Proxmox Container Toolkit

Proxmox Server Solutions GmbH
<support@proxmox.com>
version 8.2.2, Thu Apr 25 09:24:16 CEST 2024
↩Index

Table of Contents

Technology Overview
Supported Distributions
- Alpine Linux
- Arch Linux
- CentOS, Almalinux, Rocky Linux
- Debian
- Devuan
- Fedora
- Gentoo
- OpenSUSE
- Ubuntu

Container Images
Container Settings
- General Settings
- CPU
- Memory
- Mount Points
- Network
- Automatic Start and Shutdown of Containers
- Hookscripts

Security Considerations
- AppArmor
- Control Groups (cgroup)

Guest Operating System Configuration
Container Storage
- FUSE Mounts
- Using Quotas Inside Containers
- Using ACLs Inside Containers
- Backup of Container mount points
- Replication of Containers mount points

Backup and Restore
- Container Backup
- Restoring Container Backups

Managing Containers with pct
- CLI Usage Examples
- Obtaining Debugging Logs

Migration
Configuration
- File Format
- Snapshots
- Options

Locks

Containers are a lightweight alternative to fully virtualized machines (VMs). They use the kernel of the host system that they run on, instead of emulating a full operating system (OS). This means that containers can access resources on the host system directly.

The runtime costs for containers is low, usually negligible. However, there are some drawbacks that need be considered:

- Only Linux distributions can be run in Proxmox Containers. It is not possible to run other operating systems like, for example, FreeBSD or Microsoft Windows inside a container.
- For security reasons, access to host resources needs to be restricted. Therefore, containers run in their own separate namespaces. Additionally some syscalls (user space requests to the Linux kernel) are not allowed within containers.

Proxmox VE uses Linux Containers (LXC) as its underlying container technology. The “Proxmox Container Toolkit” (`pct`) simplifies the usage and management of LXC, by providing an interface that abstracts complex tasks.

Containers are tightly integrated with Proxmox VE. This means that they are aware of the cluster setup, and they can use the same network and storage resources as virtual machines. You can also use the Proxmox VE firewall, or manage containers using the HA framework.

Our primary goal is to offer an environment that provides the benefits of using a VM, but without the additional overhead. This means that Proxmox Containers can be categorized as “System Containers”, rather than “Application Containers”.

> If you want to run application containers, for example, *Docker* images, it is recommended that you run them inside a Proxmox QEMU VM. This will give you all the advantages of application containerization, while also providing the benefits that VMs offer, such as strong isolation from the host and the ability to live-migrate, which otherwise isn't possible with containers.

Technology Overview

- LXC (https://linuxcontainers.org/)
- Integrated into Proxmox VE graphical web user interface (GUI)
- Easy to use command-line tool `pct`
- Access via Proxmox VE REST API
- *lxcfs* to provide containerized /proc file system
- Control groups (*cgroups*) for resource isolation and limitation
- *AppArmor* and *seccomp* to improve security
- Modern Linux kernels
- Image based deployment (templates)
- Uses Proxmox VE storage library
- Container setup from host (network, DNS, storage, etc.)

Supported Distributions

List of officially supported distributions can be found below.

Templates for the following distributions are available through our repositories. You can use pveam tool or the Graphical User Interface to download them.

Alpine Linux

> Alpine Linux is a security-oriented, lightweight Linux distribution based on musl libc and busybox.

> — https://alpinelinux.org

For currently supported releases see:

https://alpinelinux.org/releases/

Arch Linux

> Arch Linux, a lightweight and flexible Linux® distribution that tries to Keep It Simple.
>
> — https://archlinux.org/

Arch Linux is using a rolling-release model, see its wiki for more details:

https://wiki.archlinux.org/title/Arch_Linux

CentOS, Almalinux, Rocky Linux

CentOS / CentOS Stream

> The CentOS Linux distribution is a stable, predictable, manageable and reproducible platform derived from the sources of Red Hat Enterprise Linux (RHEL)
>
> — https://centos.org

For currently supported releases see:

https://en.wikipedia.org/wiki/CentOS#End-of-support_schedule

Almalinux

> An Open Source, community owned and governed, forever-free enterprise Linux distribution, focused on long-term stability, providing a robust production-grade platform. AlmaLinux OS is 1:1 binary compatible with RHEL® and pre-Stream CentOS.
>
> — https://almalinux.org

For currently supported releases see:

https://en.wikipedia.org/wiki/AlmaLinux#Releases

Rocky Linux

> Rocky Linux is a community enterprise operating system designed

> to be 100% bug-for-bug compatible with America's top enterprise Linux distribution now that its downstream partner has shifted direction.
>
> — https://rockylinux.org

For currently supported releases see:

https://en.wikipedia.org/wiki/Rocky_Linux#Releases

Debian

> Debian is a free operating system, developed and maintained by the Debian project. A free Linux distribution with thousands of applications to meet our users' needs.
>
> — https://www.debian.org/intro/index#software

For currently supported releases see:

https://www.debian.org/releases/stable/releasenotes

Devuan

> Devuan GNU+Linux is a fork of Debian without systemd that allows users to reclaim control over their system by avoiding unnecessary entanglements and ensuring Init Freedom.
>
> — https://www.devuan.org

For currently supported releases see:

https://www.devuan.org/os/releases

Fedora

> Fedora creates an innovative, free, and open source platform for hardware, clouds, and containers that enables software developers and community members to build tailored solutions for their users.
>
> — https://getfedora.org

For currently supported releases see:

https://fedoraproject.org/wiki/Releases

Gentoo

> a highly flexible, source-based Linux distribution.
>
> — https://www.gentoo.org

Gentoo is using a rolling-release model.

OpenSUSE

> The makers' choice for sysadmins, developers and desktop users.
>
> — https://www.opensuse.org

For currently supported releases see:

https://get.opensuse.org/leap/

Ubuntu

> Ubuntu is the modern, open source operating system on Linux for the enterprise server, desktop, cloud, and IoT.
>
> — https://ubuntu.com/

For currently supported releases see:

https://wiki.ubuntu.com/Releases

Container Images

Container images, sometimes also referred to as “templates” or “appliances”, are `tar` archives which contain everything to run a container.

Proxmox VE itself provides a variety of basic templates for the most common Linux distributions. They can be downloaded using the GUI or the `pveam` (short for Proxmox VE Appliance Manager) command-line utility. Additionally, TurnKey Linux container templates are also available to download.

The list of available templates is updated daily through the *pve-daily-update* timer. You can also trigger an update manually by executing:

```
# pveam update
```

To view the list of available images run:

```
# pveam available
```

You can restrict this large list by specifying the `section` you are interested in, for example basic `system` images:

List available system images

```
# pveam available --section
system
system          alpine-3.12-
default_20200823_amd64.tar.xz
system          alpine-3.13-
default_20210419_amd64.tar.xz
system          alpine-3.14-
default_20210623_amd64.tar.xz
system          archlinux-
base_20210420-1_amd64.tar.gz
system          centos-7-
default_20190926_amd64.tar.xz
system          centos-8-
default_20201210_amd64.tar.xz
system          debian-9.0-
standard_9.7-1_amd64.tar.gz
system          debian-10-
standard_10.7-1_amd64.tar.gz
system          devuan-3.0-
standard_3.0_amd64.tar.gz
system          fedora-33-
default_20201115_amd64.tar.xz
system          fedora-34-
default_20210427_amd64.tar.xz
system          gentoo-current-
default_20200310_amd64.tar.xz
system          opensuse-15.2-
default_20200824_amd64.tar.xz
system          ubuntu-16.04-
standard_16.04.5-1_amd64.tar.gz
system          ubuntu-18.04-
standard_18.04.1-1_amd64.tar.gz
system          ubuntu-20.04-
standard_20.04-1_amd64.tar.gz
system          ubuntu-20.10-
standard_20.10-1_amd64.tar.gz
system          ubuntu-21.04-
standard_21.04-1_amd64.tar.gz
```

Before you can use such a template, you need to download them into one of your storages. If you're unsure to which one, you can simply use the `local` named storage for that purpose. For clustered installations, it is preferred to use a shared storage so that all nodes can access those images.

```
# pveam download local debian-
10.0-standard_10.0-1_amd64.tar.gz
```

You are now ready to create containers using that image, and you can list all downloaded images on storage `local` with:

```
# pveam list local
local:vztmpl/debian-10.0-
standard_10.0-1_amd64.tar.gz
219.95MB
```

> You can also use the Proxmox VE web interface GUI to download, list and delete container templates.

`pct` uses them to create a new container, for example:

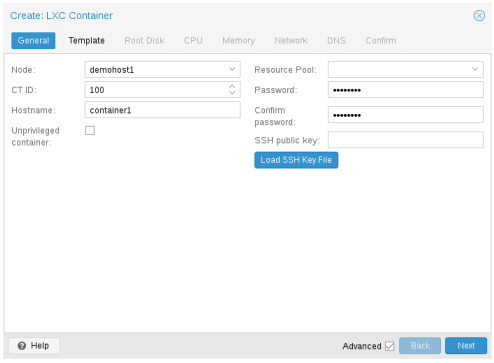
```
# pct create 999
local:vztmpl/debian-10.0-
standard_10.0-1_amd64.tar.gz
```

The above command shows you the full Proxmox VE volume identifiers. They include the storage name, and most other Proxmox VE commands can use them. For example you can delete that image later with:

```
# pveam remove
local:vztmpl/debian-10.0-
standard_10.0-1_amd64.tar.gz
```

Container Settings

General Settings



General settings of a container include

- the **Node** : the physical server on which the container will run

- the **CT ID**: a unique number in this Proxmox VE installation used to identify your container
- **Hostname**: the hostname of the container
- **Resource Pool**: a logical group of containers and VMs
- **Password**: the root password of the container
- **SSH Public Key**: a public key for connecting to the root account over SSH
- **Unprivileged container**: this option allows to choose at creation time if you want to create a privileged or unprivileged container.

Unprivileged Containers

Unprivileged containers use a new kernel feature called user namespaces. The root UID 0 inside the container is mapped to an unprivileged user outside the container. This means that most security issues (container escape, resource abuse, etc.) in these containers will affect a random unprivileged user, and would be a generic kernel security bug rather than an LXC issue. The LXC team thinks unprivileged containers are safe by design.

This is the default option when creating a new container.

> If the container uses systemd as an init system, please be aware the systemd version running inside the container should be equal to or greater than 220.

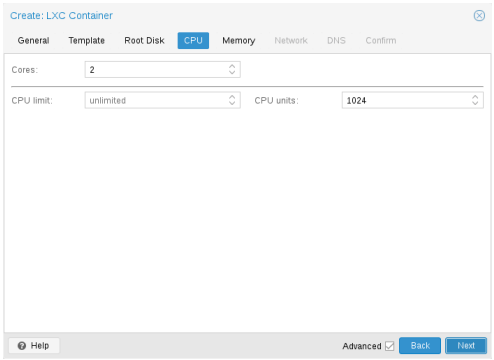
Privileged Containers

Security in containers is achieved by using mandatory access control *AppArmor* restrictions, *seccomp* filters and Linux kernel namespaces. The LXC team considers this kind of container as unsafe, and they will not consider new container escape exploits to be security issues worthy of a CVE and quick fix. That's why privileged containers should only be used in trusted environments.

CPU

You can restrict the number of visible CPUs inside the container using the `cores` option.

This is



implemented using the Linux *cpuset* cgroup (**c**ontrol **group**). A special task inside `pvestatd` tries to distribute running containers among available CPUs periodically. To view the assigned CPUs run the following command:

```
# pct cpusets
 --------------------
102:                6 7
105:       2 3 4 5
108:  0 1
 --------------------
```

Containers use the host kernel directly. All tasks inside a container are handled by the host CPU scheduler. Proxmox VE uses the Linux *CFS* (**C**ompletely **F**air **S**cheduler) scheduler by default, which has additional bandwidth control options.

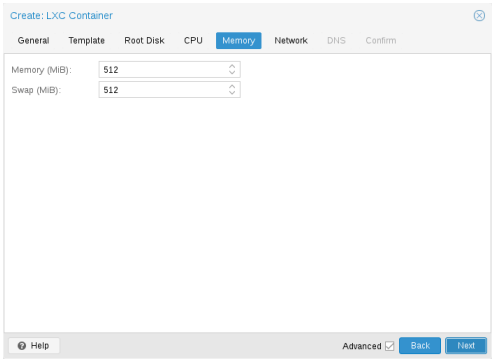
`cpulimit`: You can use this option to further limit assigned CPU time. Please note that this is a floating point number, so it is perfectly valid to assign two cores to a container, but restrict overall CPU consumption to half a core.

```
cores: 2
cpulimit: 0.5
```

`cpuunits`: This is a relative weight passed to the kernel scheduler. The larger the number is, the more CPU time this container gets. Number is relative to the weights of all the other running containers. The default is `100` (or `1024` if the host uses legacy cgroup v1). You can use this setting to prioritize some containers.

Memory

Container memory is controlled using the cgroup memory controller.

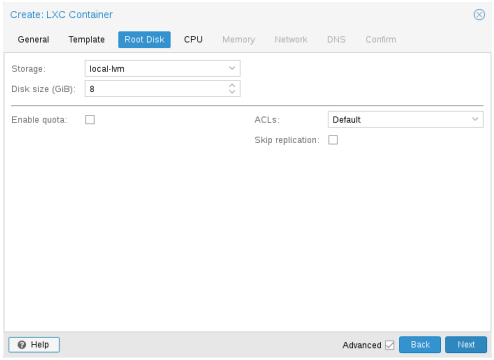


`memory`: Limit overall memory usage. This corresponds to the `memory.limit_in_bytes` cgroup setting.

`swap`: Allows the container to use additional swap memory from the host swap space. This corresponds to the `memory.memsw.limit_in_bytes` cgroup setting, which is set to the sum of both value (`memory + swap`).

Mount Points

The root mount point is configured with the `rootfs` property. You can configure up to 256 additional mount points. The corresponding options are called `mp0` to `mp255`. They can contain the following settings:



`rootfs: [volume=]<volume> [,acl=<1|0>] [,mountoptions=<opt[;opt...]>] [,quota=<1|0>] [,replicate=<1|0>] [,ro=<1|0>] [,shared=<1|0>] [,size=<DiskSize>]`

Use volume as container root. See below for a detailed description of all options.

`mp[n]: [volume=]<volume> ,mp=<Path> [,acl=<1|0>] [,backup=<1|0>] [,mountoptions=<opt[;opt...]>] [,quota=<1|0>] [,replicate=<1|0>] [,ro=<1|0>] [,shared=<1|0>] [,size=<DiskSize>]`

Use volume as container mount point. Use the special syntax

STORAGE_ID:SIZE_IN_GiB to allocate a new volume.

`acl=<boolean>`
: Explicitly enable or disable ACL support.

`backup=<boolean>`
: Whether to include the mount point in backups (only used for volume mount points).

`mountoptions=<opt[;opt...]>`
: Extra mount options for rootfs/mps.

`mp=<Path>`
: Path to the mount point as seen from inside the container.

> Must not contain any symlinks for security reasons.

`quota=<boolean>`
: Enable user quotas inside the container (not supported with zfs subvolumes)

`replicate=<boolean>` (*default* = `1`)
: Will include this volume to a storage replica job.

`ro=<boolean>`
: Read-only mount point

`shared=<boolean>` (*default* = `0`)
: Mark this non-volume mount point as available on all nodes.

> This option does not share the mount point automatically, it assumes it is shared already!

`size=<DiskSize>`
: Volume size (read only value).

`volume=<volume>`
: Volume, device or directory to mount into the container.

Currently there are three types of mount points: storage backed mount points, bind mounts, and device mounts.

Typical container `rootfs` configuration

```
rootfs: thin1:base-100-disk-
1,size=8G
```

Storage Backed Mount Points

Storage backed mount points are managed by the Proxmox VE storage subsystem and come in three different flavors:

- Image based: these are raw images containing a single ext4 formatted file system.
- ZFS subvolumes: these are technically bind mounts, but with managed storage, and thus allow resizing and snapshotting.
- Directories: passing `size=0` triggers a special case where instead of a raw image a directory is created.

> The special option syntax `STORAGE_ID:SIZE_IN_GB` for storage backed mount point volumes will automatically allocate a volume of the specified size on the specified storage. For example, calling

```
pct set 100 -mp0
thin1:10,mp=/path/in/container
```

will allocate a 10GB volume on the storage `thin1` and replace the volume ID place holder `10` with the allocated volume ID, and setup the moutpoint in the container at `/path/in/container`

Bind Mount Points

Bind mounts allow you to access arbitrary directories from your Proxmox VE host inside a container. Some potential use cases are:

- Accessing your home directory in the guest
- Accessing an USB device directory in the guest
- Accessing an NFS mount from the host in the guest

Bind mounts are considered to not be managed by the storage subsystem, so you cannot make snapshots or deal with quotas from inside the container. With unprivileged containers you

might run into permission problems caused by the user mapping and cannot use ACLs.

> The contents of bind mount points are not backed up when using `vzdump`.

- For security reasons, bind mounts should only be established using source directories especially reserved for this purpose, e.g., a directory hierarchy under `/mnt/bindmounts`. Never bind mount system directories like `/`, `/var` or `/etc` into a container - this poses a great security risk.

> The bind mount source path must not contain any symlinks.

For example, to make the directory `/mnt/bindmounts/shared` accessible in the container with ID `100` under the path `/shared`, add a configuration line such as:

```
mp0:
/mnt/bindmounts/shared,mp=/shared
```

into `/etc/pve/lxc/100.conf`.

Or alternatively use the `pct` tool:

```
pct set 100 -mp0
/mnt/bindmounts/shared,mp=/shared
```

to achieve the same result.

Device Mount Points

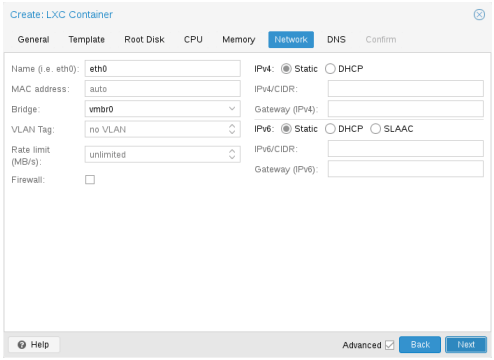
Device mount points allow to mount block devices of the host directly into the container. Similar to bind mounts, device mounts are not managed by Proxmox VE's storage subsystem, but the `quota` and `acl` options will be honored.

> Device mount points should only be used under special circumstances. In most cases a storage backed mount point offers the same performance and a lot more features.

> The contents of device mount points are not backed up when using `vzdump`.

Network

You can configure up to 10 network interfaces for a single container. The



corresponding options are called `net0` to `net9`, and they can contain the following setting:

`net[n]:name=<string> [,bridge=<bridge>] [,firewall=<1|0>] [,gw=<GatewayIPv4>] [,gw6=<GatewayIPv6>] [,hwaddr=<XX:XX:XX:XX:XX:XX>] [,ip=<(IPv4/CIDR|dhcp|manual)>] [,ip6=<(IPv6/CIDR|auto|dhcp|manual)>] [,link_down=<1|0>] [,mtu=<integer>] [,rate=<mbps>] [,tag=<integer>] [,trunks=<vlanid[;vlanid...]>] [,type=<veth>]`

Specifies network interfaces for the container.

`bridge=<bridge>`

Bridge to attach the network device to.

`firewall=<boolean>`

Controls whether this interface's firewall rules should be used.

`gw=<GatewayIPv4>`

Default gateway for IPv4 traffic.

`gw6=<GatewayIPv6>`

Default gateway for IPv6 traffic.

`hwaddr=<XX:XX:XX:XX:XX:XX>`

A common MAC address with the I/G (Individual/Group) bit not set.

`ip=<(IPv4/CIDR|dhcp|manual)>`

IPv4 address in CIDR format.

`ip6=<(IPv6/CIDR|auto|dhcp|manual)>`

IPv6 address in CIDR format.

`link_down=<boolean>`

Whether this interface should be disconnected (like pulling the plug).

`mtu=<integer> (64 - 65535)`
: Maximum transfer unit of the interface. (lxc.network.mtu)

`name=<string>`
: Name of the network device as seen from inside the container. (lxc.network.name)

`rate=<mbps>`
: Apply rate limiting to the interface

`tag=<integer> (1 - 4094)`
: VLAN tag for this interface.

`trunks=<vlanid[;vlanid...]>`
: VLAN ids to pass through the interface

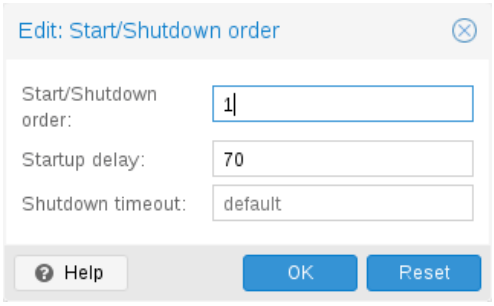
`type=<veth>`
: Network interface type.

Automatic Start and Shutdown of Containers

To automatically start a container when the host system boots, select the option *Start at boot* in the *Options* panel of the container in the web interface or run the following command:

```
# pct set CTID -onboot 1
```

Start and Shutdown Order



If you want to fine tune the boot order of your containers, you can use the following parameters:

- **Start/Shutdown order**: Defines the start order priority. For example, set it to 1 if you want the CT to be the first to be started. (We use the reverse startup order for shutdown, so a container with a start order of 1 would be the last to be shut down)
- **Startup delay**: Defines the interval between this container start and subsequent containers starts. For example,

set it to 240 if you want to wait 240 seconds before starting other containers.
- **Shutdown timeout**: Defines the duration in seconds Proxmox VE should wait for the container to be offline after issuing a shutdown command. By default this value is set to 60, which means that Proxmox VE will issue a shutdown request, wait 60s for the machine to be offline, and if after 60s the machine is still online will notify that the shutdown action failed.

Please note that containers without a Start/Shutdown order parameter will always start after those where the parameter is set, and this parameter only makes sense between the machines running locally on a host, and not cluster-wide.

If you require a delay between the host boot and the booting of the first container, see the section on Proxmox VE Node Management.

Hookscripts

You can add a hook script to CTs with the config property `hookscript`.

```
# pct set 100 -hookscript
local:snippets/hookscript.pl
```

It will be called during various phases of the guests lifetime. For an example and documentation see the example script under `/usr/share/pve-docs/examples/guest-example-hookscript.pl`.

Security Considerations

Containers use the kernel of the host system. This exposes an attack surface for malicious users. In general, full virtual machines provide better isolation. This should be considered if containers are provided to unknown or untrusted people.

To reduce the attack surface, LXC uses many security features like AppArmor, CGroups and kernel namespaces.

AppArmor

AppArmor profiles are used to restrict access to possibly dangerous actions. Some system calls, i.e. `mount`, are prohibited from execution.

To trace AppArmor activity, use:

```
# dmesg | grep apparmor
```

Although it is not recommended, AppArmor can be disabled for a container. This brings security risks with it. Some syscalls can lead to privilege escalation when executed within a container if the system is misconfigured or if a LXC or Linux Kernel vulnerability exists.

To disable AppArmor for a container, add the following line to the container configuration file located at `/etc/pve/lxc/CTID.conf`:

```
lxc.apparmor.profile = unconfined
```

Please note that this is not recommended for production use.

Control Groups (*cgroup*)

cgroup is a kernel mechanism used to hierarchically organize processes and distribute system resources.

The main resources controlled via *cgroups* are CPU time, memory and swap limits, and access to device nodes. *cgroups* are also used to "freeze" a container before taking snapshots.

There are 2 versions of *cgroups* currently available, legacy and *cgroupv2*.

Since Proxmox VE 7.0, the default is a pure *cgroupv2* environment. Previously a "hybrid" setup was used, where resource control was mainly done in *cgroupv1* with an additional *cgroupv2* controller which could take over some subsystems via the *cgroup_no_v1* kernel command-line parameter. (See the kernel parameter documentation for details.)

CGroup Version Compatibility

The main difference between pure *cgroupv2* and the old hybrid environments regarding Proxmox VE is that with *cgroupv2* memory and swap are now controlled independently. The memory and swap settings for containers can map directly to these values, whereas previously only the memory limit and the limit of the **sum** of memory and swap could be limited.

Another important difference is that the *devices* controller is configured in a completely different

way. Because of this, file system quotas are currently not supported in a pure *cgroupv2* environment.

cgroupv2 support by the container's OS is needed to run in a pure *cgroupv2* environment. Containers running *systemd* version 231 or newer support *cgroupv2* [1], as do containers not using *systemd* as init system [2].

> CentOS 7 and Ubuntu 16.10 are two prominent Linux distributions releases, which have a *systemd* version that is too old to run in a *cgroupv2* environment, you can either
>
> - Upgrade the whole distribution to a newer release. For the examples above, that could be Ubuntu 18.04 or 20.04, and CentOS 8 (or RHEL/CentOS derivatives like AlmaLinux or Rocky Linux). This has the benefit to get the newest bug and security fixes, often also new features, and moving the EOL date in the future.
> - Upgrade the Containers systemd version. If the distribution provides a backports repository this can be an easy and quick stop-gap measurement.
> - Move the container, or its services, to a Virtual Machine. Virtual Machines have a much less interaction with the host, that's why one can install decades old OS versions just fine there.
> - Switch back to the legacy *cgroup* controller. Note that while it can be a valid solution, it's not a permanent one. Starting from Proxmox VE 9.0, the legacy controller will not be supported anymore.

Changing CGroup Version

> If file system quotas are not required and all containers support *cgroupv2*, it is recommended to stick to the new default.

To switch back to the previous version the following kernel command-line parameter can be used:

```
systemd.unified_cgroup_hierarchy=0
```

See this section on editing the kernel boot command line on where to add the parameter.

Guest Operating System Configuration

Proxmox VE tries to detect the Linux distribution in the container, and modifies some files. Here is a short list of things done at container startup:

set /etc/hostname
: to set the container name

modify /etc/hosts
: to allow lookup of the local hostname

network setup
: pass the complete network setup to the container

configure DNS
: pass information about DNS servers

adapt the init system
: for example, fix the number of spawned getty processes

set the root password
: when creating a new container

rewrite ssh_host_keys
: so that each container has unique keys

randomize crontab
: so that cron does not start at the same time on all containers

Changes made by Proxmox VE are enclosed by comment markers:

```
# --- BEGIN PVE ---
<data>
# --- END PVE ---
```

Those markers will be inserted at a reasonable location in the file. If such a section already exists, it will be updated in place and will not be moved.

Modification of a file can be prevented by adding a `.pve-ignore.` file for it. For instance, if the file `/etc/.pve-ignore.hosts` exists then the `/etc/hosts` file will not be touched. This can be a simple empty file created via:

```
# touch /etc/.pve-ignore.hosts
```

Most modifications are OS dependent, so they differ between different distributions and versions. You can completely disable modifications by manually setting the `ostype` to `unmanaged`.

OS type detection is done by testing for certain files inside the container. Proxmox VE first checks the `/etc/os-release` file [3]. If that file is not present, or it does not contain a clearly recognizable distribution identifier the following distribution specific release files are checked.

Ubuntu
: inspect /etc/lsb-release (`DISTRIB_ID=Ubuntu`)

Debian
: test /etc/debian_version

Fedora
: test /etc/fedora-release

RedHat or CentOS
: test /etc/redhat-release

ArchLinux
: test /etc/arch-release

Alpine
: test /etc/alpine-release

Gentoo
: test /etc/gentoo-release

> Container start fails if the configured `ostype` differs from the auto detected type.

Container Storage

The Proxmox VE LXC container storage model is more flexible than traditional container storage

models. A container can have multiple mount points. This makes it possible to use the best suited storage for each application.

For example the root file system of the container can be on slow and cheap storage while the database can be on fast and distributed storage via a second mount point. See section Mount Points for further details.

Any storage type supported by the Proxmox VE storage library can be used. This means that containers can be stored on local (for example `lvm`, `zfs` or directory), shared external (like `iSCSI`, `NFS`) or even distributed storage systems like Ceph. Advanced storage features like snapshots or clones can be used if the underlying storage supports them. The `vzdump` backup tool can use snapshots to provide consistent container backups.

Furthermore, local devices or local directories can be mounted directly using *bind mounts*. This gives access to local resources inside a container with practically zero overhead. Bind mounts can be used as an easy way to share data between containers.

FUSE Mounts

> Because of existing issues in the Linux kernel's freezer subsystem the usage of FUSE mounts inside a container is strongly advised against, as containers need to be frozen for suspend or snapshot mode backups.

If FUSE mounts cannot be replaced by other mounting mechanisms or storage technologies, it is possible to establish the FUSE mount on the Proxmox host and use a bind mount point to make it accessible inside the container.

Using Quotas Inside Containers

Quotas allow to set limits inside a container for the amount of disk space that each user can use.

> This currently requires the use of legacy *cgroups*.

> This only works on ext4 image based storage types and currently

> only works with privileged containers.

Activating the `quota` option causes the following mount options to be used for a mount point: `usrjquota=aquota.user,grpjquota=aquota.group,jqfmt=vfsv0`

This allows quotas to be used like on any other system. You can initialize the `/aquota.user` and `/aquota.group` files by running:

```
# quotacheck -cmug /
# quotaon /
```

Then edit the quotas using the `edquota` command. Refer to the documentation of the distribution running inside the container for details.

> You need to run the above commands for every mount point by passing the mount point's path instead of just `/`.

Using ACLs Inside Containers

The standard Posix **A**ccess **C**ontrol **L**ists are also available inside containers. ACLs allow you to set more detailed file ownership than the traditional user/group/others model.

Backup of Container mount points

To include a mount point in backups, enable the `backup` option for it in the container configuration. For an existing mount point `mp0`

```
mp0: guests:subvol-100-disk-1,mp=/root/files,size=8G
```

add `backup=1` to enable it.

```
mp0: guests:subvol-100-disk-1,mp=/root/files,size=8G,backup=1
```

> When creating a new mount point in the GUI, this option is enabled by default.

To disable backups for a mount point, add `backup=0` in the way described above, or uncheck the **Backup** checkbox on the GUI.

Replication of Containers mount points

By default, additional mount points are replicated when the Root Disk is replicated. If you want the Proxmox VE storage replication mechanism to skip a mount point, you can set the **Skip replication** option for that mount point. As of Proxmox VE 5.0, replication requires a storage of type `zfspool`. Adding a mount point to a different type of storage when the container has replication configured requires to have **Skip replication** enabled for that mount point.

Backup and Restore

Container Backup

It is possible to use the `vzdump` tool for container backup. Please refer to the `vzdump` manual page for details.

Restoring Container Backups

Restoring container backups made with `vzdump` is possible using the `pct restore` command. By default, `pct restore` will attempt to restore as much of the backed up container configuration as possible. It is possible to override the backed up configuration by manually setting container options on the command line (see the `pct` manual page for details).

> `pvesm extractconfig` can be used to view the backed up configuration contained in a vzdump archive.

There are two basic restore modes, only differing by their handling of mount points:

“Simple” Restore Mode

If neither the `rootfs` parameter nor any of the optional `mpX` parameters are explicitly set, the

mount point configuration from the backed up configuration file is restored using the following steps:

1. Extract mount points and their options from backup
2. Create volumes for storage backed mount points on the storage provided with the `storage` parameter (default: `local`).
3. Extract files from backup archive
4. Add bind and device mount points to restored configuration (limited to root user)

> Since bind and device mount points are never backed up, no files are restored in the last step, but only the configuration options. The assumption is that such mount points are either backed up with another mechanism (e.g., NFS space that is bind mounted into many containers), or not intended to be backed up at all.

This simple mode is also used by the container restore operations in the web interface.

“Advanced” Restore Mode

By setting the `rootfs` parameter (and optionally, any combination of `mpX` parameters), the `pct restore` command is automatically switched into an advanced mode. This advanced mode completely ignores the `rootfs` and `mpX` configuration options contained in the backup archive, and instead only uses the options explicitly provided as parameters.

This mode allows flexible configuration of mount point settings at restore time, for example:

- Set target storages, volume sizes and other options for each mount point individually
- Redistribute backed up files according to new mount point scheme
- Restore to device and/or bind mount points (limited to root user)

Managing Containers with `pct`

The “Proxmox Container Toolkit” (`pct`) is the command-line tool to manage Proxmox VE containers. It enables you to create or destroy containers, as well as control the container execution (start, stop, reboot, migrate, etc.). It can be used to set parameters in the config file of a container, for example the network configuration or memory limits.

CLI Usage Examples

Create a container based on a Debian template (provided you have already downloaded the template via the web interface)

```
# pct create 100
/var/lib/vz/template/cache/debian
-10.0-standard_10.0-
1_amd64.tar.gz
```

Start container 100

```
# pct start 100
```

Start a login session via getty

```
# pct console 100
```

Enter the LXC namespace and run a shell as root user

```
# pct enter 100
```

Display the configuration

```
# pct config 100
```

Add a network interface called `eth0`, bridged to the host bridge `vmbr0`, set the address and gateway, while it’s running

```
# pct set 100 -net0
name=eth0,bridge=vmbr0,ip=192.168
.15.147/24,gw=192.168.15.1
```

Reduce the memory of the container to 512MB

```
# pct set 100 -memory 512
```

Destroying a container always removes it from Access Control Lists and it always removes the firewall configuration of the container. You have to activate *--purge*, if you want to additionally remove the container from replication jobs, backup jobs and HA resource configurations.

```
# pct destroy 100 --purge
```

Move a mount point volume to a different storage.

```
# pct move-volume 100 mp0 other-storage
```

Reassign a volume to a different CT. This will remove the volume `mp0` from the source CT and attaches it as `mp1` to the target CT. In the background the volume is being renamed so that the name matches the new owner.

```
#  pct move-volume 100 mp0 --target-vmid 200 --target-volume mp1
```

Obtaining Debugging Logs

In case `pct start` is unable to start a specific container, it might be helpful to collect debugging output by passing the `--debug` flag (replace `CTID` with the container's CTID):

```
# pct start CTID --debug
```

Alternatively, you can use the following `lxc-start` command, which will save the debug log to the file specified by the `-o` output option:

```
# lxc-start -n CTID -F -l DEBUG -o /tmp/lxc-CTID.log
```

This command will attempt to start the container in foreground mode, to stop the container run `pct shutdown CTID` or `pct stop CTID` in a second terminal.

The collected debug log is written to `/tmp/lxc-CTID.log`.

> If you have changed the container's configuration since the last start

> attempt with `pct start`, you need to run `pct start` at least once to also update the configuration used by `lxc-start`.

Migration

If you have a cluster, you can migrate your Containers with

```
# pct migrate <ctid> <target>
```

This works as long as your Container is offline. If it has local volumes or mount points defined, the migration will copy the content over the network to the target host if the same storage is defined there.

Running containers cannot live-migrated due to technical limitations. You can do a restart migration, which shuts down, moves and then starts a container again on the target node. As containers are very lightweight, this results normally only in a downtime of some hundreds of milliseconds.

A restart migration can be done through the web interface or by using the `--restart` flag with the `pct migrate` command.

A restart migration will shut down the Container and kill it after the specified timeout (the default is 180 seconds). Then it will migrate the Container like an offline migration and when finished, it starts the Container on the target node.

Configuration

The `/etc/pve/lxc/<CTID>.conf` file stores container configuration, where `<CTID>` is the numeric ID of the given container. Like all other files stored inside `/etc/pve/`, they get automatically replicated to all other cluster nodes.

> CTIDs < 100 are reserved for internal purposes, and CTIDs need to be unique cluster wide.

Example Container Configuration

```
ostype: debian
arch: amd64
hostname: www
memory: 512
swap: 512
net0:
bridge=vmbr0,hwaddr=66:64:66:64:6
4:36,ip=dhcp,name=eth0,type=veth
rootfs: local:107/vm-107-disk-
1.raw,size=7G
```

The configuration files are simple text files. You can edit them using a normal text editor, for example, `vi` or `nano`. This is sometimes useful to do small corrections, but keep in mind that you need to restart the container to apply such changes.

For that reason, it is usually better to use the `pct` command to generate and modify those files, or do the whole thing using the GUI. Our toolkit is smart enough to instantaneously apply most changes to running containers. This feature is called “hot plug”, and there is no need to restart the container in that case.

In cases where a change cannot be hot-plugged, it will be registered as a pending change (shown in red color in the GUI). They will only be applied after rebooting the container.

File Format

The container configuration file uses a simple colon separated key/value format. Each line has the following format:

```
# this is a comment
OPTION: value
```

Blank lines in those files are ignored, and lines starting with a `#` character are treated as comments and are also ignored.

It is possible to add low-level, LXC style configuration directly, for example:

```
lxc.init_cmd: /sbin/my_own_init
```

or

```
lxc.init_cmd = /sbin/my_own_init
```

The settings are passed directly to the LXC low-level tools.

Snapshots

When you create a snapshot, `pct` stores the configuration at snapshot time into a separate snapshot section within the same configuration file. For example, after creating a snapshot called “testsnapshot”, your configuration file will look like this:

Container configuration with snapshot

```
memory: 512
swap: 512
parent: testsnaphot
...

[testsnaphot]
memory: 512
swap: 512
snaptime: 1457170803
...
```

There are a few snapshot related properties like `parent` and `snaptime`. The `parent` property is used to store the parent/child relationship between snapshots. `snaptime` is the snapshot creation time stamp (Unix epoch).

Options

`arch`: `<amd64 | arm64 | armhf | i386 | riscv32 | riscv64>` (*default* = `amd64`)

OS architecture type.

`cmode`: `<console | shell | tty>` (*default* = `tty`)

Console mode. By default, the console command tries to open a connection to one of the available tty devices. By setting cmode to *console* it tries to attach to /dev/console instead. If you set cmode to *shell*, it simply invokes a shell inside the container (no login).

`console`: `<boolean>` (*default* = `1`)

Attach a console device (/dev/console) to the container.

`cores`: `<integer> (1 - 8192)`

The number of cores assigned to the container. A container can use all available cores by default.

`cpulimit`: `<number> (0 - 8192)` (*default* = `0`)

Limit of CPU usage.

> If the computer has 2 CPUs, it has a total of **2** CPU time. Value **0** indicates no CPU limit.

`cpuunits`: `<integer> (0 - 500000)` (*default* = `cgroup v1: 1024, cgroup v2: 100`)

CPU weight for a container. Argument is used in the kernel fair scheduler. The larger the number is, the more CPU time this container gets. Number is relative to the weights of all the other running guests.

`debug`: `<boolean>` (*default* = `0`)

Try to be more verbose. For now this only enables debug log-level on start.

`description`: `<string>`

Description for the Container. Shown in the web-interface CT's summary. This is saved as comment inside the configuration file.

`dev[n]`: `[[path=]<Path>] [,gid=<integer>] [,mode=<Octal access mode>] [,uid=<integer>]`

Device to pass through to the container

`gid=<integer> (0 - N)`

Group ID to be assigned to the device node

`mode=<Octal access mode>`

Access mode to be set on the device node

`path=<Path>`

Path to the device to pass through to the container

`uid=<integer> (0 - N)`

User ID to be assigned to the device node

`features`: `[force_rw_sys=<1|0>] [,fuse=<1|0>] [,keyctl=<1|0>] [,mknod=<1|0>] [,mount=<fstype;fstype;...>] [,nesting=<1|0>]`

Allow containers access to advanced features.

`force_rw_sys=<boolean>` (*default* = `0`)

Mount /sys in unprivileged containers as `rw` instead of `mixed`. This can break networking under newer (>= v245) systemd-network use.

`fuse=<boolean>` (*default* = `0`)
: Allow using *fuse* file systems in a container. Note that interactions between fuse and the freezer cgroup can potentially cause I/O deadlocks.

`keyctl=<boolean>` (*default* = `0`)
: For unprivileged containers only: Allow the use of the keyctl() system call. This is required to use docker inside a container. By default unprivileged containers will see this system call as non-existent. This is mostly a workaround for systemd-networkd, as it will treat it as a fatal error when some keyctl() operations are denied by the kernel due to lacking permissions. Essentially, you can choose between running systemd-networkd or docker.

`mknod=<boolean>` (*default* = `0`)
: Allow unprivileged containers to use mknod() to add certain device nodes. This requires a kernel with seccomp trap to user space support (5.3 or newer). This is experimental.

`mount=<fstype;fstype;...>`
: Allow mounting file systems of specific types. This should be a list of file system types as used with the mount command. Note that this can have negative effects on the container's security. With access to a loop device, mounting a file can circumvent the mknod permission of the devices cgroup, mounting an NFS file system can block the host's I/O completely and prevent it from rebooting, etc.

`nesting=<boolean>` (*default* = `0`)
: Allow nesting. Best used with unprivileged containers with additional id mapping. Note that this will expose procfs and sysfs contents of the host to the guest.

`hookscript`: `<string>`
: Script that will be exectued during various steps in the containers lifetime.

`hostname: <string>`
: Set a host name for the container.

`lock: <backup | create | destroyed | disk | fstrim | migrate | mounted | rollback | snapshot | snapshot-delete>`
: Lock/unlock the container.

`memory: <integer> (16 - N)` (*default* = `512`)
: Amount of RAM for the container in MB.

`mp[n]: [volume=]<volume> ,mp=<Path> [,acl=<1|0>] [,backup=<1|0>] [,mountoptions=<opt[;opt...]>] [,quota=<1|0>] [,replicate=<1|0>] [,ro=<1|0>] [,shared=<1|0>] [,size=<DiskSize>]`
: Use volume as container mount point. Use the special syntax STORAGE_ID:SIZE_IN_GiB to allocate a new volume.

  `acl=<boolean>`
  : Explicitly enable or disable ACL support.

  `backup=<boolean>`
  : Whether to include the mount point in backups (only used for volume mount points).

  `mountoptions=<opt[;opt...]>`
  : Extra mount options for rootfs/mps.

  `mp=<Path>`
  : Path to the mount point as seen from inside the container.

  > Must not contain any symlinks for security reasons.

  `quota=<boolean>`
  : Enable user quotas inside the container (not supported with zfs subvolumes)

  `replicate=<boolean>` (*default* = `1`)
  : Will include this volume to a storage replica job.

  `ro=<boolean>`
  : Read-only mount point

  `shared=<boolean>` (*default* = `0`)

Mark this non-volume mount point as available on all nodes.

> This option does not share the mount point automatically, it assumes it is shared already!

`size=<DiskSize>`
: Volume size (read only value).

`volume=<volume>`
: Volume, device or directory to mount into the container.

`nameserver: <string>`
: Sets DNS server IP address for a container. Create will automatically use the setting from the host if you neither set searchdomain nor nameserver.

`net[n]: name=<string> [,bridge=<bridge>] [,firewall=<1|0>] [,gw=<GatewayIPv4>] [,gw6=<GatewayIPv6>] [,hwaddr=<XX:XX:XX:XX:XX:XX>] [,ip=<(IPv4/CIDR|dhcp|manual)>] [,ip6=<(IPv6/CIDR|auto|dhcp|manual)>] [,link_down=<1|0>] [,mtu=<integer>] [,rate=<mbps>] [,tag=<integer>] [,trunks=<vlanid[;vlanid...]>] [,type=<veth>]`
: Specifies network interfaces for the container.

`bridge=<bridge>`
: Bridge to attach the network device to.

`firewall=<boolean>`
: Controls whether this interface's firewall rules should be used.

`gw=<GatewayIPv4>`
: Default gateway for IPv4 traffic.

`gw6=<GatewayIPv6>`
: Default gateway for IPv6 traffic.

`hwaddr=<XX:XX:XX:XX:XX:XX>`
: A common MAC address with the I/G (Individual/Group) bit not set.

`ip=<(IPv4/CIDR|dhcp|manual)>`
: IPv4 address in CIDR format.

`ip6=<(IPv6/CIDR|auto|dhcp|manual)>`
: IPv6 address in CIDR format.

link_down=<boolean>
: Whether this interface should be disconnected (like pulling the plug).

mtu=<integer> (64 - 65535)
: Maximum transfer unit of the interface. (lxc.network.mtu)

name=<string>
: Name of the network device as seen from inside the container. (lxc.network.name)

rate=<mbps>
: Apply rate limiting to the interface

tag=<integer> (1 - 4094)
: VLAN tag for this interface.

trunks=<vlanid[;vlanid...]>
: VLAN ids to pass through the interface

type=<veth>
: Network interface type.

onboot: <boolean> (*default* = 0)
: Specifies whether a container will be started during system bootup.

ostype: <alpine | archlinux | centos | debian | devuan | fedora | gentoo | nixos | opensuse | ubuntu | unmanaged>
: OS type. This is used to setup configuration inside the container, and corresponds to lxc setup scripts in /usr/share/lxc/config/<ostype>.common.conf. Value *unmanaged* can be used to skip and OS specific setup.

protection: <boolean> (*default* = 0)
: Sets the protection flag of the container. This will prevent the CT or CT's disk remove/update operation.

rootfs: [volume=]<volume> [,acl=<1|0>] [,mountoptions=<opt[;opt...]>] [,quota=<1|0>] [,replicate=<1|0>] [,ro=<1|0>] [,shared=<1|0>] [,size=<DiskSize>]
: Use volume as container root.

acl=<boolean>
: Explicitly enable or disable ACL support.

mountoptions=<opt[;opt...]>
: Extra mount options for rootfs/mps.

quota=<boolean>

Enable user quotas inside the container (not supported with zfs subvolumes)

`replicate=<boolean>` (*default* = `1`)
: Will include this volume to a storage replica job.

`ro=<boolean>`
: Read-only mount point

`shared=<boolean>` (*default* = `0`)
: Mark this non-volume mount point as available on all nodes.

> This option does not share the mount point automatically, it assumes it is shared already!

`size=<DiskSize>`
: Volume size (read only value).

`volume=<volume>`
: Volume, device or directory to mount into the container.

`searchdomain`: `<string>`
: Sets DNS search domains for a container. Create will automatically use the setting from the host if you neither set searchdomain nor nameserver.

`startup`: `[[order=]\d+] [,up=\d+] [,down=\d+] `
: Startup and shutdown behavior. Order is a non-negative number defining the general startup order. Shutdown in done with reverse ordering. Additionally you can set the *up* or *down* delay in seconds, which specifies a delay to wait before the next VM is started or stopped.

`swap`: `<integer> (0 - N)` (*default* = `512`)
: Amount of SWAP for the container in MB.

`tags`: `<string>`
: Tags of the Container. This is only meta information.

`template`: `<boolean>` (*default* = `0`)
: Enable/disable Template.

`timezone`: `<string>`
: Time zone to use in the container. If option isn’t set, then nothing will be done. Can be set to *host* to match the host time zone, or an arbitrary time zone option from /usr/share/zoneinfo/zone.tab

`tty`: `<integer> (0 - 6)` (*default* = `2`)

Specify the number of tty available to the container

`unprivileged`: `<boolean>` (*default* = `0`)

Makes the container run as unprivileged user. (Should not be modified manually.)

`unused[n]`: `[volume=]<volume>`

Reference to unused volumes. This is used internally, and should not be modified manually.

`volume=<volume>`

The volume that is not used currently.

Locks

Container migrations, snapshots and backups (`vzdump`) set a lock to prevent incompatible concurrent actions on the affected container. Sometimes you need to remove such a lock manually (e.g., after a power failure).

```
# pct unlock <CTID>
```

Only do this if you are sure the action which set the lock is no longer running.

1. this includes all newest major versions of container templates shipped by Proxmox VE

2. for example Alpine Linux

3. /etc/os-release replaces the multitude of per-distribution release files https://manpages.debian.org/stable/systemd/os-release.5.en.html

Version 8.2.2
Last updated Thu Apr 25 09:24:16 CEST 2024