

# Virtualization Tools in Linux

LinuXpert Systems

WhatsApp: 98841 65649, 90808 01382

E-mail: baskar@linuxpert.in

## TABLE OF CONTENTS


1. virsh ..... 1

2. virt-install ..... 11

3. virt-manager ..... 18

4. qemu-img ..... 24

5. VBoxManage ..... 31

6. docker ..... 38

7. podman ..... 46

8. lxc ..... 55

9. qm ..... 62

10. pct ..... 68

# 1. virsh

**`virsh`** is a **command-line management tool** used to control and manage **virtual machines (VMs)** running under the **libvirt** virtualization framework.

It allows administrators to create, start, stop, pause, monitor, and configure virtual machines from the terminal.

**`virsh`** is commonly used with:

- **KVM/QEMU** (most common)
- Xen
- LXC
- VMware (through libvirt drivers)
- Other hypervisors supported by libvirt

---

### Syntax

```
virsh [OPTIONS] COMMAND [ARGUMENTS]
```

Example:

```
virsh list
```

### Purpose

**`virsh`** is used to:

- Manage virtual machines
- Create and delete VMs
- Start and stop VMs
- View VM status
- Configure VM hardware
- Manage virtual networks
- Manage storage pools and volumes
- Create VM snapshots
- Connect to remote hypervisors

---

## Basic Examples

### 1. List running virtual machines

```
virsh list
```

Example output:

```
 Id   Name          State
-------------------------------
 1    ubuntu22      running
 2    centos9       running
```

Shows only running VMs.

## 2. List all virtual machines

```
virsh list --all
```

## Example:

```
 Id   Name        State
--------------------------------
 1    ubuntu22    running
 -    rocky9      shut off
 -    fedora40    shut off
```

---

## 3. Start a virtual machine

```
virsh start ubuntu22
```

**Output:**

```
Domain 'ubuntu22' started
```

---

## 4. Shut down a VM gracefully

```
virsh shutdown ubuntu22
```

This sends an ACPI shutdown signal (similar to pressing the power button on a physical computer).

---

## 5. Force power off a VM

```
virsh destroy ubuntu22
```

Although the command says **destroy**, it **does not delete** the VM.

It simply performs a hard power-off (similar to unplugging the power cable).

## 6. Reboot a VM

```
virsh reboot ubuntu22
```

---

## 7. Suspend a VM

```
virsh suspend ubuntu22
```

Pauses execution while preserving the VM's state in memory.

**Resume it with:**

```
virsh resume ubuntu22
```

---

## 8. View VM information

```
virsh dominfo ubuntu22
```

**Example:**

```
Id:             1
Name:           ubuntu22
UUID:           ...
OS Type:        hvm
State:          running
CPU(s):         2
Max memory:     4194304 KiB
Used memory:    4194304 KiB
```

### 9. Display VM XML configuration

```
virsh dumpxml ubuntu22
```

Shows the VM's XML definition, including:

- CPU
- RAM
- Disk devices
- Network interfaces
- Graphics
- Boot order

---

### 10. Edit VM configuration

```
virsh edit ubuntu22
```

Opens the XML configuration in the default text editor.

---

## Creating and Managing VMs

### Define a VM from an XML file

```
virsh define vm.xml
```

Registers a new VM.

---

### Remove a VM definition

```
virsh undefine ubuntu22
```

Removes the VM definition but typically leaves its disk image intact.

## Snapshots

**Create a snapshot:**

```
virsh snapshot-create-as ubuntu22 before_update
```

**List snapshots:**

```
virsh snapshot-list ubuntu22
```

**Restore snapshot:**

```
virsh snapshot-revert ubuntu22 before_update
```

**Delete snapshot:**

```
virsh snapshot-delete ubuntu22 before_update
```

---

## Storage Management

**List storage pools:**

```
virsh pool-list
```

**List all pools:**

```
virsh pool-list --all
```

**List storage volumes:**

```
virsh vol-list default
```

---

## Network Management

**List virtual networks:**

```
virsh net-list --all
```

**Start network:**

```
virsh net-start default
```

**Stop network:**

```
virsh net-destroy default
```

**Enable autostart:**

```
virsh net-autostart default
```

---

## Console Access

**Connect to the VM console:**

```
virsh console ubuntu22
```

**Exit the console with:**

```
Ctrl + ]
```

---

## Autostart a VM

**Enable:**

```
virsh autostart ubuntu22
```

**Disable:**

```
virsh autostart --disable ubuntu22
```

---

## VM Statistics

**CPU statistics:**

```
virsh cpu-stats ubuntu22
```

**Memory statistics:**

```
virsh dommemstat ubuntu22
```

**Block device statistics:**

```
virsh domblkstat ubuntu22 vda
```

**Interface statistics:**

```
virsh domifstat ubuntu22 vnet0
```

---

## Live Migration

**Move a running VM to another host:**

```
virsh migrate --live ubuntu22 qemu+ssh://server/system
```

---

## Common `virsh` Commands

| Command | Description |
|---|---|
| `virsh list` | List running VMs |
| `virsh list --all` | List all VMs |
| `virsh start vm` | Start a VM |
| `virsh shutdown vm` | Gracefully shut down a VM |
| `virsh destroy vm` | Force power off a VM |
| `virsh reboot vm` | Reboot a VM |
| `virsh suspend vm` | Pause a VM |
| `virsh resume vm` | Resume a paused VM |
| `virsh dominfo vm` | Show VM information |
| `virsh dumpxml vm` | Display VM XML configuration |
| `virsh edit vm` | Edit VM configuration |
| `virsh define file.xml` | Register a VM from an XML file |
| `virsh undefine vm` | Remove a VM definition |
| `virsh console vm` | Connect to the VM's console |
| `virsh snapshot-list vm` | List snapshots |
| `virsh pool-list` | List storage pools |
| `virsh net-list` | List virtual networks |

## Advantages

- Lightweight command-line interface
- Scriptable for automation
- Supports remote hypervisor management
- Works with multiple virtualization technologies
- Powerful VM lifecycle management
- Ideal for servers without a graphical interface

---

## Typical Workflow

```
# List all VMs
virsh list –all
```

---

```
# Start a VM
virsh start ubuntu22
```

---

```
# Check its status
virsh dominfo ubuntu22
```

---

```
# Connect to the console
virsh console ubuntu22
```

---

```
# Shut it down gracefully
virsh shutdown ubuntu22
```

## Summary

- **Command:** `virsh`
- **Meaning:** Virtualization Shell
- **Primary use:** Manage virtual machines through the **libvirt** framework.
- **Common hypervisor:** KVM / QEMU.
- **Key capabilities:** VM lifecycle management, snapshots, storage and network management, XML configuration, statistics, console access, and live migration.

**`virsh`** is one of the most important command-line tools for Linux virtualization administrators and is widely used in enterprise KVM/libvirt environments.

# 2.virt-install

**`virt-install`** is a command-line tool used to **create and install virtual machines (VMs)** using the **KVM (Kernel-based Virtual Machine)** hypervisor and the **libvirt** virtualization framework.

It automates the process of creating a VM, allocating CPU, RAM, storage, networking, and installing an operating system.

## Syntax

```
virt-install [OPTIONS]
```

---

## Purpose

- Create a new virtual machine
- Install Linux or Windows inside the VM
- Configure virtual hardware
- Connect the VM to virtual networks
- Perform unattended installations
- Import existing virtual machine images

## Common Options

| Option | Description |
|---|---|
| `--name` | VM name |
| `--memory` | RAM size (MB) |
| `--vcpus` | Number of virtual CPUs |
| `--disk` | Virtual disk |
| `--os-variant` | Guest OS type |
| `--cdrom` | Install from ISO image |
| `--location` | Install from network repository |
| `--network` | Network configuration |
| `--graphics` | Graphics type (VNC/SPICE/none) |
| `--import` | Import an existing disk image |
| `--boot` | Boot configuration |
| `--console` | Configure text console |
| `--noautoconsole` | Don't automatically connect after creation |

## Example 1: Install Ubuntu from an ISO

```
virt-install \
--name ubuntu22 \
--memory 4096 \
--vcpus 2 \
--disk size=20 \
--cdrom /iso/ubuntu-22.04.iso \
--os-variant ubuntu22.04 \
--network default \
--graphics spice
```

**This command:**

- Creates a VM named **ubuntu22**
- Allocates **4 GB RAM**
- Uses **2 virtual CPUs**
- Creates a **20 GB virtual disk**
- Boots from the Ubuntu ISO
- Connects to the default virtual network
- Uses SPICE graphics

---

## Example 2: Create a Minimal Server (Console Only)

```
virt-install \
--name rocky9 \
--memory 2048 \
--vcpus 2 \
--disk size=15 \
--location http://mirror.example.com/rocky/9/BaseOS/x86_64/os/ \
--graphics none \
--console pty,target_type=serial
```

The installation runs entirely in the terminal through a serial console.

### Example 3: Import an Existing QCOW2 Image

```
virt-install \
--name debian \
--memory 4096 \
--vcpus 2 \
--disk /var/lib/libvirt/images/debian.qcow2 \
--import
```

Instead of installing a new OS, this imports an existing virtual disk.

---

### Example 4: Create a Windows VM

```
virt-install \
--name windows11 \
--memory 8192 \
--vcpus 4 \
--disk size=80 \
--cdrom Win11.iso \
--os-variant win11 \
--network default
```

Creates a Windows 11 virtual machine using the installation ISO.

---

### Example 5: Automatically Create a Disk

```
virt-install \
--name testvm \
--memory 2048 \
--disk size=10 \
--cdrom centos.iso
```

Creates a new 10 GB virtual disk automatically.

### Example 6: Specify an Existing Disk

```
virt-install \
--disk path=/var/lib/libvirt/images/test.qcow2
```

Uses the specified disk image instead of creating a new one.

---

## Virtual Disk Formats

Common formats include:

- `qcow2` (supports snapshots and thin provisioning)
- `raw` (best performance, larger size)
- `vmdk` (VMware)
- `vhd`
- `vdi` (VirtualBox)

**Example:**

```
--disk path=/vm/rhel.qcow2,size=30,format=qcow2
```

---

## Network Options

Use the default NAT network:

```
--network network=default
```

Use a bridge:

```
--network bridge=br0
```

Disconnect networking:

```
--network none
```

## Graphics Options

**SPICE:**

```
--graphics spice
```

**VNC:**

```
--graphics vnc
```

**No graphical display:**

```
--graphics none
```

---

## OS Variant

Specifying the OS variant allows libvirt to optimise virtual hardware.

**Examples:**

```
--os-variant ubuntu22.04
```

```
--os-variant rhel9
```

```
--os-variant win11
```

**To list supported variants:**

```
osinfo-query os
```

---

## Boot Options

Boot from CD-ROM:

```
--boot cdrom
```

Boot from the virtual hard disk:

```
--boot hd
```

Enable UEFI firmware (if supported):

```
--boot uefi
```

## Common Workflow

1. Install virtualization packages.
2. Start the libvirt service.
3. Create a virtual disk.
4. Run `virt-install`.
5. Install the guest operating system.
6. Manage the VM with tools such as `virsh` or graphical managers like Virtual Machine Manager (`virt-manager`).

---

## Verify Existing VMs

**List virtual machines:**

```
virsh list --all
```

**Start a VM:**

```
virsh start ubuntu22
```

**Stop a VM:**

```
virsh shutdown ubuntu22
```

## Common Errors

### "No KVM support"

Hardware virtualization (Intel VT-x or AMD-V) may be disabled in the BIOS/UEFI, or your CPU may not support it.

---

### "Cannot connect to libvirt"

The `libvirtd` or `virtqemud` service may not be running, or your user may not have permission to access the libvirt socket.

---

### "Storage pool not found"

The default storage pool has not been created or activated.

---

### "Permission denied"

The specified ISO or disk image may not be readable by the libvirt/QEMU process.

---

### Summary

- **Command:** `virt-install`
- **Purpose:** Create and install KVM/libvirt virtual machines from the command line.
- **Requires:** KVM, QEMU, and libvirt.
- **Common uses:** Install Linux or Windows guests, create virtual disks, import existing VM images, configure networking, and automate VM deployments.

# 3. virt-manager

**`virt-manager`** (Virtual Machine Manager) is a **graphical application** used to create, manage, and monitor **virtual machines (VMs)** on Linux systems. It provides a user-friendly interface for the **libvirt** virtualization framework.

Unlike commands such as **`virsh`**, **`virt-manager`** opens a GUI instead of working only in the terminal.

---

## Syntax

```
virt-manager [OPTIONS]
```

---

## What it Does

**`virt-manager`** allows you to:

- Create new virtual machines
- Start, stop, pause, and restart VMs
- Install guest operating systems
- Allocate CPU, RAM, and storage
- Configure virtual networks
- Create snapshots (if supported)
- Connect to local or remote hypervisors
- View the VM's console

---

## Common Features

- Supports **KVM/QEMU**
- Supports **Xen**
- Supports **LXC**
- Supports remote virtualization hosts over SSH
- Uses **libvirt** as the backend

# Basic Usage

## 1. Launch Virtual Machine Manager

```
virt-manager
```

Opens the graphical Virtual Machine Manager window.

---

## 2. Connect to the Local Hypervisor

Normally happens automatically.

```
QEMU/KVM
    localhost (Running)
```

---

## 3. Create a New Virtual Machine

Run:

```
virt-manager
```

Then click

`File → New Virtual Machine`

The wizard lets you install from:

- ISO image
- Physical CD/DVD
- Network install (PXE)
- Existing disk image
- Import an existing VM

### 4. Connect to a Remote Host

**Example:**

```
File
    Add Connection
```

**Choose:**

```
QEMU/KVM

Connect to remote host over SSH
```

**Enter:**

```
hostname
username
```

Now you can manage VMs on another Linux server.

---

## Useful Options

### Open a specific connection

```
virt-manager --connect qemu:///system
```

Uses the system-wide QEMU/KVM connection.

---

### Connect to a remote server

```
virt-manager --connect qemu+ssh://user@server/system
```

**Example:**

```
virt-manager --connect qemu+ssh://admin@192.168.1.20/system
```

---

### Enable debugging

```
virt-manager --debug
```

Shows detailed debugging information in the terminal.

### Show version

```
virt-manager --version
```

Example output:

```
4.1.0
```

---

## Installing `virt-manager`

### Ubuntu/Debian

```
sudo apt install virt-manager
```

### Fedora

```
sudo dnf install virt-manager
```

### RHEL/CentOS

```
sudo dnf install virt-manager
```

### openSUSE

```
sudo zypper install virt-manager
```

### Typical Workflow

```
Start virt-manager
        ↓
Connect to libvirt
        ↓
Create a virtual machine
        ↓
Install an operating system
        ↓
Start the VM
        ↓
Manage resources
        ↓
Take snapshots (if supported)
        ↓
Shutdown or restart the VM
```

## Common Companion Commands

| Command | Purpose |
|---|---|
| `virsh` | Command-line VM management |
| `virt-install` | Create VMs from the terminal |
| `virt-viewer` | Open a VM's display console |
| `virt-clone` | Clone an existing VM |
| `virt-sysprep` | Prepare a VM image for cloning |
| `qemu-system-x86_64` | Launch a QEMU virtual machine directly |
| `systemctl status libvirtd` | Check the libvirt service status (on systems using libvirtd) |

## Example

**Launch the graphical manager:**

```
virt-manager
```

**Connect directly to the system hypervisor:**

```
virt-manager --connect qemu:///system
```

**Connect to a remote KVM host over SSH:**

```
virt-manager --connect qemu+ssh://admin@192.168.1.100/system
```

---

## Advantages

- Easy graphical interface
- Beginner-friendly
- Supports multiple virtualization technologies
- Remote VM management
- Resource monitoring
- VM snapshots (when supported)
- Console access without extra configuration

## Limitations

- Requires a graphical desktop environment
- Less convenient for automation than command-line tools
- Depends on the `libvirt` service and supported hypervisors

---

## Summary

**`virt-manager`** is a GUI-based virtualization management tool for Linux. It simplifies creating, configuring, and managing virtual machines using the **`libvirt`** framework. It is particularly useful for users who prefer a graphical interface over command-line tools like `virsh`, while still providing access to advanced virtualization features.

# 4. qemu-img

**`qemu-img`** is a command-line utility used to **create, inspect, convert, resize, and manage virtual machine disk images**. It is part of the **QEMU** virtualization package and is also widely used with **KVM**, **libvirt**, and virtualization platforms such as **OpenStack**.

Unlike **`qemu-system-x86_64`**, which runs virtual machines, **`qemu-img`** only manages the virtual disk files.

---

## Syntax

```
qemu-img [options] command [command-options]
```

---

## Common Uses

- Create virtual hard disks
- Convert disk image formats
- Resize virtual disks
- Display disk information
- Create snapshots
- Check image integrity
- Compress disk images

## Common Disk Image Formats

| Format | Description |
|---|---|
| `qcow2` | QEMU Copy-On-Write format (most popular) |
| `raw` | Raw disk image (fastest) |
| `vmdk` | VMware disk image |
| `vdi` | VirtualBox disk image |
| `vhd` | Microsoft Hyper-V image |
| `vpc` | Virtual PC image |
| `qed` | Older QEMU format (deprecated) |

# Frequently Used Commands

## 1. Create a Virtual Disk

```
qemu-img create -f qcow2 disk.qcow2 20G
```

Creates a 20 GB virtual disk.

**Options**

- `create` → Create new image
- `-f qcow2` → Image format
- `20G` → Virtual size

**Output:**

```
Formatting 'disk.qcow2', fmt=qcow2 size=21474836480
```

---

## 2. Create a Raw Disk

```
qemu-img create -f raw disk.img 10G
```

Creates a raw 10 GB image.

---

## 3. Display Image Information

```
qemu-img info disk.qcow2
```

Example:

```
image: disk.qcow2
file format: qcow2
virtual size: 20 GiB
disk size: 3.5 GiB
cluster_size: 65536
```

**Notice:**

- Virtual Size = maximum capacity
- Disk Size = actual space used

## 4. Convert Between Formats

**QCOW2 → RAW**

```
qemu-img convert -f qcow2 -O raw disk.qcow2 disk.img
```

**RAW → QCOW2**

```
qemu-img convert -f raw -O qcow2 disk.img disk.qcow2
```

**VMware VMDK → QCOW2**

```
qemu-img convert -f vmdk -O qcow2 vm.vmdk vm.qcow2
```

---

## 5. Resize a Disk

**Increase to 40 GB**

```
qemu-img resize disk.qcow2 40G
```

**Increase by 10 GB**

```
qemu-img resize disk.qcow2 +10G
```

**Shrink (not generally recommended)**

```
qemu-img resize disk.qcow2 15G
```

After resizing, the guest operating system's partition and filesystem must also be expanded (or shrunk) to use the new size.

---

## 6. Check Disk Image

```
qemu-img check disk.qcow2
```

Checks for corruption.

## 7. Measure Required Storage

```
qemu-img measure -O qcow2 disk.img
```

Shows estimated storage requirements.

---

## 8. Create a Snapshot

```
qemu-img snapshot -c before-update disk.qcow2
```

**List snapshots**

```
qemu-img snapshot -l disk.qcow2
```

**Delete snapshot**

```
qemu-img snapshot -d before-update disk.qcow2
```

---

## 9. Commit Changes

```
qemu-img commit overlay.qcow2
```

Writes changes from an overlay image back into its backing image.

---

## 10. Rebase an Image

```
qemu-img rebase -b base.qcow2 overlay.qcow2
```

Changes the backing file for a copy-on-write image.

---

## 11. Compare Two Images

```
qemu-img compare disk1.qcow2 disk2.qcow2
```

Checks whether the contents of two disk images are identical.

### 12. Map Image Allocation

```
qemu-img map disk.qcow2
```

Shows which parts of the virtual disk are allocated versus sparse.

---

### 13. Amend Image Options

```
qemu-img amend disk.qcow2
```

Changes certain image metadata (for example, supported QCOW2 features) without recreating the image.

---

### Useful Options

| Option | Meaning |
|---|---|
| `-f` | Input format |
| `-O` | Output format |
| `-o` | Image creation options |
| `-c` | Compress output (during conversion) |
| `-p` | Show progress |
| `-q` | Quiet mode |
| `-U` | Ignore image locks (use with caution) |

## Typical Workflow

Create a virtual disk

```
qemu-img create -f qcow2 ubuntu.qcow2 30G
↓
Install an operating system into the virtual machine using this disk.
↓
Check disk usage
```

```
qemu-img info ubuntu.qcow2
↓
Convert for Vmware
```

```
qemu-img convert -f qcow2 -O vmdk ubuntu.qcow2 ubuntu.vmdk
```

## Real-World Examples

### Create a 50 GB Ubuntu virtual disk

```
qemu-img create -f qcow2 ubuntu22.qcow2 50G
```

---

### Convert a VMware image to QCOW2

```
qemu-img convert -f vmdk -O qcow2 server.vmdk server.qcow2
```

---

### Increase a VM disk by 20 GB

```
qemu-img resize ubuntu22.qcow2 +20G
```

---

### Check image information

```
qemu-img info ubuntu22.qcow2
```

---

### Advantages of QCOW2

- Supports snapshots
- Thin provisioning (grows as data is written)
- Compression
- Encryption (older feature; modern deployments often use other encryption methods)
- Backing files for efficient cloning

---

### Advantages of RAW Images

- Maximum performance
- Simple format
- Supported by many virtualization platforms
- Easy to mount and inspect

## Summary

| Command | Purpose |
|---|---|
| `qemu-img create` | Create a new disk image |
| `qemu-img info` | Show image details |
| `qemu-img convert` | Convert image formats |
| `qemu-img resize` | Change virtual disk size |
| `qemu-img check` | Verify image integrity |
| `qemu-img snapshot` | Manage snapshots |
| `qemu-img compare` | Compare two disk images |
| `qemu-img map` | Display allocated regions |
| `qemu-img commit` | Merge overlay into backing image |
| `qemu-img rebase` | Change the backing file |
| `qemu-img measure` | Estimate storage requirements |
| `qemu-img amend` | Modify image metadata |

**`qemu-img`** is an essential utility for managing virtual disk images in QEMU/KVM environments, allowing administrators to create, inspect, convert, resize, and maintain virtual machine storage without starting the virtual machine.

# 5. VBoxManage

**`VBoxManage`** is the **command-line management tool** for **Oracle VirtualBox**. It allows you to create, configure, start, stop, and manage virtual machines (VMs) without using the VirtualBox graphical interface.

It is especially useful for:

- Automation using shell scripts
- Remote VM management
- Server environments without a GUI
- Advanced VirtualBox configuration

---

### Syntax

```
VBoxManage <command> [options]
```

**Example:**

```
VBoxManage list vms
```

---

## Common Operations

### 1. List all virtual machines

```
VBoxManage list vms
```

**Example output**

```
"Ubuntu22" {1f23ab45-6789-1234-5678-abcdef123456}
"Windows11" {4b12cd34-9876-4321-abcd-1234567890ef}
```

Lists all registered VMs.

## 2. List running virtual machines

```
VBoxManage list runningvms
```

Shows only VMs that are currently powered on.

---

## 3. Display VM information

```
VBoxManage showvminfo Ubuntu22
```

Displays:

- RAM
- CPU count
- Storage controllers
- Virtual disks
- Network adapters
- Graphics settings
- Boot order

---

## 4. Create a new virtual machine

```
VBoxManage createvm --name Ubuntu22 --ostype Ubuntu_64 --register
```

Options:

- `--name` → VM name
- `--ostype` → Guest operating system type
- `--register` → Registers the VM with VirtualBox

## 5. Modify VM settings

**Example:**

```
VBoxManage modifyvm Ubuntu22 --memory 4096 --cpus 2
```

Sets

- RAM = 4096 MB
- CPUs = 2

---

## 6. Start a VM

**Graphical mode**

```
VBoxManage startvm Ubuntu22
```

**Headless mode (no GUI)**

```
VBoxManage startvm Ubuntu22 --type headless
```

Useful on servers.

---

## 7. Shut down a VM

**Graceful shutdown**

```
VBoxManage controlvm Ubuntu22 acpipowerbutton
```

Equivalent to pressing the computer's power button.

## 8. Force power off

```
VBoxManage controlvm Ubuntu22 poweroff
```

Similar to unplugging the power cable.

Use only when necessary.

---

## 9. Pause a VM

```
VBoxManage controlvm Ubuntu22 pause
```

**Resume:**

```
VBoxManage controlvm Ubuntu22 resume
```

---

## 10. Reset a VM

```
VBoxManage controlvm Ubuntu22 reset
```

Equivalent to pressing the hardware reset button.

---

## 11. Take a snapshot

```
VBoxManage snapshot Ubuntu22 take BeforeUpdate
```

Creates a restore point.

---

## 12. List snapshots

```
VBoxManage snapshot Ubuntu22 list
```

## 13. Restore a snapshot

```
VBoxManage snapshot Ubuntu22 restore BeforeUpdate
```

Returns the VM to the saved state.

---

## 14. Delete a VM

```
VBoxManage unregistervm Ubuntu22 --delete
```

**Deletes:**

- VM configuration
- Virtual disks
- Snapshots

---

## 15. Clone a VM

```
VBoxManage clonevm Ubuntu22 --name UbuntuClone --register
```

Creates a copy of the VM.

---

## 16. Create a virtual hard disk

```
VBoxManage createmedium disk --filename Ubuntu22.vdi --size 20000
```

Creates a 20 GB virtual disk (size is specified in MB).

---

## 17. Attach a virtual disk

```
VBoxManage storageattach Ubuntu22 \
  --storagectl SATA \
  --port 0 \
  --device 0 \
  --type hdd \
  --medium Ubuntu22.vdi
```

Attaches a virtual hard disk to the VM.

### 18. Export a VM

```
VBoxManage export Ubuntu22 --output Ubuntu22.ova
```

Creates an OVA appliance.

---

### 19. Import a VM

```
VBoxManage import Ubuntu22.ova
```

Imports an OVA appliance.

---

### 20. Enable bridged networking

```
VBoxManage modifyvm Ubuntu22 --nic1 bridged --bridgeadapter1 eth0
```

Allows the VM to appear as another device on the physical network.

---

### Frequently Used `VBoxManage` Subcommands

| Command | Purpose |
|---|---|
| `list` | List VMs, disks, host information |
| `createvm` | Create a VM |
| `modifyvm` | Change VM settings |
| `startvm` | Start a VM |
| `controlvm` | Control a running VM (pause, reset, power off, etc.) |
| `showvminfo` | Display VM details |
| `snapshot` | Manage snapshots |
| `clonevm` | Clone a VM |
| `storageattach` | Attach storage devices |
| `createmedium` | Create virtual disks |
| `import` | Import an appliance |
| `export` | Export an appliance |
| `unregistervm` | Remove a VM |

## Advantages

- Works without the VirtualBox GUI.
- Ideal for scripting and automation.
- Supports remote and headless VM management.
- Provides access to advanced configuration options.
- Suitable for server environments.

---

## Things to Remember

- **`VBoxManage`** is installed with Oracle VirtualBox.
- It can manage almost every aspect of VirtualBox from the terminal.
- Some operations (such as changing memory or CPU count) require the VM to be powered off.
- Running **`VBoxManage`** without arguments displays help and a list of available subcommands.

---

## Example Workflow

```
# Create a VM
VBoxManage createvm --name Ubuntu22 --ostype Ubuntu_64 --register

# Allocate resources
VBoxManage modifyvm Ubuntu22 --memory 4096 --cpus 2

# Create a virtual disk
VBoxManage createmedium disk --filename Ubuntu22.vdi --size 25000

# Start the VM
VBoxManage startvm Ubuntu22

# Display VM information
VBoxManage showvminfo Ubuntu22

# Gracefully shut down the VM
VBoxManage controlvm Ubuntu22 acpipowerbutton
```

This sequence demonstrates a typical lifecycle: create a VM, configure it, create storage, start it, inspect its configuration, and shut it down cleanly.

# 6. docker

The **`docker`** command is the primary command-line interface (CLI) used to interact with the Docker platform. Docker allows you to **build, run, manage, and distribute applications inside lightweight containers**.

A **container** packages an application together with all its dependencies, libraries, and configuration files, ensuring it runs consistently across different systems.

---

## Syntax

```
docker [OPTIONS] COMMAND [ARGUMENTS]
```

- **OPTIONS** – Global options that affect the Docker client.
- **COMMAND** – The Docker operation to perform.
- **ARGUMENTS** – Additional parameters required by the command.

---

## What Docker Can Do

- Build container images
- Run containers
- Manage images
- Manage networks
- Manage storage volumes
- Connect to Docker registries
- Monitor running containers
- Create multi-container applications

### Common Docker Commands

| Command | Description |
|---|---|
| `docker version` | Display Docker client and server versions |
| `docker info` | Show Docker system information |
| `docker images` | List downloaded images |
| `docker ps` | Show running containers |
| `docker ps -a` | Show all containers |
| `docker run` | Create and start a container |
| `docker start` | Start an existing container |
| `docker stop` | Stop a running container |
| `docker restart` | Restart a container |
| `docker rm` | Remove a container |
| `docker rmi` | Remove an image |
| `docker pull` | Download an image |
| `docker push` | Upload an image |
| `docker build` | Build an image from a Dockerfile |
| `docker exec` | Execute a command inside a running container |
| `docker logs` | View container logs |
| `docker inspect` | Display detailed container or image information |
| `docker network ls` | List Docker networks |
| `docker volume ls` | List Docker volumes |

## Basic Examples

### 1. Check Docker Version

```
docker version
```

**Example output:**

```
Client:
 Version: 28.1.0

Server:
 Engine:
  Version: 28.1.0
```

## 2. Display Docker Information

```
docker info
```

**Shows:**

- Number of containers
- Number of images
- Storage driver
- CPU count
- Memory
- Docker Root Directory
- Operating System

---

## 3. Download an Image

```
docker pull ubuntu
```

Downloads the latest Ubuntu image.

---

## 4. List Images

```
docker images
```

**Example:**

```
REPOSITORY   TAG       IMAGE ID       SIZE
ubuntu       latest    a04dc4851cbc   78MB
nginx        latest    d453dd892d93   192MB
```

---

## 5. Run a Container

```
docker run ubuntu
```

Runs an Ubuntu container.

**Interactive mode:**

```
docker run -it ubuntu bash
```

- `-i` → Interactive
- `-t` → Allocate a terminal

---

## 6. Run a Web Server

```
docker run -d -p 8080:80 nginx
```

**Meaning:**

- `-d` → Detached mode
- `-p 8080:80` → Map host port 8080 to container port 80

**Access:**

```
http://localhost:8080
```

---

## 7. List Running Containers

```
docker ps
```

**Example:**

```
CONTAINER ID   IMAGE    STATUS      PORTS
ab1234         nginx    Up 5 mins   0.0.0.0:8080->80/tcp
```

---

## 8. Stop a Container

```
docker stop ab1234
```

### 9. Remove a Container

```
docker rm ab1234
```

---

### 10. View Logs

```
docker logs ab1234
```

Displays the application's output from the container.

---

### 11. Execute a Command Inside a Container

```
docker exec -it ab1234 bash
```

Starts a Bash shell inside the running container.

---

### 12. Build an Image

```
docker build -t myapp .
```

Meaning:

- `-t` → Assign the image name `myapp`
- `.` → Use the current directory as the build context

---

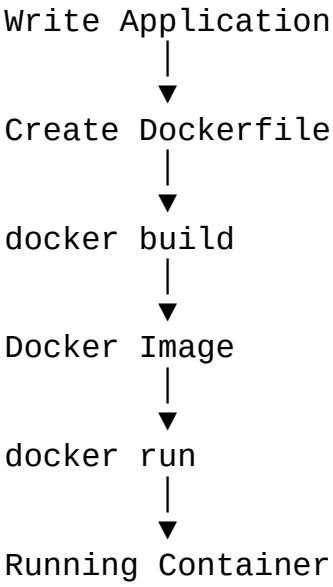
### Frequently Used Options

| Option | Meaning |
|---|---|
| `-d` | Run in background (detached mode) |
| `-it` | Interactive terminal |
| `-p` | Publish port |
| `-v` | Mount a volume |
| `--name` | Assign a container name |
| `--rm` | Automatically remove the container after it exits |
| `-e` | Set environment variables |
| `--network` | Connect the container to a specific network |

## Docker Object Types

| Object | Purpose |
|---|---|
| Image | Read-only template used to create containers |
| Container | Running instance of an image |
| Volume | Persistent data storage |
| Network | Communication between containers |
| Registry | Repository for storing images (for example, Docker Hub) |

## Typical Workflow




---

## When to Use Docker

- Application development
- Software testing
- Continuous Integration/Continuous Deployment (CI/CD)
- Microservices
- Web server deployment
- Database testing
- Cloud-native applications
- Isolated development environments

## Advantages

- Lightweight compared to virtual machines
- Fast startup time
- Portable across systems
- Consistent environments
- Efficient resource usage
- Easy application deployment
- Simplified dependency management

---

## Disadvantages

- Containers share the host OS kernel (less isolation than virtual machines)
- Linux containers cannot run directly on a Windows kernel (and vice versa) without additional compatibility layers or virtualisation
- Persistent data requires proper use of volumes or bind mounts
- Networking and storage can become complex in large deployments

## Quick Reference

| Task | Command |
|---|---|
| Check version | `docker version` |
| System information | `docker info` |
| Download image | `docker pull ubuntu` |
| List images | `docker images` |
| Run container | `docker run ubuntu` |
| Interactive shell | `docker run -it ubuntu bash` |
| List running containers | `docker ps` |
| Stop container | `docker stop <container>` |
| Remove container | `docker rm <container>` |
| Remove image | `docker rmi <image>` |
| Build image | `docker build -t myapp .` |
| Execute command | `docker exec -it <container> bash` |
| View logs | `docker logs <container>` |

**In summary:** `docker` is the main command used to manage Docker images, containers, networks, and volumes. It is one of the most important tools in modern software development and DevOps because it enables applications to run reliably in isolated, portable environments.

# 7. podman

**`podman`** (**Pod Manager**) is a **container management tool** used to create, run, stop, and manage **OCI (Open Container Initiative)** containers and container images. It is a popular alternative to Docker and is designed to run **without a central daemon**, making it more secure and lightweight.

---

## Syntax

```
podman [OPTIONS] COMMAND [ARGUMENTS]
```

**Example:**

```
podman run -it ubuntu bash
```

This starts an Ubuntu container and opens a Bash shell.

---

## Why use Podman ?

Podman offers several advantages:

- Daemonless architecture (no background service required)
- Rootless containers (run containers as a normal user)
- Docker-compatible commands
- Better security
- Supports pods (similar to Kubernetes pods)

### Basic Workflow

```
Pull Image
     |
     ▼
Create Container
     |
     ▼
Run Container
     |
     ▼
Stop / Restart
     |
     ▼
Remove Container
```

---

## Common Commands

### 1. Show Podman version

```
podman --version
```

**Example:**

```
$ podman --version
podman version 5.2.0
```

---

### 2. Display help

```
podman --help
```

Shows all available commands.

## 3. Pull an image

```
podman pull ubuntu
```

Downloads the Ubuntu image.

**Output:**

```
Trying to pull docker.io/library/ubuntu...

Downloaded newer image
```

---

## 4. List downloaded images

```
podman images
```

**Example:**

```
REPOSITORY    TAG      IMAGE ID
ubuntu        latest   6b7dfa7e8fdb
```

---

## 5. Run a container

```
podman run ubuntu
```

Runs the Ubuntu image.

**Interactive mode:**

```
podman run -it ubuntu bash
```

**Options:**

- `-i` → Interactive input
- `-t` → Allocate terminal

## 6. Run a container in background

```
podman run -d nginx
```

`-d` means detached mode.

---

## 7. List running containers

```
podman ps
```

**Example:**

```
CONTAINER ID
NAME
STATUS
```

---

## 8. List all containers

```
podman ps -a
```

Shows running and stopped containers.

---

## 9. Stop a container

```
podman stop mycontainer
```

or

```
podman stop <container-id>
```

---

## 10. Start a stopped container

```
podman start mycontainer
```

### 11. Restart a container

```
podman restart mycontainer
```

---

### 12. Remove a container

```
podman rm mycontainer
```

**Force removal:**

```
podman rm -f mycontainer
```

---

### 13. Remove an image

```
podman rmi ubuntu
```

---

### 14. Execute a command inside a running container

```
podman exec -it mycontainer bash
```

---

### 15. View container logs

```
podman logs mycontainer
```

---

### 16. Inspect a container

```
podman inspect mycontainer
```

Displays detailed JSON information.

## 17. View resource usage

```
podman stats
```

**Shows:**

- CPU usage
- Memory usage
- Network usage
- Block I/O

---

## 18. Create a pod

```
podman pod create --name webpod
```

Pods group one or more containers together.

---

## 19. List pods

```
podman pod ps
```

---

## 20. Remove unused objects

```
podman system prune
```

**Removes:**

- Stopped containers
- Unused images
- Networks
- Build cache

### Frequently Used Options

| Option | Meaning |
|---|---|
| `-d` | Detached mode |
| `-it` | Interactive terminal |
| `--name` | Assign container name |
| `-p` | Publish port |
| `-v` | Mount volume |
| `-e` | Set environment variable |
| `--rm` | Remove container after exit |
| `--network` | Specify network |
| `--privileged` | Give extended privileges |

## Examples

### Run Ubuntu interactively

```
podman run -it ubuntu bash
```

---

### Run Nginx web server

```
podman run -d -p 8080:80 nginx
```

Access the web server:

```
http://localhost:8080
```

---

### Mount a local directory

```
podman run -it -v /home/user/data:/data ubuntu bash
```

---

### Set an environment variable

```
podman run -e APP_ENV=production nginx
```

### Automatically remove the container

```
podman run --rm ubuntu echo "Hello"
```

The container is deleted after it exits.

---

### Podman vs Docker

| Feature | Podman | Docker |
|---|---|---|
| Daemon required | No | Yes |
| Rootless support | Excellent | Supported |
| Docker-compatible CLI | Yes | Yes |
| Supports Pods | Yes | Limited |
| OCI compliant | Yes | Yes |
| Kubernetes integration | Strong | Good |
| Security | Higher (daemonless and rootless bydefault) | Good |

---

### Typical Administration Tasks

```
# Download image
podman pull ubuntu

# Run container
podman run -it ubuntu bash

# List running containers
podman ps

# Stop container
podman stop <container-id>

# Remove container
podman rm <container-id>

# Remove image
podman rmi ubuntu

# Clean unused resources
podman system prune
```

## Exit Status

- `0` – Command completed successfully.
- Non-zero (for example, `1` or `125`) – An error occurred. `125` often indicates Podman itself could not run the container, while the container's own process may return other exit codes.

---

## When is `podman` used ?

System administrators and developers commonly use Podman to:

- Develop and test applications in isolated containers.
- Run containerized services such as web servers and databases.
- Build and manage container images.
- Experiment with Kubernetes-style pods on a local system.
- Run containers securely as a non-root user.
- Automate container deployments in scripts and CI/CD pipelines.

In modern Linux distributions (especially those based on Red Hat technologies), Podman is often the default container engine because of its daemonless architecture, strong rootless support, and compatibility with Docker workflows.

# 8. lxc

The `lxc` command is the primary command-line tool used to **create, manage, and interact with Linux containers**. It is part of the **LXC (Linux Containers)** project, which provides lightweight operating-system-level virtualization.

Unlike a virtual machine, an LXC container shares the host's Linux kernel while maintaining its own isolated environment.

---

## Syntax

```
lxc <subcommand> [options]
```

---

## Common Subcommands

| Command | Description |
|---|---|
| `lxc launch` | Create and start a new container |
| `lxc list` | List all containers |
| `lxc start` | Start a stopped container |
| `lxc stop` | Stop a running container |
| `lxc restart` | Restart a container |
| `lxc exec` | Execute a command inside a container |
| `lxc shell` | Open a shell inside a container |
| `lxc delete` | Delete a container |
| `lxc info` | Show container information |
| `lxc config` | Configure container settings |
| `lxc file` | Transfer files to/from containers |
| `lxc image` | Manage container images |
| `lxc snapshot` | Create or restore snapshots |
| `lxc network` | Manage virtual networks |
| `lxc storage` | Manage storage pools and volumes |

# Examples

## 1. List all containers

```
lxc list
```

Example output:

```
+---------+---------+----------------------+-----------------------------------------+
| NAME    | STATE   | IPV4                 | TYPE                                    |
+---------+---------+----------------------+-----------------------------------------+
| web01   | RUNNING | 10.10.10.15 (eth0)   | CONTAINER                               |
+---------+---------+----------------------+-----------------------------------------+
```

---

## 2. Launch a new Ubuntu container

```
lxc launch ubuntu:22.04 mycontainer
```

- Downloads the Ubuntu 22.04 image (if not already cached).
- Creates a container named **mycontainer**.
- Starts it automatically.

---

## 3. Start a container

```
lxc start mycontainer
```

Starts an existing container.

---

## 4. Stop a container

```
lxc stop mycontainer
```

Stops the running container gracefully.

Force stop:

```
lxc stop mycontainer --force
```

### 5. Restart a container

```
lxc restart mycontainer
```

---

### 6. Open a shell inside the container

```
lxc shell mycontainer
```

or

```
lxc exec mycontainer -- bash
```

You will get a Bash shell inside the container.

---

### 7. Run a single command inside the container

```
lxc exec mycontainer -- ls /home
```

Output:

```
ubuntu
```

---

### 8. Show container information

```
lxc info mycontainer
```

Displays:

- Status
- Process ID
- CPU usage
- Memory usage
- Disk usage
- Network information

## 9. Delete a container

**First stop it:**

```
lxc stop mycontainer
```

**Then delete it:**

```
lxc delete mycontainer
```

**Delete even if it's running:**

```
lxc delete mycontainer --force
```

---

## 10. Create a snapshot

```
lxc snapshot mycontainer snap1
```

**List snapshots:**

```
lxc info mycontainer
```

**Restore snapshot:**

```
lxc restore mycontainer snap1
```

---

## 11. Copy a file into the container

```
lxc file push test.txt mycontainer/root/
```

**Copy a file from the container:**

```
lxc file pull mycontainer/root/test.txt .
```

---

## 12. Configure CPU limits

```
lxc config set mycontainer limits.cpu 2
```

Limits the container to **2 CPU cores**.

### 13. Configure memory limit

```
lxc config set mycontainer limits.memory 2GiB
```

Limits memory usage to **2 GiB**.

---

### 14. View available images

```
lxc image list images:
```

Shows downloadable Linux distributions.

---

### 15. Launch an Alpine Linux container

```
lxc launch images:alpine/3.20 alpine1
```

---

### 16. Delete an image

```
lxc image delete <fingerprint>
```

---

### Common Options

| Option | Description |
|---|---|
| `--help` | Show help |
| `--version` | Show version |
| `--force` | Force the action |
| `--project` | Specify a project |
| `--target` | Target a specific cluster member |
| `--debug` | Enable debug output |
| `--verbose` | Show detailed output |

## When is `lxc` Used ?

- Running lightweight Linux containers
- Development and testing environments
- Application isolation
- CI/CD pipelines
- Hosting multiple isolated services on a single Linux host
- Learning Linux without full virtual machines

---

## `lxc` -vs- Docker -vs- Virtual Machines

| Feature | LXC | Docker | Virtual Machine |
|---|---|---|---|
| Shares host kernel | | | |
| Full Linux userspace | | Partial (application-focused) | |
| Lightweight | | | |
| Boots like a Linux system | | Usually runs one main process | |
| Runs systemd | | Usually not | |
| Performance | Very high | Very high | Lower than containers |
| Best for | System containers | Application containers | Full OS virtualization |

### Practical Workflow

```
# Launch a container
lxc launch ubuntu:22.04 webserver

# Verify it's running
lxc list

# Open a shell
lxc shell webserver

# Install software inside the container
apt update
apt install nginx -y

# Exit the container
exit

# Stop the container
lxc stop webserver

# Start it again later
lxc start webserver
```

---

### Summary

- **Command:** `lxc`
- **Purpose:** Manage Linux system containers.
- **Key uses:** Create, start, stop, configure, and delete containers.
- **Advantages:** Lightweight, fast, efficient, and ideal for running isolated Linux environments that share the host kernel.
- **Common subcommands:** `launch`, `list`, `start`, `stop`, `exec`, `shell`, `config`, `snapshot`, `file`, `image`, and `delete`.

# 9. qm

The `qm` command is a **virtual machine management tool** used in **Proxmox Virtual Environment (Proxmox VE)**. It allows you to create, configure, start, stop, and manage **KVM/QEMU virtual machines** from the command line.

### Syntax

```
qm <command> [options]
```

---

### What is QEMU?

- **QEMU** = Quick Emulator
- It is an open-source machine emulator and virtualizer.
- In Proxmox, `qm` is the command-line interface for managing **QEMU/KVM virtual machines**.

### Common `qm` Commands

| Command | Description |
|---|---|
| `qm list` | List all virtual machines |
| `qm create` | Create a new VM |
| `qm start` | Start a VM |
| `qm stop` | Stop a VM immediately |
| `qm shutdown` | Gracefully shut down a VM |
| `qm reset` | Restart a VM (hard reset) |
| `qm destroy` | Delete a VM |
| `qm clone` | Clone an existing VM |
| `qm config` | Show VM configuration |
| `qm set` | Modify VM configuration |
| `qm status` | Show VM status |
| `qm suspend` | Suspend a VM |
| `qm resume` | Resume a suspended VM |
| `qm migrate` | Migrate a VM to another node |
| `qm snapshot` | Create or manage snapshots |
| `qm rollback` | Restore a snapshot |
| `qm terminal` | Open a serial console |
| `qm monitor` | Access the QEMU monitor |

# Examples

## 1. List all virtual machines

```
qm list
```

**Example output:**

```
VMID NAME         STATUS
100  Ubuntu       running
101  Windows11    stopped
102  RockyLinux   running
```

---

## 2. Show VM configuration

```
qm config 100
```

Displays settings such as:

- CPU
- RAM
- Disk
- Network
- Boot order

---

## 3. Start a VM

```
qm start 100
```

Starts VM **100**.

---

## 4. Gracefully shut down a VM

```
qm shutdown 100
```

This sends an ACPI shutdown signal, similar to pressing the power button on a physical computer.

### 5. Force stop a VM

```
qm stop 100
```

Immediately powers off the VM.

---

### 6. Check VM status

```
qm status 100
```

Example:

```
status: running
```

---

### 7. Create a VM

```
qm create 100 \
    --name ubuntu \
    --memory 4096 \
    --cores 2 \
    --net0 virtio,bridge=vmbr0
```

Creates VM **100** with:

- 4 GB RAM
- 2 CPU cores
- VirtIO network adapter

---

### 8. Change VM memory

```
qm set 100 --memory 8192
```

Changes memory to **8 GB**.

## 9. Add a virtual disk

```
qm set 100 --scsi0 local-lvm:50
```

Adds a **50 GB** virtual disk.

---

## 10. Clone a VM

```
qm clone 100 200 --name ubuntu-clone
```

Creates VM **200** as a clone of VM **100**.

---

## 11. Delete a VM

```
qm destroy 100
```

Deletes VM **100**.

**Note:** This permanently removes the VM. Depending on the options used and storage configuration, its disks may also be deleted.

---

## 12. Create a snapshot

```
qm snapshot 100 before-update
```

Creates a snapshot named **before-update**.

---

## 13. Roll back to a snapshot

```
qm rollback 100 before-update
```

Restores the VM to the saved snapshot.

### 14. Migrate a VM

```
qm migrate 100 node2
```

Moves VM **100** to another Proxmox node named **node2**.

---

### Useful Options

| Option | Description |
|---|---|
| `--memory` | RAM size (MB) |
| `--cores` | Number of CPU cores |
| `--sockets` | Number of CPU sockets |
| `--name` | VM name |
| `--boot` | Boot order |
| `--cdrom` | Attach an ISO image |
| `--net0` | Configure the first network interface |
| `--scsi0` | Attach a SCSI disk |
| `--ide0` | Attach an IDE disk |
| `--agent` | Enable QEMU Guest Agent |

---

### When is `qm` used?

System administrators use `qm` to:

- Create virtual machines
- Configure CPU, RAM, and storage
- Attach ISO images
- Install operating systems
- Take snapshots
- Clone VMs
- Migrate VMs between Proxmox nodes
- Automate VM management with shell scripts

## Difference Between `qm` and `pct`

| Command | Used For |
|---|---|
| `qm` | QEMU / KVM virtual machines (full virtualization) |
| `pct` | LXC containers (operating-system-level virtualization) |

## Summary

- **Command:** `qm`
- **Purpose:** Manage **QEMU / KVM virtual machines** in Proxmox VE.
- **Common tasks:** Create, start, stop, configure, clone, snapshot, migrate, and delete VMs.
- **Typical usage:** `qm <subcommand> <vmid> [options]`

If you're learning Linux administration or Proxmox, `qm` is one of the core commands for managing full virtual machines from the terminal.

# 10. pct

**`pct`** is **not a standard Linux command**. It is a command-line utility provided by **Proxmox VE (Virtual Environment)** to **manage Linux Containers (LXC)**.

It allows administrators to create, configure, start, stop, and manage containers from the terminal.

---

### Syntax

```
pct <command> <container-id> [options]
```

- `<command>` – The action to perform.
- `<container-id>` – The numeric ID of the container (for example, `100`).

---

### Common Commands

| Command | Description |
|---|---|
| `pct list` | Lists all containers |
| `pct create` | Creates a new container |
| `pct start <id>` | Starts a container |
| `pct stop <id>` | Stops a container |
| `pct shutdown <id>` | Gracefully shuts down a container |
| `pct restart <id>` | Restarts a container |
| `pct destroy <id>` | Deletes a container |
| `pct status <id>` | Shows container status |
| `pct config <id>` | Displays container configuration |
| `pct set <id>` | Modifies container settings |
| `pct exec <id> <command>` | Runs a command inside the container |
| `pct enter <id>` | Opens a shell inside the container |
| `pct clone <id>` | Clones a container |
| `pct snapshot <id>` | Creates a snapshot |
| `pct restore` | Restores a backup |

# Examples

## List all containers

```
pct list
```

**Example output:**

```
VMID  Status   Lock  Name
100   running        webserver
101   stopped        database
```

---

## Start a container

```
pct start 100
```

Starts container **100**.

---

## Stop a container

```
pct stop 100
```

Immediately stops the container.

---

## Gracefully shut down

```
pct shutdown 100
```

Sends a shutdown request to the operating system running inside the container.

---

## Check container status

```
pct status 100
```

**Output:**

```
status: running
```

### Enter a container

```
pct enter 100
```

You will get a shell inside the container:

```
root@container:~#
```

---

### Execute a single command inside a container

```
pct exec 100 -- apt update
```

Runs `apt update` without opening an interactive shell.

---

### View container configuration

```
pct config 100
```

**Example:**

```
hostname: webserver
memory: 2048
cores: 2
net0: name=eth0,bridge=vmbr0
rootfs: local-lvm:8
```

---

### Change memory

```
pct set 100 --memory 4096
```

Assigns **4 GB RAM** to container 100.

---

### Create a snapshot

```
pct snapshot 100 before-update
```

Creates a snapshot named **before-update**.

### Clone a container

```
pct clone 100 200 --hostname webserver-copy
```

Creates a new container (ID **200**) from container **100**.

---

### Frequently Used Options

| Option | Description |
|---|---|
| `--memory` | Set RAM size |
| `--cores` | Set CPU cores |
| `--hostname` | Set hostname |
| `--rootfs` | Configure root filesystem |
| `--net0` | Configure network interface |
| `--storage` | Specify storage backend |
| `--password` | Set root password |
| `--features` | Enable container features |

---

### When is `pct` used?

You use `pct` when administering **LXC containers on a Proxmox VE host**. It provides functionality similar to:

- `virsh` for KVM virtual machines
- `docker` for Docker containers
- `lxc`/`lxc-*` for generic LXC management

but is specifically integrated with Proxmox VE.

---

### Summary

- **Command:** `pct`
- **Purpose:** Manage LXC containers in Proxmox VE.
- **Platform:** Proxmox VE hosts.
- **Common tasks:** Create, start, stop, configure, clone, snapshot, back up, restore, and access Linux containers.