Proxmox Backup Documentation

Release 2.4.5-1

Proxmox Support Team

Wednesday, 24 January 2024

TABLE OF CONTENTS

Version 2.4.5 -- Wednesday, 24 January 2024

CHAPTER

ONE

INTRODUCTION

1.1 What is Proxmox Backup Server?

Proxmox Backup Server is an enterprise-class, client-server backup solution that is capable of backing up *virtual machine*s, *container*s, and physical hosts. It is specially optimized for the Proxmox Virtual Environment platform and allows you to back up your data securely, even between remote sites, providing easy management through a web-based user interface.

It supports deduplication, compression, and authenticated encryption (AE). Using *Rust* as the implementation language guarantees high performance, low resource usage, and a safe, high-quality codebase.

Proxmox Backup uses state of the art cryptography for both client-server communication and backup content *encryption*. All client-server communication uses TLS, and backup data can be encrypted on the client-side before sending, making it safer to back up data to targets that are not fully trusted.

1.2 Architecture

Proxmox Backup Server uses a client-server model. The server stores the backup data and provides an API to create and manage datastores. With the API, it's also possible to manage disks and other server-side resources.

The backup client uses this API to access the backed up data. You can use the `proxmox-backup-client` command line tool to create and restore file backups. For QEMU and LXC within Proxmox Virtual Environment, we deliver an integrated client.

A single backup is allowed to contain several archives. For example, when you backup a *virtual machine*, each disk is stored as a separate archive inside that backup. The VM configuration itself is stored as an extra file. This way, it's easy to access and restore only the important parts of the backup, without the need to scan the whole backup.

1.3 Main Features

Support for Proxmox VE The Proxmox Virtual Environment is fully supported, and you can easily backup *virtual machine*s and *container*s.

Performance The whole software stack is written in *Rust*, in order to provide high speed and memory efficiency.

Deduplication Periodic backups produce large amounts of duplicate data. The deduplication layer avoids redundancy and minimizes the storage space used.

Incremental backups Changes between backups are typically low. Reading and sending only the delta reduces the storage and network impact of backups.

Data integrity The built-in SHA-256 checksum algorithm ensures accuracy and consistency in your backups.

Remote sync It is possible to efficiently synchronize data to remote sites. Only deltas containing new data are transferred.

Compression The ultra-fast Zstandard compression is able to compress several gigabytes of data per second.

Encryption Backups can be encrypted on the client-side, using AES-256 GCM. This authenticated encryption (AE) mode provides very high performance on modern hardware. In addition to client-side encryption, all data is transferred via a secure TLS connection.

Tape backup For long-term archiving of data, Proxmox Backup Server also provides extensive support for backing up to tape and managing tape libraries.

Ransomware protection *Protect your critical data from ransomware attacks* with Proxmox Backup Server's fine-grained access control, data integrity verification, and offsite backup through remote sync and tape backup.

Web interface Manage the Proxmox Backup Server with the integrated, web-based user interface.

Open source No secrets. Proxmox Backup Server is free and open-source software. The source code is licensed under AGPL, v3.

No limits Proxmox Backup Server has no artificial limits for backup storage or backup-clients.

Enterprise support Proxmox Server Solutions GmbH offers enterprise support in the form of Proxmox Backup Server Subscription Plans. Users at every subscription level get access to the Proxmox Backup *Enterprise Repository*. In addition, with a Basic, Standard or Premium subscription, users have access to the *Proxmox Customer Portal*.

1.4 Reasons for Data Backup?

The main purpose of a backup is to protect against data loss. Data loss can be caused by both faulty hardware and human error.

A common mistake is to accidentally delete a file or folder which is still required. Virtualization can even amplify this problem, as deleting a whole virtual machine can be as easy as pressing a single button.

For administrators, backups can serve as a useful toolkit for temporarily storing data. For example, it is common practice to perform full backups before installing major software updates. If something goes wrong, you can easily restore the previous state.

Another reason for backups are legal requirements. Some data, especially business records, must be kept in a safe place for several years by law, so that they can be accessed if required.

In general, data loss is very costly as it can severely damage your business. Therefore, ensure that you perform regular backups and run restore tests.

1.5 Software Stack

Proxmox Backup Server consists of multiple components:

- A server-daemon providing, among other things, a RESTful API, super-fast asynchronous tasks, lightweight usage statistic collection, scheduling events, strict separation of privileged and unprivileged execution environments
- A JavaScript management web interface
- A management CLI tool for the server (*proxmox-backup-manager*)
- A client CLI tool (*proxmox-backup-client*) to access the server easily from any *Linux amd64* environment

Aside from the web interface, most parts of Proxmox Backup Server are written in the Rust programming language.

> "The Rust programming language helps you write faster, more reliable software. High-level ergonomics and low-level control are often at odds in programming language design; Rust challenges that conflict. Through balancing powerful technical capacity and a great developer experience, Rust gives you the option to control low-level details (such as memory usage) without all the hassle traditionally associated with such control."
>
> —The Rust Programming Language

1.6 Getting Help

1.6.1 Enterprise Support

Users with a Proxmox Backup Server Basic, Standard or Premium Subscription Plan have access to the Proxmox Customer Portal. The customer portal provides support with guaranteed response times from the Proxmox developers. For more information or for volume discounts, please contact sales@proxmox.com.

1.6.2 Community Support Forum

We always encourage our users to discuss and share their knowledge using the Proxmox Community Forum. The forum is moderated by the Proxmox support team. The large user base is spread out all over the world. Needless to say that such a large forum is a great place to get information.

1.6.3 Mailing Lists

Proxmox Backup Server is fully open-source and contributions are welcome! Here is the primary communication channel for developers:

Mailing list for developers **'Proxmox Backup Sever Development List'_**

1.6.4 Bug Tracker

Proxmox runs a public bug tracker at https://bugzilla.proxmox.com. If an issue appears, file your report there. An issue can be a bug, as well as a request for a new feature or enhancement. The bug tracker helps to keep track of the issue and will send a notification once it has been solved.

1.7 License

1.8 History

Backup is, and always has been, a central aspect of IT administration. The need to recover from data loss is fundamental and only increases with virtualization.

For this reason, we've been shipping a backup tool with Proxmox VE, from the beginning. This tool is called `vzdump` and is able to make consistent snapshots of running LXC containers and KVM virtual machines.

However, `vzdump` only allows for full backups. While this is fine for small backups, it becomes a burden for users with large VMs. Both backup duration and storage usage are too high for this case, especially for users who want to keep many backups of the same VMs. To solve these problems, we needed to offer deduplication and incremental backups.

Back in October 2018, development started. We investigated several technologies and frameworks and finally decided to use *Rust* as the implementation language, in order to provide high speed and memory efficiency. The 2018-edition of Rust seemed promising for our requirements.

In July 2020, we released the first beta version of Proxmox Backup Server, followed by the first stable version in November 2020. With support for encryption and incremental, fully deduplicated backups, Proxmox Backup offers a secure environment, which significantly reduces network load and saves valuable storage space.

CHAPTER

TWO

INSTALLATION

Proxmox Backup is split into a server and client part. The server part can either be installed with a graphical installer or on top of Debian from the provided package repository.

2.1 System Requirements

We recommend using high quality server hardware when running Proxmox Backup in production. To further decrease the impact of a failed host, you can set up periodic, efficient, incremental *datastore synchronization* from other Proxmox Backup Server instances.

2.1.1 Minimum Server Requirements, for Evaluation

These minimum requirements are for evaluation purposes only and should not be used in production.

- CPU: 64bit (*x86-64* or *AMD64*), 2+ Cores
- Memory (RAM): 2 GB RAM
- Hard drive: more than 8GB of space.
- Network card (NIC)

2.1.2 Recommended Server System Requirements

- CPU: Modern AMD or Intel 64-bit based CPU, with at least 4 cores
- Memory: minimum 4 GiB for the OS, filesystem cache and Proxmox Backup Server daemons. Add at least another GiB per TiB storage space.
- OS storage:
  - 32 GiB, or more, free storage space
  - Use a hardware RAID with battery protected write cache (*BBU*) or a redundant ZFS setup (ZFS is not compatible with a hardware RAID controller).
- Backup storage:
  - Use only SSDs, for best results
  - If HDDs are used: Using a metadata cache is highly recommended, for example, add a ZFS *special device mirror*.
- Redundant Multi-GBit/s network interface cards (NICs)

2.1.3 Supported Web Browsers for Accessing the Web Interface

To access the server's web-based user interface, we recommend using one of the following browsers:

- Firefox, a release from the current year, or the latest Extended Support Release
- Chrome, a release from the current year
- Microsoft's currently supported version of Edge
- Safari, a release from the current year

2.2 Debian Package Repositories

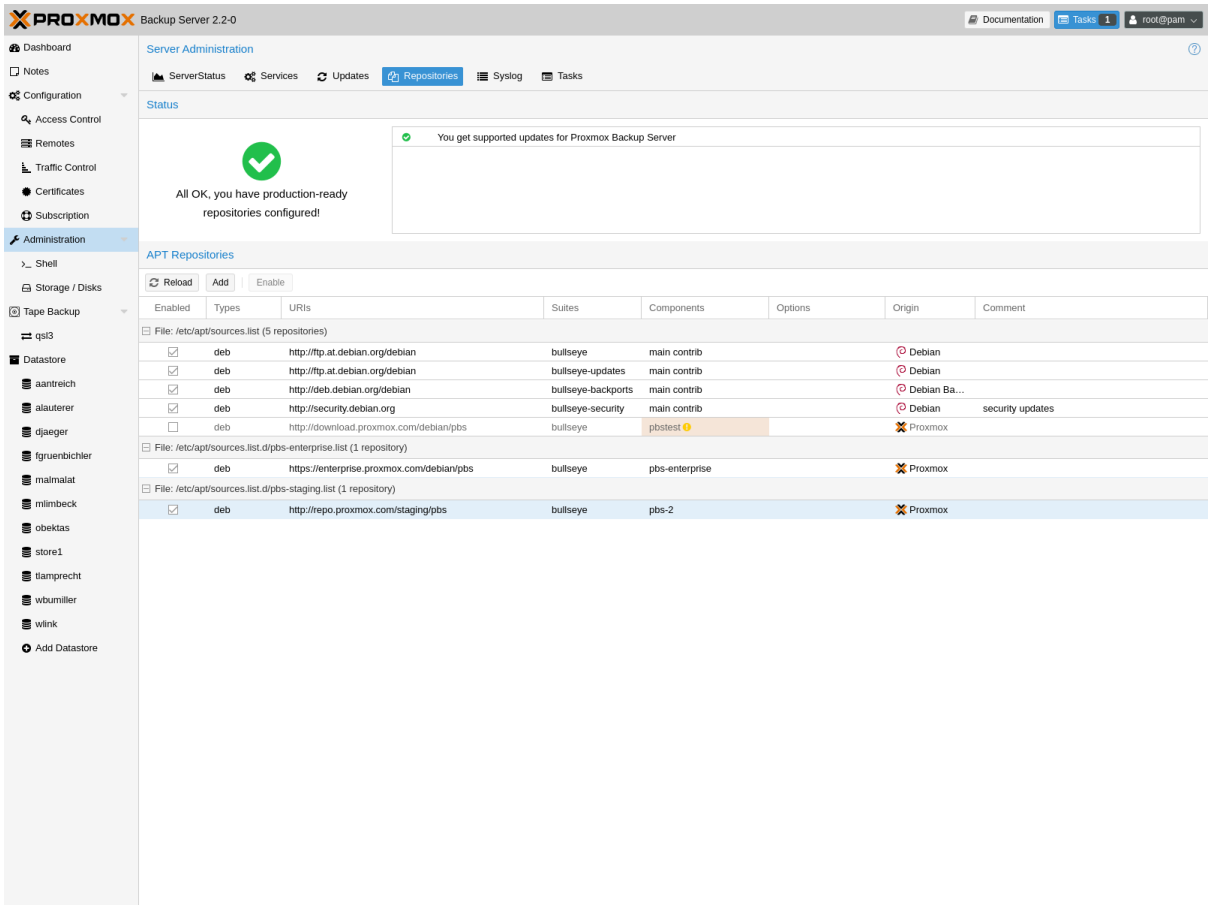
All Debian based systems use APT as a package management tool. The lists of repositories are defined in `/etc/apt/sources.list` and the `.list` files found in the `/etc/apt/sources.d/` directory. Updates can be installed directly with the `apt` command line tool, or via the GUI.

APT `sources.list` files list one package repository per line, with the most preferred source listed first. Empty lines are ignored and a `#` character anywhere on a line marks the remainder of that line as a comment. The information available from the configured sources is acquired by `apt update`.

Listing 1: File: `/etc/apt/sources.list`

```
deb http://ftp.debian.org/debian bullseye main contrib
deb http://ftp.debian.org/debian bullseye-updates main contrib

# security updates
deb http://security.debian.org/debian-security bullseye-security main contrib
```

In addition, you need a package repository from Proxmox to get Proxmox Backup updates.



2.2.1 SecureApt

The *Release* files in the repositories are signed with GnuPG. APT is using these signatures to verify that all packages are from a trusted source.

If you install Proxmox Backup Server from an official ISO image, the verification key is already installed.

If you install Proxmox Backup Server on top of Debian, download and install the key with the following commands:

```
# wget https://enterprise.proxmox.com/debian/proxmox-release-bullseye.gpg -O /etc/apt/trusted.
↪gpg.d/proxmox-release-bullseye.gpg
```

Verify the SHA512 checksum afterwards with the expected output below:

```
# sha512sum /etc/apt/trusted.gpg.d/proxmox-release-bullseye.gpg
7fb03ec8a1675723d2853b84aa4fdb49a46a3bb72b9951361488bfd19b29aab0a789a4f8c7406e71a69aabbc727c936d3549731c4659f
↪ /etc/apt/trusted.gpg.d/proxmox-release-bullseye.gpg
```

and the md5sum, with the expected output below:

```
# md5sum /etc/apt/trusted.gpg.d/proxmox-release-bullseye.gpg
bcc35c7173e0845c0d6ad6470b70f50e /etc/apt/trusted.gpg.d/proxmox-release-bullseye.gpg
```

2.2.2 Proxmox Backup Enterprise Repository

This is the stable, recommended repository. It is available for all Proxmox Backup subscription users. It contains the most stable packages, and is suitable for production use. The `pbs-enterprise` repository is enabled by default:

Listing 2: File: `/etc/apt/sources.list.d/pbs-enterprise.list`

```
deb https://enterprise.proxmox.com/debian/pbs bullseye pbs-enterprise
```

To never miss important security fixes, the superuser (`root@pam` user) is notified via email about new packages as soon as they are available. The change-log and details of each package can be viewed in the GUI (if available).

Please note that you need a valid subscription key to access this repository. More information regarding subscription levels and pricing can be found at https://www.proxmox.com/en/proxmox-backup-server/pricing

Note: You can disable this repository by commenting out the above line using a # (at the start of the line). This prevents error messages if you do not have a subscription key. Please configure the `pbs-no-subscription` repository in that case.

2.2.3 Proxmox Backup No-Subscription Repository

As the name suggests, you do not need a subscription key to access this repository. It can be used for testing and non-production use. It is not recommended to use it on production servers, because these packages are not always heavily tested and validated.

We recommend to configure this repository in `/etc/apt/sources.list`.

Listing 3: File: `/etc/apt/sources.list`

```
deb http://ftp.debian.org/debian bullseye main contrib
deb http://ftp.debian.org/debian bullseye-updates main contrib

# Proxmox Backup Server pbs-no-subscription repository provided by proxmox.com,
# NOT recommended for production use
deb http://download.proxmox.com/debian/pbs bullseye pbs-no-subscription

# security updates
deb http://security.debian.org/debian-security bullseye-security main contrib
```

2.2.4 Proxmox Backup Test Repository

This repository contains the latest packages and is heavily used by developers to test new features.

You can access this repository by adding the following line to `/etc/apt/sources.list`:

Listing 4: sources.list entry for `pbstest`

```
deb http://download.proxmox.com/debian/pbs bullseye pbstest
```

2.2.5 Proxmox Backup Client-only Repository

If you want to *use the the Proxmox Backup Client* on systems using a Linux distribution not based on Proxmox projects, you can use the client-only repository.

Currently there's only a client-repository for APT based systems.

APT-based Proxmox Backup Client Repository

For modern Linux distributions using *apt* as package manager, like all Debian and Ubuntu Derivative do, you may be able to use the APT-based repository.

In order to configure this repository you need to first *setup the Proxmox release key*. After that, add the repository URL to the APT sources lists.

Repositories for Debian 11 (Bullseye) based releases

This repository is tested with:

- Debian Bullseye

Edit the file `/etc/apt/sources.list.d/pbs-client.list` and add the following snippet

Listing 5: File: `/etc/apt/sources.list`

```
deb http://download.proxmox.com/debian/pbs-client bullseye main
```

Repositories for Debian 10 (Buster) based releases

This repository is tested with:

- Debian Buster
- Ubuntu 20.04 LTS

It may work with older, and should work with more recent released versions.

Edit the file `/etc/apt/sources.list.d/pbs-client.list` and add the following snippet

Listing 6: File: `/etc/apt/sources.list`

```
deb http://download.proxmox.com/debian/pbs-client buster main
```

2.2.6 Repository Access Behind HTTP Proxy

Some setups have restricted access to the internet, sometimes only through a central proxy. You can setup a HTTP proxy through the Proxmox Backup Server's web-interface in the *Configuration -> Authentication* tab.

Once configured this proxy will be used for apt network requests and for checking a Proxmox Backup Server support subscription.

Standard HTTP proxy configurations are accepted, *[http://]<host>[:port]* where the *<host>* part may include an authorization, for example: *http://user:pass@proxy.example.org:12345*

2.3 Server Installation

The backup server stores the actual backed up data and provides a web based GUI for various management tasks such as disk management.

Note: You always need a backup server. It is not possible to use Proxmox Backup without the server part.

The disk image (ISO file) provided by Proxmox includes a complete Debian system as well as all necessary packages for the Proxmox Backup Server.

The installer will guide you through the setup process and allow you to partition the local disk(s), apply basic system configuration (for example timezone, language, network), and install all required packages. The provided ISO will get you started in just a few minutes, and is the recommended method for new and existing users.

Alternatively, Proxmox Backup Server can be installed on top of an existing Debian system.

2.3.1 Install Proxmox Backup Server using the Installer

Download the ISO from https://www.proxmox.com/downloads. It includes the following:

- The Proxmox Backup Server installer, which partitions the local disk(s) with ext4, xfs or ZFS, and installs the operating system
- Complete operating system (Debian Linux, 64-bit)
- Proxmox Linux kernel with ZFS support
- Complete tool-set to administer backups and all necessary resources
- Web based management interface

Note: During the installation process, the complete server is used by default and all existing data is removed.

2.3.2 Install Proxmox Backup Server on Debian

Proxmox ships as a set of Debian packages which can be installed on top of a standard Debian installation. After configuring the *Debian Package Repositories*, you need to run:

```
# apt-get update
# apt-get install proxmox-backup-server
```

The above commands keep the current (Debian) kernel and install a minimal set of required packages.

If you want to install the same set of packages as the installer does, please use the following:

```
# apt-get update
# apt-get install proxmox-backup
```

This will install all required packages, the Proxmox kernel with ZFS support, and a set of common and useful packages.

Caution: Installing Proxmox Backup on top of an existing Debian installation looks easy, but it assumes that the base system and local storage have been set up correctly. In general this is not trivial, especially when LVM or ZFS is used. The network configuration is completely up to you as well.

Note: You can access the web interface of the Proxmox Backup Server with your web browser, using HTTPS on port 8007. For example at `https://<ip-or-dns-name>:8007`

2.3.3 Install Proxmox Backup Server on Proxmox VE

After configuring the *Debian Package Repositories*, you need to run:

```
# apt-get update
# apt-get install proxmox-backup-server
```

Caution: Installing the backup server directly on the hypervisor is not recommended. It is safer to use a separate physical server to store backups. Should the hypervisor server fail, you can still access the backups.

Note: You can access the web interface of the Proxmox Backup Server with your web browser, using HTTPS on port 8007. For example at `https://<ip-or-dns-name>:8007`

2.4 Client Installation

2.4.1 Install Proxmox Backup Client on Debian

Proxmox ships as a set of Debian packages to be installed on top of a standard Debian installation. After configuring the *APT-based Proxmox Backup Client Repository*, you need to run:

```
# apt-get update
# apt-get install proxmox-backup-client
```

Note: The client-only repository should be usable by most recent Debian and Ubuntu derivatives.

CHAPTER

THREE

TERMINOLOGY

3.1 Backup Content

When doing deduplication, there are different strategies to get optimal results in terms of performance and/or deduplication rates. Depending on the type of data, it can be split into *fixed* or *variable* sized chunks.

Fixed sized chunking requires minimal CPU power, and is used to backup virtual machine images.

Variable sized chunking needs more CPU power, but is essential to get good deduplication rates for file archives.

The Proxmox Backup Server supports both strategies.

3.1.1 Image Archives: `<name>.img`

This is used for virtual machine images and other large binary data. Content is split into fixed-sized chunks.

3.1.2 File Archives: `<name>.pxar`

A file archive stores a full directory tree. Content is stored using the *Proxmox File Archive Format (.pxar)*, split into variable-sized chunks. The format is optimized to achieve good deduplication rates.

3.1.3 Binary Data (BLOBs)

This type is used to store smaller (< 16MB) binary data such as configuration files. Larger files should be stored as image archives.

Caution: Please do not store all files as BLOBs. Instead, use the file archive to store entire directory trees.

3.1.4 Catalog File: `catalog.pcat1`

The catalog file is an index for file archives. It contains the list of included files and is used to speed up search operations.

3.1.5 The Manifest: `index.json`

The manifest contains a list of all backed up files, and their sizes and checksums. It is used to verify the consistency of a backup.

3.2 Backup Namespace

Namespaces allow for the reuse of a single chunk store deduplication domain for multiple sources, while avoiding naming conflicts and enabling more fine-grained access control.

Essentially, they're implemented as a simple directory structure and don't require separate configuration.

3.3 Backup Type

The backup server groups backups by *type*, where *type* is one of:

`vm` This type is used for *virtual machine*s. It typically consists of the virtual machine's configuration file and an image archive for each disk.

`ct` This type is used for *container*s. It consists of the container's configuration and a single file archive for the filesystem's contents.

`host` This type is used for file/directory backups created from within a machine. Typically this would be a physical host, but could also be a virtual machine or container. Such backups may contain file and image archives; there are no restrictions in this regard.

3.4 Backup ID

A unique ID for a specific Backup Type and Backup Namespace. Usually the virtual machine or container ID. `host` type backups normally use the hostname.

3.5 Backup Time

The time when the backup was made with second resolution.

3.6 Backup Group

The tuple `<type>/<id>` is called a backup group. Such a group may contain one or more backup snapshots.

3.7 Backup Snapshot

The triplet `<type>/<ID>/<time>` is called a backup snapshot. It uniquely identifies a specific backup within a namespace.

Listing 1: Backup Snapshot Examples

```
vm/104/2019-10-09T08:01:06Z
host/elsa/2019-11-08T09:48:14Z
```

As you can see, the time format is RFC3339 with Coordinated Universal Time (UTC, identified by the trailing *Z*).

CHAPTER

FOUR

GRAPHICAL USER INTERFACE

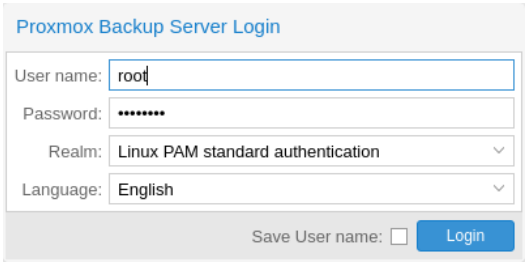
Proxmox Backup Server offers an integrated, web-based interface to manage the server. This means that you can carry out all administration tasks through your web browser, and that you don't have to worry about installing extra management tools. The web interface also provides a built-in console, so if you prefer the command line or need some extra control, you have this option.

The web interface can be accessed via https://youripaddress:8007. The default login is *root*, and the password is either the one specified during the installation process or the password of the root user, in case of installation on top of Debian.

4.1 Features

- Simple management interface for Proxmox Backup Server
- Monitoring of tasks, logs and resource usage
- Management of users, permissions, datastores, etc.
- Secure HTML5 console
- Support for multiple authentication sources
- Support for multiple languages
- Based on ExtJS 6.x JavaScript framework

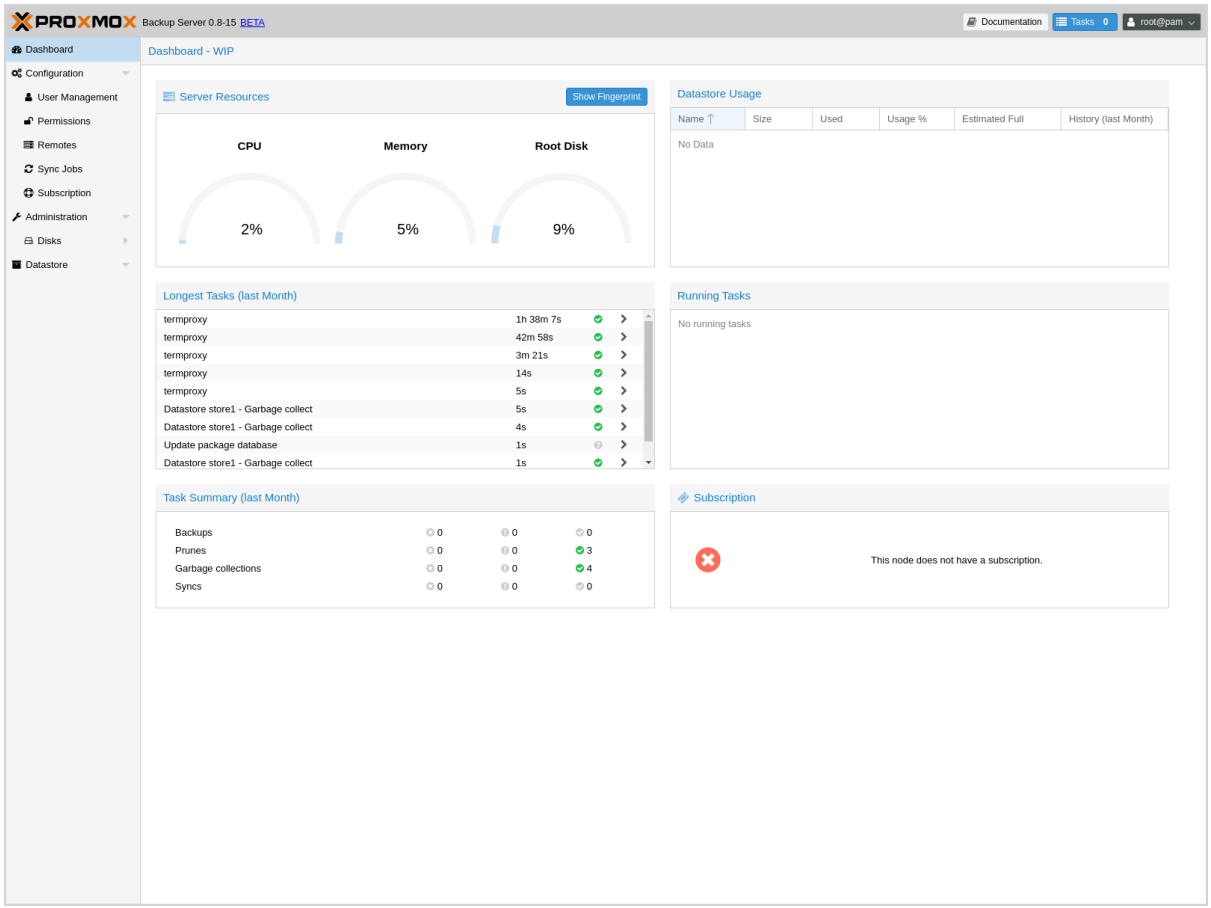
4.2 Login



When you connect to the web interface, you will first see the login window. Proxmox Backup Server supports various languages and authentication back ends (*Realms*), both of which can be selected here.

Note: For convenience, you can save the username on the client side, by selecting the "Save User name" checkbox at the bottom of the window.

4.3 GUI Overview



The Proxmox Backup Server web interface consists of 3 main sections:

- **Header**: At the top. This shows version information and contains buttons to view documentation, monitor running tasks, set the language, configure various display settings, and logout.
- **Sidebar**: On the left. This contains the administration options for the server.
- **Configuration Panel**: In the center. This contains the respective control interfaces for the administration options in the *Sidebar*.

4.4 Sidebar

In the sidebar, on the left side of the page, you can see various items relating to specific management activities.

4.4.1 Dashboard

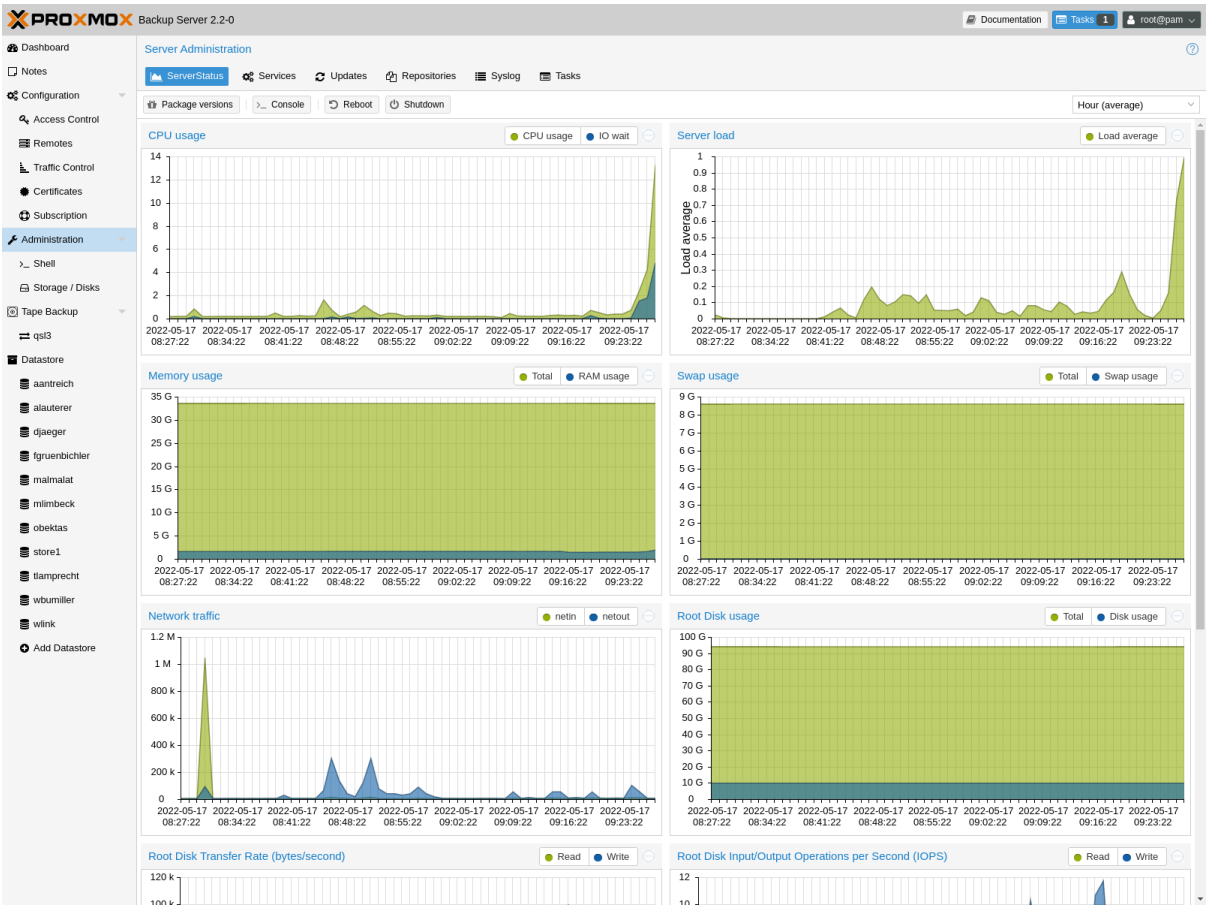
The Dashboard shows a summary of activity and resource usage on the server. Specifically, this displays hardware usage, a summary of previous and currently running tasks, and subscription information.

4.4.2 Configuration

The Configuration section contains some system options, such as time, network, WebAuthn, and HTTP proxy configuration. It also contains the following subsections:

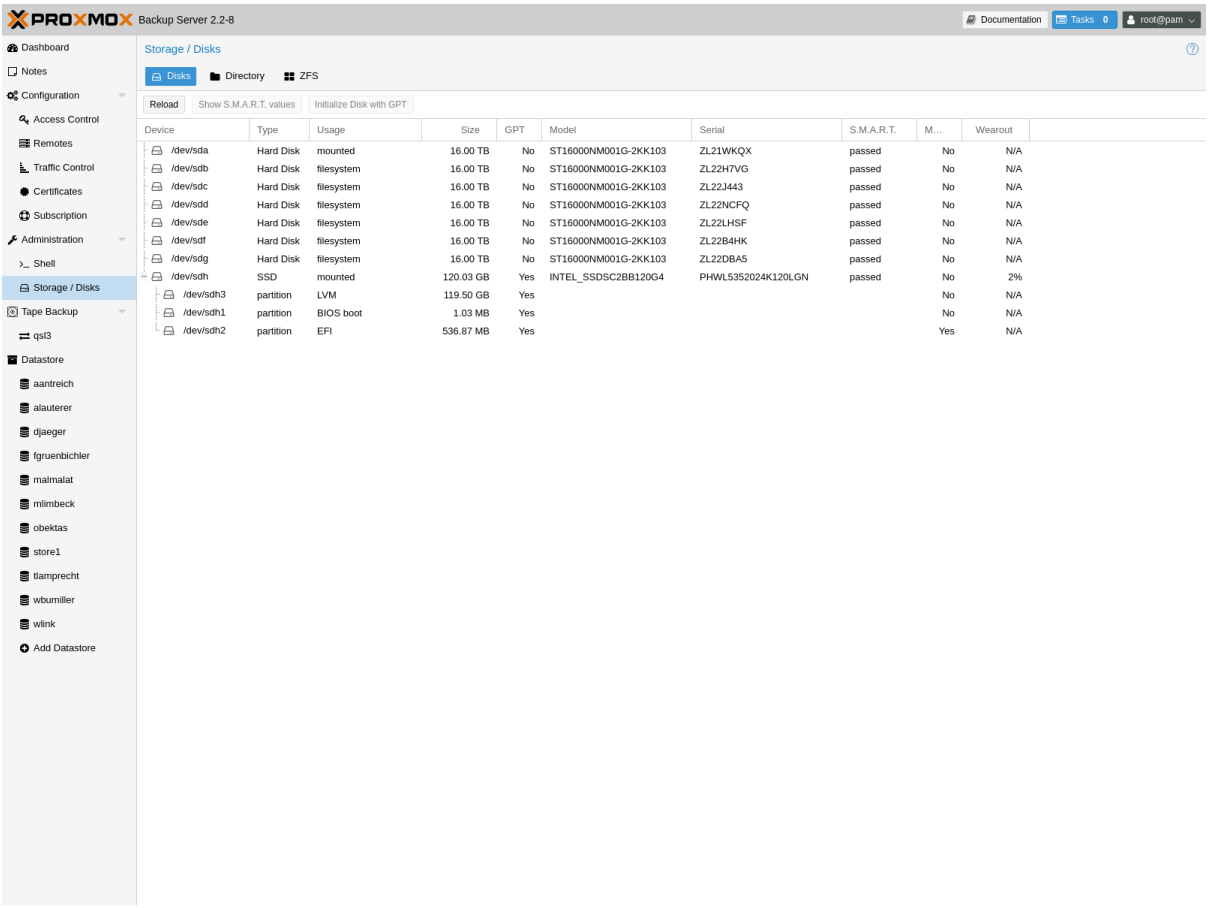
- **Access Control**: Add and manage users, API tokens, and the permissions associated with these items
- **Remotes**: Add, edit and remove remotes (see *Remote*)
- **Certificates**: Manage ACME accounts and create SSL certificates.
- **Subscription**: Upload a subscription key, view subscription status and access a text-based system report.

4.4.3 Administration



The Administration section contains a top panel, with further administration tasks and information. These are:

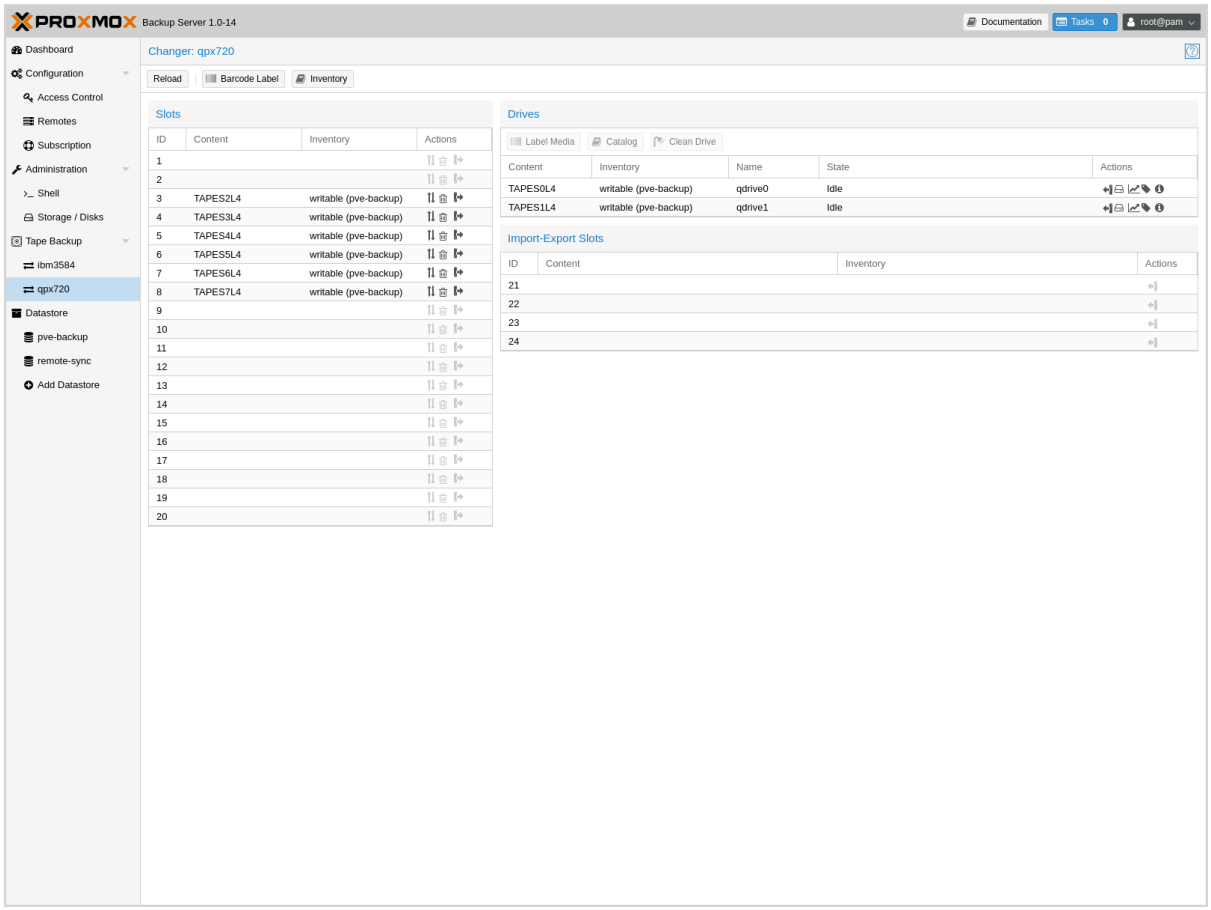
- **ServerStatus**: Provides access to the console, power options, and various resource usage statistics
- **Services**: Manage and monitor system services
- **Updates**: An interface for upgrading packages
- **Repositories**: An interface for configuring APT repositories
- **Syslog**: View log messages from the server
- **Tasks**: Task history with multiple filter options



The administration menu item also contains a disk management subsection:

- **Disks**: View information on available disks
  - **Directory**: Create and view information on *ext4* and *xfs* disks
  - **ZFS**: Create and view information on *ZFS* disks

4.4.4 Tape Backup

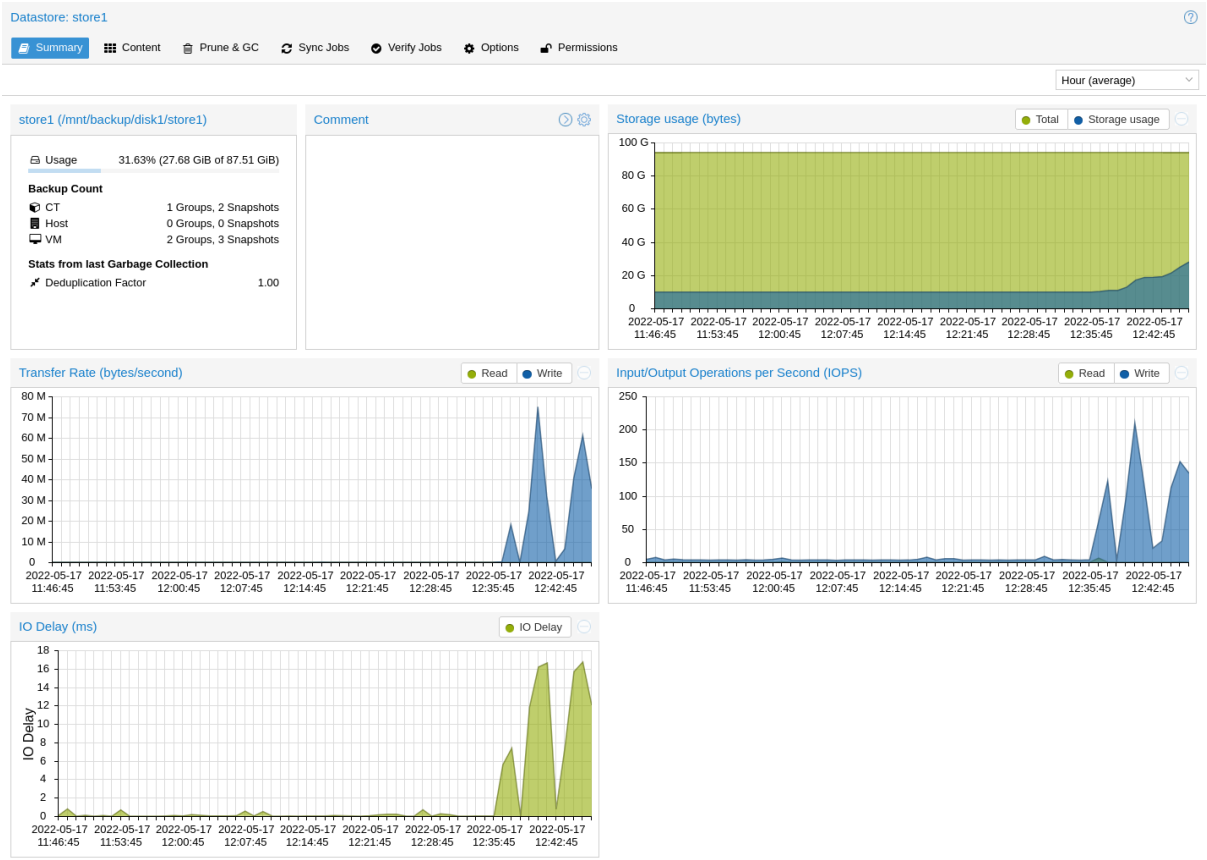


The *Tape Backup* section contains a top panel, with options for managing tape media sets, inventories, drives, changers, encryption keys, and the tape backup jobs itself. The tabs are as follows:

- **Content**: Information on the contents of the tape backup
- **Inventory**: Manage the tapes attached to the system
- **Changers**: Manage tape loading devices
- **Drives**: Manage drives used for reading and writing to tapes
- **Media Pools**: Manage logical pools of tapes
- **Encryption Keys**: Manage tape backup encryption keys
- **Backup Jobs**: Manage tape backup jobs

The section also contains a subsection per standalone drive and per changer, with a status and management view for those devices.

4.4.5 Datastore



The Datastore section contains interfaces for creating and managing datastores. It also contains a button for creating a new datastore on the server, as well as a subsection for each datastore on the system, in which you can use the top panel to view:

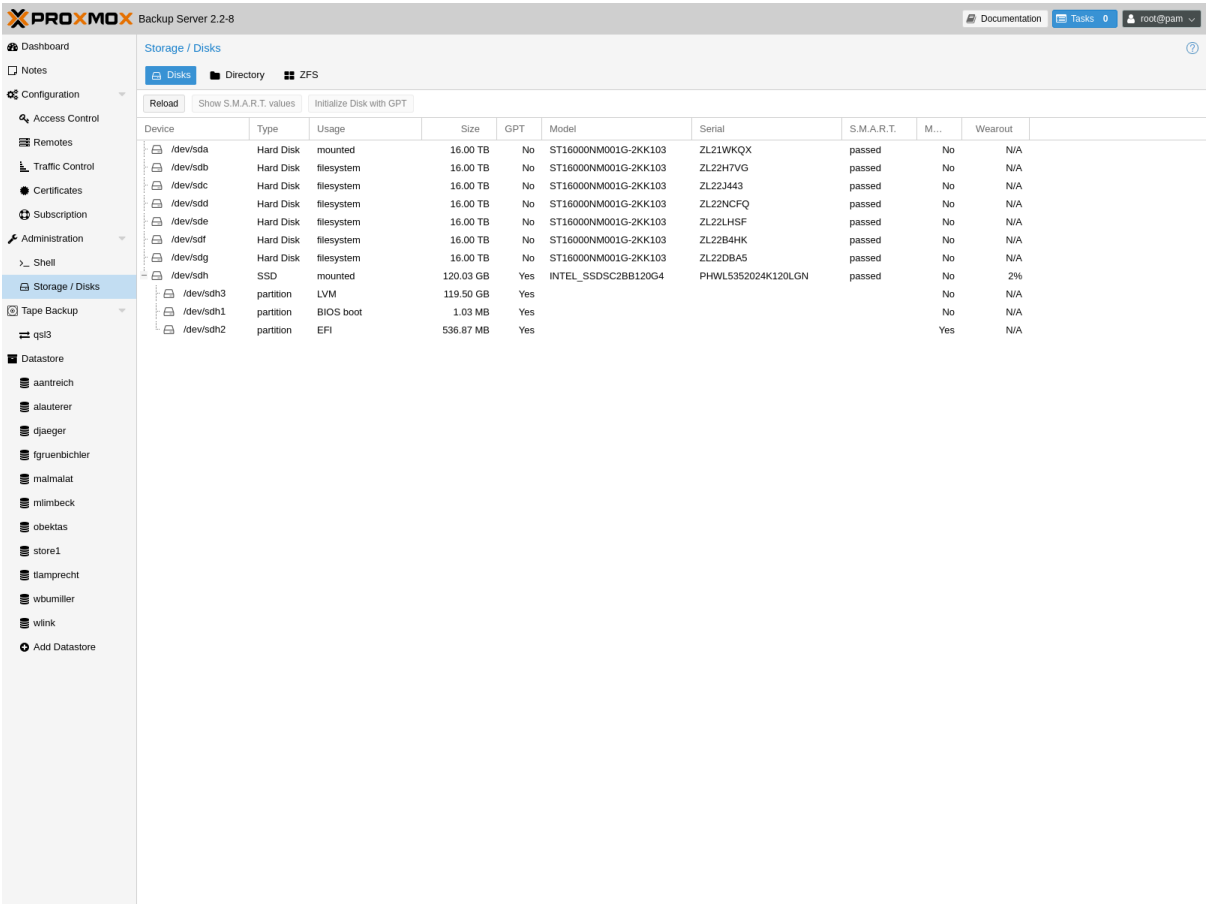
- **Summary**: Access a range of datastore usage statistics
- **Content**: Information on the datastore's backup groups and their respective contents
- **Prune & GC**: Schedule *pruning* and *garbage collection* operations, and run garbage collection manually
- **Sync Jobs**: Create, manage and run *Sync Jobs* from remote servers
- **Verify Jobs**: Create, manage and run *Verification* jobs on the datastore
- **Options**: Configure notification and verification settings
- **Permissions**: Manage permissions on the datastore

CHAPTER

FIVE

BACKUP STORAGE

5.1 Disk Management



Proxmox Backup Server comes with a set of disk utilities, which are accessed using the `disk` subcommand or the web interface. This subcommand allows you to initialize disks, create various filesystems, and get information about the disks.

To view the disks connected to the system, navigate to **Administration -> Storage/Disks** in the web interface or use the `list` subcommand of `disk`:

```
# proxmox-backup-manager disk list
```

name	used	gpt	disk-type	size	model	wearout	status
sda	lvm	1	hdd	34359738368	QEMU_HARDDISK	-	passed
sdb	unused	1	hdd	68719476736	QEMU_HARDDISK	-	passed

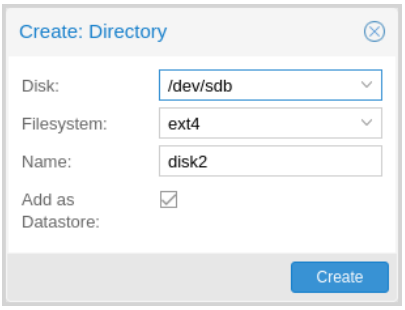
(continues on next page)

(continued from previous page)

sdc	unused	1	hdd	68719476736	QEMU_HARDDISK	-	passed

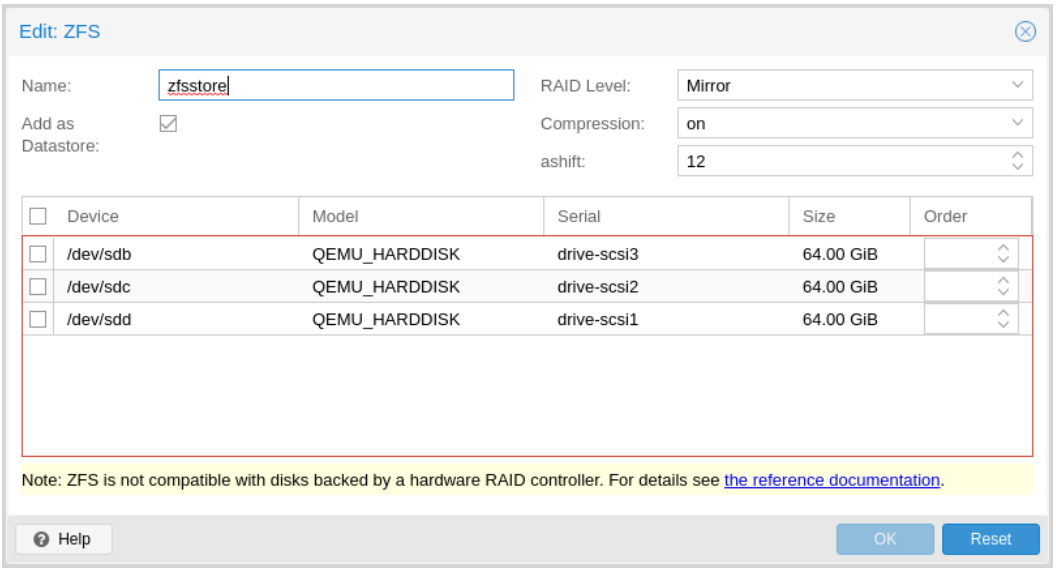
To initialize a disk with a new GPT, use the `initialize` subcommand:

```
# proxmox-backup-manager disk initialize sdX
```



You can create an `ext4` or `xfs` filesystem on a disk using `fs create`, or by navigating to **Administration -> Storage/Disks -> Directory** in the web interface and creating one from there. The following command creates an `ext4` filesystem and passes the `--add-datastore` parameter, in order to automatically create a datastore on the disk (in this case `sdd`). This will create a datastore at the location `/mnt/datastore/store1`:

```
# proxmox-backup-manager disk fs create store1 --disk sdd --filesystem ext4 --add-datastore␣
↪true
```



You can also create a `zpool` with various raid levels from **Administration -> Storage/Disks -> ZFS** in the web interface, or by using `zpool create`. The command below creates a mirrored `zpool` using two disks (`sdb` & `sdc`) and mounts it under `/mnt/datastore/zpool1`:

```
# proxmox-backup-manager disk zpool create zpool1 --devices sdb,sdc --raidlevel mirror
```

Note: You can also pass the `--add-datastore` parameter here, to automatically create a datastore from the disk.

You can use `disk fs list` and `disk zpool list` to keep track of your filesystems and zpools respectively.

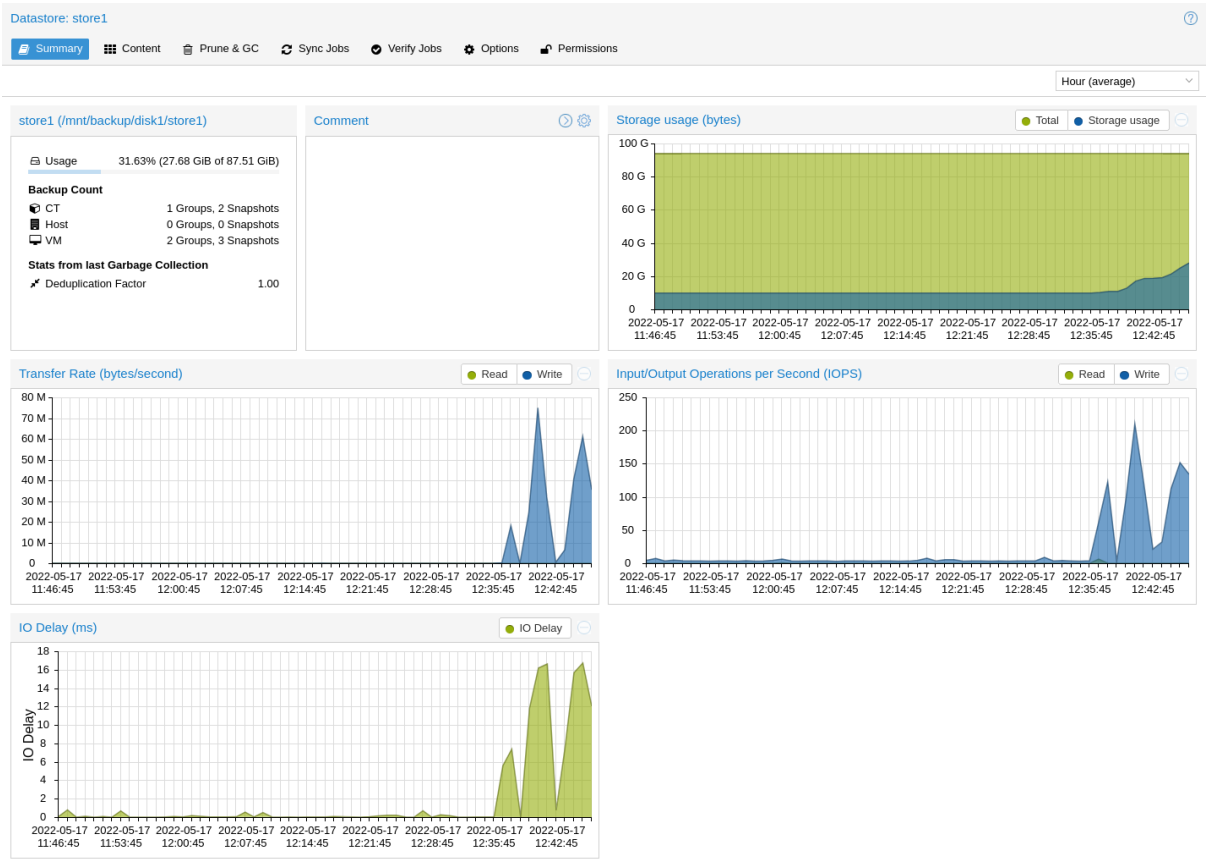
Proxmox Backup Server uses the package smartmontools. This is a set of tools used to monitor

and control the S.M.A.R.T. system for local hard disks. If a disk supports S.M.A.R.T. capability, and you have this enabled, you can display S.M.A.R.T. attributes from the web interface or by using the command:

```
# proxmox-backup-manager disk smart-attributes sdX
```

Note: This functionality may also be accessed directly through the use of the `smartctl` command, which comes as part of the smartmontools package (see `man smartctl` for more details).

5.2 Datastore



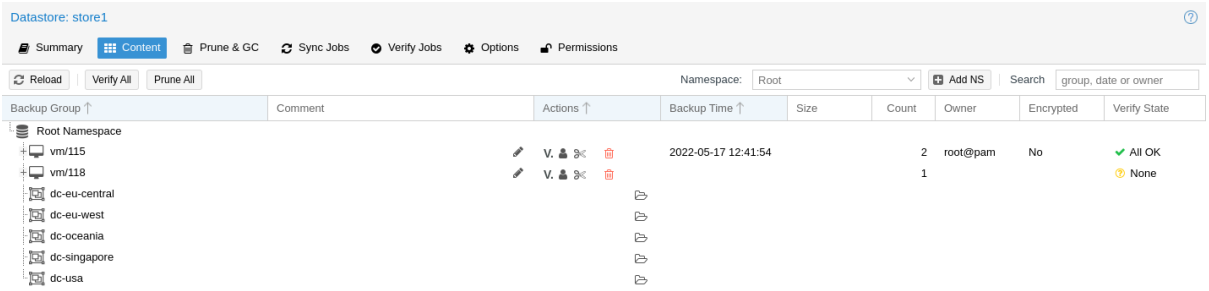
A datastore refers to a location at which backups are stored. The current implementation uses a directory inside a standard Unix file system (`ext4`, `xfs` or `zfs`) to store the backup data.

Datastores are identified by a simple *ID*. You can configure this when setting up the datastore. The configuration information for datastores is stored in the file `/etc/proxmox-backup/datastore.cfg`.

Note: The *File Layout* requires the file system to support at least *65538* subdirectories per directory. That number comes from the 2^{16} pre-created chunk namespace directories, and the `.` and `..` default directory entries. This requirement excludes certain filesystems and filesystem configurations from being supported for a datastore. For example, `ext3` as a whole or `ext4` with the

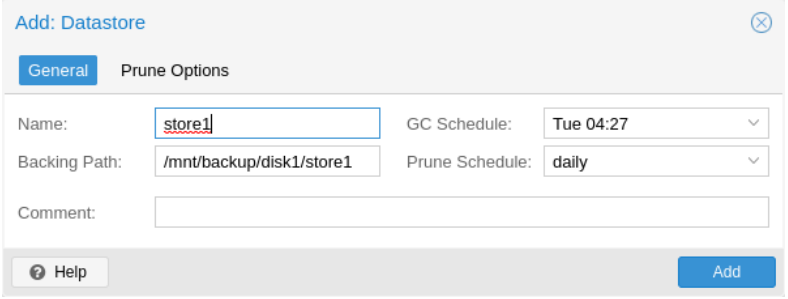
`dir_nlink` feature manually disabled.

5.2.1 Datastore Configuration



You can configure multiple datastores. A minimum of one datastore needs to be configured. The datastore is identified by a simple *name* and points to a directory on the filesystem. Each datastore also has associated retention settings of how many backup snapshots for each interval of `hourly`, `daily`, `weekly`, `monthly`, `yearly` as well as a time-independent number of backups to keep in that store. *Pruning and Removing Backups* and *garbage collection* can also be configured to run periodically, based on a configured schedule (see *Calendar Events*) per datastore.

Creating a Datastore



You can create a new datastore from the web interface, by clicking **Add Datastore** in the side menu, under the **Datastore** section. In the setup window:

- *Name* refers to the name of the datastore
- *Backing Path* is the path to the directory upon which you want to create the datastore
- *GC Schedule* refers to the time and intervals at which garbage collection runs
- *Prune Schedule* refers to the frequency at which pruning takes place
- *Prune Options* set the amount of backups which you would like to keep (see *Pruning and Removing Backups*).
- *Comment* can be used to add some contextual information to the datastore.

Alternatively you can create a new datastore from the command line. The following command creates a new datastore called `store1` on `/backup/disk1/store1`

```
# proxmox-backup-manager datastore create store1 /backup/disk1/store1
```

Managing Datastores

To list existing datastores from the command line, run:

```
# proxmox-backup-manager datastore list
```

name	path	comment
store1	/backup/disk1/store1	This is my default storage.

You can change the garbage collection and prune settings of a datastore, by editing the datastore from the GUI or by using the `update` subcommand. For example, the below command changes the garbage collection schedule using the `update` subcommand and prints the properties of the datastore with the `show` subcommand:

```
# proxmox-backup-manager datastore update store1 --gc-schedule 'Tue 04:27'
# proxmox-backup-manager datastore show store1
```

Name	Value
name	store1
path	/backup/disk1/store1
comment	This is my default storage.
gc-schedule	Tue 04:27
keep-last	7
prune-schedule	daily

Finally, it is possible to remove the datastore configuration:

```
# proxmox-backup-manager datastore remove store1
```

Note: The above command removes only the datastore configuration. It does not delete any data from the underlying directory.

File Layout

After creating a datastore, the following default layout will appear:

```
# ls -arilh /backup/disk1/store1
276493 -rw-r--r-- 1 backup backup       0 Jul  8 12:35 .lock
276490 drwxr-x--- 1 backup backup 1064960 Jul  8 12:35 .chunks
```

.lock is an empty file used for process locking.

The *.chunks* directory contains folders, starting from *0000* and increasing in hexadecimal values until *ffff*. These directories will store the chunked data, categorized by checksum, after a backup operation has been executed.

```
# ls -arilh /backup/disk1/store1/.chunks
545824 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 ffff
545823 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 fffe
415621 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 fffd
415620 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 fffc
353187 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 fffb
344995 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 fffa
144079 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 fff9
144078 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 fff8
144077 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 fff7
...
403180 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 000c
403179 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 000b
403177 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 000a
402530 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 0009
402513 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 0008
402509 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 0007
276509 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 0006
276508 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 0005
276507 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 0004
276501 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 0003
276499 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 0002
276498 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 0001
276494 drwxr-x--- 2 backup backup 4.0K Jul  8 12:35 0000
276489 drwxr-xr-x 3 backup backup 4.0K Jul  8 12:35 ..
276490 drwxr-x--- 1 backup backup 1.1M Jul  8 12:35 .
```

Once you've uploaded some backups or created namespaces, you may see the backup type (*ct*, *vm*, *host*) and the start of the namespace hierachy (*ns*).

5.2.2 Backup Namespaces

A datastore can host many backups, as long as the underlying storage is large enough and provides the performance required for a user's use case. However, without any hierarchy or separation, it's easy to run into naming conflicts, especially when using the same datastore for multiple Proxmox VE instances or multiple users.

The backup namespace hierarchy allows you to clearly separate different users or backup sources in general, avoiding naming conflicts and providing a well-organized backup content view.

Each namespace level can host any backup type, CT, VM or Host, but also other namespaces, up to a depth of 8 levels, where the root namespace is the first level.

Namespace Permissions

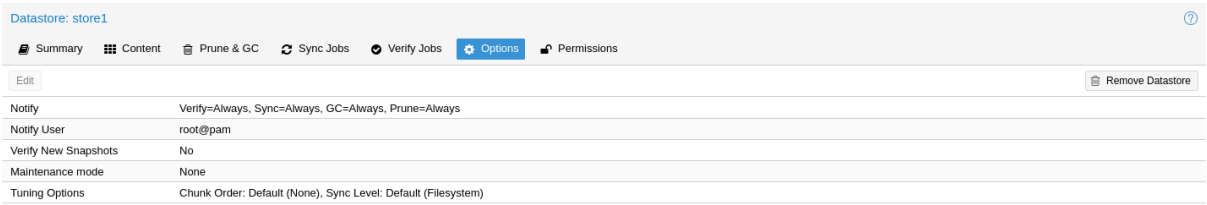
You can make the permission configuration of a datastore more fine-grained by setting permissions only on a specific namespace.

To view a datastore, you need a permission that has at least an *AUDIT*, *MODIFY*, *READ* or *BACKUP* privilege on any namespace it contains.

To create or delete a namespace, you require the modify privilege on the parent namespace. Thus, to initially create namespaces, you need to have a permission with an access role that includes the *MODIFY* privilege on the datastore itself.

For backup groups, the existing privilege rules still apply. You either need a privileged enough permission or to be the owner of the backup group; nothing changed here.

5.2.3 Options



There are a few per-datastore options:

- *Notifications*
- *Maintenance Mode*
- Verification of incoming backups

Tuning

There are some tuning related options for the datastore that are more advanced:

- `chunk-order`: Chunk order for verify & tape backup:

  You can specify the order in which Proxmox Backup Server iterates the chunks when doing a verify or backing up to tape. The two options are:

  - *inode* (default): Sorts the chunks by inode number of the filesystem before iterating over them. This should be fine for most storages, especially spinning disks.
  - *none* Iterates the chunks in the order they appear in the index file (.fidx/.didx). While this might slow down iterating on many slow storages, on very fast ones (for example: NVMEs) the collecting and sorting can take more time than gained through the sorted iterating.

  This option can be set with:

```
# proxmox-backup-manager datastore update <storename> --tuning 'chunk-order=none'
```

- `sync-level`: Datastore fsync level:

  You can set the level of syncing on the datastore for chunks, which influences the crash resistance of backups in case of a powerloss or hard shutoff. There are currently three levels:

  - *none* : Does not do any syncing when writing chunks. This is fast and normally OK, since the kernel eventually flushes writes onto the disk. The kernel sysctls *dirty_expire_centisecs* and *dirty_writeback_centisecs* are used to tune that behaviour, while the default is to flush old data after ~30s.
  - *filesystem* (default): This triggers a `syncfs(2)` after a backup, but before the task returns *OK*. This way it is ensured that the written backups are on disk. This is a good balance between speed and consistency. Note that the underlying storage device still needs to protect itself against powerloss to flush its internal ephemeral caches to the permanent storage layer.
  - *file* With this mode, a fsync is triggered on every chunk insertion, which makes sure each and every chunk reaches the disk as soon as possible. While this reaches the highest level of consistency, for many storages (especially slower ones) this comes at the cost of speed. For many users the *filesystem* mode is better suited, but for very fast storages this mode can be OK.

  This can be set with:

```
# proxmox-backup-manager datastore update <storename> --tuning 'sync-level=filesystem'
```

If you want to set multiple tuning options simultaneously, you can separate them with a comma, like this:

```
# proxmox-backup-manager datastore update <storename> --tuning 'sync-level=filesystem,chunk-
↪order=none'
```

5.3 Ransomware Protection & Recovery

Ransomware is a type of malware that encrypts files until a ransom is paid. Proxmox Backup Server includes features that help mitigate and recover from ransomware attacks by offering off-server and off-site synchronization and easy restoration from backups.

5.3.1 Built-in Protection

Proxmox Backup Server does not rewrite data for existing blocks. This means that a compromised Proxmox VE host or any other compromised system that uses the client to back up data cannot corrupt or modify existing backups in any way.

5.3.2 The 3-2-1 Rule with Proxmox Backup Server

The 3-2-1 rule is simple but effective in protecting important data from all sorts of threats, be it fires, natural disasters or attacks on your infrastructure by adversaries. In short, the rule states that one should create *3* backups on at least *2* different types of storage media, of which *1* copy is kept off-site.

Proxmox Backup Server provides tools for storing extra copies of backups in remote locations and on various types of media.

By setting up a remote Proxmox Backup Server, you can take advantage of the *remote sync jobs* feature and easily create off-site copies of your backups. This is recommended, since off-site instances are less likely to be infected by ransomware in your local network. You can configure sync jobs to not remove snapshots if they vanished on the remote-source to avoid that an attacker that took over the source can cause deletions of backups on the target hosts. If the source-host became victim of a ransomware attack, there is a good chance that sync jobs will fail, triggering an *error notification*.

It is also possible to create *tape backups* as a second storage medium. This way, you get an additional copy of your data on a different storage medium designed for long-term storage. Additionally, it can easily be moved around, be it to an off-site location or, for example, into an on-site fireproof vault for quicker access.

5.3.3 Restrictive User & Access Management

Proxmox Backup Server offers a comprehensive and fine-grained *user and access management* system. The *Datastore.Backup* privilege, for example, allows only to create, but not to delete or alter existing backups.

The best way to leverage this access control system is to:

- Use separate API tokens for each host or Proxmox VE Cluster that should be able to back data up to a Proxmox Backup Server.
- Configure only minimal permissions for such API tokens. They should only have a single permission that grants the *DataStore* access role on a very narrow ACL path that is restricted to a specific namespace on a specific datastore, for example */datastore/tank/pve-abc-cluster*.

Tip: One best practice to protect against ransomware is not to grant delete permissions, but to perform backup pruning directly on Proxmox Backup Server using *prune jobs*.

Please note that the same also applies for sync jobs. By limiting a sync user's or an access token's right to only write backups, not delete them, compromised clients cannot delete existing backups.

5.3.4 Ransomware Detection

A Proxmox Backup Server might still get compromised within insecure networks, if physical access to the server is attained, or due to weak or insufficiently protected credentials. If that happens, and your on-site backups are encrypted by ransomware, the SHA-256 checksums of the backups will not match the previously recorded ones anymore, hence, restoring the backup will fail.

To detect ransomware inside a compromised guest, it is recommended to frequently test restoring and booting backups. Make sure to restore to a new guest and not to overwrite your current guest. In the case of many backed-up guests, it is recommended to automate this restore testing. If this is not possible, restoring random samples from the backups periodically (for example, once a week or month), is advised'.

In order to be able to react quickly in case of a ransomware attack, it is recommended to regularly test restoring from your backups. Make sure to restore to a new guest and not to overwrite your current guest. Restoring many guests at once can be cumbersome, which is why it is advisable to automate this task and verify that your automated process works. If this is not feasible, it is recommended to restore random samples from your backups. While creating backups is important, verifying that they work is equally important. This ensures that you are able to react quickly in case of an emergency and keeps disruption of your services to a minimum.

Verification jobs can also assist in detecting a ransomware presence on a Proxmox Backup Server. Since verification jobs regularly check if all backups still match the checksums on record, they will start to fail if a ransomware starts to encrypt existing backups. Please be aware, that an advanced enough ransomware could circumvent this mechanism. Hence, consider verification jobs only as an additional, but not a sufficient protection measure.

5.3.5 General Prevention Methods and Best Practices

It is recommended to take additional security measures, apart from the ones offered by Proxmox Backup Server. These recommendations include, but are not limited to:

- Keeping the firmware and software up-to-date to patch exploits and vulnerabilities (such as Spectre or Meltdown).
- Following safe and secure network practices, for example using logging and monitoring tools and dividing your network so that infrastructure traffic and user or even public traffic are separated, for example by setting up VLANs.
- Set up a long-term retention. Since some ransomware might lay dormant a couple of days or weeks before starting to encrypt data, it can be that older, existing backups are compromised. Thus, it is important to keep at least a few backups over longer periods of time.

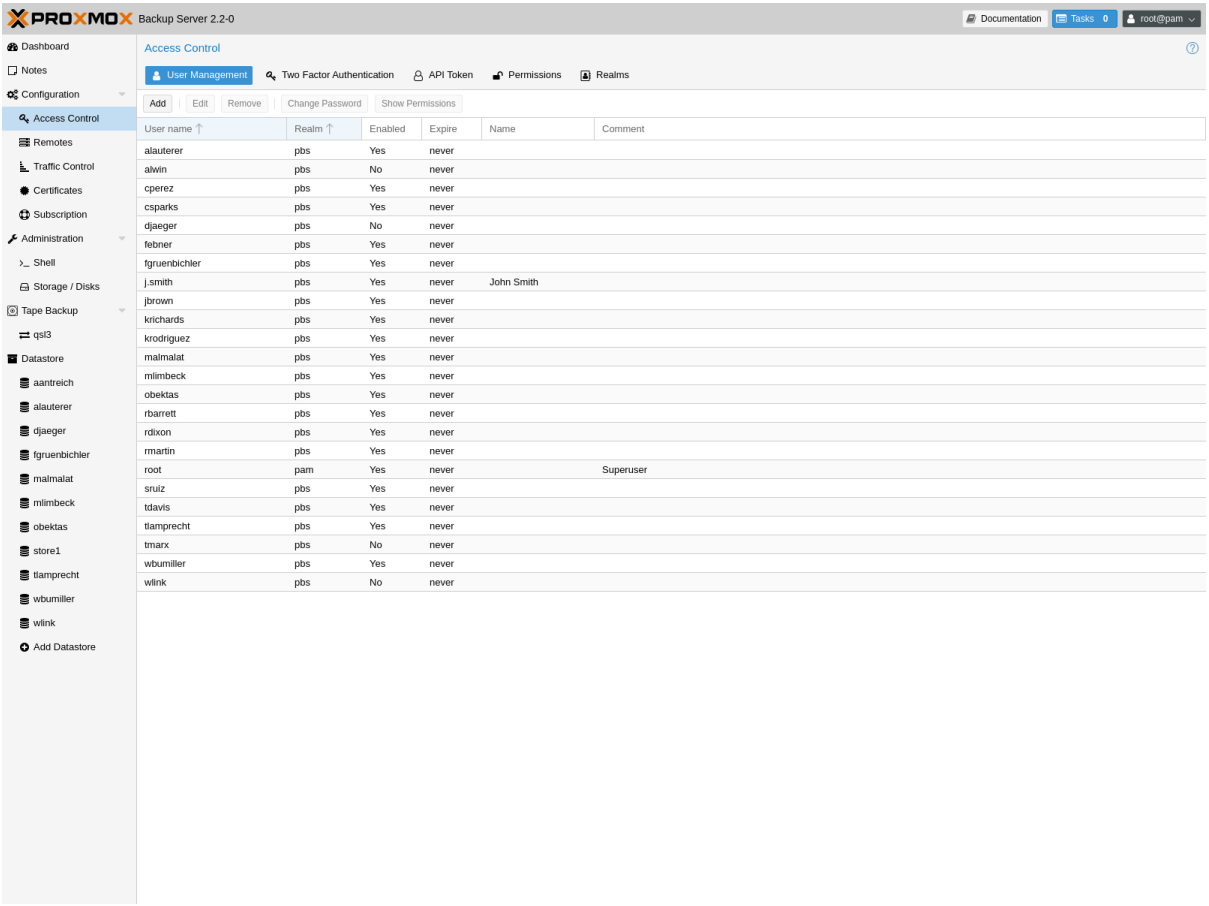
For more information on how to avoid ransomware attacks and what to do in case of a ransomware infection, see official government recommendations like CISA's (USA) guide or EU resources like ENSIA's Threat Landscape for Ransomware Attacks or nomoreransom.org.

CHAPTER

SIX

USER MANAGEMENT

6.1 User Configuration



Proxmox Backup Server supports several authentication realms, and you need to choose the realm when you add a new user. Possible realms are:

pam Linux PAM standard authentication. Use this if you want to authenticate as a Linux system user (users need to exist on the system).

pbs Proxmox Backup Server realm. This type stores hashed passwords in `/etc/proxmox-backup/shadow.json`.

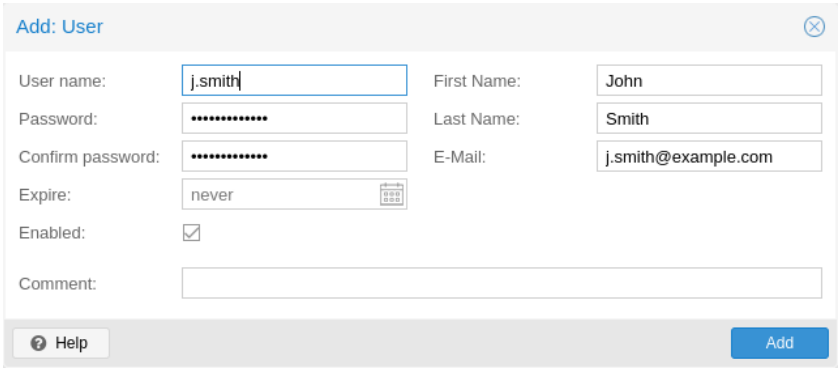
openid OpenID Connect server. Users can authenticate against an external OpenID Connect server.

ldap LDAP server. Users can authenticate against external LDAP servers.

After installation, there is a single user, `root@pam`, which corresponds to the Unix superuser. User configuration information is stored in the file `/etc/proxmox-backup/user.cfg`. You can use the `proxmox-backup-manager` command line tool to list or manipulate users:

```
# proxmox-backup-manager user list
```

userid	enable	expire	firstname	lastname	email	comment
root@pam	1					Superuser



The superuser has full administration rights on everything, so it's recommended to add other users with less privileges. You can add a new user with the `user create` subcommand or through the web interface, under the **User Management** tab of **Configuration -> Access Control**. The `create` subcommand lets you specify many options like `--email` or `--password`. You can update or change any user properties using the `user update` subcommand later (**Edit** in the GUI):

```
# proxmox-backup-manager user create john@pbs --email john@example.com
# proxmox-backup-manager user update john@pbs --firstname John --lastname Smith
# proxmox-backup-manager user update john@pbs --comment "An example user."
```

The resulting user list looks like this:

```
# proxmox-backup-manager user list
```

userid	enable	expire	firstname	lastname	email	comment
john@pbs	1		John	Smith	john@example.com	An example user.
root@pam	1					Superuser

Newly created users do not have any permissions. Please read the *Access Control* section to learn how to set access permissions.

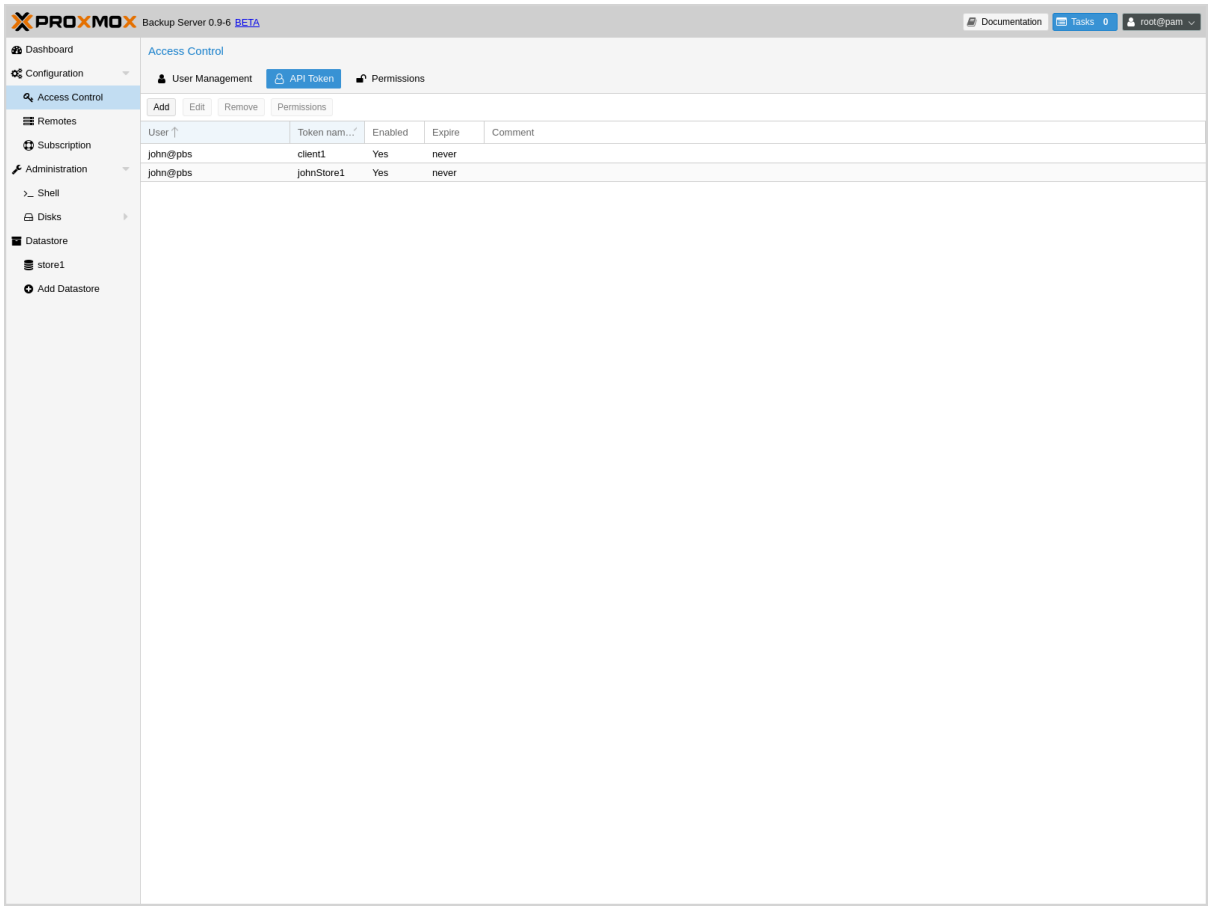
You can disable a user account by setting `--enable` to `0`:

```
# proxmox-backup-manager user update john@pbs --enable 0
```

Or completely remove a user with:

```
# proxmox-backup-manager user remove john@pbs
```

6.2 API Tokens

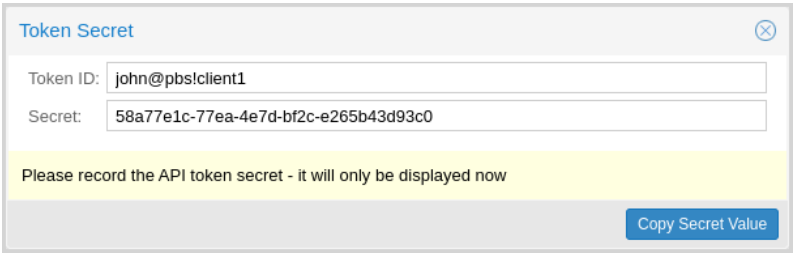


Any authenticated user can generate API tokens, which can in turn be used to configure various clients, instead of directly providing the username and password.

API tokens serve two purposes:

1. Easy revocation in case client gets compromised
2. Limit permissions for each client/token within the users' permission

An API token consists of two parts: an identifier consisting of the user name, the realm and a tokenname (`user@realm!tokenname`), and a secret value. Both need to be provided to the client in place of the user ID (`user@realm`) and the user password, respectively.



The API token is passed from the client to the server by setting the `Authorization` HTTP header with method `PBSAPIToken` to the value `TOKENID:TOKENSECRET`.

You can generate tokens from the GUI or by using `proxmox-backup-manager`:

```
# proxmox-backup-manager user generate-token john@pbs client1
Result: {
```

(continues on next page)

(continued from previous page)

```
  "tokenid": "john@pbs!client1",
  "value": "d63e505a-e3ec-449a-9bc7-1da610d4ccde"
}
```

Note: The displayed secret value needs to be saved, since it cannot be displayed again after generating the API token.

The `user list-tokens` sub-command can be used to display tokens and their metadata:

```
# proxmox-backup-manager user list-tokens john@pbs
```

tokenid	enable	expire	comment
john@pbs!client1	1		

Similarly, the `user delete-token` subcommand can be used to delete a token again.

Newly generated API tokens don't have any permissions. Please read the next section to learn how to set access permissions.

6.3 Access Control

By default, new users and API tokens do not have any permissions. Instead you need to specify what is allowed and what is not.

Proxmox Backup Server uses a role- and path-based permission management system. An entry in the permissions table allows a user, group or token to take on a specific role when accessing an 'object' or 'path'. This means that such an access rule can be represented as a triple of '(path, user, role)', '(path, group, role)' or '(path, token, role)', with the role containing a set of allowed actions, and the path representing the target of these actions.

6.3.1 Privileges

Privileges are the building blocks of access roles. They are internally used to enforce the actual permission checks in the API.

We currently support the following privileges:

Sys.Audit Sys.Audit allows a user to know about the system and its status.

Sys.Modify Sys.Modify allows a user to modify system-level configuration and apply updates.

Sys.PowerManagement Sys.Modify allows a user to power-off and reboot the system.

Datastore.Audit Datastore.Audit allows a user to know about a datastore, including reading the configuration entry and listing its contents.

Datastore.Allocate Datastore.Allocate allows a user to create or delete datastores.

Datastore.Modify Datastore.Modify allows a user to modify a datastore and its contents, and to create or delete namespaces inside a datastore.

Datastore.Read Datastore.Read allows a user to read arbitrary backup contents, independent of the backup group owner.

Datastore.Verify Allows verifying the backup snapshots in a datastore.

Datastore.Backup Datastore.Backup allows a user create new backup snapshots and also provides the privileges of Datastore.Read and Datastore.Verify, but only if the backup group is owned by the user or one of its tokens.

Datastore.Prune Datastore.Prune allows a user to delete snapshots, but additionally requires backup ownership.

Permissions.Modify Permissions.Modify allows a user to modify ACLs.

> **Note:** A user can always configure privileges for their own API tokens, as they will be limited by the users privileges anyway.

Remote.Audit Remote.Audit allows a user to read the remote and the sync configuration entries.

Remote.Modify Remote.Modify allows a user to modify the remote configuration.

Remote.Read Remote.Read allows a user to read data from a configured *Remote*.

Sys.Console Sys.Console allows a user to access the system's console, note that for all but *root@pam* a valid system login is still required.

Tape.Audit Tape.Audit allows a user to read the configuration and status of tape drives, changers and backups.

Tape.Modify Tape.Modify allows a user to modify the configuration of tape drives, changers and backups.

Tape.Write Tape.Write allows a user to write to a tape media.

Tape.Read Tape.Read allows a user to read tape backup configuration and contents from a tape media.

Realm.Allocate Realm.Allocate allows a user to view, create, modify and delete authentication realms for users.

6.3.2 Access Roles

An access role combines one or more privileges into something that can be assigned to a user or API token on an object path.

Currently, there are only built-in roles, meaning you cannot create your own, custom role.

The following roles exist:

NoAccess Disable Access - nothing is allowed.

Admin Can do anything, on the object path assigned.

Audit Can view the status and configuration of things, but is not allowed to change settings.

DatastoreAdmin Can do anything on *existing* datastores.

DatastoreAudit Can view datastore metrics, settings and list content. But is not allowed to read the actual data.

DatastoreReader Can inspect a datastore's or namespace's content and do restores.

DatastoreBackup Can backup and restore owned backups.

DatastorePowerUser Can backup, restore, and prune *owned* backups.

RemoteAdmin Can do anything on remotes.

RemoteAudit Can view remote settings.

RemoteSyncOperator Is allowed to read data from a remote.

TapeAdmin Can do anything related to tape backup.

TapeAudit Can view tape-related metrics, configuration and status.

TapeOperator Can do tape backup and restore, but cannot change any configuration.

TapeReader Can read and inspect tape configuration and media content.

6.3.3 Objects and Paths

Access permissions are assigned to objects, such as a datastore, namespace or some system resources.

We use filesystem-like paths to address these objects. These paths form a natural tree, and permissions of higher levels (shorter paths) can optionally be propagated down within this hierarchy.

Paths can be templated, meaning they can refer to the actual id of a configuration entry. When an API call requires permissions on a templated path, the path may contain references to parameters of the API call. These references are specified in curly brackets.

Some examples are:

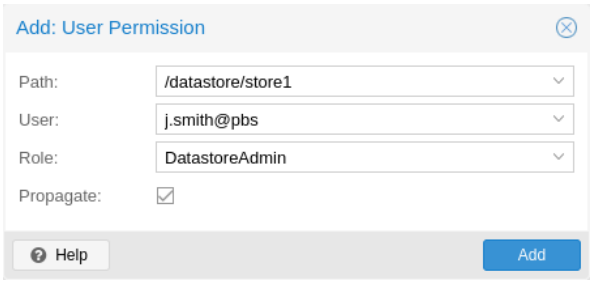
- */datastore*: Access to *all* datastores on a Proxmox Backup server
- */datastore/{store}*: Access to a specific datastore on a Proxmox Backup server
- */datastore/{store}/{ns}*: Access to a specific namespace on a specific datastore
- */remote*: Access to all remote entries
- */system/network*: Access to configure the host network
- */tape/*: Access to tape devices, pools and jobs
- */access/users*: User administration
- */access/openid/{id}*: Administrative access to a specific OpenID Connect realm

Inheritance

As mentioned earlier, object paths form a file system like tree, and permissions can be inherited by objects down that tree through the propagate flag, which is set by default. We use the following inheritance rules:

- Permissions for API tokens are always limited to those of the user.
- Permissions on deeper, more specific levels replace those inherited from an upper level.

6.3.4 Configuration & Management



Access permission information is stored in `/etc/proxmox-backup/acl.cfg`. The file contains 5 fields, separated using a colon (':') as a delimiter. A typical entry takes the form:

```
acl:1:/datastore:john@pbs:DatastoreBackup
```

The data represented in each field is as follows:

1. `acl` identifier
2. A `1` or `0`, representing whether propagation is enabled or disabled, respectively
3. The object on which the permission is set. This can be a specific object (single datastore, remote, etc.) or a top level object, which with propagation enabled, represents all children of the object also.
4. The user(s)/token(s) for which the permission is set
5. The role being set

You can manage permissions via **Configuration -> Access Control -> Permissions** in the web interface. Likewise, you can use the `acl` subcommand to manage and monitor user permissions from the command line. For example, the command below will add the user `john@pbs` as a **DatastoreAdmin** for the datastore `store1`, located at `/backup/disk1/store1`:

```
# proxmox-backup-manager acl update /datastore/store1 DatastoreAdmin --auth-id john@pbs
```

You can list the ACLs of each user/token using the following command:

```
# proxmox-backup-manager acl list
```

ugid	path	propagate	roleid
john@pbs	/datastore/store1	1	DatastoreAdmin

A single user/token can be assigned multiple permission sets for different datastores.

Note: Naming convention is important here. For datastores on the host, you must use the convention `/datastore/{storename}`. For example, to set permissions for a datastore mounted at `/mnt/backup/disk4/store2`, you would use `/datastore/store2` for the path. For remote stores, use the convention `/remote/{remote}/{storename}`, where `{remote}` signifies the name of the remote (see *Remote* below) and `{storename}` is the name of the datastore on the remote.

6.3.5 API Token Permissions

API token permissions are calculated based on ACLs containing their ID, independently of those of their corresponding user. The resulting permission set on a given path is then intersected with that of the corresponding user.

In practice this means:

1. API tokens require their own ACL entries
2. API tokens can never do more than their corresponding user

6.3.6 Effective Permissions

To calculate and display the effective permission set of a user or API token, you can use the `proxmox-backup-manager user permission` command:

```
# proxmox-backup-manager user permissions john@pbs --path /datastore/store1
Privileges with (*) have the propagate flag set

Path: /datastore/store1
- Datastore.Audit (*)
- Datastore.Backup (*)
- Datastore.Modify (*)
```

(continues on next page)

(continued from previous page)

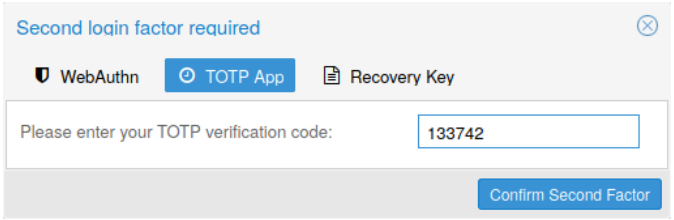
```
- Datastore.Prune (*)
- Datastore.Read (*)
- Datastore.Verify (*)

# proxmox-backup-manager acl update /datastore/store1 DatastoreBackup --auth-id 'john@pbs!
↪client1'
# proxmox-backup-manager user permissions 'john@pbs!client1' --path /datastore/store1
Privileges with (*) have the propagate flag set

Path: /datastore/store1
- Datastore.Backup (*)
```

6.4 Two-Factor Authentication

6.4.1 Introduction

With simple authentication, only a password (single factor) is required to successfully claim an identity (authenticate), for example, to be able to log in as *root@pam* on a specific instance of Proxmox Backup Server. In this case, if the password gets leaked or stolen, anybody can use it to log in - even if they should not be allowed to do so.

With two-factor authentication (TFA), a user is asked for an additional factor to verify their authenticity. Rather than relying on something only the user knows (a password), this extra factor requires something only the user has, for example, a piece of hardware (security key) or a secret saved on the user's smartphone. This prevents a remote user from gaining unauthorized access to an account, as even if they have the password, they will not have access to the physical object (second factor).



6.4.2 Available Second Factors

You can set up multiple second factors, in order to avoid a situation in which losing your smartphone or security key locks you out of your account permanently.

Proxmox Backup Server supports three different two-factor authentication methods:

- TOTP (Time-based One-Time Password). A short code derived from a shared secret and the current time, it changes every 30 seconds.
- WebAuthn (Web Authentication). A general standard for authentication. It is implemented by various security devices, like hardware keys or trusted platform modules (TPM) from a computer or smart phone.
- Single use Recovery Keys. A list of keys which should either be printed out and locked in a secure place or saved digitally in an electronic vault. Each key can be used only once. These are perfect for ensuring that you are not locked out, even if all of your other second factors are lost or corrupt.

6.4.3 Setup

TOTP

There is no server setup required. Simply install a TOTP app on your smartphone (for example, FreeOTP) and use the Proxmox Backup Server web-interface to add a TOTP factor.

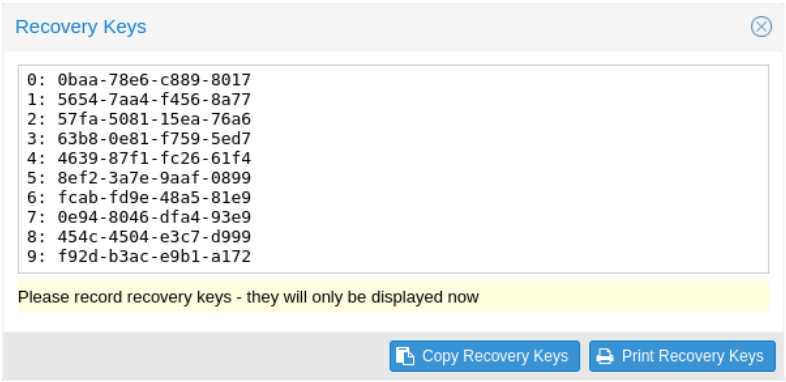
WebAuthn

For WebAuthn to work, you need to have two things:

- A trusted HTTPS certificate (for example, by using Let's Encrypt). While it probably works with an untrusted certificate, some browsers may warn or refuse WebAuthn operations if it is not trusted.
- Setup the WebAuthn configuration (see **Configuration -> Other** in the Proxmox Backup Server web interface). This can be auto-filled in most setups.

Once you have fulfilled both of these requirements, you can add a WebAuthn configuration in the **Two Factor Authentication** tab of the **Access Control** panel.

Recovery Keys



Recovery key codes do not need any preparation; you can simply create a set of recovery keys in the **Two Factor Authentication** tab of the **Access Control** panel.

Note: There can only be one set of single-use recovery keys per user at any time.

6.4.4 TFA and Automated Access

Two-factor authentication is only implemented for the web-interface. You should use *API Tokens* for all other use cases, especially non-interactive ones (for example, adding a Proxmox Backup Server to Proxmox VE as a storage).

6.5 Authentication Realms

6.5.1 LDAP

Proxmox Backup Server can utilize external LDAP servers for user authentication. To achieve this, a realm of the type `ldap` has to be configured.

In LDAP, users are uniquely identified by their domain (`dn`). For instance, in the following LDIF dataset, the user `user1` has the unique domain `uid=user1,ou=People,dc=ldap-test,dc=com`:

```
# user1 of People at ldap-test.com
dn: uid=user1,ou=People,dc=ldap-test,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
uid: user1
cn: Test User 1
sn: Testers
description: This is the first test user.
```

In in similar manner, Proxmox Backup Server uses user identifiers (`userid`) to uniquely identify users. Thus, it is necessary to establish a mapping between a Proxmox Backup Server `userid` and an LDAP `dn`. This mapping is established by the `user-attr` configuration parameter - it contains the name of the LDAP attribute containing a valid Proxmox Backup Server user identifier.

For the example above, setting `user-attr` to `uid` will have the effect that the user `user1@<realm-name>` will be mapped to the LDAP entity `uid=user1,ou=People,dc=ldap-test,dc=com`. On user login, Proxmox Backup Server will perform a *subtree search* under the configured Base Domain (`base-dn`) to query the user's `dn`. Once the `dn` is known, an LDAP bind operation is performed to authenticate the user against the LDAP server.

As not all LDAP servers allow *anonymous* search operations, it is possible to configure a bind domain (`bind-dn`) and a bind password (`password`). If set, Proxmox Backup Server will bind to the LDAP server using these credentials before performing any search operations.

A full list of all configuration parameters can be found at *domains.cfg*.

Note: In order to allow a particular user to authenticate using the LDAP server, you must also add them as a user of that realm in Proxmox Backup Server. This can be carried out automatically with syncing.

User Synchronization in LDAP realms

It is possible to automatically sync users for LDAP-based realms, rather than having to add them to Proxmox VE manually. Synchronization options can be set in the LDAP realm configuration dialog window in the GUI and via the `proxmox-backup-manager ldap create/update` command. User synchronization can started in the GUI at Configuration > Access Control > Realms by selecting a realm and pressing the *Sync* button. In the sync dialog, some of the default options set in the realm configuration can be overridden. Alternatively, user synchronization can also be started via the `proxmox-backup-manager ldap sync` command.

CHAPTER

SEVEN

BACKUP CLIENT USAGE

The command line client for Proxmox Backup Server is called **`proxmox-backup-client`**.

7.1 Backup Repository Locations

The client uses the following format to specify a datastore repository on the backup server (where username is specified in the form of user@realm):

[[username@]server[:port]:]datastore

The default value for `username` is `root@pam`. If no server is specified, the default is the local host (`localhost`).

You can specify a port if your backup server is only reachable on a non-default port (for example, with NAT and port forwarding configurations).

Note that if the server uses an IPv6 address, you have to write it with square brackets (for example, *[fe80::01]*).

You can pass the repository with the `--repository` command line option, or by setting the `PBS_REPOSITORY` environment variable.

Below are some examples of valid repositories and their corresponding real values:

Example	User	Host:Port	Datastore
mydatastore	`root@pam`	localhost:8007	mydatastore
myhostname:mydatastore	`root@pam`	myhostname:8007	mydatastore
user@pbs@myhostname:mydatastore	`user@pbs`	myhostname:8007	mydatastore
user@pbs!token@host:store	`user@pbs!token`	host:8007	store
192.168.55.55:1234:mydatastore	`root@pam`	192.168.55.55:1234	mydatastore
[ff80::51]:mydatastore	`root@pam`	[ff80::51]:8007	mydatastore
[ff80::51]:1234:mydatastore	`root@pam`	[ff80::51]:1234	mydatastore

7.2 Environment Variables

PBS_REPOSITORY The default backup repository.

PBS_PASSWORD When set, this value is used as the password for the backup server. You can also set this to an API token secret.

PBS_PASSWORD_FD, PBS_PASSWORD_FILE, PBS_PASSWORD_CMD Like PBS_PASSWORD, but read data from an open file descriptor, a file name or from the *stdout* of a command, respectively. The first defined environment variable from the order above is preferred.

PBS_ENCRYPTION_PASSWORD When set, this value is used to access the secret encryption key (if protected by password).

PBS_ENCRYPTION_PASSWORD_FD, PBS_ENCRYPTION_PASSWORD_FILE, PBS_ENCRYPTION_PASSWORD_CMD Like PBS_ENCRYPTION_PASSWORD, but read data from an open file descriptor, a file name or from the *stdout* of a command, respectively. The first defined environment variable from the order above is preferred.

PBS_FINGERPRINT When set, this value is used to verify the server certificate (only used if the system CA certificates cannot validate the certificate).

ALL_PROXY When set, the client uses the specified HTTP proxy for all connections to the backup server. Currently only HTTP proxies are supported. Valid proxy configurations have the following format: *[http://][user:password@]<host>[:port]*. Default *port* is 1080, if not otherwise specified.

Note: The recommended solution for shielding hosts is using tunnels such as wireguard, instead of using an HTTP proxy.

Note: Passwords must be valid UTF-8 and may not contain newlines. For your convenience, Proxmox Backup Server only uses the first line as password, so you can add arbitrary comments after the first newline.

7.3 Output Format

Most commands that produce output support the `--output-format` parameter. This accepts the following values:

text Text format (default). Structured data is rendered as a table.

json JSON (single line).

json-pretty JSON (multiple lines, nicely formatted).

Also, the following environment variables can modify output behavior:

PROXMOX_OUTPUT_FORMAT Defines the default output format.

PROXMOX_OUTPUT_NO_BORDER If set (to any value), do not render table borders.

PROXMOX_OUTPUT_NO_HEADER If set (to any value), do not render table headers.

Note: The `text` format is designed to be human readable, and not meant to be parsed by automation tools. Please use the `json` format if you need to process the output.

7.4 Creating Backups

This section explains how to create a backup from within the machine. This can be a physical host, a virtual machine, or a container. Such backups may contain file and image archives. There are no restrictions in this case.

Note: If you want to backup virtual machines or containers on Proxmox VE, see *Proxmox VE Integration*.

For the following example, you need to have a backup server set up, have working credentials, and know the repository name. In the following examples, we use `backup-server:store1`.

```
# proxmox-backup-client backup root.pxar:/ --repository backup-server:store1
Starting backup: host/elsa/2019-12-03T09:35:01Z
Client name: elsa
skip mount point: "/boot/efi"
skip mount point: "/dev"
skip mount point: "/run"
skip mount point: "/sys"
Uploaded 12129 chunks in 87 seconds (564 MB/s).
End Time: 2019-12-03T10:36:29+01:00
```

This will prompt you for a password, then upload a file archive named `root.pxar` containing all the files in the `/` directory.

Caution: Please note that proxmox-backup-client does not automatically include mount points. Instead, you will see a short `skip mount point` message for each of them. The idea is to create a separate file archive for each mounted disk. You can explicitly include them using the `--include-dev` option (i.e. `--include-dev /boot/efi`). You can use this option multiple times for each mount point that should be included.

The `--repository` option can get quite long and is used by all commands. You can avoid having to enter this value by setting the environment variable `PBS_REPOSITORY`. Note that if you would like this to remain set over multiple sessions, you should instead add the below line to your `.bashrc` file.

```
# export PBS_REPOSITORY=backup-server:store1
```

After this, you can execute all commands without having to specify the `--repository` option.

A single backup is allowed to contain more than one archive. For example, if you want to back up two disks mounted at `/mnt/disk1` and `/mnt/disk2`:

```
# proxmox-backup-client backup disk1.pxar:/mnt/disk1 disk2.pxar:/mnt/disk2
```

This creates a backup of both disks.

If you want to use a namespace for the backup target, you can add the *--ns* parameter:

```
# proxmox-backup-client backup disk1.pxar:/mnt/disk1 disk2.pxar:/mnt/disk2 --ns a/b/c
```

The backup command takes a list of backup specifications, which include the archive name on the server, the type of the archive, and the archive source at the client. The format is:

<archive-name>.<type>:<source-path>

Common types are `.pxar` for file archives and `.img` for block device images. To create a backup of a block device, run the following command:

```
# proxmox-backup-client backup mydata.img:/dev/mylvm/mydata
```

7.4.1 Excluding Files/Directories from a Backup

Sometimes it is desired to exclude certain files or directories from a backup archive. To tell the Proxmox Backup client when and how to ignore files and directories, place a text file named `.pxarexclude` in the filesystem hierarchy. Whenever the backup client encounters such a file in a directory, it interprets each line as a glob match pattern for files and directories that are to be excluded from the backup.

The file must contain a single glob pattern per line. Empty lines and lines starting with `#` (indicating a comment) are ignored. A `!` at the beginning of a line reverses the glob match pattern from an exclusion to an explicit inclusion. This makes it possible to exclude all entries in a directory except for a few single files/subdirectories. Lines ending in `/` match only on directories. The directory containing the `.pxarexclude` file is considered to be the root of the given patterns. It is only possible to match files in this directory and its subdirectories.

Note: Patterns without a leading `/` will also match in subdirectories, while patterns with a leading `/` will only match in the current directory.

`\` is used to escape special glob characters. `?` matches any single character. `*` matches any character, including an empty string. `**` is used to match current directory and subdirectories. For example, with the pattern `**/*.tmp`, it would exclude all files ending in `.tmp` within a directory and its subdirectories. `[...]` matches a single character from any of the provided characters within the brackets. `[!...]` does the complementary and matches any single character not contained within the brackets. It is also possible to specify ranges with two characters separated by `-`. For example, `[a-z]` matches any lowercase alphabetic character, and `[0-9]` matches any single digit.

The order of the glob match patterns defines whether a file is included or excluded, that is to say, later entries override earlier ones. This is also true for match patterns encountered deeper down the directory tree, which can override a previous exclusion.

Note: Excluded directories will **not** be read by the backup client. Thus, a `.pxarexclude` file in an excluded subdirectory will have no effect. `.pxarexclude` files are treated as regular files and will be included in the backup archive.

For example, consider the following directory structure:

```
# ls -aR folder
folder/:
.  ..  .pxarexclude  subfolder0  subfolder1

folder/subfolder0:
.  ..  file0  file1  file2  file3  .pxarexclude

folder/subfolder1:
.  ..  file0  file1  file2  file3
```

The different `.pxarexclude` files contain the following:

```
# cat folder/.pxarexclude
/subfolder0/file1
/subfolder1/*
!/subfolder1/file2
```

```
# cat folder/subfolder0/.pxarexclude
file3
```

This would exclude `file1` and `file3` in `subfolder0` and all of `subfolder1` except `file2`.

Restoring this backup will result in:

```
ls -aR restored
restored/:
.  ..  .pxarexclude  subfolder0  subfolder1

restored/subfolder0:
.  ..  file0  file2  .pxarexclude

restored/subfolder1:
.  ..  file2
```

7.5 Encryption

Proxmox Backup supports client-side encryption with AES-256 in GCM mode. To set this up, you first need to create an encryption key:

```
# proxmox-backup-client key create my-backup.key
Encryption Key Password: **************
```

The key is password protected by default. If you do not need this extra protection, you can also create it without a password:

```
# proxmox-backup-client key create /path/to/my-backup.key --kdf none
```

Having created this key, it is now possible to create an encrypted backup, by passing the `--keyfile` parameter, with the path to the key file.

```
# proxmox-backup-client backup etc.pxar:/etc --keyfile /path/to/my-backup.key
Password: *********
Encryption Key Password: **************
...
```

Note: If you do not specify the name of the backup key, the key will be created in the default location `~/.config/proxmox-backup/encryption-key.json`. `proxmox-backup-client` will also search this location by default, in case the `--keyfile` parameter is not specified.

You can avoid entering the passwords by setting the environment variables `PBS_PASSWORD` and `PBS_ENCRYPTION_PASSWORD`.

7.5.1 Using a Master Key to Store and Recover Encryption Keys

You can also use `proxmox-backup-client key` to create an RSA public/private key pair, which can be used to store an encrypted version of the symmetric backup encryption key alongside each backup and recover it later.

To set up a master key:

1. Create an encryption key for the backup:

   ```
   # proxmox-backup-client key create
   creating default key at: "~/.config/proxmox-backup/encryption-key.json"
   Encryption Key Password: **********
   ...
   ```

   The resulting file will be saved to `~/.config/proxmox-backup/encryption-key.json`.

2. Create an RSA public/private key pair:

   ```
   # proxmox-backup-client key create-master-key
   Master Key Password: *********
   ...
   ```

This will create two files in your current directory, `master-public.pem` and `master-private.pem`.

3. Import the newly created `master-public.pem` public certificate, so that `proxmox-backup-client` can find and use it upon backup.

```
# proxmox-backup-client key import-master-pubkey /path/to/master-public.pem
Imported public master key to "~/.config/proxmox-backup/master-public.pem"
```

4. With all these files in place, run a backup job:

```
# proxmox-backup-client backup etc.pxar:/etc
```

The key will be stored in your backup, under the name `rsa-encrypted.key`.

Note: The `--keyfile` parameter can be excluded, if the encryption key is in the default path. If you specified another path upon creation, you must pass the `--keyfile` parameter.

---

5. To test that everything worked, you can restore the key from the backup:

```
# proxmox-backup-client restore /path/to/backup/ rsa-encrypted.key /path/to/target
```

Note: You should not need an encryption key to extract this file. However, if a key exists at the default location (`~/.config/proxmox-backup/encryption-key.json`) the program will prompt you for an encryption key password. Simply moving `encryption-key.json` out of this directory will fix this issue.

---

6. Then, use the previously generated master key to decrypt the file:

```
# proxmox-backup-client key import-with-master-key /path/to/target --master-keyfile /
↪path/to/master-private.pem --encrypted-keyfile /path/to/rsa-encrypted.key
Master Key Password: ******
New Password: ******
Verify Password: ******
```

7. The target file will now contain the encryption key information in plain text. The success of this can be confirmed by passing the resulting `json` file, with the `--keyfile` parameter, when decrypting files from the backup.

Warning: Without their key, backed up files will be inaccessible. Thus, you should keep keys ordered and in a place that is separate from the contents being backed up. It can happen, for example, that you back up an entire system, using a key on that system. If the system then becomes inaccessible for any reason and needs to be restored, this will not be possible, as the encryption key will be lost along with the broken system.

It is recommended that you keep your master key safe, but easily accessible, in order for quick disaster recovery. For this reason, the best place to store it is in your password manager, where it is immediately recoverable. As a backup to this, you should also save the key to a USB drive and store that in a secure place. This way, it is detached from any system, but is still easy to recover from, in case of emergency. Finally, in preparation for the worst case scenario, you should also consider keeping a paper copy of your master key locked away in a safe place. The `paperkey` subcommand can be used to create a QR encoded version of your master key. The following command sends the output of the `paperkey` command to a text file, for easy printing.

```
proxmox-backup-client key paperkey --output-format text > qrkey.txt
```

7.6 Restoring Data

The regular creation of backups is a necessary step in avoiding data loss. More importantly, however, is the restoration. It is good practice to perform periodic recovery tests to ensure that you can access the data in case of disaster.

First, you need to find the snapshot which you want to restore. The snapshot list command provides a list of all the snapshots on the server:

```
# proxmox-backup-client snapshot list
```

snapshot	size	files
host/elsa/2019-12-03T09:30:15Z	51788646825	root.pxar catalog.pcat1 index.json
host/elsa/2019-12-03T09:35:01Z	51790622048	root.pxar catalog.pcat1 index.json

```
...
```

Tip: List will by default only output the backup snapshots of the root namespace itself. To list backups from another namespace use the `--ns <ns>` option

You can inspect the catalog to find specific files.

```
# proxmox-backup-client catalog dump host/elsa/2019-12-03T09:35:01Z
...
d "./root.pxar.didx/etc/cifs-utils"
l "./root.pxar.didx/etc/cifs-utils/idmap-plugin"
d "./root.pxar.didx/etc/console-setup"
...
```

The restore command lets you restore a single archive from the backup.

```
# proxmox-backup-client restore host/elsa/2019-12-03T09:35:01Z root.pxar /target/path/
```

To get the contents of any archive, you can restore the `index.json` file in the repository to the target path '-'. This will dump the contents to the standard output.

```
# proxmox-backup-client restore host/elsa/2019-12-03T09:35:01Z index.json -
```

7.6.1 Interactive Restores

If you only want to restore a few individual files, it is often easier to use the interactive recovery shell.

```
# proxmox-backup-client catalog shell host/elsa/2019-12-03T09:35:01Z root.pxar
Starting interactive shell
pxar:/ > ls
bin         boot        dev         etc         home        lib         lib32
...
```

The interactive recovery shell is a minimal command line interface that utilizes the metadata stored in the catalog to quickly list, navigate and search for files in a file archive. To restore files, you can select them individually or match them with a glob pattern.

Using the catalog for navigation reduces the overhead considerably because only the catalog needs to be downloaded and, optionally, decrypted. The actual chunks are only accessed if the metadata in the catalog is insufficient or for the actual restore.

Similar to common UNIX shells, `cd` and `ls` are the commands used to change working directory and list directory contents in the archive. `pwd` shows the full path of the current working directory with respect to the archive root.

The ability to quickly search the contents of the archive is a commonly required feature. That's where the catalog is most valuable. For example:

```
pxar:/ > find etc/**/*.txt --select
"/etc/X11/rgb.txt"
pxar:/ > list-selected
etc/**/*.txt
pxar:/ > restore-selected /target/path
...
```

This will find and print all files ending in `.txt` located in `etc/` or its subdirectories, and add the corresponding pattern to the list for subsequent restores. `list-selected` shows these patterns and `restore-selected` finally restores all files in the archive matching the patterns to `/target/path` on the local host. This will scan the whole archive.

The `restore` command can be used to restore all the files contained within the backup archive. This is most helpful when paired with the `--pattern <glob>` option, as it allows you to restore all files matching a specific pattern. For example, if you wanted to restore configuration files located in `/etc`, you could do the following:

```
pxar:/ > restore target/ --pattern etc/**/*.conf
...
```

The above will scan through all the directories below `/etc` and restore all files ending in `.conf`.

7.6.2 Mounting of Archives via FUSE

The *FUSE* implementation for the pxar archive allows you to mount a file archive as a read-only filesystem to a mount point on your host.

```
# proxmox-backup-client mount host/backup-client/2020-01-29T11:29:22Z root.pxar /mnt/
↪mountpoint
# ls /mnt/mountpoint
bin   dev  home  lib32  libx32      media  opt   root  sbin  sys  usr
boot  etc  lib   lib64  lost+found  mnt    proc  run   srv   tmp  var
```

This allows you to access the full contents of the archive in a seamless manner.

Note: As the FUSE connection needs to fetch and decrypt chunks from the backup server's datastore, this can cause some additional network and CPU load on your host, depending on the operations you perform on the mounted filesystem.

To unmount the filesystem, use the `umount` command on the mount point:

```
# umount /mnt/mountpoint
```

7.7 Login and Logout

The client tool prompts you to enter the login password as soon as you want to access the backup server. The server checks your credentials and responds with a ticket that is valid for two hours. The client tool automatically stores that ticket and uses it for further requests to this server.

You can also manually trigger this login/logout using the login and logout commands:

```
# proxmox-backup-client login
Password: **********
```

To remove the ticket, issue a logout:

```
# proxmox-backup-client logout
```

7.8 Changing the Owner of a Backup Group

By default, the owner of a backup group is the user which was used to originally create that backup group (or in the case of sync jobs, `root@pam`). This means that if a user `mike@pbs` created a backup, another user `john@pbs` can not be used to create backups in that same backup group. In case you want to change the owner of a backup, you can do so with the below command, using a user that has `Datastore.Modify` privileges on the datastore.

```
# proxmox-backup-client change-owner vm/103 john@pbs
```

This can also be done from within the web interface, by navigating to the *Content* section of the datastore that contains the backup group and selecting the user icon under the *Actions* column. Common cases for this could be to change the owner of a sync job from `root@pam`, or to repurpose a backup group.

7.9 Pruning and Removing Backups

You can manually delete a backup snapshot using the `forget` command:

```
# proxmox-backup-client snapshot forget <snapshot>
```

Caution: This command removes all archives in this backup snapshot. They will be inaccessible and *unrecoverable*.

Don't forget to add the namespace `--ns` parameter if you want to forget a snapshot that is contained in the root namespace:

```
# proxmox-backup-client snapshot forget <snapshot> --ns <ns>
```

Although manual removal is sometimes required, the `prune` command is normally used to systematically delete older backups. Prune lets you specify which backup snapshots you want to keep. The following retention options are available:

`--keep-last <N>` Keep the last <N> backup snapshots.

`--keep-hourly <N>` Keep backups for the last <N> hours. If there is more than one backup for a single hour, only the latest is kept.

`--keep-daily <N>` Keep backups for the last <N> days. If there is more than one backup for a single day, only the latest is kept.

`--keep-weekly <N>` Keep backups for the last <N> weeks. If there is more than one backup for a single week, only the latest is kept.

> **Note:** Weeks start on Monday and end on Sunday. The software uses the ISO week date system and handles weeks at the end of the year correctly.

`--keep-monthly <N>` Keep backups for the last <N> months. If there is more than one backup for a single month, only the latest is kept.

`--keep-yearly <N>` Keep backups for the last <N> years. If there is more than one backup for a single year, only the latest is kept.

The retention options are processed in the order given above. Each option only covers backups within its time period. The next option does not take care of already covered backups. It will only consider older backups.

Unfinished and incomplete backups will be removed by the prune command unless they are newer than the last successful backup. In this case, the last failed backup is retained.

```
# proxmox-backup-client prune <group> --keep-daily 7 --keep-weekly 4 --keep-monthly 3
```

You can use the `--dry-run` option to test your settings. This only shows the list of existing snapshots and what actions prune would take.

```
# proxmox-backup-client prune host/elsa --dry-run --keep-daily 1 --keep-weekly 3
```

snapshot	keep
host/elsa/2019-12-04T13:20:37Z	1
host/elsa/2019-12-03T09:35:01Z	0
host/elsa/2019-11-22T11:54:47Z	1
host/elsa/2019-11-21T12:36:25Z	0
host/elsa/2019-11-10T10:42:20Z	1

Note: Neither the `prune` command nor the `forget` command free space in the chunk-store. The chunk-store still contains the data blocks. To free space you need to perform *Garbage Collection*.

It is also possible to protect single snapshots from being pruned or deleted:

```
# proxmox-backup-client snapshot protected update <snapshot> true
```

This will set the protected flag on the snapshot and prevent pruning or manual deletion of this snapshot until the flag is removed again with:

```
# proxmox-backup-client snapshot protected update <snapshot> false
```

When a group with a protected snapshot is deleted, only the non-protected ones are removed, and the rest will remain.

Note: This flag will not be synced when using pull or sync jobs. If you want to protect a synced snapshot, you have to do this again manually on the target backup server.

7.10 Garbage Collection

The `prune` command removes only the backup index files, not the data from the datastore. This task is left to the garbage collection command. It is recommended to carry out garbage collection on a regular basis.

The garbage collection works in two phases. In the first phase, all data blocks that are still in use are marked. In the second phase, unused data blocks are removed.

Note: This command needs to read all existing backup index files and touches the complete chunk-store. This can take a long time depending on the number of chunks and the speed of the underlying disks.

Note: The garbage collection will only remove chunks that haven't been used for at least one day (exactly 24h 5m). This grace period is necessary because chunks in use are marked by touching the chunk which updates the `atime` (access time) property. Filesystems are mounted with the `relatime` option by default. This results in a better performance by only updating the `atime` property if the last access has been at least 24 hours ago. The downside is that touching a chunk within these 24 hours will not always update its `atime` property.

Chunks in the grace period will be logged at the end of the garbage collection task as *Pending removals*.

```
# proxmox-backup-client garbage-collect
starting garbage collection on store store2
Start GC phase1 (mark used chunks)
Start GC phase2 (sweep unused chunks)
percentage done: 1, chunk count: 219
percentage done: 2, chunk count: 453
...
percentage done: 99, chunk count: 21188
Removed bytes: 411368505
Removed chunks: 203
Original data bytes: 327160886391
Disk bytes: 52767414743 (16 %)
Disk chunks: 21221
Average chunk size: 2486565
TASK OK
```

Garbage collection can also be scheduled using `proxmox-backup-manager` or from the Proxmox Backup Server's web interface.

7.11 Benchmarking

The backup client also comes with a benchmarking tool. This tool measures various metrics relating to compression and encryption speeds. If a Proxmox Backup repository (remote or local) is specified, the TLS upload speed will get measured too.

You can run a benchmark using the `benchmark` subcommand of `proxmox-backup-client`:

Note: The TLS speed test is only included if a *backup server repository is specified*.

```
# proxmox-backup-client benchmark
Uploaded 1517 chunks in 5 seconds.
Time per request: 3309 microseconds.
TLS speed: 1267.41 MB/s
SHA256 speed: 2066.73 MB/s
Compression speed: 775.11 MB/s
Decompress speed: 1233.35 MB/s
AES256/GCM speed: 3688.27 MB/s
Verify speed: 783.43 MB/s
```

Name	Value
TLS (maximal backup upload speed)	1267.41 MB/s (103%)
SHA256 checksum computation speed	2066.73 MB/s (102%)
ZStd level 1 compression speed	775.11 MB/s (103%)
ZStd level 1 decompression speed	1233.35 MB/s (103%)
Chunk verification speed	783.43 MB/s (103%)

(continues on next page)

(continued from previous page)

```
| AES256 GCM encryption speed | 3688.27 MB/s (101%) |
```

Note: The percentages given in the output table correspond to a comparison against a Ryzen 7 2700X.

You can also pass the `--output-format` parameter to output stats in `json`, rather than the default table format.

CHAPTER

EIGHT

PROXMOX VE INTEGRATION

Proxmox Backup Server can be integrated into a Proxmox VE standalone or cluster setup, by adding it as a storage in Proxmox VE.

See also the Proxmox VE Storage - Proxmox Backup Server section of the Proxmox VE Administration Guide for Proxmox VE specific documentation.

8.1 Using the Proxmox VE Web-Interface

Proxmox VE has native API and web interface integration of Proxmox Backup Server as of Proxmox VE 6.3.

A Proxmox Backup Server can be added under `Datacenter -> Storage`.

8.2 Using the Proxmox VE Command-Line

You need to define a new storage with type 'pbs' on your Proxmox VE node. The following example uses `store2` as the storage's name, and assumes the server address is `localhost` and you want to connect as `user1@pbs`.

```
# pvesm add pbs store2 --server localhost --datastore store2
# pvesm set store2 --username user1@pbs --password <secret>
```

Note: If you would rather not enter your password as plain text, you can pass the `--password` parameter, without any arguments. This will cause the program to prompt you for a password upon entering the command.

If your backup server uses a self signed certificate, you need to add the certificate fingerprint to the configuration. You can get the fingerprint by running the following command on the backup server:

```
# proxmox-backup-manager cert info | grep Fingerprint
Fingerprint (sha256): 64:d3:ff:3a:50:38:53:5a:9b:f7:50:...:ab:fe
```

Please add that fingerprint to your configuration to establish a trust relationship:

```
# pvesm set store2 --fingerprint  64:d3:ff:3a:50:38:53:5a:9b:f7:50:...:ab:fe
```

After that, you should be able to view storage status with:

```
# pvesm status --storage store2
Name             Type     Status           Total            Used       Available        %
store2            pbs     active      3905109820      1336687816      2568422004   34.23%
```

Having added the Proxmox Backup Server datastore to Proxmox VE, you can backup VMs and containers in the same way you would for any other storage device within the environment (see Proxmox VE Admin Guide: Backup and Restore.

CHAPTER

NINE

PXAR COMMAND LINE TOOL

`pxar` is a command line utility for creating and manipulating archives in the *Proxmox File Archive Format (.pxar)*. It is inspired by casync file archive format, which caters to a similar use-case. The `.pxar` format is adapted to fulfill the specific needs of the Proxmox Backup Server, for example, efficient storage of hard links. The format is designed to reduce the required storage on the server by achieving a high level of deduplication.

9.1 Creating an Archive

Run the following command to create an archive of a folder named `source`:

```
# pxar create archive.pxar /path/to/source
```

This will create a new archive called `archive.pxar` with the contents of the `source` folder.

Note: `pxar` will not overwrite any existing archives. If an archive with the same name is already present in the target folder, the creation will fail.

By default, `pxar` will skip certain mount points and will not follow device boundaries. This design decision is based on the primary use case of creating archives for backups. It makes sense to ignore the contents of certain temporary or system specific files in a backup. To alter this behavior and follow device boundaries, use the `--all-file-systems` flag.

It is possible to exclude certain files and/or folders from the archive by passing the `--exclude` parameter with `gitignore`-style match patterns.

For example, you can exclude all files ending in `.txt` from the archive by running:

```
# pxar create archive.pxar /path/to/source --exclude '**/*.txt'
```

Be aware that the shell itself will try to expand glob patterns before invoking `pxar`. In order to avoid this, all globs have to be quoted correctly.

It is possible to pass the `--exclude` parameter multiple times, in order to match more than one pattern. This allows you to use more complex file inclusion/exclusion behavior. However, it is recommended to use `.pxarexclude` files instead for such cases.

For example you might want to exclude all `.txt` files except a specific one from the archive. This would be achieved via the negated match pattern, prefixed by `!`. All the glob patterns are relative to the `source` directory.

```
# pxar create archive.pxar /path/to/source --exclude '**/*.txt' --exclude '!/folder/file.txt'
```

Note: The order of the glob match patterns matters, as later ones override earlier ones. Permutations of the same patterns lead to different results.

`pxar` will store the list of glob match patterns passed as parameters via the command line, in a file called `.pxarexclude-cli`, at the root of the archive. If a file with this name is already present in the source folder during archive creation, this file is not included in the archive, and the file containing the new patterns is added to the archive instead. The original file is not altered.

A more convenient and persistent way to exclude files from the archive is by placing the glob match patterns in `.pxarexclude` files. It is possible to create and place these files in any directory of the filesystem tree. These files must contain one pattern per line, and later patterns override earlier ones. The patterns control file exclusions of files present within the given directory or further below it in the tree. The behavior is the same as described in *Creating Backups*.

9.2 Extracting an Archive

An existing archive, `archive.pxar`, is extracted to a `target` directory with the following command:

```
# pxar extract archive.pxar /path/to/target
```

If no target is provided, the contents of the archive is extracted to the current working directory.

In order to restore only parts of an archive, single files, and/or folders, it is possible to pass the corresponding glob match patterns as additional parameters or to use the patterns stored in a file:

```
# pxar extract etc.pxar /restore/target/etc --pattern '**/*.conf'
```

The above example restores all `.conf` files encountered in any of the sub-folders in the archive `etc.pxar` to the target `/restore/target/etc`. A path to the file containing match patterns can be specified using the `--files-from` parameter.

9.3 List the Contents of an Archive

To display the files and directories contained in an archive `archive.pxar`, run the following command:

```
# pxar list archive.pxar
```

This displays the full path of each file or directory with respect to the archive's root.

9.4 Mounting an Archive

`pxar` allows you to mount and inspect the contents of an archive via FUSE. In order to mount an archive named `archive.pxar` to the mount point `/mnt`, run the command:

```
# pxar mount archive.pxar /mnt
```

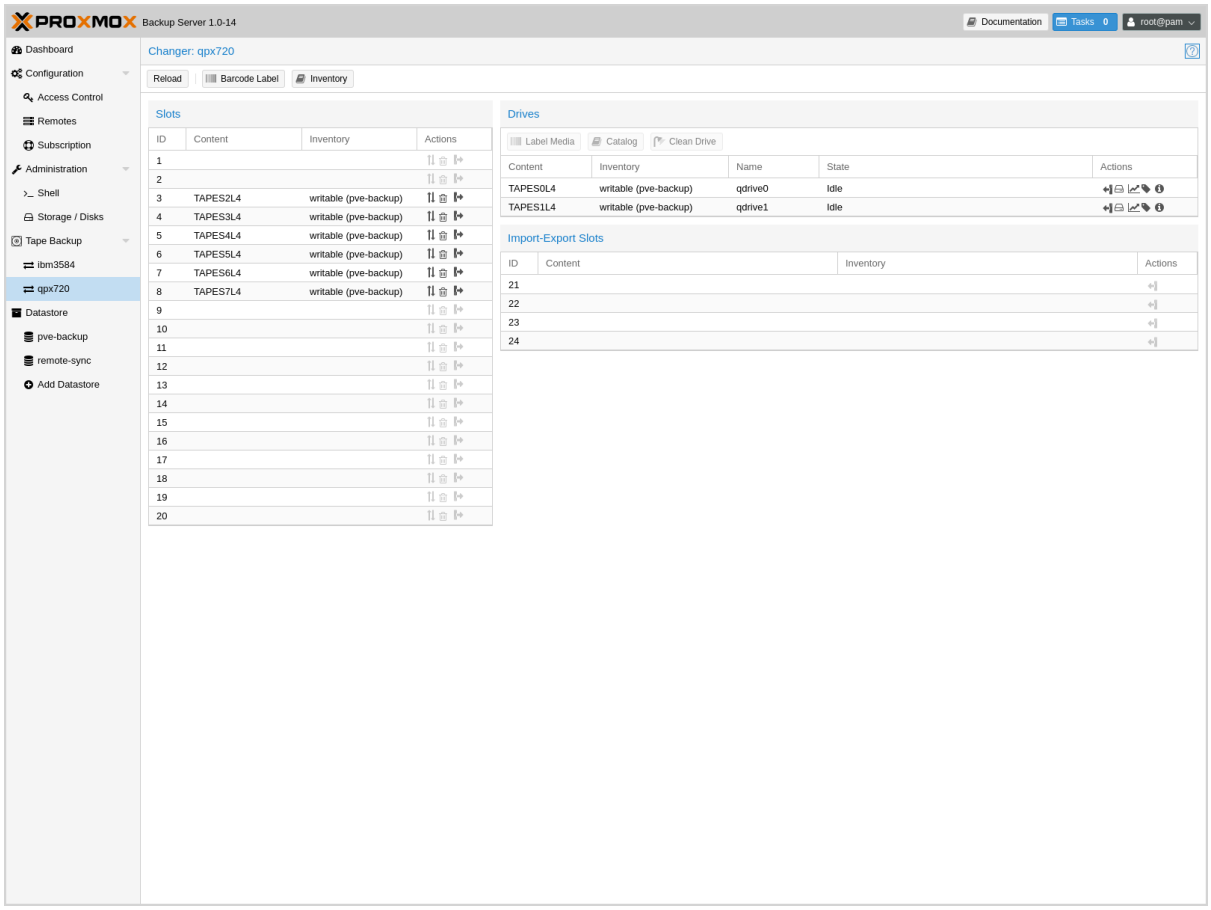
Once the archive is mounted, you can access its content under the given mount point.

```
# cd /mnt
# ls
bin   dev  home  lib32  libx32      media  opt   root  sbin  sys  usr
boot  etc  lib   lib64  lost+found  mnt    proc  run   srv   tmp  var
```

CHAPTER

TEN

TAPE BACKUP



Proxmox tape backup provides an easy way to store datastore content onto magnetic tapes. This increases data safety because you get:

- an additional copy of the data,
- on a different media type (tape),
- to an additional location (you can move tapes off-site)

In most restore jobs, only data from the last backup job is restored. Restore requests further decline, the older the data gets. Considering this, tape backup may also help to reduce disk usage, because you can safely remove data from disk, once it's archived on tape. This is especially true if you need to retain data for several years.

Tape backups do not provide random access to the stored data. Instead, you need to restore the data to disk, before you can access it again. Also, if you store your tapes off-site (using some kind of tape vaulting service), you need to bring them back on-site, before you can do any restores. So please consider that restoring from tape can take much longer than restoring from disk.

10.1 Tape Technology Primer

As of 2021, the only widely available tape technology standard is Linear Tape-Open (LTO). Different vendors offer LTO Ultrium tape drives, auto-loaders, and LTO tape cartridges.

There are a few vendors that offer proprietary drives with slight advantages in performance and capacity. Nevertheless, they have significant disadvantages:

- proprietary (single vendor)
- a much higher purchase cost

So we currently do not test such drives.

In general, LTO tapes offer the following advantages:

- Durability (30 year lifespan)
- High Capacity (12 TB)
- Relatively low cost per TB
- Cold Media
- Movable (storable inside vault)
- Multiple vendors (for both media and drives)
- Built in AES-GCM Encryption engine

Note that *Proxmox Backup Server* already stores compressed data, so using the tape compression feature has no advantage.

10.2 Supported Hardware

Proxmox Backup Server supports Linear Tape-Open generation 5 (LTO-5) or later and has best-effort support for generation 4 (LTO-4). While many LTO-4 systems are known to work, some might need firmware updates or do not implement necessary features to work with Proxmox Backup Server.

Tape changing is carried out using the SCSI Medium Changer protocol, so all modern tape libraries should work.

Note: We use a custom user space tape driver written in Rust. This driver directly communicates with the tape drive using the SCSI generic interface. This may have negative side effects when used with the old Linux kernel tape driver, so you should not use that driver with Proxmox tape backup.

10.2.1 Drive Performance

Current LTO-8 tapes provide read/write speeds of up to 360 MB/s. This means, that it still takes a minimum of 9 hours to completely write or read a single tape (even at maximum speed).

The only way to speed that data rate up is to use more than one drive. That way, you can run several backup jobs in parallel, or run restore jobs while the other dives are used for backups.

Also consider that you first need to read data from your datastore (disk). However, a single spinning disk is unable to deliver data at this rate. We measured a maximum rate of about 60MB/s to 100MB/s in practice, so it takes 33 hours to read the 12TB needed to fill up an LTO-8 tape. If you want to write to your tape at full speed, please make sure that the source datastore is able to deliver that performance (for example, by using SSDs).

10.3 Terminology

Tape Labels: are used to uniquely identify a tape. You would normally apply a sticky paper label to the front of the cartridge. We additionally store the label text magnetically on the tape (first file on tape).

Barcodes: are a special form of tape labels, which are electronically readable. Most LTO tape robots use an 8 character string encoded as Code 39, as defined in the LTO Ultrium Cartridge Label Specification.

You can either buy such barcode labels from your cartridge vendor, or print them yourself. You can use our LTO Barcode Generator app, if you would like to print them yourself.

Note: Physical labels and the associated adhesive should have an environmental performance to match or exceed the environmental specifications of the cartridge to which it is applied.

Media Pools: A media pool is a logical container for tapes. A backup job targets one media pool, so a job only uses tapes from that pool. The pool additionally defines how long a backup job can append data to tapes (allocation policy) and how long you want to keep the data (retention policy).

Media Set: A group of continuously written tapes (all from the same media pool).

Tape drive: The device used to read and write data to the tape. There are standalone drives, but drives are usually shipped within tape libraries.

Tape changer: A device which can change the tapes inside a tape drive (tape robot). They are usually part of a tape library.

Tape library: A storage device that contains one or more tape drives, a number of slots to hold tape cartridges, a barcode reader to identify tape cartridges, and an automated method for loading tapes (a robot).

This is also commonly known as an 'autoloader', 'tape robot' or 'tape jukebox'.

Inventory: The inventory stores the list of known tapes (with additional status information).

Catalog: A media catalog stores information about the media content.

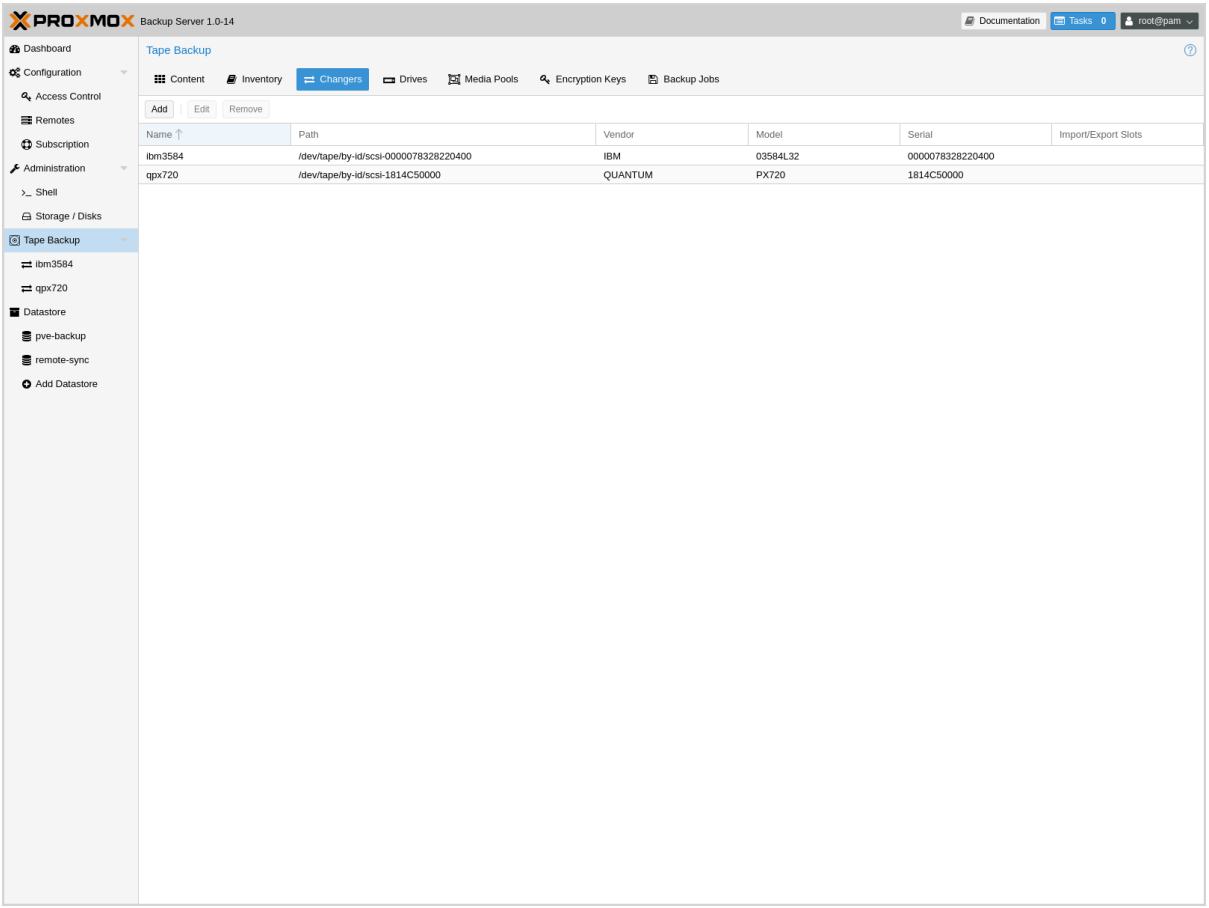
10.4 Tape Quick Start

1. Configure your tape hardware (drives and changers)
2. Configure one or more media pools
3. Label your tape cartridges
4. Start your first tape backup job ...

10.5 Configuration

Please note that you can configure anything using the graphical user interface or the command line interface. Both methods result in the same configuration.

10.5.1 Tape changers



Tape changers (robots) are part of a Tape Library. They contain a number of slots to hold tape cartridges, a barcode reader to identify tape cartridges and an automated method for loading tapes.

You can skip this step if you are using a standalone drive.

Linux is able to auto detect these devices, and you can get a list of available devices using:

```
# proxmox-tape changer scan
```

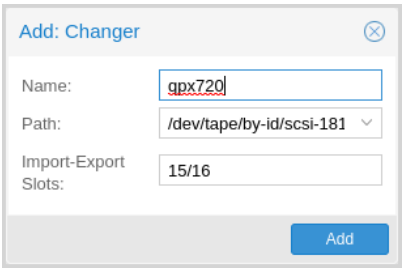
path	vendor	model	serial
/dev/tape/by-id/scsi-CC2C52	Quantum	Superloader3	CC2C52

In order to use a device with Proxmox Backup Server, you need to create a configuration entry:

```
# proxmox-tape changer create sl3 --path /dev/tape/by-id/scsi-CC2C52
```

Where `sl3` is an arbitrary name you can choose.

Note: Please use the persistent device path names from inside `/dev/tape/by-id/`. Names like `/dev/sg0` may point to a different device after reboot, and that is not what you want.



This operation can also be carried out from the GUI, by navigating to the **Changers** tab of **Tape Backup** and clicking **Add**.

You can display the final configuration with:

```
# proxmox-tape changer config sl3
```

Name	Value
name	sl3
path	/dev/tape/by-id/scsi-CC2C52

Or simply list all configured changer devices (as seen in the **Changers** tab of the GUI):

```
# proxmox-tape changer list
```

name	path	vendor	model	serial
sl3	/dev/tape/by-id/scsi-CC2C52	Quantum	Superloader3	CC2C52

The Vendor, Model and Serial number are auto-detected, but only shown if the device is online.

To test your setup, please query the status of the changer device with:

```
# proxmox-tape changer status sl3
```

entry-kind	entry-id	changer-id	loaded-slot
drive	0	vtape1	1
slot	1		
slot	2	vtape2	
...	...		
slot	16		

Tape libraries usually provide some special import/export slots (also called "mail slots"). Tapes inside those slots are accessible from outside, making it easy to add/remove tapes to/from the library. Those tapes are considered to be "offline", so backup jobs will not use them. Those special slots are auto-detected and marked as an `import-export` slot in the status command.

It's worth noting that some of the smaller tape libraries don't have such slots. While they have something called a "Mail Slot", that slot is just a way to grab the tape from the gripper. They are unable to hold media while the robot does other things. They also do not expose that "Mail Slot" over the SCSI interface, so you won't see them in the status output.

As a workaround, you can mark some of the normal slots as export slot. The software treats those slots like real `import-export` slots, and the media inside those slots are considered to be 'offline' (not available for backup):

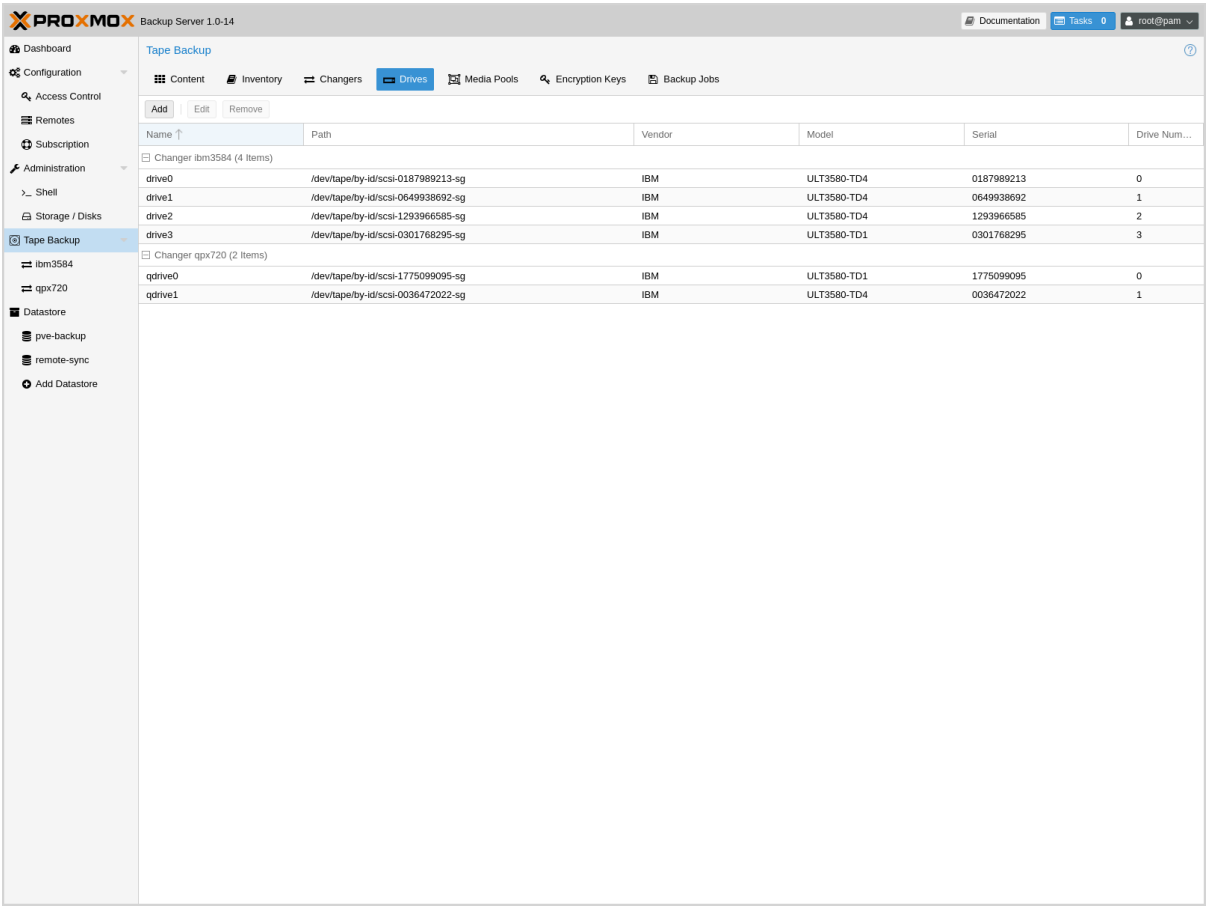
```
# proxmox-tape changer update sl3 --export-slots 15,16
```

After that, you can see those artificial `import-export` slots in the status output:

```
# proxmox-tape changer status sl3
```

entry-kind	entry-id	changer-id	loaded-slot
drive	0	vtape1	1
import-export	15		
import-export	16		
slot	1		
slot	2	vtape2	
...	...		
slot	14		

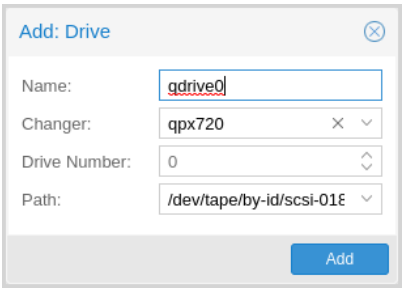
10.5.2 Tape drives



Linux is able to auto detect tape drives, and you can get a list of available tape drives using:

```
# proxmox-tape drive scan
```

path	vendor	model	serial
/dev/tape/by-id/scsi-12345-sg	IBM	ULT3580-TD4	12345



In order to use that drive with Proxmox, you need to create a configuration entry. This can be done through **Tape Backup -> Drives** in the GUI or by using the command below:

```
# proxmox-tape drive create mydrive --path  /dev/tape/by-id/scsi-12345-sg
```

Note: Please use the persistent device path names from inside `/dev/tape/by-id/`. Names like `/dev/sg0` may point to a different device after reboot, and that is not what you want.

If you have a tape library, you also need to set the associated changer device:

```
# proxmox-tape drive update mydrive --changer sl3  --changer-drivenum 0
```

The `--changer-drivenum` is only necessary if the tape library includes more than one drive (the changer status command lists all drive numbers).

You can display the final configuration with:

```
# proxmox-tape drive config mydrive
```

Name	Value
name	mydrive
path	/dev/tape/by-id/scsi-12345-sg
changer	sl3

Note: The `changer-drivenum` value 0 is not stored in the configuration, because it is the default.

To list all configured drives use:

```
# proxmox-tape drive list
```

name	path	changer	vendor	model	serial
mydrive	/dev/tape/by-id/scsi-12345-sg	sl3	IBM	ULT3580-TD4	12345

The Vendor, Model and Serial number are auto detected and only shown if the device is online.

For testing, you can simply query the drive status with:

```
# proxmox-tape status --drive mydrive
```

Name	Value
blocksize	0
density	LTO4
compression	1
buffer-mode	1

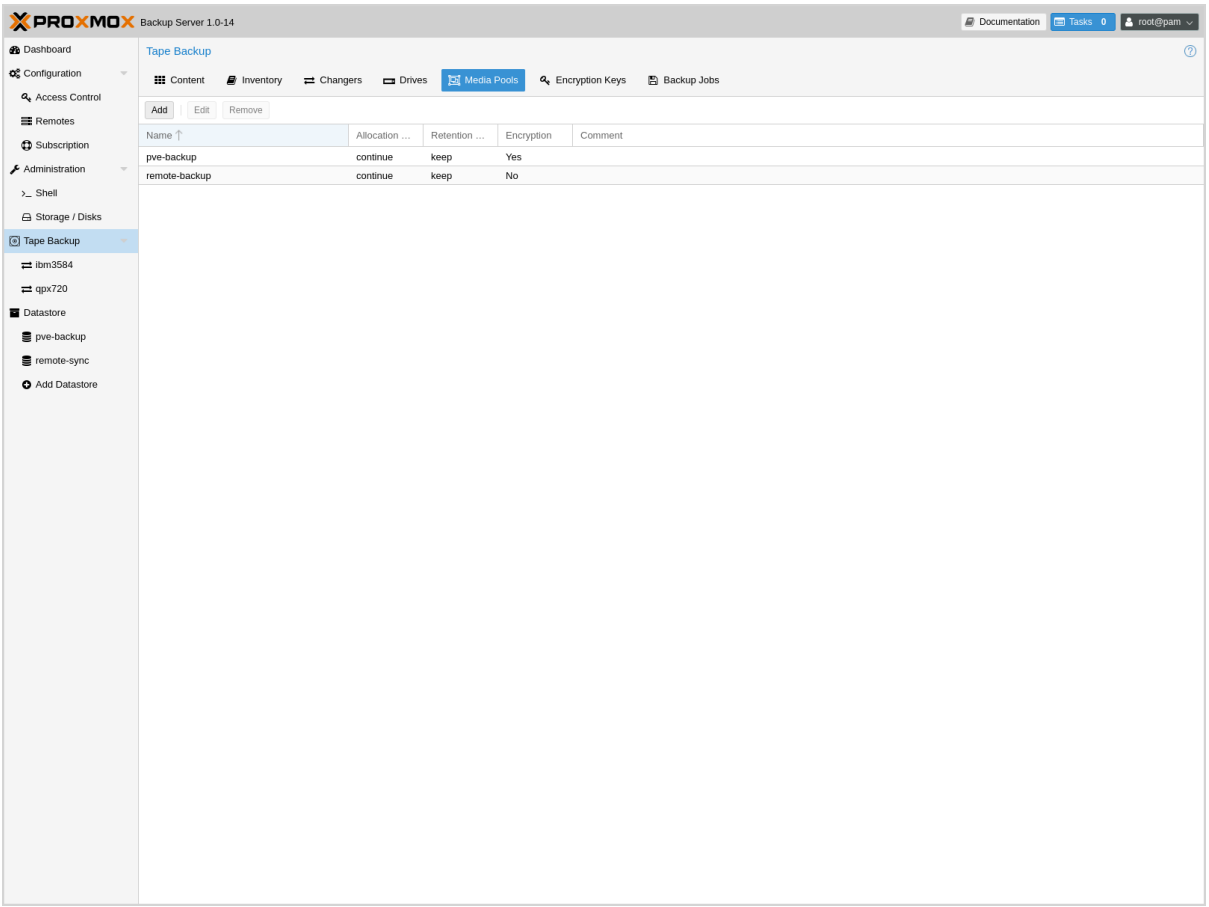
(continues on next page)

(continued from previous page)

alert-flags	(empty)
file-number	0
block-number	0
manufactured	Fri Dec 13 01:00:00 2019
bytes-written	501.80 GiB
bytes-read	4.00 MiB
medium-passes	20
medium-wearout	0.12%
volume-mounts	2

Note: Blocksize should always be 0 (variable block size mode). This is the default anyway.

10.5.3 Media Pools



A media pool is a logical container for tapes. A backup job targets a single media pool, so a job only uses tapes from that pool.

Media Set

A media set is a group of continuously written tapes, used to split the larger pool into smaller, restorable units. One or more backup jobs write to a media set, producing an ordered group of tapes. Media sets are identified by a unique ID. That ID and the sequence number are stored on each tape of that set (tape label).

Media sets are the basic unit for restore tasks. This means that you need every tape in the set to restore the media set contents. Data is fully deduplicated inside a media set.

Media Set Allocation Policy

The pool additionally defines how long backup jobs can append data to a media set. The following settings are possible:

- Try to use the current media set (`continue`).

  This setting produces one large media set. While this is very space efficient (deduplication, no unused space), it can lead to long restore times, because restore jobs need to read all tapes in the set.

  Note: Data is fully deduplicated inside a media set. This also means that data is randomly distributed over the tapes in the set. Thus, even if you restore a single VM, data may have to be read from all tapes inside the media set.

  Larger media sets are also more error-prone, because a single damaged tape makes the restore fail.

  Usage scenario: Mostly used with tape libraries. You manually trigger new set creation by running a backup job with the `--export` option.

  Note: Retention period starts with the existence of a newer media set.

- Always create a new media set (`always`).

  With this setting, each backup job creates a new media set. This is less space efficient, because the media from the last set may not be fully written, leaving the remaining space unused.

  The advantage is that this produces media sets of minimal size. Small sets are easier to handle, can be moved more conveniently to an off-site vault, and can be restored much faster.

  Note: Retention period starts with the creation time of the media set.

- Create a new set when the specified Calendar Event triggers.

  This allows you to specify points in time by using systemd like Calendar Event specifications (see systemd.time manpage).

  For example, the value `weekly` (or `Mon *-*-* 00:00:00`) will create a new set each week.

  This balances between space efficiency and media count.

  Note: Retention period starts on the creation time of the next media-set or, if that does not exist, when the calendar event next triggers after the current media-set start time.

Additionally, the following events may allocate a new media set:

- Required tape is offline (and you use a tape library).
- Current set contains damaged or retired tapes.
- Media pool encryption has changed
- Database consistency errors, for example, if the inventory does not contain the required media information, or it contains conflicting information (outdated data).

Retention Policy

Defines how long we want to keep the data.

- Always overwrite media.
- Protect data for the duration specified.

  We use systemd like time spans to specify durations, e.g. `2 weeks` (see systemd.time man-page).
- Never overwrite data.

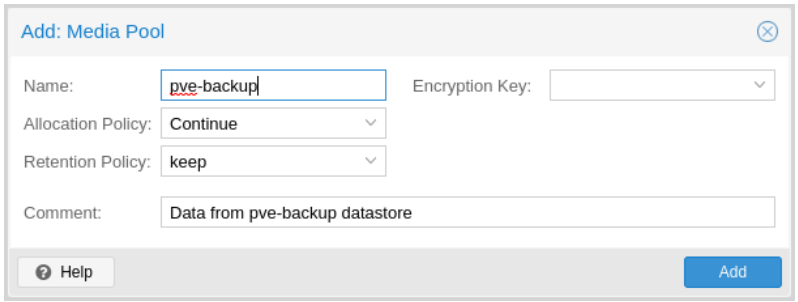
Hardware Encryption

LTO-4 (or later) tape drives support hardware encryption. If you configure the media pool to use encryption, all data written to the tapes is encrypted using the configured key.

This way, unauthorized users cannot read data from the media, for example, if you loose a tape while shipping to an offsite location.

Note: If the backup client also encrypts data, data on the tape will be double encrypted.

The password protected key is stored on each medium, so that it is possbible to *restore the key* using the password. Please make sure to remember the password, in case you need to restore the key.



To create a new media pool, add one from **Tape Backup -> Media Pools** in the GUI, or enter the following command:

```
// proxmox-tape pool create <name> --drive <string> [OPTIONS]

# proxmox-tape pool create daily --drive mydrive
```

Additional options can be set later, using the update command:

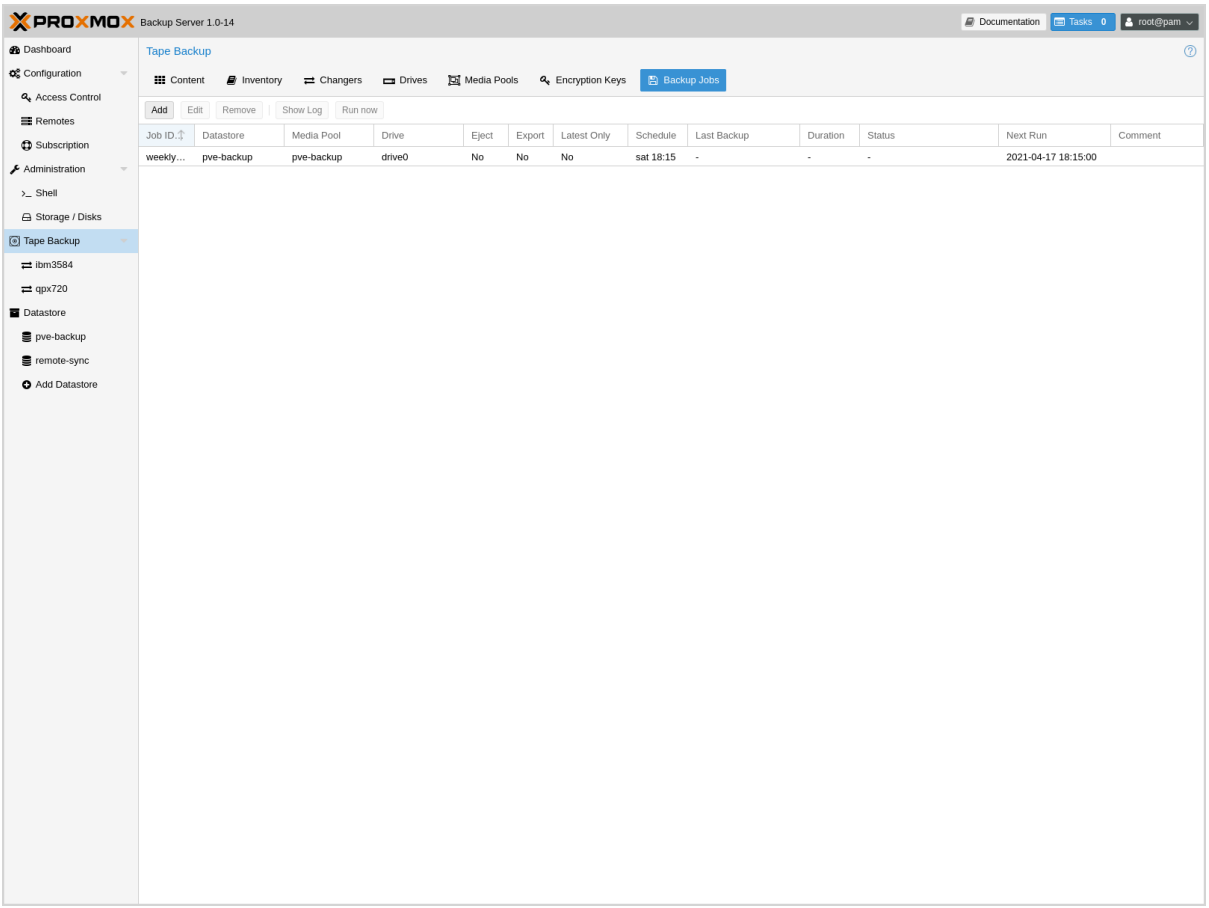
```
# proxmox-tape pool update daily --allocation daily --retention 7days
```

To list all configured pools use:

```
# proxmox-tape pool list
```

name	drive	allocation	retention	template
daily	mydrive	daily	7days	

10.5.4 Tape Backup Jobs



To automate tape backup, you can configure tape backup jobs which write datastore content to a media pool, based on a specific time schedule. The required settings are:

- `store`: The datastore you want to backup
- `pool`: The media pool - only tape cartridges from that pool are used.
- `drive`: The tape drive.
- `schedule`: Job schedule (see *Calendar Events*)

For example, to configure a tape backup job for datastore `vmstore1` use:

```
# proxmox-tape backup-job create job2 --store vmstore1 \
  --pool yourpool --drive yourdrive --schedule daily
```

The backup includes all snapshots from a backup group by default. You can set the `latest-only` flag to include only the latest snapshots:

```
# proxmox-tape backup-job update job2 --latest-only
```

Backup jobs can use email to send tape request notifications or report errors. You can set the notification user with:

```
# proxmox-tape backup-job update job2 --notify-user root@pam
```

Note: The email address is a property of the user (see *User Management*).

It is sometimes useful to eject the tape from the drive after a backup. For a standalone drive, the `eject-media` option ejects the tape, making sure that the following backup cannot use the tape (unless someone manually loads the tape again). For tape libraries, this option unloads the tape to a free slot, which provides better dust protection than inside a drive:

```
# proxmox-tape backup-job update job2 --eject-media
```

Note: For failed jobs, the tape remains in the drive.

For tape libraries, the `export-media` option moves all tapes from the media set to an export slot, making sure that the following backup cannot use the tapes. An operator can pick up those tapes and move them to a vault.

```
# proxmox-tape backup-job update job2 --export-media
```

Note: The `export-media` option can be used to force the start of a new media set, because tapes from the current set are no longer online.

It is also possible to run backup jobs manually:

```
# proxmox-tape backup-job run job2
```

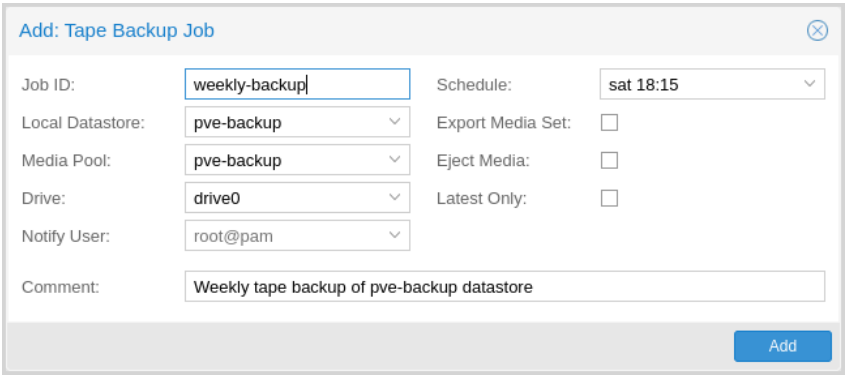
To remove a job, please use:

```
# proxmox-tape backup-job remove job2
```

By default, all (recursive) namespaces of the datastore are included in a tape backup. You can specify a single namespace with `ns` and a depth with `max-depth`. For example:

```
# proxmox-tape backup-job update job2 --ns mynamespace --max-depth 3
```

If no *max-depth* is given, it will include all recursive namespaces.



This same functionality also exists in the GUI, under the **Backup Jobs** tab of **Tape Backup**, where *Local Datastore* relates to the datastore you want to backup and *Media Pool* is the pool to back up to.

10.6 Administration

Many sub-commands of the `proxmox-tape` command line tools take a parameter called `--drive`, which specifies the tape drive you want to work on. For convenience, you can set this in an environment variable:

```
# export PROXMOX_TAPE_DRIVE=mydrive
```

You can then omit the `--drive` parameter from the command. If the drive has an associated changer device, you may also omit the changer parameter from commands that need a changer device, for example:

```
# proxmox-tape changer status
```

should display the changer status of the changer device associated with drive `mydrive`.

10.6.1 Label Tapes

By default, tape cartridges all look the same, so you need to put a label on them for unique identification. First, put a sticky paper label with some human readable text on the cartridge.

If you use a Tape Library, you should use an 8 character string encoded as Code 39, as defined in the LTO Ultrium Cartridge Label Specification. You can either buy such barcode labels from your cartridge vendor, or print them yourself. You can use our LTO Barcode Generator app to print them.

Next, you need to write that same label text to the tape, so that the software can uniquely identify the tape too.

For a standalone drive, manually insert the new tape cartridge into the drive and run:

```
# proxmox-tape label --changer-id <label-text> [--pool <pool-name>]
```

You may omit the `--pool` argument to allow the tape to be used by any pool.

Note: For safety reasons, this command fails if the tape contains any data. If you want to overwrite it anyway, erase the tape first.

You can verify success by reading back the label:

```
# proxmox-tape read-label
```

Name	Value
changer-id	vtape1
uuid	7f42c4dd-9626-4d89-9f2b-c7bc6da7d533
ctime	Wed Jan 6 09:07:51 2021
pool	daily
media-set-uuid	00000000-0000-0000-0000-000000000000
media-set-ctime	Wed Jan 6 09:07:51 2021

Note: The `media-set-uuid` using all zeros indicates an empty tape (not used by any media set).

If you have a tape library, apply the sticky barcode label to the tape cartridges first. Then load those empty tapes into the library. You can then label all unlabeled tapes with a single command:

```
# proxmox-tape barcode-label [--pool <pool-name>]
```

10.6.2 Run Tape Backups

To manually run a backup job click *Run Now* in the GUI or use the command:

```
# proxmox-tape backup <store> <pool> [OPTIONS]
```

The following options are available:

--eject-media Eject media upon job completion.

It is normally good practice to eject the tape after use. This unmounts the tape from the drive and prevents the tape from getting dusty.

--export-media-set Export media set upon job completion.

After a successful backup job, this moves all tapes from the used media set into import-export slots. The operator can then pick up those tapes and move them to a media vault.

--ns The namespace to backup.

Used if you only want to backup a specific namespace. If omitted, the root namespace is assumed.

--max-depth The depth to recurse namespaces.

`0` means no recursion at all (only the given namespace). If omitted, all namespaces are recursed (below the given one).

10.6.3 Restore from Tape

Restore is done at media-set granularity, so you first need to find out which media set contains the data you want to restore. This information is stored in the media catalog. If you do not have media catalogs, you need to restore them first. Please note that you need the catalog to find your data, but restoring a complete media-set does not need media catalogs.

The following command lists the media content (from catalog):

```
# proxmox-tape media content
┌────────────┬──────┬──────────────────────────┬────────┬─────────────────────────────┐␣
↪media-set-uuid                        │
╞════════════╪══════╪══════════════════════════╪════════╪═════════════════════════════╡
│ TEST01L8   │ p2   │ Wed Jan 13 13:55:55 2021 │      0 │ vm/201/2021-01-11T10:43:48Z │␣
↪9da37a55-aac7-4deb-91c6-482b3b675f30 │
├────────────┼──────┼──────────────────────────┼────────┼─────────────────────────────┤
│        ... │ ...  │                      ... │    ... │ ...                         │␣
↪                                  ... │
└────────────┴──────┴──────────────────────────┴────────┴─────────────────────────────┘
```

A restore job reads the data from the media set and moves it back to data disk (datastore):

```
// proxmox-tape restore <media-set-uuid> <datastore>

# proxmox-tape restore 9da37a55-aac7-4deb-91c6-482b3b675f30 mystore
```

Single Snapshot Restore

Sometimes it is not necessary to restore an entire media-set, but only some specific snapshots from the tape. This can be achieved with the `snapshots` parameter:

```
// proxmox-tape restore <media-set-uuid> <datastore> [<snapshot>]

# proxmox-tape restore 9da37a55-aac7-4deb-91c6-482b3b675f30 mystore sourcestore:host/hostname/
↪2022-01-01T00:01:00Z
```

This first restores the snapshot to a temporary location, then restores the relevant chunk archives, and finally restores the snapshot data to the target datastore.

The `snapshot` parameter can be passed multiple times, in order to restore multiple snapshots with one restore action.

Note: When using the single snapshot restore, the tape must be traversed more than once, which, if you restore many snapshots at once, can take longer than restoring the whole datastore.

Namespaces

It is also possible to select and map specific namespaces from a media-set during a restore. This is possible with the `namespaces` parameter. The format for the parameter is:

```
store=<source-datastore>[,source=<source-ns>][,target=<target-ns>][,max-depth=<depth>]
```

If `source` or `target` is not given, the root namespace is assumed. When no `max-depth` is given, the source namespace will be fully recursed.

An example restore command:

```
# proxmox-tape restore 9da37a55-aac7-4deb-91c6-482b3b675f30 mystore --namespaces␣
↪store=sourcedatastore,source=ns1,target=ns2,max-depth=2
```

The parameter can be given multiple times. It can also be combined with the `snapshots` parameter to only restore those snapshots and map them to different namespaces.

10.6.4 Update Inventory

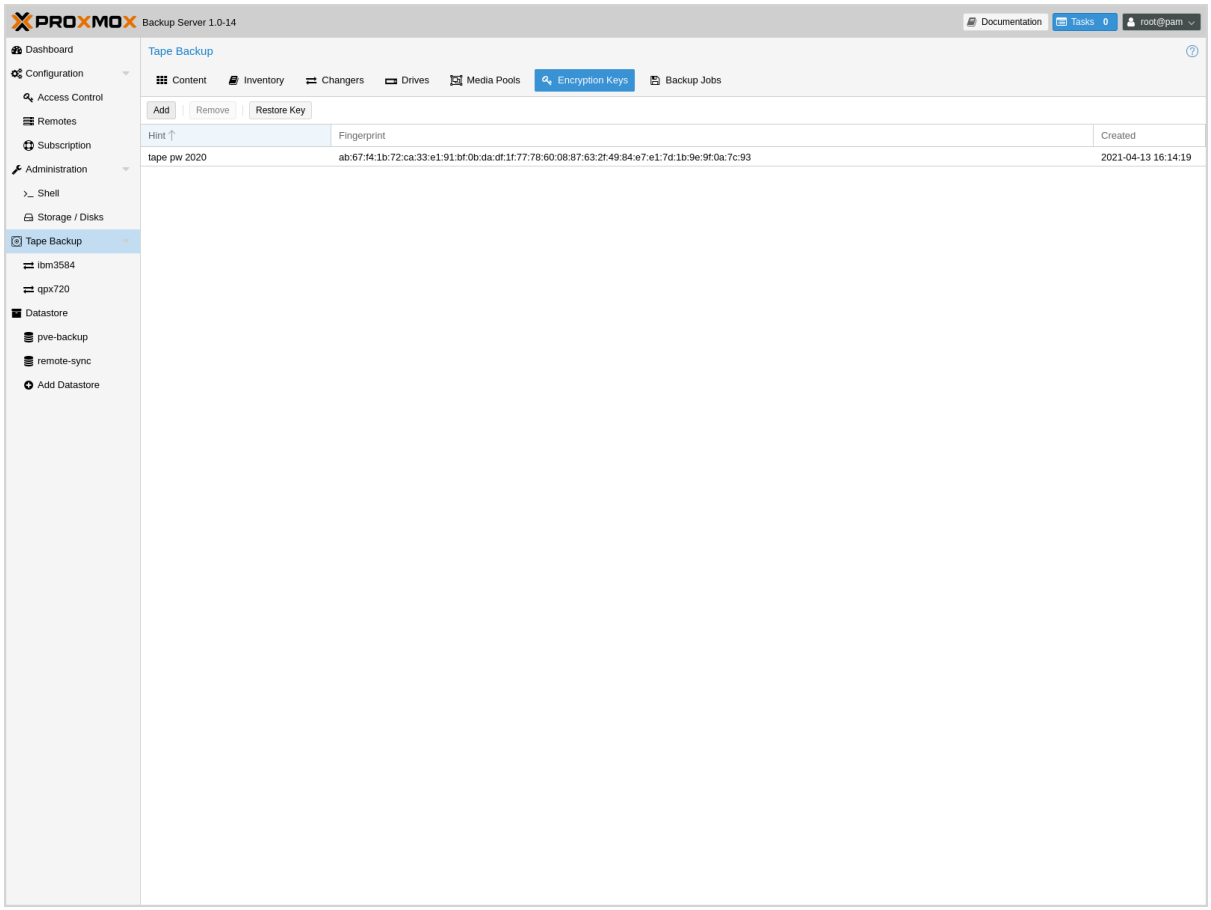
10.6.5 Restore Catalog

To restore a catalog from an existing tape, just insert the tape into the drive and execute:

```
# proxmox-tape catalog
```

You can restore from a tape even without an existing catalog, but only the whole media set. If you do this, the catalog will be automatically created.

10.6.6 Encryption Key Management



Proxmox Backup Server also provides an interface for handling encryption keys on the backup server. Encryption keys can be managed from the **Tape Backup -> Encryption Keys** section of the GUI or through the `proxmox-tape key` command line tool. To create a new encryption key from the command line:

```
# proxmox-tape key create --hint "tape pw 2020"
Tape Encryption Key Password: **********
Verify Password: **********

↪"14:f8:79:b9:f5:13:e5:dc:bf:b6:f9:88:48:51:81:dc:79:bf:a0:22:68:47:d1:73:35:2d:b6:20:e1:7f:f5:0f
↪"
```

List existing encryption keys:

```
# proxmox-tape key list
```

fingerprint	hint
14:f8:79:b9:f5:13:e5:dc: ... :b6:20:e1:7f:f5:0f	tape pw 2020

To show encryption key details:

```
# proxmox-tape key show 14:f8:79:b9:f5:13:e5:dc:...:b6:20:e1:7f:f5:0f
```

Name	Value
kdf	scrypt
created	Sat Jan 23 14:47:21 2021

(continues on next page)

(continued from previous page)

modified	Sat Jan 23 14:47:21 2021
fingerprint	14:f8:79:b9:f5:13:e5:dc:...:b6:20:e1:7f:f5:0f
hint	tape pw 2020

The `paperkey` subcommand can be used to create a QR encoded version of a tape encryption key. The following command sends the output of the `paperkey` command to a text file, for easy printing:

```
proxmox-tape key paperkey <fingerprint> --output-format text > qrkey.txt
```

Restoring Encryption Keys

You can restore the encryption key from the tape, using the password used to generate the key. First, load the tape you want to restore into the drive. Then run:

```
# proxmox-tape key restore
Tepe Encryption Key Password: ***********
```

If the password is correct, the key will get imported to the database. Further restore jobs automatically use any available key.

10.6.7 Tape Cleaning

LTO tape drives require regular cleaning. This is done by loading a cleaning cartridge into the drive, which is a manual task for standalone drives.

For tape libraries, cleaning cartridges are identified using special labels starting with letters "CLN". For example, our tape library has a cleaning cartridge inside slot 3:

```
# proxmox-tape changer status sl3
```

entry-kind	entry-id	changer-id	loaded-slot
drive	0	vtape1	1
slot	1		
slot	2	vtape2	
slot	3	CLN001CU	
...	...		

To initiate a cleaning operation simply run:

```
# proxmox-tape clean
```

This command does the following:

- find the cleaning tape (in slot 3)
- unload the current media from the drive (back to slot 1)
- load the cleaning tape into the drive
- run drive cleaning operation
- unload the cleaning tape (to slot 3)

10.7 WORM Tapes

WORM (write once, read many) tapes are special cartridges that cannot be deleted or overwritten. This may be useful for legal or protection purposes.

If you want to use them, you must use a media pool with a retention policy of *keep*. Otherwise, a backup job can fail when it tries to erase or overwrite the tape.

Proxmox Backup Server makes no distinction between normal and WORM tapes. To avoid confusion, use a different naming scheme for WORM backups and use dedicated media pools for them. Do not mix WORM and non-WORM tapes in the same media pool.

10.8 Example Setups

Here are a few example setups for managing media pools and schedules. This is not an exhaustive list, and there are many more possible combinations of useful settings.

10.8.1 Single Continued Media Set

The most simple setup: always continue the media-set and never expire.

Allocation policy: continue

Retention policy: keep

This setup has the advantage of being easy to manage and reuses the benefits from deduplication as much as possible. But, it also provides no redundancy, meaning a failure of any single tape would render all backups referring to chunks from that tape unusable.

If you want to start a new media-set manually, you can set the currently writable media of the set either to 'full', or set the location to an off-site vault.

10.8.2 Weekday Scheme

A slightly more complex scheme, where the goal is to have an independent tape or media set for each weekday, for example from Monday to Friday. This can be solved by having a separate media pool for each day, so 'Monday', 'Tuesday', etc.

Allocation policy: should be 'mon' for the 'Monday' pool, 'tue' for the Tuesday pool and so on.

Retention policy: overwrite

There should be one or more tape-backup jobs for each pool on the corresponding weekday. This scheme is still very manageable with one media set per weekday, and could be moved off-site easily.

10.8.3 Multiple Pools with Different Policies

Complex setups are also possible, with multiple media pools configured with different allocation and retention policies.

An example would be to have two media pools. The first configured with weekly allocation and a few weeks of retention:

Allocation policy: mon

Retention policy: 3 weeks

The second pool configured with yearly allocation that does not expire:

Allocation policy: yearly

Retention policy: keep

In combination with fitting prune settings and tape backup schedules, this achieves long-term storage of some backups, while keeping the recent backups on smaller media sets that expire roughly every 4 weeks (that is, three plus the current week).

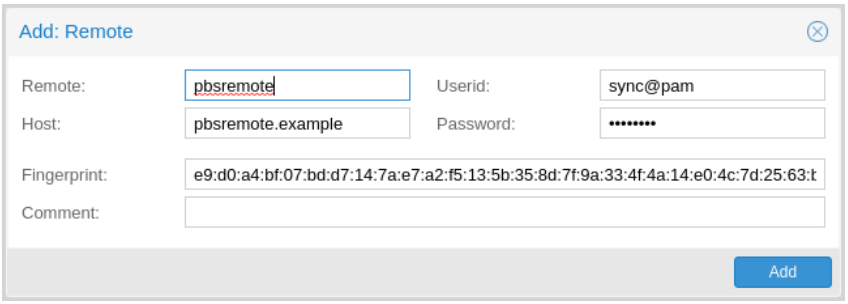
CHAPTER

ELEVEN

MANAGING REMOTES & SYNC

11.1 Remote

A remote refers to a separate Proxmox Backup Server installation and a user on that installation, from which you can *sync* datastores to a local datastore with a *Sync Job*. You can configure remotes in the web interface, under **Configuration -> Remotes**. Alternatively, you can use the `remote` subcommand. The configuration information for remotes is stored in the file `/etc/proxmox-backup/remote.cfg`.



To add a remote, you need its hostname or IP address, a userid and password on the remote, and its certificate fingerprint. To get the fingerprint, use the `proxmox-backup-manager cert info` command on the remote, or navigate to **Dashboard** in the remote's web interface and select **Show Fingerprint**.

```
# proxmox-backup-manager cert info |grep Fingerprint
Fingerprint (sha256): 64:d3:ff:3a:50:38:53:5a:9b:f7:50:...:ab:fe
```

Using the information specified above, you can add a remote from the **Remotes** configuration panel, or by using the command:

```
# proxmox-backup-manager remote create pbs2 --host pbs2.mydomain.example --userid sync@pam --
↪password 'SECRET' --fingerprint 64:d3:ff:3a:50:38:53:5a:9b:f7:50:...:ab:fe
```

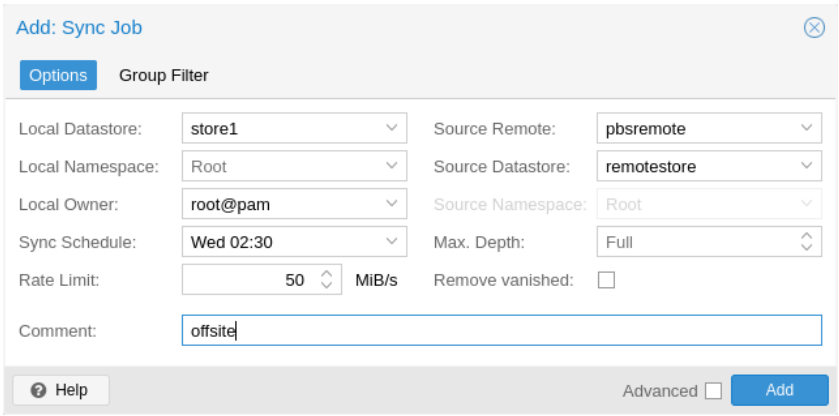
Use the `list`, `show`, `update`, `remove` subcommands of `proxmox-backup-manager remote` to manage your remotes:

```
# proxmox-backup-manager remote update pbs2 --host pbs2.example
# proxmox-backup-manager remote list
```

name	host	userid	fingerprint	comment
pbs2	pbs2.example	sync@pam	64:d3:ff:3a:50:38:53:5a:9b:f7:50:...:ab:fe	

```
# proxmox-backup-manager remote remove pbs2
```

11.2 Sync Jobs



Sync jobs are configured to pull the contents of a datastore on a **Remote** to a local datastore. You can manage sync jobs in the web interface, from the **Sync Jobs** tab of the **Datastore** panel or from that of the Datastore itself. Alternatively, you can manage them with the `proxmox-backup-manager sync-job` command. The configuration information for sync jobs is stored at `/etc/proxmox-backup/sync.cfg`. To create a new sync job, click the add button in the GUI, or use the `create` subcommand. After creating a sync job, you can either start it manually from the GUI or provide it with a schedule (see *Calendar Events*) to run regularly.

```
# proxmox-backup-manager sync-job create pbs2-local --remote pbs2 --remote-store local --
↪store local --schedule 'Wed 02:30'
# proxmox-backup-manager sync-job update pbs2-local --comment 'offsite'
# proxmox-backup-manager sync-job list
```

id	store	remote	remote-store	schedule	comment
pbs2-local	local	pbs2	local	Wed 02:30	offsite

```
# proxmox-backup-manager sync-job remove pbs2-local
```

To set up sync jobs, the configuring user needs the following permissions:

1. `Remote.Read` on the `/remote/{remote}/{remote-store}` path
2. At least `Datastore.Backup` on the local target datastore (`/datastore/{store}`)

Note: A sync job can only sync backup groups that the configured remote's user/API token can read. If a remote is configured with a user/API token that only has `Datastore.Backup` privileges, only the limited set of accessible snapshots owned by that user/API token can be synced.

If the `remove-vanished` option is set, `Datastore.Prune` is required on the local datastore as well. If the `owner` option is not set (defaulting to `root@pam`) or is set to something other than the configuring user, `Datastore.Modify` is required as well.

If the `group-filter` option is set, only backup groups matching at least one of the specified criteria are synced. The available criteria are:

- **Backup type, for example, to only sync groups of the *ct* (Container) type:**

```
# proxmox-backup-manager sync-job update ID --group-filter type:ct
```

- **Full group identifier, to sync a specific backup group:**

```
# proxmox-backup-manager sync-job update ID --group-filter group:vm/100
```

- **Regular expression, matched against the full group identifier**

```
# proxmox-backup-manager sync-job update ID --group-filter regex:'^vm/1\d{2,3}$'
```

The same filter is applied to local groups, for handling of the `remove-vanished` option.

Note: The `protected` flag of remote backup snapshots will not be synced.

11.2.1 Namespace Support

Sync jobs can be configured to not only sync datastores, but also subsets of datastores in the form of namespaces or namespace sub-trees. The following parameters influence how namespaces are treated as part of a sync job's execution:

- `remote-ns`: the remote namespace anchor (default: the root namespace)
- `ns`: the local namespace anchor (default: the root namespace)
- `max-depth`: whether to recursively iterate over sub-namespaces of the remote namespace anchor (default: *None*)

If `max-depth` is set to *0*, groups are synced from `remote-ns` into `ns`, without any recursion. If it is set to *None* (left empty), recursion depth will depend on the value of `remote-ns` and the remote side's availability of namespace support:

- `remote-ns` set to something other than the root namespace: remote *must* support namespaces, full recursion starting at `remote-ns`.
- `remote-ns` set to root namespace and remote *supports* namespaces: full recursion starting at root namespace.
- `remote-ns` set to root namespace and remote *does not support* namespaces: backwards-compat mode, only root namespace will be synced into `ns`, no recursion.

Any other value of `max-depth` will limit recursion to at most `max-depth` levels, for example: `remote-ns` set to *location_a/department_b* and `max-depth` set to *1* will result in *location_a/department_b* and at most one more level of sub-namespaces being synced.

The namespace tree starting at `remote-ns` will be mapped into `ns` up to a depth of `max-depth`.

For example, with the following namespaces at the remote side:

- *location_a*
  - *location_a/department_x*
    - *location_a/department_x/team_one*
    - *location_a/department_x/team_two*
  - *location_a/department_y*
    - *location_a/department_y/team_one*
    - *location_a/department_y/team_two*
- *location_b*

and `remote-ns` being set to *location_a/department_x* and `ns` set to *location_a_dep_x* resulting in the following namespace tree on the sync target:

- *location_a_dep_x* (containing the remote's *location_a/department_x*)
  - *location_a_dep_x/team_one* (containing the remote's *location_a/department_x/team_one*)
  - *location_a_dep_x/team_two* (containing the remote's *location_a/department_x/team_two*)

with the rest of the remote namespaces and groups not being synced (by this sync job).

If a remote namespace is included in the sync job scope, but does not exist locally, it will be created (provided the sync job owner has sufficient privileges).

If the `remove-vanished` option is set, namespaces that are included in the sync job scope but only exist locally are treated as vanished and removed (provided the sync job owner has sufficient privileges).

Note: All other limitations on sync scope (such as remote user/API token privileges, group filters) also apply for sync jobs involving one or multiple namespaces.

11.2.2 Bandwidth Limit

Syncing a datastore to an archive can produce a lot of traffic and impact other users of the network. In order to avoid network or storage congestion, you can limit the bandwidth of the sync job by setting the `rate-in` option either in the web interface or using the `proxmox-backup-manager` command-line tool:

```
# proxmox-backup-manager sync-job update ID --rate-in 20MiB
```

CHAPTER

TWELVE

MAINTENANCE TASKS

12.1 Pruning

Prune lets you specify which backup snapshots you want to keep. The following retention options are available:

keep-last <N> Keep the last <N> backup snapshots.

keep-hourly <N> Keep backups for the last <N> hours. If there is more than one backup for a single hour, only the latest is retained.

keep-daily <N> Keep backups for the last <N> days. If there is more than one backup for a single day, only the latest is retained.

keep-weekly <N> Keep backups for the last <N> weeks. If there is more than one backup for a single week, only the latest is retained.

> **Note:** Weeks start on Monday and end on Sunday. The software uses the ISO week date system and handles weeks at the end of the year correctly.

keep-monthly <N> Keep backups for the last <N> months. If there is more than one backup for a single month, only the latest is retained.

keep-yearly <N> Keep backups for the last <N> years. If there is more than one backup for a single year, only the latest is retained.

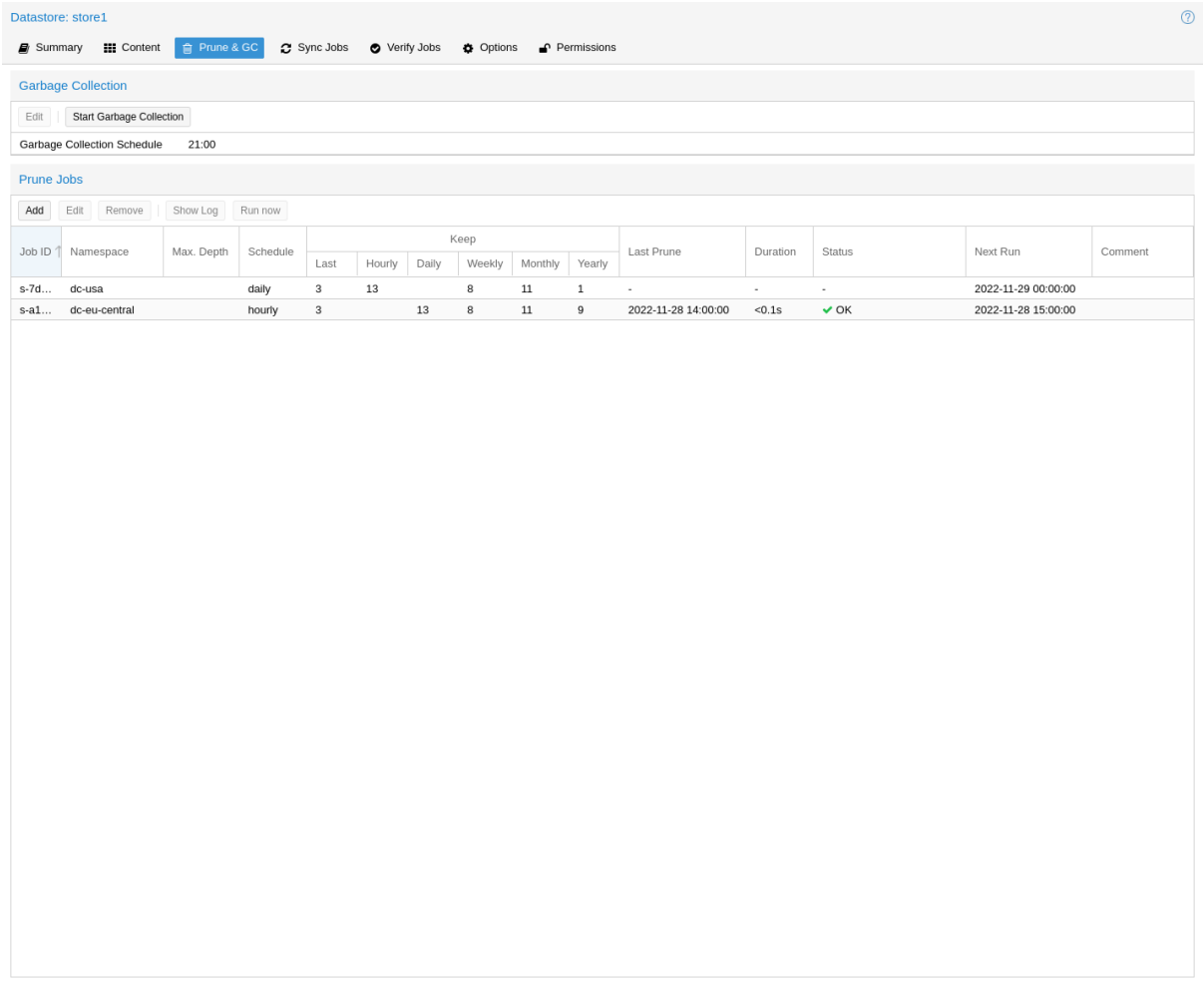
The retention options are processed in the order given above. Each option only covers backups within its time period. The next option does not take care of already covered backups. It will only consider older backups.

Old unfinished or incomplete backups will be removed by the prune command, unless they are newer than the last successful backup. In this case, the last failed backup is retained.

12.1.1 Prune Simulator

You can use the built-in prune simulator to explore the effect of different retention options with various backup schedules.

12.1.2 Prune Jobs



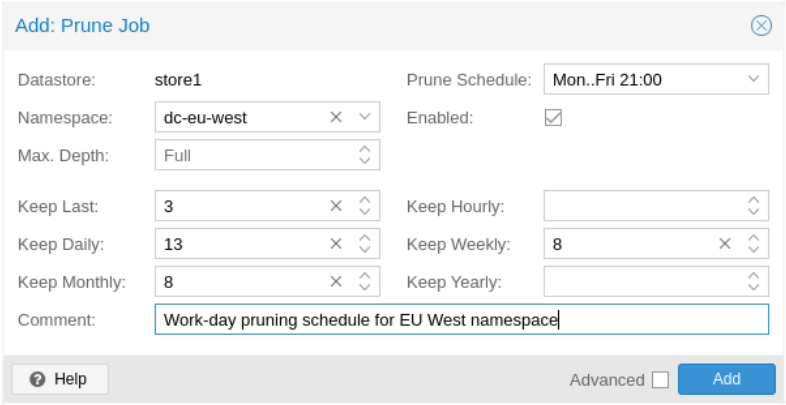
Prune jobs are configured to periodically prune a datastore or a subset of it. You can manage prune jobs in the web interface, from the **Prune & GC** tab of the **Datastore** panel or from that of the Datastore itself. Alternatively, you can manage them with the `proxmox-backup-manager prune-job` command. The configuration information for prune jobs is stored at `/etc/proxmox-backup/prune.cfg`. To create a new prune job, click the add button in the GUI, or use the manager CLI's `create` subcommand. After creating a prune job, you can either start it manually from the GUI or provide it with a schedule (see *Calendar Events*) to run regularly.

Each prune job has settings for retention, limitation of scope and frequency.

`store <datastore>` The datastore you want to run this prune job on.

`ns <namespace>` Limit the prune job to a specific namespace.

`max-depth <N>` Configure the namespace depth it should prune from below the configured namespace. For example, *0* to only prune the backup groups available directly on the configured namespace itself. Omit the parameter to scan to the full depth below.



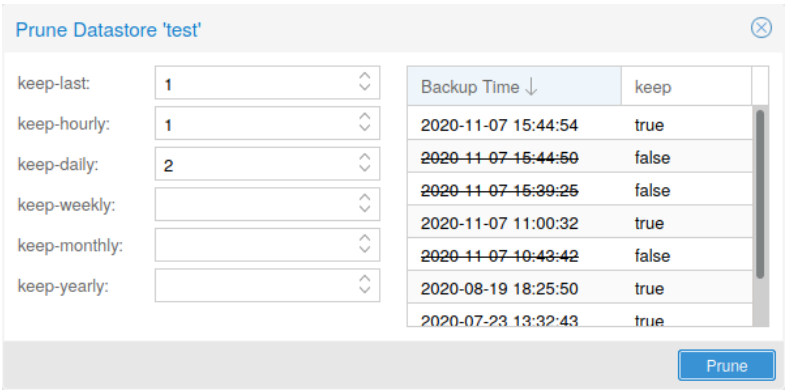
schedule Configure a *calendar event interval* for when to automatically trigger this job. You can omit this if you want to trigger a job only manually.

keep-X See the description of the various retention options above.

disable Set to disable a job temporarily while keeping its settings.

comment You can add a short comment for a job, for example about it intentions.

12.1.3 Manual Pruning



To manually prune a specific backup group, you can use `proxmox-backup-client`'s `prune` subcommand, discussed in *Pruning and Removing Backups*, or navigate to the **Content** tab of the datastore and click the scissors icon in the **Actions** column of the relevant backup group.

12.1.4 Retention Settings Example

The backup frequency and retention of old backups may depend on how often data changes and how important an older state may be in a specific workload. When backups act as a company's document archive, there may also be legal requirements for how long backup snapshots must be kept.

For this example, we assume that you are doing daily backups, have a retention period of 10 years, and the period between backups stored gradually grows.

- **keep-last:** 3 - even if only daily backups, an admin may want to create an extra one just before or after a big upgrade. Setting keep-last ensures this.
- **keep-hourly:** not set - for daily backups this is not relevant. You cover extra manual backups already, with keep-last.
- **keep-daily:** 13 - together with keep-last, which covers at least one day, this ensures that you have at least two weeks of backups.

- **keep-weekly:** 8 - ensures that you have at least two full months of weekly backups.
- **keep-monthly:** 11 - together with the previous keep settings, this ensures that you have at least a year of monthly backups.
- **keep-yearly:** 9 - this is for the long term archive. As you covered the current year with the previous options, you would set this to nine for the remaining ones, giving you a total of at least 10 years of coverage.

We recommend that you use a higher retention period than is minimally required by your environment; you can always reduce it if you find it is unnecessarily high, but you cannot recreate backup snapshots from the past.

12.2 Garbage Collection

Garbage collection (GC) is the process that frees up space in a datastore by deleting all unused backup chunks from chunk storage. GC completes the pruning of backup snapshots, which deletes only the metadata, not the underlying backup data.

It's recommended to setup a schedule to ensure that unused space is cleaned up periodically. For most setups a weekly schedule provides a good interval to start.

12.2.1 GC Background

In Proxmox Backup Server, backup data is not saved directly, but rather as chunks that are referred to by the indexes of each backup snapshot. This approach enables reuse of chunks through deduplication, among other benefits that are detailed in the *Technical Overview*.

When deleting a backup snapshot, Proxmox Backup Server cannot directly remove the chunks associated with it because other backups, even ones that are still running, may have references to those chunks. To avoid excessive load and slow performance, the whole datastore cannot be locked to scan all other indexes for references to the same chunks on every snapshot deletion. Moreover, locking the entire datastore is not feasible because new backups would be blocked until the deletion process was complete.

Therefore, Proxmox Backup Server uses a garbage collection (GC) process to identify and remove the unused backup chunks that are no longer needed by any snapshot in the datastore. The GC process is designed to efficiently reclaim the space occupied by these chunks with low impact on the performance of the datastore or interfering with other backups.

The garbage collection (GC) process is performed per datastore and is split into two phases:

- Phase one: Mark All index files are read, and the access time of the referred chunk files is updated.
- Phase two: Sweep The task iterates over all chunks, checks their file access time, and if it is older than the cutoff time (i.e., the time when GC started, plus some headroom for safety and Linux file system behavior), the task knows that the chunk was neither referred to in any backup index nor part of any currently running backup that has no index to scan for. As such, the chunk can be safely deleted.

12.2.2 Manually Starting GC

You can monitor and run *garbage collection* on the Proxmox Backup Server using the `garbage-collection` subcommand of `proxmox-backup-manager`. You can use the `start` subcommand to manually start garbage collection on an entire datastore and the `status` subcommand to see attributes relating to the *garbage collection*.

This functionality can also be accessed in the web UI using the *Start Garbage Collection* button found in each datastore's **Prune & GC** tab.

12.2.3 Scheduled GC

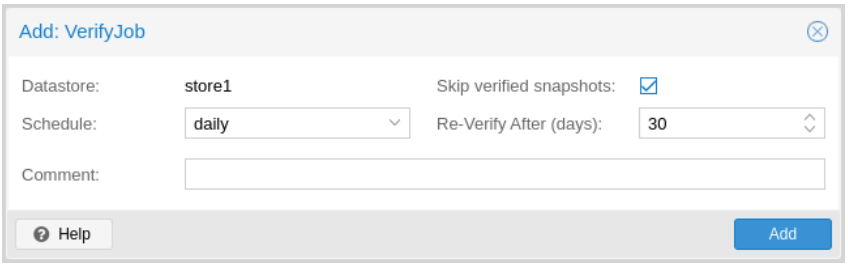
Normally, datastore admins don't want to bother triggering GC's manually. That's why you can configure a schedule to let Proxmox Backup Server handle it.

Setting or editing a datastore's GC schedule can be either done by using the `proxmox-backup-manager datastore update <datastore> --gc-schedule <schedule>` CLI command or the edit window in the web UI in each datastore's **Prune & GC** tab.

The GC scheduling uses the *Calendar Events* format.

Tip: You can disable automatic GC runs by clearing the schedule by either clearing the content of the field in the web UI or using the `proxmox-backup-manager datastore update <datastore> --delete gc-schedule` CLI command. This might be, for example, useful during maintenance or if you archive a datastore for good.

12.3 Verification



Proxmox Backup Server offers various verification options to ensure that backup data is intact. Verification is generally carried out through the creation of verify jobs. These are scheduled tasks that run verification at a given interval (see *Calendar Events*). With these, you can also set whether already verified snapshots are ignored, as well as set a time period, after which snapshots are checked again. The interface for creating verify jobs can be found under the **Verify Jobs** tab of the datastore.

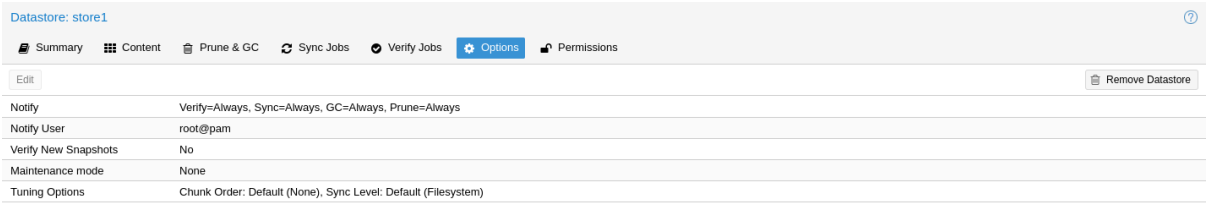
Note: It is recommended that you reverify all backups at least monthly, even if a previous verification was successful. This is because physical drives are susceptible to damage over time, which can cause an old, working backup to become corrupted in a process known as bit rot/data degradation. It is good practice to have a regularly recurring (hourly/daily) verification job, which checks new and expired backups, then another weekly/monthly job that will reverify everything. This way, there will be no surprises when it comes to restoring data.

Aside from using verify jobs, you can also run verification manually on entire datastores, backup groups or snapshots. To do this, navigate to the **Content** tab of the datastore and either click *Verify All* or select the *V.* icon from the **Actions** column in the table.

12.4 Notifications

Proxmox Backup Server can send you notification emails about automatically scheduled verification, garbage-collection and synchronization tasks results.

By default, notifications are sent to the email address configured for the *root@pam* user. You can instead set this user for each datastore.



You can also change the level of notification received per task type, the following options are available:

- Always: send a notification for any scheduled task, independent of the outcome
- Errors: send a notification for any scheduled task that results in an error
- Never: do not send any notification at all

12.5 Maintenance Mode

Proxmox Backup Server supports setting *read-only* and *offline* maintenance modes on a datastore.

Once enabled, depending on the mode, new reads and/or writes to the datastore are blocked, allowing an administrator to safely execute maintenance tasks, for example, on the underlying storage.

Internally Proxmox Backup Server tracks whether each datastore access is a write or read operation, so that it can gracefully enter the respective mode, by allowing conflicting operations that started before enabling the maintenance mode to finish.

CHAPTER

THIRTEEN

HOST SYSTEM ADMINISTRATION

Proxmox Backup is based on the famous Debian Linux distribution. This means that you have access to the entire range of Debian packages, and that the base system is well documented. The Debian Administrator's Handbook is available online, and provides a comprehensive introduction to the Debian operating system.

A standard Proxmox Backup installation uses the default repositories from Debian, so you get bug fixes and security updates through that channel. In addition, we provide our own package repository to roll out all Proxmox related packages. This includes updates to some Debian packages when necessary.

We also deliver a specially optimized Linux kernel, based on the Ubuntu kernel. This kernel includes drivers for ZFS.

The following sections will concentrate on backup related topics. They will explain things which are different on Proxmox Backup, or tasks which are commonly used on Proxmox Backup. For other topics, please refer to the standard Debian documentation.

13.1 ZFS on Linux

ZFS is a combined file system and logical volume manager, designed by Sun Microsystems. There is no need to manually compile ZFS modules - all packages are included.

By using ZFS, it's possible to achieve maximum enterprise features with low budget hardware, and also high performance systems by leveraging SSD caching or even SSD only setups. ZFS can replace expensive hardware raid cards with moderate CPU and memory load, combined with easy management.

General advantages of ZFS:

- Easy configuration and management with GUI and CLI.
- Reliable
- Protection against data corruption
- Data compression on file system level
- Snapshots
- Copy-on-write clone
- Various raid levels: RAID0, RAID1, RAID10, RAIDZ-1, RAIDZ-2 and RAIDZ-3
- Can use SSD for cache
- Self healing
- Continuous integrity checking
- Designed for high storage capacities

- Asynchronous replication over network
- Open Source
- Encryption

13.1.1 Hardware

ZFS depends heavily on memory, so it's recommended to have at least 8GB to start. In practice, use as much you can get for your hardware/budget. To prevent data corruption, we recommend the use of high quality ECC RAM.

If you use a dedicated cache and/or log disk, you should use an enterprise class SSD (for example, Intel SSD DC S3700 Series). This can increase the overall performance significantly.

IMPORTANT: Do not use ZFS on top of a hardware controller which has its own cache management. ZFS needs to directly communicate with disks. An HBA adapter or something like an LSI controller flashed in `IT` mode is recommended.

13.1.2 ZFS Administration

This section gives you some usage examples for common tasks. ZFS itself is really powerful and provides many options. The main commands to manage ZFS are *zfs* and *zpool*. Both commands come with extensive manual pages, which can be read with:

```
# man zpool
# man zfs
```

Create a new zpool

To create a new pool, at least one disk is needed. The *ashift* should have the same sector-size (2 power of *ashift*) or larger as the underlying disk.

```
# zpool create -f -o ashift=12 <pool> <device>
```

Create a new pool with RAID-0

Minimum 1 disk

```
# zpool create -f -o ashift=12 <pool> <device1> <device2>
```

Create a new pool with RAID-1

Minimum 2 disks

```
# zpool create -f -o ashift=12 <pool> mirror <device1> <device2>
```

Create a new pool with RAID-10

Minimum 4 disks

```
# zpool create -f -o ashift=12 <pool> mirror <device1> <device2> mirror <device3> <device4>
```

Create a new pool with RAIDZ-1

Minimum 3 disks

```
# zpool create -f -o ashift=12 <pool> raidz1 <device1> <device2> <device3>
```

Create a new pool with RAIDZ-2

Minimum 4 disks

```
# zpool create -f -o ashift=12 <pool> raidz2 <device1> <device2> <device3> <device4>
```

Create a new pool with cache (L2ARC)

It is possible to use a dedicated cache drive partition to increase the performance (use SSD).

For *<device>*, you can use multiple devices, as is shown in "Create a new pool with RAID*".

```
# zpool create -f -o ashift=12 <pool> <device> cache <cache_device>
```

Create a new pool with log (ZIL)

It is possible to use a dedicated cache drive partition to increase the performance (SSD).

For *<device>*, you can use multiple devices, as is shown in "Create a new pool with RAID*".

```
# zpool create -f -o ashift=12 <pool> <device> log <log_device>
```

Add cache and log to an existing pool

You can add cache and log devices to a pool after its creation. In this example, we will use a single drive for both cache and log. First, you need to create 2 partitions on the SSD with *parted* or *gdisk*

Important: Always use GPT partition tables.

The maximum size of a log device should be about half the size of physical memory, so this is usually quite small. The rest of the SSD can be used as cache.

```
# zpool add -f <pool> log <device-part1> cache <device-part2>
```

Changing a failed device

```
# zpool replace -f <pool> <old device> <new device>
```

Changing a failed bootable device

Depending on how Proxmox Backup was installed, it is either using *grub* or *systemd-boot* as a bootloader.

In either case, the first steps of copying the partition table, reissuing GUIDs and replacing the ZFS partition are the same. To make the system bootable from the new disk, different steps are needed which depend on the bootloader in use.

```
# sgdisk <healthy bootable device> -R <new device>
# sgdisk -G <new device>
# zpool replace -f <pool> <old zfs partition> <new zfs partition>
```

Note: Use the *zpool status -v* command to monitor how far the resilvering process of the new disk has progressed.

With *systemd-boot*:

```
# proxmox-boot-tool format <new ESP>
# proxmox-boot-tool init <new ESP>
```

Note: *ESP* stands for EFI System Partition, which is setup as partition #2 on bootable disks setup by the Proxmox Backup installer. For details, see *Setting up a new partition for use as synced ESP*.

With *grub*:

Usually *grub.cfg* is located in */boot/grub/grub.cfg*

```
# grub-install <new disk>
# grub-mkconfig -o /path/to/grub.cfg
```

Activate e-mail notification

ZFS comes with an event daemon, `ZED`, which monitors events generated by the ZFS kernel module. The daemon can also send emails upon ZFS events, such as pool errors. Newer ZFS packages ship the daemon in a separate package `zfs-zed`, which should already be installed by default in Proxmox Backup.

You can configure the daemon via the file `/etc/zfs/zed.d/zed.rc`, using your preferred editor. The required setting for email notfication is `ZED_EMAIL_ADDR`, which is set to `root` by default.

```
ZED_EMAIL_ADDR="root"
```

Please note that Proxmox Backup forwards mails to *root* to the email address configured for the root user.

Limit ZFS memory usage

It is good to use at most 50 percent (which is the default) of the system memory for ZFS ARC, to prevent performance degradation of the host. Use your preferred editor to change the configuration in */etc/modprobe.d/zfs.conf* and insert:

```
options zfs zfs_arc_max=8589934592
```

The above example limits the usage to 8 GiB ('8 * 2^30^').

Important: In case your desired *zfs_arc_max* value is lower than or equal to *zfs_arc_min* (which defaults to 1/32 of the system memory), *zfs_arc_max* will be ignored. Thus, for it to work in this case, you must set *zfs_arc_min* to at most *zfs_arc_max - 1*. This would require updating the configuration in */etc/modprobe.d/zfs.conf*, with:

```
options zfs zfs_arc_min=8589934591
options zfs zfs_arc_max=8589934592
```

This example setting limits the usage to 8 GiB ('8 * 2^30^') on systems with more than 256 GiB of total memory, where simply setting *zfs_arc_max* alone would not work.

Important: If your root file system is ZFS, you must update your initramfs every time this value changes.

```
# update-initramfs -u
```

Swap on ZFS

Swap-space created on a zvol may cause some issues, such as blocking the server or generating a high IO load.

We strongly recommend using enough memory, so that you normally do not run into low memory situations. Should you need or want to add swap, it is preferred to create a partition on a physical disk and use it as a swap device. You can leave some space free for this purpose in the advanced options of the installer. Additionally, you can lower the *swappiness* value. A good value for servers is 10:

```
# sysctl -w vm.swappiness=10
```

To make the swappiness persistent, open */etc/sysctl.conf* with an editor of your choice and add the following line:

```
vm.swappiness = 10
```

Table 1: Linux kernel *swappiness* parameter values :widths:auto

Value	Strategy
vm.swappiness = 0	The kernel will swap only to avoid an 'out of memory' condition
vm.swappiness = 1	Minimum amount of swapping without disabling it entirely.
vm.swappiness = 10	Sometimes recommended to improve performance when sufficient memory exists in a system.
vm.swappiness = 60	The default value.
vm.swappiness = 100	The kernel will swap aggressively.

ZFS compression

To activate compression:

```
# zpool set compression=lz4 <pool>
```

We recommend using the *lz4* algorithm, since it adds very little CPU overhead. Other algorithms such as *lzjb*, *zstd* and *gzip-N* (where *N* is an integer from *1-9* representing the compression ratio, where 1 is fastest and 9 is best compression) are also available. Depending on the algorithm and how compressible the data is, having compression enabled can even increase I/O performance.

You can disable compression at any time with:

```
# zfs set compression=off <dataset>
```

Only new blocks will be affected by this change.

ZFS special device

Since version 0.8.0, ZFS supports *special* devices. A *special* device in a pool is used to store metadata, deduplication tables, and optionally small file blocks.

A *special* device can improve the speed of a pool consisting of slow spinning hard disks with a lot of metadata changes. For example, workloads that involve creating, updating or deleting a large number of files will benefit from the presence of a *special* device. ZFS datasets can also be configured to store small files on the *special* device, which can further improve the performance. Use fast SSDs for the *special* device.

Important: The redundancy of the *special* device should match the one of the pool, since the *special* device is a point of failure for the entire pool.

Warning: Adding a *special* device to a pool cannot be undone!

To create a pool with *special* device and RAID-1:

```
# zpool create -f -o ashift=12 <pool> mirror <device1> <device2> special mirror <device3>
↪<device4>
```

Adding a *special* device to an existing pool with RAID-1:

```
# zpool add <pool> special mirror <device1> <device2>
```

ZFS datasets expose the *special_small_blocks*=<*size*> property. *size* can be *0* to disable storing small file blocks on the *special* device, or a power of two in the range between *512B* to *128K*. After setting this property, new file blocks smaller than *size* will be allocated on the *special* device.

Important: If the value for *special_small_blocks* is greater than or equal to the *recordsize* (default *128K*) of the dataset, *all* data will be written to the *special* device, so be careful!

Setting the *special_small_blocks* property on a pool will change the default value of that property for all child ZFS datasets (for example, all containers in the pool will opt in for small file blocks).

Opt in for all files smaller than 4K-blocks pool-wide:

```
# zfs set special_small_blocks=4K <pool>
```

Opt in for small file blocks for a single dataset:

```
# zfs set special_small_blocks=4K <pool>/<filesystem>
```

Opt out from small file blocks for a single dataset:

```
# zfs set special_small_blocks=0 <pool>/<filesystem>
```

Troubleshooting

Corrupt cache file

zfs-import-cache.service imports ZFS pools using the ZFS cache file. If this file becomes corrupted, the service won't be able to import the pools that it's unable to read from it.

As a result, in case of a corrupted ZFS cache file, some volumes may not be mounted during boot and must be mounted manually later.

For each pool, run:

```
# zpool set cachefile=/etc/zfs/zpool.cache POOLNAME
```

then, update the *initramfs* by running:

```
# update-initramfs -u -k all
```

and finally, reboot the node.

Another workaround to this problem is enabling the *zfs-import-scan.service*, which searches and imports pools via device scanning (usually slower).

13.2 Host Bootloader

Proxmox Backup currently uses one of two bootloaders, depending on the disk setup selected in the installer.

For EFI Systems installed with ZFS as the root filesystem `systemd-boot` is used. All other deployments use the standard `grub` bootloader (this usually also applies to systems which are installed on top of Debian).

13.2.1 Partitioning Scheme Used by the Installer

The Proxmox Backup installer creates 3 partitions on all disks selected for installation.

The created partitions are:

- A 1 MB BIOS Boot Partition (gdisk type EF02)
- A 512 MB EFI System Partition (ESP, gdisk type EF00)
- A third partition spanning the configured `hdsize` parameter or the remaining space available for the chosen storage type

Systems using ZFS as a root filesystem are booted with a kernel and initrd image stored on the 512 MB EFI System Partition. For legacy BIOS systems, `grub` is used, for EFI systems `systemd-boot` is used. Both are installed and configured to point to the ESPs.

`grub` in BIOS mode (`--target i386-pc`) is installed onto the BIOS Boot Partition of all selected disks on all systems booted with `grub` (that is, all installs with root on `ext4` or `xfs`, and installs with root on ZFS on non-EFI systems).

13.2.2 Synchronizing the Content of the ESP with `proxmox-boot-tool`

`proxmox-boot-tool` is a utility used to keep the contents of the EFI System Partitions properly configured and synchronized. It copies certain kernel versions to all ESPs and configures the respective bootloader to boot from the `vfat` formatted ESPs. In the context of ZFS as root filesystem, this means that you can use all the optional features on your root pool, instead of the subset which is also present in the ZFS implementation in `grub` or having to create a small, separate boot-pool (see: Booting ZFS on root with grub).

In setups with redundancy, all disks are partitioned with an ESP by the installer. This ensures the system boots, even if the first boot device fails or if the BIOS can only boot from a particular disk.

The ESPs are not kept mounted during regular operation. This helps to prevent filesystem corruption in the `vfat` formatted ESPs in case of a system crash, and removes the need to manually adapt `/etc/fstab` in case the primary boot device fails.

`proxmox-boot-tool` handles the following tasks:

- Formatting and setting up a new partition
- Copying and configuring new kernel images and initrd images to all listed ESPs
- Synchronizing the configuration on kernel upgrades and other maintenance tasks
- Managing the list of kernel versions which are synchronized
- Configuring the boot-loader to boot a particular kernel version (pinning)

You can view the currently configured ESPs and their state by running:

```
# proxmox-boot-tool status
```

Setting up a New Partition for use as Synced ESP

To format and initialize a partition as synced ESP, for example, after replacing a failed vdev in an rpool, `proxmox-boot-tool` from `pve-kernel-helper` can be used.

WARNING: the `format` command will format the `<partition>`. Make sure to pass in the right device/partition!

For example, to format an empty partition `/dev/sda2` as ESP, run the following:

```
# proxmox-boot-tool format /dev/sda2
```

To setup an existing, unmounted ESP located on `/dev/sda2` for inclusion in Proxmox Backup's kernel update synchronization mechanism, use the following:

```
# proxmox-boot-tool init /dev/sda2
```

Following this, *'/etc/kernel/proxmox-boot-uuids'* should contain a new line with the UUID of the newly added partition. The `init` command will also automatically trigger a refresh of all configured ESPs.

Updating the Configuration on all ESPs

To copy and configure all bootable kernels and keep all ESPs listed in `/etc/kernel/proxmox-boot-uuids` in sync, you just need to run:

```
# proxmox-boot-tool refresh
```

(Equivalent to running `update-grub` on systems with `ext4` or `xfs` on root).

This is necessary after making changes to the kernel commandline, or if you want to sync all kernels and initrds.

Note: Both `update-initramfs` and `apt` (when necessary) will automatically trigger a refresh.

Kernel Versions Considered by `proxmox-boot-tool`

The following kernel versions are configured by default:

- The currently running kernel
- The version being newly installed on package updates
- The two latest, already installed kernels
- The latest version of the second-to-last kernel series (e.g. 5.0, 5.3), if applicable
- Any manually selected kernels

Manually Keeping a Kernel Bootable

Should you wish to add a certain kernel and initrd image to the list of bootable kernels, use `proxmox-boot-tool kernel add`.

For example, run the following to add the kernel with ABI version `5.0.15-1-pve` to the list of kernels to keep installed and synced to all ESPs:

```
# proxmox-boot-tool kernel add 5.0.15-1-pve
```

`proxmox-boot-tool kernel list` will list all kernel versions currently selected for booting:

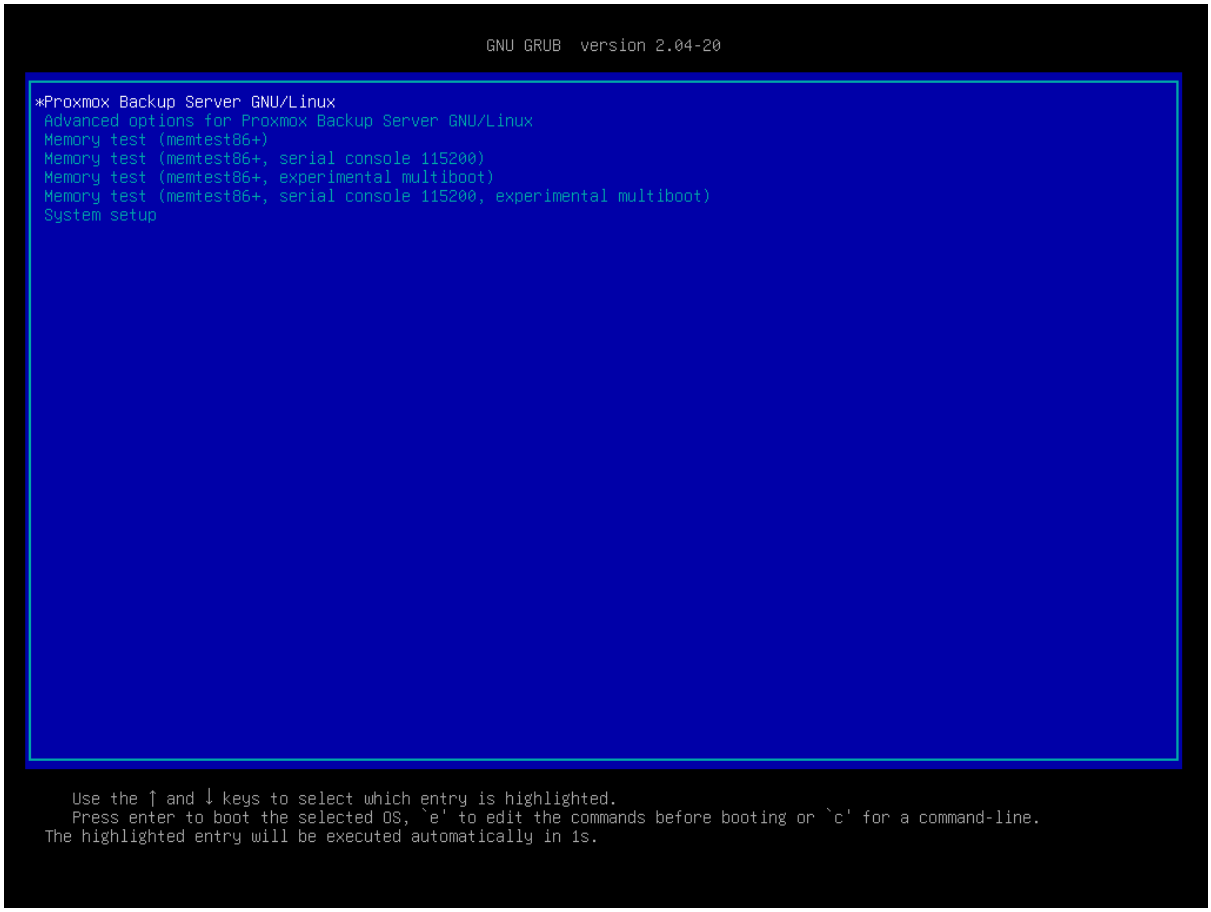
```
# proxmox-boot-tool kernel list
Manually selected kernels:
5.0.15-1-pve

Automatically selected kernels:
5.0.12-1-pve
4.15.18-18-pve
```

Run `proxmox-boot-tool kernel remove` to remove a kernel from the list of manually selected kernels, for example:

```
# proxmox-boot-tool kernel remove 5.0.15-1-pve
```

Note: It's required to run `proxmox-boot-tool refresh` to update all EFI System Partitions (ESPs) after a manual kernel addition or removal from above.

13.2.3 Determine which Bootloader is Used



The simplest and most reliable way to determine which bootloader is used, is to watch the boot process of the Proxmox Backup node.

You will either see the blue box of `grub` or the simple black on white `systemd-boot`.

Determining the bootloader from a running system might not be 100% accurate. The most reliable way is to run the following command:

```
# efibootmgr -v
```

If it returns a message that EFI variables are not supported, `grub` is used in BIOS/Legacy mode.

If the output contains a line that looks similar to the following, `grub` is used in UEFI mode.

```
Boot0005* proxmox     [...] File(\EFI\proxmox\grubx64.efi)
```

If the output contains a line similar to the following, `systemd-boot` is used.

```
Boot0006* Linux Boot Manager  [...] File(\EFI\systemd\systemd-bootx64.efi)
```

By running the following command, you can find out if `proxmox-boot-tool` is configured, which is a good indication of how the system is booted:

```
# proxmox-boot-tool status
```

13.2.4 Grub

`grub` has been the de facto standard for booting Linux systems for many years and is quite well documented (see the Grub Manual).

Configuration

Changes to the `grub` configuration are done via the defaults file `/etc/default/grub` or via config snippets in `/etc/default/grub.d`. To regenerate the configuration file after a change to the configuration, run:

```
# update-grub
```

Note: Systems using `proxmox-boot-tool` will call `proxmox-boot-tool refresh` upon `update-grub`

13.2.5 Systemd-boot

`systemd-boot` is a lightweight EFI bootloader. It reads the kernel and initrd images directly from the EFI Service Partition (ESP) where it is installed. The main advantage of directly loading the kernel from the ESP is that it does not need to reimplement the drivers for accessing the storage. In Proxmox Backup, *proxmox-boot-tool* is used to keep the configuration on the ESPs synchronized.

Configuration

`systemd-boot` is configured via the file `loader/loader.conf` in the root directory of an EFI System Partition (ESP). See the `loader.conf(5)` manpage for details.

Each bootloader entry is placed in a file of its own, in the directory `loader/entries/`

An example entry.conf looks like this (`/` refers to the root of the ESP):

```
title    Proxmox
version  5.0.15-1-pve
options   root=ZFS=rpool/ROOT/pve-1 boot=zfs
linux    /EFI/proxmox/5.0.15-1-pve/vmlinuz-5.0.15-1-pve
initrd   /EFI/proxmox/5.0.15-1-pve/initrd.img-5.0.15-1-pve
```

13.2.6 Editing the Kernel Commandline

You can modify the kernel commandline in the following places, depending on the bootloader used:

Grub

The kernel commandline needs to be placed in the variable `GRUB_CMDLINE_LINUX_DEFAULT` in the file `/etc/default/grub`. Running `update-grub` appends its content to all `linux` entries in `/boot/grub/grub.cfg`.

systemd-boot

The kernel commandline needs to be placed as one line in `/etc/kernel/cmdline`. To apply your changes, run `proxmox-boot-tool refresh`, which sets it as the `option` line for all config files in `loader/entries/proxmox-*.conf`.

13.2.7 Override the Kernel-Version for next Boot

To select a kernel that is not currently the default kernel, you can either:

- Use the boot loader menu that is displayed at the beginning of the boot process
- Use the `proxmox-boot-tool` to `pin` the system to a kernel version either once or permanently (until pin is reset).

This should help you work around incompatibilities between a newer kernel version and the hardware.

Note: Such a pin should be removed as soon as possible, so that all recent security patches from the latest kernel are also applied to the system.

For example, to permanently select the version `5.15.30-1-pve` for booting, you would run:

```
# proxmox-boot-tool kernel pin 5.15.30-1-pve
```

Tip: The pinning functionality works for all Proxmox Backup systems, not only those using `proxmox-boot-tool` to synchronize the contents of the ESPs, if your system does not use `proxmox-boot-tool` for synchronizing, you can also skip the `proxmox-boot-tool refresh` call in the end.

You can also set a kernel version to be booted on the next system boot only. This is useful, for example, to test if an updated kernel has resolved an issue, which caused you to `pin` a version in the first place:

```
# proxmox-boot-tool kernel pin 5.15.30-1-pve --next-boot
```

To remove any pinned version configuration, use the `unpin` subcommand:

```
# proxmox-boot-tool kernel unpin
```

While `unpin` has a `--next-boot` option as well, it is used to clear a pinned version set with `--next-boot`. As that happens already automatically on boot, invoking it manually is of little use.

After setting or clearing pinned versions, you also need to synchronize the content and configuration on the ESPs by running the `refresh` subcommand.

Tip: You will be prompted to automatically do for `proxmox-boot-tool` managed systems if you call the tool interactively.

```
# proxmox-boot-tool refresh
```

13.3 Certificate Management

Access to the API and thus the web-based administration interface is always encrypted through `https`. Each Proxmox Backup host creates by default its own (self-signed) certificate. This certificate is used for encrypted communication with the host's `proxmox-backup-proxy` service, for any API call between a user or backup-client and the web-interface.

Certificate verification when sending backups to a Proxmox Backup server is either done based on pinning the certificate fingerprints in the storage/remote configuration, or by using certificates, signed by a trusted certificate authority.

13.3.1 Certificates for the API and SMTP

Proxmox Backup stores its certificate and key in:

- `/etc/proxmox-backup/proxy.pem`
- `/etc/proxmox-backup/proxy.key`

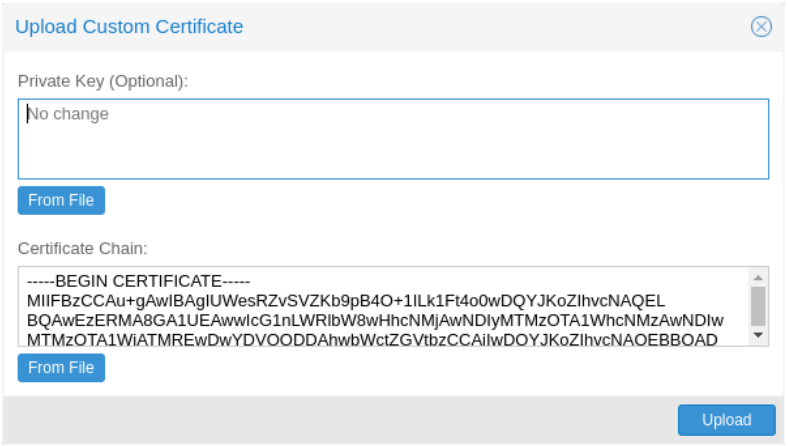
You have the following options for the certificate:

1. Keep using the default self-signed certificate in `/etc/proxmox-backup/proxy.pem`.
2. Use an externally provided certificate (for example, signed by a commercial Certificate Authority (CA)).
3. Use an ACME provider like Let's Encrypt to get a trusted certificate with automatic renewal; this is also integrated in the Proxmox Backup API and web interface.

Certificates are managed through the Proxmox Backup web-interface/API or using the the `proxmox-backup-manager` CLI tool.

13.3.2 Upload Custom Certificate

If you already have a certificate which you want to use for a Proxmox Mail Gateway host, you can simply upload that certificate over the web interface.



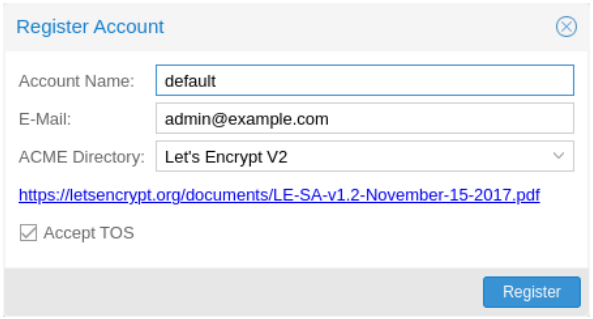
Note that any certificate key files must not be password protected.

13.3.3 Trusted certificates via Let's Encrypt (ACME)

Proxmox Backup includes an implementation of the **A**utomatic **C**ertificate **M**anagement **E**nvironment (**ACME**) protocol, allowing Proxmox Backup admins to use an ACME provider like Let's Encrypt for easy setup of TLS certificates, which are accepted and trusted by modern operating systems and web browsers out of the box.

Currently, the two ACME endpoints implemented are the Let's Encrypt (LE) production and staging environments. Our ACME client supports validation of `http-01` challenges using a built-in web server and validation of `dns-01` challenges using a DNS plugin supporting all the DNS API endpoints acme.sh does.

ACME Account



You need to register an ACME account per cluster, with the endpoint you want to use. The email address used for that account will serve as the contact point for renewal-due or similar notifications from the ACME endpoint.

You can register or deactivate ACME accounts over the web interface `Certificates -> ACME Accounts` or using the `proxmox-backup-manager` command line tool.

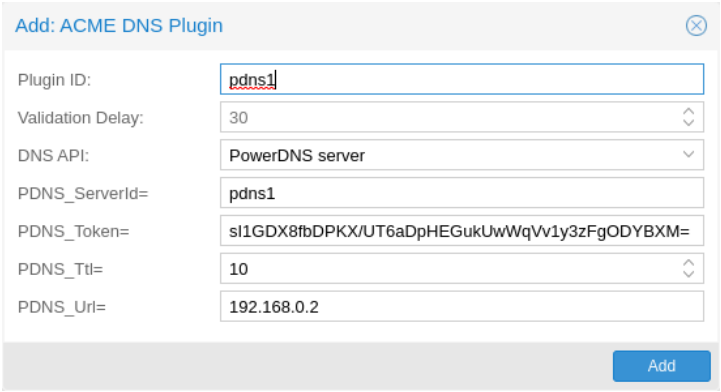
```
proxmox-backup-manager acme account register <account-name> <mail@example.com>
```

Tip: Because of rate-limits you should use LE `staging` for experiments or if you use ACME for the very first time until all is working there, and only then switch over to the production directory.

ACME Plugins

The ACME plugin's role is to provide automatic verification that you, and thus the Proxmox Backup server under your operation, are the real owner of a domain. This is the basic building block of automatic certificate management.

The ACME protocol specifies different types of challenges, for example the `http-01`, where a web server provides a file with a specific token to prove that it controls a domain. Sometimes this isn't possible, either because of technical limitations or if the address of a record is not reachable from the public internet. The `dns-01` challenge can be used in such cases. This challenge is fulfilled by creating a certain DNS record in the domain's zone.

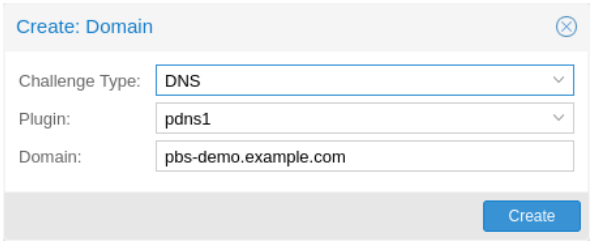


Proxmox Backup supports both of those challenge types out of the box, you can configure plugins either over the web interface under `Certificates -> ACME Challenges`, or using the `proxmox-backup-manager acme plugin add` command.

ACME Plugin configurations are stored in `/etc/proxmox-backup/acme/plugins.cfg`.

Domains

You can add new or manage existing domain entries under `Certificates`, or using the `proxmox-backup-manager` command.



After configuring the desired domain(s) for a node and ensuring that the desired ACME account is selected, you can order your new certificate over the web-interface. On success, the interface will reload after roughly 10 seconds.

Renewal will happen *automatically*

13.3.4 ACME HTTP Challenge Plugin

There is always an implicitly configured `standalone` plugin for validating `http-01` challenges via the built-in web server spawned on port 80.

Note: The name `standalone` means that it can provide the validation on its own, without any third party service.

There are a few prerequisites to use this for certificate management with Let's Encrypts ACME.

- You have to accept the ToS of Let's Encrypt to register an account.
- **Port 80** of the node needs to be reachable from the internet.
- There **must** be no other listener on port 80.
- The requested (sub)domain needs to resolve to a public IP of the Proxmox Backup host.

13.3.5 ACME DNS API Challenge Plugin

On systems where external access for validation via the `http-01` method is not possible or desired, it is possible to use the `dns-01` validation method. This validation method requires a DNS server that allows provisioning of TXT records via an API.

Configuring ACME DNS APIs for validation

Proxmox Backup re-uses the DNS plugins developed for the `acme.sh`[1] project. Please refer to its documentation for details on configuration of specific APIs.

The easiest way to configure a new plugin with the DNS API is using the web interface (`Certificates -> ACME Accounts/Challenges`).

Here you can add a new challenge plugin by selecting your API provider and entering the credential data to access your account over their API.

Tip: See the acme.sh How to use DNS API wiki for more detailed information about getting API credentials for your provider. Configuration values do not need to be quoted with single or double quotes; for some plugins that is even an error.

As there are many DNS providers and API endpoints, Proxmox Backup automatically generates the form for the credentials, but not all providers are annotated yet. For those you will see a bigger text area, into which you simply need to copy all the credential's `KEY=VALUE` pairs.

DNS Validation through CNAME Alias

A special `alias` mode can be used to handle validation on a different domain/DNS server, in case your primary/real DNS does not support provisioning via an API. Manually set up a permanent `CNAME` record for `_acme-challenge.domain1.example` pointing to `_acme-challenge.domain2.example`, and set the `alias` property in the Proxmox Backup node configuration file `/etc/proxmox-backup/node.cfg` to `domain2.example` to allow the DNS server of `domain2.example` to validate all challenges for `domain1.example`.

Wildcard Certificates

Wildcard DNS names start with a `*.` prefix and are considered valid for all (one-level) subdomain names of the verified domain. So a certificate for `*.domain.example` is valid for `foo.domain.example` and `bar.domain.example`, but not for `baz.foo.domain.example`.

Currently, you can only create wildcard certificates with the DNS challenge type.

Combination of Plugins

Combining `http-01` and `dns-01` validation is possible in case your node is reachable via multiple domains with different requirements / DNS provisioning capabilities. Mixing DNS APIs from multiple providers or instances is also possible by specifying different plugin instances per domain.

Tip: Accessing the same service over multiple domains increases complexity and should be avoided if possible.

[1] acme.sh https://github.com/acmesh-official/acme.sh

13.3.6 Automatic renewal of ACME certificates

If a node has been successfully configured with an ACME-provided certificate (either via `proxmox-backup-manager` or via the web-interface/API), the certificate will be renewed automatically by the `proxmox-backup-daily-update.service`. Currently, renewal is triggered if the certificate either has already expired or if it will expire in the next 30 days.

13.3.7 Manually Change Certificate over Command-Line

If you want to get rid of certificate verification warnings, you have to generate a valid certificate for your server.

Log in to your Proxmox Backup via ssh or use the console:

```
openssl req -newkey rsa:2048 -nodes -keyout key.pem -out req.pem
```

Follow the instructions on the screen, for example:

```
Country Name (2 letter code) [AU]: AT
State or Province Name (full name) [Some-State]:Vienna
Locality Name (eg, city) []:Vienna
Organization Name (eg, company) [Internet Widgits Pty Ltd]: Proxmox GmbH
Organizational Unit Name (eg, section) []:Proxmox Backup
Common Name (eg, YOUR name) []: yourproxmox.yourdomain.com
Email Address []:support@yourdomain.com

Please enter the following 'extra' attributes to be sent with your certificate request
A challenge password []: not necessary
An optional company name []: not necessary
```

After you have finished the certificate request, you have to send the file `req.pem` to your Certification Authority (CA). The CA will issue the certificate (BASE64 encoded), based on your request – save this file as `cert.pem` to your Proxmox Backup.

To activate the new certificate, do the following on your Proxmox Backup

```
cp key.pem /etc/proxmox-backup/proxy.key
cp cert.pem /etc/proxmox-backup/proxy.pem
```

Then restart the API servers:

```
systemctl restart proxmox-backup-proxy
```

Test your new certificate, using your browser.

Note: To transfer files to and from your Proxmox Backup, you can use secure copy: If your desktop runs Linux, you can use the `scp` command line tool. If your desktop PC runs windows, please use an scp client like WinSCP (see https://winscp.net/).

13.4 Service Daemons

13.4.1 `proxmox-backup-proxy`

This daemon exposes the whole Proxmox Backup Server API on TCP port 8007 using HTTPS. It runs as user `backup` and has very limited permissions. Operations requiring more permissions are forwarded to the local `proxmox-backup` service.

13.4.2 proxmox-backup

This daemon exposes the Proxmox Backup Server management API on `127.0.0.1:82`. It runs as `root` and has permission to do all privileged operations.

NOTE: The daemon listens to a local address only, so you cannot access it from outside. The `proxmox-backup-proxy` daemon exposes the API to the outside world.

13.5 Command Line Tools

13.5.1 proxmox-backup-client

This tool implements a backup server client, i.e. it can connect to a backup servers to issue management commands and to create or restore backups.

13.5.2 proxmox-backup-manager

This tool exposes the whole backup server management API on the command line.

13.5.3 proxmox-tape

This tool can configure and manage tape backups.

13.5.4 pmt

The `pmt` command controls Linux tape devices.

13.5.5 pmtx

The `pmtx` command controls SCSI media changer devices (tape autoloader).

13.5.6 pxar

`pxar` is a command line utility for creating and manipulating archives in the *Proxmox File Archive Format (.pxar)*. It is inspired by casync file archive format, which caters to a similar use-case. The `.pxar` format is adapted to fulfill the specific needs of the Proxmox Backup Server, for example, efficient storage of hard links. The format is designed to reduce the required storage on the server by achieving a high level of deduplication.

Creating an Archive

Run the following command to create an archive of a folder named `source`:

```
# pxar create archive.pxar /path/to/source
```

This will create a new archive called `archive.pxar` with the contents of the `source` folder.

Note: `pxar` will not overwrite any existing archives. If an archive with the same name is already present in the target folder, the creation will fail.

By default, `pxar` will skip certain mount points and will not follow device boundaries. This design decision is based on the primary use case of creating archives for backups. It makes sense to ignore the contents of certain temporary or system specific files in a backup. To alter this behavior and follow device boundaries, use the `--all-file-systems` flag.

It is possible to exclude certain files and/or folders from the archive by passing the `--exclude` parameter with `gitignore`-style match patterns.

For example, you can exclude all files ending in `.txt` from the archive by running:

```
# pxar create archive.pxar /path/to/source --exclude '**/*.txt'
```

Be aware that the shell itself will try to expand glob patterns before invoking `pxar`. In order to avoid this, all globs have to be quoted correctly.

It is possible to pass the `--exclude` parameter multiple times, in order to match more than one pattern. This allows you to use more complex file inclusion/exclusion behavior. However, it is recommended to use `.pxarexclude` files instead for such cases.

For example you might want to exclude all `.txt` files except a specific one from the archive. This would be achieved via the negated match pattern, prefixed by `!`. All the glob patterns are relative to the `source` directory.

```
# pxar create archive.pxar /path/to/source --exclude '**/*.txt' --exclude '!/folder/file.txt'
```

Note: The order of the glob match patterns matters, as later ones override earlier ones. Permutations of the same patterns lead to different results.

`pxar` will store the list of glob match patterns passed as parameters via the command line, in a file called `.pxarexclude-cli`, at the root of the archive. If a file with this name is already present in the source folder during archive creation, this file is not included in the archive, and the file containing the new patterns is added to the archive instead. The original file is not altered.

A more convenient and persistent way to exclude files from the archive is by placing the glob match patterns in `.pxarexclude` files. It is possible to create and place these files in any directory of the filesystem tree. These files must contain one pattern per line, and later patterns override earlier ones. The patterns control file exclusions of files present within the given directory or further below it in the tree. The behavior is the same as described in *Creating Backups*.

Extracting an Archive

An existing archive, `archive.pxar`, is extracted to a `target` directory with the following command:

```
# pxar extract archive.pxar /path/to/target
```

If no target is provided, the contents of the archive is extracted to the current working directory.

In order to restore only parts of an archive, single files, and/or folders, it is possible to pass the corresponding glob match patterns as additional parameters or to use the patterns stored in a file:

```
# pxar extract etc.pxar /restore/target/etc --pattern '**/*.conf'
```

The above example restores all `.conf` files encountered in any of the sub-folders in the archive `etc.pxar` to the target `/restore/target/etc`. A path to the file containing match patterns can be specified using the `--files-from` parameter.

List the Contents of an Archive

To display the files and directories contained in an archive `archive.pxar`, run the following command:

```
# pxar list archive.pxar
```

This displays the full path of each file or directory with respect to the archive's root.

Mounting an Archive

`pxar` allows you to mount and inspect the contents of an archive via FUSE. In order to mount an archive named `archive.pxar` to the mount point `/mnt`, run the command:

```
# pxar mount archive.pxar /mnt
```

Once the archive is mounted, you can access its content under the given mount point.

```
# cd /mnt
# ls
bin   dev  home  lib32  libx32      media  opt   root  sbin  sys  usr
boot  etc  lib   lib64  lost+found  mnt    proc  run   srv   tmp  var
```

13.5.7 `proxmox-file-restore`

Command line tool for restoring files and directories from Proxmox Backup archives. In contrast to proxmox-backup-client, this supports both container/host and VM backups.

13.5.8 `proxmox-backup-debug`

Implements debugging functionality to inspect Proxmox Backup datastore files, verify the integrity of chunks.

The 'diff' subcommand allows comparing .pxar archives for two arbitrary snapshots. A list of added/modified/deleted files will be displayed.

Also contains an 'api' subcommand where arbitrary api paths can be called (get/create/set/delete) as well as display their parameters (usage) and their child-links (ls).

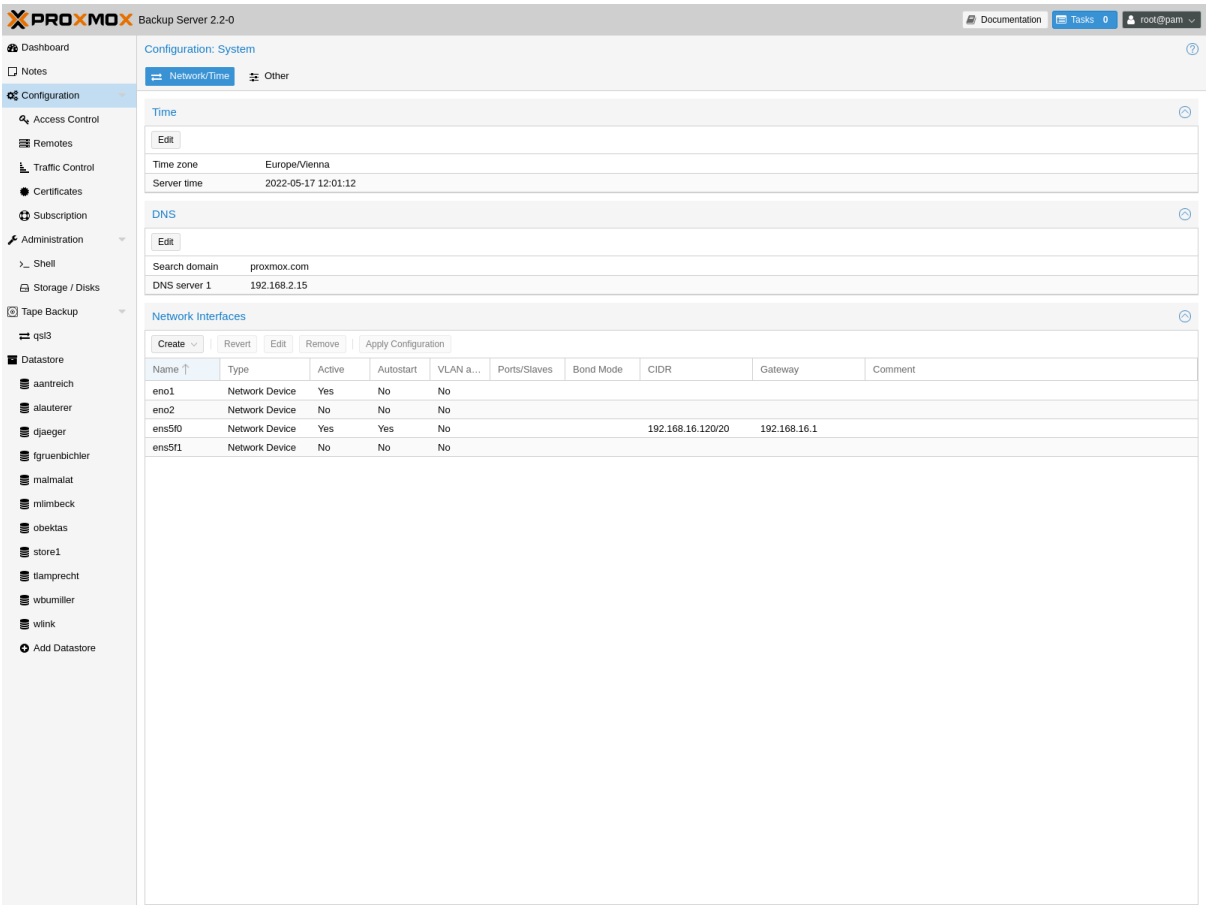
By default, it connects to the proxmox-backup-proxy on localhost via https, but by setting the environment variable *PROXMOX_DEBUG_API_CODE* to *1* the tool directly calls the corresponding code.

Warning: Using *PROXMOX_DEBUG_API_CODE* can be dangerous and is only intended for debugging purposes. It is not intended for use on a production system.

CHAPTER

FOURTEEN

NETWORK MANAGEMENT



Proxmox Backup Server provides both a web interface and a command line tool for network configuration. You can find the configuration options in the web interface under the **Network Interfaces** section of the **Configuration** menu tree item. The command line tool is accessed via the `network` subcommand. These interfaces allow you to carry out some basic network management tasks, such as adding, configuring, and removing network interfaces.

Note: Any changes made to the network configuration are not applied, until you click on **Apply Configuration** or enter the `network reload` command. This allows you to make many changes at once. It also allows you to ensure that your changes are correct before applying them, as making a mistake here can render the server inaccessible over the network.

To get a list of available interfaces, use the following command:

```
# proxmox-backup-manager network list
┌──────┬──────┬──────┬──────┬──────┬──────┬──────┐
```

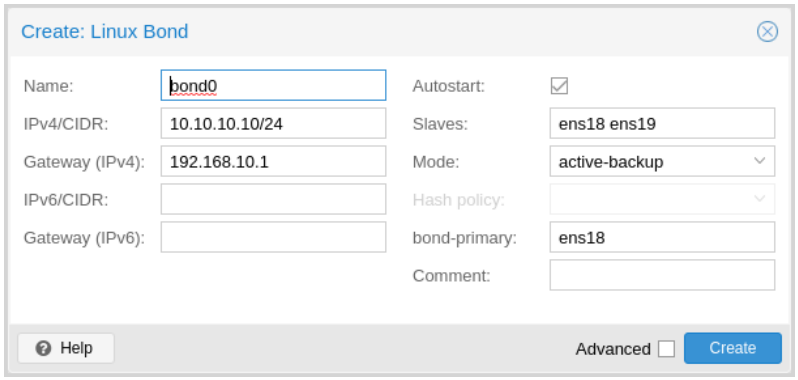
(continues on next page)

(continued from previous page)

name	type	autostart	method	address	gateway	ports/slaves
bond0	bond	1	static	x.x.x.x/x	x.x.x.x	ens18 ens19
ens18	eth	1	manual			
ens19	eth	1	manual			

To add a new network interface, use the `create` subcommand with the relevant parameters. For example, you may want to set up a bond, for the purpose of network redundancy. The following command shows a template for creating the bond shown in the list above:

```
# proxmox-backup-manager network create bond0 --type bond --bond_mode active-backup --slaves␣
↪ens18,ens19 --autostart true --cidr x.x.x.x/x --gateway x.x.x.x
```



You can make changes to the configuration of a network interface with the `update` subcommand:

```
# proxmox-backup-manager network update bond0 --cidr y.y.y.y/y
```

You can also remove a network interface:

```
# proxmox-backup-manager network remove bond0
```

The pending changes for the network configuration file will appear at the bottom of the web interface. You can also view these changes, by using the command:

```
# proxmox-backup-manager network changes
```

If you would like to cancel all changes at this point, you can either click on the **Revert** button or use the following command:

```
# proxmox-backup-manager network revert
```

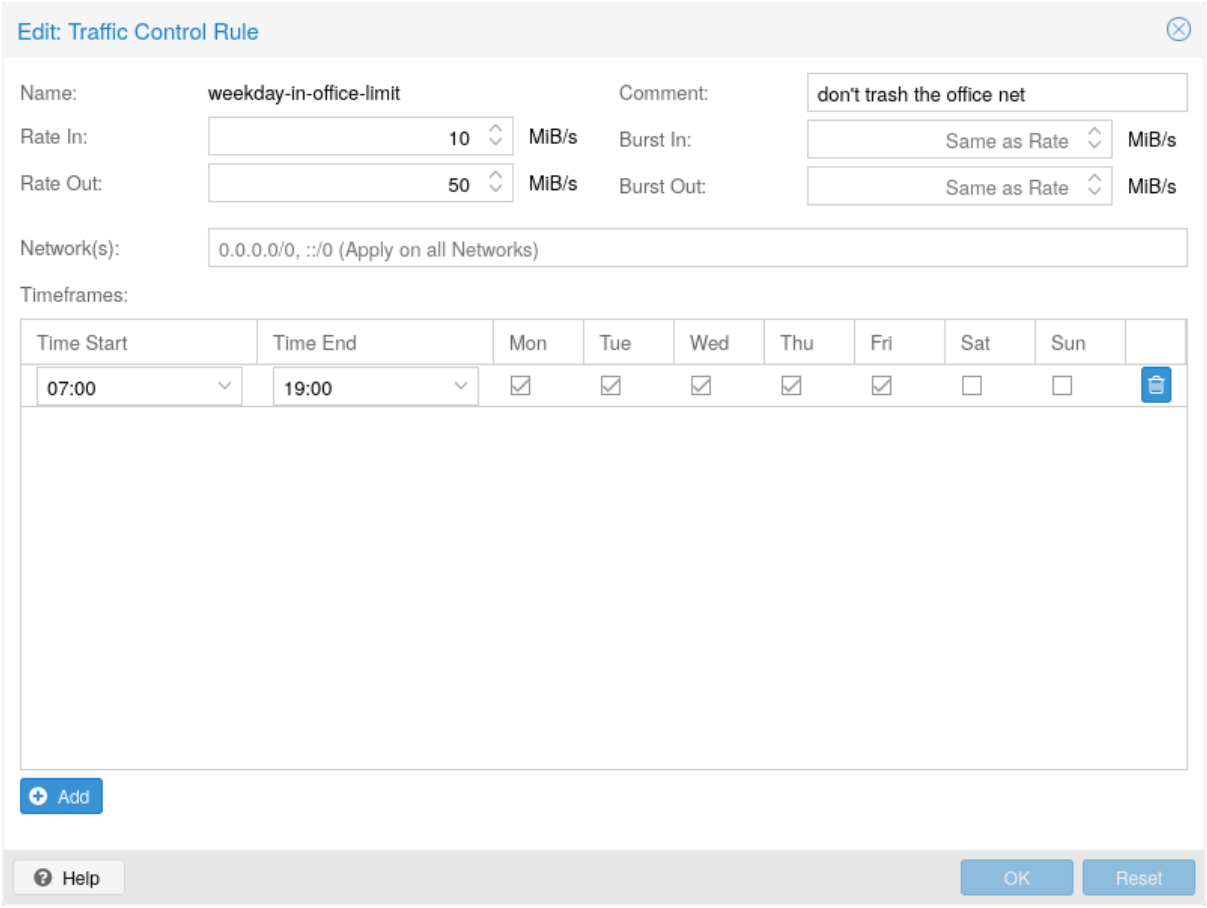
If you are happy with the changes and would like to write them into the configuration file, select **Apply Configuration**. The corresponding command is:

```
# proxmox-backup-manager network reload
```

Note: This command and corresponding GUI button rely on the `ifreload` command, from the package `ifupdown2`. This package is included within the Proxmox Backup Server installation, however, you may have to install it yourself, if you have installed Proxmox Backup Server on top of Debian or a Proxmox VE version prior to version 7.

You can also configure DNS settings, from the **DNS** section of **Configuration** or by using the `dns` subcommand of `proxmox-backup-manager`.

14.1 Traffic Control



Creating and restoring backups can produce a lot of traffic, can impact shared storage and other users on the network.

With Proxmox Backup Server, you can constrain network traffic for clients within specified networks using a token bucket filter (TBF).

This allows you to avoid network congestion and prioritize traffic from certain hosts.

You can manage the traffic controls either via the web-interface or using the `traffic-control` commands of the `proxmox-backup-manager` command-line tool.

Note: Sync jobs on the server are not affected by the configured rate-in limits. If you want to limit the incoming traffic that a pull-based sync job generates, you need to setup a job-specific rate-in limit. See *Sync Jobs*.

The following command adds a traffic control rule to limit all IPv4 clients (network `0.0.0.0/0`) to 100 MB/s:

```
# proxmox-backup-manager traffic-control create rule0  --network 0.0.0.0/0 \
  --rate-in 100MB --rate-out 100MB \
  --comment "Default rate limit (100MB/s) for all clients"
```

Note: To limit both IPv4 and IPv6 network spaces, you need to pass two network parameters `::/0` and `0.0.0.0/0`.

It is possible to restrict rules to certain time frames, for example the company's office hours:

Tip: You can use SI (base 10: KB, MB, ...) or IEC (base 2: KiB, MiB, ...) units.

```
# proxmox-backup-manager traffic-control update rule0  \
  --timeframe "mon..fri 8-12" \
  --timeframe "mon..fri 14:30-18"
```

If there are multiple rules, the server chooses the one with the smaller network. For example, we can overwrite the setting for our private network (and the server itself) with:

```
# proxmox-backup-manager traffic-control create rule1 \
  --network 192.168.2.0/24 \
  --network 127.0.0.0/8 \
  --rate-in 20GB --rate-out 20GB \
  --comment "Use 20GB/s for the local network"
```

Note: The behavior is undefined if there are several rules for the same network.

If there are multiple rules which match a specific network, they will all be applied, which means that the smallest one wins, as it's bucket fills up the fastest.

To list the current rules, use:

```
# proxmox-backup-manager traffic-control list
```

name	rate-in	rate-out	network	timeframe	...
rule0	100 MB	100 MB	["0.0.0.0/0"]	["mon..fri	...
rule1	20 GB	20 GB	["192.168.2.0/24", ...]		...

Rules can also be removed:

```
# proxmox-backup-manager traffic-control remove rule1
```

To show the state (current data rate) of all configured rules use:

```
# proxmox-backup-manager traffic-control traffic
```

name	cur-rate-in	cur-rate-out
rule0	0 B	0 B
rule1	1.161 GiB	19.146 KiB

CHAPTER

FIFTEEN

TECHNICAL OVERVIEW

15.1 Datastores

A Datastore is the logical place where *Backup Snapshots* and their chunks are stored. Snapshots consist of a manifest, blobs, and dynamic- and fixed-indexes (see *Terminology*), and are stored in the following directory structure:

<datastore-root>/<type>/<id>/<time>/

The deduplication of datastores is based on reusing chunks, which are referenced by the indexes in a backup snapshot. This means that multiple indexes can reference the same chunks, reducing the amount of space needed to contain the data (even across backup snapshots).

15.2 Snapshots

A Snapshot is the collection of manifest, blobs and indexes that represent a backup. When a client creates a snapshot, it can upload blobs (single files which are not chunked, e.g. the client log), or one or more indexes (fixed or dynamic).

When uploading an index, the client first has to read the source data, chunk it and send the data as chunks with their identifying checksum to the server.

If there is a previous Snapshot in the backup group, the client can first download the chunk list of the previous Snapshot. If it detects a chunk that already exists on the server, it can send only the checksum instead of data and checksum. This way the actual upload of Snapshots is incremental while each Snapshot references all chunks and is thus a full backup.

After uploading all data, the client has to signal to the server that the backup is finished. If that is not done before the connection closes, the server will remove the unfinished snapshot.

15.3 Chunks

A chunk is some (possibly encrypted) data with a CRC-32 checksum at the end and a type marker at the beginning. It is identified by the SHA-256 checksum of its content.

To generate such chunks, backup data is split either into fixed-size or dynamically sized chunks. The same content will be hashed to the same checksum.

The chunks of a datastore are found in

<datastore-root>/.chunks/

This chunk directory is further subdivided by the first four bytes of the chunk's checksum, so a chunk with the checksum

a342e8151cbf439ce65f3df696b54c67a114982cc0aa751f2852c2f7acc19a8b

lives in

<datastore-root>/.chunks/a342/

This is done to reduce the number of files per directory, as having many files per directory can be bad for file system performance.

These chunk directories ('0000'-'ffff') will be preallocated when a datastore is created.

15.3.1 Fixed-Sized Chunks

For block based backups (like VMs), fixed-sized chunks are used. The content (disk image), is split into chunks of the same length (typically 4 MiB).

This works very well for VM images, since the file system on the guest most often tries to allocate files in contiguous pieces, so new files get new blocks, and changing existing files changes only their own blocks.

As an optimization, VMs in Proxmox VE can make use of 'dirty bitmaps', which can track the changed blocks of an image. Since these bitmaps are also a representation of the image split into chunks, there is a direct relation between the dirty blocks of the image and chunks which need to be uploaded. Thus, only modified chunks of the disk need to be uploaded to a backup.

Since the image is always split into chunks of the same size, unchanged blocks will result in identical checksums for those chunks, so such chunks do not need to be backed up again. This way storage snapshots are not needed to find the changed blocks.

For consistency, Proxmox VE uses a QEMU internal snapshot mechanism, that does not rely on storage snapshots either.

15.3.2 Dynamically Sized Chunks

When working with file-based systems rather than block-based systems, using fixed-sized chunks is not a good idea, since every time a file would change in size, the remaining data would be shifted around, resulting in many chunks changing and the amount of deduplication being reduced.

To improve this, Proxmox Backup Server uses dynamically sized chunks instead. Instead of splitting an image into fixed sizes, it first generates a consistent file archive (*pxar*) and uses a rolling hash over this on-the-fly generated archive to calculate chunk boundaries.

We use a variant of Buzhash which is a cyclic polynomial algorithm. It works by continuously calculating a checksum while iterating over the data, and on certain conditions, it triggers a hash boundary.

Assuming that most files on the system that is to be backed up have not changed, eventually the algorithm triggers the boundary on the same data as a previous backup, resulting in chunks that can be reused.

15.3.3 Encrypted Chunks

Encrypted chunks are a special case. Both fixed- and dynamically sized chunks can be encrypted, and they are handled in a slightly different manner than normal chunks.

The hashes of encrypted chunks are calculated not with the actual (encrypted) chunk content, but with the plain-text content, concatenated with the encryption key. This way, two chunks with the same data but encrypted with different keys generate two different checksums and no collisions occur for multiple encryption keys.

This is done to speed up the client part of the backup, since it only needs to encrypt chunks that are actually getting uploaded. Chunks that exist already in the previous backup, do not need to be encrypted and uploaded.

15.4 Caveats and Limitations

15.4.1 Notes on Hash Collisions

Every hashing algorithm has a chance to produce collisions, meaning two (or more) inputs generate the same checksum. For SHA-256, this chance is negligible. To calculate the chances of such a collision, one can use the ideas of the 'birthday problem' from probability theory. For big numbers, this is actually unfeasible to calculate with regular computers, but there is a good approximation:

$$p(n, d) = 1 - e^{-n^2/(2d)}$$

Where *n* is the number of tries, and *d* is the number of possibilities. For a concrete example, lets assume a large datastore of 1 PiB and an average chunk size of 4 MiB. That means $n = 268435456$ tries, and $d = 2^{256}$ possibilities. Inserting those values in the formula from earlier you will see that the probability of a collision in that scenario is:

$$3.1115 * 10^{-61}$$

For context, in a lottery game of guessing 6 numbers out of 45, the chance to correctly guess all 6 numbers is only $1.2277 * 10^{-7}$. This means the chance of a collision is about the same as winning 13 such lottery games *in a row*.

In conclusion, it is extremely unlikely that such a collision would occur by accident in a normal datastore.

Additionally, SHA-256 is prone to length extension attacks, but since there is an upper limit for how big the chunks are, this is not a problem, because a potential attacker cannot arbitrarily add content to the data beyond that limit.

15.4.2 File-Based Backup

Since dynamically sized chunks (for file-based backups) are created on a custom archive format (pxar) and not over the files directly, there is no relation between the files and chunks. This means that the Proxmox Backup Client has to read all files again for every backup, otherwise it would not be possible to generate a consistent, independent pxar archive where the original chunks can be reused. Note that in spite of this, only new or changed chunks will be uploaded.

15.4.3 Verification of Encrypted Chunks

For encrypted chunks, only the checksum of the original (plaintext) data is available, making it impossible for the server (without the encryption key) to verify its content against it. Instead only the CRC-32 checksum gets checked.

15.5 Troubleshooting

Index files(*.fidx*, *.didx*) contain information about how to rebuild a file. More precisely, they contain an ordered list of references to the chunks that the original file was split into. If there is something wrong with a snapshot, it might be useful to find out which chunks are referenced in it, and check whether they are present and intact. The `proxmox-backup-debug` command line tool can be used to inspect such files and recover their contents. For example, to get a list of the referenced chunks of a *.fidx* index:

```
# proxmox-backup-debug inspect file drive-scsi0.img.fidx
```

The same command can be used to inspect *.blob* files. Without the `--decode` parameter, just the size and the encryption type, if any, are printed. If `--decode` is set, the blob file is decoded into the specified file ('-' will decode it directly to stdout).

The following example would print the decoded contents of *qemu-server.conf.blob*. If the file you're trying to inspect is encrypted, a path to the key file must be provided using `--keyfile`.

```
# proxmox-backup-debug inspect file qemu-server.conf.blob --decode -
```

You can also check in which index files a specific chunk file is referenced with:

```
# proxmox-backup-debug inspect chunk␣
↪b531d3ffc9bd7c65748a61198c060678326a431db7eded874c327b7986e595e0 --reference-filter /path/
↪in/a/datastore/directory
```

Here `--reference-filter` specifies where index files should be searched. This can be an arbitrary path. If, for some reason, the filename of the chunk was changed, you can explicitly specify the digest using `--digest`. By default, the chunk filename is used as the digest to look for. If no `--reference-filter` is specified, it will only print the CRC and encryption status of the chunk. You can also decode chunks, by setting the `--decode` flag. If the chunk is encrypted, a `--keyfile` must be provided, in order to decode it.

15.5.1 Restore without a Running Proxmox Backup Server

It's possible to restore specific files from a snapshot, without a running Proxmox Backup Server instance, using the `recover` subcommand, provided you have access to the intact index and chunk files. Note that you also need the corresponding key file if the backup was encrypted.

```
# proxmox-backup-debug recover index drive-scsi0.img.fidx /path/to/.chunks
```

In the above example, the */path/to/.chunks* argument is the path to the directory that contains the chunks, and *drive-scsi0.img.fidx* is the index file of the file you'd like to restore. Both paths can be absolute or relative. With `--skip-crc`, it's possible to disable the CRC checks of the chunks. This will speed up the process slightly and allow for trying to restore (partially) corrupt chunks. It's recommended to always try without the skip-CRC option first.

CHAPTER

SIXTEEN

FAQ

16.1 What distribution is Proxmox Backup Server (PBS) based on?

Proxmox Backup Server is based on Debian GNU/Linux.

16.2 Which platforms are supported as a backup source (client)?

The client tool works on most modern Linux systems, meaning you are not limited to Debian-based backups.

16.3 Will Proxmox Backup Server run on a 32-bit processor?

Proxmox Backup Server only supports 64-bit CPUs (AMD or Intel). There are no future plans to support 32-bit processors.

16.4 How long will my Proxmox Backup Server version be supported?

Table 1: Table Title

Proxmox Backup Version	Debian Version	First Release	Debian EOL	Proxmox Backup EOL
Proxmox Backup 3	Debian 12 (Bookworm)	2023-06	TBA	TBA
Proxmox Backup 2	Debian 11 (Bullseye)	2021-07	2024-07	2024-07
Proxmox Backup 1	Debian 10 (Buster)	2020-11	2022-08	2022-07

16.5 How can I upgrade Proxmox Backup Server to the next point release?

Minor version upgrades, for example upgrading from Proxmox Backup Server in version 3.1 to 3.2 or 3.3, can be done just like any normal update. But, you should still check the release notes for any relevant noteable, or breaking change.

For the update itself use either the Web UI *Node -> Updates* panel or through the CLI with:

```
apt update
apt full-upgrade
```

Note: Always ensure you correctly setup the *package repositories* and only continue with the actual upgrade if *apt update* did not hit any error.

16.6 How can I upgrade Proxmox Backup Server to the next major release?

Major version upgrades, for example going from Proxmox Backup Server 2.4 to 3.1, are also supported. They must be carefully planned and tested and should **never** be started without having an off-site copy of the important backups, e.g., via remote sync or tape, ready.

Although the specific upgrade steps depend on your respective setup, we provide general instructions and advice of how a upgrade should be performed:

- Upgrade from Proxmox Backup Server 2 to 3
- Upgrade from Proxmox Backup Server 1 to 2

16.7 Can I copy or synchronize my datastore to another location?

Proxmox Backup Server allows you to copy or synchronize datastores to other locations, through the use of *Remotes* and *Sync Jobs*. *Remote* is the term given to a separate server, which has a datastore that can be synced to a local store. A *Sync Job* is the process which is used to pull the contents of a datastore from a *Remote* to a local datastore.

16.8 Can Proxmox Backup Server verify data integrity of a backup archive?

Proxmox Backup Server uses a built-in SHA-256 checksum algorithm, to ensure data integrity. Within each backup, a manifest file (index.json) is created, which contains a list of all the backup files, along with their sizes and checksums. This manifest file is used to verify the integrity of each backup.

16.9 When backing up to remote servers, do I have to trust the remote server?

Proxmox Backup Server transfers data via Transport Layer Security (TLS) and additionally supports client-side encryption. This means that data is transferred securely and can be encrypted before it reaches the server. Thus, in the event that an attacker gains access to the server or any point of the network, they will not be able to read the data.

Note: Encryption is not enabled by default. To set up encryption, see the *backup client encryption section*.

16.10 Is the backup incremental/deduplicated/full?

With Proxmox Backup Server, backups are sent incrementally to the server, and data is then deduplicated on the server. This minimizes both the storage consumed and the impact on the network. Each backup still references all data and such is a full backup. For details see the *Technical Overview*

APPENDIX

A

COMMAND SYNTAX

Note: Logging verbosity for the command line tools can be controlled with the PBS_LOG (for pxar: PXAR_LOG) environment variable. Possible values are *off*, *error*, *warn*, *info*, *debug* and *trace* with *info* being the default.

A.1 proxmox-backup-client

`proxmox-backup-client backup {<backupspec>} [OPTIONS]`

Create (host) backup.

<backupspec> [<string>] List of backup source specifications ([<label.ext>:<path>] ...) Can be specified more than once.

Optional parameters:

--all-file-systems <boolean> (default=false) Include all mounted subdirectories.

--backup-id <string> Backup ID.

--backup-time <integer> (1 - N) Backup time (Unix epoch.)

--backup-type vm|ct|host Backup type.

--burst <integer> (1000 - N) Size of the token bucket (for Token bucket filter) in bytes.

--chunk-size <integer> (64 - 4096) (default=4096) Chunk size in KB. Must be a power of 2.

--crypt-mode none|encrypt|sign-only (default=encrypt) Defines whether data is encrypted (using an AEAD cipher), only signed, or neither.

--dry-run <boolean> (default=false) Just show what backup would do, but do not upload anything.

--entries-max <integer> (default=1048576) Max number of entries to hold in memory.

--exclude <string> List of paths or patterns for matching files to exclude. Can be specified more than once.

--include-dev <string> Include mountpoints with same st_dev number (see `man fstat`) as specified files. Can be specified more than once.

--keyfd <integer> (0 - N) Pass an encryption key via an already opened file descriptor.

--keyfile <string> Path to encryption key. All data will be encrypted using this key.

--master-pubkey-fd <integer> (0 - N) Pass a master public key via an already opened file descriptor.

--master-pubkey-file <string> Path to master public key. The encryption key used for a backup will be encrypted using this key and appended to the backup.

--ns <string> Namespace.

--rate <integer> (100000 - N) Rate limit (for Token bucket filter) in bytes/second.

--repository <string> Repository URL.

--skip-lost-and-found <boolean> (default=false) Skip lost+found directory.

`proxmox-backup-client benchmark [OPTIONS]`

Run benchmark tests

Optional parameters:

--keyfile <string> Path to encryption key. All data will be encrypted using this key.

--output-format text|json|json-pretty Output format.

--repository <string> Repository URL.

`proxmox-backup-client catalog dump <snapshot> [OPTIONS]`

Dump catalog.

<snapshot> [<string>] Snapshot path.

Optional parameters:

--keyfd <integer> (0 - N) Pass an encryption key via an already opened file descriptor.

--keyfile <string> Path to encryption key.

--ns <string> Namespace.

--repository <string> Repository URL.

`proxmox-backup-client catalog shell <snapshot> <archive-name> [OPTIONS]`

Shell to interactively inspect and restore snapshots.

<snapshot> [<string>] Group/Snapshot path.

<archive-name> [<string>] Backup archive name.

Optional parameters:

--keyfd <integer> (0 - N) Pass an encryption key via an already opened file descriptor.

--keyfile <string> Path to encryption key.

--ns <string> Namespace.

--repository <string> Repository URL.

`proxmox-backup-client change-owner <group> <new-owner> [OPTIONS]`

Change owner of a backup group

<group> [<string>] Backup group.

<new-owner> [<string>] Authentication ID

Optional parameters:

--ns <string> Namespace.

--repository <string> Repository URL.

`proxmox-backup-client garbage-collect [OPTIONS]`

Start garbage collection for a specific repository.

Optional parameters:

`--output-format text|json|json-pretty` Output format.

`--repository <string>` Repository URL.

`proxmox-backup-client help [{<command>}] [OPTIONS]`

Get help about specified command (or sub-command).

`<command>` [`<string>`] Command. This may be a list in order to spefify nested sub-commands. Can be specified more than once.

Optional parameters:

`--verbose <boolean>` Verbose help.

`proxmox-backup-client key change-passphrase [<path>] [OPTIONS]`

Change the encryption key's password.

`<path>` [`<string>`] Key file. Without this the default key's password will be changed.

Optional parameters:

`--hint <string>` Password hint.

`--kdf none|scrypt|pbkdf2 (default=scrypt)` Key derivation function for password protected encryption keys.

`proxmox-backup-client key create [<path>] [OPTIONS]`

Create a new encryption key.

`<path>` [`<string>`] Output file. Without this the key will become the new default encryption key.

Optional parameters:

`--hint <string>` Password hint.

`--kdf none|scrypt|pbkdf2 (default=scrypt)` Key derivation function for password protected encryption keys.

`proxmox-backup-client key create-master-key`

Create an RSA public/private key pair used to put an encrypted version of the symmetric backup encryption key onto the backup server along with each backup.

`proxmox-backup-client key import-master-pubkey <path>`

Import an RSA public key used to put an encrypted version of the symmetric backup encryption key onto the backup server along with each backup.

The imported key will be used as default master key for future invocations by the same local user.

`<path>` [`<string>`] Path to the PEM formatted RSA public key.

`proxmox-backup-client key import-with-master-key [<path>] --encrypted-keyfile <string> --master-keyfile <string> [OPTIONS]`

Import an encrypted backup of an encryption key using a (private) master key.

<path> [<string>] Output file. Without this the key will become the new default encryption key.

--encrypted-keyfile <string> RSA-encrypted keyfile to import.

--master-keyfile <string> (Private) master key to use.

Optional parameters:

--hint <string> Password hint.

--kdf none|scrypt|pbkdf2 (default=scrypt) Key derivation function for password protected encryption keys.

`proxmox-backup-client key paperkey [<path>] [OPTIONS]`

Generate a printable, human readable text file containing the encryption key.

This also includes a scanable QR code for fast key restore.

<path> [<string>] Key file. Without this the default key's will be used.

Optional parameters:

--output-format text|html Paperkey output format

--subject <string> Include the specified subject as title text.

`proxmox-backup-client key show [<path>] [OPTIONS]`

Print the encryption key's metadata.

<path> [<string>] Key file. Without this the default key's metadata will be shown.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-client key show-master-pubkey [<path>] [OPTIONS]`

List information about master key

<path> [<string>] Path to the PEM formatted RSA public key. Default location will be used if not specified.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-client list [OPTIONS]`

List backup groups.

Optional parameters:

--ns <string> Namespace.

--output-format text|json|json-pretty Output format.

--repository <string> Repository URL.

`proxmox-backup-client login [OPTIONS]`

Try to login. If successful, store ticket.

Optional parameters:

`--repository <string>` Repository URL.

`proxmox-backup-client logout [OPTIONS]`

Logout (delete stored ticket).

Optional parameters:

`--repository <string>` Repository URL.

`proxmox-backup-client map <snapshot> <archive-name> [OPTIONS]`

Map a drive image from a VM backup to a local loopback device. Use 'unmap' to undo. WARNING: Only do this with *trusted* backups!

`<snapshot>` [`<string>`] Group/Snapshot path.

`<archive-name>` [`<string>`] Backup archive name.

Optional parameters:

`--keyfile <string>` Path to encryption key.

`--ns <string>` Namespace.

`--repository <string>` Repository URL.

`--verbose <boolean> (default=false)` Verbose output and stay in foreground.

`proxmox-backup-client mount <snapshot> <archive-name> <target> [OPTIONS]`

Mount pxar archive.

`<snapshot>` [`<string>`] Group/Snapshot path.

`<archive-name>` [`<string>`] Backup archive name.

`<target>` [`<string>`] Target directory path.

Optional parameters:

`--keyfile <string>` Path to encryption key.

`--ns <string>` Namespace.

`--repository <string>` Repository URL.

`--verbose <boolean> (default=false)` Verbose output and stay in foreground.

`proxmox-backup-client namespace create [<ns>] [OPTIONS]`

Create a new namespace.

`<ns>` [`<string>`] Namespace.

Optional parameters:

`--repository <string>` Repository URL.

`proxmox-backup-client namespace delete [<ns>] [OPTIONS]`

Delete an existing namespace.

<ns> [<string>] Namespace.

Optional parameters:

--repository <string> Repository URL.

`proxmox-backup-client namespace list [<ns>] [OPTIONS]`

List namespaces in a repository.

<ns> [<string>] Namespace.

Optional parameters:

--max-depth <integer> (0 - N) maximum recursion depth

--output-format text|json|json-pretty Output format.

--repository <string> Repository URL.

`proxmox-backup-client prune <group> [OPTIONS]`

Prune a backup repository.

<group> [<string>] Backup group

Optional parameters:

--dry-run <boolean> Just show what prune would do, but do not delete anything.

--output-format text|json|json-pretty Output format.

--quiet <boolean> (default=false) Minimal output - only show removals.

--repository <string> Repository URL.

--max-depth <integer> (0 - 7) How many levels of namespaces should be operated on (0 == no recursion, empty == automatic full recursion, namespace depths reduce maximum allowed value)

--ns <string> Namespace.

--keep-daily <integer> (1 - N) Number of daily backups to keep.

--keep-hourly <integer> (1 - N) Number of hourly backups to keep.

--keep-last <integer> (1 - N) Number of backups to keep.

--keep-monthly <integer> (1 - N) Number of monthly backups to keep.

--keep-weekly <integer> (1 - N) Number of weekly backups to keep.

--keep-yearly <integer> (1 - N) Number of yearly backups to keep.

`proxmox-backup-client restore <snapshot> <archive-name> <target> [OPTIONS]`

Restore backup repository.

<snapshot> [<string>] Group/Snapshot path.

<archive-name> [<string>] Backup archive name.

<target> [<string>] Target directory path. Use '-' to write to standard output.

We do not extract '.pxar' archives when writing to standard output.

Optional parameters:

--allow-existing-dirs <boolean> (default=false) Do not fail if directories already exists.

--burst <integer> (1000 - N) Size of the token bucket (for Token bucket filter) in bytes.

--crypt-mode none|encrypt|sign-only (default=encrypt) Defines whether data is encrypted (using an AEAD cipher), only signed, or neither.

--ignore-acls <boolean> (default=false) ignore acl settings

--ignore-ownership <boolean> (default=false) ignore owner settings (no chown)

--ignore-permissions <boolean> (default=false) ignore permission settings (no chmod)

--ignore-xattrs <boolean> (default=false) ignore xattr settings

--keyfd <integer> (0 - N) Pass an encryption key via an already opened file descriptor.

--keyfile <string> Path to encryption key. All data will be encrypted using this key.

--ns <string> Namespace.

--overwrite <boolean> (default=false) overwrite already existing files

--rate <integer> (100000 - N) Rate limit (for Token bucket filter) in bytes/second.

--repository <string> Repository URL.

`proxmox-backup-client snapshot files <snapshot> [OPTIONS]`

List snapshot files.

<snapshot> [<string>] Snapshot path.

Optional parameters:

--ns <string> Namespace.

--output-format text|json|json-pretty Output format.

--repository <string> Repository URL.

`proxmox-backup-client snapshot forget <snapshot> [OPTIONS]`

Forget (remove) backup snapshots.

<snapshot> [<string>] Snapshot path.

Optional parameters:

--ns <string> Namespace.

--repository <string> Repository URL.

`proxmox-backup-client snapshot list [<group>] [OPTIONS]`

List backup snapshots.

<group> [<string>] Backup group.

Optional parameters:

--ns <string> Namespace.

--output-format text|json|json-pretty Output format.

--repository <string> Repository URL.

`proxmox-backup-client snapshot notes show <snapshot> [OPTIONS]`

Show notes

<snapshot> [<string>] Snapshot path.

Optional parameters:

--ns <string> Namespace.

--output-format text|json|json-pretty Output format.

--repository <string> Repository URL.

`proxmox-backup-client snapshot notes update <snapshot> <notes> [OPTIONS]`

Update Notes

<snapshot> [<string>] Snapshot path.

<notes> [<string>] The Notes.

Optional parameters:

--ns <string> Namespace.

--repository <string> Repository URL.

`proxmox-backup-client snapshot protected show <snapshot> [OPTIONS]`

Show protection status of the specified snapshot

<snapshot> [<string>] Snapshot path.

Optional parameters:

--ns <string> Namespace.

--output-format text|json|json-pretty Output format.

--repository <string> Repository URL.

`proxmox-backup-client snapshot protected update <snapshot> <protected> [OPTIONS]`

Update Protection Status of a snapshot

<snapshot> [<string>] Snapshot path.

<protected> [<boolean>] The protection status.

Optional parameters:

--ns <string> Namespace.

--repository <string> Repository URL.

`proxmox-backup-client snapshot upload-log <snapshot> <logfile> [OPTIONS]`

Upload backup log file.

<snapshot> [<string>] Group/Snapshot path.

<logfile> [<string>] The path to the log file you want to upload.

Optional parameters:

--crypt-mode none|encrypt|sign-only (default=encrypt) Defines whether data is encrypted (using an AEAD cipher), only signed, or neither.

--keyfd <integer> (0 - N) Pass an encryption key via an already opened file descriptor.

--keyfile <string> Path to encryption key. All data will be encrypted using this key.

--ns <string> Namespace.

--repository <string> Repository URL.

`proxmox-backup-client status [OPTIONS]`

Get repository status.

Optional parameters:

--output-format text|json|json-pretty Output format.

--repository <string> Repository URL.

`proxmox-backup-client task list [OPTIONS]`

List running server tasks for this repo user

Optional parameters:

--all <boolean> Also list stopped tasks.

--limit <integer> (1 - 1000) (default=50) The maximal number of tasks to list.

--output-format text|json|json-pretty Output format.

--repository <string> Repository URL.

`proxmox-backup-client task log <upid> [OPTIONS]`

Display the task log.

<upid> [<string>] Unique Process/Task Identifier

Optional parameters:

--repository <string> Repository URL.

`proxmox-backup-client task stop <upid> [OPTIONS]`

Try to stop a specific task.

<upid> [<string>] Unique Process/Task Identifier

Optional parameters:

--repository <string> Repository URL.

`proxmox-backup-client unmap [<name>]`

Unmap a loop device mapped with 'map' and release all resources.

<name> [<string>] Archive name, path to loopdev (/dev/loopX) or loop device number. Omit to list all current mappings and force cleaning up leftover instances.

`proxmox-backup-client version [OPTIONS]`

Show client and optional server version

Optional parameters:

--output-format text|json|json-pretty Output format.

--repository <string> Repository URL.

A.1.1 Catalog Shell Commands

The following commands are available in an interactive restore shell:

```
proxmox-backup-client shell <snapshot> <name.pxar>
```

`cd [<path>]`

Change the current working directory to the new directory

<path> [<string>] target path.

`clear-selected`

Clear the list of files selected for restore.

`deselect <path>`

Deselect an entry for restore.

This will return an error if the entry was not found in the list of entries selected for restore.

<path> [<string>] path to entry to remove from list.

`exit`

Exit the shell

`find <pattern> [OPTIONS]`

Find entries in the catalog matching the given match pattern.

<pattern> [<string>] Match pattern for matching files in the catalog.

Optional parameters:

--select <boolean> (default=false) Add matching filenames to list for restore.

`help [{<command>}] [OPTIONS]`

Get help about specified command (or sub-command).

<command> [<string>] Command. This may be a list in order to spefify nested sub-commands. Can be specified more than once.

Optional parameters:

--verbose <boolean> Verbose help.

`list-selected [OPTIONS]`

List entries currently selected for restore.

Optional parameters:

--patterns <boolean> (default=false) List match patterns instead of the matching files.

`ls [<path>]`

List the content of working directory or given path.

<path> [<string>] target path.

`pwd`

List the current working directory.

`restore <target> [OPTIONS]`

Restore the sub-archive given by the current working directory to target.

By further providing a pattern, the restore can be limited to a narrower subset of this sub-archive. If pattern is not present or empty, the full archive is restored to target.

<target> [<string>] target path for restore on local filesystem.

Optional parameters:

--pattern <string> match pattern to limit files for restore.

`restore-selected <target>`

Restore the selected entries to the given target path.

Target must not exist on the clients filesystem.

<target> [<string>] target path for restore on local filesystem.

`select <path>`

Select an entry for restore.

This will return an error if the entry is already present in the list or if an invalid path was provided.

<path> [<string>] target path.

`stat <path>`

Read the metadata for a given directory entry.

This is expensive because the data has to be read from the pxar archive, which means reading over the network.

<path> [<string>] target path.

A.2 proxmox-backup-manager

`proxmox-backup-manager acl list [OPTIONS]`

Access Control list.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager acl update <path> <role> [OPTIONS]`

Update Access Control List (ACLs).

<path> [<string>] Access control path.

<role> [<role>] Enum representing roles via their [PRIVILEGES] combination.

> Since privileges are implemented as bitflags, each unique combination of privileges maps to a single, unique *u64* value that is used in this enum definition.

Optional parameters:

--auth-id <string> Authentication ID

--delete <boolean> Remove permissions (instead of adding it).

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--group <string> Group ID

--propagate <boolean> (default=true) Allow to propagate (inherit) permissions.

`proxmox-backup-manager acme account deactivate <name> [OPTIONS]`

Deactivate an ACME account.

<name> [<string>] ACME account name.

Optional parameters:

--force <boolean> (default=false) Delete account data even if the server refuses to deactivate the account.

`proxmox-backup-manager acme account info <name> [OPTIONS]`

Show acme account information.

<name> [<string>] ACME account name.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager acme account list [OPTIONS]`

List acme accounts.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager acme account register <name> <contact> [OPTIONS]`

Register an ACME account.

<name> [<string>] ACME account name.

<contact> [<string>] List of email addresses.

Optional parameters:

--directory <string> The ACME Directory.

`proxmox-backup-manager acme account update <name> [OPTIONS]`

Update an ACME account.

<name> [<string>] ACME account name.

Optional parameters:

--contact <string> List of email addresses.

`proxmox-backup-manager acme cert order [OPTIONS]`

Order a new ACME certificate.

Optional parameters:

--force <boolean> (default=false) Force renewal even if the certificate does not expire soon.

`proxmox-backup-manager acme cert revoke`

Order a new ACME certificate.

`proxmox-backup-manager acme plugin add <type> <id> --data <string> --api <string> [OPTIONS]`

Show acme account information.

<type> [<string>] The ACME challenge plugin type.

<id> [<string>] ACME Challenge Plugin ID.

--data <string> File containing the plugin data.

--api <string> DNS API Plugin Id.

Optional parameters:

--disable <boolean> (default=false) Flag to disable the config.

--validation-delay <integer> (0 - 172800) (default=30) Extra delay in seconds to wait before requesting validation.

Allows to cope with long TTL of DNS records.

`proxmox-backup-manager acme plugin config <id> [OPTIONS]`

Show acme account information.

<id> [<string>] Plugin ID

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager acme plugin list [OPTIONS]`

List acme plugins.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager acme plugin remove <id>`

Delete an ACME plugin configuration.

<id> [<string>] ACME Challenge Plugin ID.

`proxmox-backup-manager acme plugin set <id> [OPTIONS]`

Update an ACME plugin configuration.

<id> [<string>] ACME Challenge Plugin ID.

Optional parameters:

--data <string> DNS plugin data (base64 encoded with padding).

--delete disable|validation-delay List of properties to delete. Can be specified more than once.

--digest <string> Digest to protect against concurrent updates

--api <string> DNS API Plugin Id.

--disable <boolean> (default=false) Flag to disable the config.

--validation-delay <integer> (0 - 172800) (default=30) Extra delay in seconds to wait before requesting validation.

Allows to cope with long TTL of DNS records.

`proxmox-backup-manager cert info`

Display node certificate information.

`proxmox-backup-manager cert update [OPTIONS]`

Update node certificates and generate all needed files/directories.

Optional parameters:

--force <boolean> Force generation of new SSL certifate.

`proxmox-backup-manager datastore create <name> <path> [OPTIONS]`

Create new datastore config.

<name> [<string>] Datastore name.

<path> [<string>] Directory name

Optional parameters:

--output-format text|json|json-pretty Output format.

--comment <string> Comment (single line).

--gc-schedule <calendar-event> Run garbage collection job at specified schedule.

--maintenance-mode [type=<enum> [,message=<string>]] Maintenance mode, type is either 'offline' or 'read-only', message should be enclosed in "

--notify [[gc=<enum>] [,prune=<enum>] [,sync=<enum>] [,verify=<enum>]] Datastore notification setting

--notify-user <string> User ID

--prune-schedule <calendar-event> Run prune job at specified schedule.

--tuning [[chunk-order=<enum>] [,sync-level=<enum>]] Datastore tuning options

--verify-new <boolean> If enabled, all new backups will be verified right after completion.

--keep-daily <integer> (1 - N) Number of daily backups to keep.

--keep-hourly <integer> (1 - N) Number of hourly backups to keep.

--keep-last <integer> (1 - N) Number of backups to keep.

--keep-monthly <integer> (1 - N) Number of monthly backups to keep.

--keep-weekly <integer> (1 - N) Number of weekly backups to keep.

--keep-yearly <integer> (1 - N) Number of yearly backups to keep.

`proxmox-backup-manager datastore list [OPTIONS]`

Datastore list.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager datastore remove <name> [OPTIONS]`

Remove a datastore configuration.

<name> [<string>] Datastore name.

Optional parameters:

--destroy-data <boolean> (default=false) Delete the datastore's underlying contents

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--keep-job-configs <boolean> (default=false) If enabled, the job configurations related to this datastore will be kept.

`proxmox-backup-manager datastore show <name> [OPTIONS]`

Show datastore configuration

<name> [<string>] Datastore name.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager datastore update <name> [OPTIONS]`

Update datastore config.

<name> [<string>] Datastore name.

Optional parameters:

--delete comment|gc-schedule|prune-schedule|keep-last|keep-hourly|keep-daily|keep-weekly
List of properties to delete. Can be specified more than once.

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--comment <string> Comment (single line).

--gc-schedule <calendar-event> Run garbage collection job at specified schedule.

--maintenance-mode [type=<enum> [,message=<string>]] Maintenance mode, type is either 'offline' or 'read-only', message should be enclosed in "

--notify [[gc=<enum>] [,prune=<enum>] [,sync=<enum>] [,verify=<enum>]]
Datastore notification setting

--notify-user <string> User ID

--prune-schedule <calendar-event> Run prune job at specified schedule.

--tuning [[chunk-order=<enum>] [,sync-level=<enum>]] Datastore tuning options

--verify-new <boolean> If enabled, all new backups will be verified right after completion.

--keep-daily <integer> (1 - N) Number of daily backups to keep.

--keep-hourly <integer> (1 - N) Number of hourly backups to keep.

--keep-last <integer> (1 - N) Number of backups to keep.

--keep-monthly <integer> (1 - N) Number of monthly backups to keep.

--keep-weekly <integer> (1 - N) Number of weekly backups to keep.

--keep-yearly <integer> (1 - N) Number of yearly backups to keep.

`proxmox-backup-manager disk fs create <name> --disk <string> [OPTIONS]`

Create a Filesystem on an unused disk. Will be mounted under '/mnt/datastore/<name>'.

<name> [<string>] Datastore name.

--disk <string> Block device name (/sys/block/<name>).

Optional parameters:

--add-datastore <boolean> Configure a datastore using the directory.

`--filesystem ext4|xfs`

`proxmox-backup-manager disk fs list [OPTIONS]`

List systemd datastore mount units.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager disk initialize <disk> [OPTIONS]`

Initialize empty Disk with GPT

<disk> [<string>] Block device name (/sys/block/<name>).

Optional parameters:

--uuid <string> UUID for the GPT table.

`proxmox-backup-manager disk list [OPTIONS]`

Local disk list.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager disk smart-attributes <disk> [OPTIONS]`

Show SMART attributes.

<disk> [<string>] Block device name (/sys/block/<name>).

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager disk zpool create <name> --devices [<string>, ...] --raidlevel single|mirror|raid10|raidz|raidz2|raidz3 [OPTIONS]`

create a zfs pool

<name> [<string>] Datastore name.

--devices [<string>, ...] A list of disk names, comma separated.

--raidlevel single|mirror|raid10|raidz|raidz2|raidz3 The ZFS RAID level to use.

Optional parameters:

--add-datastore <boolean> Configure a datastore using the zpool.

--ashift <integer> (9 - 16) (default=12) Pool sector size exponent.

--compression gzip|lz4|lzjb|zle|zstd|on|off (default=On) The ZFS compression algorithm to use.

`proxmox-backup-manager disk zpool list [OPTIONS]`

Local zfs pools.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager dns get [OPTIONS]`

Read DNS settings

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager dns set [OPTIONS]`

Update DNS settings.

Optional parameters:

--delete dns1|dns2|dns3 List of properties to delete. Can be specified more than once.

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--dns1 <string> First name server IP address.

--dns2 <string> Second name server IP address.

--dns3 <string> Third name server IP address.

--search <string> Search domain for host-name lookup.

`proxmox-backup-manager garbage-collection start <store> [OPTIONS]`

Start garbage collection for a specific datastore.

<store> [<string>] Datastore name.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager garbage-collection status <store> [OPTIONS]`

Show garbage collection status for a specific datastore.

<store> [<string>] Datastore name.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager help [{<command>}] [OPTIONS]`

Get help about specified command (or sub-command).

<command> [<string>] Command. This may be a list in order to spefify nested sub-commands. Can be specified more than once.

Optional parameters:

--verbose <boolean> Verbose help.

`proxmox-backup-manager ldap create <realm> --base-dn <string> --server1 <string> --user-attr <string> [OPTIONS]`

Create a new LDAP realm

<realm> [<string>] Realm name.

--base-dn <string> LDAP Domain

--server1 <string> LDAP server address

--user-attr <string> Username attribute. Used to map a `userid` to LDAP to an LDAP `dn`.

Optional parameters:

--password <string> LDAP bind password

--bind-dn <string> LDAP Domain

--capath <string> CA certificate to use for the server. The path can point to either a file, or a directory. If it points to a file, the PEM-formatted X.509 certificate stored at the path will be added as a trusted certificate. If the path points to a directory, the directory replaces the system's default certificate store at */etc/ssl/certs* - Every file in the directory will be loaded as a trusted certificate.

--comment <string> Comment (single line).

--filter <string> Custom LDAP search filter for user sync

--mode ldap|ldap+starttls|ldaps (default=ldap) LDAP connection type

--port <integer> (0 - 65535) Port

--server2 <string> Fallback LDAP server address

--sync-attributes [[email=<string>] [,firstname=<string>] [,lastname=<string>]] Comma-separated list of key=value pairs for specifying which LDAP attributes map to which PBS user field. For example, to map the LDAP attribute `mail` to PBS's `email`, write `email=mail`.

--sync-defaults-options [[enable-new=<1|0>] [,remove-vanished=<string>]] sync defaults options

--user-classes [<string>, ...] (default=inetorgperson,posixaccount,person,user) Comma-separated list of allowed objectClass values for user synchronization. For instance, if `user-classes` is set to `person,user`, then user synchronization will consider all LDAP entities where `objectClass: person` *or* `objectClass: user`.

--verify <boolean> (default=false) Verify server certificate

`proxmox-backup-manager ldap delete <realm> [OPTIONS]`

Remove an LDAP realm configuration

<realm> [<string>] Realm name.

Optional parameters:

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

`proxmox-backup-manager ldap list [OPTIONS]`

List configured LDAP realms

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager ldap show <realm> [OPTIONS]`

Show LDAP realm configuration

<realm> [<string>] Realm name.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager ldap sync <realm> [OPTIONS]`

Sync a given LDAP realm

<realm> [<string>] Authentication domain ID

Optional parameters:

--dry-run <boolean> (default=false) If set, do not create/delete anything

--enable-new <boolean> Enable newly synced users immediately

--remove-vanished [acl|entry|properties, ...] A semicolon-seperated list of things to remove when they or the user vanishes during user synchronization. The following values are possible: `entry` removes the user when not returned from the sync; `properties` removes any properties on existing user that do not appear in the source. `acl` removes ACLs when the user is not returned from the sync.

`proxmox-backup-manager ldap update <realm> [OPTIONS]`

Update an LDAP realm configuration

<realm> [<string>] Realm name.

Optional parameters:

--delete server2|port|comment|verify|mode|bind-dn|password|filter|sync-defaults-options| List of properties to delete. Can be specified more than once.

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--password <string> LDAP bind password

--base-dn <string> LDAP Domain

--bind-dn <string> LDAP Domain

--capath <string> CA certificate to use for the server. The path can point to either a file, or a directory. If it points to a file, the PEM-formatted X.509 certificate stored at the path will be added as a trusted certificate. If the path points to a directory, the directory replaces the system's default certificate store at */etc/ssl/certs* - Every file in the directory will be loaded as a trusted certificate.

--comment <string> Comment (single line).

--filter <string> Custom LDAP search filter for user sync

--mode ldap|ldap+starttls|ldaps (default=ldap) LDAP connection type

--port <integer> (0 - 65535) Port

--server1 <string> LDAP server address

--server2 <string> Fallback LDAP server address

--sync-attributes [[email=<string>] [,firstname=<string>] [,lastname=<string>]] Comma-separated list of key=value pairs for specifying which LDAP attributes map to which PBS user field. For example, to map the LDAP attribute `mail` to PBS's `email`, write `email=mail`.

--sync-defaults-options [[enable-new=<1|0>] [,remove-vanished=<string>]] sync defaults options

--user-attr <string> Username attribute. Used to map a `userid` to LDAP to an LDAP `dn`.

--user-classes [<string>, ...] (default=inetorgperson,posixaccount,person,user) Comma-separated list of allowed objectClass values for user synchronization. For instance, if `user-classes` is set to `person,user`, then user synchronization will consider all LDAP entities where `objectClass: person` *or* `objectClass: user`.

--verify <boolean> (default=false) Verify server certificate

`proxmox-backup-manager network changes`

Show pending configuration changes (diff)

`proxmox-backup-manager network create <iface> [OPTIONS]`

Create network interface configuration.

<iface> [<string>] Network interface name.

Optional parameters:

--autostart <boolean> Autostart interface.

--bond-primary <string> Network interface name.

--bond_mode balance-rr|active-backup|balance-xor|broadcast|802.3ad|balance-tlb|balance-a Linux Bond Mode

--bond_xmit_hash_policy layer2|layer2+3|layer3+4 Bond Transmit Hash Policy for LACP (802.3ad)

--bridge_ports [<string>, ...] A list of network devices, comma separated.

--bridge_vlan_aware <boolean> Enable bridge vlan support.

--cidr <string> IPv4 address with netmask (CIDR notation).

--cidr6 <string> IPv6 address with netmask (CIDR notation).

--comments <string> Comments (inet, may span multiple lines)

--comments6 <string> Comments (inet5, may span multiple lines)

--gateway <string> IPv4 address.

--gateway6 <string> IPv6 address.

--method manual|static|dhcp|loopback Interface configuration method

--method6 manual|static|dhcp|loopback Interface configuration method

--mtu <integer> (46 - 65535) (default=1500) Maximum Transmission Unit.

--slaves [<string>, ...] A list of network devices, comma separated.

--type loopback|eth|bridge|bond|vlan|alias|unknown Network interface type

---

`proxmox-backup-manager network list [OPTIONS]`

Network device list.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager network reload`

Reload network configuration (requires ifupdown2).

`proxmox-backup-manager network remove <iface> [OPTIONS]`

Remove network interface configuration.

<iface> [<string>] Network interface name.

Optional parameters:

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

`proxmox-backup-manager network revert`

Revert network configuration (rm /etc/network/interfaces.new).

`proxmox-backup-manager network update <iface> [OPTIONS]`

Update network interface config.

<iface> [<string>] Network interface name.

Optional parameters:

--autostart <boolean> Autostart interface.

--bond-primary <string> Network interface name.

--bond_mode balance-rr|active-backup|balance-xor|broadcast|802.3ad|balance-tlb|balance-a Linux Bond Mode

--bond_xmit_hash_policy layer2|layer2+3|layer3+4 Bond Transmit Hash Policy for LACP (802.3ad)

--bridge_ports [<string>, ...] A list of network devices, comma separated.

--bridge_vlan_aware <boolean> Enable bridge vlan support.

--cidr <string> IPv4 address with netmask (CIDR notation).

--cidr6 <string> IPv6 address with netmask (CIDR notation).

--comments <string> Comments (inet, may span multiple lines)

--comments6 <string> Comments (inet5, may span multiple lines)

--delete cidr|cidr6|gateway|gateway6|method|method6|comments|comments6|mtu|autostart|bri
List of properties to delete. Can be specified more than once.

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--gateway <string> IPv4 address.

--gateway6 <string> IPv6 address.

--method manual|static|dhcp|loopback Interface configuration method

--method6 manual|static|dhcp|loopback Interface configuration method

--mtu <integer> (46 - 65535) (default=1500) Maximum Transmission Unit.

--slaves [<string>, ...] A list of network devices, comma separated.

--type loopback|eth|bridge|bond|vlan|alias|unknown Network interface type

`proxmox-backup-manager node show [OPTIONS]`

Show node configuration

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager node update [OPTIONS]`

Update the node configuration

Optional parameters:

--delete acme|acmedomain0|acmedomain1|acmedomain2|acmedomain3|acmedomain4|http-proxy|ema
List of properties to delete. Can be specified more than once.

--digest <string> Digest to protect against concurrent updates

--acme [account=<string>] The acme account to use on this node.

--acmedomain0 [domain=<string> [,alias=<string>] [,plugin=<string>]] ACME domain configuration string

--acmedomain1 [domain=<string> [,alias=<string>] [,plugin=<string>]] ACME domain configuration string

--acmedomain2 [domain=<string> [,alias=<string>] [,plugin=<string>]] ACME domain configuration string

--acmedomain3 [domain=<string> [,alias=<string>] [,plugin=<string>]] ACME domain configuration string

--acmedomain4 [domain=<string> [,alias=<string>] [,plugin=<string>]] ACME domain configuration string

--ciphers-tls-1.2 <string> OpenSSL cipher list used by the proxy for TLS <= 1.2

--ciphers-tls-1.3 <string> OpenSSL ciphersuites list used by the proxy for TLS 1.3

--default-lang ar|ca|da|de|en|es|eu|fa|fr|gl|he|hu|it|ja|kr|nb|nl|nn|pl|pt_BR|ru|sl|sv|t
All available languages in Proxmox. Taken from proxmox-i18n repository. pt_BR, zh_CN, and zh_TW use the same case in the translation files.

--description <string> Comment (multiple lines).

--email-from <string> E-Mail Address.

--http-proxy [http://]<host>[:port] HTTP proxy configuration [http://]<host>[:port]

--task-log-max-days <integer> (0 - N) Maximum days to keep Task logs

```
proxmox-backup-manager openid create <realm> --client-id <string>
--issuer-url <string> [OPTIONS]
```

Create a new OpenId realm

<realm> [<string>] Realm name.

--client-id <string> OpenID Client ID

--issuer-url <string> OpenID Issuer Url

Optional parameters:

--acr-values [<string>, ...] OpenID ACR List

--autocreate <boolean> (default=false) Automatically create users if they do not exist.

--client-key <string> OpenID Client Key

--comment <string> Comment (single line).

--prompt <string> OpenID Prompt

--scopes [<string>, ...] (default=email profile) OpenID Scope List

--username-claim <string> Use the value of this attribute/claim as unique user name. It is up to the identity provider to guarantee the uniqueness. The OpenID specification only guarantees that Subject ('sub') is unique. Also make sure that the user is not allowed to change that attribute by himself!

```
proxmox-backup-manager openid delete <realm> [OPTIONS]
```

Remove a OpenID realm configuration

<realm> [<string>] Realm name.

Optional parameters:

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

```
proxmox-backup-manager openid list [OPTIONS]
```

List configured OpenId realms

Optional parameters:

--output-format text|json|json-pretty Output format.

```
proxmox-backup-manager openid show <realm> [OPTIONS]
```

Show OpenID realm configuration

<realm> [<string>] Realm name.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager openid update <realm> [OPTIONS]`

Update an OpenID realm configuration

<realm> [<string>] Realm name.

Optional parameters:

--delete client-key|comment|autocreate|scopes|prompt|acr-values List of properties to delete. Can be specified more than once.

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--acr-values [<string>, ...] OpenID ACR List

--autocreate <boolean> (default=false) Automatically create users if they do not exist.

--client-id <string> OpenID Client ID

--client-key <string> OpenID Client Key

--comment <string> Comment (single line).

--issuer-url <string> OpenID Issuer Url

--prompt <string> OpenID Prompt

--scopes [<string>, ...] (default=email profile) OpenID Scope List

`proxmox-backup-manager prune-job create <id> --store <string> [OPTIONS]`

Create a new prune job.

<id> [<string>] Job ID.

--store <string> Datastore name.

Optional parameters:

--comment <string> Comment (single line).

--disable <boolean> (default=false) Disable this job.

--schedule <calendar-event> Run prune job at specified schedule.

--max-depth <integer> (0 - 7) How many levels of namespaces should be operated on (0 == no recursion, empty == automatic full recursion, namespace depths reduce maximum allowed value)

--ns <string> Namespace.

--keep-daily <integer> (1 - N) Number of daily backups to keep.

--keep-hourly <integer> (1 - N) Number of hourly backups to keep.

--keep-last <integer> (1 - N) Number of backups to keep.

--keep-monthly <integer> (1 - N) Number of monthly backups to keep.

--keep-weekly <integer> (1 - N) Number of weekly backups to keep.

--keep-yearly <integer> (1 - N) Number of yearly backups to keep.

`proxmox-backup-manager prune-job list [OPTIONS]`

List all prune jobs

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager prune-job remove <id> [OPTIONS]`

Remove a prune job configuration

<id> [<string>] Job ID.

Optional parameters:

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

`proxmox-backup-manager prune-job run <id> [OPTIONS]`

Run the specified prune job

<id> [<string>] Job ID.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager prune-job show <id> [OPTIONS]`

Show prune job configuration

<id> [<string>] Job ID.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager prune-job update <id> [OPTIONS]`

Update prune job config.

<id> [<string>] Job ID.

Optional parameters:

--delete comment|disable|ns|max-depth|keep-last|keep-hourly|keep-daily|keep-weekly|keep-
List of properties to delete. Can be specified more than once.

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--comment <string> Comment (single line).

--disable <boolean> (default=false) Disable this job.

--schedule <calendar-event> Run prune job at specified schedule.

--store <string> Datastore name.

--max-depth <integer> (0 - 7) How many levels of namespaces should be operated on (0 == no recursion, empty == automatic full recursion, namespace depths reduce maximum allowed value)

--ns <string> Namespace.

--keep-daily <integer> (1 - N) Number of daily backups to keep.

--keep-hourly <integer> (1 - N) Number of hourly backups to keep.

--keep-last <integer> (1 - N) Number of backups to keep.

--keep-monthly <integer> (1 - N) Number of monthly backups to keep.

--keep-weekly <integer> (1 - N) Number of weekly backups to keep.

--keep-yearly <integer> (1 - N) Number of yearly backups to keep.

`proxmox-backup-manager pull <remote> <remote-store> <store> [OPTIONS]`

Sync datastore from another repository

<remote> [<string>] Remote ID.

<remote-store> [<string>] Datastore name.

<store> [<string>] Datastore name.

Optional parameters:

--group-filter <type:<vm|ct|host>|group:GROUP|regex:RE> List of group filters. Can be specified more than once.

--max-depth <integer> (0 - 7) (default=7) How many levels of namespaces should be operated on (0 == no recursion)

--ns <string> Namespace.

--output-format text|json|json-pretty Output format.

--remote-ns <string> Namespace.

--remove-vanished <boolean> (default=false) Delete vanished backups. This remove the local copy if the remote backup was deleted.

--transfer-last <integer> (1 - N) Limit transfer to last N snapshots (per group), skipping others

--burst-in <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

--burst-out <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

--rate-in <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

--rate-out <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

`proxmox-backup-manager remote create <name> --password <string> --auth-id <string> --host <string> [OPTIONS]`

Create new remote.

<name> [<string>] Remote ID.

--password <string> Password or auth token for remote host.

--auth-id <string> Authentication ID

--host <string> DNS name or IP address.

Optional parameters:

--comment <string> Comment (single line).

--fingerprint <string> X509 certificate fingerprint (sha256).

--port <integer> The (optional) port

`proxmox-backup-manager remote list [OPTIONS]`

List configured remotes.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager remote remove <name> [OPTIONS]`

Remove a remote from the configuration file.

<name> [<string>] Remote ID.

Optional parameters:

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

`proxmox-backup-manager remote show <name> [OPTIONS]`

Show remote configuration

<name> [<string>] Remote ID.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager remote update <name> [OPTIONS]`

Update remote configuration.

<name> [<string>] Remote ID.

Optional parameters:

--delete comment|fingerprint|port List of properties to delete. Can be specified more than once.

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--password <string> Password or auth token for remote host.

--auth-id <string> Authentication ID

--comment <string> Comment (single line).

--fingerprint <string> X509 certificate fingerprint (sha256).

--host <string> DNS name or IP address.

--port <integer> The (optional) port

`proxmox-backup-manager report`

System report

`proxmox-backup-manager subscription get [OPTIONS]`

Read subscription info.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager subscription remove`

Delete subscription info.

`proxmox-backup-manager subscription set <key>`

Set a subscription key and check it.

<key> [<string>] Proxmox Backup Server subscription key.

`proxmox-backup-manager subscription set-offline-key <data>`

(Internal use only!) Set a signed subscription info blob as offline key

<data> [<string>] base64-encoded signed subscription info

`proxmox-backup-manager subscription update [OPTIONS]`

Check and update subscription status.

Optional parameters:

--force <boolean> (default=false) Always connect to server, even if information in cache is up to date.

`proxmox-backup-manager sync-job create <id> --remote <string> --remote-store <string> --store <string> [OPTIONS]`

Create a new sync job.

<id> [<string>] Job ID.

--remote <string> Remote ID.

--remote-store <string> Datastore name.

--store <string> Datastore name.

Optional parameters:

--comment <string> Comment (single line).

--group-filter <type:<vm|ct|host>|group:GROUP|regex:RE> List of group filters. Can be specified more than once.

--max-depth <integer> (0 - 7) How many levels of namespaces should be operated on (0 == no recursion, empty == automatic full recursion, namespace depths reduce maximum allowed value)

--ns <string> Namespace.

--owner <string> Authentication ID

--remote-ns <string> Namespace.

--remove-vanished <boolean> (default=false) Delete vanished backups. This remove the local copy if the remote backup was deleted.

--schedule <calendar-event> Run sync job at specified schedule.

--transfer-last <integer> (1 - N) Limit transfer to last N snapshots (per group), skipping others

--burst-in <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

--burst-out <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

--rate-in <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

--rate-out <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

`proxmox-backup-manager sync-job list [OPTIONS]`

Sync job list.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager sync-job remove <id> [OPTIONS]`

Remove a sync job configuration

<id> [<string>] Job ID.

Optional parameters:

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

`proxmox-backup-manager sync-job run <id> [OPTIONS]`

Run the specified sync job

<id> [<string>] Job ID.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager sync-job show <id> [OPTIONS]`

Show sync job configuration

<id> [<string>] Job ID.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager sync-job update <id> [OPTIONS]`

Update sync job config.

<id> [<string>] Job ID.

Optional parameters:

--delete owner|comment|schedule|remove-vanished|group-filter|rate-in|burst-in|rate-out|b List of properties to delete. Can be specified more than once.

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--comment <string> Comment (single line).

--group-filter <type:<vm|ct|host>|group:GROUP|regex:RE> List of group filters. Can be specified more than once.

--max-depth <integer> (0 - 7) How many levels of namespaces should be operated on (0 == no recursion, empty == automatic full recursion, namespace depths reduce maximum allowed value)

--ns <string> Namespace.

--owner <string> Authentication ID

--remote <string> Remote ID.

--remote-ns <string> Namespace.

--remote-store <string> Datastore name.

--remove-vanished <boolean> (default=false) Delete vanished backups. This remove the local copy if the remote backup was deleted.

--schedule <calendar-event> Run sync job at specified schedule.

--store <string> Datastore name.

--transfer-last <integer> (1 - N) Limit transfer to last N snapshots (per group), skipping others

--burst-in <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

--burst-out <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

--rate-in <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

--rate-out <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

`proxmox-backup-manager task list [OPTIONS]`

List running server tasks.

Optional parameters:

--all <boolean> Also list stopped tasks.

--limit <integer> (1 - 1000) (default=50) The maximal number of tasks to list.

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager task log <upid>`

Display the task log.

<upid> [<string>] Unique Process/Task Identifier

`proxmox-backup-manager task stop <upid>`

Try to stop a specific task.

<upid> [<string>] Unique Process/Task Identifier

`proxmox-backup-manager traffic-control create <name> --network <string> [OPTIONS]`

Create new traffic control rule.

<name> [<string>] Rule ID.

--network <string> Rule applies to Source IPs within this networks Can be specified more than once.

Optional parameters:

--comment <string> Comment (single line).

--timeframe <string> Enable the rule at specific times Can be specified more than once.

--burst-in <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

--burst-out <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

--rate-in <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

--rate-out <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

`proxmox-backup-manager traffic-control list [OPTIONS]`

List configured traffic control rules.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager traffic-control remove <name> [OPTIONS]`

Remove a traffic control rule from the configuration file.

<name> [<string>] Rule ID.

Optional parameters:

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

`proxmox-backup-manager traffic-control show <name> [OPTIONS]`

Show traffic control configuration

<name> [<string>] Rule ID.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager traffic-control traffic [OPTIONS]`

Show current traffic for all rules.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager traffic-control update <name> [OPTIONS]`

Update traffic control configuration.

<name> [<string>] Rule ID.

Optional parameters:

--delete rate-in|burst-in|rate-out|burst-out|comment|timeframe List of properties to delete. Can be specified more than once.

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--comment <string> Comment (single line).

--network <string> Rule applies to Source IPs within this networks Can be specified more than once.

--timeframe <string> Enable the rule at specific times Can be specified more than once.

--burst-in <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

--burst-out <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

--rate-in <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

--rate-out <string> Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

`proxmox-backup-manager user create <userid> [OPTIONS]`

Create new user.

<userid> [<string>] User ID

Optional parameters:

--password <string> User Password.

--comment <string> Comment (single line).

--email <string> E-Mail Address.

--enable <boolean> (default=true) Enable the account (default). You can set this to '0' to disable the account.

--expire <integer> (0 - N) (default=0) Account expiration date (seconds since epoch). '0' means no expiration date.

--firstname <string> First name.

--lastname <string> Last name.

`proxmox-backup-manager user delete-token <userid> <token-name> [OPTIONS]`

Delete a user's API token

<userid> [<string>] User ID

<token-name> [<string>] The token ID part of an API token authentication id.

This alone does NOT uniquely identify the API token - use a full *Authid* for such use cases.

Optional parameters:

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

`proxmox-backup-manager user generate-token <userid> <token-name> [OPTIONS]`

Generate a new API token with given metadata

<userid> [<string>] User ID

<token-name> [<string>] The token ID part of an API token authentication id.

This alone does NOT uniquely identify the API token - use a full *Authid* for such use cases.

Optional parameters:

--comment <string> Comment (single line).

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--enable <boolean> (default=true) Enable the account (default). You can set this to '0' to disable the account.

--expire <integer> (0 - N) (default=0) Account expiration date (seconds since epoch). '0' means no expiration date.

`proxmox-backup-manager user list [OPTIONS]`

List configured users.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager user list-tokens <userid> [OPTIONS]`

List tokens associated with user.

<userid> [<string>] User ID

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager user permissions <auth-id> [OPTIONS]`

List permissions of user/token.

<auth-id> [<string>] Authentication ID

Optional parameters:

--output-format text|json|json-pretty Output format.

--path <string> Access control path.

`proxmox-backup-manager user remove <userid> [OPTIONS]`

Remove a user from the configuration file.

<userid> [<string>] User ID

Optional parameters:

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

`proxmox-backup-manager user update <userid> [OPTIONS]`

Update user configuration.

<userid> [<string>] User ID

Optional parameters:

--delete comment|firstname|lastname|email List of properties to delete. Can be specified more than once.

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--password <string> User Password.

--comment <string> Comment (single line).

--email <string> E-Mail Address.

--enable <boolean> (default=true) Enable the account (default). You can set this to '0' to disable the account.

--expire <integer> (0 - N) (default=0) Account expiration date (seconds since epoch). '0' means no expiration date.

--firstname <string> First name.

--lastname <string> Last name.

`proxmox-backup-manager verify <store> [OPTIONS]`

Verify backups

<store> [<string>] Datastore name.

Optional parameters:

--ignore-verified <boolean> (default=true) Do not verify backups that are already verified if their verification is not outdated.

--outdated-after <integer> (0 - N) Days after that a verification becomes outdated. (0 is deprecated)'

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager verify-job create <id> --store <string> [OPTIONS]`

Create a new verification job.

<id> [<string>] Job ID.

--store <string> Datastore name.

Optional parameters:

--comment <string> Comment (single line).

--ignore-verified <boolean> (default=true) Do not verify backups that are already verified if their verification is not outdated.

--max-depth <integer> (0 - 7) (default=7) How many levels of namespaces should be operated on (0 == no recursion)

--ns <string> Namespace.

--outdated-after <integer> (0 - N) Days after that a verification becomes outdated. (0 is deprecated)'

--schedule <calendar-event> Run verify job at specified schedule.

`proxmox-backup-manager verify-job list [OPTIONS]`

List all verification jobs

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager verify-job remove <id> [OPTIONS]`

Remove a verification job configuration

<id> [<string>] Job ID.

Optional parameters:

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

`proxmox-backup-manager verify-job run <id> [OPTIONS]`

Run the specified verification job

<id> [<string>] Job ID.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager verify-job show <id> [OPTIONS]`

Show verification job configuration

<id> [<string>] Job ID.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-manager verify-job update <id> [OPTIONS]`

Update verification job config.

<id> [<string>] Job ID.

Optional parameters:

--delete ignore-verified|comment|schedule|outdated-after|ns|max-depth List of properties to delete. Can be specified more than once.

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--comment <string> Comment (single line).

--ignore-verified <boolean> (default=true) Do not verify backups that are already verified if their verification is not outdated.

--max-depth <integer> (0 - 7) (default=7) How many levels of namespaces should be operated on (0 == no recursion)

--ns <string> Namespace.

--outdated-after <integer> (0 - N) Days after that a verification becomes outdated. (0 is deprecated)'

--schedule <calendar-event> Run verify job at specified schedule.

--store <string> Datastore name.

`proxmox-backup-manager versions [OPTIONS]`

List package versions for important Proxmox Backup Server packages.

Optional parameters:

`--output-format text|json|json-pretty` Output format.

`--verbose <boolean> (default=false)` Output verbose package information. It is ignored if output-format is specified.

A.3 `proxmox-tape`

`proxmox-tape backup <store> <pool> [OPTIONS]`

Backup datastore to tape media pool

`<store>` [`<string>`] Datastore name.

`<pool>` [`<string>`] Media pool name.

Optional parameters:

`--drive <string>` Drive Identifier.

`--eject-media <boolean>` Eject media upon job completion.

`--export-media-set <boolean>` Export media set upon job completion.

`--force-media-set <boolean> (default=false)` Ignore the allocation policy and start a new media-set.

`--groups <type:<vm|ct|host>|group:GROUP|regex:RE>` List of group filters. Can be specified more than once.

`--latest-only <boolean>` Backup latest snapshots only.

`--max-depth <integer> (0 - 7) (default=7)` How many levels of namespaces should be operated on (0 == no recursion)

`--notify-user <string>` User ID

`--ns <string>` Namespace.

`--output-format text|json|json-pretty` Output format.

`proxmox-tape backup-job create <id> --drive <string> --pool <string> --store <string> [OPTIONS]`

Create a new tape backup job.

`<id>` [`<string>`] Job ID.

`--drive <string>` Drive Identifier.

`--pool <string>` Media pool name.

`--store <string>` Datastore name.

Optional parameters:

`--comment <string>` Comment (single line).

`--schedule <calendar-event>` Run sync job at specified schedule.

`--eject-media <boolean>` Eject media upon job completion.

`--export-media-set <boolean>` Export media set upon job completion.

--group-filter <type:<vm|ct|host>|group:GROUP|regex:RE> List of group filters. Can be specified more than once.

--latest-only <boolean> Backup latest snapshots only.

--max-depth <integer> (0 - 7) (default=7) How many levels of namespaces should be operated on (0 == no recursion)

--notify-user <string> User ID

--ns <string> Namespace.

`proxmox-tape backup-job list [OPTIONS]`

Tape backup job list.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-tape backup-job remove <id> [OPTIONS]`

Remove a tape backup job configuration

<id> [<string>] Job ID.

Optional parameters:

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

`proxmox-tape backup-job run <id>`

Run THape Backup Job

<id> [<string>] Job ID.

`proxmox-tape backup-job show <id> [OPTIONS]`

Show tape backup job configuration

<id> [<string>] Job ID.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-tape backup-job update <id> [OPTIONS]`

Update the tape backup job

<id> [<string>] Job ID.

Optional parameters:

--delete comment|schedule|eject-media|export-media-set|latest-only|notify-user|group-fil List of properties to delete. Can be specified more than once.

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--comment <string> Comment (single line).

--schedule <calendar-event> Run sync job at specified schedule.

--drive <string> Drive Identifier.

--eject-media <boolean> Eject media upon job completion.

--export-media-set <boolean> Export media set upon job completion.

--group-filter <type:<vm|ct|host>|group:GROUP|regex:RE> List of group filters. Can be specified more than once.

--latest-only <boolean> Backup latest snapshots only.

--max-depth <integer> (0 - 7) (default=7) How many levels of namespaces should be operated on (0 == no recursion)

--notify-user <string> User ID

--ns <string> Namespace.

--pool <string> Media pool name.

--store <string> Datastore name.

`proxmox-tape barcode-label [OPTIONS]`

Label media with barcodes from changer device

Optional parameters:

--drive <string> Drive Identifier.

--output-format text|json|json-pretty Output format.

--pool <string> Media pool name.

`proxmox-tape cartridge-memory [OPTIONS]`

Read Cartridge Memory (Medium auxiliary memory attributes)

Optional parameters:

--drive <string> Drive Identifier.

--output-format text|json|json-pretty Output format.

`proxmox-tape catalog [OPTIONS]`

Scan media and record content

Optional parameters:

--drive <string> Drive Identifier.

--force <boolean> Force overriding existing index.

--output-format text|json|json-pretty Output format.

--scan <boolean> Re-read the whole tape to reconstruct the catalog instead of restoring saved versions.

--verbose <boolean> Verbose mode - log all found chunks.

`proxmox-tape changer config <name> [OPTIONS]`

Get tape changer configuration

<name> [<string>] Tape Changer Identifier.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-tape changer create <name> --path <string> [OPTIONS]`

Create a new changer device

<name> [<string>] Tape Changer Identifier.

--path <string> Path to Linux generic SCSI device (e.g. '/dev/sg4')

Optional parameters:

--export-slots [<integer>, ...] A list of slot numbers, comma separated. Those slots are reserved for Import/Export, i.e. any media in those slots are considered to be 'offline'.

`proxmox-tape changer list [OPTIONS]`

List changers

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-tape changer remove <name>`

Delete a tape changer configuration

<name> [<string>] Tape Changer Identifier.

`proxmox-tape changer scan [OPTIONS]`

Scan for SCSI tape changers

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-tape changer status [<name>] [OPTIONS]`

Get tape changer status

<name> [<string>] Tape Changer Identifier.

Optional parameters:

--cache <boolean> (default=true) Use cached value.

--output-format text|json|json-pretty Output format.

`proxmox-tape changer transfer [<name>] --from <integer> (1 - N) --to <integer> (1 - N)`

Transfers media from one slot to another

<name> [<string>] Tape Changer Identifier.

--from <integer> (1 - N) Source slot number

--to <integer> (1 - N) Destination slot number

`proxmox-tape changer update <name> [OPTIONS]`

Update a tape changer configuration

<name> [<string>] Tape Changer Identifier.

Optional parameters:

--delete export-slots List of properties to delete. Can be specified more than once.

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--export-slots [<integer>, ...] A list of slot numbers, comma separated. Those slots are reserved for Import/Export, i.e. any media in those slots are considered to be 'offline'.

--path <string> Path to Linux generic SCSI device (e.g. '/dev/sg4')

`proxmox-tape clean [OPTIONS]`

Clean drive

Optional parameters:

--drive <string> Drive Identifier.

--output-format text|json|json-pretty Output format.

`proxmox-tape drive config <name> [OPTIONS]`

Get pool configuration

<name> [<string>] Drive Identifier.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-tape drive create <name> --path <string> [OPTIONS]`

Create a new drive

<name> [<string>] Drive Identifier.

--path <string> The path to a LTO SCSI-generic tape device (i.e. '/dev/sg0')

Optional parameters:

--changer <string> Tape Changer Identifier.

--changer-drivenum <integer> (0 - 255) (default=0) Associated changer drive number (requires option changer)

`proxmox-tape drive list [OPTIONS]`

List drives

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-tape drive remove <name>`

Delete a drive configuration

<name> [<string>] Drive Identifier.

`proxmox-tape drive scan [OPTIONS]`

Scan for drives

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-tape drive update <name> [OPTIONS]`

Update a drive configuration

<name> [<string>] Drive Identifier.

Optional parameters:

--delete changer|changer-drivenum List of properties to delete. Can be specified more than once.

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

--changer <string> Tape Changer Identifier.

--changer-drivenum <integer> (0 - 255) (default=0) Associated changer drive number (requires option changer)

--path <string> The path to a LTO SCSI-generic tape device (i.e. '/dev/sg0')

`proxmox-tape eject [OPTIONS]`

Eject/Unload drive media

Optional parameters:

--drive <string> Drive Identifier.

--output-format text|json|json-pretty Output format.

`proxmox-tape eod [OPTIONS]`

Move to end of media (MTEOM, used to debug)

Optional parameters:

--drive <string> Drive Identifier.

`proxmox-tape export-media <label-text> [OPTIONS]`

Export media with specified label

<label-text> [<string>] Media Label/Barcode.

Optional parameters:

--drive <string> Drive Identifier.

`proxmox-tape format [OPTIONS]`

Format media

Optional parameters:

--drive <string> Drive Identifier.

--fast <boolean> (default=true) Use fast erase.

--output-format text|json|json-pretty Output format.

`proxmox-tape help [{<command>}] [OPTIONS]`

Get help about specified command (or sub-command).

<command> [<string>] Command. This may be a list in order to spefify nested sub-commands. Can be specified more than once.

Optional parameters:

--verbose <boolean> Verbose help.

`proxmox-tape inventory [OPTIONS]`

List (and update) media labels (Changer Inventory)

Optional parameters:

--catalog <boolean> (default=false) Try to restore catalogs from tapes.

--drive <string> Drive Identifier.

--output-format text|json|json-pretty Output format.

--read-all-labels <boolean> (default=false) Load all tapes and try read labels (even if already inventoried)

--read-labels <boolean> (default=false) Load unknown tapes and try read labels

`proxmox-tape key change-passphrase <fingerprint> --hint <string> [OPTIONS]`

Change the encryption key's password.

<fingerprint> [<string>] Tape encryption key fingerprint (sha256).

--hint <string> Password hint.

Optional parameters:

--force <boolean> (default=false) Reset the passphrase for a tape key, without asking for the old one.

--kdf none|scrypt|pbkdf2 (default=scrypt) Key derivation function for password protected encryption keys.

`proxmox-tape key create --hint <string> [OPTIONS]`

Create key (read password from stdin)

--hint <string> Password restore hint.

Optional parameters:

--kdf none|scrypt|pbkdf2 (default=scrypt) Key derivation function for password protected encryption keys.

`proxmox-tape key list [OPTIONS]`

List keys

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-tape key paperkey <fingerprint> [OPTIONS]`

Generate a printable, human readable text file containing the encryption key.

This also includes a scanable QR code for fast key restore.

<fingerprint> [<string>] Tape encryption key fingerprint (sha256).

Optional parameters:

--output-format text|html Paperkey output format

--subject <string> Include the specified subject as title text.

`proxmox-tape key remove <fingerprint> [OPTIONS]`

Remove a encryption key from the database

Please note that you can no longer access tapes using this key.

<fingerprint> [<string>] Tape encryption key fingerprint (sha256).

Optional parameters:

--digest <string> Prevent changes if current configuration file has different SHA256 digest. This can be used to prevent concurrent modifications.

`proxmox-tape key restore [OPTIONS]`

Restore encryption key from tape or from a backup file/string (reads password from stdin)

Optional parameters:

--drive <string> Drive Identifier.

--key <string> Import key from json string or an exported paperkey-format.

--key-file <string> Import key from a file with either json or exported paperkey-format.

`proxmox-tape key show <fingerprint> [OPTIONS]`

Print the encryption key's metadata.

<fingerprint> [<string>] Tape encryption key fingerprint (sha256).

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-tape label --label-text <string> [OPTIONS]`

Label media

--label-text <string> Media Label/Barcode.

Optional parameters:

--drive <string> Drive Identifier.

--output-format text|json|json-pretty Output format.

--pool <string> Media pool name.

`proxmox-tape load-media <label-text> [OPTIONS]`

Load media with specified label

<label-text> [<string>] Media Label/Barcode.

Optional parameters:

--drive <string> Drive Identifier.

--output-format text|json|json-pretty Output format.

`proxmox-tape load-media-from-slot <source-slot> [OPTIONS]`

Load media from the specified slot

<source-slot> [<integer> (1 - N)] Source slot number.

Optional parameters:

--drive <string> Drive Identifier.

`proxmox-tape media content [OPTIONS]`

List media content

Optional parameters:

--output-format text|json|json-pretty Output format.

--backup-id <string> Backup ID.

--backup-type vm|ct|host Backup types.

--label-text <string> Media Label/Barcode.

--media <string> Media Uuid.

--media-set <string> MediaSet Uuid (We use the all-zero Uuid to reseve an empty media for a specific pool).

--pool <string> Media pool name.

`proxmox-tape media destroy <label-text> [OPTIONS]`

Destroy media (completely remove from database)

<label-text> [<string>] Media Label/Barcode.

Optional parameters:

--force <boolean> Force removal (even if media is used in a media set).

`proxmox-tape media list [OPTIONS]`

List pool media

Optional parameters:

--output-format text|json|json-pretty Output format.

--pool <string> Media pool name.

--update-status <boolean> (default=true) Try to update tape library status (check what tapes are online).

--update-status-changer <string> Tape Changer Identifier.

`proxmox-tape pool config <name> [OPTIONS]`

Get media pool configuration

<name> [<string>] Media pool name.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-tape pool create <name> [OPTIONS]`

Create a new media pool

<name> [<string>] Media pool name.

Optional parameters:

--allocation <string> Media set allocation policy ('continue', 'always', or a calendar event).

--comment <string> Comment (single line).

--encrypt <string> Tape encryption key fingerprint (sha256).

--retention <string> Media retention policy ('overwrite', 'keep', or time span).

--template <string> Media set naming template (may contain strftime() time format specifications).

`proxmox-tape pool list [OPTIONS]`

List media pool

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-tape pool remove <name>`

Delete a media pool configuration

<name> [<string>] Media pool name.

`proxmox-tape pool update <name> [OPTIONS]`

Update media pool settings

<name> [<string>] Media pool name.

Optional parameters:

--delete allocation|retention|template|encrypt|comment List of properties to delete. Can be specified more than once.

--allocation <string> Media set allocation policy ('continue', 'always', or a calendar event).

--comment <string> Comment (single line).

--encrypt <string> Tape encryption key fingerprint (sha256).

--retention <string> Media retention policy ('overwrite', 'keep', or time span).

--template <string> Media set naming template (may contain strftime() time format specifications).

`proxmox-tape read-label [OPTIONS]`

Read media label

Optional parameters:

--drive <string> Drive Identifier.

--inventorize <boolean> Inventorize media

--output-format text|json|json-pretty Output format.

`proxmox-tape restore <media-set> <store> [{<snapshots>}] [OPTIONS]`

Restore data from media-set

<media-set> [<string>] Media set UUID.

<store> [[(<source>=)?<target>, ...]] A list of Datastore mappings (or single datastore), comma separated. For example 'a=b,e' maps the source datastore 'a' to target 'b and all other sources to the default 'e'. If no default is given, only the specified sources are mapped.

<snapshots> [store:[ns/namespace/...]type/id/time] List of snapshots. Can be specified more than once.

Optional parameters:

--drive <string> Drive Identifier.

--namespaces [store=<string> [,max-depth=<integer>] [,source=<string>] [,target=<string> List of namespace to restore. Can be specified more than once.

--notify-user <string> User ID

--output-format text|json|json-pretty Output format.

--owner <string> Authentication ID

`proxmox-tape rewind [OPTIONS]`

Rewind tape

Optional parameters:

--drive <string> Drive Identifier.

--output-format text|json|json-pretty Output format.

`proxmox-tape scan [OPTIONS]`

Rewind, then read media contents and print debug info

Note: This reads unless the driver returns an IO Error, so this method is expected to fails when we reach EOT.

Optional parameters:

--drive <string> Drive Identifier.

`proxmox-tape status [OPTIONS]`

Get drive/media status

Optional parameters:

--drive <string> Drive Identifier.

--output-format text|json|json-pretty Output format.

`proxmox-tape unload [OPTIONS]`

Unload media via changer

Optional parameters:

--drive <string> Drive Identifier.

--output-format text|json|json-pretty Output format.

--target-slot <integer> (1 - N) Target slot number. If omitted, defaults to the slot that the drive was loaded from.

`proxmox-tape volume-statistics [OPTIONS]`

Read Volume Statistics (SCSI log page 17h)

Optional parameters:

--drive <string> Drive Identifier.

--output-format text|json|json-pretty Output format.

A.4 pmt

All commands support the following parameters to specify the tape device:

--device <path> Path to the Linux tape device

--drive <name> Use drive from Proxmox Backup Server configuration.

Commands which generate output support the `--output-format` parameter. It accepts the following values:

text Text format (default). Human readable.

json JSON (single line).

json-pretty JSON (multiple lines, nicely formatted).

`pmt asf <count>`

Position the tape at the beginning of the count file (after filemark count)

<count> [`<integer> (0 - 2147483647)`] File mark position (0 is BOT).

`pmt bsf <count>`

Backward space count files (position before file mark).

The tape is positioned on the last block of the previous file.

<count> [`<integer> (1 - 2147483647)`] File mark count.

`pmt bsfm <count>`

Backward space count files, then forward space one record (position after file mark).

This leaves the tape positioned at the first block of the file that is count - 1 files before the current file.

<count> [<integer> (1 - 2147483647)] File mark count.

`pmt bsr <count>`

Backward space records.

<count> [<integer> (1 - 2147483647)] Record count.

`pmt cartridge-memory`

Read Cartridge Memory

`pmt eject`

Eject drive media

`pmt eod`

Move to end of media

`pmt erase [OPTIONS]`

Erase media (from current position)

Optional parameters:

--fast <boolean> (default=true) Use fast erase.

`pmt format [OPTIONS]`

Format media, single partition

Optional parameters:

--fast <boolean> (default=true) Use fast erase.

`pmt fsf <count>`

Forward space count files (position after file mark).

The tape is positioned on the first block of the next file.

<count> [<integer> (1 - 2147483647)] File mark count.

`pmt fsfm <count>`

Forward space count files, then backward space one record (position before file mark).

This leaves the tape positioned at the last block of the file that is count - 1 files past the current file.

<count> [<integer> (1 - 2147483647)] File mark count.

`pmt fsr <count>`

Forward space records.

<count> [<integer> (1 - 2147483647)] Record count.

`pmt help [{<command>}] [OPTIONS]`

Get help about specified command (or sub-command).

<command> [<string>] Command. This may be a list in order to spefify nested sub-commands. Can be specified more than once.

Optional parameters:

--verbose <boolean> Verbose help.

`pmt load`

Load media

`pmt lock`

Lock the tape drive door

`pmt options [OPTIONS]`

Set varios drive options

Optional parameters:

--blocksize <integer> (0 - 8388608) Set tape drive block_length (0 is variable length).

--buffer_mode <boolean> Use drive buffer.

--compression <boolean> Enable/disable compression.

--defaults <boolean> Set default options

`pmt rewind`

Rewind the tape

`pmt scan`

Scan for existing tape changer devices

`pmt status`

Drive Status

`pmt tape-alert-flags`

Read Tape Alert Flags

`pmt unlock`

Unlock the tape drive door

`pmt volume-statistics`

Volume Statistics

`pmt weof [<count>]`

Write count (default 1) EOF marks at current position.

<count> [<integer> (1 - 2147483647)] File mark count.

A.5 pmtx

`pmtx help [{<command>}] [OPTIONS]`

Get help about specified command (or sub-command).

<command> [<string>] Command. This may be a list in order to spefify nested sub-commands. Can be specified more than once.

Optional parameters:

--verbose <boolean> Verbose help.

`pmtx inquiry`

Inquiry

`pmtx inventory`

Inventory

`pmtx load <slot> [OPTIONS]`

Load

<slot> [<integer>] Storage slot number (source).

Optional parameters:

--drivenum <integer> Target drive number (defaults to Drive 0)

`pmtx scan`

Scan for existing tape changer devices

`pmtx status`

Changer Status

`pmtx transfer <from> <to>`

Transfer

<from> [<integer>] Source storage slot number.

<to> [<integer>] Target storage slot number.

`pmtx unload [OPTIONS]`

Unload

Optional parameters:

--drivenum <integer> Target drive number (defaults to Drive 0)

--slot <integer> Storage slot number (target). If omitted, defaults to the slot that the drive was loaded from.

A.6 pxar

`pxar create <archive> <source> [OPTIONS]`

Create a new .pxar archive.

<archive> [<string>] Archive name.

<source> [<string>] Source directory.

Optional parameters:

--all-file-systems <boolean> (default=false) Include mounted sudirs.

--entries-max <integer> (0 - 9223372036854775807) (default=1048576) Max number of entries loaded at once into memory

--exclude <string> List of paths or pattern matching files to exclude. Can be specified more than once.

--no-acls <boolean> (default=false) Ignore access control list entries.

--no-device-nodes <boolean> (default=false) Ignore device nodes.

--no-fcaps <boolean> (default=false) Ignore file capabilities.

--no-fifos <boolean> (default=false) Ignore fifos.

--no-sockets <boolean> (default=false) Ignore sockets.

--no-xattrs <boolean> (default=false) Ignore extended file attributes.

`pxar extract <archive> [<target>] [OPTIONS]`

Extract an archive.

<archive> [<string>] Archive name.

<target> [<string>] Target directory

Optional parameters:

--allow-existing-dirs <boolean> (default=false) Allows directories to already exist on restore.

--files-from <string> File containing match pattern for files to restore.

--no-acls <boolean> (default=false) Ignore access control list entries.

--no-device-nodes <boolean> (default=false) Ignore device nodes.

--no-fcaps <boolean> (default=false) Ignore file capabilities.

--no-fifos <boolean> (default=false) Ignore fifos.

--no-sockets <boolean> (default=false) Ignore sockets.

--no-xattrs <boolean> (default=false) Ignore extended file attributes.

--overwrite <boolean> (default=false) overwrite already existing files

--pattern <string> List of paths or pattern matching files to restore Can be specified more than once.

--strict <boolean> (default=false) Stop on errors. Otherwise most errors will simply warn.

`pxar help [{<command>}] [OPTIONS]`

Get help about specified command (or sub-command).

<command> [<string>] Command. This may be a list in order to spefify nested sub-commands. Can be specified more than once.

Optional parameters:

--verbose <boolean> Verbose help.

`pxar list <archive>`

List the contents of an archive.

<archive> [<string>] Archive name.

`pxar mount <archive> <mountpoint> [OPTIONS]`

Mount the archive to the provided mountpoint via FUSE.

<archive> [<string>] Archive name.

<mountpoint> [<string>] Mountpoint for the file system.

Optional parameters:

--verbose <boolean> (default=false) Verbose output, running in the foreground (for debugging).

A.7 proxmox-file-restore

`proxmox-file-restore extract <snapshot> <path> [<target>] [OPTIONS]`

Restore files from a backup snapshot.

<snapshot> [<string>] Group/Snapshot path.

<path> [<string>] Path to restore. Directories will be restored as archive files if extracted to stdout.

<target> [<string>] Target directory path. Use '-' to write to standard output.

Optional parameters:

--base64 <boolean> (default=false) If set, 'path' will be interpreted as base64 encoded.

--crypt-mode none|encrypt|sign-only (default=encrypt) Defines whether data is encrypted (using an AEAD cipher), only signed, or neither.

`--driver` Qemu

`--format plain|pxar|zip|tar` The desired format of the result.

`--keyfd <integer> (0 - N)` Pass an encryption key via an already opened file descriptor.

`--keyfile <string>` Path to encryption key. All data will be encrypted using this key.

`--ns <string>` Namespace.

`--repository <string>` Repository URL.

`--verbose <boolean> (default=false)` Print verbose information

`--zstd <boolean> (default=false)` If true, output will be zstd compressed.

`proxmox-file-restore help [{<command>}] [OPTIONS]`

Get help about specified command (or sub-command).

`<command>` [`<string>`] Command. This may be a list in order to spefify nested sub-commands. Can be specified more than once.

Optional parameters:

`--verbose <boolean>` Verbose help.

`proxmox-file-restore list <snapshot> <path> [OPTIONS]`

List a directory from a backup snapshot.

`<snapshot>` [`<string>`] Group/Snapshot path.

`<path>` [`<string>`] (Sub-)Path to list.

Optional parameters:

`--base64 <boolean> (default=false)` If set, 'path' will be interpreted as base64 encoded.

`--crypt-mode none|encrypt|sign-only (default=encrypt)` Defines whether data is encrypted (using an AEAD cipher), only signed, or neither.

`--driver` Qemu

`--keyfd <integer> (0 - N)` Pass an encryption key via an already opened file descriptor.

`--keyfile <string>` Path to encryption key. All data will be encrypted using this key.

`--ns <string>` Namespace.

`--output-format text|json|json-pretty` Output format.

`--repository <string>` Repository URL.

`--timeout <integer> (1 - N)` Defines the maximum time the call can should take.

`proxmox-file-restore status [OPTIONS]`

Retrieve status information about currently running/mapped restore images

Optional parameters:

`--driver` Qemu

`--output-format text|json|json-pretty` Output format.

`proxmox-file-restore stop <name>`

Immediately stop/unmap a given image. Not typically necessary, as VMs will stop themselves after a timer anyway.

<name> [<string>] The name of the VM to stop.

A.8 proxmox-backup-debug

`proxmox-backup-debug api create <api-path> [OPTIONS]`

Call API on <api-path>

<api-path> [<string>] API path.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-debug api delete <api-path> [OPTIONS]`

Call API on <api-path>

<api-path> [<string>] API path.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-debug api get <api-path> [OPTIONS]`

Call API on <api-path>

<api-path> [<string>] API path.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-debug api ls [<path>] [OPTIONS]`

Get API usage information for <path>

<path> [<string>] API path.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-debug api set <api-path> [OPTIONS]`

Call API on <api-path>

<api-path> [<string>] API path.

Optional parameters:

--output-format text|json|json-pretty Output format.

`proxmox-backup-debug api usage <path> [OPTIONS]`

Get API usage information for <path>

<path> [<string>] API path.

Optional parameters:

--verbose <boolean> (default=false) Verbose output format.

`proxmox-backup-debug diff archive <prev-snapshot> <snapshot> <archive-name> [OPTIONS]`

Diff an archive in two snapshots. The command will output a list of added, modified and deleted files. For modified files, the file metadata (e.g. mode, uid, gid, size, etc.) will be considered. For detecting modification of file content, only mtime will be used by default. If the --compare-content flag is provided, mtime is ignored and file content will be compared.

<prev-snapshot> [<string>] Path for the first snapshot.

<snapshot> [<string>] Path for the second snapshot.

<archive-name> [<string>] Name of the .pxar archive

Optional parameters:

--color always|auto|never (default=auto) Color output options

--compare-content <boolean> (default=false) Compare file content rather than solely relying on mtime for detecting modified files.

--keyfd <integer> (0 - N) Pass an encryption key via an already opened file descriptor.

--keyfile <string> Path to encryption key.

--ns <string> Namespace.

--repository <string> Repository URL.

`proxmox-backup-debug help [{<command>}] [OPTIONS]`

Get help about specified command (or sub-command).

<command> [<string>] Command. This may be a list in order to spefify nested sub-commands. Can be specified more than once.

Optional parameters:

--verbose <boolean> Verbose help.

`proxmox-backup-debug inspect chunk <chunk> [OPTIONS]`

Inspect a chunk

<chunk> [<string>] The chunk file.

Optional parameters:

--decode <string> Path to the file to which the chunk should be decoded, '-' -> decode to stdout.

--digest <string> Needed when searching for references, if set, it will be used for verification when decoding.

--keyfile <string> Path to the keyfile with which the chunk was encrypted.

--output-format text|json|json-pretty Output format.

--reference-filter <string> Path to the directory that should be searched for references.

--use-filename-as-digest <boolean> (default=true) The filename should be used as digest for reference search and decode verification, if no digest is specified.

`proxmox-backup-debug inspect file <file> [OPTIONS]`

Inspect a file, for blob file without decode only the size and encryption mode is printed

<file> [<string>] Path to the file.

Optional parameters:

--decode <string> Path to the file to which the file should be decoded, '-' -> decode to stdout.

--keyfile <string> Path to the keyfile with which the file was encrypted.

--output-format text|json|json-pretty Output format.

`proxmox-backup-debug recover index <file> <chunks> [OPTIONS]`

Restore the data from an index file, given the directory of where chunks are saved, the index file and a keyfile, if needed for decryption.

<file> [<string>] Path to the index file, either .fidx or .didx.

<chunks> [<string>] Path to the directory that contains the chunks, usually <datastore>/.chunks.

Optional parameters:

--ignore-corrupt-chunks <boolean> (default=false) If a chunk is corrupt, warn and write 0-bytes instead to attempt partial recovery.

--ignore-missing-chunks <boolean> (default=false) If a chunk is missing, warn and write 0-bytes instead to attempt partial recovery.

--keyfile <string> Path to a keyfile, if the data was encrypted, a keyfile is needed for decryption.

--output-path <string> Output file path, defaults to *file* without extension, '-' means STDOUT.

--skip-crc <boolean> (default=false) Skip the crc verification, increases the restore speed by lot.

APPENDIX

B

CONFIGURATION FILES

All Proxmox Backup Server configuration files reside in the directory `/etc/proxmox-backup/`.

B.1 `acl.cfg`

B.1.1 File Format

This file contains the access control list for the Proxmox Backup Server API.

Each line starts with `acl:`, followed by 4 additional values separated by colon.

propagate Propagate permissions down the hierarchy

path The object path

User/Token List of users and tokens

Role List of assigned roles

Here is an example list:

```
acl:1:/:root@pam!test:Admin
acl:1:/datastore/store1:user1@pbs:DatastoreAdmin
```

You can use the `proxmox-backup-manager acl` command to manipulate this file.

B.1.2 Roles

The following roles exist:

`Admin` Administrator

`Audit` Auditor

`NoAccess` Disable Access

`DatastoreAdmin` Datastore Administrator

`DatastoreReader` Datastore Reader (inspect datastore content and do restores)

`DatastoreBackup` Datastore Backup (backup and restore owned backups)

`DatastorePowerUser` Datastore PowerUser (backup, restore and prune owned backup)

`DatastoreAudit` Datastore Auditor

`RemoteAudit` Remote Auditor

`RemoteAdmin` Remote Administrator

`RemoteSyncOperator` Syncronisation Opertator

TapeAudit Tape Auditor

TapeAdmin Tape Administrator

TapeOperator Tape Operator

TapeReader Tape Reader

B.2 datastore.cfg

B.2.1 File Format

This file contains a list of datastore configuration sections. Each section starts with the header `datastore: <name>`, followed by the datastore configuration options.

```
datastore: <name1>

   path <path1>
   <option1> <value1>
   ...

datastore: <name2>

   path <path2>
   ...
```

You can use the `proxmox-backup-manager datastore` command to manipulate this file.

B.2.2 Options

Required properties:

path [<string>] Directory name

Optional properties:

comment [<string>] Comment (single line).

gc-schedule [<calendar-event>] Run garbage collection job at specified schedule.

maintenance-mode [[type=<enum> [,message=<string>]]] Maintenance mode, type is either 'offline' or 'read-only', message should be enclosed in "

type = read-only|offline|delete Maintenance type.

message = <string> Message describing the reason for the maintenance.

notify [[[gc=<enum>] [,prune=<enum>] [,sync=<enum>] [,verify=<enum>]]] Datastore notification setting

gc = never|always|error When do we send notifications

prune = never|always|error When do we send notifications

sync = never|always|error When do we send notifications

verify = never|always|error When do we send notifications

notify-user [<string>] User ID

prune-schedule [<calendar-event>] Run prune job at specified schedule.

tuning [[[chunk-order=<enum>] [,sync-level=<enum>]]] Datastore tuning options

chunk-order = none|inode (default=inode) The order to sort chunks by

sync-level = none|file|filesystem (default=filesystem) The level of syncing that is done when writing into a datastore.

verify-new [<boolean>] If enabled, all new backups will be verified right after completion.

keep-daily [<integer> (1 - N)] Number of daily backups to keep.

keep-hourly [<integer> (1 - N)] Number of hourly backups to keep.

keep-last [<integer> (1 - N)] Number of backups to keep.

keep-monthly [<integer> (1 - N)] Number of monthly backups to keep.

keep-weekly [<integer> (1 - N)] Number of weekly backups to keep.

keep-yearly [<integer> (1 - N)] Number of yearly backups to keep.

B.3 domains.cfg

B.3.1 File Format

This file contains the list authentication realms.

Each user configuration section starts with the header <realm-type>: <name>, followed by the realm's configuration options.

For LDAP realms, the LDAP bind password is stored in ldap_passwords.json.

```
openid: master

      client-id pbs
      comment
      issuer-url http://192.168.0.10:8080/realms/master
      username-claim username

ldap: ldap-server

      base-dn OU=People,DC=ldap-server,DC=example,DC=com
      mode ldaps
      server1 192.168.0.10
      sync-attributes email=mail
      sync-defaults-options enable-new=0,remove-vanished=acl;entry
      user-attr uid
      user-classes inetorgperson,posixaccount,person,user
```

You can use the proxmox-backup-manager openid and proxmox-backup-manager ldap commands to manipulate this file.

B.3.2 Options

Section type 'ldap': LDAP configuration properties.

Required properties:

base-dn [<string>] LDAP Domain

server1 [<string>] LDAP server address

user-attr [<string>] Username attribute. Used to map a userid to LDAP to an LDAP dn.

Optional properties:

bind-dn [<string>] LDAP Domain

capath [<string>] CA certificate to use for the server. The path can point to either a file, or a directory. If it points to a file, the PEM-formatted X.509 certificate stored at the path will be added as a trusted certificate. If the path points to a directory, the directory replaces the system's default certificate store at */etc/ssl/certs* - Every file in the directory will be loaded as a trusted certificate.

comment [<string>] Comment (single line).

filter [<string>] Custom LDAP search filter for user sync

mode [ldap|ldap+starttls|ldaps (default=ldap)] LDAP connection type

port [<integer> (0 - 65535)] Port

server2 [<string>] Fallback LDAP server address

sync-attributes [[[email=<string>] [,firstname=<string>] [,lastname=<string>]]] Comma-separated list of key=value pairs for specifying which LDAP attributes map to which PBS user field. For example, to map the LDAP attribute `mail` to PBS's `email`, write `email=mail`.

email = <string> Name of the LDAP attribute containing the user's email address

firstname = <string> Name of the LDAP attribute containing the user's first name

lastname = <string> Name of the LDAP attribute containing the user's last name

sync-defaults-options [[[enable-new=<1|0>] [,remove-vanished=<string>]]] sync defaults options

enable-new = <boolean> Enable new users after sync

remove-vanished = [acl|entry|properties, ...] A semicolon-seperated list of things to remove when they or the user vanishes during user synchronization. The following values are possible: `entry` removes the user when not returned from the sync; `properties` removes any properties on existing user that do not appear in the source. `acl` removes ACLs when the user is not returned from the sync.

user-classes [[<string>, ...] (default=inetorgperson,posixaccount,person,user)] Comma-separated list of allowed objectClass values for user synchronization. For instance, if `user-classes` is set to `person,user`, then user synchronization will consider all LDAP entities where `objectClass: person` *or* `objectClass: user`.

verify [<boolean> (default=false)] Verify server certificate

Section type 'openid': OpenID configuration properties.

Required properties:

client-id [<string>] OpenID Client ID

issuer-url [<string>] OpenID Issuer Url

Optional properties:

acr-values [[<string>, ...]] OpenID ACR List

autocreate [<boolean> (default=false)] Automatically create users if they do not exist.

client-key [<string>] OpenID Client Key

comment [<string>] Comment (single line).

prompt [<string>] OpenID Prompt

scopes [[<string>, ...] (default=email profile)] OpenID Scope List

username-claim [<string>] Use the value of this attribute/claim as unique user name. It is up to the identity provider to guarantee the uniqueness. The OpenID specification only guarantees that Subject ('sub') is unique. Also make sure that the user is not allowed to change that attribute by himself!

B.4 media-pool.cfg

B.4.1 File Format

Each entry starts with the header `pool: <name>`, followed by the media pool configuration options.

```
pool: company1

      allocation always
      retention overwrite

pool: ...
```

You can use the `proxmox-tape pool` command to manipulate this file.

B.4.2 Options

Optional properties:

`allocation` [`<string>`] Media set allocation policy ('continue', 'always', or a calendar event).

`comment` [`<string>`] Comment (single line).

`encrypt` [`<string>`] Tape encryption key fingerprint (sha256).

`retention` [`<string>`] Media retention policy ('overwrite', 'keep', or time span).

`template` [`<string>`] Media set naming template (may contain strftime() time format specifications).

B.5 tape.cfg

B.5.1 File Format

Each LTO drive configuration section starts with the header `lto: <name>`, followed by the drive configuration options.

Tape changer configurations start with the header `changer: <name>`, followed by the changer configuration options.

```
lto: hh8

      changer sl3
      path /dev/tape/by-id/scsi-10WT065325-nst

changer: sl3

      export-slots 14,15,16
      path /dev/tape/by-id/scsi-CJ0JBE0059
```

You can use the `proxmox-tape drive` and `proxmox-tape changer` commands to manipulate this file.

Note: The `virtual:` drive type is experimental and should only be used for debugging.

B.5.2 Options

Section type 'virtual': Simulate tape drives (only for test and debug)

Required properties:

path [<string>] Path to directory

Optional properties:

max-size [<integer> (0 - N)] Virtual tape size

Section type 'lto': Lto SCSI tape driver

Required properties:

path [<string>] The path to a LTO SCSI-generic tape device (i.e. '/dev/sg0')

Optional properties:

changer [<string>] Tape Changer Identifier.

changer-drivenum [<integer> (0 - 255) (default=0)] Associated changer drive number (requires option changer)

Section type 'changer': SCSI tape changer

Required properties:

path [<string>] Path to Linux generic SCSI device (e.g. '/dev/sg4')

Optional properties:

export-slots [[<integer>, ...]] A list of slot numbers, comma separated. Those slots are reserved for Import/Export, i.e. any media in those slots are considered to be 'offline'.

B.6 tape-job.cfg

B.6.1 File Format

Each entry starts with the header backup: <name>, followed by the job configuration options.

```
backup: job1

      drive hh8
      pool p4
      store store3
      schedule daily
backup: ...
```

You can use the proxmox-tape backup-job command to manipulate this file.

B.6.2 Options

Required properties:

drive [<string>] Drive Identifier.

pool [<string>] Media pool name.

store [<string>] Datastore name.

Optional properties:

comment [<string>] Comment (single line).

schedule [<calendar-event>] Run sync job at specified schedule.

eject-media [<boolean>] Eject media upon job completion.

export-media-set [<boolean>] Export media set upon job completion.

group-filter [<type:<vm|ct|host>|group:GROUP|regex:RE>] List of group filters. Can be specified more than once.

latest-only [<boolean>] Backup latest snapshots only.

max-depth [<integer> (0 - 7) (default=7)] How many levels of namespaces should be operated on (0 == no recursion)

notify-user [<string>] User ID

ns [<string>] Namespace.

B.7 user.cfg

B.7.1 File Format

This file contains the list of API users and API tokens.

Each user configuration section starts with the header `user: <name>`, followed by the user configuration options.

API token configuration starts with the header `token: <userid!token_name>`, followed by the token configuration. The data used to authenticate tokens is stored in a separate file (`token.shadow`).

```
user: root@pam

      comment Superuser
      email test@example.local
      ...

token: root@pam!token1

      comment API test token
      enable true
      expire 0
user: ...
```

You can use the `proxmox-backup-manager user` command to manipulate this file.

B.7.2 Options

Section type 'token': ApiToken properties.

Optional properties:

comment [<string>] Comment (single line).

enable [<boolean> (default=true)] Enable the account (default). You can set this to '0' to disable the account.

expire [<integer> (0 - N) (default=0)] Account expiration date (seconds since epoch). '0' means no expiration date.

Section type 'user': User properties.

Optional properties:

comment [<string>] Comment (single line).

email [<string>] E-Mail Address.

enable [<boolean> (default=true)] Enable the account (default). You can set this to '0' to disable the account.

expire [<integer> (0 - N) (default=0)] Account expiration date (seconds since epoch). '0' means no expiration date.

firstname [<string>] First name.

lastname [<string>] Last name.

B.8 remote.cfg

B.8.1 File Format

This file contains information used to access remote servers.

Each entry starts with the header `remote: <name>`, followed by the remote configuration options.

```
remote: server1

      host server1.local
      auth-id sync@pbs
      ...

remote: ...
```

You can use the `proxmox-backup-manager remote` command to manipulate this file.

B.8.2 Options

Required properties:

password [<string>] Password or auth token for remote host (stored as base64 string).

auth-id [<string>] Authentication ID

host [<string>] DNS name or IP address.

Optional properties:

comment [<string>] Comment (single line).

fingerprint [<string>] X509 certificate fingerprint (sha256).

port [<integer>] The (optional) port

B.9 sync.cfg

B.9.1 File Format

Each entry starts with the header `sync: <name>`, followed by the job configuration options.

```
sync: job1

      store store1
      remote-store store1
      remote lina

sync: ...
```

You can use the `proxmox-backup-manager sync-job` command to manipulate this file.

B.9.2 Options

Required properties:

remote [<string>] Remote ID.

remote-store [<string>] Datastore name.

store [<string>] Datastore name.

Optional properties:

comment [<string>] Comment (single line).

group-filter [<type:<vm|ct|host>|group:GROUP|regex:RE>] List of group filters. Can be specified more than once.

max-depth [<integer> (0 - 7)] How many levels of namespaces should be operated on (0 == no recursion, empty == automatic full recursion, namespace depths reduce maximum allowed value)

ns [<string>] Namespace.

owner [<string>] Authentication ID

remote-ns [<string>] Namespace.

remove-vanished [<boolean> (default=false)] Delete vanished backups. This remove the local copy if the remote backup was deleted.

schedule [<calendar-event>] Run sync job at specified schedule.

transfer-last [<integer> (1 - N)] Limit transfer to last N snapshots (per group), skipping others

burst-in [<string>] Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

burst-out [<string>] Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

rate-in [<string>] Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

rate-out [<string>] Byte size with optional unit (B, KB (base 10), MB, GB, ..., KiB (base 2), MiB, Gib, ...).

B.10 verification.cfg

B.10.1 File Format

Each entry starts with the header `verification: <name>`, followed by the job configuration options.

```
verification: verify-store2

      ignore-verified true
      outdated-after 7
      schedule daily
      store store2
verification: ...
```

You can use the `proxmox-backup-manager verify-job` command to manipulate this file.

B.10.2 Options

Required properties:

store [<string>] Datastore name.

Optional properties:

comment [<string>] Comment (single line).

ignore-verified [<boolean> (default=true)] Do not verify backups that are already verified if their verification is not outdated.

max-depth [<integer> (0 - 7) (default=7)] How many levels of namespaces should be operated on (0 == no recursion)

ns [<string>] Namespace.

outdated-after [<integer> (0 - N)] Days after that a verification becomes outdated. (0 is deprecated)'

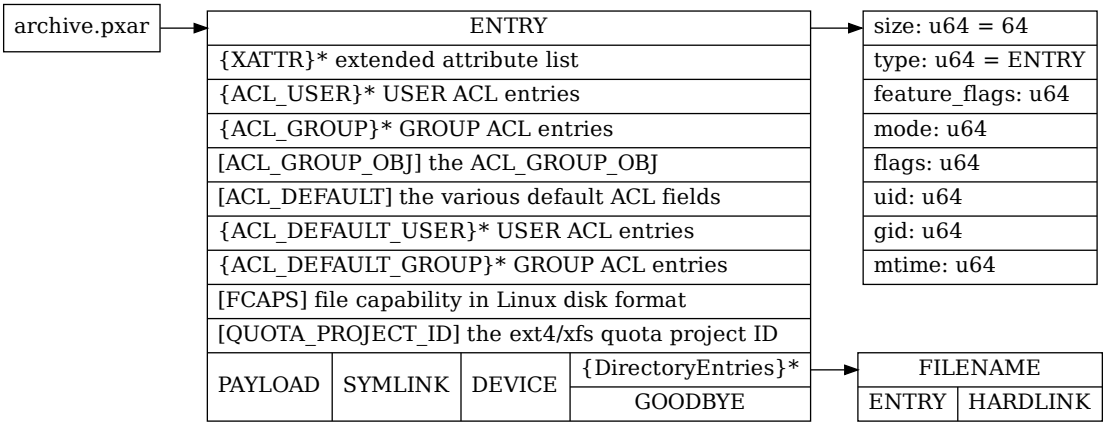
schedule [<calendar-event>] Run verify job at specified schedule.

APPENDIX

C

FILE FORMATS

C.1 Proxmox File Archive Format (`.pxar`)



C.2 Data Blob Format (`.blob`)

The data blob format is used to store small binary data. The magic number decides the exact format:

`[66, 171, 56, 7, 190, 131, 112, 161]`	unencrypted	uncompressed
`[49, 185, 88, 66, 111, 182, 163, 127]`	unencrypted	compressed
`[123, 103, 133, 190, 34, 45, 76, 240]`	encrypted	uncompressed
`[230, 89, 27, 191, 11, 191, 216, 11]`	encrypted	compressed

The compression algorithm used is `zstd`. The encryption cipher is `AES_256_GCM`.

Unencrypted blobs use the following format:

`MAGIC: [u8; 8]`
`CRC32: [u8; 4]`
`Data: (max 16MiB)`

Encrypted blobs additionally contain a 16 byte initialization vector (IV), followed by a 16 byte authenticated encryption (AE) tag, followed by the encrypted data:

MAGIC: [u8; 8]
CRC32: [u8; 4]
IV: [u8; 16]
TAG: [u8; 16]
Data: (max 16MiB)

C.3 Fixed Index Format (`.fidx`)

All numbers are stored as little-endian.

MAGIC: [u8; 8]	[47, 127, 65, 237, 145, 253, 15, 205]
uuid: [u8; 16],	Unique ID
ctime: i64,	Creation Time (epoch)
index_csum: [u8; 32],	SHA-256 over the index (without header) SHA256(digest1||digest2||...)
size: u64,	Image size
chunk_size: u64,	Chunk size
reserved: [u8; 4016],	Overall header size is one page (4096 bytes)
digest1: [u8; 32]	First chunk digest
digest2: [u8; 32]	Second chunk digest
...	Next chunk digest ...

C.4 Dynamic Index Format (`.didx`)

All numbers are stored as little-endian.

MAGIC: [u8; 8]	[28, 145, 78, 165, 25, 186, 179, 205]
uuid: [u8; 16],	Unique ID
ctime: i64,	Creation Time (epoch)
index_csum: [u8; 32],	SHA-256 over the index (without header) SHA256(offset1||digest1||offset2||digest2||...)
reserved: [u8; 4032],	Overall header size is one page (4096 bytes)
offset1: u64	End of first chunk
digest1: [u8; 32]	First chunk digest
offset2: u64	End of second chunk
digest2: [u8; 32]	Second chunk digest
...	Next chunk offset/digest

APPENDIX

D

BACKUP PROTOCOL

Proxmox Backup Server uses a REST-based API. While the management interface uses normal HTTP, the actual backup and restore interface uses HTTP/2 for improved performance. Both HTTP and HTTP/2 are well known standards, so the following section assumes that you are familiar with how to use them.

D.1 Backup Protocol API

To start a new backup, the API call `GET /api2/json/backup` needs to be upgraded to a HTTP/2 connection using `proxmox-backup-protocol-v1` as the protocol name:

```
GET /api2/json/backup HTTP/1.1
UPGRADE: proxmox-backup-protocol-v1
```

The server replies with the `HTTP 101 Switching Protocol` status code, and you can then issue REST commands on the updated HTTP/2 connection.

The backup protocol allows you to upload three different kind of files:

- Chunks and blobs (binary data)
- Fixed indexes (List of chunks with fixed size)
- Dynamic indexes (List of chunks with variable size)

The following section provides a short introduction on how to upload such files. Please use the API Viewer for details about the available REST commands.

D.1.1 Upload Blobs

Blobs are uploaded using `POST /blob`. The HTTP body contains the data encoded as *Data Blob*.

The file name must end with `.blob`, and is automatically added to the backup manifest, following the call to `POST /finish`.

D.1.2 Upload Chunks

Chunks belong to an index, so you first need to open an index (see below). After that, you can upload chunks using `POST /fixed_chunk` and `POST /dynamic_chunk`. The HTTP body contains the chunk data encoded as *Data Blob*).

D.1.3 Upload Fixed Indexes

Fixed indexes are used to store VM image data. The VM image is split into equally sized chunks, which are uploaded individually. The index file simply contains a list of chunk digests.

You create a fixed index with `POST /fixed_index`. Then, upload chunks with `POST /fixed_chunk`, and append them to the index with `PUT /fixed_index`. When finished, you need to close the index using `POST /fixed_close`.

The file name needs to end with `.fidx`, and is automatically added to the backup manifest, following the call to `POST /finish`.

D.1.4 Upload Dynamic Indexes

Dynamic indexes are used to store file archive data. The archive data is split into dynamically sized chunks, which are uploaded individually. The index file simply contains a list of chunk digests and offsets.

You can create a dynamically sized index with `POST /dynamic_index`. Then, upload chunks with `POST /dynamic_chunk`, and append them to the index with `PUT /dynamic_index`. When finished, you need to close the index using `POST /dynamic_close`.

The filename needs to end with `.didx`, and is automatically added to the backup manifest, following the call to `POST /finish`.

D.1.5 Finish Backup

Once you have uploaded all data, you need to call `POST /finish`. This commits all data and ends the backup protocol.

D.2 Restore/Reader Protocol API

To start a new reader, the API call `GET /api2/json/reader` needs to be upgraded to a HTTP/2 connection using `proxmox-backup-reader-protocol-v1` as protocol name:

```
GET /api2/json/reader HTTP/1.1
UPGRADE: proxmox-backup-reader-protocol-v1
```

The server replies with the `HTTP 101 Switching Protocol` status code, and you can then issue REST commands on that updated HTTP/2 connection.

The reader protocol allows you to download three different kinds of files:

- Chunks and blobs (binary data)
- Fixed indexes (list of chunks with fixed size)
- Dynamic indexes (list of chunks with variable size)

The following section provides a short introduction on how to download such files. Please use the API Viewer for details about the available REST commands.

D.2.1 Download Blobs

Blobs are downloaded using `GET /download`. The HTTP body contains the data encoded as *Data Blob*.

D.2.2 Download Chunks

Chunks are downloaded using `GET /chunk`. The HTTP body contains the data encoded as *Data Blob*.

D.2.3 Download Index Files

Index files are downloaded using `GET /download`. The HTTP body contains the data encoded as *Fixed Index* or *Dynamic Index*.

APPENDIX

E

CALENDAR EVENTS

E.1 Introduction and Format

Certain tasks, for example pruning and garbage collection, need to be performed on a regular basis. Proxmox Backup Server uses a format inspired by the systemd Time and Date Specification (see systemd.time manpage) called *calendar events* for its schedules.

Calendar events are expressions to specify one or more points in time. They are mostly compatible with systemd's calendar events.

The general format is as follows:

Listing 1: Calendar event

```
[WEEKDAY] [[YEARS-]MONTHS-DAYS] [HOURS:MINUTES[:SECONDS]]
```

Note that there either has to be at least a weekday, date or time part. If the weekday or date part is omitted, all (week)days are included. If the time part is omitted, the time 00:00:00 is implied. (e.g. '2020-01-01' refers to '2020-01-01 00:00:00')

Weekdays are specified with the abbreviated English version: *mon, tue, wed, thu, fri, sat, sun*.

Each field can contain multiple values in the following formats:

- comma-separated: e.g., 01,02,03
- as a range: e.g., 01..10
- as a repetition: e.g, 05/10 (means starting at 5 every 10)
- and a combination of the above: e.g., 01,05..10,12/02
- or a * for every possible value: e.g., *:00

There are some special values that have a specific meaning:

Value	Syntax
minutely	**-*-* *:*:00*
hourly	**-*-* *:00:00*
daily	**-*-* 00:00:00*
weekly	*mon *-*-* 00:00:00*
monthly	**-*-01 00:00:00*
yearly or *annually*	**-01-01 00:00:00*
quarterly	**-01,04,07,10-01 00:00:00*
semiannually or *semi-annually*	**-01,07-01 00:00:00*

Here is a table with some useful examples:

Example	Alternative	Explanation
mon,tue,wed,thu,fri	*mon..fri*	Every working day at 00:00
sat,sun	*sat..sun*	Only on weekends at 00:00
mon,wed,fri	--	Monday, Wednesday, Friday at 00:00
12:05	--	Every day at 12:05 PM
*:00/5	*0/1:0/5*	Every five minutes
*mon..wed *:30/10*	*mon,tue,wed *:30/10*	Monday, Tuesday, Wednesday 30, 40 and 50 minutes after every full hour
mon..fri 8..17,22:0/15	--	Every working day every 15 minutes between 8 AM and 6 PM and between 10 PM and 11 PM
fri 12..13:5/20	*fri 12,13:5/20*	Friday at 12:05, 12:25, 12:45, 13:05, 13:25 and 13:45
12,14,16,18,20,22:5	*12/2:5*	Every day starting at 12:05 until 22:05, every 2 hours
:	*0/1:0/1*	Every minute (minimum interval)
*-05	--	On the 5th day of every Month
*Sat *-1..7 15:00*	--	First Saturday each Month at 15:00
2015-10-21	--	21st October 2015 at 00:00

E.2 Differences to systemd

Not all features of systemd calendar events are implemented:

- no Unix timestamps (e.g. *@12345*): instead use date and time to specify a specific point in time
- no timezone: all schedules use the timezone of the server
- no sub-second resolution
- no reverse day syntax (e.g. 2020-03~01)
- no repetition of ranges (e.g. 1..10/2)

E.3 Notes on Scheduling

In Proxmox Backup, scheduling for most tasks is done in the *proxmox-backup-proxy*. This daemon checks all job schedules every minute, to see if any are due. This means that even though *calendar events* can contain seconds, it will only be checked once per minute.

Also, all schedules will be checked against the timezone set in the Proxmox Backup server.

APPENDIX

F

MARKDOWN PRIMER

> "Markdown is a text-to-HTML conversion tool for web writers. Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convertit to structurally valid XHTML (or HTML)."
>
> —John Gruber, https://daringfireball.net/projects/markdown/

The "Notes" panel of the Proxmox Backup Server web-interface supports rendering Markdown text.

Proxmox Backup Server supports CommonMark with most extensions of GFM (GitHub Flavoured Markdown), like tables or task-lists.

F.1 Markdown Basics

Note that we only describe the basics here. Please search the web for more extensive resources, for example on https://www.markdownguide.org/

F.1.1 Headings

```
# This is a Heading h1
## This is a Heading h2
##### This is a Heading h5
```

F.1.2 Emphasis

Use `*text*` or `_text_` for emphasis.

Use `**text**` or `__text__` for bold, heavy-weight text.

Combinations are also possible, for example:

```
_You **can** combine them_
```

F.1.3 Links

You can use automatic detection of links. For example, `https://forum.proxmox.com/` would transform it into a clickable link.

You can also control the link text, for example:

```
Now, [the part in brackets will be the link text](https://forum.proxmox.com/).
```

F.1.4 Lists

Unordered Lists

Use * or - for unordered lists, for example:

```
* Item 1
* Item 2
* Item 2a
* Item 2b
```

You can create nested lists by adding indentation.

Ordered Lists

```
1. Item 1
1. Item 2
1. Item 3

   1. Item 3a
   1. Item 3b
```

NOTE: The integer of ordered lists does not need to be correct, they will be numbered automatically.

Task Lists

Task lists use a empty box [ ] for unfinished tasks and a box with an *X* for finished tasks.

For example:

```
- [X] First task already done!
- [X] Second one too
- [ ] This one is still to-do
- [ ] So is this one
```

F.1.5 Tables

Tables use the pipe symbol | to separate columns, and - to separate the table header from the table body. In that separation, you can also set the text alignment, making one column left-, center-, or right-aligned.

```
| Left columns  | Right columns |  Some   | More | Cols.| Centering Works Too
| ------------- |--------------:|---------|------|------|:------------------:|
| left foo      | right foo     | First   | Row  | Here | >center<           |
| left bar      | right bar     | Second  | Row  | Here | 12345              |
| left baz      | right baz     | Third   | Row  | Here | Test               |
| left zab      | right zab     | Fourth  | Row  | Here | ---              |
| left rab      | right rab     | And     | Last | Here | The End            |
```

Note that you do not need to align the columns nicely with white space, but that makes editing tables easier.

F.1.6 Block Quotes

You can enter block quotes by prefixing a line with >, similar as in plain-text emails.

```
> Markdown is a lightweight markup language with plain-text-formatting syntax,
> created in 2004 by John Gruber with Aaron Swartz.
>
>> Markdown is often used to format readme files, for writing messages in online discussion␣
↪forums,
>> and to create rich text using a plain text editor.
```

F.1.7 Code and Snippets

You can use backticks to avoid processing a group of words or paragraphs. This is useful for preventing a code or configuration hunk from being mistakenly interpreted as markdown.

Inline Code

Surrounding part of a line with single backticks allows you to write code inline, for examples:

```
This hosts IP address is `10.0.0.1`.
```

Entire Blocks of Code

For code blocks spanning several lines, you can use triple-backticks to start and end such a block, for example:

```
```
This is the network config I want to remember here
auto vmbr2
iface vmbr2 inet static
        address 10.0.0.1/24
        bridge-ports ens20
        bridge-stp off
        bridge-fd 0
        bridge-vlan-aware yes
        bridge-vids 2-4094

```
```

APPENDIX

G

GLOSSARY

Virtual machine A virtual machine is a program that can execute an entire operating system inside an emulated hardware environment.

Container A container is an isolated user space. Programs run directly on the host's kernel, but with limited access to the host's resources.

Datastore A place to store backups. A directory which contains the backup data. The current implementation is file-system based.

Rust Rust is a new, fast and memory-efficient system programming language. It has no runtime or garbage collector. Rust's rich type system and ownership model guarantee memory-safety and thread-safety. This can eliminate many classes of bugs at compile-time.

Sphinx Is a tool that makes it easy to create intelligent and nicely formatted documentation. It was originally created for the documentation of the Python programming language. It has excellent facilities for the documentation of software projects in a range of languages.

reStructuredText Is an easy-to-read, what-you-see-is-what-you-get, plaintext markup syntax and parser system.

FUSE Filesystem in Userspace (FUSE) defines an interface which makes it possible to implement a filesystem in userspace as opposed to implementing it in the kernel. The fuse kernel driver handles filesystem requests and sends them to a userspace application.

Remote A remote Proxmox Backup Server installation and credentials for a user on it. You can pull datastores from a remote to a local datastore in order to have redundant backups.

APPENDIX

H

GNU FREE DOCUMENTATION LICENSE

Version 1.3, 3 November 2008

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document ”free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of ”copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The ”Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as ”you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A ”Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A ”Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The ”Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The ”Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A ”Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not ”Transparent” is called ”Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The ”Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, ”Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The ”publisher” means any person or entity that distributes copies of the Document to the public.

A section ”Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as ”Acknowledgements”, ”Dedications”, ”Endorsements”, or ”History”.) To ”Preserve the Title” of such a section when you modify the Document means that it remains a section ”Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled ”Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled ”Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled ”Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled ”History” in the various original documents, forming one section Entitled ”History”; likewise combine any sections Entitled ”Acknowledgements”, and any sections Entitled ”Dedications”. You must delete all sections Entitled ”Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an ”aggregate” if the

copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See https://www.gnu.org/licenses/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those

works. A public wiki that anybody can edit is an example of such a server. A ”Massive Multiauthor Collaboration” (or ”MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

”CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

”Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is ”eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

INDEX