -out /opt/rubackup/keys/rbm/rbmKey.key 2048
   ```

2. Создайте запрос на подпись:

   ```
   openssl req -new -key /opt/rubackup/keys/rbm/rbmKey.key -out
   /opt/rubackup/keys/rbm/rbmCert.csr
   ```

3. В интерактивном меню впишите ответ на те же вопросы, что и при создании корневого сертификата. Введенный `Common Name` должен отличаться от `Common Name` у корневого сертификата.
4. Создайте сертификат оконного менеджера и подпишите его клиентским корневым сертификатом. В представленном примере сертификат действует 20000 дней:

   ```
   openssl x509 -req -in /opt/rubackup/keys/rbm/rbmCert.csr -CA
   /opt/rubackup/keys/rootCA/clientRootCACert.crt -CAkey clientRootCAKey.key
   -CAcreateserial -out /opt/rubackup/keys/rbm/rbmCert.crt -days 20000
   ```

Проверка созданных ключей и сертификатов

```
openssl verify -no-CApath -CAfile /opt/rubackup/keys/rootCA/clientRootCACert
```

```
.crt  /opt/rubackup/keys/rbm/rbmCert.crt
```

Вывод команды должен содержать: **OK**.

Ленточные библиотеки

Система резервного копирования RuBackup позволяет работать с ленточными библиотеками. Ленточная библиотека должна быть подключена к хосту, на котором функционирует сервер RuBackup (основной, резервный или медиасервер).

Ленточные картриджи должны относиться к пулу типа `Tape library, LTFS` либо `Tape library, Native`. По умолчанию в конфигурации RuBackup создаётся пул типа с названием `TL pool`, ассоциированный с основным сервером RuBackup. Картриджи могут находиться в ленточной библиотеке или быть выгружены из неё.

Если картридж выгружен и создана задача, которой необходим доступ к этому картриджу (находящемуся вне ленточной библиотеки), эта задача перейдёт в статус *Suspended* до того, пока необходимый картридж не будет загружен в один из слотов ленточной библиотеки.

Для хранения резервных копий на ленточных картриджах может быть создана файловая система LTFS или использоваться нативное хранение. LTFS позволяет получить доступ к резервным копиям вне системы резервного копирования RuBackup. Нативное хранение позволяет сохранять резервную копию объемом больше, чем объем одного картриджа.

Подготовка к работе с ленточной библиотекой

Установка дополнительного ПО

Для корректной работы с ленточной библиотекой установите драйвер `st`. Для этого введите команду:

```
uname -r
```

Команда выведет версию ядра (например, `5.15.0-91-generic`).

Установите дополнительные модули для вашей версии ядра:

```
sudo apt install linux-modules-extra-5.15.0-91-generic
```

Подгрузите модуль st:

```
sudo modprobe st
```

Также необходимо установить пакеты для всех типов пулов:

- `open-iscsi` — нужен для работы с устройствами подключёнными по ISCSI.
- `lsscsi` — выводит список SCSI-устройств (или хостов), выводит список NVMe-устройств.
- `sg3-utils` — содержит утилиту `sg_reset`, которая отправляет сброс SCSI-устройства, целевого объекта, шины или хоста или проверяет состояние сброса.

Для пулов типа `Tape Library, LTFS` и `Tape library`, `Native` необходим дополнительный пакет:

- `mtx` — управляет устройствами смены носителей SCSI с одним или несколькими приводами, такими как устройства смены лент, автозагрузчики, ленточные библиотеки.

  Исполняемый файл `mtx` обычно устанавливается в `/usr/sbin/mtx`. Если файл находится в другом месте, создайте символическую ссылку:

  ```
  ln -s /путь/к/mtx /usr/sbin/mtx
  ```

Установка sg-драйвера

Информация в данном пункте необходима для использования пулов `Tape library`, `Native`.

Проверка наличия sg-драйвера

Для проверки наличия sg-драйвера выполните команду:

```
lsscsi -g
```

Команда должна показать подключённые устройства, в их числе привод (приводы) ленточной библиотеки и робота ленточной библиотеки.

Если в крайнем правом столбце отображаются sg-пути, то это значит, что sg-драйвер уже установлен и запущен.

Если в крайнем правом столбце отсутствуют sg-пути, то установите sg-драйвер для вашей ОС.

Установка sg-драйвера

1. Установите пакет sg-драйвера в зависимости от вашей ОС:

Astra Linux, Debian, Ubuntu

```
apt install libsgutils2-dev
```

RedOS, CentOS, Rosa Chrome

```
dnf install sg3_utils
```

CentOS 7

```
yum install mt-st
```

Alt Linux 10

```
yum install mt-st udev-rules-sgutils
```

2. Настройте sg-драйвер.

```
sg_scan
modprobe sg
find /dev/ -name "sg"
```

3. Убедитесь, что sg-драйвер установлен и запущен.

```
lsscsi -g
```

Настройки автоматического запуска sg-драйвера

1. Создайте скрипт запуска sg-драйвера:

```
touch /etc/sg_driver_startup.sh
```

Содержание скрипта `sg_driver_startup.sh`

```
#!/bin/sh -e
echo 'init sg-driver'
sg_scan
modprobe sg
echo 'done'
exit 0
```

2. Сделайте скрипт исполняемым:

```
chmod a+x /etc/sg_driver_startup.sh
```

3. Создайте конфигурационный файл для службы `systemd`:

```
touch /lib/systemd/system/sg_driver_startup.service
```

Содержание юнит-файла `sg_driver_startup.service`

```
[Unit]
Description=sg driver startup script
[Service]
ExecStart=/etc/sg_driver_startup.sh
[Install]
WantedBy=multi-user.target
```

4. Запустите сервис немедленно и настройте автозапуск:

```
systemctl enable sg_driver_startup.service --now
```

5. Добавьте зависимость от сервиса `sg_driver_startup.service` в `rubackup_server.service`:

```
systemctl edit --full rubackup_server.service
```

```
[Unit]
Description=RuBackup server
Requires=network.target
After=network.target postgresql.service sg_driver_startup.service
```

6. Перезагрузите ОС для применения настроек.
7. После перезагрузки проверьте статус сервиса `rubackup_server.service`:

```
systemctl status rubackup_server.service
```

Сборка LTFS

Информация в данном пункте необходима для использования пулов *Tape*

Library, *LTFS*.

Страница проекта: https://github.com/LinearTapeFileSystem/ltfs

Зависимости, которые должны быть установлены перед сборкой: https://github.com/LinearTapeFileSystem/ltfs/wiki/Build-Environments

Общая сборочная инструкция: https://github.com/LinearTapeFileSystem/ltfs#build-and-install-on-linux

Поддерживаемые устройства: https://github.com/LinearTapeFileSystem/ltfs#supported-tape-drives

1. Установите зависимости, необходимые для сборки LTFS:

RPM-based OS (Rosa Cobalt, RHEL)

```
yum install perl make gcc git pkg-config libxml2-dev automake autoconf
libtool uuid uuid-devel libuuid-devel icu fuse fuse-devel libicu-devel
net-snmp net-snmp-devel
```

RPM-based OS (RedOS, CentOS, Rosa Chrome)

```
dnf install perl make gcc git pkg-config libxml2-dev automake autoconf
libtool uuid uuid-devel libuuid-devel icu fuse fuse-devel libicu-devel
net-snmp net-snmp-devel
```

DEB-based OS (кроме Astra Linux 1.8 и Debian 12)

```
apt install make git pkg-config libxml2-dev automake autoconf libtool
uuid uuid-dev fuse libfuse-dev libsnmp-dev icu-devtools libicu-dev
```

Astra Linux 1.8 и Debian 12

```
apt install make git pkg-config libxml2-dev automake autoconf libtool
uuid uuid-dev libfuse2 libfuse-dev libsnmp-dev icu-devtools libicu-dev
```

Создайте файл `/usr/bin/icu-config` со следующим содержимым:

```
#!/bin/sh
opts=$1
case $opts in '--cppflags')
echo '' ;;
'--ldflags')
```

```
echo '-licuuc -licudata -ldl' ;;
*)
echo '/usr/lib/x86_64-linux-gnu/icu/pkgdata.inc' ;;
esac
```

и выполните команду:

```
chmod 755 /usr/bin/icu-config
```

2. Сборка

   Для сборки выполните:

```
git clone https://github.com/LinearTapeFileSystem/ltfs.git
cd ltfs
 ./autogen.sh
 ./configure
make
make install
ldconfig -v
```

3. Проверьте, подключена ли ленточная библиотека к хосту:

```
lsscsi -g
```

   Команда должна показать подключённые устройства, в их числе привод (приводы) ленточной библиотеки и робота ленточной библиотеки:

   `[1:0:0:0] tape IBM ULT3580-TD6 D8E4 /dev/st0 /dev/sg5`

   `[1:0:0:1] mediumx IBM3573-TL C.20 /dev/sch0 /dev/sg6`

   В данном случае у библиотеки есть один ленточный привод (магнитофон) и робот, к которому можно обращаться через `/dev/sg6`.

4. Получите информацию о ленточной библиотеке, обращаясь к роботу ленточной библиотеки:

```
mtx -f /dev/sg6 status
```

   Эта команда должна показать информацию о слотах ленточной библиотеки, о загруженных в них картриджах и о приводах ленточной библиотеки.

Пример вывода `sudo mtx -f /dev/sg6 status`

```
Storage Changer /dev/sg6:1 Drives, 24 Slots ( 1 Import/Export )
Data Transfer Element 0:Empty
Storage Element 1:Full :VolumeTag=INT020L6
Storage Element 2:Full :VolumeTag=INT023L6
Storage Element 3:Full :VolumeTag=INT033L6
Storage Element 4:Full :VolumeTag=INT026L6
Storage Element 5:Full :VolumeTag=INT029L6
Storage Element 6:Full :VolumeTag=INT022L6
Storage Element 7:Full :VolumeTag=INT034L6
Storage Element 8:Full :VolumeTag=INT025L6
Storage Element 9:Full :VolumeTag=INT028L6
Storage Element 10:Full :VolumeTag=INT021L6
Storage Element 11:Full :VolumeTag=INT024L6
Storage Element 12:Full :VolumeTag=INT039L6
Storage Element 13:Full :VolumeTag=INT012L6
Storage Element 14:Full :VolumeTag=INT011L6
Storage Element 15:Empty
Storage Element 16:Full :VolumeTag=INT036L6
Storage Element 17:Full :VolumeTag=INT014L6
Storage Element 18:Full :VolumeTag=INT010L6
Storage Element 19:Empty
Storage Element 20:Full :VolumeTag=INT038L6
Storage Element 21:Full :VolumeTag=INT037L6
Storage Element 22:Empty
Storage Element 23:Full :VolumeTag=CLNU41L1
Storage Element 24
IMPORT/EXPORT:Full :VolumeTag=INT027L6
```

В данном случае библиотека состоит из 24 слотов, один из которых — слот ввода-вывода, через который можно импортировать или экспортировать ленточные картриджи, и один ленточный привод (сейчас пуст). Слоты ленточной библиотеки заполнены картриджами с определенными VolumeTag, один из картриджей — чистящий, три слота в ленточной библиотеке пусты.

5. Загрузите картридж в ленточную библиотеку, создайте на нем файловую систему LTFS и проверьте её работу:

```
mtx -f /dev/sg6 load 1 0
```

В результате выполнения этой команды картридж из слота 1 будет загружен в единственный магнитофон 0.

6. Создайте файловую систему LTFS на картридже:

```
mkltfs -f -d /dev/sg5
```

7. Проверьте файловую систему LTFS:

```
ltfsck /dev/sg5
```

8. Создайте точку монтирования:

```
mkdir /ltfs
```

9. Монтируйте файловую систему LTFS:

```
ltfs -o devname=/dev/sg5 /ltfs/
```

10. Убедитесь, что файловая система примонтирована:

```
df -k /ltfs
```

```
Filesystem 1K-blocks Used Available Use% Mounted on

ltfs:/dev/sg5 2351648768 0 2351648768 0% /ltfs
```

11. Отмонтировать файловую систему LTFS:

```
umount /ltfs
```

12. Возвратить картридж из магнитофона 0 в слот 1:

```
mtx -f /dev/sg6 unload 1 0
```

Если все действия завершились успешно, то ленточная библиотека готова к работе с сервером RuBackup.

Работа с ленточной библиотекой

Для работы с ленточными библиотеками в приложении Менеджер администратора RuBackup (RBM) смотрите раздел Ленточные библиотеки.

Для работы с ленточными библиотеками в веб-приложении Tucana смотрите раздел Ленточные библиотеки.

Для работы с ленточными библиотеками с помощью Утилит командной строки предназначены следующие утилиты:

- `rb_tape_libraries` — информация о ленточных библиотеках в системе резервного копирования и управление ими.
- `rb_tape_cartridges` — информация о коллекции ленточных картриджей, зарегистрированных в системе резервного копирования, и управление ими.
- `rb_tl_task_queue` — информация о заданиях в очереди ленточных библиотек.
- `rb_inventory` — информация о резервных копиях, которые были сделаны вне текущей конфигурации RuBackup (например, в другой серверной группировке RuBackup).

Утилиты командной строки не предназначены для настройки новой библиотеки в системе резервного копирования RuBackup. Для этого воспользуйтесь приложением Менеджер администратора RuBackup (RBM).