Community Experience Distilled

# Mastering Proxmox

Master Proxmox VE to effectively implement server virtualization technology within your network

Wasim Ahmed

[PACKT] open source* PUBLISHING
community experience distilled

# Mastering Proxmox

Master Proxmox VE to effectively implement server virtualization technology within your network

**Wasim Ahmed**

[PACKT] open source
PUBLISHING community experience distilled

BIRMINGHAM - MUMBAI

# Mastering Proxmox



First published: July 2014

Production reference: 1070714

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78398-082-6

`www.packtpub.com`

Cover image by Duraid Fatouhi (`duraidfatouhi@yahoo.com`)

# Credits


**Author**
Wasim Ahmed

**Reviewers**
Rocco Alfonzetti Jr.
Alessio Bravi
Oleg Butovich
Daniel Lench
Razique Mahroua

**Commissioning Editor**
Kartikey Pandey

**Acquisition Editor**
Mohammad Rizvi

**Content Development Editor**
Madhuja Chaudhari

**Technical Editor**
Rohit Kumar Singh

**Copy Editors**
Alisha Aranha
Sarang Chari
Mradula Hegde
Gladson Monteiro
Adithi Shetty

**Project Coordinator**
Neha Bhatnagar

**Proofreaders**
Simran Bhogal
Amy Johnson
Linda Morris

**Indexers**
Mehreen Deshmukh
Rekha Nair
Tejal Soni

**Graphics**
Ronak Dhruv

**Production Coordinator**
Komal Ramchandani

**Cover Work**
Komal Ramchandani

# About the Author

**Wasim Ahmed**, born in Bangladesh and now a citizen of Canada, is a veteran of the IT world. He was introduced to computers in the year 1992 and never looked back. Wasim has deep knowledge and understanding of network virtualization, big data storage, and network security. By profession, Wasim is the CEO of an IT support and cloud service provider company based in Calgary, Alberta. He serves many companies and organizations through his company on a daily basis. Wasim's strength comes from the experience he gained through learning and serving continually. Wasim strives on finding the most effective solution at the most competitive price point. He hand-built over a dozen enterprise production virtual infrastructures using Proxmox and Ceph storage system.

Wasim is notoriously known not to simply accept a technology based on its description alone, but put them through rigorous tests to check their validity. Any new technology that his company provides goes through months of continuous testing before it is accepted. Proxmox made the cut superbly.

---

I would like to thank all the staff at Proxmox for their support and dedication to the hypervisor community. I would also like to thank Packt Publishing for their vision of moving forward for this one-of-a-kind book on Proxmox and their support throughout the journey of making of this book.

---

# About the Reviewers

**Rocco Alfonzetti Jr.** is an IT consultant for small businesses and has specialized in Linux and open source solutions for the last 15 years. Currently, he works for a software development company as an e-mail security expert. He lives in rural Connecticut with his wife and three children, and in his spare time, he enjoys beekeeping, raising chickens, and gardening.

**Alessio Bravi** has been playing with bits since he was five. He started programming at the age of six and soon focused his attention towards network administration and IT systems security in the best growing-up period of the Internet.

When he was 19, he founded IntSec.NET, and started working as CTO and Network and Security Administrator for Italian Internet service providers (ISPs/W-ISPs) and as an IT security consultant for many companies in Europe.

Alessio works only with Unix-like operating systems and is specialized in IT security analysis, network engineering and administration, autonomous systems BGP routing, IPv4 and IPv6 routing and switching, operating system virtualization, and data center management.

His personal blog can be found at `http://blog.bravi.org/`, where he writes some technical articles to share IT hints with the digital world. More technical skills and personal details about Alessio can be found on his LinkedIn© profile page at `http://www.linkedin.com/in/alessiobravi`.

**Oleg Butovich** is a freelance senior software developer with a passion for virtualization technologies. He has over 15 years of experience in the industry. He has worked on booking systems, trading platforms, laser image generators, digital media systems, medical and life science imaging systems, automatic inspection systems, and embedded systems.

**Daniel Lench** is a self-proclaimed "fixer of all things". He is drawn to challenges, both physical and theoretical. His background includes acting as an artisan at a state museum, a production manager at a high-volume cabinet shop, AutoCAD expert for civil engineering firms and government agencies, and almost two decades of being professionally involved in the IT industry. In 2008, the challenge was to keep files in sync between multiple computers in real time. Since then he has been focused on finding the best answer. The NoFolder Project is an open source, real-time, private cloud-based backup, file synchronization, and collaboration service that is self hosted and administered in small business and enterprise settings. NoFolder addressed the policy and privacy concerns over using third-party services to store and share data. The project is for those concerned about data, the collaboration with it, and the preservation of it. The company maintains offices in the U.S. and the U.K. with additional resources in Sweden, Austria, and South Africa. Daniel is the founder as well as the CEO for NoFolder Ltd.

---

I would like to thank Rocco for introducing Proxmox to me. I would also like to thank Heather for the wonderful adventure.

---

**Razique Mahroua** is a technical consultant on High Availability systems as well as a technical writer. Currently involved in several open source projects, such as OpenStack and KVM, he has written about various technical topics for IBM and Amazon.

His experience ranges from cloud solutions, implementations (IaaS and PaaS), and by-products such as data clustering to network High Availability and data integrity. He currently assists several companies looking for best practices around cloud solutions.

# www.PacktPub.com

## Support files, eBooks, discount offers, and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.


`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

### Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

### Free access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

*To my dear wife, Brigitta, whose love, constant support, and unshakeable faith in me has allowed me to do what I do best, while she took care of the rest.*

*–Wasim Ahmed*

# Table of Contents

# Preface

This book is well overdue in the world of virtualization. When I first came in contact with Proxmox several years ago, I did not have anything to fall back on other than Proxmox Wiki and forum. I learned Proxmox through lots of trial and error and very much had to reinvent wheels on my own in some cases. Since a lot of us went through the frustration and I personally do not feel others should have to invest a lot of time just to get to know Proxmox the hard way, this book has been written.

This book shows the inner workings of Proxmox including virtual network components, shared storage systems, nested virtualization, complex network topologies, and so on. With this book, we hope that the reader will be able to better equip themselves to face any virtualization challenges of any virtual infrastructure.

## What this book covers

*Chapter 1*, *Dive into the Virtual World with Proxmox*, introduces Proxmox in general and the graphical user interface.

*Chapter 2*, *Proxmox Under the Hood*, introduces the Proxmox directory structure and configuration files.

*Chapter 3*, *Shared Storages with Proxmox*, explains how Proxmox interacts with the shared storage system and types of shared storage system supported.

*Chapter 4*, *A Virtual Machine for a Virtual World*, covers advanced virtual machine configurations such as enabling sound, USB devices, and so on.

*Chapter 5*, *Network of Virtual Networks*, explains the different networking components used in Proxmox to build virtual networks.

*Chapter 6*, *Proxmox HA – Zero Downtime*, explains the Proxmox High Availability (HA) feature and how to configure it.

*Chapter 7*, *High Availability Storage for High Availability Cluster*, explains a step-by-step process of setting up the Ceph cluster to be used as a shared storage system.

*Chapter 8*, *Proxmox Production Level Setup*, explains the type of hardware that should be and can be used in a production level Proxmox cluster setup.

*Chapter 9*, *Proxmox Troubleshooting*, lists real incidents with solutions that may arise in the Proxmox cluster.

*Chapter 10*, *Putting It All Together*, introduces several scenario-based virtual environments along with full network diagrams.

# What you need for this book

Since we will be working with the Proxmox cluster throughout the book, it will be extremely helpful to have a working Proxmox cluster of your own. A very basic cluster of two Proxmox nodes and a storage node will do just fine.

# Who this book is for

This book is for readers who want to build a virtual infrastructure purely based on Proxmox as hypervisor and Ceph as storage backend. Whether the reader is a veteran in the virtualized industry but has never worked with Proxmox, or somebody just starting out a promising career in this industry, this book will serve well.

# Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "The beginning of the tag shows the name of the cluster as `name="pmx-cluster"`."

A block of code is set as follows:

```
<?xml version="1.0"?>
<cluster name="pmx-cluster" config_version="2">
<cman keyfile="/var/lib/pve-cluster/corosync.authkey"></cman>
<clusternodes>
  <clusternode name="pmxvm01" votes="1" nodeid="1"/>
```

```
    <clusternode name="pmxvm02" votes="1" nodeid="2"/>
  </clusternodes>
  </cluster>
```

Any command-line input or output is written as follows:

```
# ssh root@192.168.145.1
```

**New terms** and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "The **Storage** tab is probably one of the most important options in the Proxmox GUI."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

# Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

# Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

## Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

# Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

# Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

# Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

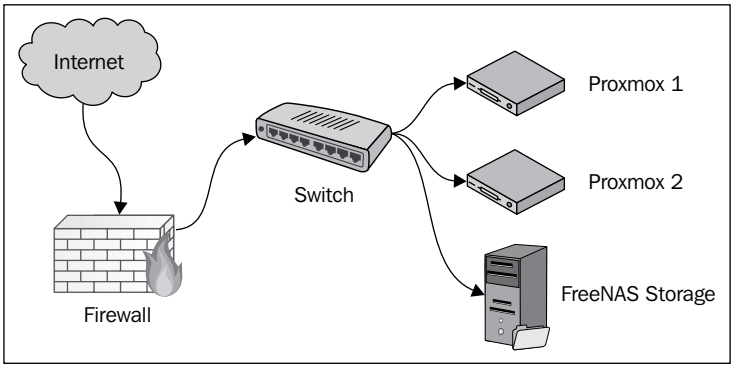
# 1
# Dive into the Virtual World with Proxmox

With the rise in energy costs and the advancement of computer processing power and memory capacity, running separate, under-utilized server hardware with specific roles is no longer a luxury, and thus was born the technology we so affectionately call **virtualization**. The term virtualization mainly refers to virtualizing servers, but of late it is also being used for network or infrastructure virtualization. Server and network virtualization enabled us to truly create a virtual world where anything is possible. Virtual data centers, virtual storage systems, and virtual desktop PCs are just a few real-world applications where virtualization is being heavily used. Although nowadays the world is being swept by virtualization, the term virtualization itself is not new. The concept was used in mainframes of the 1960s, where virtualization was a way to logically divide the mainframe's resources for different application processing.

A **hypervisor** is the underlying platform or foundation that allows a virtual world to be built upon. In a way it is the very building block of all virtualization. A bare metal hypervisor acts as a bridge between physical hardware and the virtual machines by creating an abstraction layer. Because of this unique feature, an entire virtual machine can be moved over a vast distance over the Internet and be made able to function exactly the same. A virtual machine does not see the hardware directly; instead, it sees the layer of the hypervisor, which is the same no matter on what hardware the hypervisor has been installed.

The Proxmox hypervisor is one of the best kept secrets in the modern computer world. The reason is simple. It allows for the building of an enterprise business-class virtual infrastructure at a small business-class price tag without sacrificing stability, performance, and ease of use. Whether it is a massive data center to serve millions of people, a small educational institution, or a home serving important family members, Proxmox can fulfil the needs of just about any situation. Even a novice networker can get a stable virtualization platform up and running in less than an hour.

A Proxmox cluster consists of two or more computer nodes with Proxmox as the operating system and connected in the same network. A virtual machine can migrate from one node to another in the same cluster, which allows redundancy should a node fail for any reason. Refer to the following diagram of a very basic two-node Proxmox cluster with FreeNAS shared storage. Please note that while this form of setup is good enough for learning purposes, it may not be enough for a production environment where uptime is essential. In later chapters, we will see how to add more Proxmox nodes and storage nodes into the cluster to ensure redundancy.




In this and the upcoming chapters, we will see the mighty power of Proxmox from inside out. We will deconstruct scenarios and create a very complex virtual environment, which will challenge us to think outside the box. We will also see some real incident-based issues and how to troubleshoot them. So strap yourself and let's dive into the virtual world with the mighty hypervisor, Proxmox. The following are some of the topics we are going to see in this chapter:

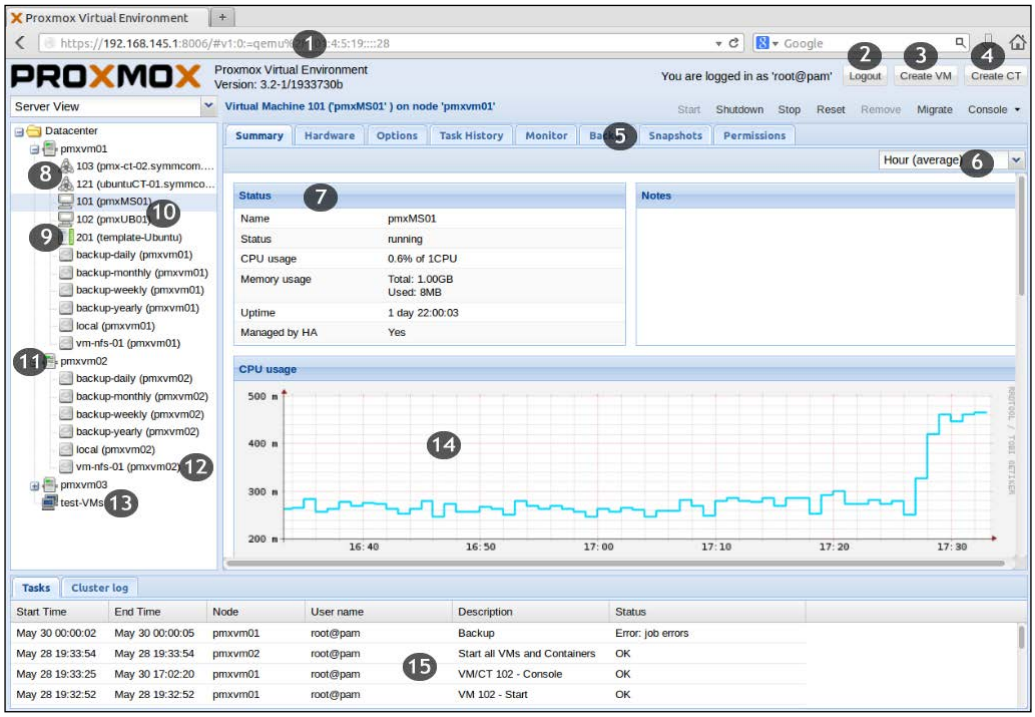
- Setting up a basic two-node Proxmox cluster
- Introduction to Proxmox **Graphical User Interface** (**GUI**)
- Setting up a virtual machine
- Proxmox virtual machine cloning and template

# Proxmox cluster required

A hands-on approach has been followed throughout this book to allow the reader to learn Proxmox in a practical way. If you do not have a Proxmox cluster set up or no access to an existing cluster, you can set up a basic-level Proxmox cluster by following the installation instructions laid out in the *Setting up a basic cluster* section of this chapter. If you already have a cluster, follow along from the next section, *The Proxmox Graphical User Interface (GUI)*.

# The Proxmox Graphical User Interface (GUI)

The Proxmox Graphical User Interface, or Proxmox GUI, allows users to interact with the Proxmox cluster graphically using menus and a visual representation of the cluster status. Even though all of the management can be done from the **Command-line Interface** (**CLI**), it can be overwhelming at times, and managing a cluster can become a daunting task. To properly utilize a Proxmox cluster, it is very important to have a clear understanding of the Proxmox GUI. The GUI can be easily accessed from just about any browser through a URL similar to `https://192.168.1.1:8006`, as shown in the following screenshot:

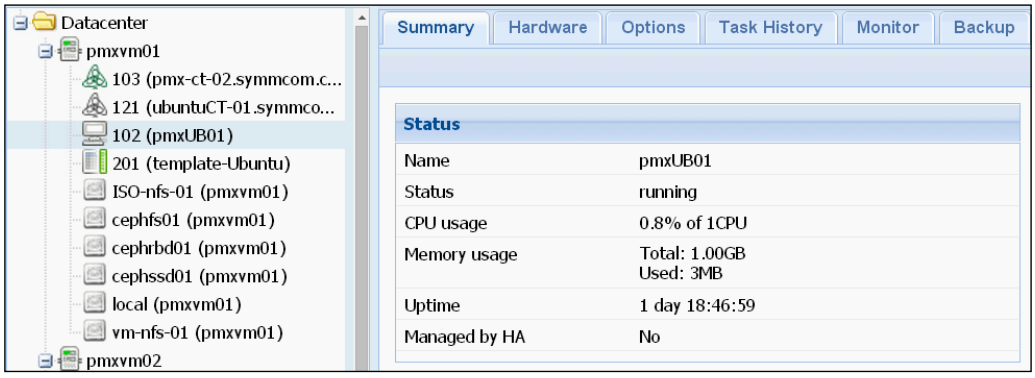


The various fields marked in the previous screenshot are as follows:

- **1** shows the URL to access the Proxmox GUI through a browser
- **2** shows the logout button to exit the Proxmox GUI
- **3** shows the button to open the virtual machine creation dialog box
- **4** shows the button to open the OpenVZ container creation dialog box
- **5** shows the Proxmox tabbed menu bar
- **6** shows the drop-down menu to change the period of the status graphs
- **7** shows the status information block for Proxmox nodes, virtual machines, or containers
- **8** shows the OpenVZ containers
- **9** shows the available virtual machine template for cloning
- **10** shows the KVM virtual machines
- **11** shows the Proxmox nodes
- **12** shows the shared storages
- **13** shows the resource pools
- **14** shows the graphical representation of various statuses
- **15** shows the task log

# The GUI menu system

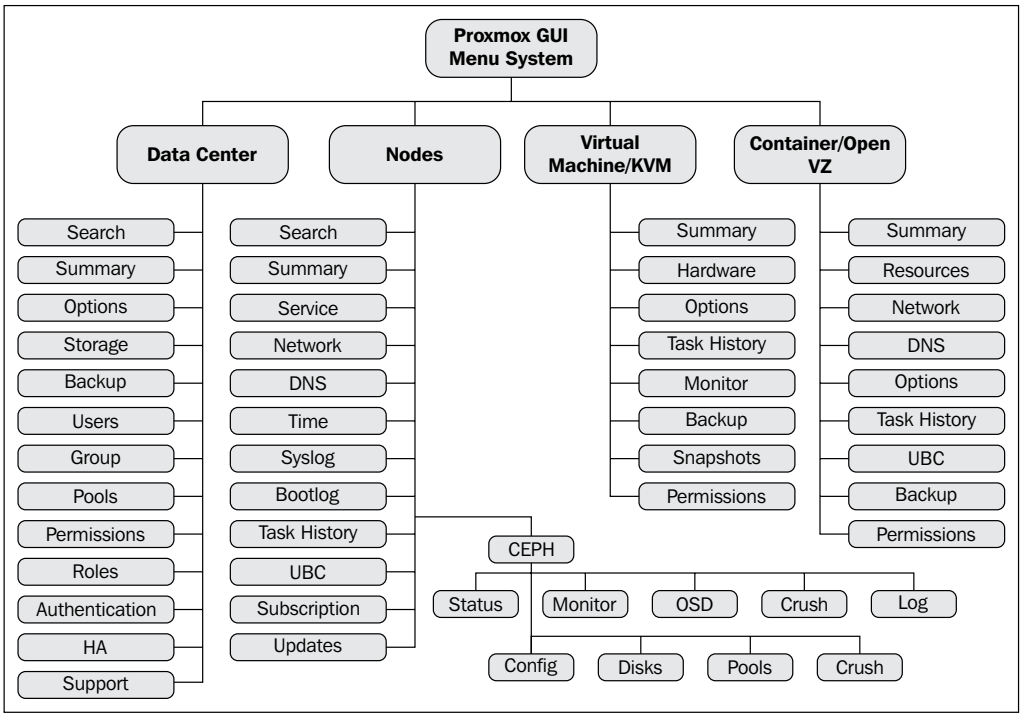
The Proxmox GUI is a one-page administration control panel. This means that no matter which feature one is managing, the browser does not open a new page or leave the existing page. Menus on the admin page change depending on the feature that is being administered. For example, in the previous screenshot, the node **pmxvm02** is selected, so the main menu only shows node-specific menus. When a virtual machine is selected, the menu looks like the following screenshot:



Some features of the Proxmox GUI, such as VNC console and shell, require Java or IcedTea (`http://icedtea.classpath.org/wiki/Main_Page`) to be installed on the computer you are accessing the GUI from. The GUI works great with Firefox and Google Chrome. The latest version of Internet Explorer may have an issue with functioning properly if not in compatibility mode. The Proxmox GUI also works with the Opera browser.

# Menu chart

The following chart is a visual representation of a Proxmox GUI menu system. Some menu options need to be set up once and do not need any regular attention, such as **DNS**, **Time**, **Services**, and so on. Other menus, such as **Summary**, **Syslog**, **Backup**, **Permissions**, and so on, are regularly used to ensure healthy cluster environment.




In this book, we will mostly look at the menu options relevant for regular maintenance of a Proxmox cluster. We will also look at some of the advanced menu options that are needed to create a complex network infrastructure, such as VLAN, bridge, and so on. The rest of the menu options are very basic in nature and are very much self-explanatory.

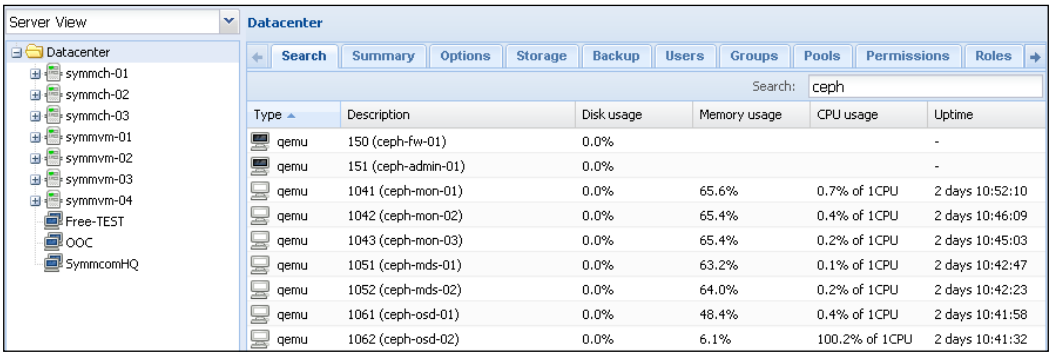
# The Datacenter menu

In the Proxmox GUI, Datacenter is the main level folder of Proxmox nodes'/VMs' tree. Each Datacenter folder can only hold one Proxmox cluster.

## The Search tab

To use the **Search** tab, navigate to **GUI** | **Server View** | **Datacenter** | **Search Tab** | **Search Box**.

It is very easy to manage a cluster with a small number of virtual machines with an even smaller number of Proxmox and storage nodes. When maintaining hundreds or thousands of virtual machines with several dozen Proxmox nodes in a cluster, the **Search** option makes it easier to find a particular virtual machine. Scrolling through a list of virtual machines to find a particular one is very time consuming. The following screenshot shows the **Search** option:

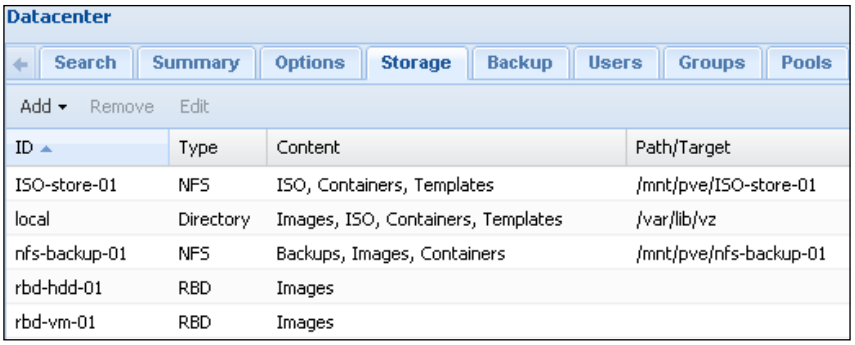



The search box under **Datacenter** | **Search** shows the result in real time as you type in the box. It can search with any string in the **Type** or **Description** columns. It can be the partial name of a VM, VMID, or VM type (qemu, openvz). The preceding screenshot shows all the virtual machines that have the word `ceph` in the description.

## The Storage tab

To use the **Storage** tab, navigate to **GUI** | **Server View** | **Datacenter** | **Storage**.

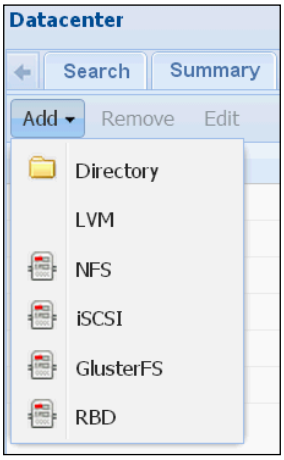
The **Storage** tab is probably one of the most important options in the Proxmox GUI. This is where the Proxmox cluster and storage systems come together. This is the only menu to attach any storage system with Proxmox. Whether it is local/shared storage, NFS/RBD/LVM, all are done here. The following screenshot shows the **Storage** tab:




As of Version 3.2, Proxmox supports the following storage types:

- Directory: This is mostly local storage
- LVM: These are local or shared iSCSI targets
- NF: This can be OmniOS, FreeNAS, Ubuntu, and so on
- GlusterFS: Visit `www.gluster.org` for more information
- RBD: Visit `www.ceph.com` for more information

In this book, we will be using NFS for basic low-end setup and RBD for advanced, distributed storage with a high level of redundancy. The following screenshot shows the storage types:

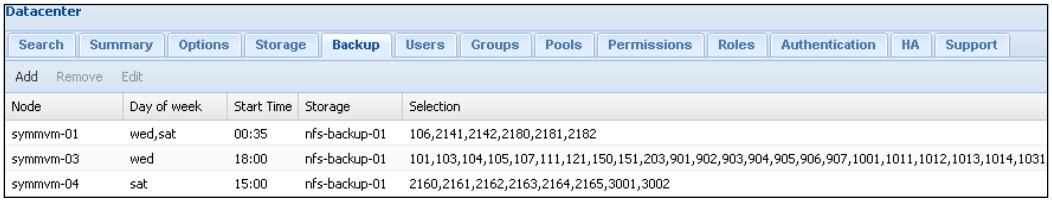


Although storage can be changed at a later time, it is very important to have a clear understanding of all storage types for better planning in the beginning. All storage types have both advantages and disadvantages. The ultimate decision will come down to performance, stability, and redundancy.

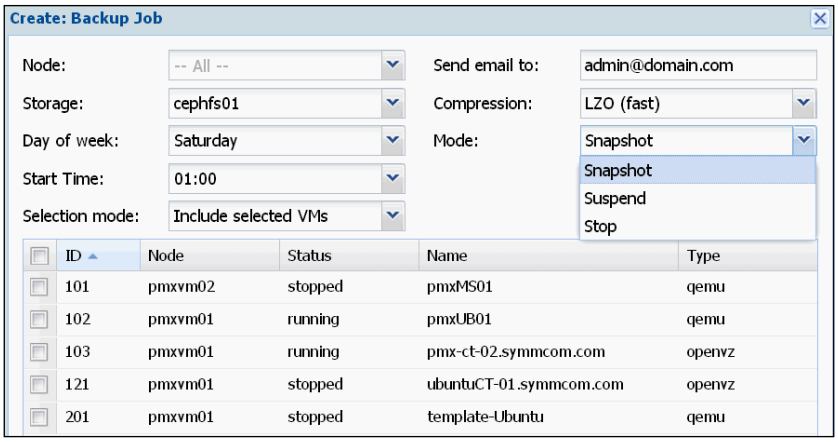
## The Backup tab

Cluster-wide backup schedules are created through this menu. Backup is the first line of defense against any form of cluster disaster. With a good backup plan, downtime can be minimized and valuable data can be saved. Although Proxmox backup system cannot do a granular file backup of a virtual machine, the ability to do full virtual machine backup is one of the strengths of Proxmox. This inclusion of a backup system is one of the best in the industry, and it "just works". The backup menu can be found in the following menu system.

To use the **Backup** tab, navigate to **GUI | Server View | Datacenter | Backup**. The following screenshot shows the **Backup** tab:




Proxmox only allows schedule creation on a daily and weekly basis. Select VMs to be backed up, day of the week, and time of the day, and backup will do the job on its own. The following screenshot shows the dialog box to create a schedule:




LZO compression and Snapshot mode are default in the Proxmox backup. Compression can be selected as **None**, **LZO**, and **GZIP**. In most cases, LZO works great. It has less compression but it is fast and easier on the hardware. GZIP can compress further, but it also takes up lot of CPU resources during backup.

The **Snapshot** mode allows live backup without needing to shut down running VM, thus minimizing downtime. In an always-on network environment, downtime may not be permitted. Other modes, such as **Suspend** and **Stop** may be used in special cases where shutting down the VM during backup is absolute necessary to ensure data integrity.

Please note that this **Snapshot** mode is not the same as the Snapshots option for a virtual machine. During full backup of a live virtual machine, LVM Snapshot is used, whereas Live Snapshots are used to preserve the state of a KVM-based virtual machine. Live Snapshots can be done for the OpenVZ container in Proxmox.

If the selected VMs are scattered over multiple nodes, it is very important to keep in mind that when backup starts at the scheduled time, it will simultaneously create a backup of VMs on multiple nodes to a single backup storage. If the backup storage is not powerful enough to handle all the incoming data from multiple Proxmox nodes, the backup process may fail.

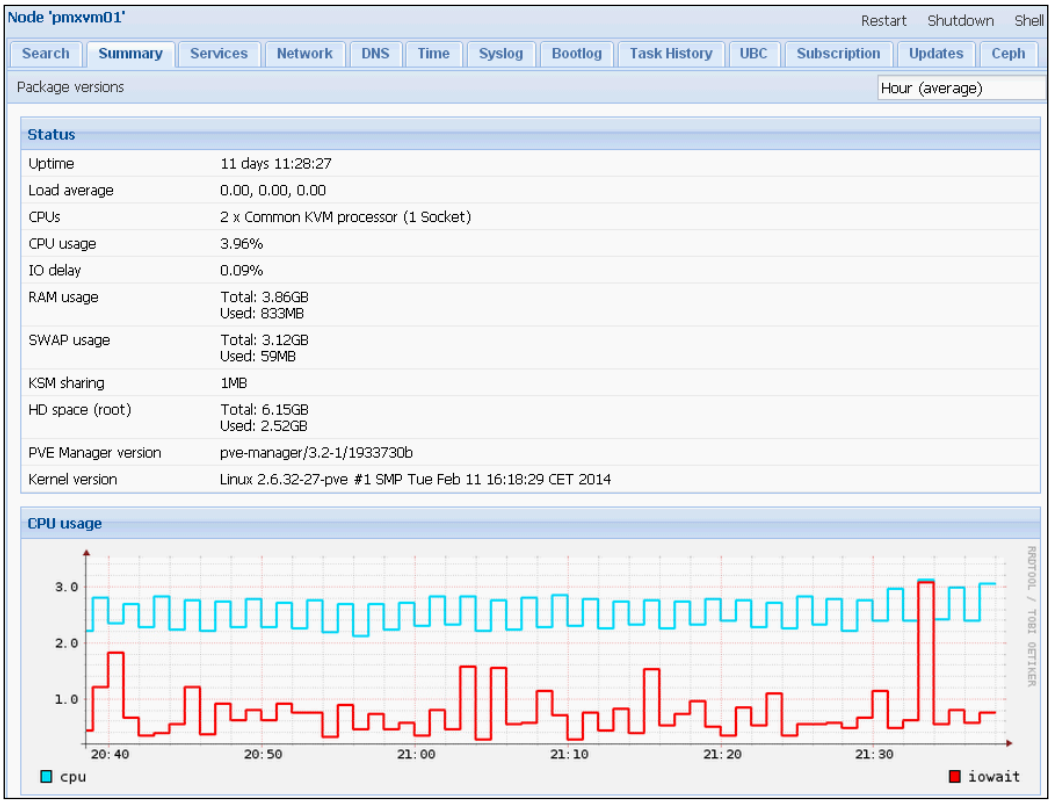
# Node-specific tabs

The node-specific tabs are specific to each node in the cluster. New menu tabs become visible when the node is selected.

## The Summary tab

To use the **Summary** tab, navigate to **GUI** | **Server View** | **Node** | **Summary**.

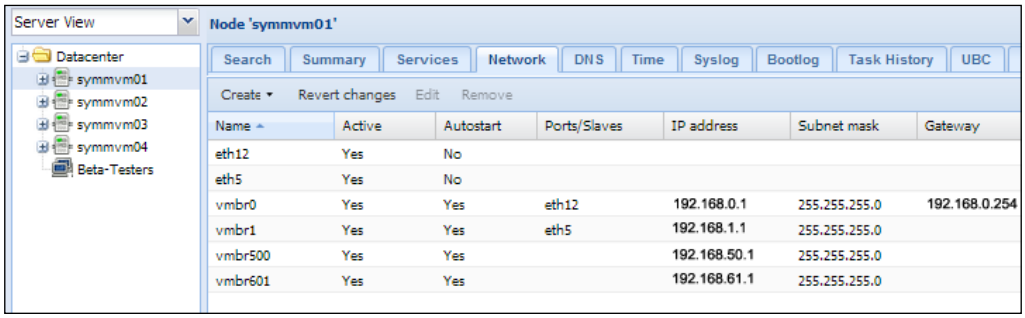
The **Summary** tab for a node is a visual representation of the node's health. It shows vital information, such as **Uptime** and **Resource Consumption**. As you can see in the following screenshot, the **Summary** screen also shows **CPU usage**, **Server Load**, **Memory Usage**, and **Network Traffic** in a very easy-to-understand graph. An administrator can get the necessary information about a node just by glancing at the summary. Summary can be viewed on hourly, daily, weekly, monthly, and yearly bases.



# The Network tab

To use the **Network** tab, navigate to **GUI** | **Server View** | **Node** | **Network**.

The **Network** menu acts as glue between all virtual machines, nodes, and shared storage systems. Without a proper **Network Interface Card** (**NIC**) or **Virtual NIC** (**vNIC**) and a virtual bridge setup, no communication can take place. Deeper understanding of this menu will allow you to create a very complex web of clusters, nodes, and virtual machines. Due to the importance of this menu option, we will look into this menu in greater detail later in this chapter.



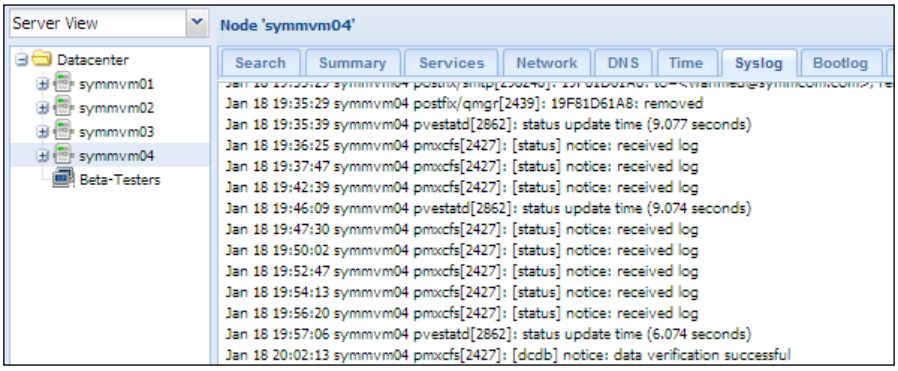

The concept of virtual network depends on the building block of the virtual bridge, virtual vNIC, and virtual LAN. Network virtualization is the future transformation of physical networks as server virtualization had been for physical servers.

# The Syslog tab

To use the **Syslog** tab, navigate to **GUI** | **Server View** | **Node** | **Syslog**.

The **Syslog** option allows an administrator to view the system log in real time. Syslog gives feedback as it happens in the node. It also allows scrolling up to view logs in the past. More importantly, if any error occurs in the node, Syslog gives that information in real time with the time and date stamp. This helps to pinpoint an issue exactly when it occurred. Here's an example of a **Syslog** menu visit scenario: if the node cannot connect to a storage system, the **Syslog** screen will show the error that is preventing connection.

The following screenshot shows the **Syslog** option:

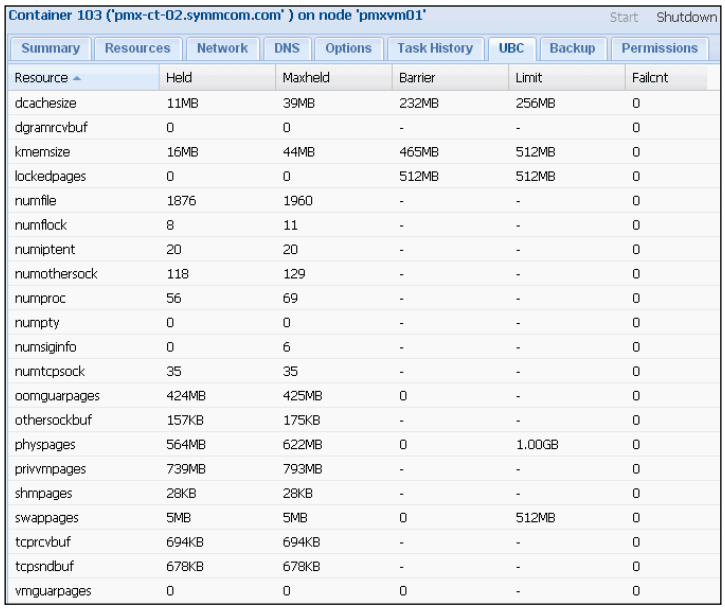



# The UBC tab

To use the **UBC** tab, navigate to **GUI** | **Server View** | **Node** | **UBC**.

**User Bean Counters** (**UBC**) is a set of limits, which guarantees resources control per container. This is a vital component for OpenVZ/container resource management. The **UBC** menu option in the Proxmox GUI is for viewing only. There is no option to edit any of the limits.

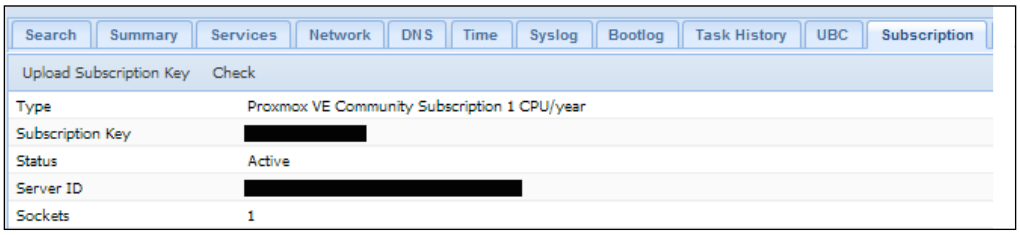
The **UBC** screen only gets populated when an OpenVZ container is selected, as shown in the following screenshot:




| Resource | Held | Maxheld | Barrier | Limit | Failcnt |
|---|---|---|---|---|---|
| dcachesize | 11MB | 39MB | 232MB | 256MB | 0 |
| dgramrcvbuf | 0 | 0 | - | - | 0 |
| kmemsize | 16MB | 44MB | 465MB | 512MB | 0 |
| lockedpages | 0 | 0 | 512MB | 512MB | 0 |
| numfile | 1876 | 1960 | - | - | 0 |
| numflock | 8 | 11 | - | - | 0 |
| numiptent | 20 | 20 | - | - | 0 |
| numothersock | 118 | 129 | - | - | 0 |
| numproc | 56 | 69 | - | - | 0 |
| numpty | 0 | 0 | - | - | 0 |
| numsiginfo | 0 | 6 | - | - | 0 |
| numtcpsock | 35 | 35 | - | - | 0 |
| oomguarpages | 424MB | 425MB | 0 | - | 0 |
| othersockbuf | 157KB | 175KB | - | - | 0 |
| physpages | 564MB | 622MB | 0 | 1.00GB | 0 |
| privvmpages | 739MB | 793MB | - | - | 0 |
| shmpages | 28KB | 28KB | - | - | 0 |
| swappages | 5MB | 5MB | 0 | 512MB | 0 |
| tcprcvbuf | 694KB | 694KB | - | - | 0 |
| tcpsndbuf | 678KB | 678KB | - | - | 0 |
| vmguarpages | 0 | 0 | 0 | - | 0 |

# The Subscription tab

To use the **Subscription** tab, navigate to **GUI** | **Server View** | **Node** | **Subscription**. Proxmox can be downloaded and used for free without any restriction for any feature. It is by no means a trialware, shareware, or an *n*-day evaluation hypervisor. However, Proxmox also has a subscription model, which allows enterprise-class repositories. The free version of Proxmox only comes with standard repositories. The main difference between enterprise and standard repositories is that enterprise repositories go through a higher level of testing to ensure a very stable cluster environment. The following screenshot shows the **Subscription** tab:




Keep in mind that even with the free version, Proxmox is still very stable. Do not let the subscription level fool you to think the free version is not even worth considering.

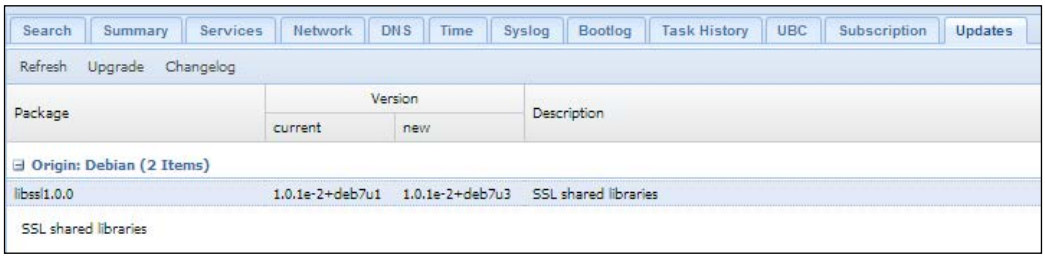
This level of tests is mandatory for an enterprise-class network environment where a small issue can cost a company a lot of money. A highly stable environment is usually not needed in a home-based platform or small business environment. The **Subscription** tab allows activating purchased subscription on a node.

Proxmox has the very best prices per subscription in the virtualization product industry. The operating cost of Proxmox cluster is very minimal compared with a giant virtual product such as VMware. Proxmox provides big business virtualization at small business cost. For details of different subscription levels, visit the link `http://proxmox.com/proxmox-ve/pricing`.

# The Updates tab

To use the **Updates** tab, navigate to **GUI | Server View | Node | Updates**.

The Proxmox node can be updated right from the GUI through the **Updates** tab. Each node checks daily for any available updates and alerts administrator through e-mail if there are any new updates. It is important to keep all nodes up to date by updating regularly. The **Updates** menu enables upgrading by just using a few mouse clicks. The following screenshot shows the **Updates** tab:

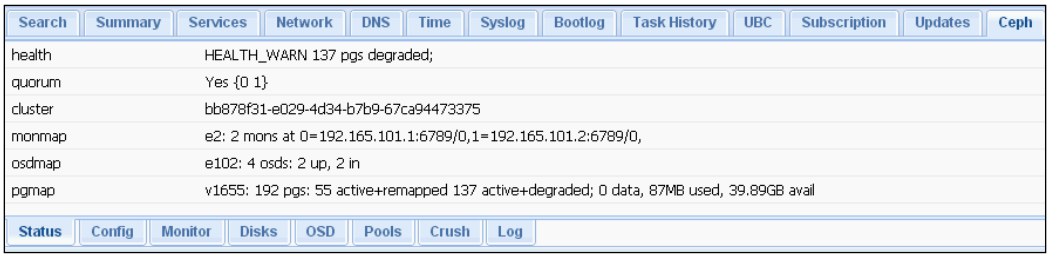



Always update one node at a time. Some updates require the node to be restarted. If uptime is important, then migrate all running virtual machines to a different node before restarting the upgraded node.

# The Ceph tab

Ceph is a robust and powerfully distributed storage system that can be used as shared storage for Proxmox cluster. Ceph provides the **RADOS Block Device** (**RBD**) storage backend. A Ceph storage cluster can scale out to several petabytes. Ceph is powerful enough to handle infrastructure of any size while being resilient enough to provide great storage redundancy. Understanding the true potential of Ceph, we have dedicated an entire chapter in this book to show you how to set up a Ceph cluster using both command line and the Proxmox GUI to build truly enterprise class complex virtual infrastructures.

Starting with Proxmox VE 3.2, Ceph server is added as technology preview. This allows both Proxmox and Ceph to co-exist on the same node. Ceph itself does not come with any graphical user interface to manage Ceph storage, with the exception being the subscription version of Ceph. Proxmox enables us to manage Ceph cluster almost entirely from the Proxmox GUI. Currently CrushMAP cannot be edited and multiple Ceph clusters cannot be managed through the GUI. The following screenshot shows the **Ceph** menu tab along with Ceph-related tabs:




We will look into the Ceph tab in greater detail in *Chapter 7, High Availability Storage for High Availability Cluster*. The following list is a short description of Ceph-related tabs and their functions:

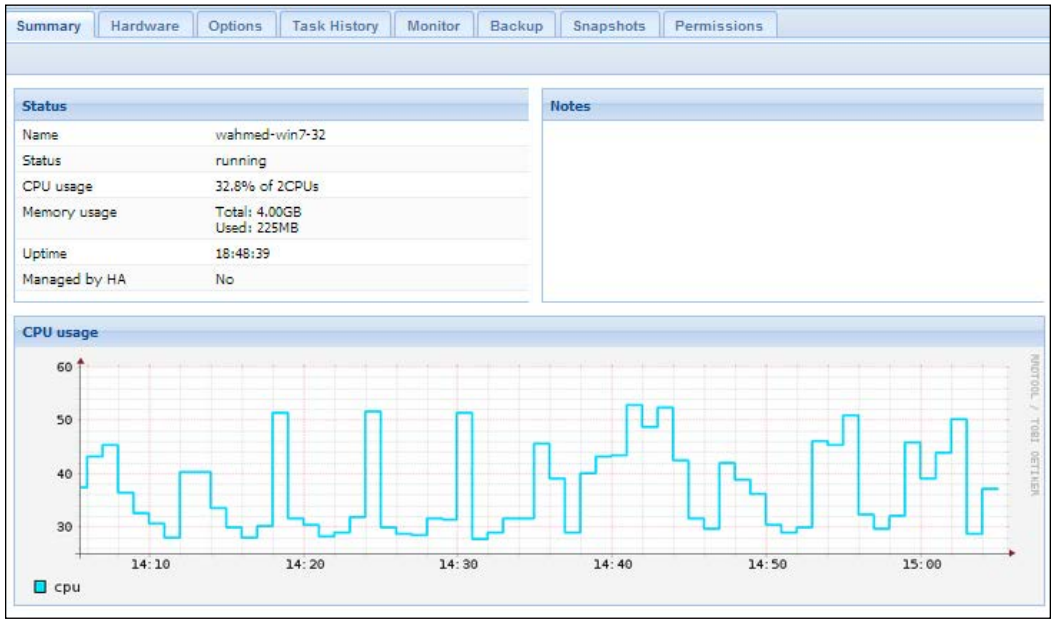
- **Status**: This shows the current status or health of a Ceph storage
- **Config**: This displays the content of the Ceph cluster configuration file `cluster.conf`
- **Monitor**: This starts, stops, creates, removes, and displays a list of Ceph Monitors (MONs)
- **Disks**: This displays the available drives attached to the Proxmox nodes and creates new OSDs
- **OSD**: This manages OSDs of Ceph clusters
- **Pools**: This creates, removes, and displays the list of pools
- **Crush**: This displays the content of CrushMAP
- **Log**: This displays the Ceph cluster log in real time

# Virtual machine tabs

The following menu tabs are available when a virtual machine is selected.

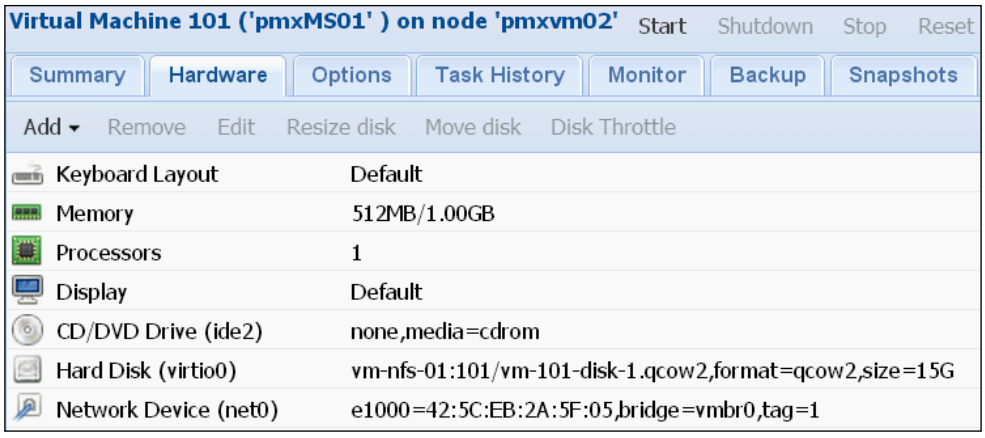
## The Summary tab

The **Summary** menu tab represents similar information such as the one accessed by navigating to **Node | Summary**. Valuable information can be gathered, which shows the real-time status of a virtual machine. One additional feature this menu has is the addition of the **Notes** textbox. By double-clicking on the **Notes** box, it brings up a multiline textbox where an administrator can enter data such as the department, the usage the VM is intended for, or just about any other information that needs to be on hand.

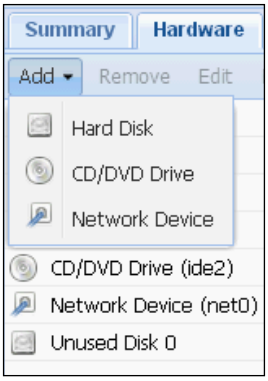



## The Hardware tab

The initially created and configured virtual machine sometimes needs further resource allocation. As the functions of VM rise, it becomes necessary to add additional virtual drives or network interfaces. The **Hardware** menu tab under the virtual machine is where the adding and removing of devices happens. The following screenshot shows the **Hardware** tab:




Through the **Add** menu, additional CD drives, hard drives, and network interfaces (bridge, vNIC, and so on) can be added to a virtual machine, as shown in the following screenshot:




Each of these additions requires the virtual machine to be fully powered off and not just the restart/reboot process. Ejecting an ISO image file to attach a different one does not require any VM power cycle. By adding some configuration arguments in the virtual machine configuration file, it is possible to hot swap a virtual hard disk into a VM. This configuration is further explained in *Chapter 4*, *A Virtual Machine for a Virtual World*.

Besides the **Add** menu, other menus such as **Remove**, **Edit**, **Resize Disk**, and **Move Disk** are also available through the **Hardware** menu. All these additional menus except **Add** require a hardware item to be selected. **Resize Disk** and **Move Disk** will be enabled for clicking when a virtual drive is selected. We will see these in greater detail in later chapters.

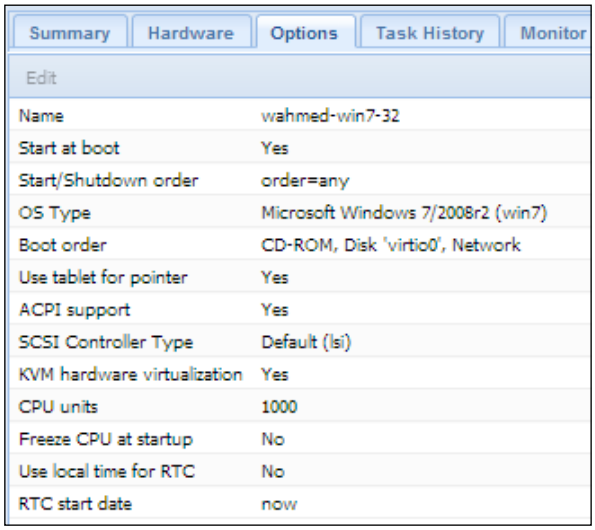
**Move Disk** is the safest way to move a virtual hard drive from one storage to another. If the virtual disk is on a shared storage, then live migration of the virtual disk is possible, allowing a great amount of time saving.

## The Options tab

The **Options** menu under virtual machine allows further tweaking, such as changing name, boot order, and so on. Most of the options here can be left to default.

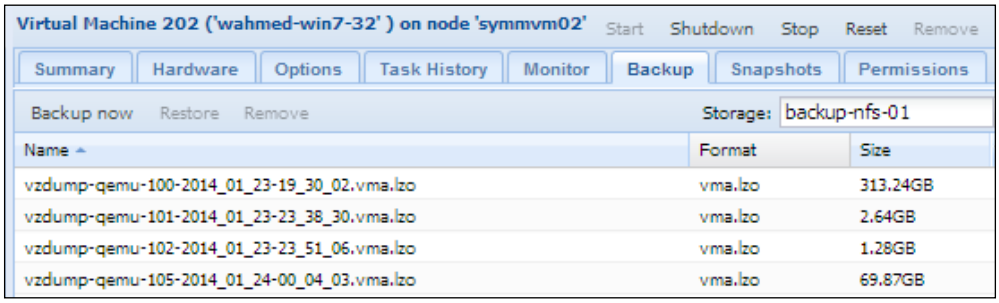
If you want the virtual machine to auto-start as soon as the Proxmox node reboots, set the **Start at boot** option to **Yes**.

The **Options** tab is shown in the following screenshot:



# The Backup tab

A good backup plan is the first line of defense against any disaster, which can cause major or minor data loss. In our ultra-modern digital world, data is much more valuable than ever before. Every virtual environment administrator struggles with backup strategy of his/her virtual environment. The following screenshot shows the **Backup** tab:



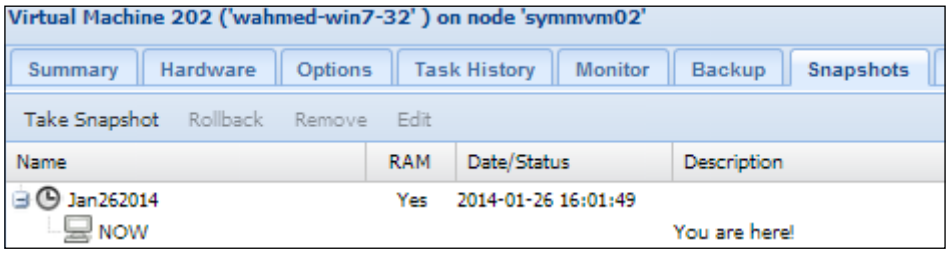

The fine line between granular files and an entire machine backup is somewhat diminished in a virtual environment. To take the daily struggle of backup plan out of the equation, Proxmox added an excellent backup system right in the hypervisor itself. Although the backup system cannot backup individual files inside a virtual machine, it works well while backing up an entire virtual machine.

Proxmox backup system can only do full backup of a virtual machine and cannot be used to backup individual files inside the virtual machine at the granular level.

Proxmox backups can be scheduled over multiple storage systems and multiple days. A backup system is only as good as the ability to restore the backup. Both backup and restore can be done from single menu under virtual machine. It also allows backups browsing and manual deletion of any backups. All these are done from a single interface with a few mouse clicks. Due to the importance of backup strategy in a virtual environment, we will look into Proxmox backup system in much greater detail in *Chapter 4*, *A Virtual Machine for a Virtual World*.

# The Snapshots tab

Proxmox Snapshots is a way to roll back a KVM-based virtual machine to a previous state. Although it provides similar protection to Proxmox Backup, it comes with speed. Proxmox Snapshot is extremely fast when compared with Proxmox Backup, thus allowing a user to take several snapshots a day. The following screenshot shows the **Snapshots** tab:

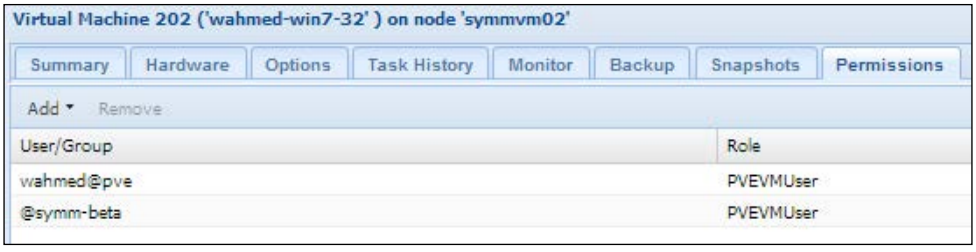



A common scenario where Snapshots can be used is when a user wants to install or update a software. He or she can take a snapshot, execute the program, and if anything goes wrong, then simply roll back to the previous state. It creates Snapshot with the RAM itself, so the virtual machine stays exactly the same as it is running. Live snapshots are not included in full virtual machine backups.

Never depend solely on Snapshots. Snapshots are not a full backup. It is merely a state when the virtual machine is frozen in time. Always do a full backup of virtual machines for maximum protection.

# The Permissions tab

The Permissions menu allows the management of user permissions for a particular virtual machine. It is possible to give multiple users access to the same virtual machine. Click on **Add** to add users or groups to the permission. The following screenshot shows the **Permissions** tab:



A common scenario of permission usage is in an office setup where there is one accounting virtual machine and multiple staff need to access data. A permission can be set either at the user or the group level.

# Setting up a basic cluster

This section will help you to create a shopping list of components that you need and provide you step-by-step instructions to set up a basic Proxmox cluster. The steps in this section are in a much simpler form to get you up and running quickly. You can see Proxmox setup instructions in greater details from Proxmox Wiki documentation at `http://pve.proxmox.com/wiki/Installation`.

To set up a shared storage to be used with a Proxmox cluster, we are going to use Ubuntu or FreeNAS storage. There are options other than FreeNAS, such as OpenMediaVault, NAS4Free, GlusterFS, and DRBD to name a few. FreeNAS is an excellent choice for shared storage due to its ZFS filesystem implementation, simplicity of installation, large active community, and no licensing cost. Although we have used FreeNAS in this book, you can use just about any flavor of shared storage with the NFS and iSCSI support you want. Installation guide to set up FreeNAS is beyond the scope of this book.

For complete setup instructions for FreeNAS, visit `http://doc.freenas.org/index.php/Installing_from_CDROM`.

Ubuntu is also a great choice to learn how shared storage works with Proxmox. Almost anything you can set up with FreeNAS, you can also set up in Ubuntu. The only difference is that there is no user-friendly graphical interface in Ubuntu as in FreeNAS. Deeper into the book, we will look at the ultimate shared storage solution using Ceph. But to get our first basic cluster up and running, we will use Ubuntu or FreeNAS to set up an NFS and iSCSI share.

For installation instructions for Ubuntu server, visit `http://www.ubuntu.com/download/server`.

# The hardware list

The following is a list of hardware components that we will need to put together our first basic Proxmox cluster. If you already have some components you would like to use to set up your cluster, it is important to check if they will support virtualization. Not all hardware platforms support virtualization, especially if they are quite old. To get details on how to check your components, visit `http://virt-tools.org/learning/check-hardware-virt/`.

A quicker way to check is through the BIOS and look for one of the following settings in the BIOS option. Any one of the these should be Enabled in order for the hypervisor to work.

- Intel ® Virtualization Technology
- Virtualization Technology (VTx)
- Virtualization

This list of hardware is to build a bare minimum Proxmox cluster for learning purposes only and not suitable for enterprise-class infrastructure.

| Component type | Brand/model | Quantity |
|---|---|---|
| CPU/Processor | Intel i3-2120 3.30 Ghz 4 Core | 2 |
| Motherboard | Asus P8B75-M/CSM | 2 |
| RAM | Kingston 8 GB 1600 Mhz DDR3 240 Pin Non-ECC | 3 |
| HDD | Seagate Momentus 250 GB 2.5" SATA | 2 |
| USB stick | Patriot Memory 4 GB | 1 |
| Power supply | 300+ Watt | 3 |
| LAN switch | Netgear GS108NA 8-Port Gigabit Switch | 1 |

# The software list

Download the software given in the following table in the ISO format from their respective URL, and then create a CD from the ISO images.

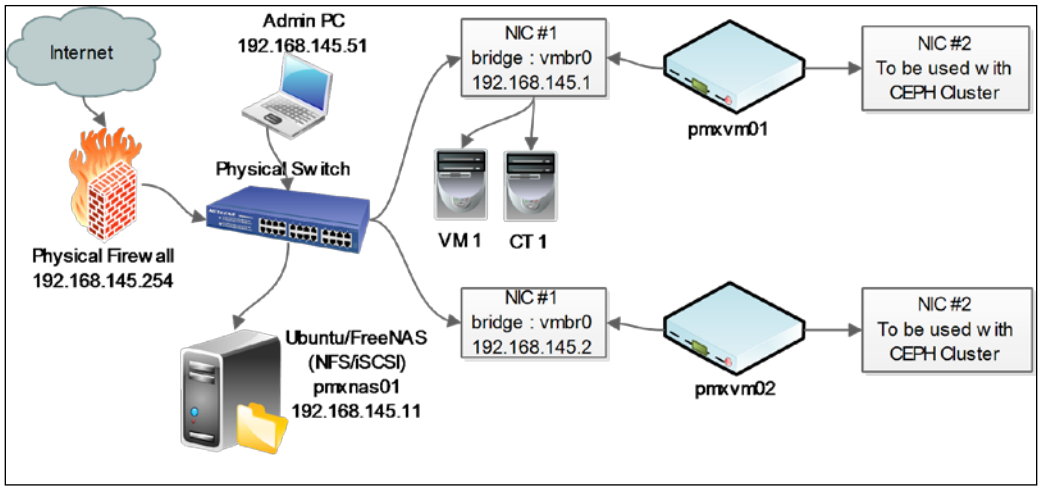
| Software | Download link |
|---|---|
| Proxmox VE | `http://proxmox.com/downloads` |
| FreeNAS | `http://www.freenas.org/download-releases.html` |
| Ubuntu Server | `http://www.ubuntu.com/download` |
| clearOS community | `http://www.clearfoundation.com/Software/downloads.html` |

# Hardware setup

The next diagram is a network diagram of a basic Proxmox cluster. We will start with two node clusters with one shared storage setup with either Ubuntu or FreeNAS. The setup in the illustration is a guideline only. Depending on the level of experience, budget, and available hardware on hand, you can set up any way you see fit. Regardless of whatever setup you use, it should meet the following requirements:

- Two Proxmox nodes with two Network Interface Cards
- One shared storage with NFS and iSCSI connectivity
- One physical firewall
- One 8+ port physical switch
- One KVM virtual machine
- One OpenVZ/container machine

This book is intended for beyond-beginner-level user and, therefore, full instruction of the hardware assembly process is not detailed here. After connecting all equipment together, it should resemble the following diagram:




# Proxmox installation

Perform the following simple steps to install Proxmox VE on Proxmox nodes:

1. Assemble all three nodes with proper components, and connect all of them with a LAN switch.
2. Power up the first node and access BIOS to make necessary changes such as enabling virtualization.

3. Boot the node from the Proxmox installation disc.
4. Follow along the Proxmox graphical installation process. Enter the IP address `192.168.145.1`, or any other subnet you wish, when prompted. Also enter `pmxvm1.domain.com` or any other hostname that you choose to use.
5. Perform step 3 and 4 for second node. Use IP address `192.168.145.2` or any other subnet. Use `pmxvm2.domain.com` as hostname or any other hostname.

# Cluster creation

We are now going to create a Proxmox cluster with two Proxmox nodes we just installed. From admin PC, (Linux/Windows), log in to Proxmox node #1 (**pmxvm01**) through secure login. If the admin PC is Windows based, use program such as PuTTY to remotely log in to Proxmox node.

Download PuTTY from `http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html`.

Linux users use the following command to securely log in to Proxmox node:

```
# ssh root@192.168.145.1
```

After logging in, it is now time to create our cluster. The command to create a Proxmox cluster is as follows:

```
# pvecm create <cluster_name>
```

This command can be executed on any of the Proxmox nodes but only once.

Never run a cluster creation command on more than one node in the same cluster. The cluster creation process must be completed on one node before adding nodes to the cluster.

The cluster does not operate on master/slave basis, but on Quorum. In order to achieve healthy cluster status, all nodes need to be online. Let's execute the following command and create the cluster:

```
root@pmxvm01:~# pvecm create pmx-cluster
```

The preceding command will display the following messages on the screen as it creates a new cluster and activates it:

```
Restarting pve cluster filesystem: pve-cluster[dcdb] notice: wrote new
cluster config '/etc/cluster/cluster.conf'
Starting cluster:
  Checking if cluster has been disabled at boot. . . [  OK  ]
  Checking Network Manager. . . [  OK  ]
  Global setup. . . [  OK  ]
  Loading kernel modules. . . [  OK  ]
  Mounting configfs. . . [  OK  ]
  Starting cman. . . [  OK  ]
  Waiting for quorum. . . [  OK  ]
  Starting fenced. . . [  OK  ]
  Starting dlm_controld. . . [  OK  ]
  Tuning DLM kernel config. . . [  OK  ]
  Unfencing self. . . [  OK  ]
root@pmxvm01:~#
```

After cluster creation is complete, check its status by using the following command:

```
root@pmxvm01:~# pvecm status
```

The preceding command will display the following output:

```
Version: 6.2.0
Config Version: 1
Cluster Name: pmx-cluster
Cluster ID: 23732
Cluster Member: Yes
Cluster Generation: 4
Membership status: Cluster-Member
Nodes: 1
Expected votes: 1
Total votes: 1
Quorum: 1
Active subsystem: 5
Flags:
Ports Bound: 0
```

```
Node name: pxvm01
Node ID: 1
Multicast addresses: 239.192.92.17
Node addresses: 192.168.145.1
root@pmxvm01:~#
```

The status shows some vital information that is needed to see how the cluster is doing and what the other member nodes of the cluster are. Although from Proxmox GUI we can visually see cluster health, command-line information gives a little bit more in-depth picture.

After the cluster has been created, the next step is to add Proxmox nodes into the cluster. Securely log in to the other node and run the following command:

```
root@pmxvm02:~# pvecm add 192.168.145.1
```

Verify that this node is now joined with the cluster with the following command:

```
root@pmxvm02:~# pvecm nodes
```

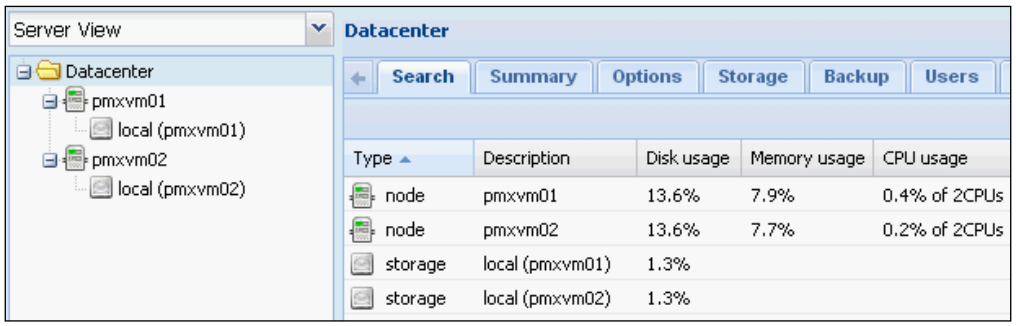
It should print the following node list that are member of the cluster we have just created:

```
Node   Sts   Inc     Joined                  Name
   1   M    4900     2014-01-26 16:02:34     pmxvm01
   2   M    4774     2014-01-26 16:12:19     pmxvm02
```

The next step is to log in to Proxmox Web GUI to see the cluster and attach shared storage. Use the URL in the following format from a browser on the admin computer to access the Proxmox graphical user interface:

```
https://<ip_proxmox_node>:8006
```

The cluster should look similar to the following screenshot:



# Proxmox subscription

On a clean installed Proxmox node, a paid subscription-based repository is enabled by default. When you log in to the Proxmox GUI, the following message will pop up on entering login information:


If you want to continue using Proxmox without subscription, perform the following steps to remove the enterprise repository and enable a subscription-less repository. This needs to be done on all Proxmox nodes in the cluster.

1. Run the following command:

   ```
   # nano /etc/apt/sources.list.d/pve-enterprise.list
   ```

2. Comment out enterprise repository as follows:

   ```
   #deb https://enterprise.proxmox.com/debian wheezy pve-
     enterprise
   ```

3. Run the following command:

   ```
   # nano /etc/apt/sources.list
   ```

4. Add a subscription-less repository as follows:

   ```
   deb http://download.proxmox.com/debian wheezy pve-no-
     subscription
   ```

# Attaching shared storage

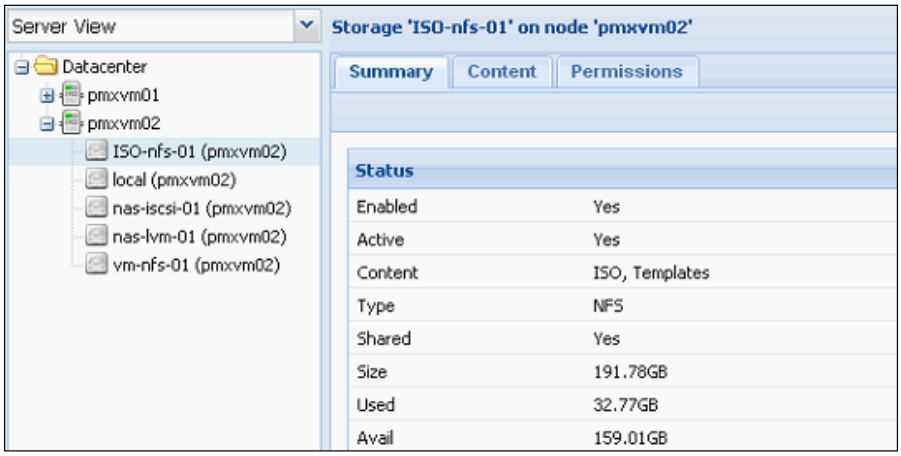
A Proxmox cluster can function with local storage just fine. But shared storage has many advantages over local storage, especially when we throw migration and disaster-related downtime in the mix. Live migration while a virtual machine is powered on is not possible without shared storage. We will start our journey into Proxmox with NFS/iSCSI shared storage, such as Ubuntu or FreeNAS.

Take a pause reading right here, and set up the third node with Ubuntu or FreeNAS. Both websites Ubuntu and FreeNAS have complete instructions to get you up and running.

After you have the choice of shared storage server setup, attach the storage with Proxmox by navigating to **Datacenter | Storage**. There should be three shares as shown in the following table:

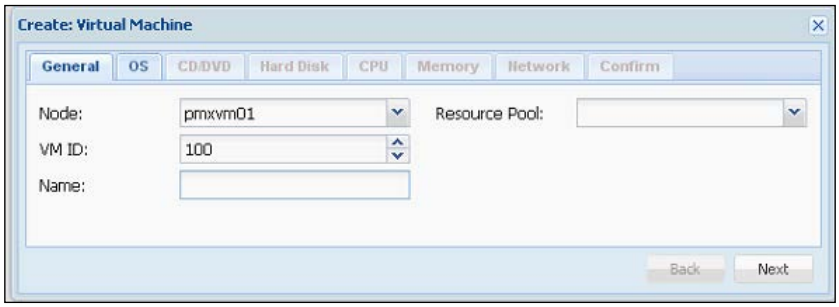
| Share ID | Share type | Content | Purpose |
|---|---|---|---|
| **ISO-nfs-01** | NFS | ISO, templates | To store ISO images |
| **vm-nfs-01** | NFS | Image, containers | To store VM with the `qcow2 vmdk` image and containers |
| **nas-lvm-01** | iSCSI | Image | To store a raw VM |

After setting up both the NFS and iSCSI shares, the Proxmox GUI should look like the following screenshot:




# Adding virtual machines

With our cluster up and running, it is time to add some virtual machines to it. Click on **Create VM** to start a KVM virtual machine creation process. The option window to create a virtual machine looks like the following screenshot:



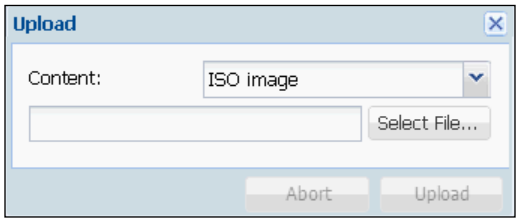
# Main virtual machine

The virtual machine we are going to create will act as main server for the rest of the virtual machines in the cluster. This will provide services such as DHCP, DNS, and so on. You can use any Linux flavor you are familiar with to get the DHCP/DNS Server set up. The ClearOS Community edition is a great choice since it allows putting all services in one machine and actually works very well.

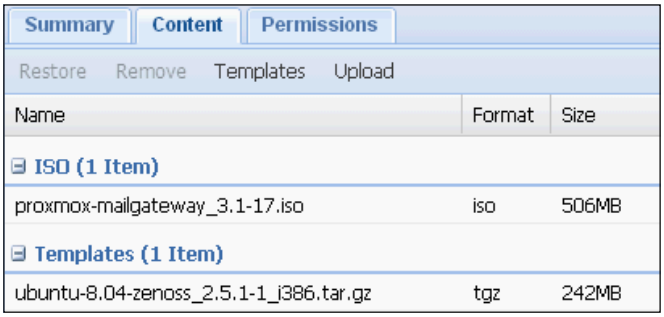
ClearOS is an open source, server-in-a-box Linux distribution, which means it can pull the weight of multiple servers/services in one setup. ClearOS is a Linux replacement of Windows Small Business Server. Learn more details and download it from `http://www.clearfoundation.com/Software/overview.html`.

Before creating a KVM-based virtual machine from scratch, we have to upload an ISO image of an operating system into Proxmox. This also applies to any ISO image we want a user to have access to, such as an ISO image of the installation disk for Microsoft Office or any other software. Not all storage types are supported to store ISO images. As of this writing, only local Proxmox storage, NFS, Ceph FS, and GlusterFS can be used to store ISO images. To upload an ISO image, perform the following steps:

1. Select proper storage from the **Datacenter** or **Storage** view on the Proxmox GUI.
2. Click on the **Content** tab.
3. Click on the **Upload** button to open the upload dialog box as shown in the following screenshot:



4. Click on the **Select File…** button to select the ISO image, and then click on the **Upload** button. After uploading, the ISO will show up on the content page as shown in the following screenshot:




Since the upload happens through the browser, it may cause a timeout error while uploading a large ISO file. In these cases, use a client program such FileZilla to upload the ISO image. Usually the Proxmox directory path to upload an ISO file is `/mnt/pve/<storage_name>/template/iso`.

After the ISO image is in place, we can proceed with KVM virtual machine creation using the configuration in the following table:

| VM creation tab | Specification | Selection |
|---|---|---|
| General | Node | pmxvm01 |
| | Virtual machine ID | 101 |
| | Virtual machine name | pmxMS01 |
| OS | Linux/other OS types | Linux 3.x/2.6 Kernel |
| CD/DVD | Use CD/DVD disc image file | ClearOS 6 Community |
| Hard disk | Bus/device | virtio |
| | Storage | vm-nfs-01 |
| | Disk size (GB) | 25 |
| | Format | QEMU image (qcow2) |
| CPU | Sockets | 1 |
| | Cores | 1 |
| | Type | Default (kvm64) |
| Memory | Automatically allocate memory within range | Max. 1024 MB<br>Minimum 512 MB |
| Network | Bridged mode | vmbr0 |
| | Model | Intel E1000 / VirtIO |

# Creating a KVM virtual machine

After the main server is set up, we are now going to create a second virtual machine with Ubuntu as the operating system. Proxmox has a cloning feature, which saves lot of time when deploying VMs with the same operating system and configuration. We will use the Ubuntu virtual machine as the template for all Linux-based VMs throughout this book. Create the Ubuntu VM with the following configuration:

| VM creation tab | Specification | Selection |
|---|---|---|
| General | Node | pmxvm01 |
| | Virtual machine ID | 201 |
| | Virtual machine name | template-Ubuntu |
| OS | Linux/other OS types | Linux 3.x/2.6 Kernel |
| CD/DVD | Use CD/DVD disc image file | Ubuntu server ISO |
| Hard disk | Bus/device | virtio |
| | Storage | vm-nfs-01 |
| | Disk size (GB) | 30 |
| | Format | QEMU image (qcow2) |
| CPU | Sockets | 1 |
| | Cores | 1 |
| | Type | Default (kvm64) |
| Memory | Automatically allocate memory within range | Maximum 1024 MB |
| | | Minimum 512 MB |
| Network | Bridged mode | vmbr0 |
| | Model | Intel E1000 / VirtIO |

# Creating an OpenVZ virtual machine

Now we will create one OpenVZ/container virtual machine. OpenVZ is container-based virtualization for Linux where all containers share the base host operating system. At this moment, only the Linux OpenVZ virtual machine is possible, and no Windows-based container. Although OpenVZ containers act as independent virtual machines, they rely heavily on the underlying Linux kernel of the hypervisor. All containers in a cluster share the same Linux kernel of the same version. The biggest advantage of the OpenVZ container is soft memory allocation where memory not used in one container can be used by other containers. Since each container does not have its own full version of the operating system, the backup size of containers is much smaller than the KVM-based virtual machine. OpenVZ is a great option for an environment such as a web hosting provider, where many instances can run simultaneously to host client sites.

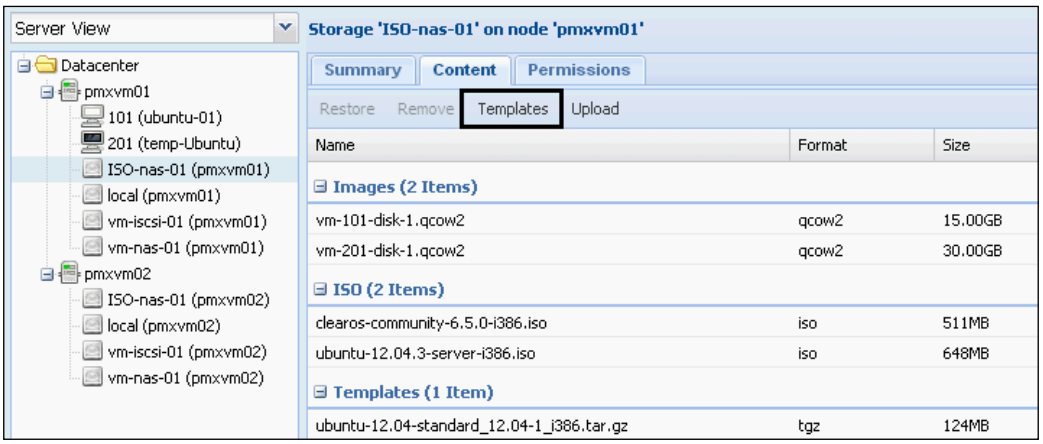
Go to `http://openvz.org/Main_Page` for more details on OpenVZ.

Unlike a KVM virtual machine, OpenVZ containers cannot be installed using an ISO image. Proxmox uses templates to create OpenVZ container virtual machines and comes with the very nice feature of templates repository. At the time of this writing, the repository has close to 400 templates ready to download through the Proxmox GUI.

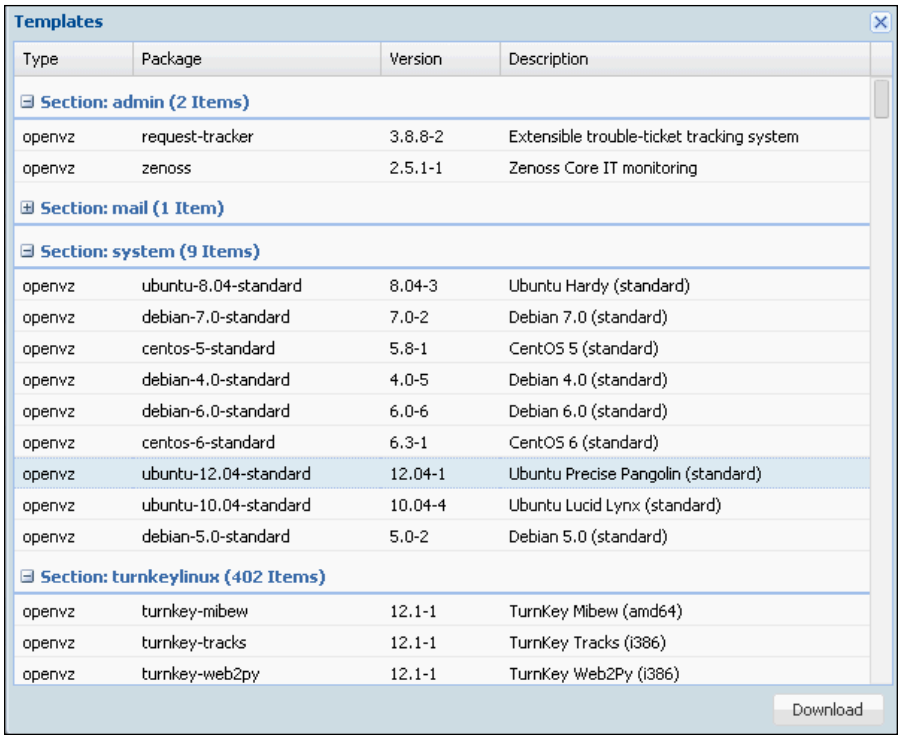
Templates could also be user-created with specific configurations. Creating your own template can be a difficult task and usually requires extensive knowledge of the operating system. To take the difficulties out of the equation, Proxmox provides an excellent script called **Debian Appliance Builder** (**DAB**) to create OpenVZ templates. Visit the following links before undertaking OpenVZ templates:

- `http://wiki.openvz.org/Category:Templates`
- `http://pve.proxmox.com/wiki/Debian_Appliance_Builder`

From the Proxmox GUI, click on the **Templates** button as shown in the following screenshot to open the built-in template browser dialog box and to download templates:




For our OpenVZ virtual machine lesson, we will be using the Ubuntu 12.04 template under **Section: system** as shown in the following screenshot:

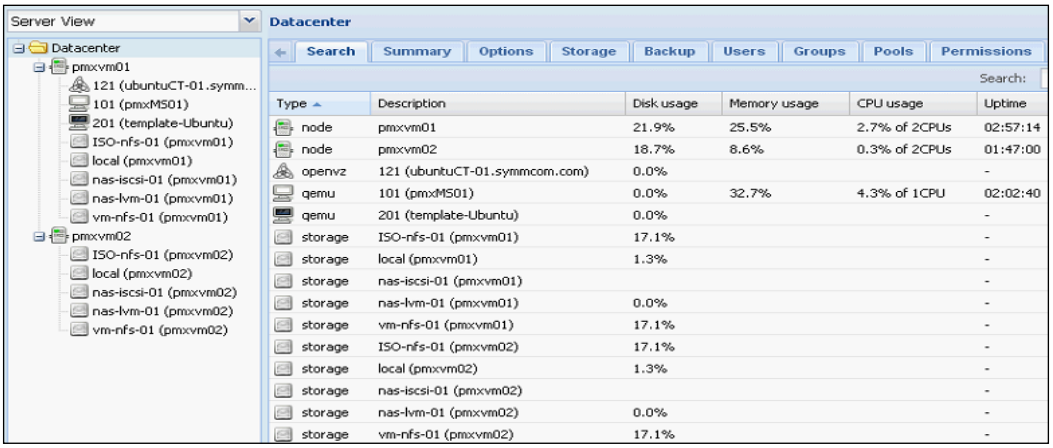



Create the OpenVZ container using the following specifications:

| OpenVZ creation tab | Specification | Selection |
|---|---|---|
| General | Node | pxvm01 |
| | Virtual machine ID | 121 |
| | Virtual machine hostname | ubuntuCT-01 |
| | Storage | vm-nfs-01 |
| | Password | any |
| Template | Storage | ISO-nfs-01 |
| | Template | ubuntu-12.04-standard |
| Resources | Memory | 1024 MB |
| | Swap | 512 MB |
| | Disk size (GB) | 30 |
| | CPUs | 1 |
| Network | Bridged mode | vmbr0 |

OpenVZ containers cannot be cloned for mass deployment. If such mass deployment is required, then the container can be backed up and restored with different VM IDs as many times as required.

With all three of the virtual machines created, the Proxmox cluster GUI should look like the following screenshot:




# Proxmox cloning/template

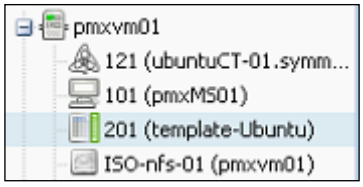
One of the great features of Proxmox is the ability to clone a virtual machine for mass deployment. It saves an enormous amount of time while deploying virtual machines with similar operating systems.

## Introducing cloning using a template

It is entirely possible to clone a virtual machine without ever creating a template. The main advantage of creating a template is virtual machine organizing within the cluster.

The template for cloning is not the same as the template required to create OpenVZ/container.

A template has a distinct icon as seen in the following screenshot, which easily identifies it from a standard virtual machine. Just create a VM with desired configuration, and then by the touch of a mouse click, turn the VM into a template. Whenever a new VM is required, just clone the template.




For a small cluster with few virtual machines, it is not an issue. But an enterprise cluster with hundreds, if not thousands, of virtual machines, finding the right template can become a tedious task. By right-clicking on a VM, you can pull up a context menu, which shows the option related to that VM.

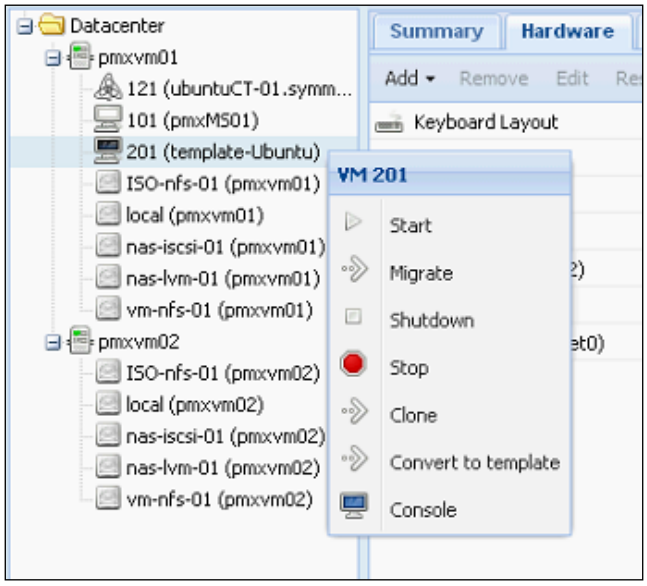
| Menu item | Function |
|---|---|
| **Start** | Starts virtual machine. |
| **Migrate** | Allow online/offline migration of virtual machine. |
| **Shutdown** | Safely powers down virtual machine. |
| **Stop** | Powers down virtual machine immediately. Might cause data loss. Similar to holding down the Power button for 6 seconds on a physical machine. |
| **Clone** | Clones virtual machine. |
| **Convert to Template** | Transforms a virtual machine into a template for cloning. Templates themselves cannot be used as a regular virtual machine. |
| **Console** | Opens a virtual machine in a VNC console. |

## Transforming VM into a template

Let's turn our Ubuntu virtual machine we created in the *Creating a KVM virtual machine* section into a template. Perform the following steps:

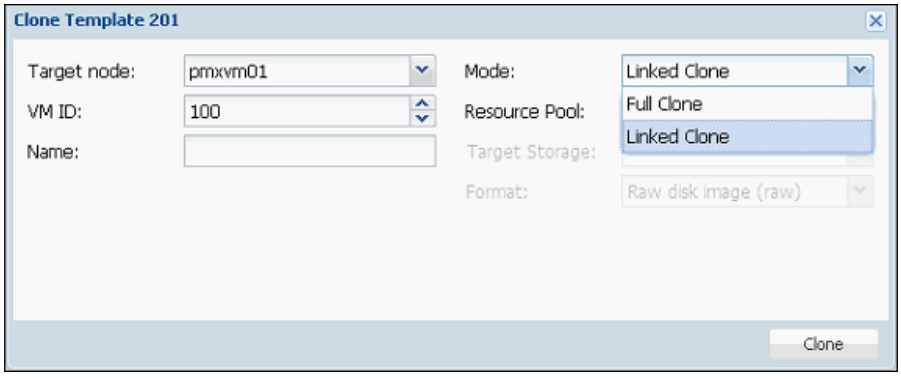
1. Right-click on a virtual machine to open the context menu.

2. Click on **Convert to template** as shown in the following screenshot. This will convert the VM into a template that can be used to clone an unlimited number of virtual machines. While creating a template, keep in mind that a template itself cannot be used as a virtual machine. But it can be migrated to different hosts just like a virtual machine.




## Cloning using a template

The template is now ready for cloning. Right-clicking on the template will open up the context menu, which will have only two menu options: **Migrate** and **Clone**. Click on **Clone** to open the template cloning option window as shown in the following screenshot:



The most important option to notice in this menu is the **Mode** option. A clone can be created from a template using either **Full Clone** or **Linked Clone**.

## Full Clone versus Linked Clone

The following is a comparison table with features of **Full Clone** and **Linked Clone**:

| Full clone | Linked clone |
|---|---|
| Fully independent from original VM/template | Linked with original VM/template it was created from |
| Takes the same amount of disk space as original VM/template | Takes less disk space than original VM/template |
| Supported file types: `raw`, `qcow2`, and `vmdk` in LVM, NFS or iSCSI | Does not support storage file in lvm and iSCSI |
| If original VM/template is lost or damaged, the cloned VM stays intact | If original VM/template is lost, the cloned VM can no longer function. All linked VMs are connected to original VM/template |
| Full Clone has greater performance over Linked Clone | Performance degrades in Linked VM as more people share the same original VM/template |
| Full Clone takes much longer to create | Linked Clone can be created within minutes |
| Full Clone is a replica of the original | Linked Clone is created from a snapshot of original VM/template |

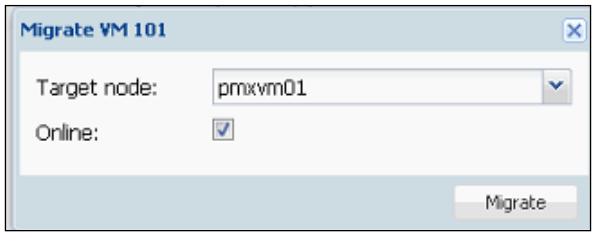
From the previous table, we can see that both Full Clone and Linked Clone have pros and cons. One rule of thumb is that if performance is the main focus, go with Full Clone. If storage space conservation is the focus, then go with Linked Clone.

Attention: a damaged original VM/template can render all linked VMs unusable!

# VM migration

Proxmox migration allows a VM or OpenVZ container to be moved to a Proxmox node in both offline and online modes. The most common scenario of VM migration is when a Proxmox node needs a reboot due to a major kernel update. Without the migration option, each reboot would be very difficult for an administrator as all the running VMs have to be stopped first before reboot occurs, which will cause major downtime in a mission-critical virtual environment.

With the migration option, a running VM can be moved to another node without a single downtime. During live migration, VM does not experience any major slowdown. After the node reboots, simply migrate the VMs back to the original node. Any offline VMs can also be moved with ease.




Proxmox takes a very minimalistic approach to the migration process. Just select the destination node and online/offline check box. Then hit the **Migrate** button to get the migration process started. Depending on the size of virtual drive and allocated memory of the VM, the entire migration process time can vary.

Live/online migration also migrates virtual memory content of the VM. The bigger the memory, the longer it will take to migrate.

# Summary

In this chapter, we saw what a basic Proxmox cluster looks like and went through the setup process of Proxmox nodes. We took a closer look at the Proxmox GUI where we will spend almost all of our virtual infrastructure administrative life. We also have set up a basic-level Proxmox cluster, which will serve us as a foundation for the rest of the book and help us to gain knowledge of inner workings of Proxmox.

We created virtual machines in our cluster and learned how cloning and template can save an enormous amount of time. We attached a shared storage with our cluster using FreeNAS. FreeNAS is an excellent open source choice for all **Network Attached Storage** (**NAS**) needs. It supports NFS, CIFS, AFP, iSCSI, FTP, TFTP, RSYNC, ZFS, and many more storage-related features.

There is a lot of information on Proxmox available at the official wiki page at `https://pve.proxmox.com/wiki/Main_Page`.

With the introductory chapter out of the way, in the next chapter, we will take a look at what is under the hood of Proxmox hypervisor. We will see how the Proxmox folder structure is laid out to hold some of the important files, which make Proxmox run so effectively. Most importantly, we will go deeper into some of the configuration and see their function line by line. In order to build a complex enterprise-class Proxmox cluster, it is important to be quite familiar with these configurations. Proxmox cluster can be tweaked and tailored further beyond the GUI through these files.

For more information, you can visit `http://www.masteringproxmox.com/`. You can use this forum for discussing about Proxmox and related topics.

# 2
# Proxmox Under the Hood

In the previous chapter, we have seen how a Proxmox GUI looks like and set up a basic cluster, which we are going to use for hands-on learning of Proxmox. In this chapter, we will look at how configuration files hold a Proxmox virtualization platform together, files to be used for advanced configuration, and how to troubleshoot a Proxmox platform. Proxmox is built on Debian Linux, which is very stable with a large active community. So, it inherited the heavy dependency on configuration or the `.conf` files as we Linux users lovingly like to call them. The Proxmox GUI provides you with the ability to manage a cluster, but does not provide direct access to any configuration files. Any direct changes by an advanced user have to be done through CLI. Commonly used scenarios such as adding special arguments in configuration files are also done through CLI. In this chapter, we will look at the following topics:

- The Proxmox directory structure
- Configuration files' locations and their functions
- Arguments and syntaxes used in configuration files

# The Proxmox cluster directory structure

Proxmox is a cluster-based hypervisor. It is meant to be used with several server nodes. By using multiple nodes in a cluster, we provide redundancy or High Availability to the platform while increasing uptime. A production virtual environment may have several dozens to several hundreds of nodes in a cluster. As an administrator, it may not be a realistic scenario to change configuration files in the cluster one node at a time. Depending on the number of nodes in a cluster, it may take several hours just to change one small argument in a configuration file of all the nodes. An administrator would have to go from node to node to apply these changes. To alleviate wastage of precious time, Proxmox implemented the clustered filesystem to keep all the configuration files, or any other common files shared by all the nodes in the cluster, in a synchronous state. Its official name is **Proxmox Cluster file system** or **pmxcfs**, which is a database-driven filesystem to store configuration files. Any changes made to any files or copied/deleted in this filesystem get replicated in real time to all the nodes using **Corosync**. **Corosync Cluster Engine** is a group communication system used to implement High Availability within an application. You can learn more about Corosync by visiting the link `http://corosync.github.io/corosync/`.

Any file added to this filesystem almost instantly gets replicated to all the nodes in the cluster, thus saving enormous amount of time for a system administrator.

The pmxcfs filesystem is database-driven and is used to store the Proxmox cluster configuration files or any other files commonly shared by all the nodes in the Proxmox cluster. To know more about pmxcfs, please visit Proxmox Wiki at `http://pve.proxmox.com/wiki/Proxmox_Cluster_file_system_(pmxcfs)`.

The pmxcfs filesystem is mounted at:

```
# /etc/pve
```

All cluster-related files are stored in this folder path. Now, let us take a look at the folder tree, which shows the location of the files stored and their function:

| Filename/location | File function |
|---|---|
| `# /etc/pve/datacenter.cfg` | Proxmox VE datacentre configuration file. Used to change options, such as default language, keyboard layout, default console, and so on. |
| `# /etc/pve/cluster.conf` | Cluster main configuration file. Can also be used to change the vote of a particular node. |

| Filename/location | File function |
|---|---|
| `# /etc/pve/storage.cfg` | PVE storage configuration file. Holds all the information about local or shared storage system. |
| `# /etc/pve/user.cfg` | User list and access control configuration for all users and groups in the cluster. |
| `# /etc/pve/authkey.pub` | Public key used by the ticket system. |
| `# /etc/pve/priv/shadow.cfg` | Shadow password file for users. |
| `# /etc/pve/priv/authkey.key` | Private key used by the ticket system. |
| `# /etc/pve/nodes/<name>/pve-ssl.pem` | Public SSL key for the web server. Used to access Proxmox WebGUI. |
| `# /etc/pve/nodes/<name>/priv/pve-ssl.key` | Private SSL key. |
| `# /etc/pve/nodes/<name>/qemu-server/<vmid>.conf` | Virtual machine configuration data for KVM VMs. |
| `# /etc/pve/nodes/<name>/openvz/ <vmid>.conf` | Virtual machine configuration data for OpenVZ containers. |
| `# /etc/pve/.version` | File versions' data to detect file modifications. |
| `# /etc/pve/.members` | Information nodes that are members of the cluster. |
| `# /etc/pve/.vmlist` | List of all VMs in the cluster. |
| `# /etc/pve/.clusterlog` | Last 50 entries of the cluster log. |
| `# /etc/pve/.rrd` | Most recent entries of RRD data. |

Any changes on these files or any other files inside pmxcfs mounted under folder `/etc/pve` will get replicated automatically. For this reason, we will have to take extra care about what we do to these files. For example, if we delete a `.conf` file from one node by mistake, it will also be deleted from all the other nodes in the Proxmox cluster.

Regular manual backup of the `/etc/pve` folder should be a common practice, in case the cluster needs rebuilding after any disaster or accidental file deletion/change.

On a regular day-to-day basis, a system administrator will not need to access these files from the command line since almost all of these are editable from the Proxmox GUI. But knowing the location of these files and what they hold might save the day when GUI becomes inaccessible for whatever reason.

# Dissecting the configuration files

We now know where all the important files that hold a Proxmox cluster together are placed. We will now go inside some of these files for better understanding of what they do and the command arguments they use. You can use any Linux editor to view/edit these configuration files. In this book, we will use `#nano` to view and edit.

During the learning process, it will be a good idea to make a backup of the configuration files before editing them. In case something goes wrong, you will be able to replace it with the original working configuration file. Simply copy a configuration file using the following command:

```
# cp /etc/pve/<config_file> /home/<any_folder>
```

We can also use the SCP command to back up files to another node:

```
# scp /etc/pve/<config_file> <user>@<ip_or_hostname>:/<folder>
```

# The cluster configuration file

The following code is what our `cluster.conf` file currently looks like from the cluster we created earlier in *Chapter 1*, *Dive into the Virtual World with Proxmox*. The Proxmox cluster configuration file is located under `/etc/pve/cluster.conf`. Its contents are as follows:

```
<?xml version="1.0"?>
<cluster name="pmx-cluster" config_version="2">
<cman keyfile="/var/lib/pve-cluster/corosync.authkey"></cman>
<clusternodes>
  <clusternode name="pmxvm01" votes="1" nodeid="1"/>
  <clusternode name="pmxvm02" votes="1" nodeid="2"/>
</clusternodes>
</cluster>
```

Now, let us see what function each XML tag performs in this configuration file.

Corosync uses XML for configuration files. Since all cluster messaging between nodes is handled by Corosync, XML syntaxes are used in the Proxmox cluster configuration file. The `<?xml .. >` tag shows the XML version number as shown:

```
<?xml version="1.0"?>
```

Now let us take a look at the following code:

```
<cluster name="pmx-cluster" config_version="2">
. . . .
. . . .
</cluster>
```

The `<cluster>..</cluster>` tag acts as the main body of the configuration file. All other syntaxes reside inside this tag. The beginning of this tag shows the name of the cluster as `name="pmx-cluster"` and the config file version number as `config_version="2"`. This number represents how many times the file has been changed. Each change increases the version number incrementally:

```
<cman keyfile="/var/lib/pve-cluster/corosync.authkey"></cman>
```

The `<cman>` tag represents the location of authorization key file for Corosync. This key file is needed by Corosync for authentication within cluster communication from one node to another.

Now have a look at the following code:

```
<clusternodes>
  <clusternode name="pmxvm01" votes="1" nodeid="1"/>
  <clusternode name="pmxvm02" votes="1" nodeid="2"/>
</clusternodes>
```

The `<clusternode>` tag shows the names of all the member nodes in the cluster, node ID, and expected votes to form Quorum. In order to have a healthy cluster or the cluster to be ready, it is very important to form Quorum. A cluster has a specific number of nodes and each node is considered as one vote. When majority (over 50 percent) of the votes or nodes are available, Quorum is achieved. If the number of votes falls below the majority, the cluster stops providing services. But nodes will continue listening for incoming connections from other nodes in case failed nodes appear again on the network.

When Quorum is lost, pmxcfs becomes inaccessible to prevent data inconsistency. If it is very important to access the pmxcfs on a node when Quorum is lost, then executing the following command on the same node will form a Quorum:

```
# pvecm expected 1
```

This command will let the node know that only one vote is expected to have Quorum, even though there are other nodes in the cluster.

A Quorum is the minimum number of nodes needed to achieve a healthy cluster. Proxmox needs a minimum of three nodes to form a Quorum, or a shared storage system to store a Quorum disk for a two-node cluster.

Although not recommended, it is possible to create a Proxmox cluster with just two nodes and achieve High Availability. Look into the Wiki page at `http://pve.proxmox.com/wiki/Two-Node_High_Availability_Cluster` to get details on how to create a two-node High Availability cluster.

It is possible to operate a cluster with just two nodes. But this form of setup is only good for very small usage, such as the home or test platform where downtime really has no effect on operation.

# The storage configuration file

The following is the storage configuration file for our basic cluster. This file is located under `/etc/pve/storage.cfg`. Currently, we have local, nfs, iscsi, and lvm shared storage attached to the basic Proxmox cluster we created in *Chapter 1*, *Dive into the Virtual World With Proxmox*. Later in the book, we will add a Ceph-distributed storage system to the cluster, which provides RBD-shared storage. We will revisit this `storage.cfg` file at that time to learn about options for RBD storage. But for now, this is how our storage is configured through `/etc/pve/storage.cfg` for our hands-on cluster:

```
dir: local
  path /var/lib/vz
  content images,iso,vztmpl,rootdir
  maxfiles 0

nfs: ISO-nfs-01
  path /mnt/pve/ISO-nfs-01
  server 192.168.145.11
  export /mnt/pmxnas01
  options vers=3
  content iso,vztmpl
  maxfiles 1

iscsi: nas-iscsi-01
  target iqn.2011-03.org.example.istgt:pmxtgt01
  portal 192.168.145.11
  content none

lvm: nas-lvm-01
  vgname nas-lvm-01
  base nas-iscsi-01:0.0.0.scsi-330000000391132dd
  shared
  content images

nfs: vm-nfs-01
  path /mnt/pve/vm-nfs-01
  server 192.168.145.11
  export /mnt/pmxnas01
```

```
  options vers=3
  content images,vztmpl,backup,rootdir
  maxfiles 1
```

Almost all the settings in `storage.cfg` can be changed from the Proxmox GUI without using any CLI.

Attached storages abide by the following common format in the `storage.cfg` file:

```
storage_type : storage_name
  path </path to folder>
    target <target file name> (for iSCSI)
    portal <server IP address> (for iSCSI)
    vgname <volume group name> (for LVM)
    base <base volume group> (for LVM)
    server <storage server IP address>
    export </shared location on NFS server>
    content <type of files the storage can hold>
    maxfile <maximum number of old backup to keep>
```

# Local directory-based storage

Based on the previous storage configuration format, let us take a look at the local storage segment in the `storage.cfg` file:

```
dir: local
  path /var/lib/vz
  content images,iso,vztmpl,rootdir
  maxfiles 0
```

From the previous configuration, we can tell that this is a local directory-based storage, which is mounted on the `/var/lib/vz` path in the local Proxmox node. The following are the content types it can hold:

- Images: Actual virtual machine images
- ISOs: ISO images of CDs
- OpenVZ templates: OpenVZ/container templates

With `maxfiles` set to `0`, this storage location cannot do any VM backup.

# NFS-shared storage

Now, we will look at the second storage section in the `storage.cfg` file as follows:

```
nfs: ISO-nfs-01
  path /mnt/pve/ISO-nfs-01
```

```
server 192.168.145.11
export /mnt/pmxnas01
options vers=3
content iso,vztmpl
maxfiles 1
```

Now, we will explain the individual lines of the previous code.

The following line is an NFS storage type with the name `ISO-nfs-01`. We can provide any alphanumeric name. But a meaningful name helps to easily identify the purpose of the storage. In this instance, the name represents an NFS share, which holds ISO files:

```
nfs: ISO-nfs-01
```

The NFS share is mounted on all nodes present in the path `/mnt/pve/ISO-nfs-01`, as shown in the following code:

```
path /mnt/pve/ISO-nfs-01
```

The address in the following line is the NFS server IP address:

```
server 192.168.145.11
```

The path in the following line is the path where the actual NFS share is mounted on the NFS server:

```
export /mnt/pmxnas01
```

By default, Proxmox uses NFS Version 3 due to its maturity and stability as shown in the following line:

```
options vers=3
```

There is no clear performance advantage to moving from NFSv3 to NFSv4. There is no option to change to NFSv4 through GUI. It has to be done through CLI by editing the `storage.cfg` file. Change the `vers` option to `4` to use NFSv4. NFSv4 does add some new improvements in Version 3. The following table shows a comparison between NFSv3 and NFSv4:

| **Features** | **NFSv3** | **NFSv4** |
|---|---|---|
| State | Stateless | Stateful |
| Authentication | AUTH_SYS (weak) | Kerberos (strong) |
| Data transport | UDP and TCP | TCP |
| Permissions | Unix based | MS Windows-like |
| ID | 32-bit UID/GID | String based |

The following line shows the NFS-shared storage configured to hold an ISO image and OpenVZ/container template only:

```
content iso,vztmpl
```

The NFS-shared storage can keep a maximum of one latest backup as shown:

```
maxfiles 1
```

Older backups get deleted. When setting the `maxfiles` parameter, always be sure that the storage has enough space to keep up with the retained backups. For example, if `maxfiles` is set to `10`, the storage will keep 10 latest backups and delete the older ones. When backing up 200 GB of data of virtual machines with `maxfiles` set to `10`, the NFS-shared storage will need 2 TB of space at all times to hold 10 backups.

The same NFS share can be attached to Proxmox multiple times for different purposes. For example, an NFS share/nfs_server/share_name can be attached to nfs-iso-01 to store ISO and as nfs-vm-01 to store virtual machines.

## iSCSI/LVM shared storage

iSCSI/LVM requires extra steps than NFS to set up. For this type of share to work with Proxmox, an iSCSI share first needs to be attached with Proxmox, and then an LVM storage can be created on top of it. The following is the iSCSI/LVM configuration from our basic cluster:

```
iscsi: nas-iscsi-01
  target iqn.2011-03.org.example.istgt:pmxtgt01
  portal 192.168.145.11
  content none
lvm: nas-lvm-01
  vgname nas-lvm-01
  base nas-iscsi-01:0.0.0.scsi-330000000391132dd
  shared
  content images
```

Let's take a look at each line of the previous code.

The following line is an iSCSI type share with name `nas-iscsi-01`. Always give iSCSI share a distinguished name, since the Proxmox GUI shows both iSCSI and LVM as storage, but only LVM has usable storage space. The iSCSI share will always show available space as zero (0). Also, Proxmox identifies an iSCSI base storage with this name:

```
iscsi: nas-iscsi-01
```

The following line shows the iSCSI extent stored on the iSCSI server, which could be backed up by a physical disk or a file. In our setup, we have used a file-based iSCSI extent:

```
target iqn.2011-03.org.example.istgt:pmxtgt01
```

The address in the following line is the IP address for the iSCSI server:

```
portal 192.168.145.11
```

Content for iSCSI is always set to none as shown in the following line. The Proxmox GUI has no option to set content on iSCSI. The content is configured on LVM created based on the iSCSI target. iSCSI is a media object without any partition or filesystem. Proxmox also allows iSCSI LUNs to be directly connected with Proxmox, but due to stability issues this is not recommended.

```
content none
```

The following line shows the name of the LVM storage created based on the iSCSI target attached. It should be named different than an iSCSI share that is easily identifiable.

```
lvm: nas-lvm-01
```

The following line shows a volume group name for the LVM storage:

```
vgname nas-lvm-01
```

The following line shows the base iSCSI target that was attached in the iSCSI section:

```
base nas-iscsi-01:0.0.0.scsi-330000000391132dd
```

If shared is enabled, the LVM storage will be available through all the nodes. It is worth mentioning here that a shared storage is required to configure node failovers or High Availability (HA). Shared is enabled as follows:

```
shared
```

The iSCSI LVM storage can only hold VM image files. It can be used to hold ISO, template, or any backup files, as shown in the following line:

```
content images
```

# User configuration files

The `user.cfg` file holds all user and group information in the cluster and is located under `/etc/pve/`. It follows the following format to store all user information:

- For user information, the format is as follows:

  ```
  <type>:<user>@realm:enable:expiry:f_name:l_name:email:
    comment
  ```

- For group information, the format is as follows:

  ```
  <type>:<group_name>:user@realm:comment
  ```

Based on this format, the following is what our `user.cfg` file looks like right now. Depending on user information you have entered, the file content may vary:

```
user:root@pam:1:0:::admin@domain.com::
user:pmxuser@pve:1:0:ABC:XYZ:pmxuser@domain.com:A User:

group:test:wahmed@pve:This is Test Group:
```

Note that the file `user.cfg` does not hold any user password. This information is stored under `/etc/pve/priv/shadow.cfg`.

# The password configuration file

The password configuration file is located under `/etc/pve/priv/shadow.cfg` and stores all the passwords for users in the cluster. The format is rather simple but the function of this file is rather crucial. The format to store password information is as follows:

```
<user_name>:<encrypted_password>
```

We have one user pmxuser in the cluster. The file looks as follows:

```
wahmed:aLKJ98213498lsdfMN/KSDkwhkjfh290342340823:
```

Notice that the password file is in a `/priv` folder inside `/etc/pve`. Sensitive information such as password, private authorization key, and known hosts are kept in the `/etc/pve/priv` folder. When a new user is created through the Proxmox GUI, a new entry is added here as password entry.

# The virtual machine configuration file

The `vmid.conf` file stores configuration information for each virtual machine and is located under `/etc/pve/nodes/<name>/qemu-server/<vmid.conf>`. The folder structure divides all VM configuration files into categories based on nodes. For example, the configuration file for our VM with `ID#101` is stored in the following location:

```
# /etc/pve/nodes/pmxvm01/qemu-server/101.conf
```

When we migrate a VM from one node to another, Proxmox just moves the configuration file to the destination node. If the VM is powered on during the migration, then the entire memory content of the VM is also migrated to the destination node. For our VM 101, if we migrate it to the second node in the cluster, which is pmxvm02, then the location of the `101.conf` file would be as follows:

```
# /etc/pve/nodes/pmxvm02/qemu-server/101.conf
```

If a node with virtual machines in it becomes inaccessible, simply moving the `<vm_id>.conf` files to a different node will allow accessing all the VMs from a different node. Any files of the folder inside `/etc/pve` can be seen from any node in the cluster.

We will now look at a `<vm_id>.conf` file itself to see what makes up a virtual machine behind the scenes. This configuration file follows a simple `OPTION: value` format. The following is the configuration file for our VM 101:

```
ballon: 512
bootdisk: virtio0
cores: 1
ide2: none,media=cdrom
kvm: 0
memory: 1024
name: pmxms01
net0: e1000=42:5C:EB:2A:5F:05,bridge=vmbr0
ostype: 126
sockets: 1
virtio0: vm-nfs-01:101/vm-101-disk-1.qcow2,format=qcow2,size=15G
```

Almost all the options in this file can be set through the Proxmox GUI under the KVM virtual machine option menu tab. Some option values such as args, hotplug, and so on have to be added through CLI. The following is a chart of the possible options. The values can be used as virtual machine configurations:

| Options | Description | Possible values |
|---|---|---|
| `acpi:` | Enable/disable ACPI for VM. | 1<br>0 |
| `args:` | Allows passing arguments to VM. Features such as sound can be activated by using KVM arguments. See the *Arguments in the KVM configuration file* section for more details on arguments used in KVM. | See the *Arguments in the KVM configuration file* section |
| `autostart:` | Autorestart virtual machine after crash. Default is 0. | 1<br>0 |
| `ballon:` | Targeted RAM for VM in MB. | Integer number |
| `boot:` | Default boot device. | c \| d \| n<br>c=hdd; d=cd-rom; n=network |
| `bootdisk:` | Enable booting from a specific disk. | ide \| sata \| scsi \| virtio |
| `core:` | Number of cores per socket. Default is 1. | Integer number |
| `cpu:` | Emulated CPU types. Default is kvm64. | 486 \| kvm32 \| kvm64 \| qemu32 \| qemu64 \| conroe \| haswell \| nehalem \| opteron_ G1/G2/G3/G4/G5 \| penryn \| sandybridge \| westmere \| athlon \| core2duo \| coreduo \| host \| pentium \| pentium2 \| pentium3 \| phenom |
| `cpuunits:` | This is the CPU weight for the VM. This value is used by the kernel fair scheduler. The larger the value is, the more CPU time VM will get. Note that this value is relative to the weights of all other running VMs in the cluster. Default is 1000. | Integer 0 to 500000 |
| `description:` | Notes for the VM. | Plain texts |
| `freeze:` | Freezes CPU at startup. | 1 \| 0 |

| Options | Description | Possible values |
|---|---|---|
| `hostpci(n):` | The option allows a VM direct access to host the hardware. When this option is used, it is no longer possible to migrate the VM. Caution should be used for this option as it is still in the experimental stage. Not recommended for the production environment. | HOSTPCIDEVICE<br>Syntax for HOSTPCIDEVICE is:<br>`bus: <pci_device_ number>`<br>Get pci_device_number from the command `#lspci` |
| `hotplug:` | Enables hotplug for disk and network devices. Default is 0. | 1 \| 0 |
| `ide(n):` | Allows volume to be used as IDE disk or CD-ROM. *n* in `ide(n)` is limited to 0-3. | [volume=]image_ name],[media=cdrom\|disk], [cyls=c,heads=h, secs=s,[trans=t]], [snapshot=on\|off], [cache =none\|writethrough\|writ eback\|unsafe\|directsync], [format=f], [backup=yes\|no], [rerror=ignore\|report\|stop], [werror=enospc\|ignore\|repor t\|stop], [aio=native\|threads] |
| `kvm:` | Enable/disable KVM hardware virtualization. This option disables any hardware acceleration within a VM. A possible usage scenario is when you are setting up a nested virtualized cluster. Default is 1. | 1 \| 0 |
| `lock:` | Enables locking/unlocking for VM. | backup \| migrate \| rollback \| snapshot |
| `memory:` | Allocated amount of RAM for the VM. | Integer number from 16 to n |
| `migrate_ downtime:` | Value in seconds for the maximum tolerated downtime for migration. Default is 0.1 | Number 0 to n |
| `migrate_ speed:` | Value for maximum speed in MBps for VM migrations. Set value to 0 for no limit. Default is 0 | Integer from 0 to n |
| `name:` | Name for the VM | Text |

| Options | Description | Possible values |
|---|---|---|
| `net(n):` | Specified network devices.<br>MODEL=XX:XX:XX:XX:XX:XX, [bridge=<dev>],[rate=<mbps>], [tag=<vlanid>] | MODEL= e1000 \| i82551 \| i82557b \| i82559er \| ne2k_isa \| ne2k_pci \| pcnet \| rtl8139 \| virtio |
| `onboot:` | Enable/disable VM autostart during host node reboot | 1 \| 0 |
| `sata(n):` | Allows volume to be used as an SATA disk or CD-ROM. *n* in `sata(n)` is limited to 0-5. | [volume=]volume], [media=cdrom\|disk], [cyl s=c,heads=h,secs=s,[trans =t]], [snapshot=on\|off] ,[cac he=none\|writethrough\|wr iteback\|unsafe\|directsync], [format=f], [backup=yes\|no], [rerror=ignore\|report\|stop], [werror=enospc\|ignore\|repor t\|stop], [aio=native\|threads] |
| `scsi(n):` | Allows volume to be used as an SCSI disk or CD-ROM. *n* in `scsi(n)` is limited to 0-13. | [volume=]volume], [media=cdrom\|disk], [cyl s=c,heads=h,secs=s[,trans =t]], [snapshot=on\|off], [cac he=none\|writethrough\|wr iteback\|unsafe\|directsync], [format=f], [backup=yes\|no], [rerror=ignore\|report\|stop], [werror=enospc\|ignore\|repor t\|stop], [aio=native\|threads] |
| `scsihw:` | SCSI controller type. Default is lsi. | lsi \| megasas \| virtio-scsi-pci |
| `shares:` | This is the value-allocated amount of RAM for autoballooning. The larger this value is, the more RAM the VM will get. The value 0 disables this option. Default is 1,000. | Integer from 0 to 50,000 |
| `sockets:` | Number of CPU sockets. Default is 1. | Integer from 1 to N |
| `startdate:` | This option sets the initial date of the real-time clock | now \| YYYY-MM-DD \| YYYY-MM-DDTHH:MM:SS |

| Options | Description | Possible values |
|---|---|---|
| `startup:` | This option sets the behavior for VM startup and shutdown. Order is a positive integer number, which sets the order in which VMs will start. Shutdown follows the order value in reverse. The delay of startup and shutdown can be set up and down in seconds. | [order=+ Int], [up=+ Int], [down=+ Int] |
| `tablet:` | Enables/disables the USB tablet device in VM. When running a lot of consoles on the only VM on a host, disabling this feature can save **context switches** (`http://en.wikipedia.org/wiki/Context_switch`). Default is 1. | 1 \| 0 |
| `unused(n):` | Unused volumes in VM. When a virtual drive is deleted from a VM, the volume does not get deleted instantly. Instead, the status changes to `unused:<volume_name>`. At a later time, if the volume is needed, it can be reattached to the VM by changing the option to `ide(n): \| scsi(n): \| sata(n):`. | String |
| `usb(n):` | Enables pass-through direct access to a USB device. *n* can be set to 0 to 4. When this option is used, it is no longer possible to migrate the VM. | HOSTUSBDEVICE<br>Syntax for HOSTUSBDEVICE is:<br>`<vendor_id:product_id>`. Get pci_device_number from the command `#lsusb -t` |
| `vga:` | VM display type. | cirrus \| std \| vmware \| qxl |
| `virtio(n):` | Allows volume to be used as a virtio disk. *n* in `virtio(n)` is limited to 0-15. | [volume=]volume], [media=cdrom\|disk], [cyl s=c,heads=h,secs=s[,trans =t]], [snapshot=on\|off], [cac he=none\|writethrough\|wr iteback\|unsafe\|directsync], [format=f], [backup=yes\|no], [rerror=ignore\|report\|stop], [werror=enospc\|ignore\|repor t\|stop], [aio=native\|threads] |

## Arguments in the KVM configuration file

Arguments in a virtual machine configuration file is a way to extend the capability of the VM beyond just the default. For example, by default, sound is not enabled for a VM. In order to give a VM the ability to play audio/video, an argument has to be passed through the VM configuration file. The following are some examples of arguments that can be used in a Proxmox VM configuration file. Arguments can be added in the following format:

```
args: -<device_arguments_1> -<device_arguments_2> . . . .
ballon: 512
bootdisk: virtio0
cores: 1
ide2: none,media=cdrom
. . . .
. . . .
```

Enable serial device in VM by using the following code:

```
args: -serial /dev/ttyS0
```

Enable sound in Windows XP VM by using the following line:

```
args: -device AC97,addr=0x18
```

Enable sound in Windows 7 VM by using the following code:

```
args -device intel-hda,id=sound5,bus=pci.0,addr=0x18 –device had-
  micro,id=sound5-codec0,bus=sound5.0,cad=0 –device had-
    duplex,id=sound5-codec1,bus=sound5.0,cad=1
```

Enable UUID in VM using the following line:

```
args –uuid fl234a93-20d32-2398-129032ds-2322
```

Enables support for `aio=native` in VM as follows:

```
args: -drive
  file=/dev/VGGRP/VOL,if=virtio,index=1,cache=none,aio=native
```

# The Proxmox OpenVZ configuration file

The following is the OpenVZ configuration file of the container `#121` that we created in *Chapter 1, Dive into the Virtual World with Proxmox*.

```
ONBOOT="no"

PHYSPAGES="0:1024M"    //RAM Allocated in bytes
```

```
SWAPPAGES="0:1024M"    //Swap Allocated in bytes
KMEMSIZE="465M:512M"   //Size of unswappable memory in bytes
DCACHESIZE="232M:256M"
LOCKEDPAGES="512M"
PRIVVMPAGES="unlimited"  //Memory allocation limit in pages
SHMPAGES="unlimited"
NUMPROC="unlimited"
VMGUARPAGES="0:unlimited"   //Guarantees the available physical
                              resources of high memory consumption
                              by system processes
OOMGUARPAGES="0:unlimited"  //Guaranteed amount of memory during
                              out-of-memory situation
NUMTCPSOCK="unlimited"
NUMFLOCK="unlimited"
NUMPTY="unlimited"
NUMSIGINFO="unlimited"
TCPSNDBUF="unlimited"       //Size of buffers used to send data
                              over tcp network connections
TCPRCVBUF="unlimited"   //Size of buffers used to temporarily store
                          data coming from tcp network connections
OTHERSOCKBUF="unlimited"  //Size of buffers used by local
                            connections between processes inside
                            the system
DGRAMRCVBUF="unlimited"   //Size of buffers used to temporarily
                            store the incoming packets of UDP and
                            other datagram protocols
NUMOTHERSOCK="unlimited"
NUMFILE="unlimited"
NUMIPTENT="unlimited"

# Disk quota parameters (in form of softlimit:hardlimit)
DISKSPACE="15G:16896M"
DISKINODES="3000000:3300000"
QUOTATIME="0"
QUOTAUGIDLIMIT="0"

# CPU fair scheduler parameter
CPUUNITS="1000"       //CPU Time Weights
CPUS="1"              //Number of CPU core
HOSTNAME="ubuntuCT-01.domain.com"   //Container Hostname
```

```
SEARCHDOMAIN="domain.com"           //Container Domain
NAMESERVER="208.67.222.222"         //IP address of Nameserver
NETIF="ifname=eth0,mac=3E:FF:77:32:B2:A0,host_ifname=veth121.0,
  host_mac=16:0D:1E:20:B5:7A,bridge=vmbr0"     //network device
```

Almost all the parameters in the OpenVZ configuration file can be left at their default values. For a complete list of parameters that can be used in OpenVZ configuration, visit the link `http://openvz.org/Man/ctid.conf.5`.

OpenVZ containers that are created with Linked Cloning are heavily dependent on the original KVM virtual machine template they are created from. Any damages to the main template will render all the linked containers unusable.

# The version configuration file

The version configuration file shows the version numbers of configuration files in the cluster and is located under `/etc/pve/.version`. Every time a configuration file is edited, the version number increases in the `.version` file. The following is the `.version` file in our cluster at this moment:

```
{
  "starttime": 1390947588,
  "clinfo": 9,
  "vmlist": 24,
  "cluster.conf": 1,
  "cluster.conf.new": 1,
  "storage.cfg": 2,
  "user.cfg": 6,
  "domains.cfg": 1,
  "priv/shadow.cfg": 2,
  "datacenter.cfg": 1,
  "vzdump.cron": 1,
  "kvstore": {
  "pmxvm01": {
  "tasklist": 127987}
  ,
  "pmxvm02": {
    "tasklist": 127925}
  }
}
```

# Member nodes

Located under `/etc/pve/.members`, the member node file shows all the member nodes that are a part of the Proxmox cluster. It is a great way to see the cluster status, when the Proxmox GUI becomes inaccessible for any reason. The following is the `.members` file in our basic cluster:

```
{
  "nodename": "pmxvm01",
  "version": 9,
  "cluster": { "name": "pmx-cluster", "version": 2, "nodes": 2,
    "quorate": 1 },
  "nodelist": {
    "pmxvm01": { "id": 1, "online": 1, "ip": "192.168.145.1"},
    "pmxvm02": { "id": 2, "online": 1, "ip": "192.168.145.2"}
  }
}
```

# The .members file

The content of a `.members` file is as follows:

```
"nodename": "pmxvm01"
```

The previous code section shows the current node where the `.members` file is being accessed.

```
"version": 9
```

The `.members` file has its own version numbering system. Like the `.version` file, every time `.members` is changed, the version increases. For example, when a node is added or removed from the cluster, the version number moves upward.

```
"cluster": { "name": "pmx-cluster", "version": 2, "nodes": 2,
  "quorate": 1 },
```

The previous section shows cluster information, such as cluster name, cluster version, number of member nodes, and the number of votes (quorate) needed to form a quorum.

```
"pmxvm01": { "id": 1, "online": 1, "ip": "192.168.145.1"
```

Nodes mentioned in the node list section give information, such as ID, online/offline status, and IP address belonging to each node.

# The virtual machine list file

Located under `/etc/pve/.vmlist`, the virtual machine list file stores a list of all the virtual machines within the Proxmox cluster. The `.vmlist` file uses the following format to store the list:

```
"<vmid>": { "node": "<nodename>", "type": "<vm_type>", "version":
  <int> }
```

We have two virtual machines and one template in our basic cluster. The following is the information stored in the `.vmlist` file:

```
{
  "version": 24,
  "ids": {
  "121": { "node": "pmxvm01", "type": "openvz", "version": 20 },
  "101": { "node": "pmxvm01", "type": "qemu", "version": 11 },
  "201": { "node": "pmxvm01", "type": "qemu", "version": 22 }}
}
```

Please note that a VM with ID 201 is a KVM template in our cluster. But the `.vmlist` file does not show any distinguishable information. One way to separate them is by assigning IDs to all the templates within a particular VM. For example, all the templates can have an assigned VM ID between 501 to 599.

# The cluster logfile

The cluster logfile is for the cluster itself and is located under `/etc/pve/.clusterlog`. It mostly maintains a log of login authentication of users. The following is a snippet of our `.clusterlog` file:

```
{
  "data": [
    {"uid": 931, "time": 1392231446, "pri": 6, "tag": "pvedaemon",
      "pid": 118644, "node": "pmxvm01", "user": "root@pam", "msg":
        "successful auth for user 'root@pam'"},
    {"uid": 930, "time": 1392230545, "pri": 6, "tag": "pvedaemon",
      "pid": 115769, "node": "pmxvm01", "user": "root@pam", "
        msg": "successful auth for user 'root@pam'"},
    {"uid": 929, "time": 1392229645, "pri": 6, "tag": "pvedaemon",
      "pid": 115769, "node": "pmxvm01", "user": "root@pam", "
        msg": "successful auth for user 'root@pam'"},
. . . .
. . . .
```

# Summary

In this chapter, we looked at the location of important configuration files needed to run a healthy Proxmox cluster. We also looked through the configuration files from inside to have a better understanding at the parameters used and other possible values for different parameters. As mentioned earlier, most of these configuration files can be changed via a Proxmox GUI. But when the GUI becomes inaccessible for any reason, knowing where these files are located can save a tremendous amount of time by accessing them through CLI.

In the next chapter, we will look at the exciting options of shared storage system, VM image formats used by Proxmox, commercial/noncommercial shared storage options, and set up an advanced shared storage using FreeNAS.

# 3
# Shared Storages with Proxmox

In simple terms, **shared storage** is a medium to store data for simultaneous access by multiple devices or nodes in a network. As server and desktop virtualizations become the norm, shared storage today is much more critical for a virtual environment.

In *Chapter 1*, *Dive into the Virtual World with Proxmox*, we set up our cluster with NFS file sharing using FreeNAS or any other operating system you may have chosen. Although a Proxmox cluster can fully function with **Direct Attached Storage** (**DAS**) or a local storage system in the same Proxmox node, shared storage system has many benefits in a production environment, such as increased manageability, seamless storage expansion, and redundancy just to name a few. In this chapter, we are going to look at the following topics:

- Difference between local and shared storage
- Virtual disk images supported by Proxmox
- Storage types supported by Proxmox
- Commercial and noncommercial shared storage options
- FreeNAS as a shared storage option

Local or shared, a storage system is a vital component of a Proxmox cluster. A storage system is where all the virtual machines reside. Therefore, a deeper understanding of different storage systems will allow an administrator to properly plan storage requirements for any cluster environment.

# Local storage versus shared storage

Shared storage is not absolutely necessary in a Proxmox cluster environment, but without a doubt it makes storage management a walk in the park. In a small business with an extremely limited budget and IT staff, a local storage system will suffice. But when data grows beyond several terabytes and keeps growing exponentially every month due to virtual machine backups, shared storage starts to make a whole lot of sense. In most enterprise virtual environments with critical data, shared storage is the only logical choice due to the benefits it brings to the whole cluster operation. The following are considered benefits of using shared storage:

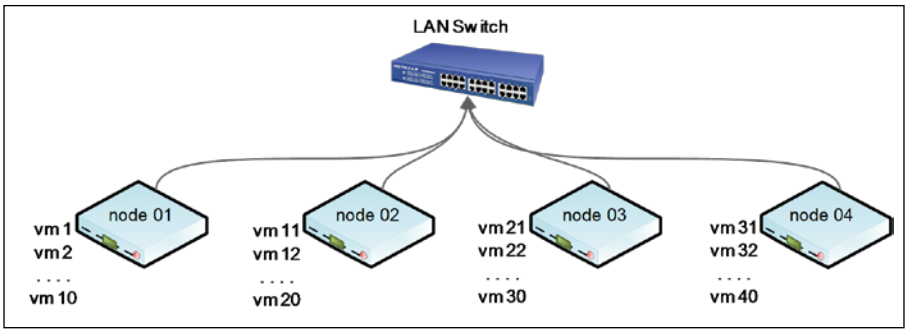
- Live migration of a virtual machine
- Seamless expansion of multinode storage space
- Centralized backup
- Multilevel data tiering
- Central storage management

# Live migration of a virtual machine

This is probably one of the important sought after reasons to go for shared storage system. **Live migration** is when a virtual machine can be moved to a different node without shutting it down first. **Offline migration** is when the virtual machine is powered off prior migration. Proxmox nodes need updates and patches from time to time. Some updates require immediate reboot while some require none at all. The primary function of Proxmox nodes in a cluster is to run virtual machines. When a node needs to be rebooted, all the running VMs must be stopped or migrated to other nodes. Then, migrate them back to the original node after the reboot cycle is complete. In Proxmox, a powered-on VM cannot be migrated using live migration without powering it down first if the VM is on a local disk of the node in question.

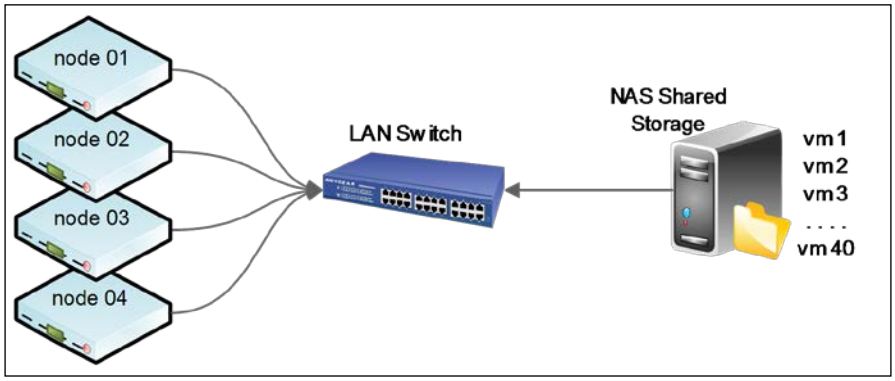
The other drawback of local storage is that if a total Proxmox node failure occurs for any reason, all the VMs stored in that node will be completely unavailable until the node is fixed or replaced. This is because VMs cannot be accessed to be moved to a different node until the issue node is powered up.

In most cases, shutting down all VMs just to reboot the host node is not an option. This causes too much downtime depending on the number of VMs the node handles. In order to migrate locally stored VMs, they must be stopped and migration should be initiated from the Proxmox GUI. Migration from local storage to another local storage takes a long time depending on the size of the VM, since Proxmox moves an entire image file using rsync to relocate the VM to another node. Let us take a look at the following diagram of a cluster with 40 locally stored virtual machines with 10 on each of the four Proxmox nodes:




In the previous overly simplified diagram, there are four Proxmox nodes with 10 virtual machines on each. If **node 01** needs to reboot, all the 10 virtual machines have to be stopped, the node needs to be rebooted, and then all the virtual machines must be powered up. If **node 01** needs to be taken offline for hardware upgrade, then all virtual machines need to be stopped and we have to manually migrate each virtual machine to a different node. If **node 01** fails completely, then all these 10 virtual machines will be inaccessible till **node 01** is back on again. A brave administrator might just take the hard drives out of the failed node, plug them into another Linux machine, copy the virtual machine image files directly to another node, and then power them up. But it will not work if **node 01** had anything but mirror RAID in it for all the hard drives.

So clearly, a cluster setup with a local storage for virtual machines can bring chaos when migration is needed. Now, let us take a look at the following diagram where four Proxmox nodes with 40 virtual machines are stored on a shared storage system:



In the previous diagram, all 40 virtual machines are stored on a shared storage system. The Proxmox node only holds the configuration files for each virtual machine. In this scenario, if **node 01** needs to be rebooted due to a security patch, all the virtual machines can be simply migrated to another node without powering down a single virtual machine. A virtual machine user will never notice that his or her machine has actually moved to a different node. If total Proxmox node failure occurs, the virtual machine configuration file can simply be moved from `/etc/pve/nodes/node01/qemu-server/<vmid>.conf` to `/etc/pve/nodes/node02/qemu-server/<vmid>.conf`.

Since all virtual machine configuration files are in a Proxmox-clustered filesystem, they can be accessed from any node. Refer to *Chapter 2*, *Proxmox Under the Hood*, for Proxmox cluster filesystem or pmxcfs. With virtual machine image files on shared storage, Proxmox migration does not have to move all the image files using rsync from one node to another, which allows much faster virtual machine migration. The rsync is an open source program and network protocol for Unix-based systems. It provides nonencrypted or encrypted incremental file transfers from one location to another.

The bigger the memory (RAM), the longer it will take to employ live migration on a powered-on virtual machine.

It should be noted that shared storage can cause a single point of failure if a single node-based shared storage is set up, such as FreeNAS or NAS4Free without High Availability configured. By using multinode or distributed shared storage such as Ceph Gluster and DRBD, the single point of failure can be eliminated. On a single -node shared storage, all virtual machines are stored on one node. If a node failure occurs, the storage will become inaccessible by Proxmox cluster, thus rendering all running virtual machines unusable.

# Seamless expansion of multinode storage space

Digital data is growing faster than ever before in our modern 24x7 digitally connected world. The growth has been exponential since the introduction of virtualization. Since it is much easier to set up a virtual server at a moment's notice, an administrator can simply clone a virtual server template and within minutes a new virtual server is up and running, consuming storage space. If left unchecked, this regular creating and retiring of virtual machines can force a company to grow out of available storage space. Shared storage system is designed with this very specific need in mind.

In an enterprise environment, storage space should be added seamlessly without shutting down critical nodes or virtual machines. By using a shared storage system, virtual machines can now go beyond one node to scattered multiple nodes spanned across regions. Shared storage such as distributed filesystem (for example, Ceph) can span across several racks and comprise well over several petabytes of usable storage space. Simply add a new node with a full bay of drives and tell the storage cluster to recognize the new node to increase storage space for the entire cluster. Since shared storage is separated from the virtual machine host nodes, storage can be increased or decreased without disturbing any running virtual machines. Later in the book, we will see how the Ceph-distributed filesystem can be expanded as much as your budget or needs allow you to.

# Centralized backup

Shared storage makes centralized backup possible by allowing each virtual machine host node to create a backup on one central location. This helps a backup manager or an administrator to implement a solid backup plan and manage the existing backups. Since a Proxmox node failure will not take the shared storage system down, virtual machines can be easily restored to a new node to reduce downtime.

Always use a separate node for backup purposes. It is not a good practice to store both virtual machines and their backups on the same node.

# Multilevel data tiering

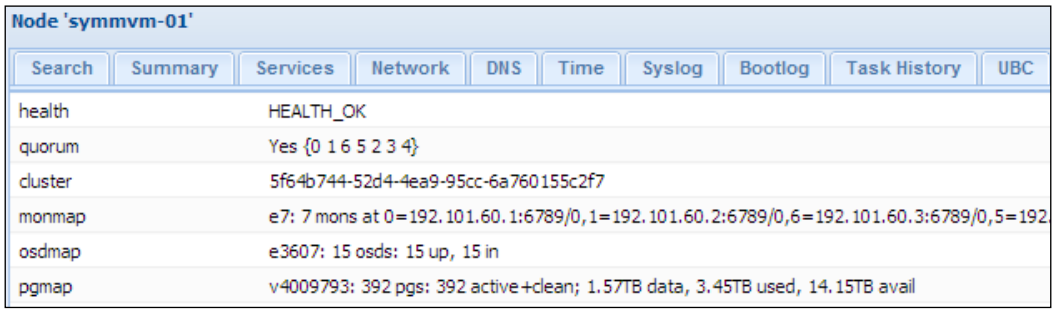
Data tiering is a concept where different files can be stored on different storage pools based on their performance requirement. For example, a virtual file server can provide very fast service if this VM is stored on an SSD storage pool, while a virtual backup server can be stored on a slower HDD storage since backup files are not frequently accessed, and thus, do not require very fast I/O. Tiering can be set up using different shared storage nodes with different performance levels. It can also be set up on the same node by assigning volumes or pools to specific sets of drives.

# Central storage management

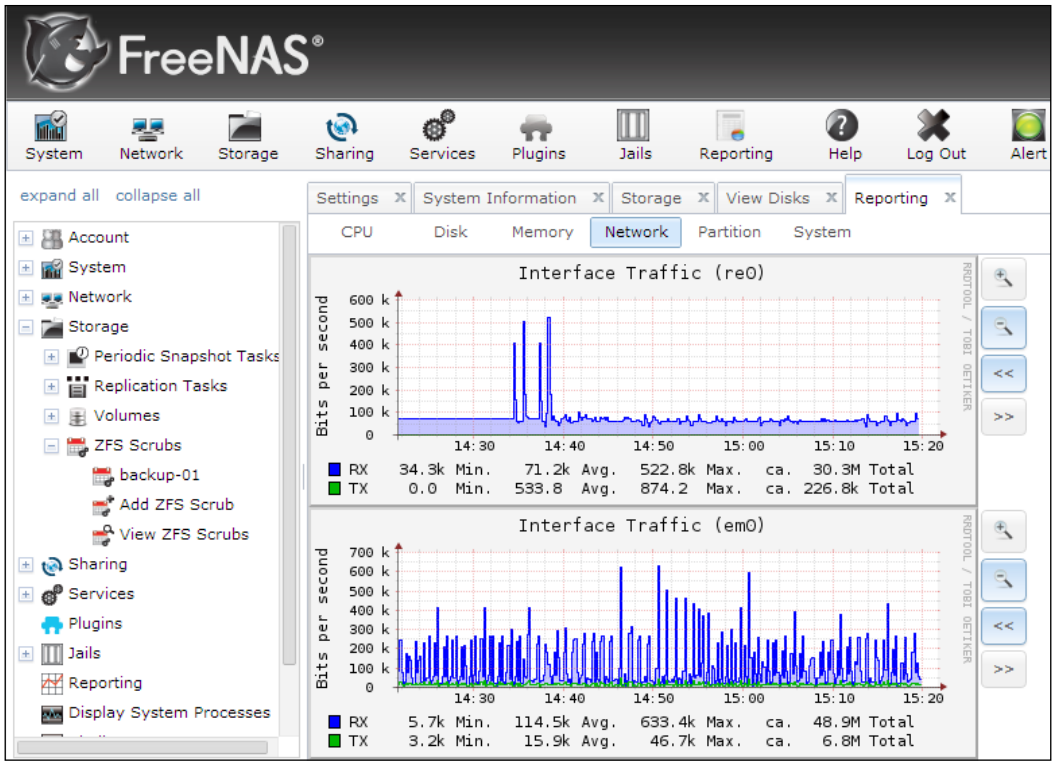
By separating shared storage clusters from primary Proxmox clusters, we can manage two clusters without them interfering with each other. Since shared storage systems can be set up with separate nodes and physical switches, managing them based on different authorizations and permissions becomes an easier task. NAS, SAN, and other types of viable shared storage solutions come with their own management programs, from where an administrator or operator can check storage cluster health, disk's online/offline status, storage availability, and so on. The Ceph storage itself does not have a graphical interface for management. The following screenshot is what a typical Ceph command-line status looks like in order to check all the attached nodes and health of the storage cluster:

```
root@symmvm-01:/# ceph -s
    cluster 5f64b744-52d4-4ea9-95cc-6a760155c2f7
     health HEALTH_OK
     monmap e7: 7 mons at {0=192.101.60.1:6789/0,1=192.101.60.2:6789/0
92.101.60.42:6789/0,4=192.101.60.43:6789/0,5=192.101.60.4:6789/0,6=192
 epoch 162, quorum 0,1,2,3,4,5,6 0,1,2,3,4,5,6
     osdmap e5782: 21 osds: 21 up, 21 in
      pgmap v4100340: 840 pgs, 4 pools, 3777 GB data, 980 kobjects
            4703 GB used, 34296 GB / 38999 GB avail
                 840 active+clean
  client io 12966 kB/s rd, 221 kB/s wr, 196 op/s
```

With the recent version of Proxmox, Ceph has been included as a technology preview. Using API, Proxmox can now collect Ceph cluster data and display them through Proxmox GUI, as shown in the following screenshot:




Other NAS solutions such as FreeNAS, OpenMediaVault, and NAS4Free also have a nice, user-friendly GUI. The following screenshot is an example of the hard drive status from a FreeNAS GUI window:




# Local and shared storage comparison

The following table shows a comparison of both, the local and shared storage for quick reference:

| | Local storage | Shared storage |
|---|---|---|
| Virtual machine live migration | No | Yes |
| High Availability | No | Yes, when used in multinode shared storage |
| Cost | Lower | Significantly higher |
| I/O performance | Native disk drive speed | Slower than native disk drive speed |
| Skill requirements | No special storage skills required | Must be skilled on the shared storage option used |

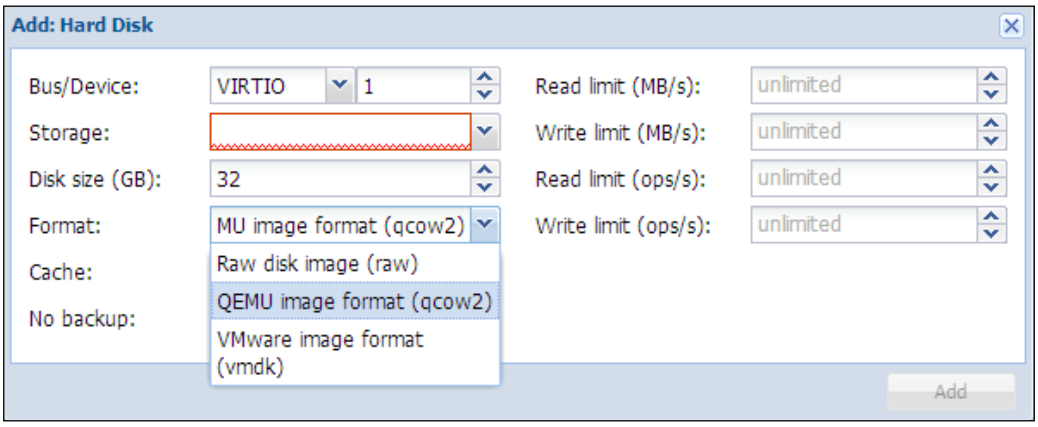
| | Local storage | Shared storage |
|---|---|---|
| Expandability | Limited to available drive bays of a node | Expandable over multiple nodes when multinode or distributed shared storage used |
| Maintenance complexity | Virtually maintenance free | Storage cluster requires regular monitoring |

# Virtual disk image

We now know the great benefits of using a shared storage system over local storage system. Now, let us take a look at the type of virtual disk image formats Proxmox supports. Virtual disk images are the medium where a virtual machine stores its data. If a VM configuration file is destroyed, we can just recreate it and then attach the saved virtual image to bring everything back online. But if the image itself is lost, so is the virtual machine. There are different types of virtual disk image formats available to be used with a virtual machine. It is essential to know the different types of image formats in order to have an optimal performing VM. Knowing the disk images also helps to prevent premature shortage of space by over provisioning virtual disks.

# Supported image formats

Proxmox supports `.raw`, `.qcow2`, and `.vmdk` virtual disk formats. Each format has its own set of strengths and weaknesses. The image format is usually chosen based on the function of the virtual machine, storage system in use, performance requirement, and available budget. The following screenshot shows the menu where we can choose an image type during a virtual disk creation through the GUI:



The following table is a brief summary of the different image formats and their possible usage:

| Image type | Storage supported | Strength | Weakness |
|---|---|---|---|
| `.qcow2` | NFS and directory | Allows dynamic virtual storage image files | Complex file formats with additional software layers |
| | | Stable and secure | High I/O overhead |
| | | Most features rich among image types | |
| `.raw` | LVM, RBD, iSCSI, and directory | No additional software layer. Direct access to image files | Fixed virtual image only. Cannot be used to store dynamic images |
| | | Stable, secure, and fastest | VM takes longer to back up due to the size of image files |
| `.vmdk` | NFS and directory | Works exceptionally well with VMware infrastructure | Additional software layer, thus slower performance |
| | | Allows dynamic virtual storage image files | Not completely tested with Proxmox |

Setting up virtual machines with the wrong image format is very forgiving. You can always convert these image types from one format to another. Conversion can be done from both, CLI and GUI. Virtual disk image conversion is explained later in this chapter.

# The .qcow2 image type

The `.qcow2` type is a very stable and matured VM image format. Proxmox completely supports this file format. A VM disk created using `.qcow2` is much smaller since by default it creates thin-provisioned disk images. For example, an Ubuntu VM created with 50 GB storage space may have an image file with a size around 1 GB. As a user stores data in the VM, this image file will grow gradually. The `.qcow2` image format allows an administrator to overpopulate shared storage by over provisioning VMs with the `.qcow2` image file. This is not a problem if users do not store data rapidly and fill up their virtual machines. In that case, the shared storage will run out of space to accommodate all the growing virtual image files. Available storage space should be regularly monitored in such an environment. It is a good practice to add additional storage space when the overall storage space consumption reaches around 80 percent.

**Thin provisioning** is when the virtual disk image file does not preallocate all the blocks, thus keeping the size of the image file to only what we want. As more data is stored in the virtual machine, the thin provisioned image file grows till it reaches the maximum size allocated. **Thick provisioning**, on the other hand, is when the virtual disk image file preallocates all the blocks, thus creating an image file which is exactly of the size set prior to creating the virtual disk image file.

The `.qcow2` format also has very high I/O overhead due to its additional software layer. Thus, it is a bad choice of image format for a VM such as a database server. Any data being read or written into the image format goes through the `qcow2` software layer, which increases the I/O making it slower. A backup created from a `qcow2` image can only be restored to an NFS or local directory.

When budget is the main concern and storage space is very limited, `qcow2` is an excellent choice. This image type supports KVM live snapshots to preserve states of virtual machines.

# The .raw image type

The `.raw` image type is also a very stable and matured VM image format. Its primary strength lies in performance. There is no additional software layer for data to go through. VM has direct pass-through access to the raw file, which makes it much faster. Also, there is no software component attached to it, so it is much less problem prone. The raw format can only create a fixed-size or thick-provisioned VM image file. For example, an Ubuntu VM created with 50 GB storage space will have a 50 GB image file. This helps an administrator to know exactly how much storage is in use, so there is no chance of an uncontrolled out-of-storage situation.

The `.raw` type is the preferred file format for all Proxmox VMs. A raw image format VM can be restored to just about any storage type. In a virtual environment, additional virtual disk image files can be added to a virtual machine at any time. So, it is not necessary to initially allocate a larger size `.raw` virtual disk image file with possible future growth in mind. The VM can start with a smaller `.raw` image file and add more disk images as needed. For example, a VM with 50 GB data starts with an 80 GB `.raw` image file. It then adds more virtual disk images at a 50 GB increment as need arises. Proxmox allows quite a few additional virtual drives to be added to a VM. The following is a table of the maximum number of allowed disk device per VM by Proxmox:

| Bus/device type | Maximum allowed |
|---|---|
| IDE | 3 |
| SATA | 5 |
| VirtIO | 15 |
| SCSI | 13 |

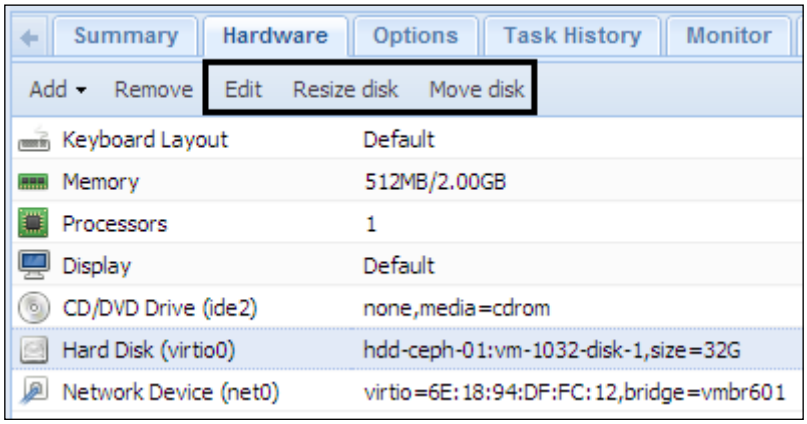
Since all the `.raw` disk image files are preallocated, there are no risks of over provisioning beyond the total available storage space. KVM live snapshots are also supported by the `.raw` image format.

# The .vmdk image type

The `.vmdk` image format is very common in the VMware infrastructure. The only advantage of Proxmox supporting `.vmdk` is the ease of VM migration from VMware to Proxmox cluster. A VM created in VMware with a `.vmdk` image format can easily be configured for use in the Proxmox cluster and converted. There are no possible reasons to keep using `.vmdk` without converting it to `.raw` or `.qcow2`, except during a transitional period such as converting virtual machines from VMware infrastructure.

# Image file manipulation

A Proxmox virtual image file can be manipulated from both, WebGUI and CLI. The WebGUI allows the resize (increase only), move, and delete options.



When a disk image file is selected, the **Resize disk** and **Move disk** buttons become available. Virtual machine image files can also be manipulated using CLI commands. The following are a few examples of the most common commands to delete, convert, and resize an image file:

| Usage: `qemu-img command [command options]` | |
|---|---|
| `#qemu-img create –f <type> -o <filename> <size>` | Create image file |
| `#qemu-img create –f raw test.raw 1024M` | |
| `#qemu-img convert <source> -O <type> <destination>` | Convert image file |
| `#qemu-img convert test.vmdk –O qcow2 test.qcow2` | |
| `#qemu-img resize <filename> <+\|-><size>` | Resize image file |
| `#qemu-img resize test.qcow2 +1024M` | |

# Resizing virtual disk image

Resize disk only supports increasing the size of the virtual disk image file. It has no shrink function. The Proxmox resize option only adjusts the size of the virtual disk image file. After any resizing, the partition must be adjusted from inside the VM. The safest way to resize partitions is to boot a Linux-based virtual machine with a partitioning ISO image such as **gparted** (`http://gparted.org/download.php`), and then resize the partitions using gparted graphical interface. It is also possible to perform an online partition resizing while the virtual machine is powered on. Resizing a virtual disk image file involves the following three steps:

1. Resize the virtual disk image file in Proxmox using any one of the following:
    - **From GUI**: Select the virtual disk and then click on **Resize disk**
    - **From CLI**: Run the command `# qm resize <vm_id> <virtual_disk> +<size>G`
2. Resize the partition of the virtual disk image file from inside VM using any one of the following:
    - For Windows VM: Resize the disk by navigating to **Administrative Tools | Computer Management**
    - For Linux VM with RAW partition, use the following command:

      **`# cfdisk <disk_image>`**
    - For Linux VM with LVM partition, use the following command:

      **`# cfdisk </dev/XXX/disk_image>`**

- For Linux VM with QCOW2 partition, use the following command lines:

```
# apt-get install nbd-client
# qemu-ndb -connect /dev/nbd0     <disk_image>
# cfdisk /dev/nbd0
# qemu-nbd -d /dev/nbd0
```

3. Resize the filesystem in the partition of the virtual disk image file using any one of the following methods:

   - For a Linux client with LVM:

```
# pvscan   (find PV name)
# pvresize /dev/xxx  (/dev/xxx found from pvscan)
# lvscan   (find LVname)
# lvresize -L+<size>G /dev/xxx/lv_<disk>
```

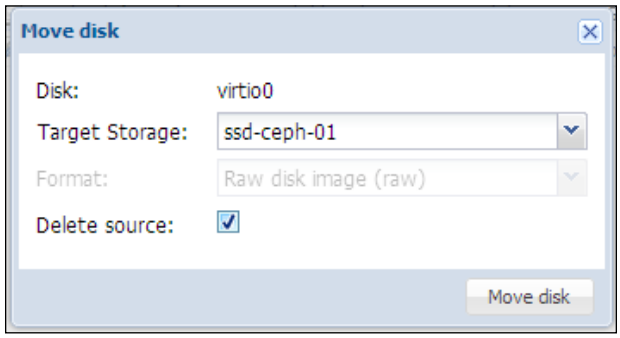
   - To use 100 percent free space:

```
# lvresize -l +100%FREE /dev/xxx/lv_<disk>
# resize2fs /dev/xxx/lv_<disk> (resize filesystem)
```

Steps 2 and 3 are only necessary if online resizing is done without shutting down a VM. If gparted or another bootable partitioning medium is used, then only step 1 is needed before booting the VM with ISO.

## Moving a virtual disk image

Move disk allows the image file to be moved to a different storage or converted to a different image type, as shown in the following screenshot:



In the **Move disk** option menu, just select the **Target Storage** and **Format** type, and then click on **Move disk** to move the image file. Moving can be done live without shutting down the VM. Checking **Delete source** will delete the source image file after the move is complete.

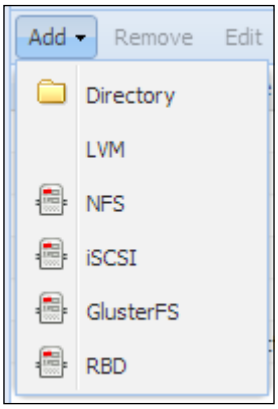
The **Format** type in the **Move disk** option will be greyed out if the destination storage only supports one image format type. In the previous example, **ssd-ceph-01** is an RBD storage in Ceph Pool. Since RBD only supports the RAW format, the format type has been greyed out automatically. Learn more about Ceph in *Chapter 7*, *High Availability Storage for High Availability Cluster*.

# Storage types in Proxmox

In the previous section, we saw the image types that can be used in a Proxmox cluster to create a virtual machine. We will now look at different storage types where we can store those image files. Proxmox supports a wide selection of storage types for virtual machine storing:

- Directory: This is mostly a locally mounted folder
- LVM: This stands for the Logical Volume Management group
- NFS: This stands for the Network File System share
- iSCSI: This stands for the Internet SCSI target
- GlusterFS: This stands for Gluster File System
- RBD: This stands for RADOS Block Devices

The storage types are shown in the following screenshot:



# Directory

Directory storage is mostly a mounted folder on the Proxmox local node. It is mainly used as local storage. By default, this location is mounted under `/var/lib/vz`.

Any VM stored in this Directory storage does not allow live migration. The VM must be stopped before migrating to another node. All virtual disk image file types can be stored in the Directory storage.

# Logical Volume Management

**Logical Volume Management** (**LVM**) provides a method of storage space allocation by using one or more disk partitions or drives as the underlying base storage. LVM storage requires a base storage to be set up and function properly. In Proxmox, the base storage is provided through an iSCSI device. LVM allows scalable storage space since the base storage could be on the same node or on a different one. LVM storage only supports a RAW virtual disk image format. For more details on LVM, visit the link `http://en.wikipedia.org/wiki/Logical_Volume_Manager_(Linux)`.

# Network File System

**Network File System** or **NFS** in short is a well-matured filesystem protocol originally developed by Sun Microsystems in 1984. Currently, Version 4 of the NFS protocol is in effect. But it was not as widely accepted as Version 3 due to a few compatibility issues. But the gap is closing fast between Version 3 and 4. Proxmox, by default, uses Version 3 of the NFS protocol, while administrators can change to Version 4 through the use of option in `storage.cfg`. NFS storage can store the `.qcow2`, `.raw`, and `.vmdk` image formats, providing versatility and flexibility in a clustered environment. NFS is also the easiest to set up and requires the least amount of upfront hardware cost, thus allowing a budget-conscious small business to get their hands on a stable shared storage system.

Care should be taken when using NFS Version 4 instead of 3 in Proxmox. There are still few bugs that exist in NFSv4, such as kernel panic during system startup while mounting the NFSv4 share.

Although NFS is a long-standing storage type, supported by just about any Linux-based operating system, NFS should not be used for VMs requiring a high performing I/O. For in-depth details on NFS, visit the link `http://en.wikipedia.org/wiki/Network_File_System`.

# RADOS Block Device

**RADOS Block Device** or **RBD** storage is provided by Ceph distributed storage system. It is the most complex storage system, which requires multiple nodes to be set up. By design, Ceph is a distributed storage system and can be spanned over several dozen nodes. The RBD storage can only store raw image formats. To expand a Ceph cluster, simply add a hard drive or a node and let Ceph know about the new addition. Ceph will automatically rebalance data to accommodate the new hard drive or node. Ceph can be scaled to several petabytes or more. Ceph also allows multiple pool creations for different disk drives. For example, we can store database servers' VM images on an SSD-driven pool and backup server images on a slower-spinning drive pool. Ceph is the recommended storage system for medium to large cluster environments due to its resiliency against data loss and the simplicity of storage expandability.

From Proxmox VE Version 3.2, the Ceph server has been integrated into Proxmox to coexist on the same node. The ability to manage Ceph clusters through the Proxmox GUI has also been added. In *Chapter 7*, *High Availability Storage for High Availability Cluster*, we will see Ceph in greater details. To know more about Ceph storage, visit the link `http://ceph.com/docs/master/start/intro/`.

# GlusterFS

GlusterFS is a powerful, distributed filesystem, which can be scaled to several Petabytes under a single mount point. Gluster is a fairly new addition into Proxmox, which allows GlusterFS users to take full advantage of the Proxmox cluster. GlusterFS uses the Stripe, Replicate, or Distribute mode to store files. Although the Distribute mode offers the option of scalability, it should be noted that in the Stripe mode when a GlusterFS node goes down, all the files in that server become inaccessible. Meaning, if a particular file is saved by the GlusterFS translator into that server, only that node holds the entire data of that file. Even though all other nodes are operational, that particular file will no longer be available. GlusterFS can be scaled up to petabytes inside a single mount. The GlusterFS storage can be set up with just two nodes and supports NFS, thus allowing to store any image file format. To know more details on GlusterFS, visit `http://www.gluster.org/community/documentation/index.php/GlusterFS_Concepts`.

# Noncommercial/commercial storage options

We have discussed which virtual machine image formats and storage types are supported by Proxmox. To better acquaint ourselves for test or practice labs, we are now going to take a look at what noncommercial and commercial options we have out there to set up a storage system for a Proxmox-clustered environment. By noncommercial, I mean they are free without any primary features missing and without any trial limits.

These noncommercial options will allow you to set up a fully functional shared storage system with some hard work. Commercial versions usually come with full support from the provider company and in some cases the ongoing **Service Level Agreement** (**SLA**) contract. The following list is by no means a complete one, but a guideline to point you to the direction you need to go when planning and implementing a Proxmox cluster environment. Each of these products can provide everything you need to set up a shared storage:

| **Noncommercial** | | **Commercial** | |
|---|---|---|---|
| Solaris+napp-IT | `www.napp-it.org` | Nexenta | `www.nexenta.com` |
| FreeNAS | `www.freenas.org` | FalconStor | `www.falconstor.com` |
| GlusterFS | `www.gluster.org` | EMC2 | `www.emc.com` |
| Ceph | `www.ceph.com` | Yottabyte | `www.yottabyte.com` |
| | | Open-E DSS | `www.open-e.com` |
| | | NetApp | `www.netapp.com` |

A question often asked is "Can I set up a Proxmox production cluster environment by using only noncommercial solutions?" The short answer is, "YES!"

It is indeed possible to create an entire complex Proxmox cluster by using only noncommercial storage solutions. But you have to be prepared for the unexpected and spend a significant amount of time learning the system. Commercial solutions aside, just studying a system will give an administrator an advantage when unforeseen issues arise. The main difference between these noncommercial and commercial solutions is the company support behind it. Typically, noncommercial solutions only have community-driven support through forums and message boards. Commercial offerings come with technical support with the response time varying from anything between immediate time to 24 hours.

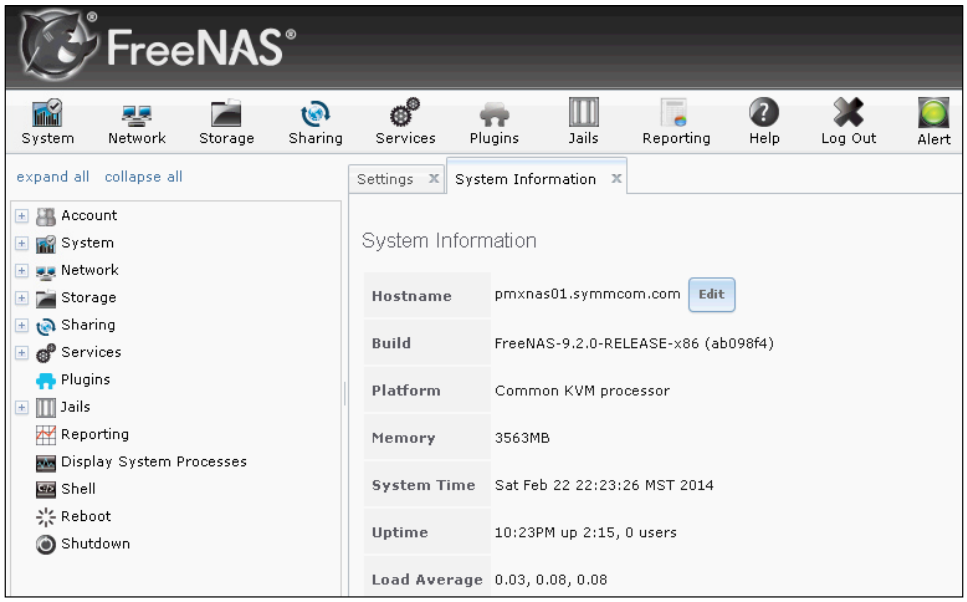
Trade-off of noncommercial open source solutions is the money that is saved, which usually gets exchanged with the time spent on research and mistakes.

In this book, we will look at two noncommercial storage options, FreeNAS and Ceph. Both are stable enough to set up a fully functional Proxmox cluster. FreeNAS is more targeted for small businesses or home lab usage, while Ceph is capable enough to take on just about any simple or complex cluster environment. There are some scenarios where FreeNAS has been used as backup servers to back up virtual machines. Multiple FreeNAS servers can be set up to perform separate backup tasks.

Although Ceph is on the noncommercial side of the table, it also has paid commercial support. Inktank, the maker of Ceph has complete details of commercial support subscription. Just visit `www.inktank.com` for more information on Inktank.

# FreeNAS – budget shared storage

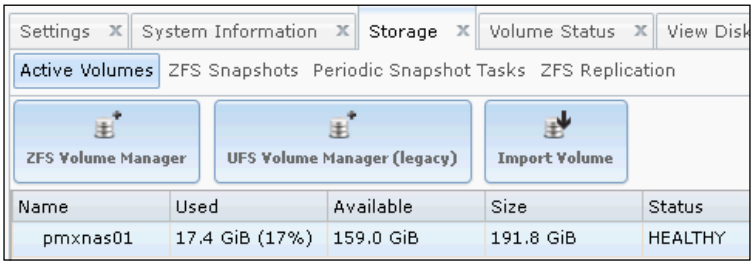
For an environment where the budget is extremely tight and minor downtime is not a big issue, FreeNAS is very much capable to provide all shared storage needs. It is easy to set up, feature rich, and best of all, free. The following screenshot is a typical FreeNAS dashboard presented upon login. The screenshot has been taken from FreeNAS 9.2.1.5. Other versions may be slightly different.



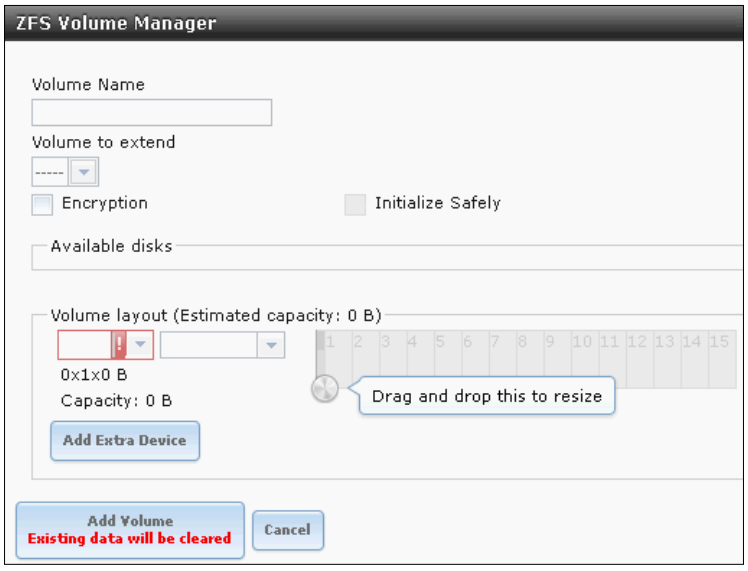
FreeNAS provides many services such as AFP, NFS, CIFS, iSCSI, FTP, SSH, SNMP, RSYNC, and so on. FreeNAS can take advantage of ZFS, which offers a stable platform to build a virtual environment on.

ZFS was originally designed by Sun Microsystems combining filesystem and logical volume manager to create a new type of filesystem, which supports high storage capacities, provides protection against data corruption, performs continuous integrity checking, and so on. To know more about ZFS, visit `http://en.wikipedia.org/wiki/ZFS`.

At a glance from the FreeNAS volume status page, as shown in the following screenshot, an administrator can check the health of all the volumes in FreeNAS:




The whole of FreeNAS is menu driven and organized with a very user-friendly graphical interface. All the tasks of storage management can be accomplished without a single CLI command. The following screenshot is a menu-driven window to create a new ZFS volume:



FreeNAS also supports advanced networking features such as network bonding to increase storage node bandwidth. The only drawback of noncommercial FreeNAS is that it does not have High Availability or an out-of-the-box clustering option. With some work, it is possible to set up High Availability FreeNAS cluster using heartbeat, but it is not stable to be used in a production environment. But a setup of two replicated non-High Availability FreeNAS nodes can be easily achieved using RSYNC built-in with FreeNAS.

For those who really want to try a shared storage such as FreeNAS but with availability, give NAS4Free (`http://www.nas4free.org`) a try. By following the instructions at `http://blackcatsoftware.us/inprogress-configure-nas4free-high-availabilty-storage-carphastzfs/`, High Availability shared storage can be created with NAS4Free.

Despite the lack of High Availability in FreeNAS, it is a better choice over NAS4Free from the stability point of view. NAS4Free is built on the very latest kernel and thus prone to have issues due to bugs.

To take full potential advantage of FreeNAS, it is recommended to use the ZFS volume system. ZFS has very high resiliency against data loss. FreeNAS also supports Stripe, Mirror, RAIDZ1=1 disk failure, RAIDZ2 =2 disks failure, and RAIDZ3=3 disks failure, providing a high-level of redundancy. In case of total node failure, all the hard drives can be moved to a new node with FreeNAS already set up and brought by importing into new FreeNAS.

One thing to keep in mind is that the ZFS filesystem consumes a large amount of memory for caching. Any leftover memory will be consumed by ZFS to provide fast I/O. So, if you see that your FreeNAS node is running out of memory, do not panic. It is not always necessary to go out and buy more RAM every time FreeNAS runs out of memory. A general rule of thumb is 1 GB of RAM for every TB of hard drive space. FreeNAS will perform satisfactorily if it has a minimum of 8 GB RAM. If RAM is extremely scarce, try to use UFS instead of the ZFS volume. Memory consumption ratio is much smaller in UFS than ZFS.

Like Ceph or GlusterFS, it is also possible to create different volumes on different pools of disk drive. For example, a volume of SSD pool can be created just by selecting SSD drives, while an HDD volume can comprise of the remaining spinning hard disk drives. This will allow testing different pool performances while giving somebody a chance to learn about pooling and data tiering. The only drawback of FreeNAS in this scenario is pools cannot be spanned over multiple nodes as Ceph can.

The most common usage of FreeNAS with the Proxmox cluster is for backup purpose. Since backup nodes do not need to be high performing nodes and downtime is tolerable, FreeNAS is an excellent choice for this task. With two FreeNAS backup nodes, VM backups can be replicated to each FreeNAS node and provide backup redundancy. NFS share is most commonly used for a Proxmox VM backup.

FreeNAS has an excellent built-in reporting system, which shows minute-by-minute information of different critical parts of the FreeNAS node. Data is presented in a very easy way to understand the graphical format. Information about CPU, memory, storage quota, network bandwidth, and so on can prove very valuable when there is an issue. Sometimes, this data can also prevent disasters by spotting symptoms such as out of memory or high CPU consumption continuously. The following screenshot is an example of the **Reporting** system showing **CPU Usage** and **System Load**:


As mentioned earlier, FreeNAS is a great choice to learn what shared storage is all about and have a fully functional Proxmox cluster set up and running in no time. It can also be used in a small business setup where the budget is extremely limited and redundancy is not the top priority.

# Summary

In this chapter, we took a look at what storage options are supported by Proxmox and their advantages and disadvantages. We also saw the types of virtual image files that can be used with Proxmox and when to use them. Storage is an important component for Proxmox clustering. Since a storage system is where virtual machines are created and operated from, a properly implemented storage system is very critical to make any cluster a successful one. With proper planning of different storage needs and choosing the right format and option, a lot of hassle and frustration can be minimized later on.

In the previous chapter, we very briefly saw the arguments usage in the Proxmox configuration files. In the next chapter, *Chapter 4*, *A Virtual Machine for a Virtual World*, we will analyze those arguments in greater detail to extend the cluster ability beyond just its default. Using arguments, we can enable features such as sound, USB, backup performance, and so on for a virtual machine.

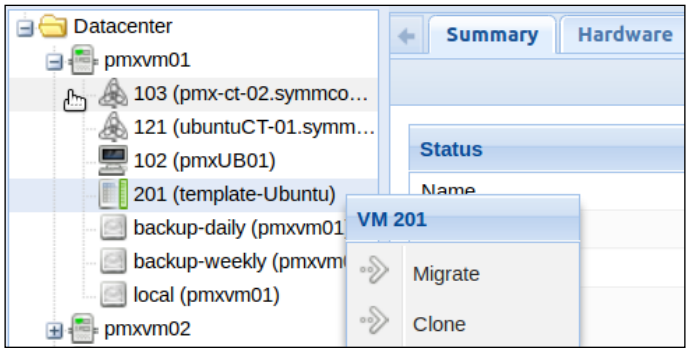
# 4
# A Virtual Machine for a Virtual World

We are about to embark upon a new kind of virtual world, which is beyond just basic. Through previous chapters, we have familiarized ourselves with Proxmox up close and personal. We saw what a Proxmox GUI looks like and got our hands dirty with the Proxmox configuration files and directory structure. We also learned what shared storage system is and how it is integrated with the Proxmox cluster. In this chapter, we are going to take it one step further by covering at the following topics:

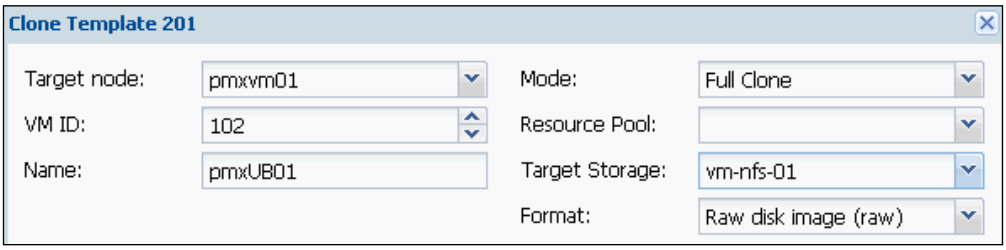
- Advanced configuration options for virtual machines
- Nested virtual environment
- Proxmox backup/restore system
- Virtual machine snapshots
- Command-line backup

# Creating a VM from a template

Let us start by creating our second virtual machine from the Ubuntu template we created in the *Transforming VM into a Template* section in *Chapter 1*, *Dive into the Virtual World with Proxmox*. Right-click on the template and select **Clone**, as shown in the following screenshot:




Use the settings shown in the following screenshot for the new virtual machine. You can also use any virtual machine name you like. A VM name can only be alphanumeric without any special characters.




You can also use any other VM you have already created in your own virtual environment. Access the virtual machine through the Proxmox console after cloning and setting up network connectivity such as IP address, hostname, and so on. For our Ubuntu virtual machine, we are going to edit `interfaces` in `/etc/network/`, `hostname` in `/etc/`, and `hosts` in `/etc/`.

# Advanced configuration options for a VM

We will now look at some of the advanced configuration options we can use to extend the capability of a KVM virtual machine.

# The hotplugging option for a VM

Although it is not a very common occurrence, a virtual machine can run out of storage unexpectedly whether due to over provisioning or improper storage requirement planning. For a physical server with hot swap bays, we can simply add a new hard drive and then partition it, and you are up and running. Imagine another situation when you have to add some virtual network interface to the VM right away, but you cannot afford shutting down the VM to add the vNICs. The hotplug option also allows hotplugging virtual network interfaces without shutting down a VM.

Proxmox virtual machines by default do not support hotplugging. There are some extra steps needed to be followed in order to enable hotplugging for devices such as virtual disks and virtual network interfaces. Without the hotplugging option, the virtual machine needs to be completely powered off and then powered on after adding a new virtual disk or virtual interface. Simply rebooting the virtual machine will not activate the newly added virtual device. In Proxmox 3.2 and later, the hotplug option is not shown on the Proxmox GUI. It has to be done through CLI by adding options to the `<vmid>.conf` file. Enabling the hotplug option for a virtual machine is a three-step process:

1. Shut down VM and add the hotplug option into the `<vmid>.conf` file.
2. Power up VM and then load modules that will initiate the actual hotplugging.
3. Add a virtual disk or virtual interface to be hotplugged into the virtual machine.

# The hotplugging option for <vmid>.conf

Shut down the cloned virtual machine we created earlier and then open the configuration file from the following location. Securely log in to the Proxmox node or use the console in the Proxmox GUI using the following command:

```
# nano /etc/pve/nodes/<node_name>/qemu-server/102.conf
```

With default options added during the virtual machine creation process, the following code is what the VM configuration file looks like:

```
ballon: 512
bootdisk: virtio0
cores: 1
ide2: none, media=cdrom
kvm: 0
memory: 1024
name: pmxUB01
net0: e1000=56:63:C0:AC:5F:9D,bridge=vmbr0
ostype: l26
sockets: 1
virtio0: vm-nfs-01:102/vm-102-disk-1.qcow2,format=qcow2,size=32G
```

Now, at the bottom of the `102.conf` configuration file located under `/etc/pve/nodes/<node_name>/qemu-server/`, we will add the following option to enable hotplugging in the virtual machine:

```
hotplug: 1
```

Save the configuration file and power up the virtual machine.

To find out more details on the possible options available for Proxmox virtual machines, refer to *Chapter 2*, *Proxmox Under the Hood*, and visit Proxmox Wiki at `https://pve.proxmox.com/wiki/Manual:_vm.conf`.

## Loading modules

After the hotplug option is added and the virtual machine is powered up, it is now time to load two modules into the virtual machine, which will allow hotplugging a virtual disk anytime without rebooting the VM. Securely log in to VM or use the Proxmox GUI console to get into the command prompt of the VM. Then, run the following commands to load the `acpiphp` and `pci_hotplug` modules. Do not load these modules to the Proxmox node itself:

```
# sudo modprobe acpiphp
# sudo modprobe pci_hotplug
```

The `acpiphp` and `pci_hotplug` modules are two hot plug drivers for the Linux operating system. These drivers allow addition of a virtual disk image or virtual network interface card without shutting down the Linux-based virtual machine.

The modules can also be loaded automatically during the virtual machine boot by inserting them in `/etc/modules`. Simply add `acpiphp` and `pci_hotplug` on two separate lines in `/etc/modules`.

## Adding virtual disk/vNIC

After loading both the `acpiphp` and `pci_hotplug` modules, all that remains is adding a new virtual disk or virtual network interface in the virtual machine through a web GUI. On adding a new disk image, check that the virtual machine operating system recognizes the new disk through the following command:

```
#sudo fdisk -l
```

For a virtual network interface, simply add a new virtual interface from a web GUI and the operating system will automatically recognize a new vNIC. After adding the interface, check that the vNIC is recognized through the following command:

```
#sudo ifconfig –a
```

Please note that while the hotplugging option works great with Linux-based virtual machines, it is somewhat problematic on Windows XP/7-based VMs. Hotplug seems to work great with both 32- and 64-bit versions of the Windows Server 2003/2008/2012 VMs. The best practice for a Windows XP/7-based virtual machine is to just power cycle the virtual machine to activate newly added virtual disk images. Forcing the Windows VM to go through hotplugging will cause an unstable operating environment. This is a limitation of the KVM itself.

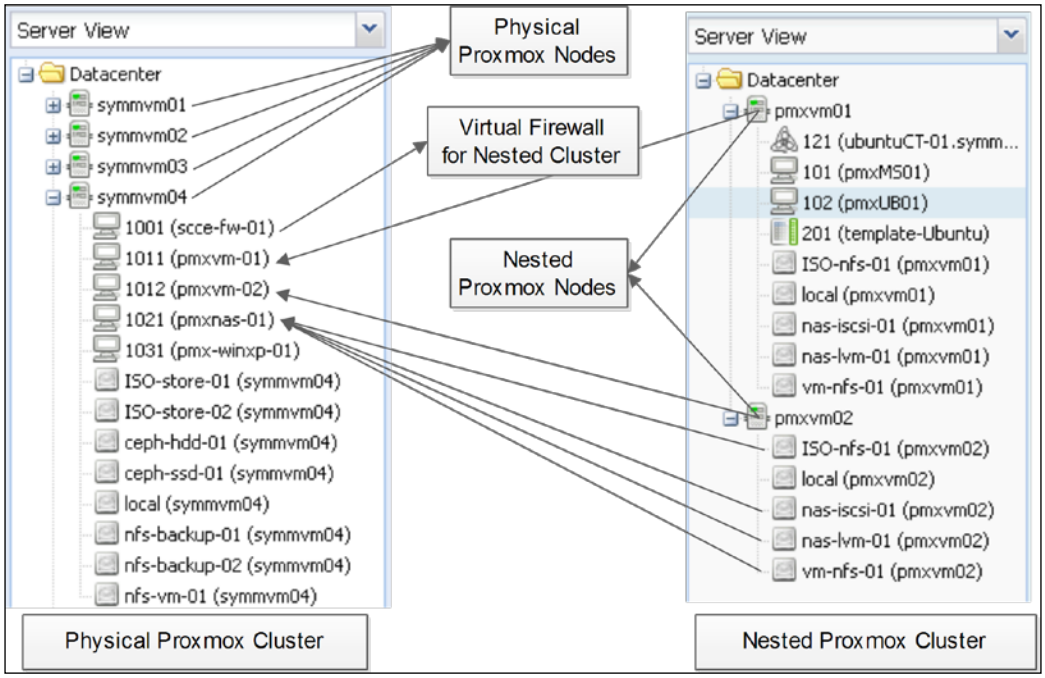
# Nested virtual environment

In simple terms, a virtual environment inside another virtual environment is known as a nested virtual environment. If the hardware resource permits, a nested virtual environment can open up whole new possibilities for a company. The most common scenario of a nested virtual environment is to set up a fully isolated test environment to test software such as hypervisor, or operating system updates/patches before applying them in a live environment.

A nested environment can also be used as a training platform to teach computer and network virtualization, where students can set up their own virtual environment from the ground without breaking the main system. This eliminates the high cost of hardware for each student or for the test environment. When an isolated test platform is needed, it is just a matter of cloning some real virtual machines and giving access to authorized users. A nested virtual environment has the potential to give the network administrator an edge in the real world by allowing cost cutting and just getting things done with limited resources.

One very important thing to keep in mind is that a nested virtual environment will have a significantly lower performance than a real virtual environment. If the nested virtual environment also has virtualized storage, performance will degrade significantly. The loss of performance can be offset by creating a nested environment with an SSD storage backend. When a nested virtual environment is created, it usually also contains virtualized storage to provide virtual storage for nested virtual machines. This allows for a fully isolated nested environment with its own subnet and virtual firewall.

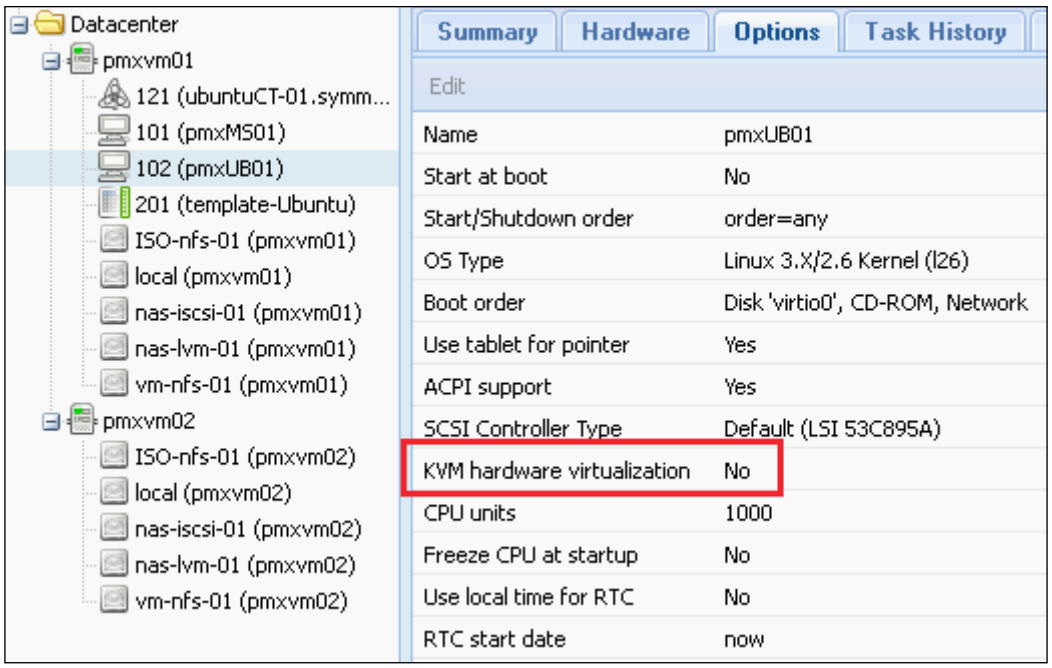
There are many debates about the viability of a nested virtual environment. Both pros and cons can be argued equally. But it will come down to the administrator's grasp on his or her existing virtual environment and good understanding of the nature of requirement. The cluster that we have been building through out this book so far is a nested virtual cluster itself, specifically created to write this book. This allowed us to build a fully functional Proxmox cluster from the ground up without using additional hardware. The following screenshot is a side-by-side representation of a nested virtual environment scenario:




In the previous comparison, on the right-hand side we have our basic cluster we have been building so far. On the left-hand side we have the actual physical nodes and virtual machines used to create the nested virtual environment to help us along the book.

Our nested cluster is completely isolated from the rest of the physical cluster with a separate subnet. Internet connectivity is provided to the nested environment by using a virtualized firewall **1001-scce-fw-01**.

Like the hotplugging option, nesting is also not enabled in the Proxmox cluster by default. Enabling nesting will allow nested virtual machines to have KVM hardware virtualization, which increases the performance of nested virtual machines. To enable KVM hardware virtualization, we have to edit the `modules` in `/etc/` of the physical Proxmox node and `<vmid>.conf` of the virtual machine. We can see that the option is disabled for our cloned nested virtual machine in the following screenshot:




# Enabling KVM hardware virtualization

KVM hardware virtualization can be added just by performing the following few additional steps:

1. In each Proxmox node, add the following line in the `/etc/modules` file:

   ```
   kvm-amd nested=1
   ```

2. Migrate or shut down all virtual machines of Proxmox nodes and then reboot.

3. After the Proxmox nodes reboot, add the following argument in the `<vmid>.conf` file of the virtual machines used to create a nested virtual environment:

   ```
   args: -enable-nesting
   ```

4. Enable KVM hardware virtualization from the virtual machine option menu through GUI. Restart the nested virtual machine.

# Network virtualization

Network virtualization is a software approach to set up and maintain network without physical hardware. Proxmox has great features to virtualize the network for both real and nested virtual environments. By using virtualized networking, management becomes simpler and centralized. Since there is no physical hardware to deal with, the network ability can be extended within a minute's notice. Especially in a nested virtual environment, the use of virtualized network is very prominent. We will look at network virtualization at greater length in the next chapter. But it suffices to say that in order to set up a successful nested virtual environment, a better grasp of the Proxmox network feature is required. With the introduction of Open vSwitch (`www.openvswitch.org`) in Proxmox 3.2 and later, network virtualization is now much more efficient.

# Backing up a virtual machine

A good backup strategy is the last line of defense against disasters, such as hardware failure, environmental damages, accidental deletions, and misconfigurations. In a virtual environment, a backup strategy turns into a daunting task because of the number of machines needed to be backed up. In a busy production environment, virtual machines may be created and discarded whenever needed or not needed. Without a proper backup plan, the entire backup task can go out of control. Gone are those days when we only had few server hardware to deal with and backing them up was an easy task. Today's backup solutions have to deal with several dozens or possibly several hundred virtual machines.

Depending on the requirement, an administrator may have to backup all the virtual machines regularly instead of just the files inside them. Backing up an entire virtual machine takes up a very large amount of space after a while depending on how many previous backups we have. A granular file backup helps to quickly restore just the file needed but sure is a bad choice if the virtual server is badly damaged to a point that it becomes inaccessible. Here, we will see different backup options available in Proxmox, their advantages, and disadvantages.

# Proxmox backup and snapshot options

Proxmox has the following two backup options:

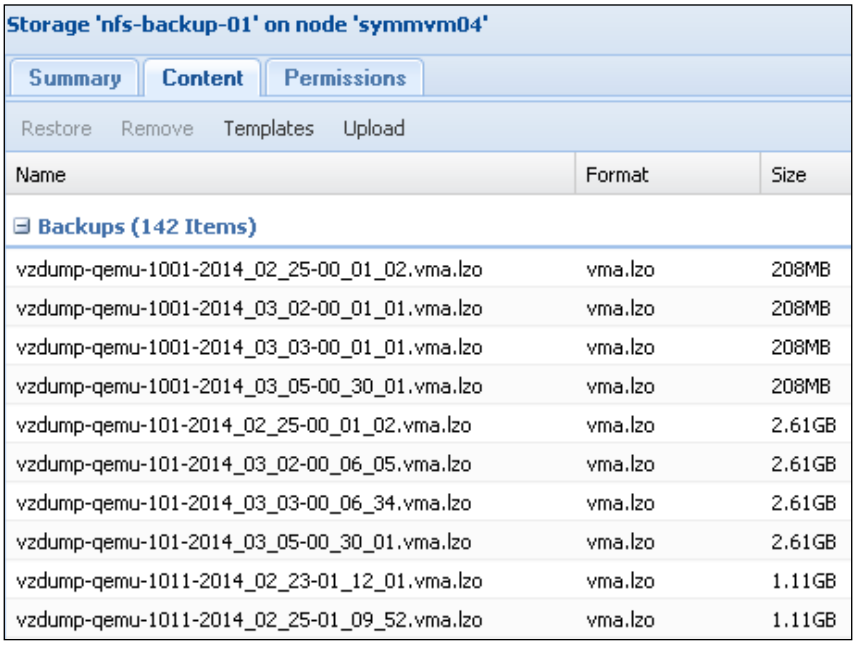
- Full backup: This backs up the entire virtual machine.
- Snapshot: This only creates a snapshot image of the virtual machine. Proxmox 3.2 and above can only do a full backup and cannot do any granular file backup from inside a virtual machine. Proxmox also does not use any backup agent.

## Backing up a VM with a full backup

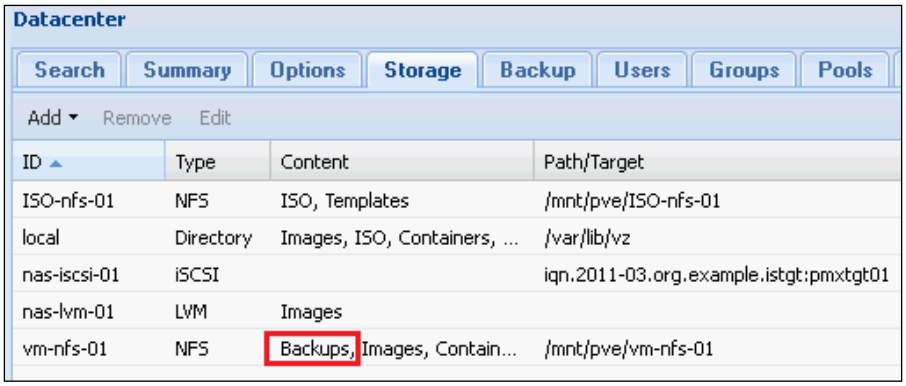
All full backups are in the `.tar` format containing both the configuration file and virtual disk image file. The TAR file is all you need to restore the virtual machine on any nodes and on any storage. Full backups can also be scheduled on a daily and weekly basis. Full virtual backup files are named based on the following format:

```
vzdump-qemu-<vm_id>-YYYY_MM_DD-HH_MM_SS.vma.lzo
```

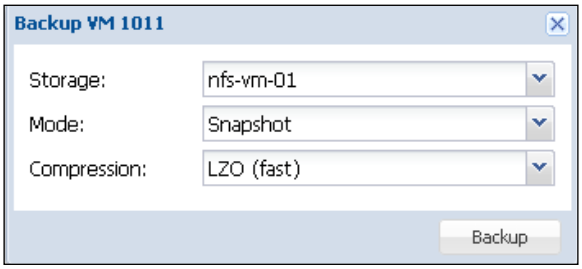
The following screenshot shows what a typical list of virtual machine backups looks like:



Proxmox 3.2 and above cannot do full backups on LVM and Ceph RBD storage. Full backups can only occur on local, Ceph FS, and NFS-based storages, which are defined as backup during storage creation. Please note that Ceph FS and RBD are not the same type of storage even though they both coexist on the same Ceph cluster. In *Chapter 7*, *High Availability Storage for High Availability Cluster*, we will create both RBD and Ceph FS in a Ceph cluster. The following screenshot shows the storage feature through the Proxmox GUI with backup-enabled attached storages:

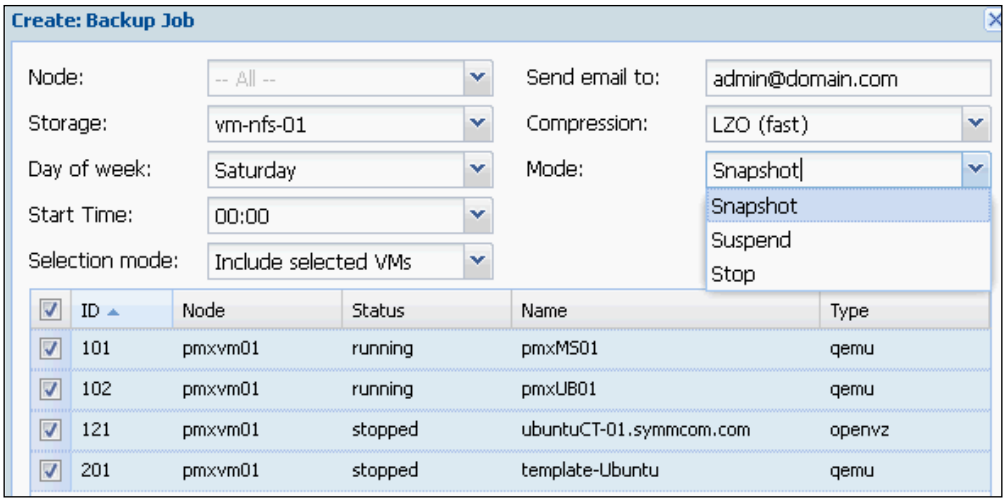



The backup menu in Proxmox is a true example of simplicity. With only three choices to select, it is as easy as it can get. The following screenshot is an example of a Proxmox backup menu. Just select the backup storage, backup mode, and compression type and that's it:




## Creating a schedule for Backup

Schedules can be created from the virtual machine backup option. We will see each option box in detail in the following sections. The options are shown in the following screenshot:




## Node

By default, a backup job applies to all nodes. If you want to apply the backup job to a particular node, then select it here. With a node selected, backup job will be restricted to that node only. If a virtual machine on node 1 was selected for backup and later on the virtual machine was moved to node 2, it will not be backed up since only node 1 was selected for this backup task.

## Storage

Select a backup storage destination where all full backups will be stored. Typically an NFS server is used for backup storage. They are easy to set up and do not require a lot of upfront investment due to their low performance requirements. Backup servers are much leaner than computing nodes since they do not have to run any virtual machines. Backups are supported on local, NFS, and Ceph FS storage systems. Ceph FS storages are mounted locally on Proxmox nodes and selected as a local directory. Both Ceph FS and RBD coexist on the same Ceph cluster. More details about Ceph/RBD/Ceph FS are given in *Chapter 7*, *High Availability Storage for High Availability Cluster*.

## Day of Week

Select which day or days the backup task applies to. Days' selection is clickable in a drop-down menu. If the backup task should run daily, then select all the days from the list.

## Start Time

Unlike Day of Week, only one time slot can be selected. Multiple selections of time to backup different times of the day are not possible. If the backup must run multiple times a day, create a separate task for each time slot.

## Selection mode

The **All selection** mode will select all the virtual machines within the whole Proxmox cluster. The **Exclude selected VMs** mode will back up all VMs except the ones selected. **Include selected VMs** will back up only the ones selected.

## Send email to

Enter a valid e-mail address here so that the Proxmox backup task can send an e-mail upon backup task completion or if there was any issue during backup. The e-mail includes the entire log of the backup tasks. It is highly recommended to enter the e-mail address here so that an administrator or backup operator can receive backup task feedback e-mails. This will allow us to find out if there was an issue during backup or how much time it actually takes to see if any performance issue occurred during backup. The following screenshot is a sample of a typical e-mail received after a backup task:

| VMID | NAME | STATUS | TIME | SIZE | FILENAME |
|---|---|---|---|---|---|
| 122 | VM 122 | FAILED | 00:00:00 | unable to find VM '122' | |
| 151 | ceph-admin-01 | OK | 00:02:04 | 2.55GB | /mnt/pve/nfs-backup-02/dump/vzdump-qemu-151-2014_03_17-22_45_02.vma.lzo |
| 152 | ceph-mds-01 | OK | 00:01:35 | 2.81GB | /mnt/pve/nfs-backup-02/dump/vzdump-qemu-152-2014_03_17-22_47_06.vma.lzo |
| 153 | ceph-mds-02 | OK | 00:01:06 | 1.99GB | /mnt/pve/nfs-backup-02/dump/vzdump-qemu-153-2014_03_17-22_48_41.vma.lzo |
| 154 | ceph-mon-01 | OK | 00:01:55 | 2.95GB | /mnt/pve/nfs-backup-02/dump/vzdump-qemu-154-2014_03_17-22_49_47.vma.lzo |
| 158 | VM 158 | FAILED | 00:00:00 | unable to find VM '158' | |
| 2001 | ooc-FW01 | OK | 00:06:10 | 127MB | /mnt/pve/nfs-backup-02/dump/vzdump-qemu-2001-2014_03_17-22_51_42.vma.lzo |
| 2002 | ooc-FW02 | OK | 00:06:20 | 137MB | /mnt/pve/nfs-backup-02/dump/vzdump-qemu-2002-2014_03_17-22_57_52.vma.lzo |
| 2102 | ooc-Miles | OK | 00:33:14 | 15.15GB | /mnt/pve/nfs-backup-02/dump/vzdump-qemu-2102-2014_03_17-23_04_13.vma.lzo |
| 2121 | ooc-Church | OK | 00:26:17 | 13.61GB | /mnt/pve/nfs-backup-02/dump/vzdump-qemu-2121-2014_03_17-23_37_27.vma.lzo |
| TOTAL | | | 01:18:43 | 39.33GB | |

## Compression

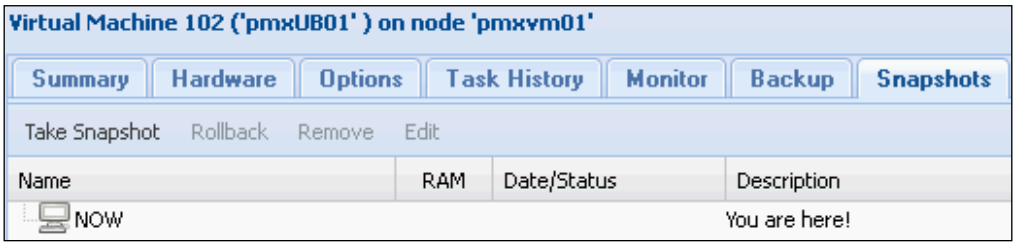
By default, the LZO compression method is selected. LZO (`http://en.wikipedia.org/wiki/Lempel-Ziv-Oberhumer`) is based on a lossless data compression algorithm, designed with the decompression ratio in mind. LZO is capable to do fast compression and even faster decompressions. GZIP will create smaller backup files at the cost of high CPU usage to achieve a higher compression ratio. Since higher compression ratio is the main focal point, it is a slow backup process. Do not select the **None** compression option, since it will create large backups without compression. With the **None** method, a 200 GB RAW disk image with 50 GB used will have a 200 GB backup image. With compression turned on, the backup image size will be around 70-80 GB.

## Mode

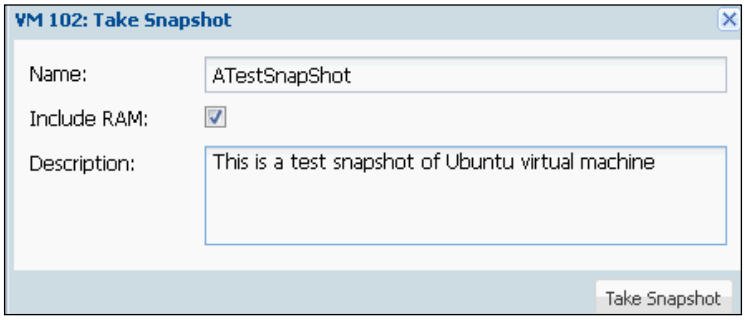
Typically, all running virtual machine backups occur with the **Snapshot** option. Do not confuse this **Snapshot** option with Live Snapshots of VM. The Snapshot mode allows live backup while the virtual machine is turned on, while Live Snapshots captures the state of the virtual machine for a certain point in time. With the **Suspend** or **Stop** mode, the backup task will try to suspend the running virtual machine or forcefully stop it prior to commencing full backup. After backup is done, Proxmox will resume or power up the VM. Since **Suspend** only freezes the VM during backup, it has less downtime than the **Stop** mode because VM does not need to go through the entire reboot cycle. Both the **Suspend** and **Stop** modes backup can be used for VM, which can have partial or full downtime without disrupting regular infrastructure operation, while the Snapshot mode is used for VMs that can have a significant impact due to their downtime.

# Creating snapshots

Snapshots are a great way to preserve the state of a virtual machine. It is much faster compared to full backup since it does not copy the data. Snapshot is not really a backup in a way and does not perform granular level backup. It captures the state in a point in time and allows rollback to that previous state. Snapshot is a great feature to be used in between full backups. The Proxmox snapshot even allows to capture memory of the virtual machine, so when rolled back, it is almost as if it never changed. The following screenshot is our virtual machine pmxUB01 without any snapshot:

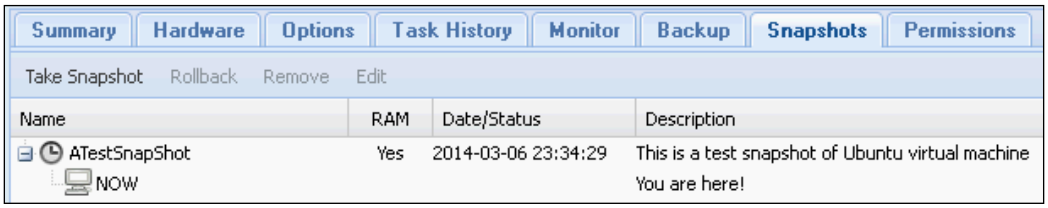


The actual snapshot creation process is very straightforward. Just enter a name, select the RAM content, and type in some description. The **Name** textbox does not allow any spaces and the name must start with an alphabet as shown in the following screenshot:




Keep in mind that when you select to include RAM in the snapshot, the bigger the RAM allocation is for the virtual machine, the longer it will take to create a snapshot. But it is still much faster than full backup. The Snapshot feature is only available for KVM virtual machines and not for OpenVZ containers.

Now, we have our very first snapshot of a virtual machine. If we want to go back to the snapshot image, just select the snapshot we want to go back to and click on **Rollback**. The following screenshot shows the **Snapshots** tab:



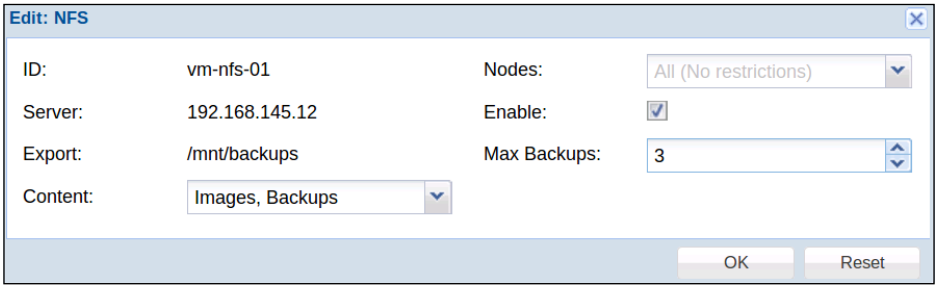

**Rollback** will erase all the changes that happened to the virtual machine between the time of rolling back and the snapshot being rolled back to. When full backup occurs on a virtual machine with snapshots, Proxmox only backs up the main configuration file and disk image file. It does not include any snapshot images.

Proxmox Snapshot does not offer any scheduling option. All snapshots are taken through a manual process. A lot of feature requests have been made to make snapshot scheduling available. Hopefully, in the future versions this feature will be included. In an environment with several dozen virtual machines, manual snapshots can become a time-consuming task. It is possible to set up snapshot scheduling by using bash, cron, and qm, but it is known to be somewhat unstable and, therefore, not recommended for production environment.

Snapshots are also a great way to test new software or configuration on a virtual machine. Take a snapshot before installing a software or applying new configuration. After the software test or if the configuration does not work as intended, simply roll back to the previous snapshot. This is much faster and cleaner instead of uninstalling the tested software itself.

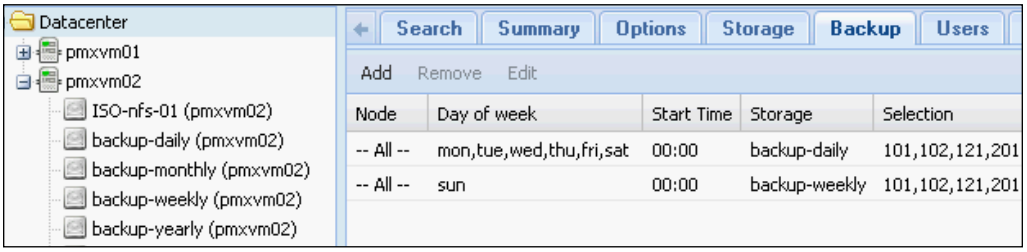
## Deleting old backups

Depending on the backup strategy and business requirement, there may be a need to keep certain periods of backup history. Proxmox allows both automatic and manual deletion of any backups outside the required history range. The following screenshot shows the storage edit dialog box where we can set the maximum number of backups we want to keep. This maximum backup value is for each virtual machine backup:

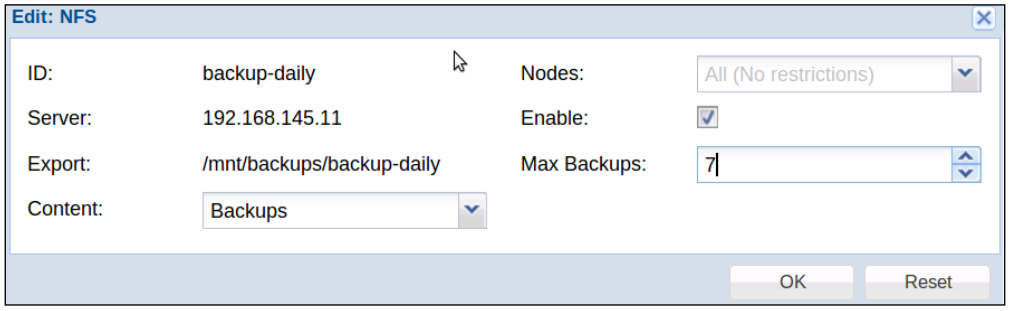



We can enter any numeric number between **0** and **365** as **Max Backups**. For example, our NFS storage has a **Max Backups** value of **3**. This means that during full backup, Proxmox will keep three newest backups of each virtual machine and delete anything beyond that. If we did daily backup, we could potentially keep 365 days or 1 year worth of backups at any given time. If we did backup every other day, then it would be two years worth of backup. It is possible to create separate shared storages daily, weekly, monthly, and yearly in order to store backups. As long as we have the schedule created just right, Proxmox will do the rest.

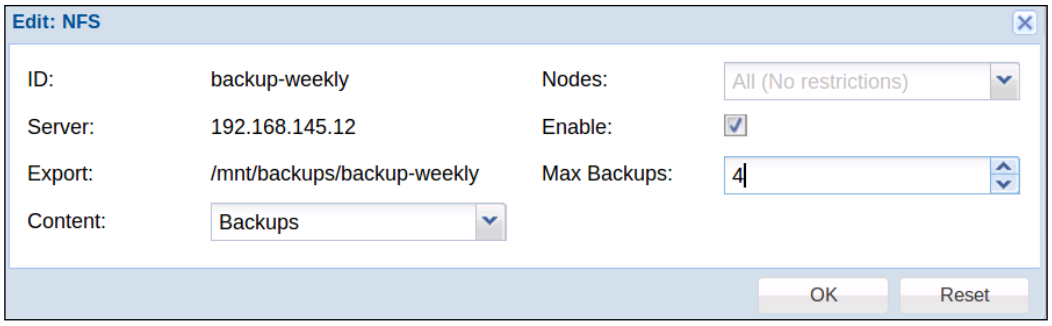
For higher backup redundancy and assurance, we can attach multiple shared storages to store backups. In the example shown in the following screenshot, there are four separate shared storages attached for daily, weekly, monthly, and yearly backup. There are also two backup tasks scheduled: one for daily backup, which keeps six days of backups, and the second task for weekly backup, which keeps four backups The following screenshot shows the **Backup** tab:




Since Proxmox does not have any option to backup monthly or yearly, the only option we have is to modify the second backup task in the previous screenshot and change the destination storage appropriately based on monthly or yearly backup needs. The reason to separate the storages from weekly or monthly backup is so that wrong backups do not get deleted automatically. If we store all backups under one share and set the **Max Backups** value of that storage, then regardless of when the backups were made older backups beyond the **Max Backups** range will get deleted. The following screenshot shows the configuration for daily backup on host `192.168.145.11` set up to be daily backup node.



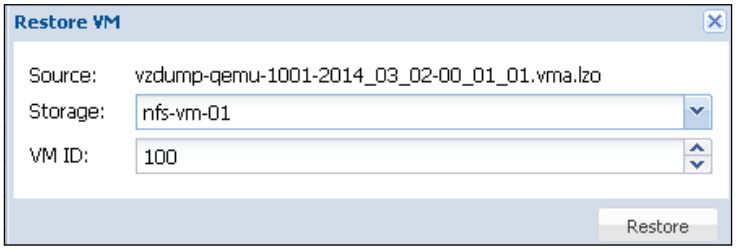

The following screenshot shows the configuration for weekly backup on host `192.168.145.12` setup to be weekly backup node:




Having multiple physical hosts to store backups separately provides a very high level of redundancy and assurance. This way not all backups are kept in one backup node. The monthly or yearly backup node can actually be taken offsite when not in use and brought back once a month or once a year for monthly or yearly backup. Although Proxmox cannot perform granular file backup inside VM or detailed scheduling options, it is still a powerful and simple enough feature to support all sizes of Proxmox clusters.

## Restoring a virtual machine

To keep up with the simplicity theme, the Proxmox Restore option also features a very simple interface. We just need to select which virtual machine we want to restore, the destination storage, and virtual machine ID, as shown in the following screenshot:




If the same VM ID is kept, then the existing virtual machine with the same ID will be deleted and restored from the backup version. One important thing to remember is a full backup created for a virtual machine with the `qcow2` or `vmdk` image format can only be restored to local, Ceph FS, or NFS-like storages. But a virtual machine with the `RAW` image format can be restored on just about any storage system. RBD or LVM storages do not support image types such as `qcow2` or `vmdk`. There is no restoration for snapshot images. The snapshots can only be rolled back to the previous state.

# Command-line vzdump

The entire backup process can be handled from the command line in case the GUI becomes inaccessible. The command to start a backup is as follows:

```
# vzdump <vmid> <options>
```

There is a long list of the `vzdump` options that can be used with the command. The following are just a few of the most commonly used ones:

| Options | Description |
|---|---|
| `-all` | Default is 0. This option will back up all the available virtual machines in a Proxmox node. |
| `-bwlimit` | Adjust the backup bandwidth in KBPS. |
| `-compress` | Default is LZO. Sets the compression type or disables compression. The available options are 0, 1, gzip, and lzo. |
| `-mailto` | E-mail address to send backup report. |
| `-maxfiles` | Integer number. Sets maximum number of backup files to be kept. |
| `-mode` | Default is Stop. Sets backup mode. Available options are snapshot, stop, and suspend. |
| `-remove` | Default is 1. Removes older backups if more than the value entered in –maxfiles. |

There are two commands available to restore the KVM and OpenVZ virtual machines. They are as follows:

- For KVM machines, the command is as follows:

  ```
  #qmrestore <backup_file> <vmid> <options>
  ```

| Options | Description |
|---|---|
| `-force` | 0 or 1. This option allows overwriting the existing VM. Use this option with caution. |
| `-unique` | 0 or 1. Assigns a unique random Ethernet address to the virtual network interface. |

- For OpenVZ machines, the command is as follows:

  ```
  #vzrestore <backup_file> <vmid> <options>
  ```

| Options | Description |
|---|---|
| `-force` | 0 or 1. This option allows overwriting the existing VM. Use this option with caution. |
| `-unique` | 0 or 1. Assigns a unique random Ethernet address to the virtual network interface. |

For a complete list of options for `vzdump`, `qmrestore`, and `vzrestore`, visit the following pages:

- `http://pve.proxmox.com/wiki/Vzdump_manual`
- `http://pve.proxmox.com/wiki/Qmrestore_manual`
- `http://pve.proxmox.com/wiki/Vzrestore_manual`

# Backup configuration file – vzdump.conf

The `vzdump.conf` file in Proxmox allows advanced backup configuration to go beyond just the default. For example, if we want to limit the backup speed so that the backup task does not consume all of the available network bandwidth, we can limit it with the `#bwlimit` option. In Proxmox Version 3.2 and above, the `vzdump.conf` file cannot be edited from GUI. It has to be done from CLI using an editor. The following code is the default `vzdump.conf` file on a new Proxmox cluster:

```
# vzdump default settings
#tmpdir: DIR
#dumpdir: DIR
#storage: STORAGE_ID
#mode: snapshot|suspend|stop
#bwlimit: KBPS
#ionice: PRI
#lockwait: MINUTES
#stopwait: MINUTES
#size: MB
#maxfiles: N
#script: FILENAME
#exclude-path: PATHLIST
```

All the options are commented by default in the file because Proxmox has a set of default options already encoded in the operating system. Changing the `vzdump.conf` file overwrites the default settings and allows us to customize the Proxmox backup.

## #bwlimit

The most common option to edit in `vzdump.conf` is to adjust the backup speed. This is usually done in case of remotely stored backups and interface saturation if the backup interface is the same used for the VM production traffic. For example, to limit backup to 200 Mbps, make the following adjustment:

```
bwlimit: 200000
```

# #lockwait

The Proxmox backup uses a global lock file to prevent multiple instances running simultaneously. More instances put extra load on the server. The default lock wait in Proxmox is 180 minutes. Depending on different virtual environments and number of virtual machines, the lock wait time may need to be increased. If the limit needs to be 10 hours or 600 minutes, adjust the option as follows:

```
lockwait: 600
```

The lock prevents the VM from migrating or shutting down while the backup task is running. Any backup interruptions such as failback storage, I/O bottleneck, and so on can cause the VM to remain locked. In such cases, the VM needs to be unlocked with the following command from CLI:

```
# qm unlock <vmid>
```

# #stopwait

The `#stopwait` value is the maximum time in minutes the backup will wait till a VM is stopped. A use case scenario is a VM, which takes much longer to shut down, for example, exchange server or database server. If V M is not stopped within the allocated time, backup is skipped for that VM.

# #script

It is possible to create backup scripts and hook them up with a backup task. This script basically is a set instruction that can be called upon during entire backup tasks to accomplish various backup-related tasks such as start/stop a backup, shut down/suspend VM, and so on. We can add customized scripts as:

```
script: /etc/pve/script/my-script.pl
```

Here is a sample publicly available hook script for a backup task. The following script is courtesy Dietmar Maurer from the Proxmox staff. The script is shown here to point you to the right direction. Full customization details of the script language are beyond the scope of this book:

```
#!/usr/bin/perl -w

# example hook script for vzdump (--script option)

use strict;

print "HOOK: " . join (' ', @ARGV) . "\n";

my $phase = shift;
```

```
if ($phase eq 'job-start' ||
    $phase eq 'job-end'  ||
    $phase eq 'job-abort') {

    my $dumpdir = $ENV{DUMPDIR};

    my $storeid = $ENV{STOREID};

    print "HOOK-ENV: dumpdir=$dumpdir;storeid=$storeid\n";

    # do what you want

} elsif ($phase eq 'backup-start' ||
    $phase eq 'backup-end' ||
    $phase eq 'backup-abort' ||
    $phase eq 'log-end' ||
    $phase eq 'pre-stop' ||
    $phase eq 'pre-restart') {

    my $mode = shift; # stop/suspend/snapshot

    my $vmid = shift;

    my $vmtype = $ENV{VMTYPE}; # openvz/qemu

    my $dumpdir = $ENV{DUMPDIR};

    my $storeid = $ENV{STOREID};

    my $hostname = $ENV{HOSTNAME};

    # tarfile is only available in phase 'backup-end'
    my $tarfile = $ENV{TARFILE};

    # logfile is only available in phase 'log-end'
    my $logfile = $ENV{LOGFILE};

    print "HOOK-ENV:
      vmtype=$vmtype;dumpdir=$dumpdir;storeid=$storeid;
        hostname=$hostname;tarfile=$tarfile;logfile=$logfile\n";

    # example: copy resulting backup file to another host
      using scp
    if ($phase eq 'backup-end') {
        #system ("scp $tarfile backup-host:/backup-dir") == 0 ||
        #    die "copy tar file to backup-host failed";
```

```
        }

        # example: copy resulting log file to another host using scp
        if ($phase eq 'log-end') {
            #system ("scp $logfile backup-host:/backup-dir") == 0 ||
            #    die "copy log file to backup-host failed";
        }

    } else {

        die "got unknown phase '$phase'";
    }

    exit (0);
```

**Downloading the example code**

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

# #exclude-path

To ignore certain folders from backing up, use the `exclude-path` option. All paths must be entered on one line without breaks. Please keep in mind that this option is only for OpenVZ containers:

```
exclude-path: "/log/.+" "/var/cache/.+"
```

The previous example will exclude all the files and directories under `/log` and `/var/cache`. To manually exclude other directories from being backed up, simply use the following format:

```
exclude-path: "/<directory_tree>/.+" 3
```

# Summary

In this chapter, we have covered some interesting topics, such as nested virtual environment, the hotplug feature, and Proxmox backup/restore. A good backup plan can save the day many times over. Although Proxmox does not provide everything you need for backup such as a granular file backup, to back up a virtual machine is very helpful. Backup features in the Proxmox platform have proven to be reliable in production environments and during actual disaster scenarios.

In the next chapter, we are going to look at an even more interesting topic: network virtualization. Although it is a fairly new concept, it is by no means immature. In an ever-growing virtual environment, a good virtualized network design can alleviate management issues without adding any overhead. We will see how we can take complete advantage of network virtualization in Proxmox to bind volumes of virtual machines together and work in harmony.

# 5
# Network of Virtual Networks

In this chapter, we are going to take an in-depth look at how we can create a virtualized network within a virtual environment. We will learn what the network building blocks are that make up the Proxmox hypervisor and how it manages both internal and external network connectivity. We will examine several network diagrams to see how Proxmox can be pushed further to create an entire colony of virtual machines connected with virtual networks. We will also look at the Proxmox network configuration file, network bonding, VLAN, and so on. We can create dozens of virtual machines at will, but without a planned network model, we will fail to run a good virtual environment. If we compare virtual machines with bricks as building blocks, then it is the network that acts as mortar to create anything from a basic hut to a cathedral.

In this chapter, we will cover the following topics:

- Definition of a virtual network
- Networking components of Proxmox, such as bridge, vNIC, VLAN, bonding, and so on
- Proxmox network configuration file
- Adding network components in a VM
- Sample virtual networks
- Multitenant virtual environments

# Introduction to a virtual network

A virtual network is a software-defined network where all links and components may or may not have direct interaction with physical hardware. In most cases, direct interaction with physical hardware is made by the hypervisor or the host controller. All links between virtual machines, virtual switches, virtual bridges, and virtual network interfaces are made completely virtually. The following are the two types of network virtualization:

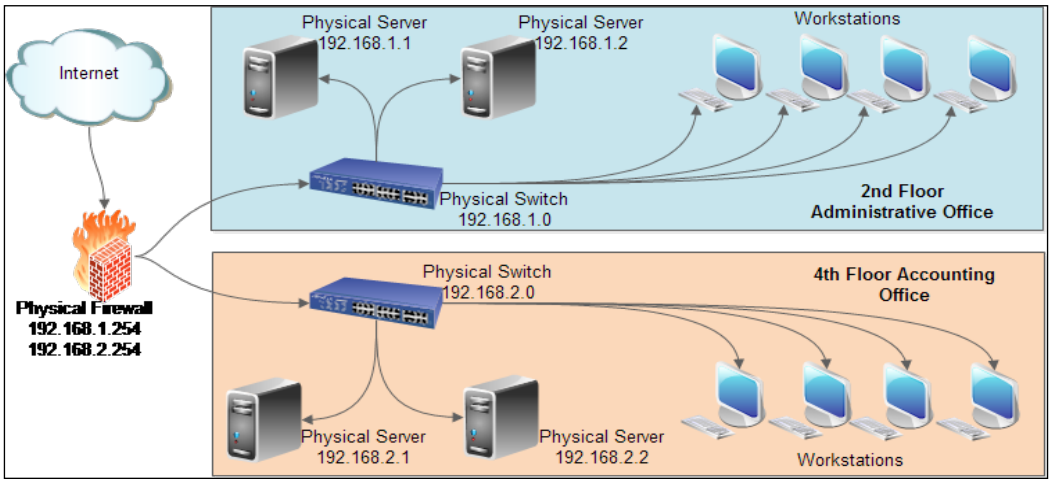
- **External network virtualization**: This consists of several local networks operating as one virtual network. Physical LANs could be in the same location or spread over multiple locations. Usually, external virtualization is a cloud network service-based model that multiple companies can use to connect their multisite virtual environment for a service fee. External network virtualization can be easily achieved by combining several internal virtual networks into a single virtualized network using a WAN or the Internet.
- **Internal network virtualization**: This usually happens locally within a hypervisor between virtual machines. Do not confuse this with the local area network. Here, internal network virtualization is referring to the network connectivity between VMs, bridges, vNICs, and so on, which do not necessarily have to utilize external LAN. This provides company IT staff total control over virtual network operation. Network issues can be diagnosed faster; customization of expansion or contraction can happen without delay. Internal virtualization heavily uses virtual components, such as virtual bridges and vNIC, to form a virtual network.

For in-depth information on external and internal network virtualizations, have a look at the link `http://en.wikipedia.org/wiki/Network_virtualization`. Specially follow the *References* and *Further reading* book list at the bottom of the Wiki page.

In this book, we will mainly look at internal network virtualization in the Proxmox hypervisor. We will see some network diagrams of internal and external virtual network combinations later in the book in *Chapter 10*, *Putting It All Together*.

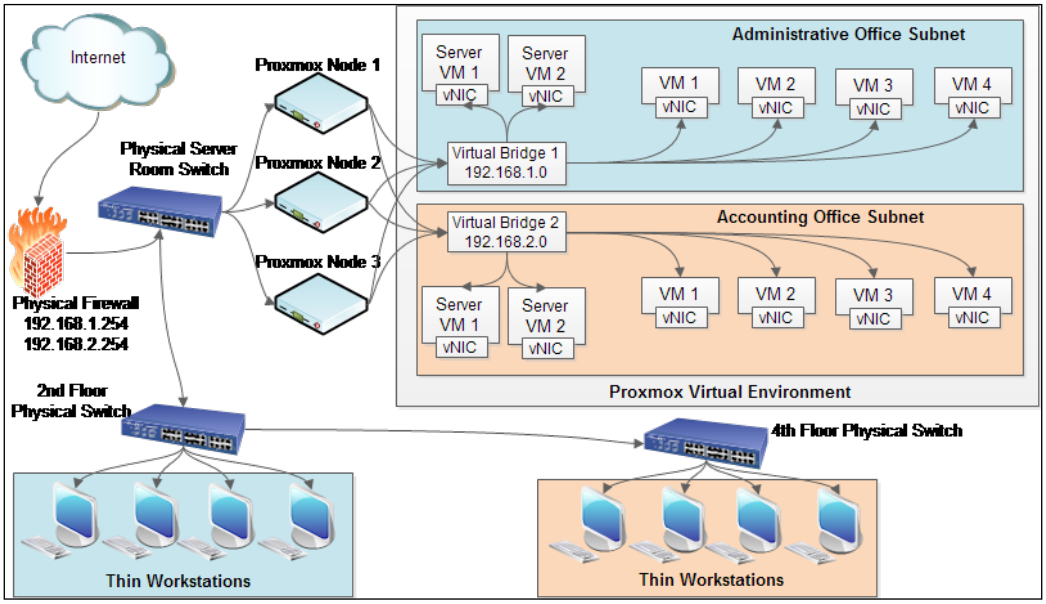
# Physical network versus virtual network

Let us look at the following diagrams to see the difference between a physical network and a virtual network. The following diagram represents a physical network without any virtualization platform:




Physical network diagram

The following diagram represents virtualization as main infrastructure:




Virtual network diagram

Do not worry if the virtualization diagram is overwhelming at first glance. We will learn about all the components used in both diagrams later on. Both diagrams are in basic form. Some non-relevant components may have been omitted to keep the diagrams simpler.

Before we dive into virtual network building blocks, we need to understand how networks are set up in the preceding diagrams. Both the diagrams represent the same office setup where the main administrative department is on the second floor, and the accounting department is on the fourth floor of the building. It is apparent from the diagrams that a physical network is undoubtedly simpler than a virtual network, but by leveraging virtualization, we can cut cost, increase efficiency, reduce hardware maintenance complexity, and increase portability.

# Physical network

In the first diagram of the physical network, there is no virtualization platform set up. The whole network is set up with physical devices, such as firewalls, switches, servers, and full desktops. Each department has its own servers and separate networks. A centralized management for the whole company does not exist. This is a costly solution due to all the physical hardware. If redundancy is a requirement, it will incur twice the cost since we will need identical physical servers. All connectivity in this network is done with physical cable links. Backups in this setup are quite challenging since all the physical servers in two departments have to be backed up individually.

# Virtual network

The virtual network diagram represents how Proxmox can handle a multidepartment setup. All connections between servers and the user's virtual machines happen virtually without a physical network device. Using virtual bridges and vNICs, both the administrative and accounting departments can coexist on the same Proxmox cluster. Since all computing happens in the hypervisor, end users can have thin workstations to minimize cost significantly. Users connect to their virtual machines with remote protocols, such as SPICE, VNC, or RDP.

Thin workstations are very underpowered, cheap, basic computers for the end user. Since all processing happens in a virtual environment, thin workstations do not need to be very powerful. The main purpose of a thin workstation is to allow the user to connect peripherals such as the monitor, keyboard, mouse, and network cable. A thin workstation can be purchased under $200.

In this setup, all servers and user machines are virtualized. If there is a need for a new server, it is just a matter of creating a virtual server with vNIC with a few clicks. Since Proxmox as a hypervisor has redundancy by design, a dead physical server node is no longer an issue. In such a scenario, all virtual machines can simply be migrated to another available Proxmox node, and everything is up and running in minutes. Both the departments are separated by two virtual bridges.

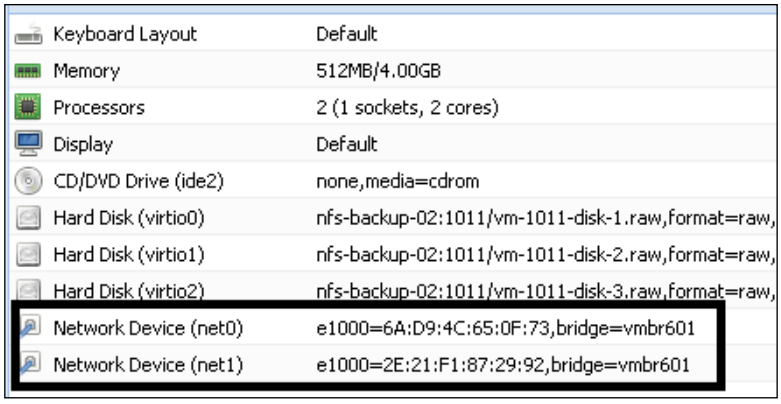
Through the use of the Proxmox GUI, all management can be done from one location, including backup and restore. Virtual servers can be sent to another remote company location through WAN. Although a virtual network setup is much more robust and feature-rich, it has a much lower budget requirement. New departments can be added by creating new virtual bridges for separate subnets and using VLANS on existing physical network switches.

# Networking components in Proxmox

We will now look at the components used in Proxmox that make the virtual network possible within the Proxmox virtual environment.

# Virtual Network Interface Card (vNIC)

**Virtual Network Interface Card** (**vNIC**) is a software-defined representation of a **Media Access Control** (**MAC**) interface of physical network interfaces. It is basically a virtual network card for a virtual machine. Multiple vNICs can share a physical network interface of a host node. In a way, networking starts with vNIC when a virtual machine sends data to reach other virtual machines or networking devices within a virtual environment or physical environment. In the following diagram, the virtual machine has two virtual network interfaces assigned with an Intel e1000 driver. Both of them are configured with bridge **vmbr601**.



Intel e1000 is a Linux kernel driver to virtualize Intel architecture-based virtual network interfaces. This is the default vNIC for new virtual machines in Proxmox since it is supported by all major operating systems, such as Linux, Windows, and Mac OS X, without needing additional drivers. Proxmox has four models of virtual network interfaces: Intel e1000, VirtIO, Realtek RTL8139, and VMware vmxnet3. It should be noted that out of these four models, VirtIO provides the maximum network performance for a VM. All Linux-based operating systems come equipped with VirtIO drivers. For Windows, the VirtIO interface driver can be downloaded from `http://www.linux-kvm.org/page/WindowsGuestDrivers/Download_Drivers`. For Mac OS, the VirtIO interface driver can be downloaded from `https://github.com/pmj/virtio-net-osx`.

# Virtual bridge

Just as a real-world bridge connects the two sides of a river, a virtual bridge connects a Proxmox virtual network with a physical network. A virtual bridge is like a physical network switch where all virtual machines connect to and can be configured using the **Spanning Tree Protocol** (**STP**). A virtual bridge is a great way to create separate subnets. All VMs in the same subnet can connect to their respective bridges. Each Proxmox node can support up to 4,094 bridges. When the same bridge configuration is entered on all nodes, the bridge can be used from any nodes in the cluster, thus making live migration possible without network connectivity interruption. The common naming format of a bridge is **vmbrX**, where X represents an integer between 0 and 4,094.

# Virtual LAN (VLAN)

**Virtual Local Area Network** (**VLAN**) is a logical local area network within a physical local area network. It can be compared with partitions within a physical disk storage. A physical network interface can be partitioned to transport data for multiple separate subnets. This partition is achieved using VLAN ID. For details on VLAN or IEEE 802.1q standard, refer to `http://en.wikipedia.org/wiki/IEEE_802.1Q`.

Once VLAN data leaves the virtual environment, a physical network switch with the VLAN feature tags each data with an ID and then directs the data to its proper destination. Each subnet should have the same VLAN ID on the virtual environment and on the physical network switch. VLAN helps reduce the broadcast traffic of multiple domains on the same network. By segmenting a large network into smaller VLAN, broadcasts can be sent only to relevant VLAN without interrupting other data traffic on the network.

Imagine a 10-lane highway connecting point A to point B. Each lane is designated to one manufacturing plant that needs to transport goods from point A to point B. Plant 1 is designated to lane number 1 (VLAN ID 1); plant 2 is designated to lane number 2 (VLAN ID 2); and so on. Even though all plants share the same highway, they each have their own lane not shared by others. VLAN follows the same logic.

VLAN also provides an added security layer on a multidomain network since a user can no longer just plug into the network and capture just about any data of any domains on the network. Network segmentation is usually done with a layer 3 device such as a router. But using VLAN, significant cost saving can be achieved with an already-existing layer 2 device on the network such as a switch. There are seven layers defined by the **Open Systems Interconnection** (**OSI**) model by which network communication takes place. For in-depth details on OSI, follow the link `http://en.wikipedia.org/wiki/OSI_model`.

# Network Address Translation/Translator (NAT)

**Network Address Translation/Translator** (**NAT**) is the translation of an IP address between known and/or unknown networks. NAT allows an outbound internal network to have a designated public IP address on the Internet. So when a network device initiates an outbound connection, it uses the designated IP address instead of the local IP address designated to the device. NAT reduces the exposure of the internal IP address to the public network, thus providing protection from internal IP exposure. NAT is usually configured in the router or firewall of a network, where the policy is created for local-to-global and global-to-local IP address mapping. It should be noted that NAT is relevant for IPv4 networks. An IPv6 network diminishes the need to use NAT, because IPv6 addressing is always public.

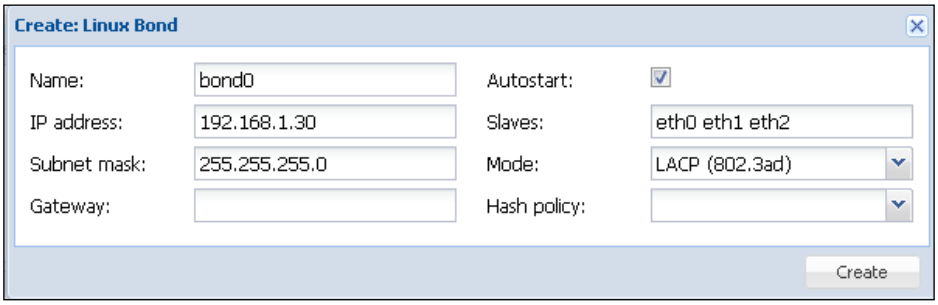
# Network bonding

**Network bonding** or **Teaming** or **Link aggregation** is a concept where multiple interfaces are combined to increase throughput, set up network redundancy, and balance network load. This concept is heavily used in high-demand environments where downtime and slow network I/O are not acceptable. The Proxmox GUI provides the excellent feature of creating and managing bonding within the cluster node. Bonding modes supported by Proxmox are balance-rr, active-backup, balance -xor, broadcast, **Link Aggregation Control Protocol** (**LACP**), or 802.3ad, balance-tlb, and balance-alb. The following table lists the various bonding modes as well their policies and descriptions:

| Bonding Mode | Policy | Description |
|---|---|---|
| balance-rr<br>or<br>Mode 0 | Round robin | Packet transmission takes places sequentially from the first participating network interface to the last. This provides load balancing and fault tolerance |
| active-backup<br>or<br>Mode 1 | Active backup | Only one participating network interface is active. The next interface becomes active when the previous active interface fails. This only provides fault tolerance. |
| balance-xor<br>or<br>Mode 2 | XOR | This mode selects the same participating interface for each destination MAC address. Transmission takes place based on bonded network interfaces of MAC address XOR'd with the destination MAC address. This provides both load balancing and fault tolerance. |
| broadcast<br>or<br>Mode 3 | Broadcast | Transmission takes place on all participating bonded network interfaces. Provides fault tolerance only. |
| 802.3ad<br>or<br>Mode 4 | Dynamic link aggregation | All participating network interfaces in the aggregated group share the same speed and duplex settings.<br>All interfaces are utilized according to the 802.3ad specification. A network switch with 802.3ad or LACP feature is required. This provides fault tolerance. |

| Bonding Mode | Policy | Description |
|---|---|---|
| balance-tlb<br>or<br>Mode 5 | Adaptive transmit load balancing | Outgoing packets are distributed according to current load on each participated interface. Incoming packets are received on current interface, and if the same interfaces fails, then the next available interface takes over. This provides fault tolerance and load balancing for only outbound packets. |
| balance-alb<br>or<br>Mode 6 | Adaptive load balancing | Same as balance-tlb with the inclusion of load balancing for incoming packets on all interfaces. This provides fault tolerance and load balancing for both incoming and outgoing traffic. |

The following screenshot shows the Proxmox menu system for creating new bonding:



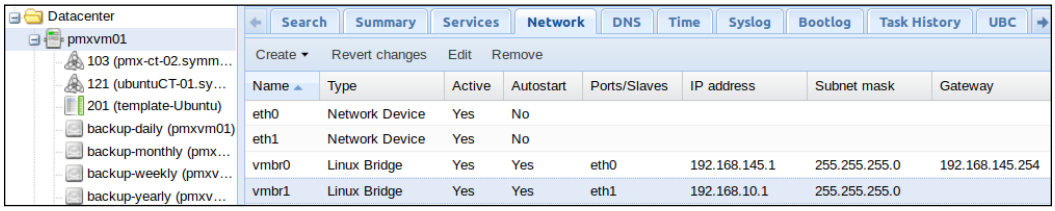

# Components naming convention

The following table shows different components used in Proxmox and their naming conventions. A proper understanding of how the components are named helps to pinpoint an interface much faster. It also helps to organize or map a network much more efficiently.

| Interface | Naming convention | Example |
|---|---|---|
| Physical Ethernet devices | ethX; X = 0 to 99 | eth0, eth1, eth20, and so on |
| Virtual bridges | vmbrX; X = 0 to 4094 | vmbr0, vmbr1, vmbr2000, and so on |
| Network bonds | bondX; X = 0 to 9 | bond0, bond1, bond9, and so on |
| VLAN | ethX.<VLAN ID>; X = 0 to 4095 | VLAN with ID 3 = eth0.3<br>VLAN with ID 10 = eth0.10 |

# Network configuration file

We are now going to a look at how a Proxmox network configuration file works and how we can modify it to adapt to different networking environments. This configuration file is stored at `/etc/network/interfaces`. Network configuration can be modified both from the GUI and the CLI. Any changes made from the GUI are saved temporarily in `/etc/network/interfaces.new`. The changes get transferred to `/etc/network/interface` only after reboot. You can also directly edit the `interfaces` file at `/etc/network/` and activate or deactivate components with `#ifup <name>` or `#ifdown <name>` without needing to reboot. All virtual bridges must be configured on all the Proxmox nodes in the cluster. If there is a lot of bridge configurations in the interface file, simply copy them to all nodes and modify only the physical interfaces on each node. The following screenshot is the GUI representation of the configuration file for node **pmxvm01**:




The following code is how the configuration file in `/etc/network/interfaces` looks like from the command line:

```
auto lo
iface lo inet loopback

auto vmbr0
iface vmbr0 inet static
        address 192.168.145.1
        netmask 255.255.255.0
        gateway 192.168.145.254
        bridge_ports eth0
        bridge_stp off
        bridge_fd 0

auto vmbr1
iface vmbr1 inet static
        address 192.168.10.1
        netmask 255.255.255.0
        bridge_ports eth1
        bridge_stp off
        bridge_fd 0
```

Based on this configuration, the **pmxvm01** node has two virtual bridges set up. Bridge vmbr0 is tied to physical network interface eth0, and bridge vmbr1 is tied to eth1 of the node. If we want to create a fully isolated subnet without any outgoing traffic, we could edit bridge vmbr1 to look like the following code:

```
auto vmbr1
iface vmbr1 inet manual
        bridge_ports none
        bridge_stp off
        bridge_fd 0
```

# bridge_stp

The `bridge_stp` option allows multiple bridges to communicate with each other for network discovery and loop avoidance. This is useful to eliminate data cycles to provide optimal packet routing, because with STP on, bridges can talk to each other and figure out how they are connected and then provide best routing possible for data packet transmission. STP also allows fault tolerance since it will check the network topology if a bridge fails. To turn on the STP option, just modify the bridge configuration as follows:

```
bridge_stp on
```

STP increases bandwidth efficiency while posing security issues. Do not use STP when a virtual subnet requires isolation from other virtual subnets in the same cluster, and you do not want the bridges to talk to each other. It is a useful option when working in a virtual environment of the same company where data can flow freely between department subnets. STP does not have any authentication and assumes all network interfaces to be trustworthy. When a hostile bridge enquires about the network topology from another bridge, information is freely shared without any authentication. Thus, a user in the hostile bridge could potentially gather data of the entire network topology and other bridges in the network. This leads to a dangerous situation with bridging between the internal environment and the Internet. STP is turned off by default.

# bridge_fd

**FD** refers to **Forwarding Delay**. The `bridge_fd` option sets the delay of how long before the interface will be ready. During the delay, bridge tries to discover other bridges and checks there are no network loops if STP is on. By default, the forwarding delay is set to 0 as shown in the following line of code:

```
bridge_fd 0
```

For the most part, the default value of 0 is enough. In a very complex virtual environment with several dozen bridges, increasing this number to 3 or 4 might help. Without this delay, the bridge will start transmitting data packet regardless of whether the other destination bridge is available or not. Increasing the delay time allows the source bridge to check all bridges and not transmit any data if the destination bridge is down, thus preventing unnecessary network bandwidth consumption.

There are many more `bridge_` options to be used in the network configuration file, such as `bridge_hello`, `bridge_maxage`, `bridge_bridgeprio`, and so on. Bridge options are Linux-specific and beyond the topic of this book. For in-depth information on bridges, visit `http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge`.

# Adding a virtual bridge

We have seen the networking components used in Proxmox; we went through the configuration file and saw how those components come together. We are now going to create a new virtual bridge for our cluster. We will use this bridge to create an isolated network later on in this book. Perform the following steps to create a virtual bridge in Proxmox:

1. Securely log in to the node **pmxvm01**.
2. Open the interface file `# nano /etc/network/interfaces` using an editor.
3. Add the following lines at the end of the file:

```
auto vmbr200
iface vmbr1 inet static
        bridge_ports none
        bridge_stp off
        bridge_fd 0
```

4. Save the file and exit the editor
5. Activate the bridge from the CLI using the following command:

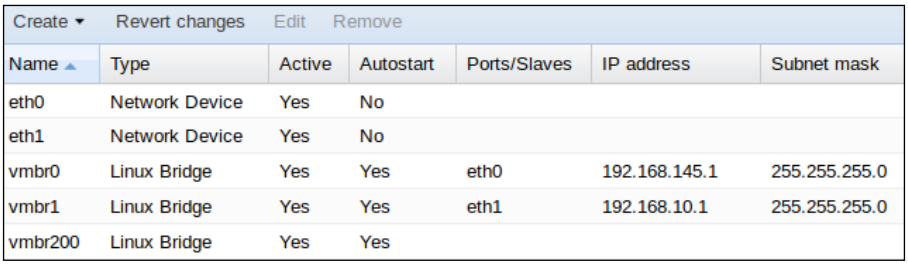
```
# ifup vmbr200
```

6. Do the same for the second node **pmxvm02**.

The new virtual bridge vmbr200 should now be activated and running. If there are more cluster nodes, then the steps from 1 to 5 must be completed on all the nodes. The `/etc/network/interfaces` configuration file should look as follows:

```
auto lo
iface lo inet loopback

auto vmbr0
iface vmbr0 inet static
        address 192.168.145.1
        netmask 255.255.255.0
        gateway 192.168.145.254
        bridge_ports eth0
        bridge_stp off
        bridge_fd 0

auto vmbr1
iface vmbr1 inet static
        address 192.168.10.1
        netmask 255.255.255.0
        bridge_ports eth1
        bridge_stp off
        bridge_fd 0

auto vmbr200
iface vmbr1 inet static
        bridge_ports none
        bridge_stp off
        bridge_fd 0
```

We can verify the new bridge from the Proxmox GUI as shown in the following screenshot:

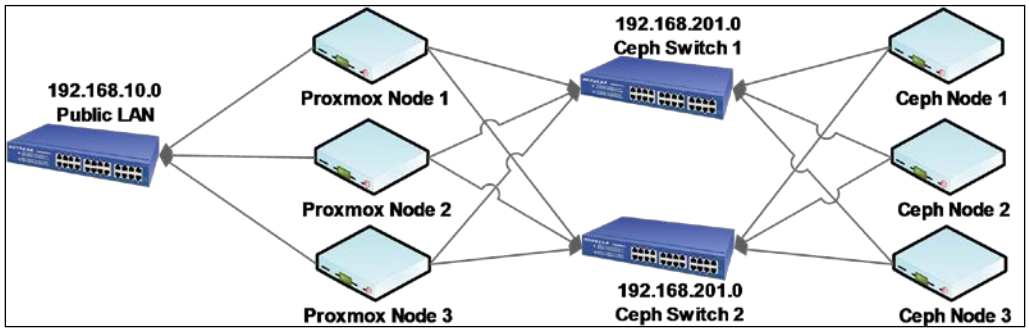



| Name | Type | Active | Autostart | Ports/Slaves | IP address | Subnet mask |
|---|---|---|---|---|---|---|
| eth0 | Network Device | Yes | No | | | |
| eth1 | Network Device | Yes | No | | | |
| vmbr0 | Linux Bridge | Yes | Yes | eth0 | 192.168.145.1 | 255.255.255.0 |
| vmbr1 | Linux Bridge | Yes | Yes | eth1 | 192.168.10.1 | 255.255.255.0 |
| vmbr200 | Linux Bridge | Yes | Yes | | | |

# Adding a bonding interface

We will now see network bonding in our cluster. There are several types of bonding options available. But only balance-rr, active-backup, and LACP (802.3ad) are most widely used. The balance-rr option provides the round robin method to increase overall interface bandwidth with failover. The balance-rr option does not require any special network switch. Just about any switch can be used to make this work. The major drawback of balance-rr is waste of data packets. LACP is known as industry quality bonding.

In this book, we will only look at the LACP bonding protocol. But to give you an idea of how a balance-rr bonding looks like, the following diagram shows balance-rr bonding between Proxmox nodes and Ceph distributed storage clusters. In this example, the Proxmox public network is on 192.168.10.0/24, while the storage backend is on a private 192.180.201.0/24 subnet. Separate switches are used for the Ceph storage network to increase redundancy. Each Proxmox node has three 1-gigabit NICs. One is used from the main cluster to server virtual machines, and the remaining two are used for balance-rr bonding. This type of bonding is a very economical way to provide network redundancy.




LACP can combine multiple interfaces to increase the total throughput but not the actual connection. For example, an LACP bonding of four 1-gigabit network interfaces will still have total connection speed of 1 gigabit, but it will be able to respond to more simultaneous requests at closer to 1 gigabit speed.

To know more about Link aggregation/Bonding/Teaming, visit `http://en.wikipedia.org/wiki/Link_Aggregation_Control_Protocol#Link_Aggregation_Control_Protocol`.

For LACP to work, it is very important to know if the physical switch supports this feature. A quick visit to a switch manufacturer's website will give us the information if the LACP feature is supported. Some manufacturers will list this feature as 802.3ad.

In our cluster, we are now going to set up LACP bonding for the bridge vmbr1. Later in the book, we will set up an enterprise-standard distributed shared storage system using Ceph. We will use the bridge vmbr1 to connect the shared storage with the Proxmox cluster. Let's start by adding a third network interface card on each node if you have not done so. So, both nodes now should have three physical network interface cards on each. Perform the following steps to configure LACP or 802.3ad network bonding:

1. Securely log in to the node **pmxvm01**.
2. Open the interface file `# nano /etc/network/interfaces` using an editor.
3. Add the following lines at the beginning of the file:

```
# bring up interfaces
auto eth1
iface eth1 inet manual
auto eth2
iface eth2 inet manual

# bonding interfaces
auto bond0
iface bond0 inet manual
        slaves eth1 eth2
        bond_miimon 100
        bond_mode 802.3ad
```

4. Change ports for the bridge vmbr1 as follows:

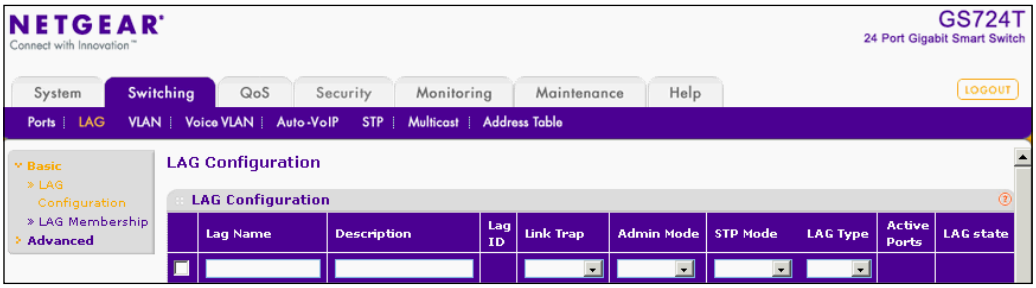
```
auto vmbr1
iface vmbr1 inet static
        address 192.168.10.1
        netmask 255.255.255.0
        # change bridge port to bond0 interface
        bridge_ports bond0
        bridge_stp off
        bridge_fd 0
```

5. Save the file and exit the editor.
6. Activate the bridge by rebooting the node or from CLI by stopping and restarting the bridge. Use the following commands:

```
# ifup bond0
# ifdown vmbr1
# ifup vmbr1
```

7. Do the same for the second node pmxvm02.

8. Set up LACP or link aggregation on a physical switch. The following screenshot is an example of LACP setting on Netgear GS724T 24 port switch:

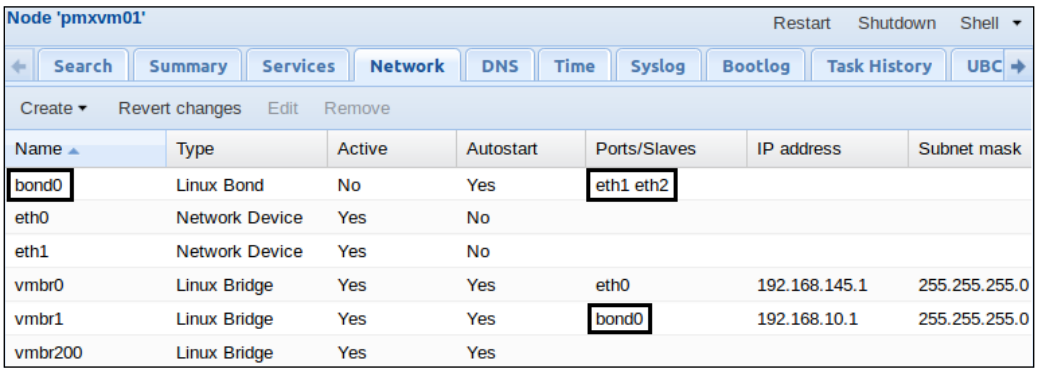



The final interface configuration of our node should be as follows:

```
auto lo
iface lo inet loopback

# bring up interfaces
auto eth1
iface eth1 inet manual
auto eth2
iface eth2 inet manual

# bonding interfaces
auto bond0
iface bond0 inet manual
        slaves eth1 eth2
        bond_miimon 100
        bond_mode 802.3ad


auto vmbr0
iface vmbr0 inet static
        address 192.168.145.1
        netmask 255.255.255.0
        gateway 192.168.145.254
        bridge_ports eth0
        bridge_stp off
        bridge_fd 0

auto vmbr1
iface vmbr1 inet static
        address 192.168.10.1
```

```
        netmask 255.255.255.0
        bridge_ports bond0
        bridge_stp off
        bridge_fd 0

auto vmbr200
iface vmbr1 inet static
        bridge_ports none
        bridge_stp off
        bridge_fd 0
```

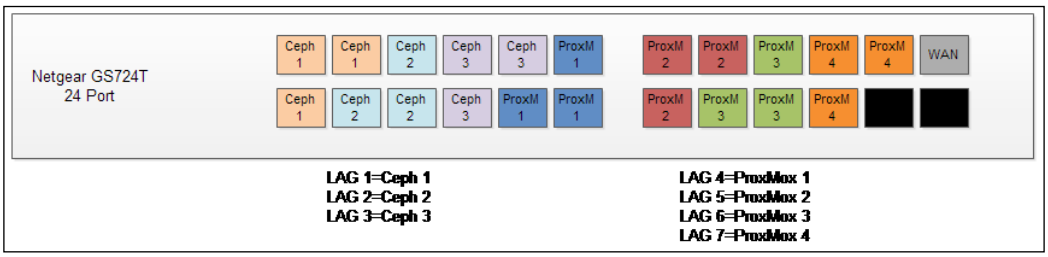
We can also verify it from the Proxmox GUI as shown in the following screenshot:




The following diagram is an example of an advanced cluster with four Proxmox nodes, three Ceph nodes, and a Netgear switch with LACP bonding set up. Four **Link Aggregation Groups** (**LAG**s) have been created.

LAG is a combination of multiple links between switches or nodes to form a single large link.

The following diagram shows a setup of three NICs bonding:



The following several options can be used while configuring bonding:

| | |
|---|---|
| bond_miimon X | X = integer number. It sets the MII link monitoring frequency in milliseconds. This option checks the frequency at which each slave port is inspected for link failures. |
| bond_downdelay X | X = integer number. It sets the time in milliseconds to wait before disabling a slave after a link failure has been detected. |
| bond_updelay X | X = integer number. It sets the time in milliseconds to wait before enabling a slave after a link recovery has been detected. |

# Adding NAT/masquerading

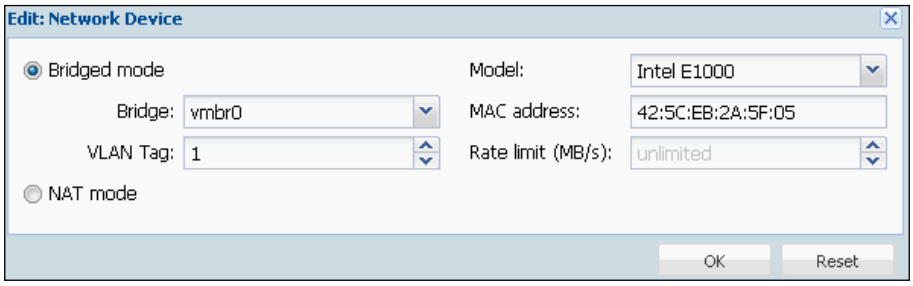
NAT is a way to hide internal network IP addresses from the external network, such as the Internet. Any outgoing traffic uses the main host IP address instead of using own local IP address. Add the last three lines of the following post-up and post-down settings in the configuration file `/etc/network/interfaces`. Only add these lines under the virtual bridge configuration which needs the NAT option. Have a look at the following code snippet:

```
auto vmbr0
iface vmbr0 inet static
address 192.168.145.1
netmask 255.255.255.0
bridge_ports none
bridge_stp off
bridge_fd 0post-up echo 1 > /proc/sys/net/ipv4/ip_forward
post-up iptables -t nat -A POSTROUTING -s '192.168.145.0/24' -o
  eth0 -j MASQUERADE
post-down iptables -t nat -D POSTROUTING -s '192.168.145.0/24' -o
  eth0 -j MASQUERADE
```

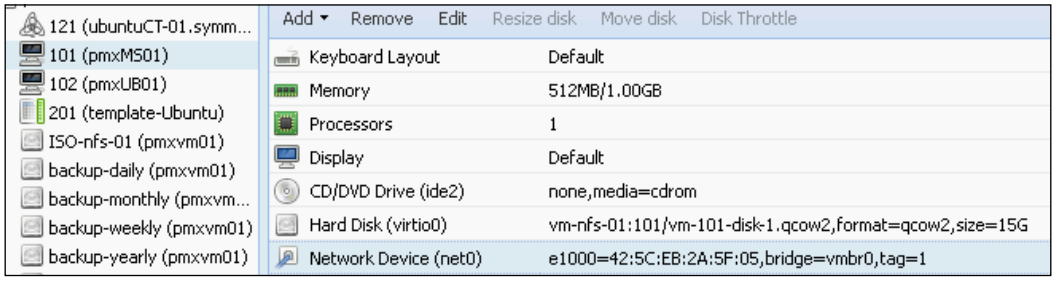
It is much easier and manageable to handle NAT using a physical or virtual firewall. Most of the firewalls have the NAT option out of the box. Also, using virtualized firewalls, we can create truly isolated virtual networks for multiple clients on the same Proxmox cluster. Having a virtual firewall provides the client control over their own filtering while keeping their network hidden from other client networks in the cluster.

# Adding VLAN

VLAN can be set up on both virtual machines and bridges. If the VLAN traffic leaves the virtual environment, it is important to set VLAN on physical network switch by configuring port trunking and VLAN tagging. Tagging VMs with a VLAN ID is very straightforward through the Proxmox GUI. Just enter the VLAN ID when adding a network interface to the VM or edit already added vNICs. The following screenshot shows the dialog box to edit a network device through the Proxmox GUI:




In the preceding example, we tagged virtual machine pmxMS01 in our cluster with VLAN ID 1. After adding VLAN with the VM, the following screenshot is what it looks like from the Proxmox GUI. Notice the VLAN ID 1 is **tag=1** at the end of **Network Device**. It should be noted that when using VLAN, all devices and virtual machines should be VLAN compatible.




We are now going to create a VLAN on the bonded interface bond0 on our Proxmox node **pmxvm01**. Perform the following steps to do so:

1. Securely log in to the node **pmxvm01**.
2. Open the interface file `# nano /etc/network/interfaces` using an editor.

3. Add the following lines at the beginning of the file:

```
auto bond0.10
iface bond0.10 inet manual
        vlan-raw-device bond0
```

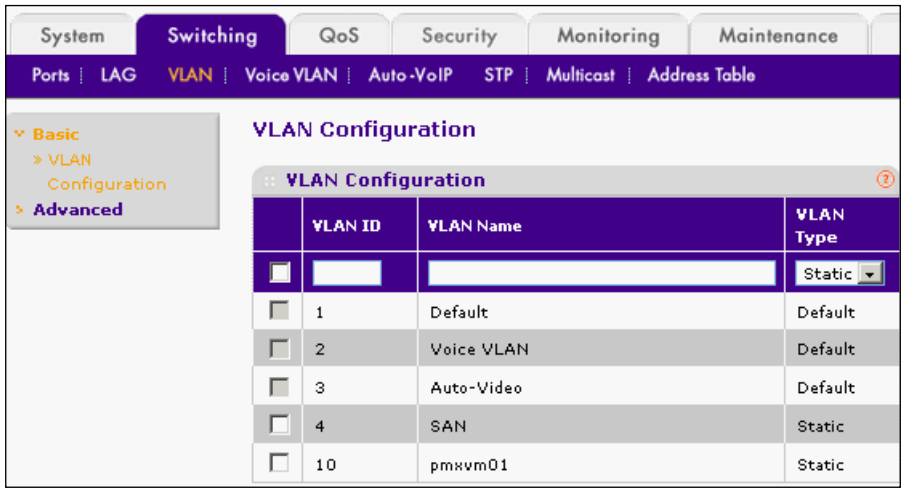
4. Create a new bridge `vmbr10` as follows:

```
auto vmbr10
iface vmbr10 inet manual
        bridge_ports bond0.10
        bridge_stp off
        bridge_fd 0
```

5. Save the file and exit the editor.
6. Activate the bridge by rebooting the node or from CLI by stopping and restarting the bridge using the following command:

```
# ifup vmbr10
```

7. Do the same for the second node pmxvm02.
8. Set up VLAN on the physical switch by configuring trunk ports. The following screenshot is an example of a VLAN setup on Netgear GS724T 24 port switch. VLAN pmxvm01 with ID 10 is set up for the vmbr10.



Our final network interface configuration should look as follows:

```
auto lo
iface lo inet loopback

# bring up interfaces
auto eth1
iface eth1 inet manual
auto eth2
iface eth2 inet manual
auto bond0.10
iface bond0.10 inet manual
    vlan-raw-device bond0

# bonding interfaces
auto bond0
iface bond0 inet manual
        slaves eth1 eth2
        bond_miimon 100
        bond_mode 802.3ad

auto vmbr0
iface vmbr0 inet static
        address 192.168.145.1
        netmask 255.255.255.0
        gateway 192.168.145.254
        bridge_ports eth0
        bridge_stp off
        bridge_fd 0

auto vmbr1
iface vmbr1 inet static
        address 192.168.10.1
        netmask 255.255.255.0
        bridge_ports bond0
        bridge_stp off
        bridge_fd 0

auto vmbr200
iface vmbr1 inet static
```

```
        bridge_ports none
        bridge_stp off
        bridge_fd 0


auto vmbr10
iface vmbr1 inet static
        bridge_ports bond0.10
        bridge_stp off
        bridge_fd 0
```

A good practice to identify which VLAN belongs to which bridge is to use the same numeric number for both the interfaces. For example, a bridge vmbr10 will have the same VLAN ID 10. Without some order in the beginning, bridges and VLANs will quickly get out of control as network grows over time.

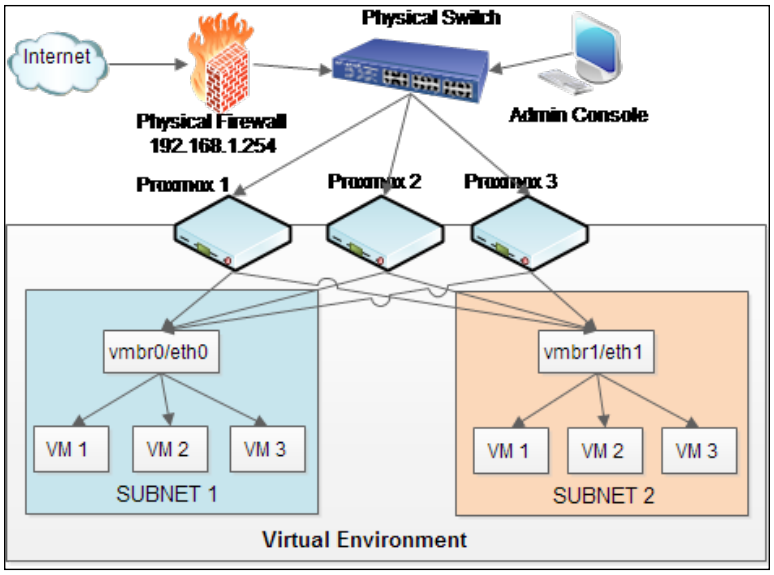
# Sample virtual networks

At this stage, we have covered components of virtual networks within the Proxmox cluster environment. We know the components Proxmox uses to hold everything together.

We are going to take a look at a few virtual environment scenarios to solidify our understanding of networking in the Proxmox virtual environment. These are scenario-based network diagrams, and some of them are taken from the real production environment.

# Network #1 – Proxmox in its simplest form

This is a small-scale Proxmox cluster with three nodes and two subnets within the virtual environment. Each Proxmox node has two NICs, and both bridges vmbr0 and vmbr1 are attached with eth0 and eth1, respectively. Each bridge has three virtual machines attached to them. Outside the virtual environment, there is a physical switch, which connects Proxmox nodes, and an admin console for all management work. This is Proxmox in its simplest form in a production environment. This type of network can be used as a learning platform or in a very small business environment with a less demanding workload. Internet connectivity is provided to the second subnet directly from the firewall with a second NIC, as shown in the following diagram:




# Network #2 – multitenant environment

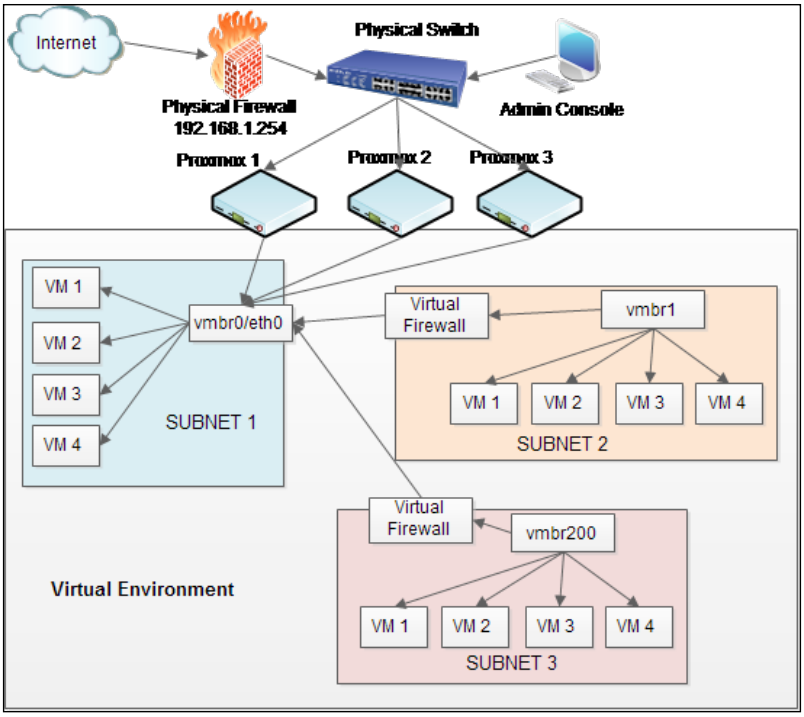
This network setup is almost the same as the previous network with the added benefit of fully multitenant virtual platform. In a physical firewall, we can only add a very small number of NICs to provide Internet connectivity to isolated subnets. Using a virtualized firewall, we can add as many firewalls or vNICs as we want. This setup is especially useful when multiple, isolated client subnets need to be hosted and each subnet requires its own firewall control for filtering purposes. In this example, vmbr0 is directly served by the physical firewall. Bridge vmbr1 and vmbr200 have their own virtualized firewalls. The firewalls also act as bridges between bridges. For example, the firewall for subnet 2 has two vNICs. One of these setups was WAN, where vmbr0 acts as an Internet provider. The second vNIC is LAN-facing, which serves the subnet 2.

This is a common scenario for infrastructure service providers who host virtual networks for multiple clients. Since multiple companies can access their virtual networks remotely, it puts extra workload on the physical firewall. Single-point firewall failure should be avoided at all costs by creating a cluster of physical firewalls to provide load balance and failover firewall service.

Never use a virtualized firewall on the same cluster to connect to the Internet directly. Always use separate physical hardware as the main firewall to act as a barrier between the Internet and internal network.

For firewall virtualization, **pfsense** is a great choice to set up. It is easy to set up, yet extremely powerful and customizable. Get pfsense and more information from `www.pfsense.org`.

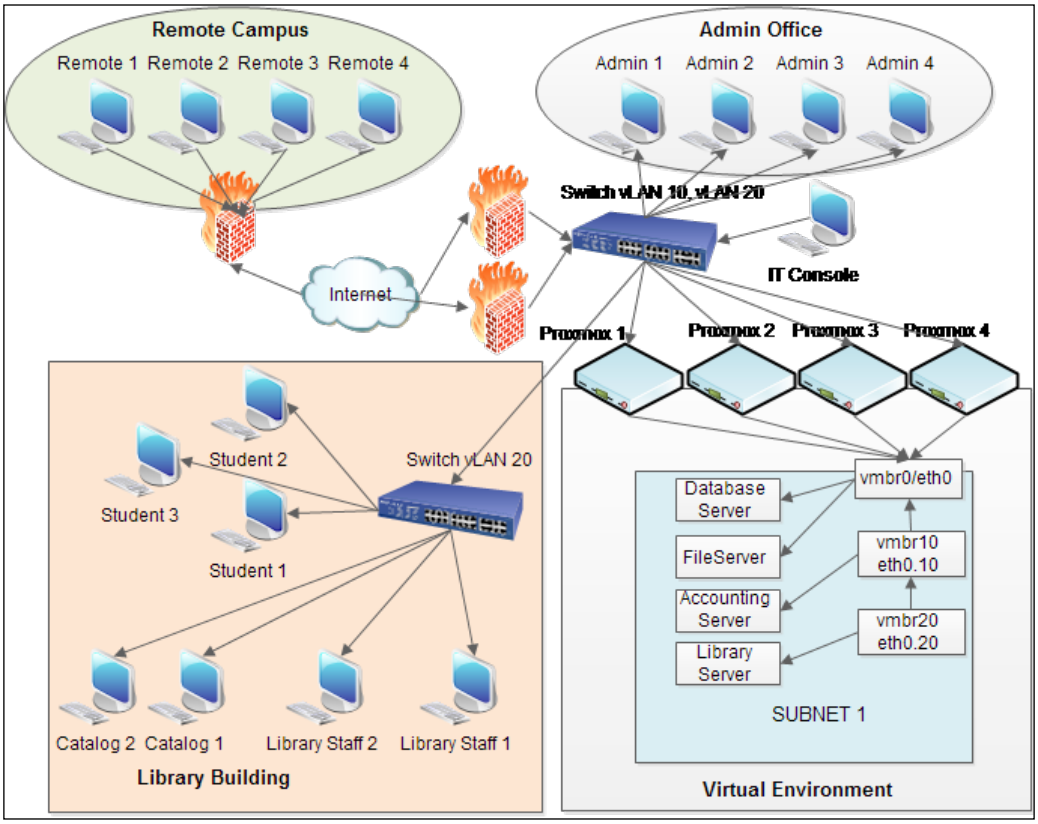
The following diagram is an example of a multitenant virtual environment:




# Network #3 – academic institution

This network diagram is an example of an academic institution network. The following diagram shows network connectivity between the admin office, library, and a remote campus. There are two physical firewalls providing Internet connectivity redundancies. The main virtual network consists of the database server, file server, accounting server, and library catalog server. The database server and the file server are connected with bridge vmbr0. The accounting server is connected with bridge vmbr10 and VLAN ID 10. The library server is connected with bridge vmbr20 and VLAN ID 20. The main switch is set up with VLAN 10 and VLAN 20. The library switch is set up with VLAN 20. In this set up, accounting server data goes straight to the admin office and the library catalog server data goes to the library building without causing additional stress on the network. Remote campus students and staff can access main campus network through VPN, thus eliminating the need to set up a separate virtual environment.

Of course, the following diagram is a very simplified form of the actual network topology of an academic institution. But the basics of using VLANs and bridges are the same for any network size.




# Multitenant virtual environment

Multitenancy is a very frequently used word in the world of cloud computing, where a virtual environment is regularly used by different clients from different organizations set up with fully isolated networks. Multitenancy is an integral part for a service provider who provides **Infrastructure-As-A-Service** (**IAAS**) to many clients.

To know more about cloud computing, visit `http://en.wikipedia.org/wiki/Cloud_computing`.

In this type of setup, the service provider hosts or "rents out" computing time and storage space to their clients. Because of standard monthly subscription or SLA-based payment methods required for this type of service, the term multitenancy quickly gained popularity. Basically, a multitenant virtual environment is where several isolated networks coexist on the same platform without interfering with one another. Almost all public datacenters are multitenancy platforms.

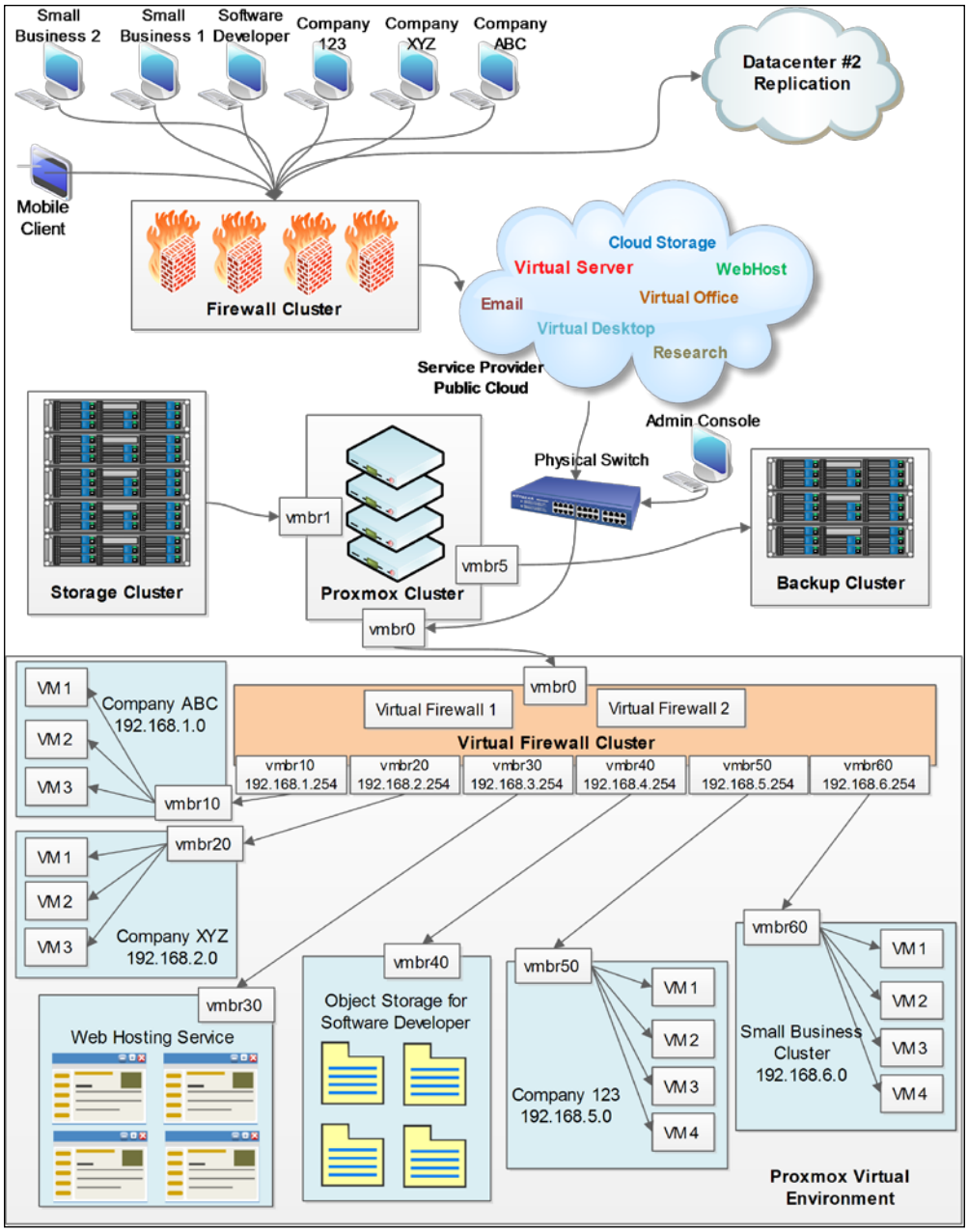
Multitenancy is not new in the world of information. The first multitenant environment appeared back in the 1960s, when companies rented processing time and storage space on mainframe computers to reduce giant expenses of mainframe operation. The virtual environment only augmented the same idea exponentially by leveraging all the virtualization features Proxmox provides. By combining virtualization with cloud computing, multitenancy is able to get a very strong footing to serve better and to more customers without increasing financial overheads. Prior to virtualization, the physical space and power requirements to host customers in an IAAS environment meant it was rare and very expensive, thus not many people enjoyed its benefit.

The Proxmox hypervisor is well capable of setting up a stable and scalable multitenant virtual environment. All the networking components we have seen so far, such as vNIC, virtual bridge, and VLAN, are the building blocks to setting up a multitenant virtual environment. Once we understand the relationships between virtual machines and virtual bridges, it is fairly easy to set up a multitenant virtual environment with Proxmox.

When setting up a multitenant virtual environment, it is very important to take special care so that one network traffic does not get intercepted by another network. Without a proper VLAN and subnet, it is possible for one network to sniff network packets on the entire virtual environment, thus stealing data from other tenant organizations on the network.

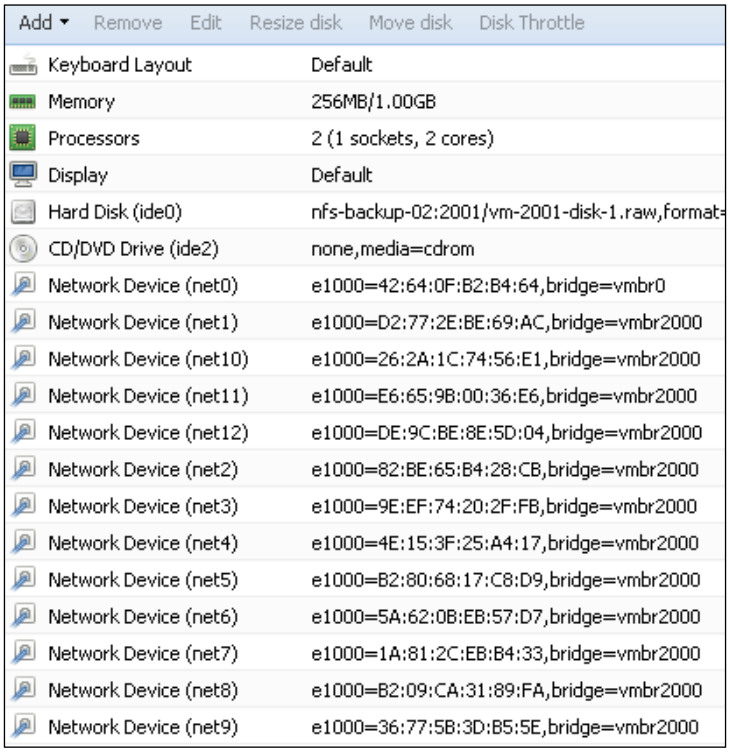
# Multitenant network diagram

The following is an example of a network diagram of a typical cloud service provider who provides IAAS to their clients. The entire client network is virtualized within the service provider's virtual environment.


Small Business 2
Small Business 1
Software Developer
Company 123
Company XYZ
Company ABC
Datacenter #2 Replication
Mobile Client
Firewall Cluster
Cloud Storage
Virtual Server
WebHost
Email
Virtual Office
Virtual Desktop
Research
Service Provider Public Cloud
Admin Console
Physical Switch
Storage Cluster
vmbr1
Proxmox Cluster
vmbr5
Backup Cluster
vmbr0
VM 1
VM 2
VM 3
Company ABC 192.168.1.0
vmbr10
vmbr0
Virtual Firewall 1
Virtual Firewall 2
Virtual Firewall Cluster
vmbr10 192.168.1.254
vmbr20 192.168.2.254
vmbr30 192.168.3.254
vmbr40 192.168.4.254
vmbr50 192.168.5.254
vmbr60 192.168.6.254
vmbr20
VM 1
VM 2
VM 3
Company XYZ 192.168.2.0
vmbr30
Web Hosting Service
vmbr40
Object Storage for Software Developer
vmbr50
VM 1
VM 2
VM 3
VM 4
Company 123 192.168.5.0
vmbr60
VM 1
VM 2
VM 3
VM 4
Small Business Cluster 192.168.6.0
Proxmox Virtual Environment

On the client side, they only have simple desktop computers and mobile devices to access their virtual cloud resources, such as desktop, storage, processing power, and so on. Clients access these resources through a virtual means, such as **Virtual Network Computing** (**VNC**), SPICE, or **Remote Desktop Protocol** (**RDP**).

Virtual networks are isolated with separate subnets. VLANs are set up (not shown in the diagram) to reduce mass broadcast traffic. All virtual machine data is stored on a separate storage cluster with full redundancy. A backup cluster does regular backup of all virtual machines, and granular file backup with histories are done with a third-party backup software. A virtual firewall cluster is set up in between the virtual environment and the host Ethernet interface to provide Internet connectivity to all client virtual machines. Each virtualized firewall has several vNICs to connect to each subnet. A typical virtual firewall with multiple vNIC would look as shown in the following screenshot:



Since the firewall is virtualized, we can add any number of virtual network interfaces without worrying about running out of physical slots. A virtualized clustered firewall provides maximum uptime. Each company network in this example has its own virtual bridge, which only talks to that company's virtual machines and firewall interface, eliminating any chance of packet sniffing by other company networks.

**Packet sniffing** is a process when data packets passing through a network interface are captured and analyzed. A packet sniffer software can be placed in a subnet to capture data. This is common practice of someone with malicious intention to capture sensitive unencrypted data passing through such as usernames and passwords in clear texts.

This environment is serving multiple clients or organizations, so uptime is a big concern. To eliminate this issue, the entire virtual environment is replicated to another datacenter for ensuring 99.9 percent uptime. The previous diagram is an overly simplified version of what really goes on inside a very busy Proxmox virtual environment. Studying this diagram will give you a clear understanding of virtual network mechanics. From the previous diagram, we can see that this network environment heavily uses virtual bridges. So it is imperative to understand the role of bridges and plan out a draft diagram before actually setting up a virtual network of this level of complexity.

When working with a complex virtual network, always keep a network diagram handy and update it whenever you make any changes. An up-to-date network diagram will help greatly to have total control over a virtual network. Especially when any issue arises, it is easy to pinpoint the cause of issue with a diagram.

# Summary

We were very busy in this lively chapter. We looked at the differences between physical and virtual networks. We learned about the Proxmox network components that make up a Proxmox-based virtual network. We saw how to work with the configuration file to add and remove virtual interfaces. We even got to analyze a few network diagrams from the basic to the advanced to get a better understanding of how the Proxmox virtual network really comes to life. We even looked at what a multitenant network is and how the common datacenter environment looks like under the hood.

Proxmox provides all the tools we need to build any level of virtual network. It comes down to the network administrator's imagination, the company's budget, and the need to foresee how all pieces should come together to form a well-designed and efficient virtual network. The best part is that any mistake is easily correctable in a virtual environment. We can always go back and change things until we are satisfied. For this very reason a virtual network is always evolving. Over time, a virtual network becomes an extension of the network administrator's mind network. The configurations and design of a virtual network infrastructure can give us a window into how that administrator thinks and the logic they used to construct the environment.

In the next chapter, we are going to learn the High Availability feature of Proxmox, which provides automatic redundancy to the Proxmox virtual environment. To reduce downtime, it is important to have a functional High Availability to automigrate virtual machines to healthy nodes during any incident. We will look at some configuration files to enable High Availability in our cluster.

# 6
# Proxmox HA – Zero Downtime

In this chapter, we are going to see one of the most important features, which makes Proxmox an enterprise-class hypervisor. Proxmox High Availability or Proxmox HA gives the ability to move or migrate virtual machines from one node to another without any user interaction. We will look at the following topics:

- Understanding High Availability
- Which environment can fully take advantage of HA
- How to set up HA in Proxmox
- Downside of High Availability in Proxmox

## Understanding High Availability

High Availability is a combination of components and configurations that allow continuous operation of a computational environment on a daily basis. Basically, it means that even when unattended server hardware goes bad in a live environment, High Availability can manage the server on its own and keep a virtual environment running by automatically migrating virtual machines from one node to another. A properly set up HA requires very little actual user interaction during hardware failure. Without HA in place, a system administrator will have to monitor the environment for failures and manually move or migrate virtual machines to healthy nodes.

In a small environment this can be manageable, but in a large environment of hundreds of virtual machines and nodes, constant monitoring is just not realistic. Although there can be monitoring software in place to automatically alert administrators about any node failure, without HA the administrator will have to manually move or migrate any virtual machine from a faulty node. This can cause longer downtime due to human response time. HA takes the longer response time out of the equation by simply moving or migrating virtual machines to a node as soon as server hardware failure occurs.

# High Availability in Proxmox

To set up a functional HA, it is important to have all the virtual machines on a shared storage. Proxmox HA only handles Proxmox nodes and virtual machines within a Proxmox cluster. These HA features are not be confused with shared storage redundancy, which Proxmox can utilize for it's HA deployment.
A third-party shared storage may provide its own HA features.

There can be levels of redundancy in a Proxmox computing node, such as the use of RAID, redundant power supply, aggregated network link or bond, and so on. HA in Proxmox is not a replacement for any of these layers. It just facilitates redundancy features for virtual machines to keep running during a node failure.

It should be noted that in a Proxmox node, a reboot due to an applied update will cause all HA-enabled virtual machines to shut down and move to the next available Proxmox node and restart again. In such a situation, it may be necessary to manually live migrate virtual machines first, which requires minimum interruption, before rebooting the node.

# Requirements for HA setup

Proxmox HA can be set up with just two Proxmox nodes. But since this type of setup is not ideal for a production environment, three or more Proxmox nodes are recommended to set up a Proxmox HA because with three nodes or more, achieving Quorum is possible. Quorum is the minimum number of votes required for a Proxmox cluster operation. This minimum number is the total vote by a majority of the nodes. For example, in a cluster of three Proxmox nodes, a minimum vote of two Proxmox nodes is required to form a Quorum. In another example of clusters with eight-nodes, a minimum vote of 5 Proxmox nodes is required to form a Quorum. With just two nodes, the ratio of vote remains at 1:1, so no Quorum is possible. To know more about how to set up High Availability with just two Proxmox nodes, visit `http://pve.proxmox.com/wiki/Two-Node_High_Availability_Cluster`. Almost all the configurations of Proxmox HA can be done from the GUI, except the **fencing** configuration. This portion of the setup must be done through the CLI.

# Fencing

Fencing is a concept of isolating a node or its resources during node failure so that other nodes cannot access the same resources causing data corruption. In Proxmox, fencing prevents multiple nodes from running on the same virtual machine or cluster-specific services. Without fencing, Proxmox HA cannot be set up. Fencing ensures data integrity during a failure by preventing all nodes from running on the same virtual machine or cluster services at the same time. There are several ways to set up fencing such as the following:

- Using `fence_xvmd`: The host node needs to be configured with `fence_xvmd` while the virtual machine needs to be configured with `fence_xvm`
- Using `libvirt`: Both the host and virtual machines need to be configured with `libvirt`
- Using **Intelligent Platform Managament Interface** or **IPMI**
- Using managed switch with SNMP
- Using managed **Power Distribution Unit** or **PDU**

In this chapter, we are going to learn how to set up fence and HA with managed PDU as a hardware fencing device.

Managed PDUs are physical hardware devices, which enable a Proxmox node to power cycle another node during a failover before starting the HA service. An example of such a fencing device is APC-managed PDU AP7921 (`http://www.apc.com/products/resource/include/techspec_index.cfm?base_sku=AP7921`).

A managed PDU allows an administrator to remotely power cycle a device connected to the power outlet of the unit. This is useful when IPMI is not available and a node needs to be rebooted. By logging in remotely into a PDU user interface, any power outlet of the PDU can be turned on or off.

There are the following two types of fencing available:

- Resource-level fencing
- Node-level fencing

**Resource-level fencing** prevents the same cluster-related services or resources from starting on all nodes in the cluster simultaneously. **Node-level fencing** makes sure that the node in question does not run any services at all by simply power cycling the node with the help of a managed PDU or IPMI. This method is also known as **STONITH** or **Shoot The Offending Node In The Head**. There are several types of fencing devices, such as a network-managed switch, APC master switch, Dell drac card, and so on. In this chapter, we are going to use APC-managed PDU AP7921 to set up device-based fencing. AP7921 is completely tested with Proxmox and very much dependable. Also, setting up fencing with PDU is very simple. If you would like to try setting up fencing using other methods, visit the Proxmox Wiki at `https://pve.proxmox.com/wiki/Fencing`.

We can summarize the requirements of device-based fencing and HA setup as follows:

- A minimum of three properly set up and updated Proxmox nodes. Currently, a maximum of 16 Proxmox nodes per cluster can be used for HA.
- Shared storage for all virtual machines.
- A fencing device setup.

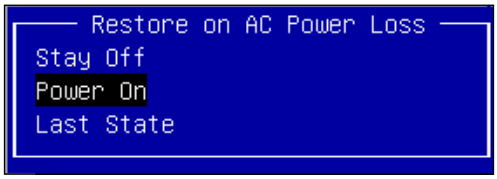
# Configuring Proxmox HA

High Availability configuration in Proxmox is a five-step process as follows:

1. Set up node BIOS
2. Create an APC-managed PDU user
3. Set up fencing
4. Configure a VM/OpenVZ container
5. Test HA

# Setting up node BIOS

Before we set up fencing and Proxmox HA, we have to make sure that nodes can boot immediately after a power cycle or power loss. Usually, this feature is disabled by default, as shown in the following screenshot:

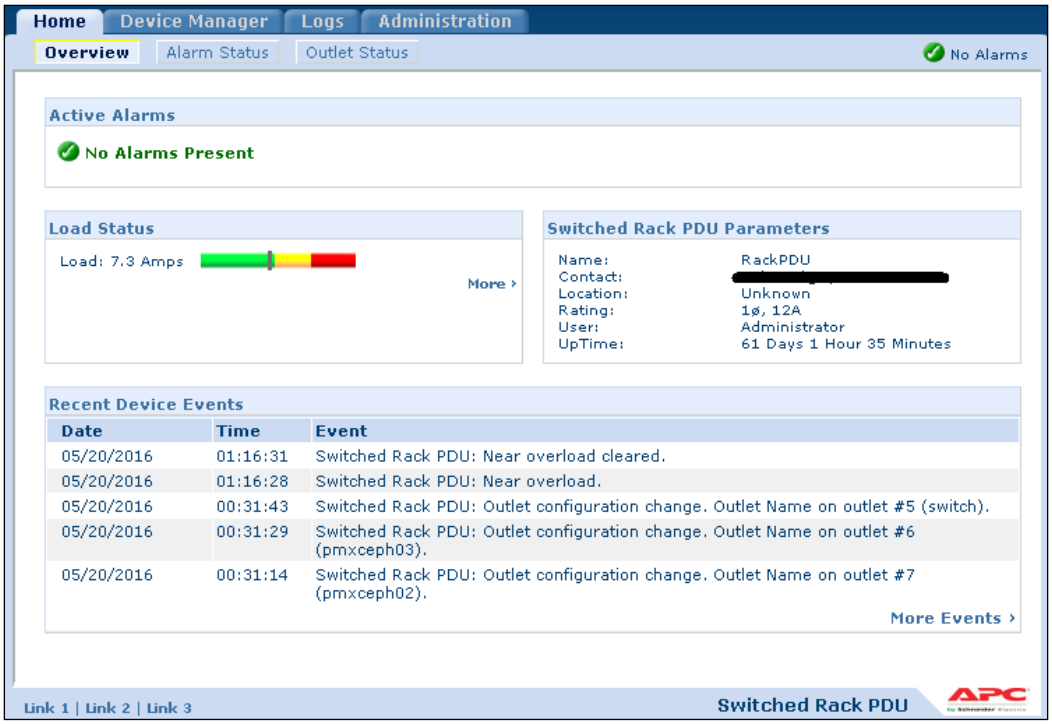


To make sure that this feature is truly enabled, try to unplug the power cord and then plug it back in to see if the node powers up. Without this feature enabled, the node cannot be power cycled by the fencing device.

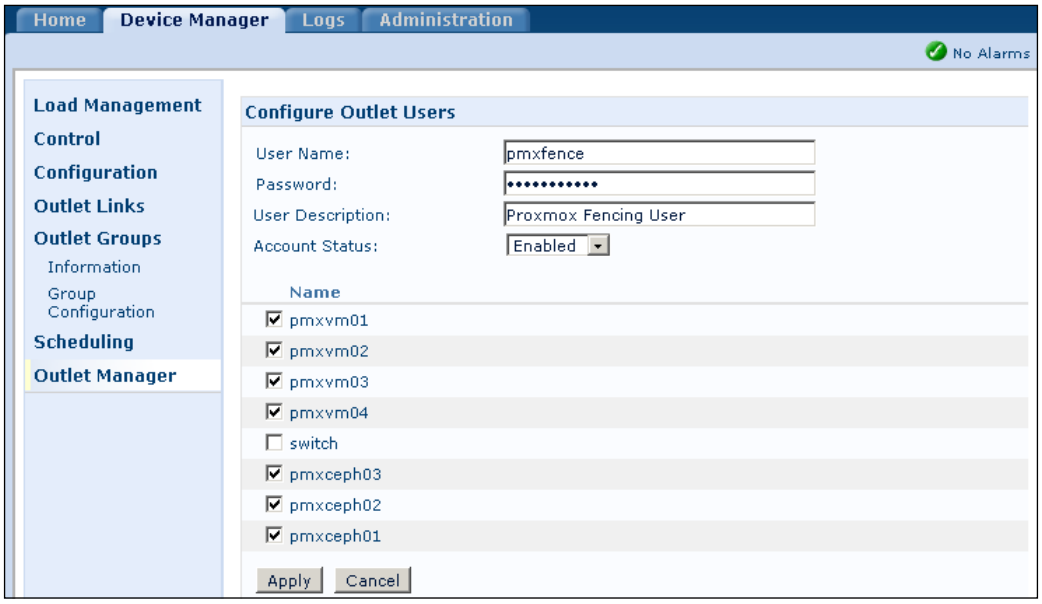
# Creating an APC-managed PDU user

To create an APC-managed PDU user, we have to create a user on the APC PDU fencing device, which we will use with our fencing configuration. The following steps create a limited access for the user without any admin privileges. This allows the user account to only interact with power outlets such as the power cycle and not the administration of the PDU itself. This unit comes with its own web-based control panel. Go to the admin section securely with a browser simply by entering the device IP address as `https://<apc_device_ip_address>`.

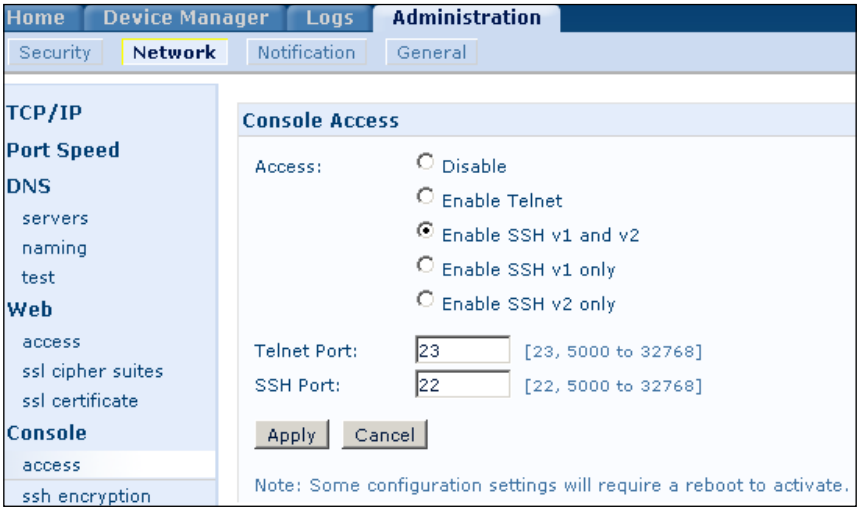
A typical APC web admin section is shown in the following screenshot:



We can create the new user from **Outlet Manager** under the **Device Manager** tab. Fill in the information as shown in the following screenshot and then simply hit **Apply**:

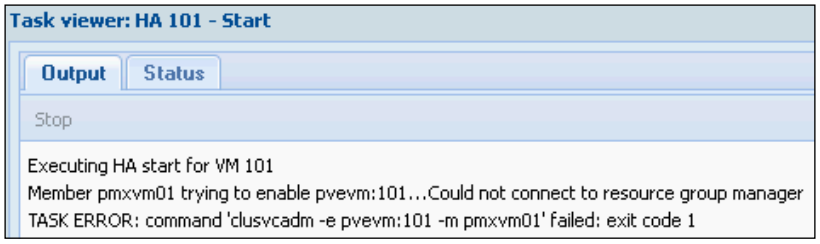



Next, make sure to have the SSH access enabled by navigating to **Administration | Network**. SSH will allow this newly created user to perform a power cycle without a login via graphical interfaces. The following screenshot shows the **Network** menu:



# Configuring Proxmox fencing

Fencing must be set up before starting the actual Proxmox HA configuration process. Without a properly set up fencing, the error message shown in the following screenshot will be produced by Proxmox:




Before we get to the actual fencing configuration, we have to enable fencing on all nodes by performing the following steps:

1. Uncomment the very last line in the file `#/etc/default/redhat-cluster-pve`. Do this only on one node. The edited file should look as follows:

```
# this file is sourced by the following init scripts:
# /etc/init.d/cpglockd
# /etc/init.d/cman
# /etc/init.d/rgmanager

FENCE_JOIN="yes"
```

2. Join each node by using the following command from CLI. Do not join a node into a fence if you do not want it be part of High Availability:

```
# fence_tool join
```

   If the node ever needs to be detached from fencing, then simply run the following command from the node itself:

```
# fence_tool leave
```

3. After adding all the nodes, check the status using the following command:

```
# fence_tool ls
```

   If all went okay, then the output should look like this:

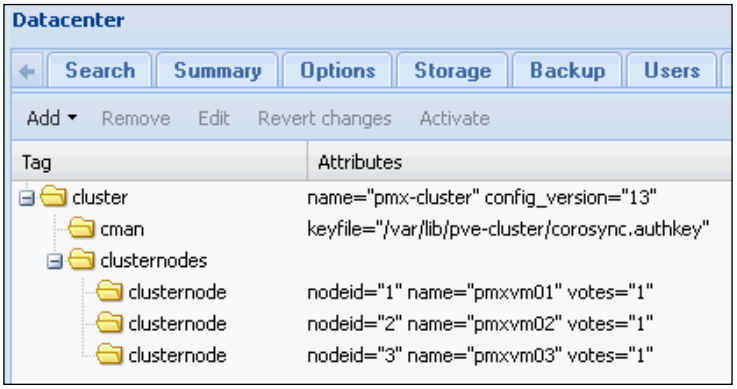
```
fence domain
member count    3
victim count    0
victim now      0
master nodeid   1
wait state      none
member          1 2 3
```

The `fence_tool` program is in the Linux operating system. It is used to join or leave nodes in the default fence domain. It also starts the fence daemon `fenced`. Even though `fenced` can be started or stopped without `fence_tool`, this small tool takes some added measures that are very helpful in a production environment. To get more information on `fence_tool` and other options that can be used, visit `http://linux.die.net/man/8/fence_tool`.

4. We are now going to edit the `/etc/pve/cluster.conf` file to add a fencing device. We can edit the file from the CLI and activate it from the GUI. We must remember to increase the `config_version` number when we make any changes to the `cluster.conf` file. This ensures that all the nodes will apply the new settings. Our current `cluster.conf` file looks as follows:

```
<?xml version="1.0"?>
<cluster config_version="13" name="pmx-cluster">

  <cman keyfile="/var/lib/pve-cluster/corosync.authkey">
  </cman>

  <clusternodes>
    <clusternode name="pmxvm01" nodeid="1" votes="1"/>
    <clusternode name="pmxvm02" nodeid="2" votes="1"/>
    <clusternode name="pmxvm02" nodeid="3" votes="1"/>
  </clusternodes>

</cluster>
```

We can also see the `cluster.conf` file from the Proxmox GUI by navigating to **Datacenter** | **HA**. We will see `cluster.conf` as shown in the following screenshot:



Before we start editing the `cluster.conf` file, we are going to copy the file and edit the copied file. That way we can activate it from the GUI. The command to do so is as follows:

```
# cp /etc/pve/cluster.conf /etc/pve/cluster.conf.new
```

Any changes made to a configuration file with the extension `.new` at the end are treated as temporary changes. When activated from the GUI or through node reboot, these temporary changes permanently get transferred to the main configuration file. In this case, our main cluster file is `/etc/pve/cluster.conf`. This also helps to validate any new changes before they are applied. Validation checks each line to make sure there are no syntax errors in the configuration file. Validation can be done through the following command:

```
# ccs_config_validate -v -f /etc/pve/cluster.conf.new
```

Let us now add the fencing to the cluster configuration file `cluster.conf.new`. In this configuration, the `power_wait` option sets the delay between performing power cycles. Without this option, nodes will be turned off immediately and turned back on without any delay. Setting a delay ensures that nodes will be turned off for a set amount of time before powering them back on. The following code shows delay set for 10 seconds:

```
# nano /etc/pve/cluster.conf.new

<?xml version="1.0"?>
<cluster config_version="13" name="pmx-cluster">

  <cman keyfile="/var/lib/pve-cluster/corosync.authkey">
  </cman>

  <fencedevices>
    <fencedevice agent="fence_apc" ipaddr="192.168.145.250"
      login="pmxfence" name="apc" passwd="99999" power_wait="10"/>
  </fencedevices>

<clusternodes>
  <clusternode name="pmxvm01" nodeid="1" votes="1">
    <fence>
      <method name="power">
        <device name="apc" port="1" secure="on"/>
      </method>
    </fence>
```

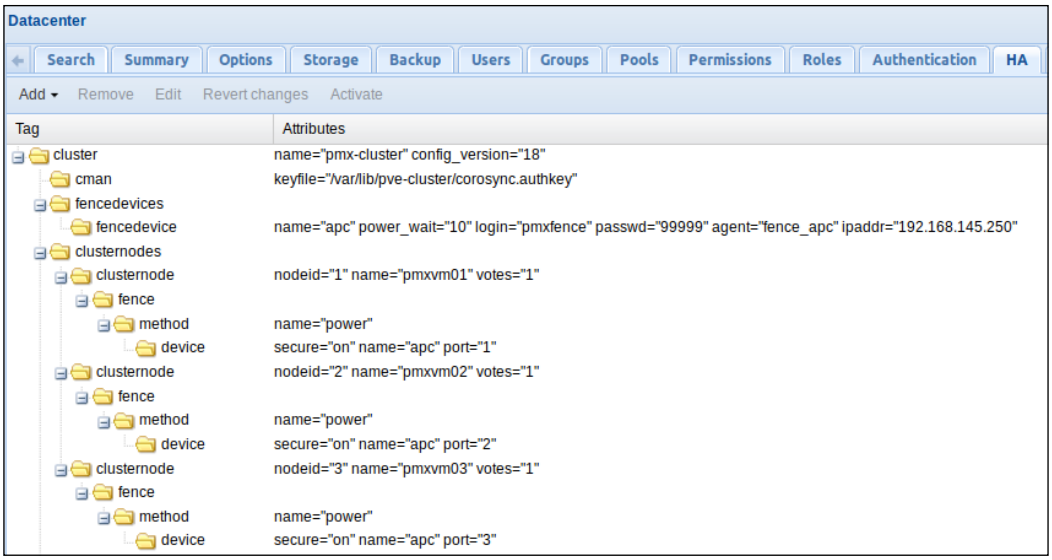
```
    </clusternode>

    <clusternode name="pmxvm02" nodeid="2" votes="1">
      <fence>
        <method name="power">
          <device name="apc" port="2" secure="on"/>
        </method>
      </fence>
    </clusternode>

    <clusternode name="pmxvm03" nodeid="3" votes="1">
      <fence>
        <method name="power">
          <device name="apc" port="3" secure="on"/>
        </method>
      </fence>
    </clusternode>

  </clusternodes>

  </cluster>
```

Take special care to enter the right port for the right node. In the APC PDU, each power socket is referred to as a port. If a node is plugged into the wrong port or socket, the wrong node might go through the power cycle. Always label the power cords so that they do not get mixed up.

After we have edited the `cluster.conf` file, let us check the file for any errors with the `ccs_config_validate` command. If there are no errors, it should show the following messages:

```
Creating temporary file: /tmp/tmp.jOGM6BP15k
Config interface set to:
Configuration stored in temporary file
Updating relaxng schema
Validating..
Configuration validates
Validation completed
```

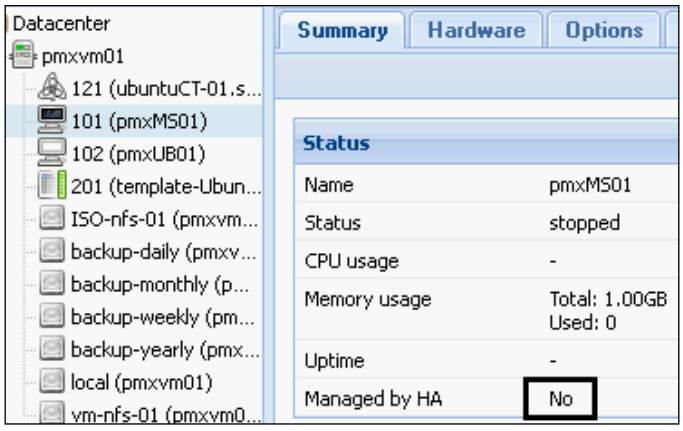
Let us now look at the temporary configuration by navigating to **Proxmox GUI | Datacenter | HA**. The GUI should look like the following screenshot prior to activating the changes:



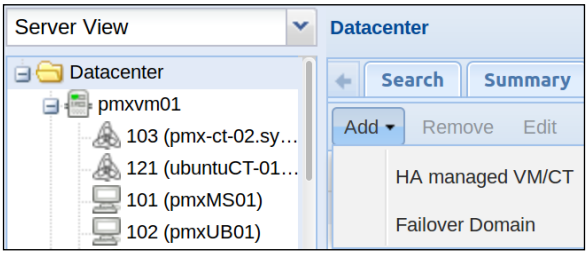

If everything looks good without any errors, click on **Activate** to apply the changes. No reboot will be required.

# Configuring virtual machine HA

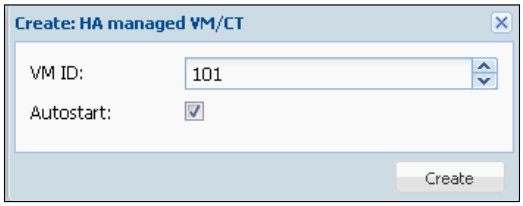
With the fencing activated, we can now proceed to configuring the virtual machine to become part of the High Availability feature. All VM configurations can be done through the GUI. We can check the HA status of a VM from the virtual machine **Summary** screen. When a VM is not part of High Availability, the **Summary** screen will show **No** for the **Managed by HA** status, as shown in the following screenshot:



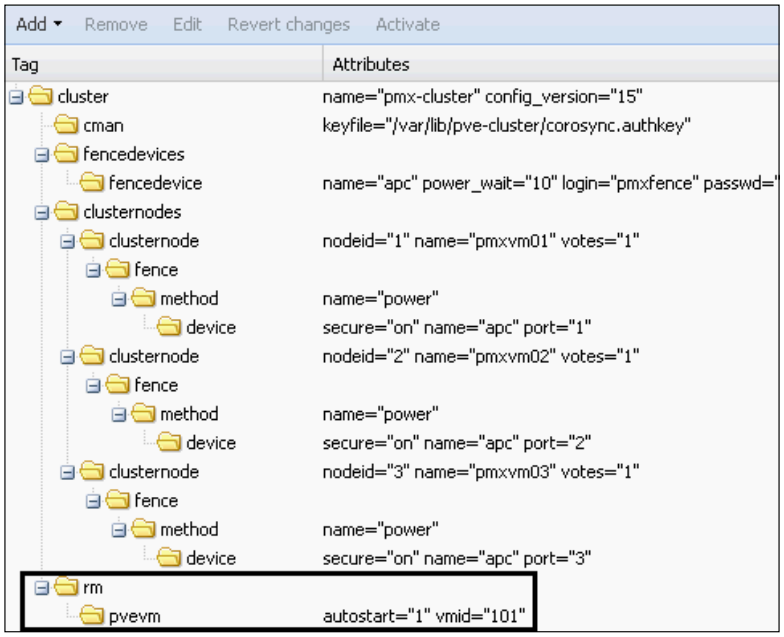
We can enable the HA feature for a virtual machine by navigating to **Datacenter** | **HA**, as shown in the following screenshot:




Click on **HA managed VM/CT** to open the option window, as shown in the following screenshot, to add a virtual machine. Simply select a VM/CT ID and then click on **Create**. In our cluster, we are going to add VM 101 into the HA.

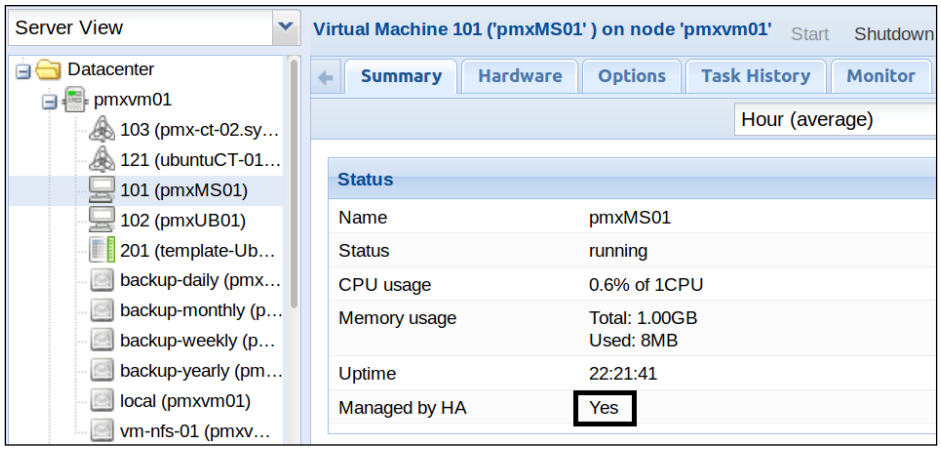



Notice the addition of a new segment in the `cluster.conf`, as shown in the following screenshot:



In the previous screenshot, **rm** simply means resource manager. We can see that the VM 101 is now part of Proxmox HA. Any virtual machine we add in HA will show under the **rm** segment of `cluster.conf`.

We can also see the virtual machine's HA status in the **Summary** screen, as shown in the following screenshot:




# Testing Proxmox HA

Now that we have properly set up High Availability, it is time to put it to test to observe its functionality. The test is as simple as pulling the power cord out of the node and watch High Availability come into action. Also, unplug the network cable from the node to simulate a node failure. This should also power cycle the failed node and move all the virtual machines to another node.

If there are any error messages, check the error log to pinpoint the cause. Usually, the error happens due to a disabled SSH on the managed PDU. The chances of getting an error are lower if the `cluster.conf.new` file has been validated with the validation command.

# Fencing manually

When fencing hardware is not present, manual fencing can be used for testing or learning purposes. During the manual fencing process, the fencing agent `fence_manual` (`http://linux.die.net/man/8/fence_manual`) writes into the system log of the node indicating that the node requires fencing. An administrator must manually reset the node and then run the following command from the command prompt to acknowledge that the node has been reset:

```
# fence_ack_manual <node_name>
```

Recovery of the node will continue only after the command has been entered.

Manual fencing requires user intervention and should not be used in a production environment.

The command `fence_ack_manual` can also be used if any fence failure occurs during fencing. Cluster operation is stopped during fencing. So by manually acknowledging node reset through the `fence_ack_manual` command, cluster can resume operation. The `fence_ack_manual` command is not dependent on any service and can be used whenever a fence failure occurs.

# Proxmox HA need to know

Although the Proxmox High Availability feature is a great addition to a virtual cluster environment, there are few things you will have to keep in mind while implementing this feature.

If a Proxmox HA node needs a reboot due to a kernel update or anything else, then the `rgmanager` program must first be stopped on the node before rebooting. It can be stopped through the CLI. The command usually takes a while to complete. Always monitor the Task Log from the GUI. When the `rgmanager` program is stopped, reboot the node. To stop the `rgmanager` program, use the following command:

```
# /etc/init.d/rgmanager stop
```

The **rgmanager** or **resource group manager** program manages High Availability resources such as cluster services to provide failover abilities. During a node failure, `rgmanager` relocates cluster-specific services to different nodes with minimal interruption.

# Summary

In this chapter, we have covered an important feature named Proxmox High Availability. It is a feature that gives control to Proxmox nodes to automigrate virtual machines during a node failure or maintenance. We also saw how this feature actually works and what are the steps involved in setting it up. We learned about new equipment called managed PDU, which allows an administrator to power up or power down an equipment remotely. We also configured one of our virtual machines to take advantage of High Availability.

In the next chapter, we are going to look at an enterprise-class distributed storage system named Ceph, and set up a Ceph cluster to work with Proxmox. We will go through a step-by-step process of the Ceph configuration and have a deeper understanding of why an administrator should use Ceph and how it works.

# 7
# High Availability Storage for High Availability Cluster

In *Chapter 3*, *Shared Storages with Proxmox*, we looked into what a shared storage is and why it is important for a Proxmox cluster. In this chapter, we are going to dig deeper into the concept of shared storage by setting up an enterprise-class storage cluster using Ceph. This storage system allows you to build a fully scalable storage cluster using commodity hardware, while providing a stable storage platform to store all virtual machines. Throughout this chapter, we are going to cover the following topics:

- Definition of Ceph
- Reasons to use Ceph as a shared storage
- Components used in Ceph
- A step-by-step process to set up a Ceph cluster
- Definition of Ceph FS
- Getting to know Ceph's CRUSH map
- Storage pool management in Ceph
- Learning Ceph through a virtual Ceph setup

# Introducing the Ceph storage

Ceph is a distributed storage system designed keeping performance, reliability, and scalability in mind. Performance, reliability, and scalability are three keys for any enterprise-class network environment, and Ceph provides just that. As the capacity of Ceph cluster grows with the addition of new hard drives or solid-state drives, performance grows with it. This is simply because when user data is sent to a Ceph cluster, it is divided into a number of pieces depending on the disk drives and is written simultaneously by all the disk drives. Of course, this is an overly simplified explanation of the entire process. We will look into Ceph mechanisms later in this chapter. Ceph's built-in self-healing system provides outstanding reliability, and the simplicity of adding disk drives or a physical node in the cluster to increase overall capacity provides unprecedented scalability.

A Ceph cluster provides the following three storage types:

- Object Storage
- Block Storage
- Filesystem

# Object Storage

Ceph **Object Storage** is the foundation of all other storage features offered by Ceph clusters. Data is written and retrieved through an API that is compatible with Amazon S3 or OpenStack Swift. This is a suitable solution for cloud storage integration and not the storage of virtual machine images. Currently, Proxmox does not support Ceph Object Storage.

For more details on Ceph Object Storage, visit `http://ceph.com/docs/master/radosgw/`.

# Block Storage

Ceph **Block Storage** stores block device images as objects by interacting with Ceph Object Storage. This storage is also known as **RADOS Block Device** (**RBD**), which is supported by Proxmox. Ceph RBD is integrated with kernel, allowing access to virtual machine images on RBD storage. Ceph Block Storage can be connected with Proxmox through a GUI using the RBD storage plugin.

For more details on Ceph Block Storage, visit `http://ceph.com/docs/master/rbd/rbd/`.

# Filesystem

Ceph **Filesystem** sits on top of Object Storage and provides a POSIX-compliant filesystem to store files and folders such as a disk drive. The Filesystem storage type uses a **MetaData Server** (**MDS**), which maps files and directories stored in the Filesystem to the objects stored in Object Storage. MDS can also rebalance data stored in Filesystem among cluster hosts, ensuring high performance and preventing higher load on a specific host. Loss of MDS will cause loss of data. Ceph FS can be integrated with Proxmox by mounting a local directory on the Ceph FS pool.

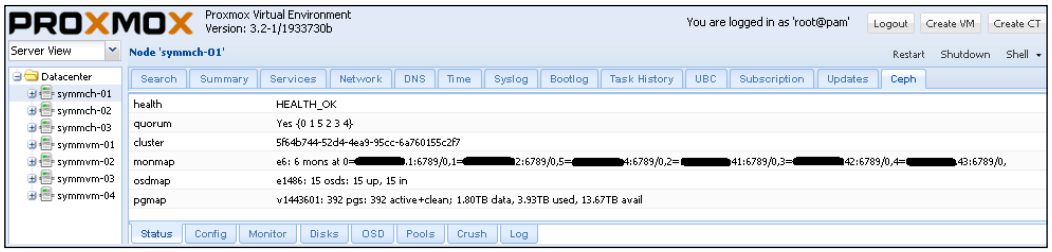
To learn more about Ceph FS, visit `http://ceph.com/docs/master/cephfs/`.

If you have never used Ceph before, it might be a little difficult at first to grasp the concept of how it works because Ceph has a non-traditional approach toward storing and distributing data in a Ceph cluster. This mechanism allows new disk drives or nodes to be added to the cluster and increase overall storage space within minutes. It does not matter if you are building a 20-terabyte or 20-petabytes storage system; the entire scalability process is the same for all sizes. Imagine a RAID 5 or 6 system with 10 HDD in a single server node. RAID will allow the failure of multiple disk drives and continue to provide node services. Now imagine a Ceph cluster with five physical nodes and eight HDDs in each node. If one of the nodes fails completely, along with all the eight HDDs in that node, the storage system will not stop working. Instead, it will go into self-healing mode while providing an uninterrupted storage service and giving you the opportunity to replace the failed node.

# Reasons to use Ceph

In the data storage world, there are many choices other than Ceph. What sets Ceph apart is the simplicity of Ceph's cluster management and the significant cost reduction by leveraging commodity hardware and the absence of a vendor license lock-in. Ceph provides all these features without compromising on any enterprise-class feature such as performance, reliability, and scalability.

Proxmox recently released Version 3.2 of the hypervisor integrated with a Ceph API, allowing users to manage a full Ceph cluster using the Proxmox GUI. Now Proxmox and Ceph can actually coexist on the same node, thus reducing management overhead. The following screenshot shows the integration of Ceph in the Proxmox GUI:




Without the Proxmox GUI, the only way to manage a Ceph cluster is through the CLI. There is a list of Ceph commands at the end of this chapter in *The Ceph command list* section.

With Ceph's architecture, a user talks to disk drives or OSDs when directly requesting for data in the cluster (explained in *The Ceph components* section). There are no single points of failure, with the exception of Ceph FS, where an MDS maintains a map of files and directories in the Ceph FS. Thus, by simply adding new disk drives or nodes, a Ceph cluster scales up simply by adding new disk drives or nodes.

Ceph was created with commodity hardware in mind. Ceph's open model eliminates vendor lock-in issues through proprietary appliance or an additional software layer. It can be installed on just about any off-the-shelf hardware. Of course, for reliability, quality hardware should be used.

# Virtual Ceph for training

It is possible to set up an entire Ceph cluster in a virtual environment. However, this cluster should only be used for training and learning purposes. If you are learning Ceph for the first time, and do not want to invest in physical hardware, then a virtualized Ceph platform is certainly possible. This will eliminate the need for physical hardware to set up Ceph nodes.

# The Ceph components

Before we dive in, let's take a look at some key components that make up a Ceph cluster.

# Physical node

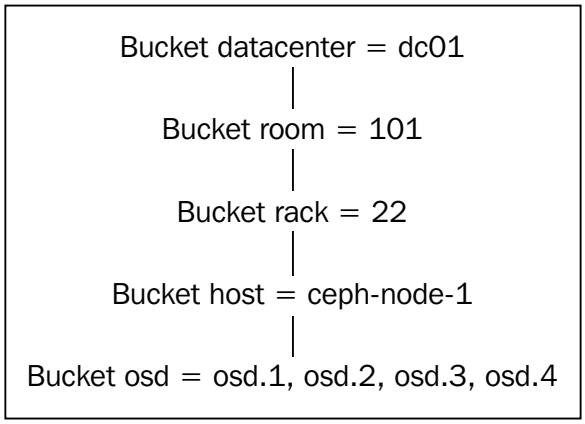
A physical node is the actual server hardware that holds **Object Storage Daemon** (**OSDs**), monitors, and MDSs.

# Maps

In Ceph, maps hold information such as a list of participating nodes in a cluster and their locations, data paths, and a list of OSDs with certain data chunks. There are several maps in a Ceph cluster: a cluster map, OSD map for a list of OSDs, monitor map for known monitor nodes, **Placement Group** (**PG**) map for the location of objects or data chunks, and a CRUSH map to determine the storage and retrieval of data by computing the data storage location.

# Cluster map

A cluster map is a map of devices and buckets that comprise a Ceph cluster. Ceph uses a bucket hierarchy to define nodes or node locations, such as a room, rack, shelf, and host. For example, let's say there are four disk drives used as four OSDs in the following bucket hierarchy:




In the previous example, osd.1 to osd.4 are in the ceph-node-1 rack in rack 22, which is in room 101 inside datacenter dc01. If osd.3 fails, and there is an on-site technician, then an administrator can quickly give the technician the previous bucket hierarchy to identify the exact disk drive location to replace it. There could be hundreds of OSDs in a cluster. A cluster map helps to pin point a single host or disk drive using the bucket hierarchy.

# CRUSH map

**Controlled Replication Under Scalable Hashing** (**CRUSH**) is an algorithm used in Ceph to store and retrieve data by computing data storage locations within the cluster. It does so by providing a per-device weight value to distribute data objects among storage devices. The value is autoassigned based on the actual size of the disk drive being used. For example, a 2 TB disk drive may have the approximate weight of 1.81. The drive will keep writing data until it reaches this weight. By design, CRUSH distributes data evenly across weighted devices to maintain a balanced utilization of storage and device bandwidth resources. A user can customize a CRUSH map to fit any cluster environment of any size.

For more details on CRUSH maps, visit `http://ceph.com/docs/master/rados/operations/crush-map/`.

# Monitor

A **Ceph Monitor** (**MON**) is a cluster monitor daemon node that holds the OSD map, PG map, CRUSH map, and Monitor map. Monitors can be set up on the same server node with OSDs or on a fully separated machine. For a stable Ceph cluster, setting up separate nodes with Monitors is highly recommended. Since Monitors only keep track of everything that happens within the cluster and not the actual read/write process of cluster data, the Monitor node can be very underpowered, and thus less expensive. To achieve a healthy status of the Ceph cluster, a minimum of three monitors is needed to be set up. A healthy status is when every status in the cluster is **OK**, without any warnings or errors. Note that with the recent integration of Ceph with Proxmox, the same Proxmox node can be used as a Monitor. Starting from Proxmox 3.2, it is possible to set up Ceph Monitors on the same Proxmox node, thus eliminating the need to use a separate node for Monitors. Monitors can also be managed using the Proxmox GUI.

For details on Ceph Monitor, visit `http://ceph.com/docs/master/man/8/ceph-mon/`.

# OSD

**Object Storage Daemon** (**OSD**) is an actual storage media or partition within media such as HDD/SSD that stores the actual cluster data. OSDs are responsible for data replication, recovery, and rebalancing. Each OSD provides monitored information to Ceph Monitors to check for a heartbeat. A Ceph cluster requires a minimum of two OSDs to be in the **active+clean** state. A Ceph cluster regularly provides feedback on a cluster status. An active+clean state expresses an error- or warning-free cluster. See the *Placement Group* section for the other states that a Ceph cluster can achieve. As of Proxmox Version 3.2, OSDs can be managed through the Proxmox GUI.

# OSD Journal

In Ceph, I/O writes are first written to an OSD Journal before they is transferred to an actual OSD. Journals are simply smaller partitions that accept smaller data at a time while the backend OSDs catch up with the writings. By putting Journals on faster access disk drives such as SSD, we can increase a Ceph operation significantly because user data is written to a Journal at a higher speed. At the same time, the Journal sends short bursts of data to OSDs, giving them time to catch up. Journals for multiple OSDs can be stored in one SSD per node. Alternatively, OSDs can be divided into multiple SSDs. For a small cluster of up to eight OSDs per node, using an SSD improves performance. However, while working with a larger cluster with a higher number OSDs per node, collocating the Journal with the same OSDs, rather than SSD, increases performance. The combined write speed for all the OSDs outperforms the speed of one or two SSDs as a Journal.

The important thing about Journal to remember is that the loss of a Journal partition causes OSD data loss. Thus, when using a single SSD to store all OSD Journals, we recommend that you add the SSD to the RAID configuration.

Note that as of the Ceph release codenamed Firefly, Journal is not needed any more. However, since these changes are too new, we recommend that you wait for several months before implementing the new release of Ceph in a production environment. By this time, the Ceph developer will have taken care of any major known or unknown bug in the new release. In this chapter, we are going to create a Ceph cluster using Journal. The installation process is the same as that for Ceph Firefly.

# MDS

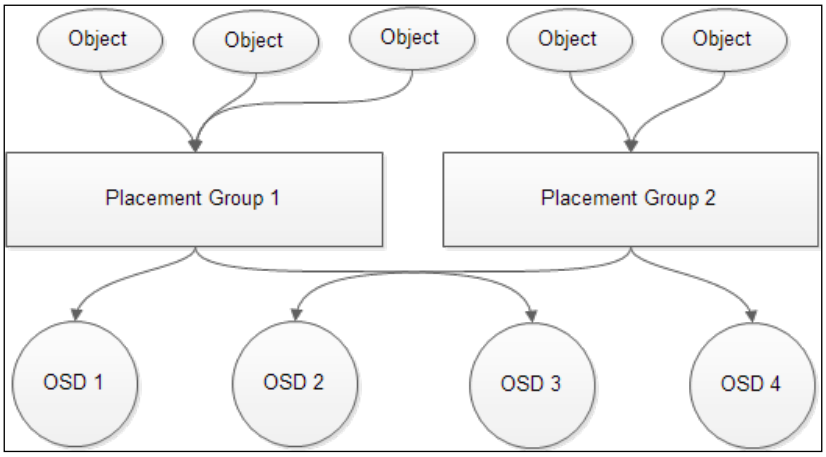
MDS stores Meta information for Ceph FS. Ceph Block and Object Storage do not use MDS. So in a cluster, if Block and Object are the only types being used, it is not necessary to set up an MDS server. Like Monitors, MDS needs to be set up on a different machine to achieve high performance. As of Proxmox Version 3.2, MDS cannot be managed or created from the Proxmox GUI.

Ceph FS is not fully standardized yet and is still in the development phase. It should not be used to store mission critical data. It is mostly stable, but unforeseen bugs may still cause major issues such as data loss. Note that there have not been many reports of mass data loss due to unstable Ceph FS. Two of the virtual machines used to write this book have been running for more than 11 months without any issue.

There should be two MDS nodes in a cluster to provide redundancy because the loss of an MDS node will cause the loss of data on Ceph FS and will render it inaccessible. Two MDS nodes will act as active+passive when one node failure is taken over by another node and vice versa. To learn about MDS and Ceph FS, visit `http://ceph.com/docs/master/cephfs/`.

# Placement Group (PG)

The main function of Placement Group (PG) is to combine several objects into a group and then map the group to several OSDs. A per-group mechanism is much more efficient than a per-object mechanism since the former uses fewer resources. When data is retrieved, it is far more efficient to call a group than an individual object in a group. The following diagram shows how PGs are related to OSD:



For better efficiency, we recommend a total of 50-100 PGs per OSD for all pools. Each PG will consume a certain number of node resources, such as CPUs and memory. A balanced distribution of PGs ensures that all nodes and OSDs in the nodes are not out of memory or that the CPU does not face overload issues. A simple formula to follow while allocating PG for a pool is as follows:

$$\text{Total PGs} = \left(\text{OSD} \times 100\right) / \text{Number of Replicas}$$

The result of the total PG should be rounded up to the nearest power of two. In a Ceph cluster with 3 nodes (replicas) and 24 OSDs, the total PG count should be as follows:

$$\text{Total PGs} = \left(24 \times 100\right) / 3 = 800$$

If we divide 800 by 24, which is the total number of OSDs, then we get 33.33. This is the number of PGs per replica per OSD. Since we have three replicas, we multiply 33.33 with 4 and get 99.99. This is the total number of PGs per OSD in the previous example. The formula will always calculate the PGs per replica. For a three-replica setup, each PG is written thrice. Thus, we multiplied 33.33 by 3 to get the total number of PGs per OSD.

Let's see another example to calculate PG. The following setup has 150 OSDs, three Ceph nodes, and two replicas:

$$\text{Total PGs} = \left(150 \times 100\right) / 2 = 7500$$

If we divide 7500 by 150, the number of total OSDs, we get 50. Since we have two replicas, we multiply 50 by 2 and get 100. Thus, each OSD in this cluster can store 100 PGs. In both examples, our total PG per OSD was within the 50-100 recommended range. Always round up the PG value to remove any decimal point.

For in-depth details on Ceph PG, visit `http://ceph.com/docs/master/rados/operations/placement-groups/`.

# Pool

Pool is like a logical partition where Ceph stores data. When we set a PG or the number of replicas, we actually set them for each pool. When creating a Ceph cluster, three pools are created by default: data, metadata, and RBD. Data and metadata pools are used by the Ceph cluster, while the pool RBD is available to store actual user data. PGs are set on a per-pool basis. The formula discussed in the *Placement Group (PG)* section calculates the PGs required per pool. Thus, when creating multiple pools, it is important to slightly modify the formula so that the total PG stays within 50-100 per OSD.

For instance, in the example of 150 OSDs, three Ceph nodes, and two replicas, our PG was 7500 per pool. This gave us 50 PGs per OSD. If we had three pools in that setup and each pool had 7500 PGs, then the total number of PGs would have been 150 per OSD. To balance the PGs across the cluster, divide 7500 by 3 for three pools and set a PG of 2500 for each pool. This gives us 2500/150 OSDs = 16 PGs per pool per OSD or 16 x 3 pools = 48 total PGs per OSD. Since we have two replicas in this setup, the final total PGs per OSD will be 48 x 2 replicas = 96 PGs. This is within the recommended 50-100 range of PGs per OSD.

Here is another example scenario. Try to see if you can come up with the final PG value on your own. The answer is given after the scenario. The calculation for this scenario is given at the end of this chapter.

Question: The Ceph cluster has five nodes, 120 OSDs, three replicas, and four pools. What is the PG value per pool? What is the total number of PGs per OSD for each replica? What is the total PGs per OSD for all pools and replicas?

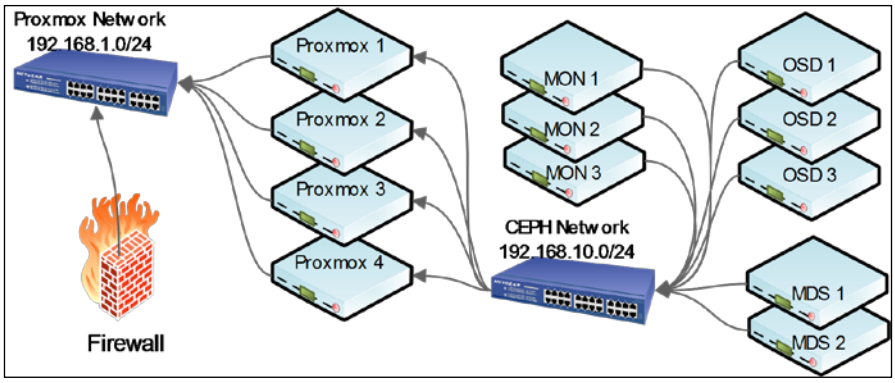
Answer: Total PG for each pool is 1000. Each replica of pools will have 32 PGs per OSD. The total PGs per OSD for all pools and replicas is 96.

# Ceph components summary

If we want to understand the relationship between all the Ceph components we have seen so far, think of it this way: each Pool comprises multiple PGs. Each PG comprises multiple OSDs. The OSD map keeps track of the number of OSDs in the cluster and in the nodes they are in. The MON map keeps track of the number of Monitors in a cluster to form a Quorum and maintains a master copy of the cluster map. A CRUSH map dictates how much data needs to be written to an OSD and how to write or read it. These are the building blocks of a Ceph cluster.

# The Ceph cluster

The following diagram is a basic representation of Proxmox and the Ceph cluster. Note that both clusters are on a separate subnet on separate switches.




A Ceph cluster should be set up with a separate subnet on a separate switch to keep it isolated from the Proxmox public subnet. The advantage of this practice is to keep Ceph's internal traffic isolated so that it does not interfere with the traffic of running virtual machines. On a healthy Ceph cluster with the active+clean state, this is not a big issue. However, when Ceph goes into self-healing mode due to OSD or a node failure, it rebalances itself by moving PGs around, which causes very high bandwidth consumption. On a bad day, separating two clusters ensures that the cluster does not slow down significantly due to the shortage of network bandwidth.

This also provides added security since the Ceph cluster network is completely hidden from any public access using a separate switch. In our previous example, we have three MONs, two MDSs, and three OSD nodes connected to a dedicated switch used only for the Ceph cluster. The Proxmox cluster connects to the Ceph cluster by creating a storage connection using the Proxmox GUI.

# Hardware requirements

Now, let's look at what hardware we are going to need to start setting up our first Ceph cluster. To build our learning cluster, we are going to use basic off-the-shelf hardware for our Ceph nodes. Note that this is only a guideline. You can use any hardware available or set up Monitors and MDSs as virtual machines and then use a physical node for OSDs. The entire Ceph cluster in this chapter can be also be set up in a virtual environment for learning purpose.

The following table shows the hardware list for a basic-level Ceph cluster:

| Node type | Components | Quantity |
|---|---|---|
| OSD = 2 nodes | Motherboard: MSI H81M-P33 LGA1150 | 2 |
| | RAM: 4 GB DDR3 | 2 |
| | CPU: Intel i3-4130 3.40 GHz | 2 |
| | HDD: Seagate 2TB 7200 3.5" | 4 |
| | SSD for OS: Corsair Force LS | 2 |
| | NIC: Intel Pro/1000 CT Gigabit Adapter | 2 |
| | Power supply: 300 Watt | 2 |
| | Chassis: In-Win BL-631 | 2 |
| MON = 3 nodes | Motherboard: MSI FM2-A55M-E33 | 5 |
| MDS = 2 nodes | RAM: 1 GB DDR3 | 5 |
| | CPU: AMD A4-6300 3.7GHz | 5 |
| | SSD for OS: Corsair Force LS | 5 |
| | NIC: Intel Pro/1000 CT Gigabit Adapter | 5 |
| | Power supply: 220 Watt | 5 |
| | Chassis: In-Win BL-631 | 5 |
| Ceph cluster | Switch: Netgear ProSafe 16 Port Gigabit | 1 |

The cost of this basic setup can be further reduced if existing hardware is reused. For MON and MDS nodes, just about any computer will do.

## Software requirements

For the Ceph cluster, it is best to use Ubuntu Server Edition since Ceph developers use this operating system to test Ceph. Any variant of Debian will also do. Ubuntu Server is stable and simple enough for just about anybody. For our cluster, Ubuntu Server 12.04.4 LTS is the only operating system we need to download and create a CD.

# Installing Ceph using an OS

As of Proxmox 3.2, a Ceph server can directly run on the Proxmox node. We can also manage the Ceph cluster from the Proxmox GUI. Ceph can function just the same whether it is installed as an additional service on Proxmox node or separately on a different node using another Debian-based operating system. In this chapter, we are going to look at both installation processes.

We will try our best to look into all aspects of the Ceph installation in this chapter. However, if we have missed out on something, Ceph's official documentation may be a good start for additional information.

For the official Ceph documentation, visit `http://www.ceph.com/docs/master/start/intro`.

For the official Proxmox documentation to install Ceph on the same Proxmox node, visit `http://pve.proxmox.com/wiki/Ceph_Server`.

The installation process of Ceph in the documentation may be a little overwhelming at first, but repeated study will allow for a better understanding of this great technology. In this chapter, you will find a much simpler process that is broken down into chunks to help you move along the Ceph installation process.

Our first task is to put the computers together and connect them with the switch. We should have two nodes for OSDs, three nodes for Monitors, and two nodes for MDSs. If you are using virtual machines for OSDs and MONs, then you should have two OSD nodes, three virtual machines for MONs, and two for MDSs. Connect the switch to the Internet so that the Ceph cluster can have internet connectivity during the installation. The operating system will need to be updated with the latest packages.

We recommend that you disconnect the Internet connection from the Ceph cluster at all times in a production environment. Allow internet connectivity on a Ceph network only to apply updates or patches. This prevents the chances of a security breach in the Ceph cluster through WAN, should the firewall be compromised.

# Installing and setting up Ubuntu

Install Ubuntu Server on all the nodes as suggested by the Ubuntu documentation (`http://www.ubuntu.com/download/server/install-ubuntu-server`). Once the installation is complete, we are going to set up network interfaces and a hostname on all nodes using the following commands:

```
# sudo nano /etc/network/interfaces
address 192.168.10.11
netmask 255.255.255.0
gateway 192.168.10.254


#if using your own local DNS server, use your IP address instead of
  public DNS server
```

```
dns-nameservers 208.67.222.222


# sudo nano /etc/hostname
ceph-admin-01


# sudo nano /etc/hosts
192.168.10.10    ceph-admin-01.domain.com    ceph-admin-01
```

We are now going to change the interface, hostname, and hosts' configuration for all Ceph nodes based on the following table:

| Hostname | IP address | Hosts |
|---|---|---|
| ceph-mon-01 | 192.168.10.11 | 192.168.10.11 ceph-mon-01.domain.com ceph-mon-01 |
| ceph-mon-02 | 192.168.10.12 | 192.168.10.12 ceph-mon-02.domain.com ceph-mon-02 |
| ceph-mon-03 | 192.168.10.13 | 192.168.10.13 ceph-mon-03.domain.com ceph-mon-03 |
| ceph-mds-01 | 192.168.10.16 | 192.168.10.16 ceph-mds-01.domain.com ceph-mds-01 |
| ceph-mds-02 | 192.168.10.17 | 192.168.10.17 ceph-mds-02.domain.com ceph-mds-02 |
| ceph-osd-01 | 192.168.10.21 | 192.168.10.21 ceph-osd-01.domain.com ceph-osd-01 |
| ceph-osd-02 | 192.168.10.22 | 192.168.10.22 ceph-osd-02.domain.com ceph-osd-02 |

We are going to use Monitor 1 as the admin node for the Ceph cluster. We are going to add all other Ceph hosts into the admin node's host file. Ceph requires a hostname rather than IP addresses to communicate with each other. We could also set up a local DNS server to handle the hostname resolution in a large Ceph cluster. We can sue the admin node or set up another node or virtual machine as a local DNS server. Perform the following steps to set up a local DNS server using Ubuntu server:

1. Install DNS server software in Ubuntu using the following command:
   ```
   # sudo apt-get install bind9
   ```
2. Configure the DNS server to cache requests and forward unknown requests to another DNS server using the following command:
   ```
   # sudo nano /etc/bind/named.conf.options
   ```
3. Uncomment or add forwarder section, followed by the IP address of any public DNS server, such as OpenDNS or Google:
   ```
   forwarders  {
     208.67.222.222;
     8.8.8.8;
   };
   ```

4. Define zones for the local domain using the following command:

   **`# sudo nano /etc/bind/named.conf.local`**

   Enter the following entries in the `named.conf.local` file:

```
zone "domain.com" IN {
  type master;
  file "/etc/bind/zones/domain.com.db";
};
```

5. We will now add a reverse DNS lookup as follows for the local network in the same `named.conf.local` file:

```
zone "10.168.192.in-addr.arpa" {
  type master;
  file "/etc/bind/zones/rev.10.168.192.in-addr.arpa";
};
```

6. Create a zones directory using the following command:

   **`# sudo mkdir /etc/bind/zones`**

7. Configure the local domain using the following command:

   **`# sudo nano /etc/bind/zones/domain.com.db`**

   Enter the following settings in the `domain.com.db` file. Match the IP addresses and hostnames with your network:

```
; Use semicolons to add comments.
; Host-to-IP Address DNS Pointers for home.lan
; Note: The extra "." at the end of the domain names are
important.

; The following parameters set when DNS records will expire, etc.
; Importantly, the serial number must always be iterated upward to
prevent
; undesirable consequences. A good format to use is YYYYMMDDII
where
; the II index is in case you make more that one change in the
same day.
$ORIGIN .
$TTL 86400      ; 1 day
domain.com. IN SOA ceph-admin-01.domain.com. hostmaster.domain.
com. (
    2014060601 ; serial
    8H ; refresh
    4H ; retry
```

```
    4W ; expire
    1D ; minimum
)

; NS indicates that ubuntu is the name server on home.lan
domain.com. IN NS ceph-admin-01.domain.com.

$ORIGIN domain.com.

; Set the address for localhost.domain.com
localhost    IN A 127.0.0.1

; Set the hostnames
ceph-admin-01 IN A 192.168.10.10
ceph-mon-01  IN A 192.168.10.11
ceph-mon-02  IN A 192.168.10.12
ceph-mon-03  IN A 192.168.10.13
ceph-mds-01  IN A 192.168.10.16
ceph-mds-02  IN A 192.168.10.17
ceph-osd-01  IN A 192.168.10.21
ceph-osd-01  IN A 192.168.10.22
```

8. Configure a reverse lookup using the following command:

```
# sudo nano /bind/zones/rev.10.168.192.in-addr.arpa
```

Enter the following settings in the `rev.10.168.192.in-addr.arpa` file. Match the IP addresses and hostnames with your network:

```
; IP Address-to-Host DNS Pointers for the 192.168.10 subnet
@ IN SOA ceph-admin-01.domain.com. hostmaster.domain.com. (
    2014060601 ; serial
    8H ; refresh
    4H ; retry
    4W ; expire
    1D ; minimum
)
; define the authoritative name server
           IN NS ceph-admin-01.domain.com.

; Set the Hostnames
1      IN PTR ceph-admin-01.domain.com.
2      IN PTR ceph-mon-01.domain.com.
3      IN PTR ceph-mon-02.domain.com.
4      IN PTR ceph-mon-03.domain.com.
```

```
5       IN PTR ceph-mds-01.domain.com.
6       IN PTR ceph-mds-02.domain.com.
7       IN PTR ceph-osd-01.domain.com.
8       IN PTR ceph-osd-02.domain.com.
```

9. Restart the bind services using the following command:

```
# sudo service bind9 restart
```

10. Enter the IP address `192.168.10.10` in the network configuration for all other Ceph nodes.

    If you are not using a dedicated local DNS server, then this is how the admin node hosts file should look to point the hostnames to IP addresses manually:

```
127.0.0.1          localhost.localdomain     localhost
192.168.10.11      ceph-mon-01.domain.com    ceph-mon-01
192.168.10.12      ceph-mon-02.domain.com    ceph-mon-02
192.168.10.13      ceph-mon-03.domain.com    ceph-mon-03
192.168.10.16      ceph-mds-01.domain.com    ceph-mds-01
192.168.10.17      ceph-mds-02.domain.com    ceph-mds-02
192.168.10.21      ceph-osd-01.domain.com    ceph-osd-01
192.168.10.22      ceph-osd-01.domain.com    ceph-osd-02
```

# Creating an admin user

Create an admin user `cephadmin` on all the Ceph nodes and add it to `sudo` file using the following command:

```
# sudo useradd -d /home/cephadmin -m cephadmin
# sudo passwd cephadmin
```

# Assigning SUDO permission to a user

Enter the following command to give the sudo permission to the admin user `cephadmin`:

```
# echo "cephadmin ALL = (root) NOPASSWD:ALL" | sudo tee
  /etc/sudoers.d/cephadmin
# sudo chmod 0440 /etc/sudoers.d/cephadmin
```

Run the previous commands on all the Ceph nodes.

# Updating Ubuntu

Once the network is set up, update Ubuntu and reboot all the nodes using the following commands:

```
# sudo apt-get update
# sudo apt-get dist-upgrade
```

# Generating an SSH Key

Log in to the `ceph-admin-01` node with the new admin user `cephadmin`; this is the user we will use for Ceph management. We will now generate SSH keys and copy it to all the nodes. This will allow a password-less login to all the Ceph nodes. The following command is used to create the SSH key from the admin node:

```
# ssh-keygen
```

Once the SSH key has been created, copy the key to all the Ceph nodes. The following commands need to be executed from the admin node:

```
# ssh-copy-id cephadmin@ceph-mon-01
# ssh-copy-id cephadmin@ceph-mon-02
# ssh-copy-id cephadmin@ceph-mon-03
# ssh-copy-id cephadmin@ceph-mds-01
# ssh-copy-id cephadmin@ceph-mds-02
# ssh-copy-id cephadmin@ceph-osd-01
# ssh-copy-id cephadmin@ceph-osd-02
```

# Installing ceph-deploy

The `ceph-deploy` tool is used to install Ceph on multiple nodes from a single node and many other useful tasks, such as formatting disk drives and creating OSDs from a single admin node. Before we install `ceph-deploy`, add a Ceph repository and download a repository key. Run the following commands on the admin node only:

```
# wget -q -O-
  'https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc'
    | sudo apt-key add -
# echo deb http://ceph.com/debian-firefly/ $(lsb_release -sc) main |
  sudo tee /etc/apt/sources.list.d/ceph.list
# sudo apt-get update
# sudo apt-get install ceph-deploy
```

As of this writing, the codename of Ceph's latest release is Firefly. Before installing `ceph-deploy`, be sure to check Ceph's latest codename. Change the name in the `echo` command accordingly:

```
# echo deb http://ceph.com/debian-<ceph_codename>/ $(lsb_release -sc)
  main | sudo tee /etc/apt/sources.list.d/ceph.list
```

# Creating a Ceph cluster

We are now going to create a cluster configuration file using the `ceph-deploy` command. It is best to store all of the Ceph cluster configuration in one folder for easy management and backup. We will create a new folder in the `ceph-admin-01` node under `/home/cephadmin` and call it `pmxceph`. The command format to create a new cluster is as follows:

```
cephadmin@ceph-admin-01:# ceph-deploy new <monitor_nodes>
```

By default, Ceph creates a cluster with the name `ceph`. If you want a different name for your cluster, then the command format will be the following:

```
# ceph-deploy  --cluster <new_name> new <monitor_nodes>
```

The option to name a cluster is useful when setting up multiple Ceph clusters on the same hardware. For example, separate clusters can coexist on the same hardware for SSD and HDD pools to create a multi-tiered storage. The commands to do so are as follows:

```
cephadmin@ceph-admin-01:# cd /home/cephadmin
cephadmin@ceph-admin-01:# mkdir pmxceph
cephadmin@ceph-admin-01:# cd pmxceph
cephadmin@ceph-admin-01:/pmxceph# ceph-deploy new ceph-mon-01 ceph-
  mon-02 ceph-mon-03
```

At this point, Ceph will create the following three files inside the `pmxceph` folder:

- Ceph cluster configuration file `ceph.conf`
- Ceph cluster creation logfile `ceph.log`
- Ceph monitor's keyring file `ceph.mon.keyring`

The following is the content of each file created during the Ceph cluster setup:

- Ceph cluster configuration file `ceph.conf`:

```
[global]
fsid = 467cefbe-19a6-40y7-a322-5d217ae89fb3
mon_initial_members = ceph-mon-01, ceph-mon-02, ceph-mon-03
```

```
mon_host = 192.168.10.11,192.168.10.12,192.168.10.13
auth_cluster_required = cephx
auth_service_required = cephx
auth_client_required = cephx
filestore_xattr_use_omap = true
```

- Ceph cluster creation log file `ceph.log`:

```
[ceph_deploy.cli][INFO  ] Invoked (1.4.0): /usr/bin/ceph-
  deploy new ceph-mon-01 ceph-mon-02 ceph-mon-03
[ceph_deploy.new][DEBUG ] Creating new cluster named ceph
[ceph_deploy.new][DEBUG ] Resolving host ceph-mon-01
[ceph_deploy.new][DEBUG ] Monitor ceph-mon-01 at
  192.168.10.11
[ceph_deploy.new][INFO  ] making sure passwordless SSH
  succeeds
[ceph_deploy.new][DEBUG ] Resolving host ceph-mon-02
[ceph_deploy.new][DEBUG ] Monitor ceph-mon-02 at
  192.168.10.12
[ceph_deploy.new][INFO  ] making sure passwordless SSH
  succeeds
[ceph-mon-02][DEBUG ] connected to host: ceph-mon-01
[ceph-mon-02][INFO  ] Running command: ssh -CT -o
  BatchMode=yes ceph-mon-02
[ceph_deploy.new][DEBUG ] Resolving host ceph-mon-03
[ceph_deploy.new][DEBUG ] Monitor ceph-mon-03 at
  192.168.10.13
[ceph_deploy.new][INFO  ] making sure passwordless SSH
  succeeds
[ceph-mon-03][DEBUG ] connected to host: ceph-mon-01
[ceph-mon-03][INFO  ] Running command: ssh -CT -o
  BatchMode=yes ceph-mon-03
[ceph_deploy.new][DEBUG ] Monitor initial members are
  ['ceph-mon-01', 'ceph-mon-02', 'ceph-mon-03']
[ceph_deploy.new][DEBUG ] Monitor addrs are
  ['192.168.10.11', '192.168.10.12', '192.168.10.13']
[ceph_deploy.new][DEBUG ] Creating a random mon key...
[ceph_deploy.new][DEBUG ] Writing initial config to
  ceph.conf...
[ceph_deploy.new][DEBUG ] Writing monitor keyring to
  ceph.mon.keyring...
```

- Ceph cluster monitors keyring file `ceph.mon.keyring`:

```
[mon.]
key = AQDgpklTAGCRTBAAZIIHqPYn0kVNmduOjB9sdA==
caps mon = allow *
```

Complete details of the Ceph configuration or other files are beyond the scope of this book. A detailed documentation is available at `http://ceph.com/docs/master/`.

# Installing Ceph on nodes

After the admin node is set up, we are now going to install Ceph on all the other nodes using a simple command as follows:

```
# ceph-deploy install <node_name>
```

Run the previous command from the admin node. Simply change the appropriate node name in the command. For our cluster, we are using one of the `ceph-mon-01` monitor nodes as the admin and monitor. The initial command will look as follows:

```
# ceph-deploy install ceph-mon-01
```

# Creating Monitors (MONs)

In our cluster, we have three monitor nodes. We are now going to create the actual Monitor daemon on all these three nodes. Simply run the following command with appropriate node name:

```
# ceph-deploy mon create ceph-mon-01
```

# Gathering the admin keys

To run the Ceph admin commands for cluster management, we must have all the Ceph admin keys in a folder. After creating MONs, we can simply use the following command to gather admin keys from any one of the MON nodes:

```
# ceph-deploy gatherkeys ceph-mon-01
```

All together, we should now have seven files in the `pmxceph` cluster folder:

- `ceph.bootstrap-mds.keyring`
- `ceph.bootstrap-osd.keyring`
- `ceph.client.admin.keyring`
- `ceph.mon.keyring`
- `ceph.conf`
- `ceph.log`
- `release.asc`

The Ceph cluster creates a default user named `admin` for cluster management during the cluster creation process. The user admin uses `ceph.client.admin.keyring` for all administrative tasks in a Ceph cluster.

Let's check our progress so far using the keys we just gathered. Simply run the following command to check the cluster status:

```
# sudo ceph -s
```

If all went ok this far, the previous command should print the following result on the screen:

```
cephadmin@ceph-mon-01:~/pmxceph$ sudo ceph -s
    cluster 472cefbe-11a6-40f7-a352-5d245ae89fb3
     health HEALTH_ERR 192 pgs stuck inactive; 192 pgs stuck unclean;
       no osds
     monmap e2: 3 mons at {ceph-mon-01=192.168.10.11:6789/0,ceph-mon-
       02=192.168.10.12:6789/0,ceph-mon-03=192.168.10.13:6789/0},
         election epoch 6, quorum 0,1,2 ceph-mon-01,ceph-mon-02,ceph-
           mon-03
     osdmap e1: 0 osds: 0 up, 0 in
      pgmap v2: 192 pgs, 3 pools, 0 bytes data, 0 objects
            0 kB used, 0 kB / 0 kB avail
                 192 creating
```

According to this status, we have successfully created our MONs. By default, Ceph creates three pools with 192 PGs. PGs are in creating mode because we have not added any OSD yet.

# Creating OSDs

With the initial cluster and MONs setup, we are now ready to start adding OSDs in the Ceph cluster. For simplicity, we are going to put Journal on the same OSD. Without journaling, an OSD cannot function. Any data coming to the Ceph cluster is first written into the journal device and then transferred to OSD. Thus, performance can be increased greatly by putting Journal on SSD while using HDD for OSD.

For a small Ceph cluster, putting Journal on SSD is acceptable. When the cluster grows beyond eight OSDs per physical node, it is wise to put Journal on the same HDD. A lost journal due to SSD failure will cause mass data loss for any OSD that stores its journal on that SSD. By putting Journal on the same HDD, we prevent the single point of failure. To see the difference in performance for OSSD on SSD and HDD, see the section *Ceph benchmarking* section later in this chapter.

Before we create OSDs, we need to know the drives available in the nodes and the drive names. The following command format will give us a list of drives for a node in the cluster:

```
# ceph-deploy disk list <node_name>
```

Using the following command format we can get a disk list from the Ceph OSD node:

```
cephadmin@ceph-mon-01:~/pmxceph$ ceph-deploy disk list ceph-osd-01
```

The following is the information displayed after the `disk list` command is executed:

```
[ceph_deploy.cli][INFO  ] Invoked (1.4.0): /usr/bin/ceph-deploy disk
  list ceph-osd-01
[ceph-osd-01][DEBUG ] connected to host: ceph-osd-01
[ceph-osd-01][DEBUG ] detect platform information from remote host
[ceph-osd-01][DEBUG ] detect machine type
[ceph_deploy.osd][INFO  ] Distro info: Ubuntu 12.04 precise
[ceph_deploy.osd][DEBUG ] Listing disks on ceph-osd-01...
[ceph-osd-01][INFO  ] Running command: sudo ceph-disk list
[ceph-osd-01][DEBUG ] /dev/sda :
[ceph-osd-01][DEBUG ]  /dev/sda1 other, ext2, mounted on /boot
[ceph-osd-01][DEBUG ]  /dev/sda2 other
[ceph-osd-01][DEBUG ]  /dev/sda5 other, LVM2_member
[ceph-osd-01][DEBUG ] /dev/vda other, unknown
[ceph-osd-01][DEBUG ] /dev/vdb other, unknown
```

We can see from the previous information that the drives `/dev/vda` and `/dev/vdb` are the HDDs we are going to use to create OSDs on the `ceph-osd-01` node.

First, we need to format the drive using the following commands:

```
# ceph-deploy disk zap ceph-osd-01:/dev/vda
# ceph-deploy disk zap ceph-osd-01:/dev/vdb
```

This will erase all the data on the drives and have them ready to accept the OSD daemon. Now we will create the OSD with a single command per OSD. For our four OSDs in this cluster, we are going to run the following four commands:

```
# ceph-deploy osd create ceph-osd-01:/dev/vda
# ceph-deploy osd create ceph-osd-01:/dev/vdb
# ceph-deploy osd create ceph-osd-02:/dev/vda
# ceph-deploy osd create ceph-osd-02:/dev/vdb
```

As soon as the OSDs are created, the Ceph cluster will achieve the active+clean status for the PGs. We can verify the status of the cluster with the following command:

```
# sudo ceph -s
cluster 472cefbe-11a6-40f7-a352-5d245ae89fb3
     health HEALTH_OK
     monmap e2: 3 mons at {ceph-mon-01=192.168.10.11:6789/0,ceph-mon-
       02=192.168.10.12:6789/0,ceph-mon-03=192.168.10.13:6789/0},
         election epoch 6, quorum 0,1,2 ceph-mon-01,ceph-mon-02,ceph-
           mon-03
     osdmap e17: 4 osds: 4 up, 4 in
      pgmap v30: 192 pgs, 3 pools, 0 bytes data, 0 objects
            142 MB used, 299 GB / 299 GB avail
                 192 active+clean
```

We can see the OSD placements and their status within the cluster from the following command:

```
# sudo ceph osd tree
# id    weight    type name      up/down      reweight
-1      0.28      root default
-2      0.14              host ceph-osd-01
0       0.06999                   osd.0        up      1
2       0.06999                   osd.2        up      1
-3      0.14              host ceph-osd-02
1       0.06999                   osd.1        up      1
3       0.06999                   osd.3        up      1
```

# Connecting Proxmox to a Ceph cluster

With the Ceph cluster creation complete, we are now going to connect the Ceph RBD storage with Proxmox so that we can start storing our VMs on it. Proxmox will not be able to talk to the Ceph cluster without the admin keyring `ceph.client.admin.keyring`. We can copy the admin keyring and connect Ceph with Proxmox by copying the admin keyring to `/etc/pve/priv/ceph` on the Proxmox node from the Ceph admin node.

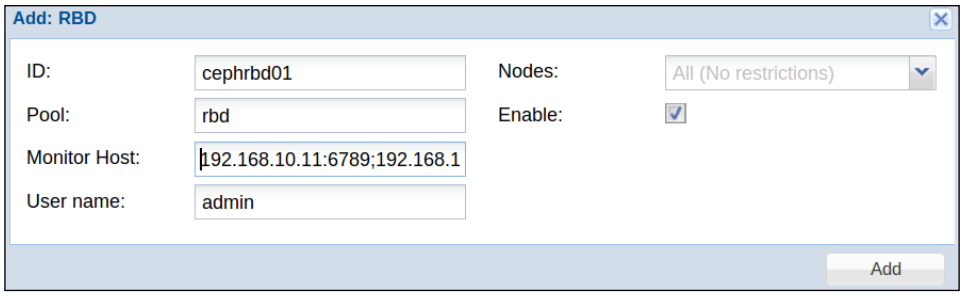
Run the following commands on any Proxmox node:

```
# mkdir /etc/pve/priv/ceph
# cd /etc/pve/priv/ceph# scp <ceph-
  admin>:/etc/ceph/ceph.client.admin.keyring
    /etc/pve/priv/ceph/<storageID>.keyring
```

Note that `storageID` is the name of the storage we are going to create through the Proxmox GUI. We are going to use `cephrbd01` as the Proxmox RBD storage name. Our command will be as follows:

```
# scp ceph-admin-01:/home/cephadmin/pmxceph
  /ceph.client.admin.keyring
    /etc/pve/priv/ceph/cephrbd01.keyring
```

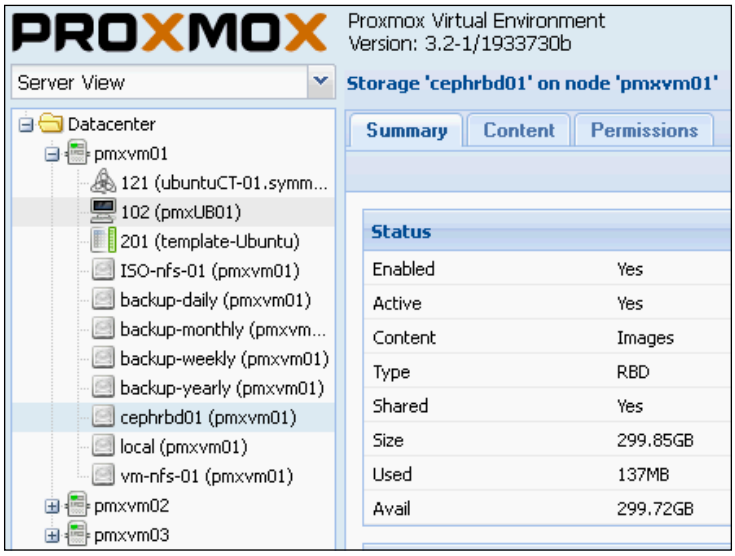
We can now proceed to the Proxmox GUI to start creating the Ceph storage. Simply go to **Datacenter** | **Storage** | **Add** on the Proxmox GUI. Fill in the boxes with the data as shown in the following screenshot:




One of the pools Ceph creates during initial setup is **rbd**. Always enter the IP address and port number in the **Monitor Host** field. All Monitor IP addresses and ports entered should be separated by a semicolon (;). This way, if one monitor node fails, another monitor will take over. FQDN of a host will not be accepted.

The default port number in the Ceph cluster is 6789. If you are creating multiple Ceph clusters on the same hardware, be sure to use a different port number for the cluster. For example, tier 1 storage with SSD as OSD can be on port 6789, while tier 2 storage HDD as OSD can be on port 6790.

If all went successful thus far, the newly created storage should look like the following screenshot from the Proxmox GUI:




The following snippet is how the `storage.cfg` file should look for the `rbd` portion:

```
rbd: cephrbd01
        monhost 192.168.10.11:6789
        pool rbd
        content images
        username admin
```

We can now start storing our virtual machines on the new Ceph RBD storage. Keep in mind that the Ceph RBD storage can only hold KVM virtual machine images at this moment. We cannot store other files such as OpenVZ containers, backups, templates, and ISO files in this type of storage. In order to store these file types, we need Ceph FS.

# Installing Ceph on Proxmox

As of Proxmox Version 3.2, it is now possible to install Ceph on the same Proxmox node, thus reducing the number of separate Ceph nodes needed, such as the admin node, Monitor node, or OSD node. Proxmox also provides GUI features that we can use to view the Ceph cluster and manage OSDs, MONs, pools, and so on. In this section, we are going to see how we can install Ceph on the Proxmox node. As of Version 3.2, MDS server and CRUSH map management is not possible from the Proxmox GUI.

# Preparing a Proxmox node for Ceph

Since we are installing Ceph on the same Proxmox node, we are going to set up the network interfaces for a separate network for Ceph traffic only. We will set up three Proxmox nodes—pmxvm01, pmxvm02, and pmxvm03—with Ceph. On all three nodes, we are going to add the following interfaces section in `/etc/network/interfaces`. You can also use any IP address that suits your network environment.

Run the following command from the Proxmox node `pmxvm01`:

```
root@pmxvm01:/# nano /etc/network/interfaces
```

Enter the following section to add the second network:

```
auto eth2
iface eth2 inet static
    address 192.168.10.1
    netmask 255.255.255.0
```

Run the following command to make the new interface active:

```
root@pmxvm01:/# ifup eth2
```

Run the following command from the Proxmox node `pmxvm02`:

```
root@pmxvm02:/# nano /etc/network/interfaces
```

Enter the following section to add the second network:

```
auto eth2
iface eth2 inet static
    address 192.168.10.2
    netmask 255.255.255.0
```

Run the following command to make the new interface active:

```
root@pmxvm02:/# ifup eth2
```

Run the following command from the Proxmox node `pmxvm03`:

```
root@pmxvm03:/# nano /etc/network/interfaces
```

Enter the following section to add the second network:

```
auto eth2
iface eth2 inet static
    address 192.168.10.3
    netmask 255.255.255.0
```

Run the following command to make the new interface active:

```
root@pmxvm03:/# ifup eth2
```

# Installing Ceph

Proxmox added a small command-line utility called `pveceph` to perform various Ceph-related tasks. The following table lists the commands and the tasks that can be performed using `pveceph`:

| Command | Task performed |
|---|---|
| `pveceph createmon` | This creates Ceph Monitors and must be run from the node to become a Monitor. |
| `pveceph createpool <name>` | This creates a new pool and can be used from any node. |
| `pveceph destroymon <mon_id>` | This removes Monitor. |
| `pveceph destroypool <name>` | This removes the Ceph pool. |
| `pveceph init --network <x.x.x.0/x>` | This creates the initial Ceph configuration file based on the network CIDR used. |
| `pveceph start <service>` | This starts Ceph daemon services such as MON, OSD, and MDS. |
| `pveceph stop <service>` | This stops Ceph daemon services such as MON, OSD, and MDS. |
| `pveceph status` | This shows cluster, Monitor, MDS, OSD status, and cluster ID. |
| `pveceph createosd </dev/X>` | This creates OSD daemons. |
| `pvecph destroyosd <osdid>` | This removes OSD daemons. |
| `pveceph install` | This installs Ceph on the Proxmox node. |
| `pveceph purge` | This removes Ceph and all Ceph-related data from the node the command is running from. |

Ceph must be installed and at least one Monitor must be created using a command line initially before managing it through the Proxmox GUI. We can perform the following steps to install Ceph on the Proxmox nodes:

1. First, we need to install the Ceph software on the Proxmox node using the following command. Run this command on all Proxmox nodes that will be part of the Ceph cluster:

```
root@pmxvm01:/# pveceph install -version emperor
root@pmxvm02:/# pveceph install -version emperor
root@pmxvm02:/# pveceph install -version emperor
```

Please note that the latest version of Ceph is codenamed Firefly, released with major changes such as the omission of Journal for OSD. You should wait several months before using Firefly in the production environment so that known or unknown bugs can be sorted out. There isn't enough usage data yet to be sure about the stability of Ceph's Firefly. If you intend to use Firefly, simply change the version name from `emperor` to `firefly`. Ceph's Emperor was the final stable release before Firefly. Run the following commands:

```
root@pmxvm01:/# pveceph install -version firefly
root@pmxvm02:/# pveceph install -version firefly
root@pmxvm02:/# pveceph install -version firefly
```

2. We are going to create an initial Ceph configuration file on the first Proxmox node. Run the following command once from one node:

```
root@pmxvm01:/# pveceph init --network 192.168.10.0/24
```

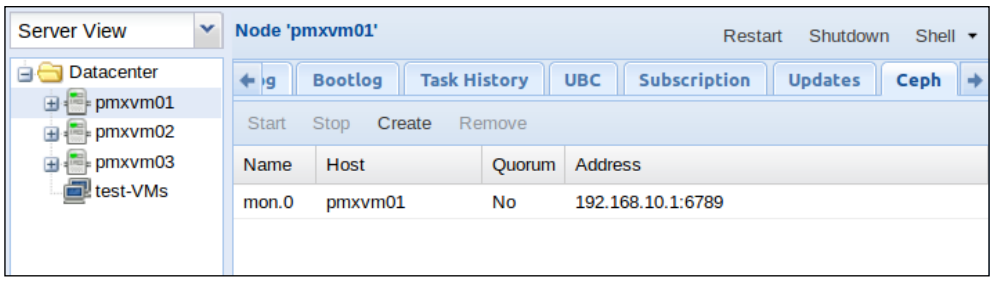
3. We will now create the first Ceph Monitor using the following command on the same node, where we just created the initial configuration file:

```
root@pmxvm01:/# pveceph createmon
```

After these steps, we can proceed with the Proxmox GUI to create more MONs, OSDs, or Pools.

# Creating MON from the Proxmox GUI

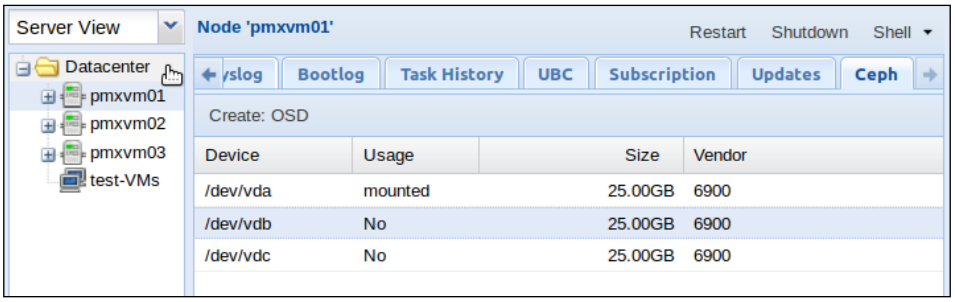
To view and create Monitors from the Proxmox GUI, navigate to **Datacenter** | **pmxvm01** | **Ceph** | **Monitor**. The following screenshot shows the first monitor we created using CLI:



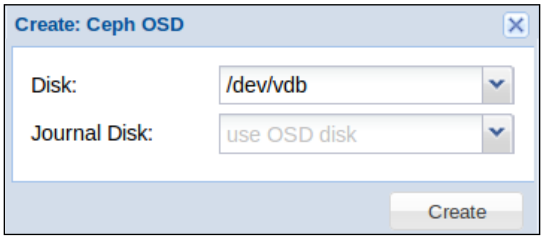
Click on **Create** to open the Monitor creation dialog box, shown in the following screenshot. Select a Proxmox node from the drop-down list and click on the **Create** button to initiate the Monitor creation.


# Creating OSD from the Proxmox GUI

To view the installed disk drives in the node and create OSDs from the Proxmox GUI, navigate to **Datacenter** | **pmxvm01** | **Ceph** | **Disk**. Select an available disk drive to create OSD. The following screenshot shows two available disk drives, `/dev/vdb` and `/dev/vdc`:

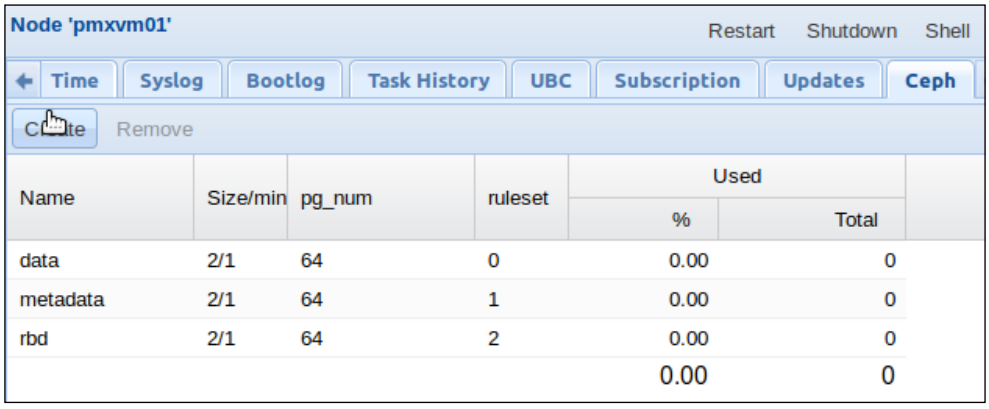



To create an OSD, click on the **Create: OSD** button to open a dialog box as shown in the following screenshot. Select an available disk drive from the drop-down list and click on the **Create** button. Click on the **Journal Disk** drop-down button to select a different disk drive to store the OSD journal.

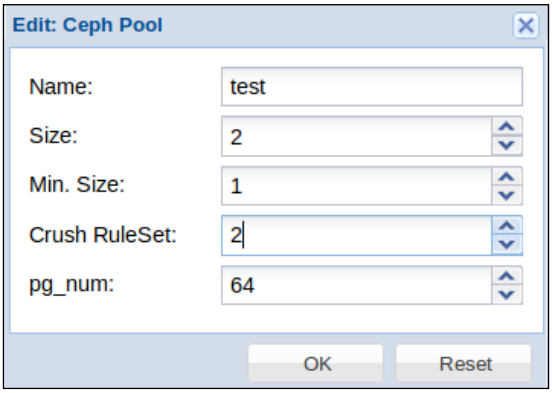


# Creating a new Ceph pool using the Proxmox GUI

To create a new pool using the Proxmox GUI, go to **Datacenter** | **<node>** | **Ceph** | **Pools**. The pool interface shows information about existing pools, such as name, replica number, PG number, and per pool percentage used, as shown in the following screenshot:




To create a new pool, click on the **Create** button to open the pool creation dialog box as shown in the following screenshot. Enter a name for the pool in the **Name** field, number of replica in **Size**, leave **Min. Size** to **1**, change **Crush Ruleset** to **2** or any other ruleset you wish to use, and enter the proper PG number based on the PG calculation formula shown in the *Placement Group (PG)* section. Click on **OK** to start the pool's creation.



# Creating a Ceph FS

Ceph FS requires MDS to function. We have already put together two physical nodes to set up MDS. We are now going to set up an MDS daemon on them to set up Ceph FS. Note that we do not recommend Ceph FS in a production environment yet to store mission-critical virtual machines or other data.

# Setting up an MDS daemon

The following steps can be performed from the admin mode to install Ceph on MDS nodes and set up MDS daemons:

1. Install Ceph on the MDS node using the following commands:

   ```
   # ceph-deploy install ceph-mds-01
   # ceph-deploy install ceph-mds-02
   ```

2. Set up the MDS daemon on an MDS nodes using the following commands:

   ```
   # ceph-deploy mds create <ceph_mds_01>
   # ceph-deploy mds create <ceph_mds_02>
   ```

3. Now let's check the status of the cluster with the MDS nodes:

   ```
   cephadmin@ceph-admin-01:/pmxceph# sudo ceph –s
   ```

We should have the following output on our screen:

```
    cluster 472cefbe-11a6-40f7-a352-5d245ae89fb3
     health HEALTH_OK
     monmap e2: 3 mons at {ceph-mon-01=192.168.10.11:6789/0,ceph-mon-
       02=192.168.10.12:6789/0,ceph-mon-03=192.168.10.13:6789/0},
         election epoch 6, quorum 0,1,2 ceph-mon-01,ceph-mon-02,ceph-
           mon-03
     mdsmap e5: 1/1/1 up {0=ceph-mds-01=up:active}, 1 up:standby
     osdmap e17: 4 osds: 4 up, 4 in
      pgmap v37: 192 pgs, 3 pools, 9470 bytes data, 21 objects
            138 MB used, 299 GB / 299 GB avail
                 192 active+clean
  client io 0 B/s rd, 10837 B/s wr, 13 op/s
```

From the previous status, we can see a new line has been added for the mdsmap and it shows one MDS node is active while the other one is standing by. Whenever one of the MDS nodes fails, the other one will pick up automatically and almost instantly.

# Setting up Ceph FS using FUSE

Unlike the RBD storage, Ceph FS works with Proxmox with mount points. That is, Ceph FS needs to be locally mounted on the Proxmox nodes, and then attach the storage as a local directory to be used with Proxmox. We have to mount the Ceph FS on a directory under `/mnt` and then connect it to Proxmox through the local directory. Let's create a directory on Proxmox to mount Ceph FS using the following command:

```
# mkdir /mnt/cephfs01
```

Install `ceph-fuse` using the following command on the Proxmox node:

```
# sudo apt-get install ceph-fuse
```

# Mounting Ceph FS

Once we have created a directory and `ceph-fuse` has been installed, the only thing remaining is to run the following command from a Proxmox node to mount Ceph FS:

```
# ceph-fuse -k <ceph_keyring> -m <mon_host:6789> /mnt/<folder>
# ceph-fuse -k /etc/pve/priv/ceph/ceph.client.admin.keyring -m
  192.168.10.11:6789 /mnt/cephfs01
```

Note that since Ceph FS is mounted locally on a Proxmox node, the storage will be only available from that particular Proxmox node. In order to access Ceph FS from all the nodes, we have to locally mount Ceph FS on all the Proxmox nodes in the cluster.

When trying to mount Ceph FS on the other nodes after it has already been mounted and some data has been stored, you may see following error messages:

```
ceph-fuse[756903]: starting ceph client
fuse: mountpoint is not empty
fuse: if you are sure this is safe, use the 'nonempty' mount option
ceph-fuse[756903]: fuse failed to initialize
2014-04-13 11:03:31.571116 7f8752aa0760 -1
  fuse_mount(mountpoint=/mnt/cephfs01) failed.
ceph-fuse[756901]: mount failed: (5) Input/output error
```

This is not a bug and it only means that the Ceph FS is not empty. This is a usual case after mounting on a node for the first time. Simply mount it again using the following command format, which includes the `-o nonempty` option:

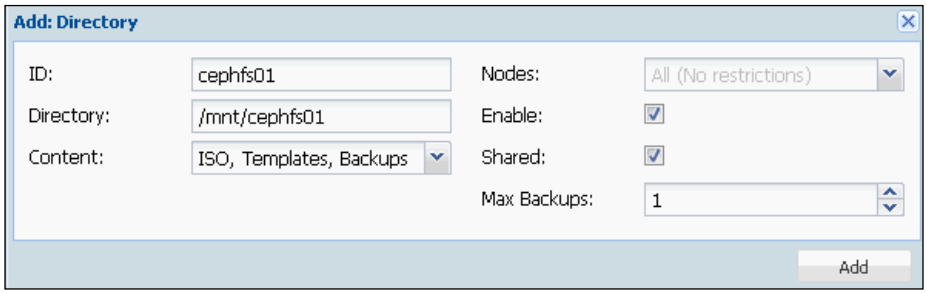
```
# ceph-fuse -k <ceph_keyring> -m <mon_host:6789> /mnt/<folder> -o
  nonempty
```

To mount Ceph FS during a Proxmox node boot, add the following line in `/etc/fstab`:

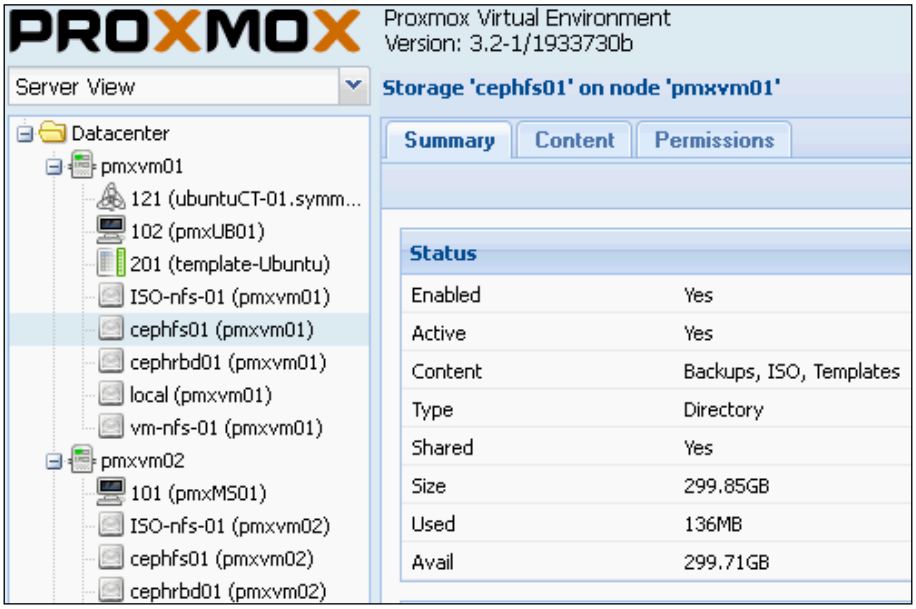
```
id=admin    /mnt/cephfs01    fuse.ceph    defaults    0    0
```

# Connecting Proxmox to Ceph FS

After the mounting is done, we now have to connect Proxmox to the Ceph FS. Simply go to **Datacenter** | **Storage** | **Add** | **Directory** and fill in the boxes with the information shown in the following screenshot:




We can check if the connection is successful through the storage status shown in the following screenshot:



We can store any files, including ISO, templates, backups, and OpenVZ containers, in this storage. Since both RBD and Ceph FS are on the same hardware, we do not recommend that you store backups primarily on Ceph FS. Backup should always be done on separate hardware. The best course of action would be setting up a separate Ceph cluster with Ceph FS only and storing Proxmox backups there. However, for a light load, such as storing ISOs and templates, Ceph FS works great. This eliminates the need to have separate file storage such as FreeNAS or any other type of storage to store ISO images, templates, or OpenVZ containers.

Ceph FS has not yet reached industry quality according to the Ceph developers. Minor occasional glitches are to be expected. But from my personal experience, even after using Ceph FS for 11 months straight on a regular basis, I have not came across any glitches or disappointment.

We now have a working Ceph cluster with RBD and Ceph FS connected to our Proxmox cluster. In the next section, we are going to look at Ceph's CRUSH map and how we can manipulate it to customize a Ceph cluster.

# Learning Ceph's CRUSH map

A CRUSH map is the heart of Ceph's storage system. The algorithm of CRUSH dictates how data is stored and retrieved through the Ceph cluster. The topic of CRUSH alone is worthy of an entire book, but here, it is sufficient to understand that learning to manipulate CRUSH provides administrator an upper hand in storage management. It is only by manipulating CRUSH that we can set up a multitiered Ceph cluster to store virtual machines based on performance requirements.

Using CRUSH, we can move OSD from one node to another or even move an entire node to a different location without causing major disruption in the storage service. A CRUSH map is not readily available in the Ceph admin directory. We have to use CRUSH commands to extract, edit, and inject a CRUSH map back into a Ceph cluster. Basically, it is a five-step process:

1. Extract the CRUSH map.
2. Decompile it.
3. Edit the map.
4. Compile it.
5. Inject the map back into the cluster.

Let's look into a CRUSH map by first extracting it from the Ceph cluster. The CRUSH map will always need to be extracted first and decompiled to make it humanly readable, and then recompiled before injecting it into Ceph cluster.

# Extracting the CRUSH map

Before any sort of CRUSH map manipulation, the map has to be extracted from the Ceph cluster. Simply run the following command to extract the CRUSH map:

```
# ceph osd getcrushmap -o <any_name>
```

Once the extraction is complete, we highly recommend that you keep the first CRUSH map in a safe place as backup. This way, we can always put the original CRUSH map back into the cluster if something goes wrong when editing later. For our cluster, we are going to run the following command from the admin node:

```
# ceph osd getcrushmap -o crushmap1
```

We are going to put away this Crush map, `crushmap1`, in a folder in `/home/cephadmin/pmxceph/backup/crushmap`.

After successful extraction, we will see something like the following information:

```
got crush map from osdmap epoch 17
```

# Decompiling the CRUSH map

To make the extracted CRUSH map humanly readable, we need to decompile it with the following command format:

```
# crushtool -d <source_map> -o <readable.txt>
```

We will now use the following command format to decompile our `crushmap1` file extracted earlier:

```
# crushtool -d crushmap1 -o crushmap1.txt
```

# Editing the CRUSH map

We now have the CRUSH map in an easily readable text format. We can use any editor to open the CRUSH map for editing. For our cluster, we are going to use the nano editor. The following code is how our CRUSH map will look like:

```
# begin crush map

# devices
device 0 osd.0
device 1 osd.1
device 2 osd.2
device 3 osd.3

# types
```

```
type 0 osd
type 1 host
type 2 rack
type 3 row
type 4 room
type 5 datacenter
type 6 root

# buckets
host ceph-osd-01 {
  id -2    # do not change unnecessarily
  # weight 0.140
  alg straw
  hash 0  # rjenkins1
  item osd.0 weight 0.070
  item osd.2 weight 0.070
}
host ceph-osd-02 {
  id -3    # do not change unnecessarily
  # weight 0.140
  alg straw
  hash 0  # rjenkins1
  item osd.1 weight 0.070
  item osd.3 weight 0.070
}
root default {
  id -1    # do not change unnecessarily
  # weight 0.280
  alg straw
  hash 0  # rjenkins1
  item ceph-osd-01 weight 0.140
  item ceph-osd-02 weight 0.140
}

# rules
rule data {
  ruleset 0
  type replicated
  min_size 1
  max_size 10
  step take default
  step chooseleaf firstn 0 type host
  step emit
```

```
}
rule metadata {
  ruleset 1
  type replicated
  min_size 1
  max_size 10
  step take default
  step chooseleaf firstn 0 type host
  step emit
}
rule rbd {
  ruleset 2
  type replicated
  min_size 1
  max_size 10
  step take default
  step chooseleaf firstn 0 type host
  step emit
}

# end crush map
```

Before editing the CRUSH map, we highly recommend studying it with the help of Ceph CRUSH documentation. The documentation can be found at `https://ceph.com/docs/master/rados/operations/crush-map/`.

There are two terms we need to look at before proceeding with the CRUSH Map. They are bucket and ruleset.

Bucket in Ceph is a term used in CRUSH map hierarchy. For example, an OSD is in the host-01 bucket node, the host is in the rack-10 bucket, the rack is in the row-05 bucket, the row is in the bucket room 101, the room is in the bucket datacenter dc-01, and the datacenter is in the root bucket default.

In our previous CRUSH Map example, we have seven types of bucket for our Ceph cluster: osd, host, rack, row, room, datacenter, and root. Depending on how meticulous we want the management control to be, we can create more bucket types. For example, if we have a bunch of host nodes connected to a managed power distribution unit and we want to treat those hosts as a group, we will create a new bucket type, let's say pdu. Our bucket types in our CRUSH map will look like the following:

```
# types
type 0 osd
type 1 host
```

```
type 2 rack
type 3 row
typw 4 pdu
type 5 room
type 6 datacenter
type 7 root
```

When creating new buckets, be sure to maintain the sequence in the bucket types. For example, a room always belongs to a datacenter, so a datacenter should not come before the room in the types.

A ruleset in Ceph defines the strategies of how object replicas are distributed among OSDs based on the CRUSH map. For example, if we have a set of SSDs in the Ceph cluster, and we want a particular pool to store data only on those SSDs, we will create a ruleset and the ruleset to that pool. Whenever data is stored, that pool will always use the SSDs we have allocated in the ruleset. There are virtually no limitations on how ruleset can be used to dictate how and where Ceph pools can store data. Each ruleset has an integer numeric value. We use this value to assign a specific ruleset to a pool.

We are now going to add a root bucket named `ssd` and a ruleset `ssd` to the CRUSH map. Add the following bucket under the root default in the map:

```
root ssd {
  id -5    # do not change unnecessarily
  # weight 0
  alg straw
  hash 0  # rjenkins1
  item ceph-ssd-01 weight 0
  item ceph-ssd-02 weight 0
}
```

Add the following ruleset at the bottom of the map:

```
rule ssd {
  ruleset 3
  type replicated
  min_size 1
  max_size 10
  step take ssd
  step chooseleaf firstn 0 type host
  step emit
}
```

Add the following host bucket under the `host` segment:

```
host ceph-ssd-01 {
  id -6
  alg straw
  hash 0
}
host ceph-ssd-02 {
  id -7
  alg straw
  hash 0
}
```

Note that we just added two hosts, `ceph-ssd-01` and `ceph-ssd-02`. However, these nodes do not exist in our cluster. In a Ceph CRUSH map, we can add virtual hosts to separate pools and OSDs. When we add some SSDs, we can simply set the mode of the OSDs to these two hosts. The final CRUSH map should like the following code:

```
# begin crush map

# devices
device 0 osd.0
device 1 osd.1
device 2 osd.2
device 3 osd.3

# types
type 0 osd
type 1 host
type 2 rack
type 3 row
type 4 room
type 5 datacenter
type 6 root

# buckets
host ceph-osd-01 {
  id -2    # do not change unnecessarily
  # weight 0.140
  alg straw
  hash 0  # rjenkins1
  item osd.0 weight 0.070
  item osd.2 weight 0.070
}
host ceph-osd-02 {
```

```
    id -3    # do not change unnecessarily
    # weight 0.140
    alg straw
    hash 0  # rjenkins1
    item osd.1 weight 0.070
    item osd.3 weight 0.070
  }
  root default {
    id -1    # do not change unnecessarily
    # weight 0.280
    alg straw
    hash 0  # rjenkins1
    item ceph-osd-01 weight 0.140
    item ceph-osd-02 weight 0.140
  }
  host ceph-ssd-01 {
    id -6    # do not change unnecessarily
    # weight 0.000
    alg straw
    hash 0  # rjenkins1
  }
  host ceph-ssd-02 {
    id -7    # do not change unnecessarily
    # weight 0.000
    alg straw
    hash 0  # rjenkins1
  }
  root ssd {
    id -5    # do not change unnecessarily
    # weight 0.000
    alg straw
    hash 0  # rjenkins1
    item ceph-ssd-01 weight 0.000
    item ceph-ssd-02 weight 0.000
  }

  # rules
  rule data {
    ruleset 0
    type replicated
    min_size 1
    max_size 10
    step take default
    step chooseleaf firstn 0 type host
```

```
    step emit
  }
  rule metadata {
    ruleset 1
    type replicated
    min_size 1
    max_size 10
    step take default
    step chooseleaf firstn 0 type host
    step emit
  }
  rule rbd {
    ruleset 2
    type replicated
    min_size 1
    max_size 10
    step take default
    step chooseleaf firstn 0 type host
    step emit
  }
  rule ssd {
    ruleset 3
    type replicated
    min_size 1
    max_size 10
    step take ssd
    step chooseleaf firstn 0 type host
    step emit
  }

  # end crush map
```

# Compiling the CRUSH map

We are now going to compile our edited CRUSH map from text to a machine-readable format with the following command:

```
# crushtool -c crushmap1.txt -o crushmap1
```

Ceph's `crushtool` has a built-in validation system. If we have made a syntax error while editing, `crushtool` will let us know the error before proceeding further.

# Injecting the CRUSH map into the cluster

To inject the new CRUSH map, run the following command:

```
# ceph osd setcrushmap –i crushmap1
```

After injecting the new CRUSH map successfully, we should see the following message on the screen:

```
# set crush map
```

# Verifying the new CRUSH map

We are now going to verify our new CRUSH map. We can verify it using the following command:

```
cephadmin@ceph-admin-01:~/pmxceph$ sudo ceph osd tree
```

After the command is executed, the following output will be displayed:

```
# id    weight    type name       up/down      reweight
-5      0         root ssd
-6      0                 host ceph-ssd-01
-7      0                 host ceph-ssd-02
-1      0.28      root default
-2      0.14              host ceph-osd-01
0       0.06999                   osd.0        up      1
2       0.06999                   osd.2        up      1
-3      0.14              host ceph-osd-02
1       0.06999                   osd.1        up      1
3       0.06999                   osd.3        up      1
```

We can see that we have successfully added a new host and root bucket. There are no OSDs in them because we have not moved any OSD yet.

Let's extract and decompile our CRUSH map from the steps we have already seen. The extracted CRUSH map should look like the following text:

```
# devices
device 0 osd.0
device 1 osd.1
```

```
device 2 osd.2
device 3 osd.3

# types
type 0 osd
type 1 host
type 2 rack
type 3 row
type 4 room
type 5 datacenter
type 6 root

# buckets
host ceph-osd-01 {
  id -2    # do not change unnecessarily
  # weight 0.140
  alg straw
  hash 0  # rjenkins1
  item osd.0 weight 0.070
  item osd.2 weight 0.070
}
host ceph-osd-02 {
  id -3    # do not change unnecessarily
  # weight 0.140
  alg straw
  hash 0  # rjenkins1
  item osd.1 weight 0.070
  item osd.3 weight 0.070
}
root default {
  id -1    # do not change unnecessarily
  # weight 0.280
  alg straw
  hash 0  # rjenkins1
  item ceph-osd-01 weight 0.140
  item ceph-osd-02 weight 0.140
}
host ceph-ssd-01 {
  id -6    # do not change unnecessarily
  # weight 0.000
  alg straw
  hash 0  # rjenkins1
}
```

```
host ceph-ssd-02 {
  id -7    # do not change unnecessarily
  # weight 0.000
  alg straw
  hash 0  # rjenkins1
}
root ssd {
  id -5    # do not change unnecessarily
  # weight 0.000
  alg straw
  hash 0  # rjenkins1
  item ceph-ssd-01 weight 0.000
  item ceph-ssd-02 weight 0.000
}

# rules
rule data {
  ruleset 0
  type replicated
  min_size 1
  max_size 10
  step take default
  step chooseleaf firstn 0 type host
  step emit
}
rule metadata {
  ruleset 1
  type replicated
  min_size 1
  max_size 10
  step take default
  step chooseleaf firstn 0 type host
  step emit
}
rule rbd {
  ruleset 2
  type replicated
  min_size 1
  max_size 10
  step take default
  step chooseleaf firstn 0 type host
  step emit
}
```

```
rule ssd {
  ruleset 3
  type replicated
  min_size 1
  max_size 10
  step take ssd
  step chooseleaf firstn 0 type host
  step emit
}

# end crush map
```

# Managing Ceph pools

Ceph stores data on pools assigned to a specific ruleset. A ruleset in a CRUSH map dictates which pools belong to which OSDs. By simply changing an assigned ruleset, we can use a complete set of OSDs to store existing data. In this example, we are going to create a new pool and assign the pool to the new ruleset `ssd`, which we created earlier. Then we will add some SSDs to the cluster.

# Creating a new Ceph pool using the CLI

The following is the command format to create the Ceph pool:

```
# ceph osd pool create <poolname> <pg> <pgs(equal to pg)>
```

Before we create the pool, we need to calculate the number of PGs for the pool. We will use the same formula seen in the *Placement Group (PG)* section earlier in this chapter:

$$\text{Total PGs} = \left(\text{OSD} \times 100\right) / \text{Number of Replicas}$$

We have two nodes and will use two SSDs in each node. According to the formula, we need the following number of PGs:

$$\text{Total PGs} = \left(4 \times 100\right) / 2 = 200$$

Our new pool should have 200 PGs. So the following is our command to create a pool:

```
# ceph osd pool create ssd 256 256
```

# Verifying the new Ceph pool

We can verify whether the new pool is created using the following command:

```
# rados lspools
```

The command should show the following result:

```
data
metadata
rbd
ssd
```

We can also check the status of a cluster along with the number of PGs using the following command:

```
# ceph -s
```

We will see the following status of the Ceph cluster:

```
cluster 472cefbe-11a6-40f7-a352-5d245ae89fb3
     health HEALTH_OK
     monmap e2: 3 mons at {ceph-mon-01=192.168.10.11:6789/0,ceph-mon-
       02=192.168.10.12:6789/0,ceph-mon-03=192.168.10.13:6789/0},
         election epoch 6, quorum 0,1,2 ceph-mon-01,ceph-mon-02,ceph-
           mon-03
     mdsmap e5: 1/1/1 up {0=ceph-mds-01=up:active}, 1 up:standby
     osdmap e20: 4 osds: 4 up, 4 in
      pgmap v71: 448 pgs, 4 pools, 18610 bytes data, 21 objects
            141 MB used, 299 GB / 299 GB avail
                 448 active+clean
```

Based on the status information, we now have four pools with 448 active+clean PGs.

# Adding OSDs to a pool

We will now add some SSDs to our new pool to be used as high-performing storage. Perform the steps mentioned in the *Creating OSDs* subsection (in the *Installing Ceph using an OS* section of this chapter) to add the OSDs. Be advised that when adding or removing OSDs, the cluster goes to recovery mode, which might slow down the network by increasing traffic on the Ceph cluster-dedicated network.

**Recovery mode** or **rebalancing** in Ceph means when there is an OSD or node failure or new OSDs are added, the Ceph cluster will automatically try to redistribute or rebalance data among the remaining active OSDs and nodes. During this rebalancing, the Ceph cluster creates heavy I/O and network bandwidth traffic causing slow downs. The period of recovery mode or rebalancing depends on how much data is stored in the cluster and how many objects the cluster has to rebalance.

It is best to set a cluster to not bring OSDs online right away to prevent rebalancing. Run the following commands to prevent the OSDs from coming online as soon as they are added:

```
# ceph osd set noin
# ceph osd set noup
```

This will prevent new OSDs from reaching the up and in statuses. Each OSD can have several statuses to let us know the condition the OSD is in. The presence of an OSD in the cluster is represented by 0 or 1 for an in or out condition, while the OSD daemon connection is represented by an up or down condition. For example, an OSD which is active in the cluster and receives acknowledgement from Ceph Monitor will have a status of up and 1. If the OSD is still in the cluster but OSD daemon is not responding to Monitor, the status will be down and 1. When the OSD is deactivated or fails completely, the status will be down and 0, that is, it is down and out of the cluster. The Ceph cluster will start balancing as soon as the OSD status changes.

By default, when we add OSD, they are put in the default ruleset. In this case, we want to put SSD OSDs into ruleset 3 created earlier. So we do not want the cluster to start distributing data to this new OSD right away before we move them to ruleset created for the SSD.

After adding the new OSDs, the following is the status information we should get from the `osd tree` command:

```
cephadmin@ceph-admin-01:~/pmxceph$ sudo ceph osd tree
```

After the command is executed, we can see the following output:

```
# id    weight  type name       up/down       reweight
-5      0       root ssd
-6      0               host ceph-ssd-01
-7      0               host ceph-ssd-02
-1      0.3599  root default
-2      0.18            host ceph-osd-01
0       0.06999                 osd.0         up      1
2       0.06999                 osd.2         up      1
4       0.01999                 osd.4         up      0
5       0.01999                 osd.5         up      0
-3      0.18            host ceph-osd-02
1       0.06999                 osd.1         down    1
3       0.06999                 osd.3         down    1
```

```
6       0.01999                 osd.6           up      0
7       0.01999                 osd.7           up      0
```

In the previous output, osd.4, osd.5, osd.6, and osd.7 are the four SSD OSDs we added. Now we will move them to their respective virtual host buckets.

Virtual host buckets are nothing but aliases for the existing physical nodes we created through the CRUSH map. The ceph-osd-01 and ceph-osd-02 nodes are main physical nodes. The ceph-ssd-01 and ceph-ssd-02 nodes are aliases for the ceph-osd-01 and ceph-osd-02 nodes, which separate the pool to be used with SSD-based OSDs. We can create as many virtual hosts as required to separate any number of pools or assign specific OSDs to a given pool.

We are going to use the following command format to move OSDs to virtual host buckets:

```
# ceph osd crush set <osd.id> <weight> <host_bucket>
```

Based on this format, we will enter the following commands to move osd.4, osd.5, osd.6, and osd.7 to the virtual host nodes ceph-ssd-01 and ceph-ssd-02:

```
# ceph osd crush set osd.4 0.0199 host=ceph-ssd-01
# ceph osd crush set osd.5 0.0199 host=ceph-ssd-02
# ceph osd crush set osd.6 0.0199 host=ceph-ssd-01
# ceph osd crush set osd.7 0.0199 host=ceph-ssd-02
```

After the OSDs are moved, we are now going to bring the OSDs online using the following commands. First we have to remove the restriction placed earlier to prevent OSDs from coming online on their own:

```
# ceph osd unset noin
# ceph osd unset noup
```

Now we will start putting all the new OSDs online using the following commands:

```
# ceph osd in osd.4
# ceph osd in osd.5
# ceph osd in osd.6
# ceph osd in osd.7
```

Our OSDs placement will look like the following code:

```
cephadmin@ceph-mon-01:~/pmxceph$ sudo ceph osd tree
```

We can see the following output after the command is executed:

```
# id    weight   type name         up/down  reweight
-5      0.07959  root ssd
-6      0.03979          host ceph-ssd-01
4       0.0199                    osd.4    up      1
5       0.0199                    osd.5    up      1
-7      0.03979          host ceph-ssd-02
6       0.0199                    osd.6    up      1
7       0.0199                    osd.7    up      1
-1      0.28     root default
-2      0.14             host ceph-osd-01
0       0.06999                   osd.0    up      1
2       0.06999                   osd.2    up      1
-3      0.14             host ceph-osd-02
1       0.06999                   osd.1    up      1
3       0.06999                   osd.3    up      1
```

# Assigning a pool to the ruleset

The only thing we have left to do is set the ruleset for the new pool, ssd. By default, ruleset 0 is set for a pool. In order to get the assigned ruleset for a pool, we will use the following command:

```
# ceph osd pool get ssd crush_ruleset
```

The previous command should print the following result on the screen:

```
crush_ruleset: 0
```

We will set ruleset 3 to the SSD pool using the following command:

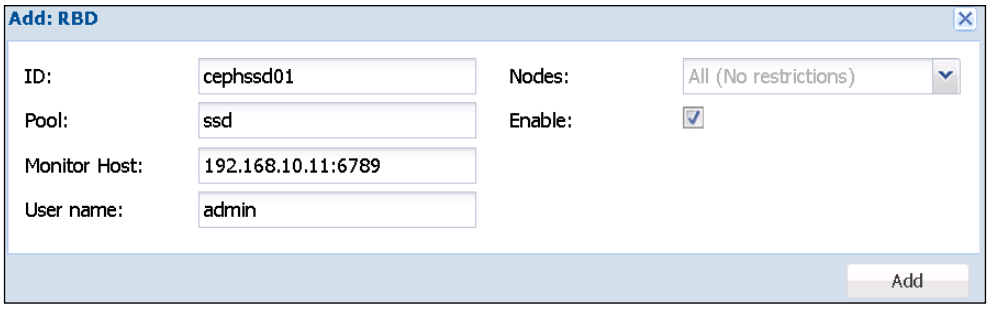
```
# ceph osd pool set ssd crush_ruleset 3
```

The previous command should print the following result on the screen:

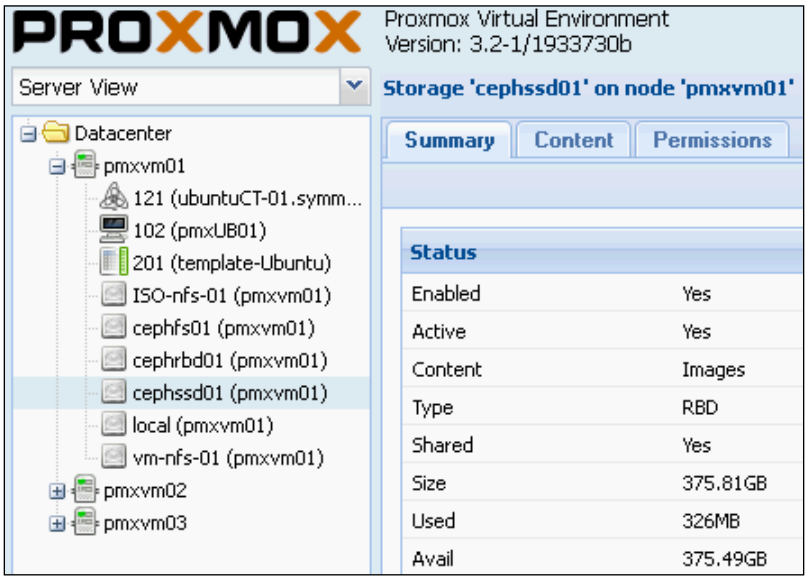
```
crush_ruleset: 3
```

# Connecting Proxmox to the new pool

After we assign the ruleset, our pool is ready to be connected to Proxmox. Perform the following steps mentioned in the *Connecting Proxmox to a Ceph cluster* subsection in the *Installing Ceph using an OS* section. We will name the storage `cephssd01`. So we need to copy the Ceph admin keyring to `/etc/pve/priv/ceph/cephssd01.keyring`. Then use the information shown in the following screenshot to connect the pool to Proxmox:




We can verify that both the RBD storages are functioning from the Proxmox GUI status page:



Please note that both storage pools will show the exact same size and usage information because they are on the same cluster. The only way to fully separate them is to create two separate Ceph clusters. Clusters can be on the same or different hardware.

# Ceph benchmarking

There are many ways to run a benchmark test on a Ceph cluster to check disk drive, network, and cluster performance. Use the following commands to test the performance of the Ceph cluster.

To write a block of data to test write performance, use the following command:

```
root@node:/# rados -p <pool> bench -b <blockSize> <seconds> <write> -
  t <threads> --no-cleanup
```

To read a block of data to test read performance, use the following command:

```
root@node:/# rados -p <pool> bench -b <blockSize> <seconds> <seq> -t
<threads>
```

After each of the previous commands, run the following command to clear cache:

```
root@node:/# echo 3 > tee /proc/sys/vm/drop_caches && sync
```

We can create a separate pool for the purpose of benchmarking so that we do not run benchmark tests on live production pools. Let us create a pool to run cluster benchmarking using the following command:

```
root@node:/# ceph osd pool create test 256 256
```

For a good spread of benchmarks, use the block sizes of 4096, 131072, and 4194304 bytes. After each write performance, run the read benchmark. The no-cleanup option will make sure that data written for write performance is not automatically deleted after the write test. We will need the written data to perform the read test. Based on the benchmark command format, the four following steps test each block size. Use 300 seconds and 32 threads for sustained performance tests:

1. To test data with block size 4096 bytes for a write performance, use the following command:

   ```
   root@node:/#rados -p test bench -b 4096 300 write -t 32 --no-
     cleanup
   ```

2. Clear cache using the following command:

   ```
   root@node:/# echo 3 > tee /proc/sys/vm/drop_caches && sync
   ```

3. Test data with block size 4096 bytes for read performance:

```
root@node:/#rados -p test bench -b 4096 300 write -t 32 --no-
  cleanup
```

4. Clear cache again:

```
root@node:/# echo 3 > tee /proc/sys/vm/drop_caches && sync
```

The following table shows some benchmark results from the four different Ceph clusters with a various number of OSDs and SSDs. All benchmarking was done with a 1 gigabit network, desktop-class 1 TB hard drives, and Kingston KC300 240 GB SSDs. The benchmark commands we saw in this section were primarily used for all benchmarks.

The following table shows benchmark results with six SSDs as the OSD:

| Blocksize (bytes) | Throughput (MBps) | | IOPS per disk | | Total IOPS | |
|---|---|---|---|---|---|---|
| | Write | Read | Write | Read | Write | Read |
| 4096 | 7.078 | 12.771 | 301.995 | 544.896 | 1811.968 | 3269.376 |
| 131,072 | 93.084 | 74.421 | 124.112 | 99.227 | 744.672 | 595.362 |
| 4,194,304 | 106.536 | 85.433 | 4.439 | 3.559 | 26.634 | 21.358 |

The following table shows benchmark results with six desktop HDDs as the OSD:

| Blocksize (bytes) | Throughput (MBps) | | IOPS per disk | | Total IOPS | |
|---|---|---|---|---|---|---|
| | Write | Read | Write | Read | Write | Read |
| 4096 | 0.242 | 4.125 | 10.325 | 176 | 61.952 | 1056 |
| 131,072 | 8.139 | 45.529 | 10.852 | 60.705 | 65.112 | 364.232 |
| 4,194,304 | 54.124 | 74.114 | 2.255 | 3.088 | 13.531 | 18.528 |

The following table shows benchmark results with eight desktop HDDs as the OSD:

| Blocksize (bytes) | Throughput (MBps) | | IOPS per disk | | Total IOPS | |
|---|---|---|---|---|---|---|
| | Write | Read | Write | Read | Write | Read |
| 4096 | 1.137 | 12.172 | 36.384 | 389.504 | 291.072 | 3116.032 |
| 131,072 | 16.422 | 56.857 | 16.422 | 56.857 | 131.376 | 454.856 |
| 4,194,304 | 88.661 | 78.277 | 2.770 | 2.447 | 22.165 | 19.569 |

The following table shows benchmark results with 26 desktop HDDs as the OSD:

| Blocksize (bytes) | Throughput (MBps) | | IOPS Per Disk | | Total IOPS | |
|---|---|---|---|---|---|---|
| | **Write** | **Read** | **Write** | **Read** | **Write** | **Read** |
| 4096 | 2.139 | 17.754 | 21.061 | 174.809 | 547.584 | 4545.024 |
| 131,072 | 18.061 | 108.86 | 5.557 | 33.495 | 144.488 | 870.88 |
| 4,194,304 | 109.787 | 112.042 | 1.056 | 1.077 | 27.447 | 28.011 |

From the previous tables of benchmark results, it can be clearly seen that the higher number of OSDs does increase performance in Ceph cluster. Out of the four setups, the benchmark with six OSDs has the worst performance. In all the previous benchmarks, Journal is co-located with each OSD.

# The Ceph command list

The following table shows a list of Ceph commands most frequently used to run a healthy cluster:

| Command | Description |
|---|---|
| `ceph-deploy install <node>` | Install Ceph on nodes |
| `ceph-deploy disk list <node>` | Disk list |
| `ceph-deploy disk zap <node>:/dev/sdX` | Format disk |
| `ceph-deploy osd create <node>:/dev/sdX` | Create OSD |
| `ceph osd out <osd.id>`<br>`stop ceph-osd <osd.id>`<br>`umount /var/lib/ceph/osd/<cluster>.<osdid>`<br>`ceph osd crush remove <osd.id>`<br>`ceph auth del <osd.id>`<br>`ceph osd rm <osd.id>` | Remove OSD manually |
| `ceph-deploy mon create <node>` | Deploy Monitor (MON) |
| `ceph-deploy mon destroy <node>` | Delete Monitor (MON) |
| `ceph-deploy mds create <node>` | Deploy MetaData Server (MDS) |
| `ceph-deploy mds destroy <node>` | Delete MetaData Sever (MDS) |
| `ceph-deploy gatherkeys <mon_node>` | Gather admin keys |
| `ceph osd lspools` | List pools |

| Command | Description |
|---|---|
| `ceph osd pool create <name> <pg> <pgs>` | Create pool |
| `ceph osd pool delete <name> [<name> --yes-i-really-really-mean-it]` | Delete pool |
| `ceph osd pool get <name> pg_num` | Get the number of PG of pool |
| `ceph osd pool set <name> crush_ ruleset <value>` | Set pool CRUSH ruleset |
| `ceph osd getcrushmap –o <name>` | Get CRUSH map |
| `crushtool –d <name> -i <name.txt>` | Decompile CRUSH map |
| `crushtool –c <name.txt> -o <name>` | Compile CRUSH map |
| `ceph osd setcrushmap –i <name>` | Inject CRUSH map |
| `ceph-fuse –k <keyring> -m <mon:port> /<folder> -o nonempty` | Mount Ceph FS |
| `Fusermount –u /<folder>` | Unmount Ceph FS |

# Summary

In this chapter, we learned to install a Ceph cluster from the ground up and connected it to our Proxmox cluster. Treat this chapter as an introduction to Ceph. There is a lot more to Ceph than what has been covered here. Read the Ceph official documentation to learn about Ceph in greater detail.

Ceph is an excellent choice for a shared storage system for Proxmox due to its stability, performance, and significant lower cost. No matter the workload, a Ceph cluster can expand to meet the demand. Whether it is a single-tier or multitier storage environment, Ceph can handle them with ease. With Proxmox Version 3.2, managing Ceph just got better.

In the next chapter, we are going to get a glimpse of an enterprise-sized virtual environment, hardware requirements, and what keeps a cluster going month after month without downtime.

**Exercise:**

Consider a Ceph cluster has five nodes, 120 OSDs, three replicas, and four pools. Based on this information, we need to find the following values:

- What is the PG value per pool?
- How many PGs per OSD for each replica?
- What is the total PGs per OSD for all pools and replicas?

Total PG for each pool is 1000. Each replica of pools will have 32 PGs per OSD. Total PGs per OSD for all pools and replicas is 96. The calculations are as follows:

$$\text{Total PGs} = \left(120 \times 100\right) / 3 \text{ replicas} = 4000$$

Since there are four pools, each pool will have 4000 / 4 = 1000 PGs.

PGs per OSD per pool for each replica = 1000 PGs / 120 OSDs = 8.

PGs per OSD for all 4 pools for each replica = 8 x 4 = 32.

PGs per OSD for all 4 pools for all 3 replicas = 32 x 3 = 96.

# 8
# Proxmox Production Level Setup

Throughout the book so far, we have seen the internal workings of Proxmox. We now know how to properly set up a fully functional Proxmox cluster. We learned about Ceph—one of the best shared storage systems—and how we can connect it with Proxmox. We also saw what a virtual network is and how it works with the Proxmox cluster.

In previous chapters, we built a very basic Proxmox cluster. For learning purposes this might be okay, but in a fully operational production environment we need a robust setup to handle the workload. That is what a production level setup is all about. In this chapter, we are going to cover the following topics:

- Definition of production level
- Key components of a production level setup
- Entry-level and advanced-level hardware requirements
- A simple way to track the hardware inventory
- AMD-based hardware in a production level environment

Throughout this chapter, you will notice that we will use a user-built hardware configuration instead of ready-made brand servers. The purpose of this is to show you what sort of node configuration is possible using the off-the-shelf commodity hardware to cut the cost while setting up a stable Proxmox cluster. All example configurations shown in this chapter are not theoretical scenarios, but taken from live clusters in service. Use the information in this chapter purely as guidelines so that you can select proper hardware for your environment on any budget.

# Defining a production level

Production level or mode is a stage where a company's cluster environment is fully functional and actively serving its users or clients on a regular basis. It is no longer considered as a platform to learn Proxmox or a test platform to test different things on. A production level setup requires much advanced planning and preparation. This is because once the setup is complete and the cluster has been brought online, it just cannot be taken offline completely at a moment's notice, especially when users become dependent on it. A properly planned production level setup can save hours or days of headache. In this section, we will see how to preplan and choose the right equipment for a production level cluster.

# Key parameters

There are several key parameters to be kept in mind while planning for a production level cluster setup due to stability and performance requirements; some of them are as follows:

- Stable and scalable hardware
- Current load versus future growth
- Budget
- Simplicity
- Tracking the hardware inventory

## Stable and scalable hardware

Stable hardware means minimum downtime. Period! Without quality hardware, it is not unusual to have a simultaneous failure of hardware in a cluster environment, causing massive downtime. It is very important to select a hardware brand with good reputation and support behind it. For example, Intel is well known for its superb stability and support. It is true that you need to pay more for Intel products, but sometimes stability outweighs the higher cost per hardware. AMD is also an excellent choice, but statistically, AMD-based hardware has more stability issues.

For a budget-conscious enterprise environment, we can mix both Intel- and AMD-based hardware in the same cluster. Since Proxmox provides us with the full migration option, we can have Intel nodes serving us full time, while AMD nodes only act as a failover, thus reducing the cost without compromising on stability. Throughout this chapter, we are going to stay primarily with Intel-based hardware. At the end of this chapter, we will look at some proven AMD-based cluster to give you some idea of how viable AMD usage is in a Proxmox cluster environment.

## Current load versus future growth

Never design a cluster with only the present day in mind; always look into the future, at least the near future. An enterprise cluster must be able to grow with the company and adapt to the increased operations. However, for the most part, plan in such a way that a cluster does not max out in a very short period of time. Both Proxmox and the Ceph cluster have the ability to grow at any time to any size: this provides the ability to simply add new hardware nodes to expand the size of the cluster and increase the resources required by virtual machines.

When installing memory in nodes, never add memory only to run the existing virtual machines, but foresee any future virtual machine migration to any nodes during a node failure. For example, let's say all the six nodes in a Proxmox cluster have 64 GB of memory, and 60 GB is consumed at all times by all the virtual machines. If node 1 fails, you will not be able to migrate all the virtual machines from node 1 to the other five nodes. This is because there will not be enough memory to go around. We could just add another spare node and migrate all the virtual machines, but we have to make sure that there are enough power outlets to even plug in the new node. Usually, 50 percent of the maximum capacity should be free in a cluster environment. This is possible if you have a large budget.

## Budget

Budgetary concerns always play a role in decision making, no matter what kind of network environment we are dealing with. However, the fact is that a setup can be adaptable to just about any budget with some clever and creative planning. Times too numerous to count, administrators have had to work with a very small IT budget. Hopefully, this chapter will help you find that missing thread to connect a budget with proper hardware components.

## Simplicity

Simplicity is often overlooked in a network environment. A lot of times, it just happens naturally. If we are not mindful about simplicity, we can very quickly make a network unnecessarily complex. By mixing hardware RAID with software RAID, putting one RAID within another RAID, or through a multidrive setup to protect the OS, we can quickly make the cluster go out of control. Both Proxmox and Ceph can run on high-grade commodity hardware. For example, just by selecting the desktop-class i7 over the server-class Xeon, we can slash the cost by 50 percent while providing a very stable and simple cluster setup, unless the task specifically calls for multi-Xeon setup.

## Tracking the hardware inventory

There are several key pieces of information for the hardware that is being used in a network that an administrator should any time have access to: these include information such as the brand, model and serial number of a hardware component, when was it purchased, who was the vendor, when is it due for replacement, and so on. A proper tracking system can save a lot of time when any of this information needs to be retrieved. Each company is different; accordingly, the tracking system could be different. Nevertheless, the responsibility of gathering this information solely falls on the network manager or administrator. If there is no system in place, then merely creating a simple spreadsheet can be enough to keep track of all hardware-related information.

## Hardware selection

Several factors affect the process of selecting a type of hardware. For example, is the cluster going to support many virtual machines with smaller resources or will it serve a few virtual machines with high resources? A cluster focused on many virtual machines needs to have many more core counts. So our goal should be to put as many cores as possible per node. A cluster is focused on a few virtual machines; therefore, if there are many users per virtual machine, we need to have a large memory. Thus, a system with less core but large memory is much more appropriate. Also, with this, a cluster can focus on both the types and create a hybrid cluster environment. A hybrid environment usually starts with an entry -level hardware setup and then matures into an advanced-level setup as the company grows and budget becomes available. For example, a small company can start its cluster infrastructure with a stable desktop-class hardware and then gradually replace them with a server-class platform such as Xeon to accommodate the company's expansion. Since we are working with a clustered environment, we will assume that all our nodes are identical to each other. Therefore, we are only going to look at one node setup.

# An entry-level Proxmox production setup

Do not let the word **entry-level** fool you to think that it is a substandard, under-performing cluster setup. An entry-level setup provides a balance between high-core and high-memory cluster hardware setup. It is usually a single CPU setup, which can be easily set up using commodity hardware. This is a setup where choices between desktop CPUs, such as i5 and i7, and server CPUs, such as Xeon, can be made.

The following are examples of desktop- and server-class components from two live clusters currently in service. Both of these examples belong to two low IT budget organizations. Please note that these components are listed only to provide you with a guideline.

## An i7-based Proxmox node

The following image shows a rack-mounted Proxmox node based on a desktop CPU and motherboard. The hardware details are as follows:

- **CPU**: Intel i7-4820K LGA 2011.
- **Motherboard**: MSI X79A-GD45 LGA 2011. The maximum amount of memory supported is 128 GB.

http://us.msi.com/product/mb/X79AGD45_Plus.html

- **Chassis**: In-Win IW-R200 ATX 500W PSU.

http://www.in-win.com.tw/Server/zh/goods.php?act=view&id=IW-R200

- **SSD for OS**: Kingston KC300 60 GB.

Based on this Intel i7 CPU and MSI motherboard, each Proxmox node can have a total of eight cores and 128 GB of memory without spending significant amount of money on server-class hardware.

## A Xeon-based Proxmox node

The following image shows a rack-mounted Proxmox node based on a Xeon CPU and motherboard. The hardware details are as follows:

- **CPU**: Intel Xeon E3-1245v3 LGA 1150.
- **Motherboard**: Intel S1200V3RPS. The maximum amount of memory supported is 32 GB.

http://ark.intel.com/products/71385/Intel-Server-Board-S1200V3RPS

- **Chassis**: In-Win IW-R200 ATX 500W PSU.
- **SSD for OS**: Kingston KC300 60 GB.

Based on this Intel Xeon CPU and server motherboard, each Proxmox node can have a total of eight cores and 32 GB of memory.

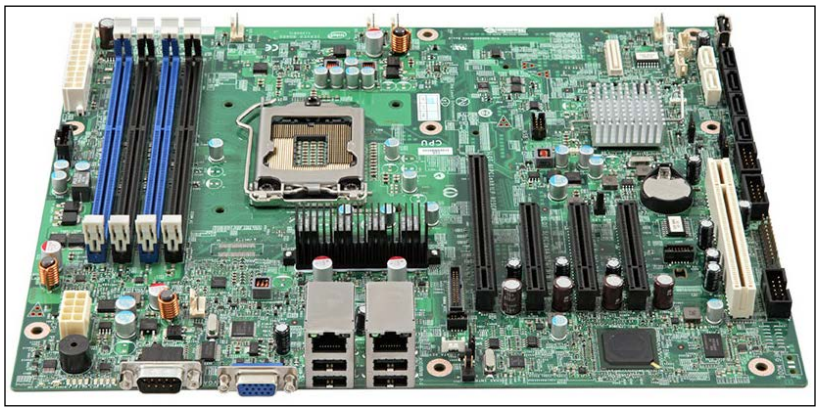
Although both the setups are of the similar price range, with the i7 setup we actually stand to gain much more. The performance of i7-4820K is slightly better than Xeon's E3-1245v3, and it provides us with the ability to have a full array of 128 GB of memory for very high-performing virtual machines.

This entry-level setup is deployed in a real production environment and has been in operation for more than a year without any issue.

# An entry-level Ceph production setup

The Ceph storage system requires very less CPU resources and a moderate amount of memory. The following is an entry-level setup for Ceph:

- **CPU**: Intel i3-3220 LGA 1155.
- **Motherboard**: Intel Server S1200BTLR LGA 1155. The installed memory is 8 GB ECC.



http://ark.intel.com/products/67494/Intel-Server-Board-S1200BTLR

- **Chassis**: In-Win IW-RS212-02 12 Bay Hot Swap.

http://www.in-win.com.tw/Server/zh/goods.php?act=view&id=IW-RS212-02

- **RAID controller**: Intel RS2WC040 6G SAS PCI-E.

http://ark.intel.com/products/43165/Intel-RAID-Controller-RS2WC040

- **RAID controller expander**: Intel RES2SV240 RAID Expander.

http://www.intel.com/content/www/us/en/servers/raid/raid-controller-res2sv240.html

- **SSD for OS**: Kingston KC300 60 GB.
- **HDD for storage**: Seagate 2 TB Standard or Seagate Hybrid 2 TB SSHD.

Note that we have added a RAID controller and an expander to the previous Ceph setup. Neither RAID nor expander cards are used to provide the RAID service; they are used to connect 12 drives. This same combination of RAID and expander is also capable of running a 24 Hot Swap bay chassis. An expander card connects to one port of the RAID controller, and the backplane of a Hot Swap port connects to the expander card.

With the entire Hot Swap bay filled with 2 TB HDD, this setup can handle 24 TB of storage without breaking a sweat. If you use 4 TB HDD across all the bays, then install an additional 8 GB of RAM to handle a total storage of 48 TB. When Ceph is set up with three identical nodes with the previously mentioned hardware components, it can provide a very stable shared storage system for the Proxmox cluster.

# An advanced-level Proxmox production setup

When setting up advanced-level enterprise class nodes, quality, stability, and performance usually come before budget. However, the selection has to start from somewhere, and the following setup should be that starting point. There is no place for desktop-class hardware at this stage. The following components or similar should be the minimum standard for an advanced-level setup. Use this information only as guidelines.

## A Xeon-based Proxmox node

The following image shows a rack-mounted Proxmox node based on an Intel Xeon CPU and Intel motherboard with 10 GB of network backing. The hardware details are as follows:

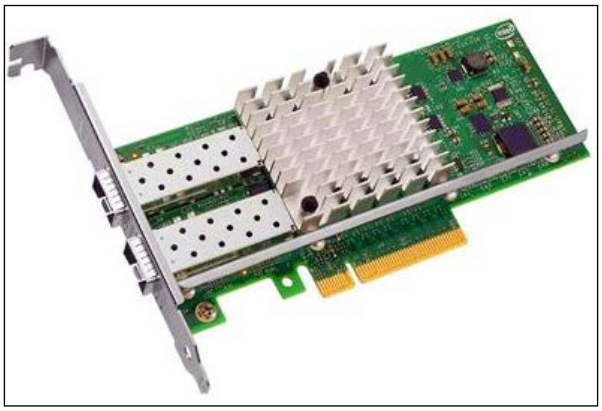
- **CPU**: Intel Xeon E5-2620v2 LGA 2011.
- **Motherboard**: Intel S2600CP2 Dual LGA 2011. The maximum amount of memory supported is 512 GB.

http://ark.intel.com/products/56333/Intel-Server-Board-S2600CP2

- **Chassis**: In-Win IW-RS104-02 Redundant PSU.

http://www.in-win.com.tw/Server/zh/goods.php?act=view&id=IW-RS104-02

- **10 GB network card**: Intel X520-SR2 Dual 10 GB.



http://ark.intel.com/products/39774/Intel-Ethernet-Converged-Network-Adapter-X520-SR2

- **SSD for OS**: Intel S3500 SSD 120 GB.

Based on the Intel Dual Xeon CPU and server motherboard, each Proxmox node can have total of 16 cores and 512 GB of memory where 10 GB network connectivity will provide a high bandwidth and reliable shared storage data transfer.

# An advanced-level Ceph production setup

The following are the components for an advanced-level Ceph setup:

- **CPU**: Intel Xeon E3-1230v2 LGA 1155.
- **Motherboard**: Intel Server S1200BTLR LGA 1155. The installed memory is 32 GB ECC.
- **Chassis**: In-Win IW-RS212-02 12 Bay Hot Swap with Redundant PSU.
- **RAID controller**: Intel RS2WC040 6G SAS PCI-E.
- **RAID controller expander**: Intel RES2SV240 RAID Expander.
- **10 GB network card**: Intel X520-SR2 Dual 10 GB.

- **SSD for OS**: Intel S3500 SSD 120 GB.
- **HDD for storage**: Seagate Constellation 2 TB Enterprise.
- **SSD for Storage**: Intel SSD DC S3700 800 GB. Use this SSD when creating all SSD-Ceph shared storage nodes.

Compared to the entry-level Ceph setup, we have not changed much in this advanced-level Ceph setup. We have replaced the Intel i3 CPU with the Intel Xeon E3 series' CPU and added a full array of 32 GB of memory. Also, we have added a dual 10 GB network interface card to create a 10 GB backing storage backbone. In both the setups, we have taken the hardware-based RAID completely out of the equation. Due to the nature of Ceph, we no longer need a RAID-based hardware to provide storage redundancy, thus keeping storage nodes simple.

The components mentioned previously for an advanced-level Proxmox and Ceph should be considered as a bare minimum configuration for a very high-demand virtual environment with mission-critical data. Since both Proxmox and Ceph have clustering, it is just a matter of adding more identical nodes to increase the capacity. Depending on high computing needs, a Proxmox node can have four CPUs set up with a higher Xeon CPU.

# Desktop class versus server class

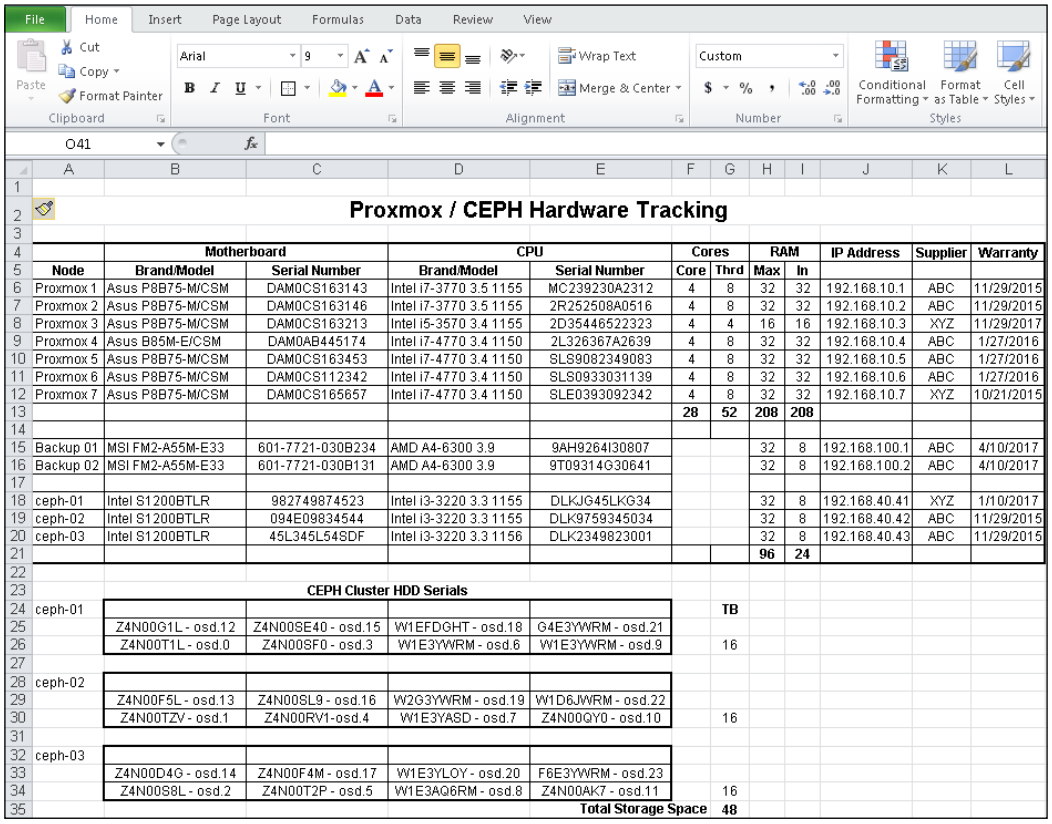
The main benefit of having desktop-class hardware over server-class hardware is to be able to have many cheaper nodes in the cluster rather than a few powerful pricey nodes. By having many nodes, we increase our chances of redundancy while spreading the computational load over many nodes. There are many real-world virtual environments that are run using an array of desktop-class components. However, do keep in mind that although you will pay a much higher price, there is really no substitute to using the extremely stable and consistent performance of pricey server-class components.

# Brand servers

The server node setup we have seen so far can easily be assembled in house. Also preassembled brand servers such as Intel, Dell, HP, IBM, or Supermicro can also be purchased based on the previous components' guidelines. The downside of brand servers is their higher cost. In some cases, it may cost twice the cost of assembling them in house. However, if you want to opt for brand servers, we recommend you to stay with Intel or Supermicro (`www.supermicro.com`). Supermicro has a great reputation for their product lines, and their price is well balanced for the performance they deliver. There are also other brands such as Dell and IBM that are worth considering while deciding on brand servers.

# Hardware tracking

No matter which tracking system is used, the goal of hardware tracking is to get information about the components used throughout the Proxmox and Ceph cluster environment. Information such as the serial number, brand/model, due date of components' replacement, and so on must be close on hand of any network manager or administrator. Some companies have a tracking system developed in house, some use a professional inventory tracking system, and some use a plain simple spreadsheet to keep track of the installed components. Although a spreadsheet is a quick and easy way to keep component information handy, it can quickly get out of hand in a cluster environment with several dozen nodes. Have a look at the following excerpt of a simple spreadsheet used by an administrator in his or her production environment:



**Proxmox / CEPH Hardware Tracking**

| | Motherboard | | CPU | | Cores | | RAM | | IP Address | Supplier | Warranty |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Node | Brand/Model | Serial Number | Brand/Model | Serial Number | Core | Thrd | Max | In | | | |
| Proxmox 1 | Asus P8B75-M/CSM | DAM0CS163143 | Intel i7-3770 3.5 1155 | MC239230A2312 | 4 | 8 | 32 | 32 | 192.168.10.1 | ABC | 11/29/2015 |
| Proxmox 2 | Asus P8B75-M/CSM | DAM0CS163146 | Intel i7-3770 3.5 1155 | 2R252508A0516 | 4 | 8 | 32 | 32 | 192.168.10.2 | ABC | 11/29/2015 |
| Proxmox 3 | Asus P8B75-M/CSM | DAM0CS163213 | Intel i5-3570 3.4 1155 | 2D35446522323 | 4 | 4 | 16 | 16 | 192.168.10.3 | XYZ | 11/29/2017 |
| Proxmox 4 | Asus B85M-E/CSM | DAM0AB445174 | Intel i7-4770 3.4 1150 | 2L326367A2639 | 4 | 8 | 32 | 32 | 192.168.10.4 | ABC | 1/27/2016 |
| Proxmox 5 | Asus P8B75-M/CSM | DAM0CS163453 | Intel i7-4770 3.4 1150 | SLS9082349083 | 4 | 8 | 32 | 32 | 192.168.10.5 | ABC | 1/27/2016 |
| Proxmox 6 | Asus P8B75-M/CSM | DAM0CS112342 | Intel i7-4770 3.4 1150 | SLS0933031139 | 4 | 8 | 32 | 32 | 192.168.10.6 | ABC | 1/27/2016 |
| Proxmox 7 | Asus P8B75-M/CSM | DAM0CS165657 | Intel i7-4770 3.4 1150 | SLE0393092342 | 4 | 8 | 32 | 32 | 192.168.10.7 | XYZ | 10/21/2015 |
| | | | | | 28 | 52 | 208 | 208 | | | |
| Backup 01 | MSI FM2-A55M-E33 | 601-7721-030B234 | AMD A4-6300 3.9 | 9AH9264I30807 | | | 32 | 8 | 192.168.100.1 | ABC | 4/10/2017 |
| Backup 02 | MSI FM2-A55M-E33 | 601-7721-030B131 | AMD A4-6300 3.9 | 9T09314G30641 | | | 32 | 8 | 192.168.100.2 | ABC | 4/10/2017 |
| ceph-01 | Intel S1200BTLR | 982749874523 | Intel i3-3220 3.3 1155 | DLKJG45LKG34 | | | 32 | 8 | 192.168.40.41 | XYZ | 1/10/2017 |
| ceph-02 | Intel S1200BTLR | 094E09834544 | Intel i3-3220 3.3 1155 | DLK9759345034 | | | 32 | 8 | 192.168.40.42 | ABC | 11/29/2015 |
| ceph-03 | Intel S1200BTLR | 45L345L54SDF | Intel i3-3220 3.3 1156 | DLK2349823001 | | | 32 | 8 | 192.168.40.43 | ABC | 11/29/2015 |
| | | | | | | | 96 | 24 | | | |

**CEPH Cluster HDD Serials**

| | | | | | |
|---|---|---|---|---|---|
| ceph-01 | | | | | TB |
| | Z4N00G1L - osd.12 | Z4N00SE40 - osd.15 | W1EFDGHT - osd.18 | G4E3YWRM - osd.21 | |
| | Z4N00T1L - osd.0 | Z4N00SF0 - osd.3 | W1E3YWRM - osd.6 | W1E3YWRM - osd.9 | 16 |
| ceph-02 | | | | | |
| | Z4N00F5L - osd.13 | Z4N00SL9 - osd.16 | W2G3YWRM - osd.19 | W1D6JWRM - osd.22 | |
| | Z4N00TZV - osd.1 | Z4N00RV1-osd.4 | W1E3YASD - osd.7 | Z4N00QY0 - osd.10 | 16 |
| ceph-03 | | | | | |
| | Z4N00D4G - osd.14 | Z4N00F4M - osd.17 | W1E3YLOY - osd.20 | F6E3YWRM - osd.23 | |
| | Z4N00S8L - osd.2 | Z4N00T2P - osd.5 | W1E3AQ6RM - osd.8 | Z4N00AK7 - osd.11 | 16 |
| | | | | Total Storage Space | 48 |

The first portion of the spreadsheet has information of all the nodes that include Proxmox, Backup, and Ceph nodes. The second portion of the spreadsheet has information of all the hard drives installed into each 12-bay Hot Swap Ceph node. We can see that eight bays out of 12 have hard drives with the OSD number and the serial numbers related to each hard drive. If we need to find a particular serial number, all we have to do is open the **Find** option and search for the serial number related to a component.

There are also open source options for inventory tracking such as vTiger and Spiceworks. vTiger and Spiceworks both have an asset-tracking option. For a robust network environment with several dozens of nodes and a couple of hundred SSDs or HDDs, the only wise option is to get a real asset-tracking system such as WASP Asset Tracking Software from Wasp Technologies (`http://www.waspbarcode.com/asset-tracking`). The paid version of a tracking program has a lot more polished features and is able to handle hardware in different categories and levels.

# AMD-based hardware selection

Although we recommend Intel-based hardware for a high-demand production environment, in many cases, AMD can take on the challenge at a reduced cost without sacrificing too much on performance and stability. It should be noted that for a mission-critical environment, an Intel-based platform should be considered as the first choice. In the following sections, there are some hardware components that are actually used in some production environments without any major issue.

# An AMD-based entry-level Proxmox

The following image shows an AMD-based CPU and motherboard that can be used for an entry-level Proxmox node. The hardware details are as follows:

- **CPU**: AMD FX-9370 4.4Ghz AM3+.

- **Motherboard**: Gigabyte GA-990FXA-UD5. The maximum amount of memory supported is 32 GB.

http://www.gigabyte.com/products/product-page.aspx?pid=3891#ov

With an AMD-based desktop motherboard, the option to install over 32 GB of memory is extremely limited. AMD FX-9370 delivers almost the same performance as that of Intel i7-4820K, but at a reduced cost.

# An AMD-based advanced-level Proxmox

The following image shows an AMD-based CPU and motherboard and an advanced-level Proxmox node. The hardware details are as follows:

- **CPU**: AMD Opteron 6308
- **Motherboard**: Asus KGPE-D16 Dual G34

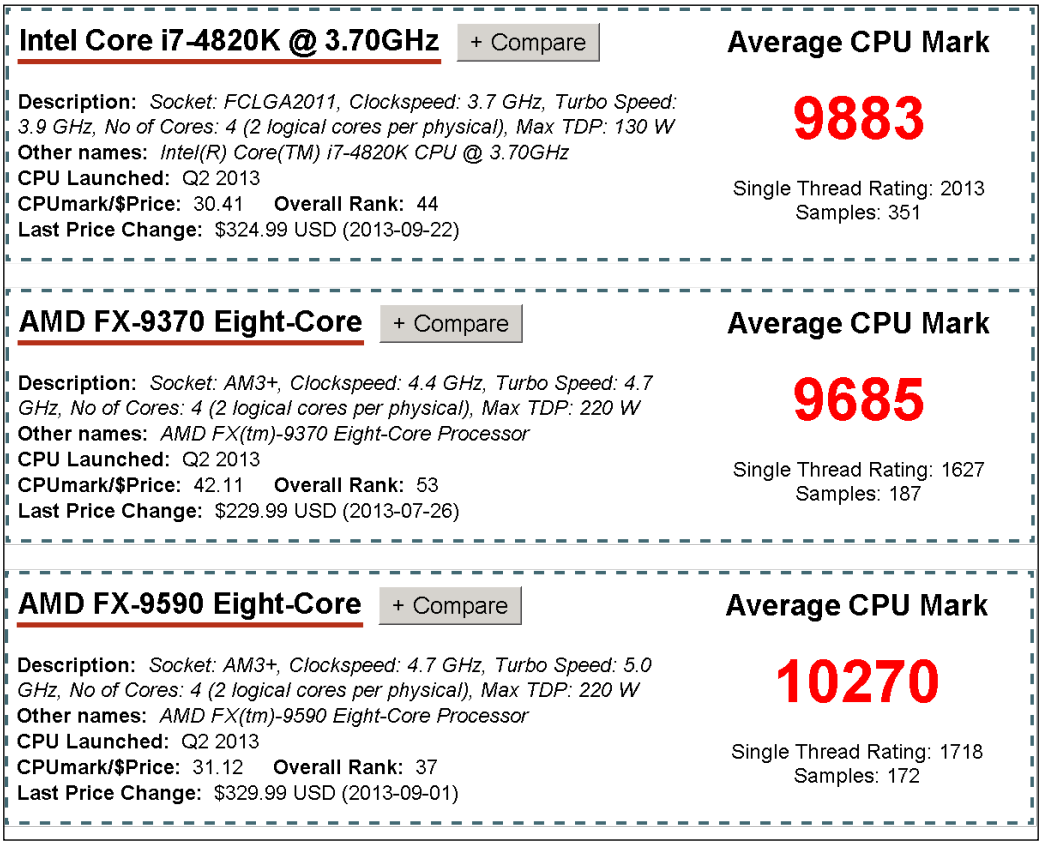
http://www.asus.com/Commercial_Servers_Workstations/KGPED16/

# An AMD-based Ceph setup

Since a Ceph node uses a RAID controller and expander card, using an AMD-based component is not recommended. On numerous occasions, we have noticed there is an incompatibility issue with the RAID controller card, AMD chipset, and Ceph. Issues such as not recognizing a RAID card are very common.

# Performance comparison

Courtesy `www.cpubenchmark.net`, the following image shows a comparison of Intel and AMD high-end desktop-class CPU performance:




Clearly, AMD has an advantage over Intel in the price arena. AMD-9590 has a higher performance rating than Intel i7-4820K at about the same price. On the other hand, AMD-9370 has a very close performance rating as compared to i7-4820K at a much lower cost.

# Summary

There are no one-setup-fits-all solutions when it comes to a server node setup. The components described in this chapter are mere examples of the possible setups. However, this should provide you with a baseline to direct your thinking when choosing components for the cluster environment you are responsible for. We hope this will aid you in your quest to find that perfect balance between performance and budget that all network administrators crave for.

In the next chapter, we are going to see some real incidental-based issues and how to troubleshoot a Proxmox cluster environment. These issues have been taken from day-to-day running clusters in production scenarios.

# 9
# Proxmox Troubleshooting

In this chapter, we are going to learn about some real-life issues taken from Proxmox's production environment and their solutions. Once a Proxmox cluster is set up, it usually runs without issues. However, when issues arise, a system administrator's knowledge is tested. Learning about troubleshooting is like learning from other people's mistakes. Throughout this chapter, we will gain some insight into Proxmox troubleshooting and hope that when we face these in our own clusters, we will know what do at a moment's notice.

All the issues explained in this chapter are those few that you may come across in your cluster environment. There could be more that are not covered in this chapter. It is just not possible to mention every possible issue that a Proxmox cluster can face. As you run your own cluster, you will have more unexpected issues that you will add to this list. It is a common practice for an administrator or operator to keep a documentation to enter new issues and their solutions. That way, we do not have to research a particular issue all over again.

The issues are divided into the following sections:

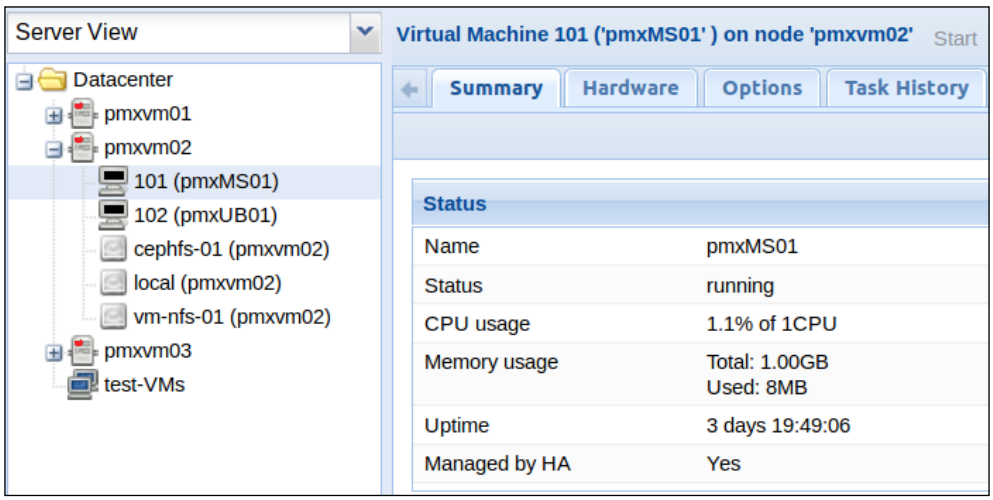
- The main cluster
- Storage
- Network connectivity
- The KVM virtual machine
- OpenVZ containers
- Backup/restore
- VNC/SPICE console

# Main cluster issues

This section contains issues related to the main Proxmox's cluster operations.

## GUI shows everything is offline

The scenario where everything in the GUI is shown offline is usually caused when one of the three services such as `pvedaemon`, `pvestatd`, or `pveproxy` crashes or stops working for any number of reasons. This situation usually looks as shown in the following screenshot:




In the previous screenshot, both virtual machines and all the three Proxmox nodes are running, but the Proxmox GUI shows everything is offline. Simply restarting them through SSH will fix this issue. Run the following commands in the given order:

```
# service pvedaemon restart
# service pvestatd restart
# service pveproxy restart
```

We can also check whether the services are running or not using the following command:

```
# ps xa | grep pve
```

If the GUI keeps showing all the nodes and VMs offline, check the syslog of nodes to spot any errors or warnings. In some cases, a failed attached storage will also cause the GUI to show that everything is offline. In such cases, check whether the storage node is functioning properly and then run restart the `pvedaemon`, `pvestatd`, and `pveproxy` services.

# Rejoining a Proxmox node with the same IP address

If you are rejoining a Proxmox node back to the cluster with the same IP address, then the joining command must run with the `--force` option. Run the following command from the node that is being rejoined:

```
# pvecm add <any_proxmox_node_ip> -force
```

# Disabling fencing temporarily

At times, it is necessary to disable fencing to diagnose an issue. There is no option in GUI to disable fencing. A simpler way to do this is to remove or comment out the fencing device for the node from the `cluster.conf` file, as shown in the following code:

```
<cluster config_version="16" name="pmx-cluster">
  <cman keyfile="/var/lib/pve-cluster/corosync.authkey"/>
  <fencedevices>
    <fencedevice agent="fence_apc" ipaddr="192.168.145.250"
      login="pmxfence" name="apc" passwd="99999" power_wait="10"/>
  </fencedevices>
  <clusternodes>
    <clusternode name="pmxvm01" nodeid="1" votes="1">
      <fence>
        <method name="power">
          <device name="apc" port="1" secure="on"/>
        </method>
      </fence>
    </clusternode>
    <clusternode name="pmxvm02" nodeid="2" votes="1">
 #     <fence>
 #       <method name="power">
 #         <device name="apc" port="2" secure="on"/>
 #       </method>
 #     </fence>
    </clusternode>
    <clusternode name="pmxvm03" nodeid="3" votes="1">
    </clusternode>
  </clusternodes>
  <rm/>
</cluster>
```

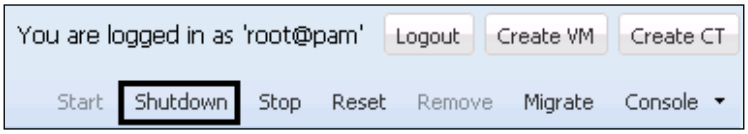
# The occurrence of kernel panic when disconnecting USB devices

On rare occasions, kernel panic occurs in Proxmox when disconnecting USB devices such as keyboard, mouse, or UPS. There is no real solution to this issue yet as the issue is not reproducible all the time. This issue has been seen on variety of hardware with both standard and nonstandard Proxmox installations. However, almost all the time, the issue does not cause the server to freeze permanently. In almost all cases, the panic can be just ignored and you can go on as usual.

Kernel panic seems to mostly occur with kernel 2.6.32-26, 2.6.32-27, and 2.6.32-28. It is nonexistent on kernel 3.2 or 3.10. For regular day-to-day operations of a cluster, this issue can be safely ignored unless it causes the node to freeze on rare occasions.

# The occurrence of VM shutdown error when initiated from GUI

The VM shutdown error issue is not consistent and is not directly related to Proxmox. The **Shutdown** button on Proxmox's GUI, as seen in the following screenshot, only sends an ACPI signal to a virtual machine to initiate the shutdown process.




Once the VM receives an ACPI signal, it starts the shutdown process. However, if the VM has a number of running processes in the memory, it might take a while to end the processes before the shutdown. The ending of processes may take longer, which causes Proxmox to issue a timeout error. The issue may occur for both Windows and Linux. The workaround for this is to access the VM through a console or SPICE and then manually shut down the VM. Also, check whether the guest VM has ACPI installed and loaded to correctly receive the poweroff signal from the host node.

# Kernel panic on Proxmox 3.2 with HP NC360T

In some cases, on Proxmox 3.2, kernel panic occurs when you use a network interface based on an Intel 82571EB chipset, such as HP NC360T. An immediate workaround is to use broadcom for the network interface card. The permanent fix is to download E1000 drivers from the Intel website and compile a module from those sources. The E1000 driver can be downloaded from `http://www.intel.com/support/network/sb/cs-006120.htm`.

# VMs not booting after you restart the network service

On a very rare occasion, virtual machines do not power up after you restart the network service on a Proxmox node. This occurs when corosync and rgmanager services are not automatically loaded when you reboot the Proxmox node. Starting these services will allow the VMs to start again. Use the following commands to start corosync and rgmanager:

```
# corosync
# service rgmanager start
```

# Proxmox cluster is out of Quorum and cluster filesystem is read only

The error Proxmox cluster is out of Quorum and cluster filesystem is read-only occurs when the Proxmox node is not included in Quorum. To prevent any occurrence of this error in cluster configuration files, Proxmox puts the cluster filesystem in the read-only mode for the node in question. Run the following commands from the node with this issue. We have to stop the cluster service, start it in local mode, delete or move the existing `cluster.conf` file, and then restart the cluster. A new `cluster.conf` file will be synced with the node with a read-only issue. Perform the following steps to overcome this issue:

1. Stop the cluster in the node using the following command:

   ```
   # /etc/init.d/pve-cluster stop
   ```

2. Start the cluster filesystem in local mode using the following command:

   ```
   # /usr/bin/pmxcfs -l
   ```

3. Remove or back up the `cluster.conf` file using the following command. In this example, we created a backup directory named `backup` under `/home`.

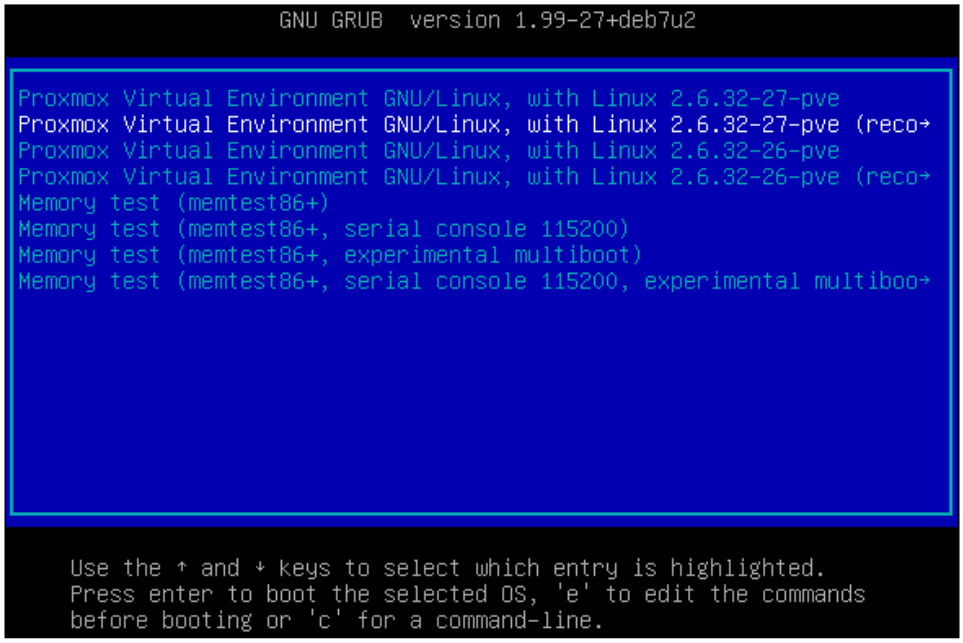
   ```
   # mv /etc/pve/cluster.conf /home/backup
   ```

4. Stop and start the cluster normally using the following commands:

   ```
   # /etc/init.d/pve-cluster stop
   # /etc/init.d/pve-cluster start
   ```

# Proxmox boot failure due to the getpwnam error

Boot the Proxmox node in recovery mode using the Proxmox installation disk, or select the **recovery** option from Proxmox's boot menu at the beginning of the boot process. Recovery usually happens on the second line of Proxmox's boot menu, as shown in the following screenshot:




After the recovery shell is loaded, run the following command from the command prompt and then reboot:

```
# apt-get update && apt-get dist-upgrade
```

# Cannot log in to GUI as ROOT

Sometimes the Proxmox GUI may prevent a root login through a browser after you reinstall Proxmox on the same node. In order to log in as **root** into the Proxmox GUI, local loopback must be enabled in the network interface file. Look for the following two lines to make sure they are not commented out in `/etc/network/interfaces`:

```
auto lo
iface lo inet loopback
```

# Booting with a USB stick fails in Proxmox

The recommended storage to install Proxmox is SSD or HDD. The only way to install it on a USB stick is to install Debian Linux first and then install Proxmox on top of it by adding the Proxmox VE repositories and installing packages.

A USB thumb drive or flash drive should never be used as the primary Proxmox OS installation storage in a production environment.

# The Upgrade from Proxmox 3.1 to Proxmox 3.2 is disabled through GUI

There are three most common reasons why the **Upgrade** button could be disabled on Proxmox GUI. Check the following alternatives to fix this issue:

- If the node does not have a valid subscription, ensure that the pve-no-subscription repository is added. For Proxmox repository information, visit `https://pve.proxmox.com/wiki/Package_repositories`.
- Refresh the browser cache.
- A very basic mistake, but not unheard of, is to make sure the root user is logged in to facilitate the upgrade. The **Upgrade** button is only visible when you log in with the root privilege.

# VZ kernel 2.6.32-28-pve breaks libnl/netlink in host and VM

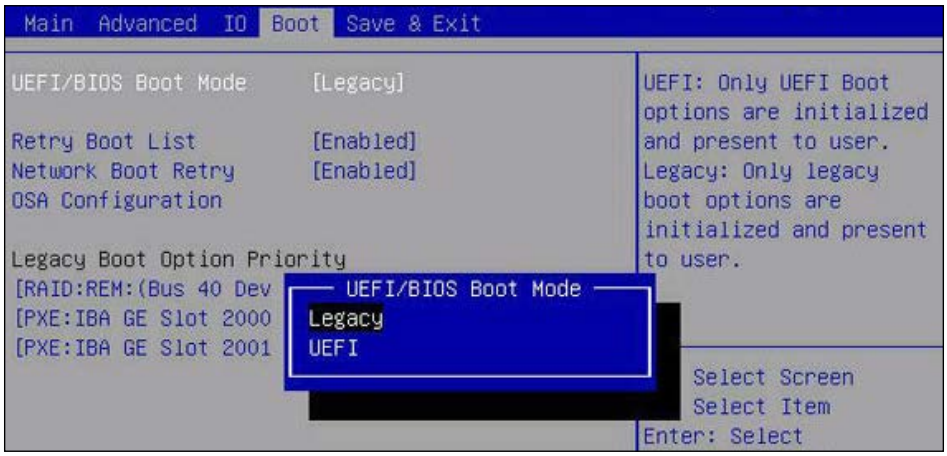
A kernel that breaks `libnl/netlink` in host and VM is an error caused by a known issue in `libnl` and not directly related to Proxmox. The `libnl` library is known to work with `vzkernel-2.6.32-042stab084.20.src.rpm` but not with newer kernels, including the latest testing kernel, namely, `vzkernel-2.6.32-042stab088.4`. At the time of writing this, the bus issue is open in a ticket at `https://bugzilla.openvz.org/show_bug.cgi?id=2939`.

# Nodes not visible on the Proxmox GUI after an upgrade

Proxmox nodes may not be visible after you upgrade to Proxmox VE 3.2. It is a common issue when accessing the Proxmox GUI with the Internet Explorer 11 browser. Use Firefox or the Chrome browser to correctly view the Proxmox GUI. There is a bug report filed for this known issue that is available at `https://bugzilla.proxmox.com/show_bug.cgi?id=475`.

# GRUB is in an endless loop after Proxmox installation

After installing Proxmox, GRUB may go in an endless loop, which is a common occurrence when Proxmox is installed on a newer desktop class with UEFI BIOS. Simply disabling the UEFI mode and enabling the Legacy mode in BIOS will allow the system to boot. This option interface varies from motherboard to motherboard, but usually it resembles the following screenshot:




If this does not work, Proxmox should be installed manually over Debian Wheezy. To get instructions on how to install Proxmox when the ISO installer does not work, visit `http://pve.proxmox.com/wiki/Install_Proxmox_VE_on_Debian_Wheezy`.

# SSH access is possible but Proxmox node does not reboot

On numerous occasions, a Proxmox node may not reboot when the process of rebooting is initiated through GUI or from the command prompt. In such cases, the only option available is to physically power cycle a node. In almost all the cases, the node can still be accessed through SSH. If the node is on PDU, it can be forcefully power cycled remotely. However, for those nodes that are not on PDU and physical access is not possible right away, there are the following two methods that you can use to force the process of rebooting the node.

- In the first method, you can log in to the node through SSH. Then, you simply apply the `reboot` command once or twice:

  ```
  # reboot
  ```

- In the second method, you can run the following commands from the command prompt if the `reboot` command has no effect:

  ```
  # echo 1 > /proc/sys/kernel/sysrq
  # echo b > /proc/sysrq-trigger
  ```

Please note that these commands will forcefully reboot a Proxmox node and any running VMs will be forcefully shut down. These may cause data loss or corruption within a VM. Before rebooting a Proxmox node, always make sure that there are no VMs running or VMs have been migrated to a different node.

# Storage issues

This section contains issues related to storage systems supported by Proxmox, such as local, NFS, Ceph, GlusterFS, and so on.

# Deleting damaged LVM with error read failed from 0 to 4096

Sometimes it may be necessary to remove damaged LVM storages from a cluster. Run the following command from the CLI to remove such LVMs. Please be aware that it will remove the LVM permanently.

```
# dmsetup remove /dev/<volume_group>/<lvm_name>
```

# Proxmox cannot mount NFS share due to time-out error

Some NFS servers such as FreeNAS do a reverse lookup for hostnames. We have to add Proxmox hostnames to the host files of the NFS server. Use any editor of your choice to open the `hosts` file:

```
# nano /etc/hosts
```

Add Proxmox nodes' hostnames as in the following examples:

```
192.168.145.1    pmxvm01.domain.com        pmxvm01
192.168.145.2    pmxvm02.domain.com        pmxvm02
192.168.145.3    pmxvm03.domain.com        pmxvm03
192.168.145.4    pmxvm04.domain.com        pmxvm04
```

# Removing stale NFS shares when a stale file handle error occurs

When NFS shares are deleted from Proxmox storage, there are cases where they still remain mounted; this causes the NFS stale file handle error. Simply manually unmounting the share and removing the NFS mount point folder from the Proxmox directory will fix this issue. Run the following commands from the Proxmox node:

```
# umount -f /mnt/<nfs_share>
# rmdir <nfs_share>
```

# The occurrence of '--mode session exit code 21' errors while accessing iSCSI target

Run the following command from the Proxmox node to fix the error:

```
# iscsiadm -m node -l ALL
```

# Cannot read an iSCSI target even after it has been deleted from Proxmox storage

When trying to read the same iSCSI target after it had been deleted from Proxmox storage, an error occurs mentioning the target that has already been added to Proxmox. In these cases, the iSCSI daemon has to be restarted to clear this issue. Run the following command from all the Proxmox nodes. Beware that this command will disconnect all the existing iSCSI targets:

```
# /etc/init.d/open-iscsi restart
```

# OSDs still show up in Proxmox after you remove the Ceph node

This is a common occurrence when a Ceph node is taken offline without removing all the Ceph-related processes first. The OSDs in the node must be removed or moved to another node before taking the node offline. Run the following commands to remove OSDs:

```
# ceph osd out <osd.id>
# ceph osd crush remove osd <osd.id>
# ceph auth del osd.<id>
# ceph osd rm <osd.id>
```

# The 'No Such Block Device' error that shows up during creation of an OSD

When creating an OSD through the Proxmox GUI on Proxmox 3.2, sometimes this error occurs. This is not a common occurrence and is not reproducible at all times. Although there are no permanent fixes for this issue, it can be ignored. So just retry to create an OSD. Keep in mind that the inclusion of Ceph is a technology preview in Proxmox. Minor glitches are to be expected. This is not a shortcoming of Ceph itself, but the attempt of combining Proxmox and Ceph in the same node.

# The fstrim command does not trim unused blocks for Ceph

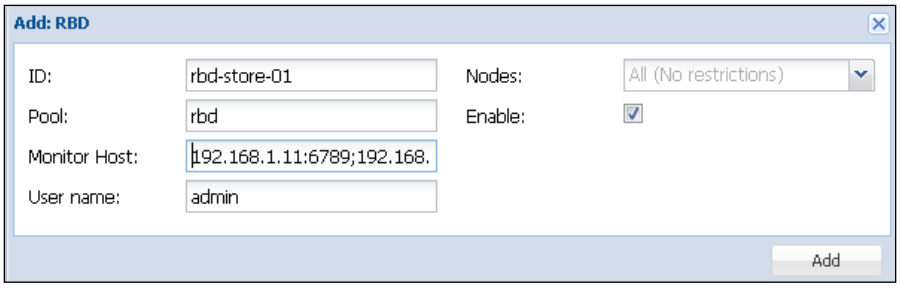
To properly trim unused blocks for virtual disks stored on the Ceph storage, perform the following steps:

1. Use a virtio disk type for a virtual disk.
2. Enable the discard option through `<vm_id>.conf`. Add `discard=on` to the drive properties of virtio0 as shown in the following command line:

```
<rbd_storage>:<virtual_disk>,cache=writethrough,size=50G,
  discard=on
```

# The 'RBD Couldn't Connect To Cluster (500)' error when connecting Ceph with Proxmox

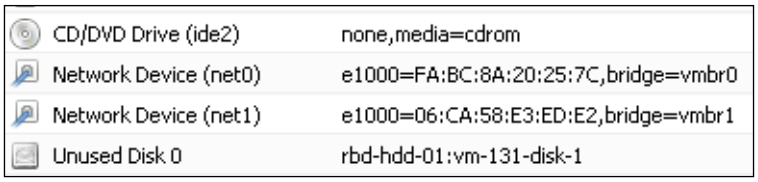
Authentication failure is the most common cause for this error when Ceph RBD storage cannot connect to Proxmox. Proxmox requires a copy of the Ceph admin keyring to authenticate. The name of the keyring must match with the storage ID assigned through a Proxmox GUI as shown in the following screenshot:



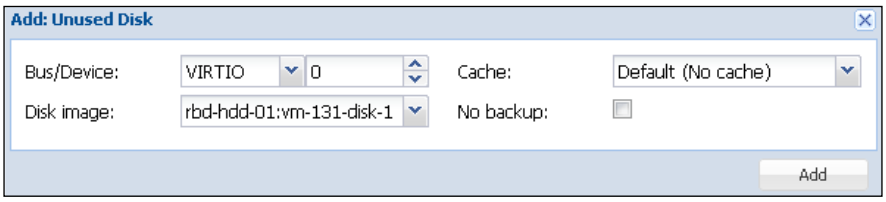

In the previous example, the Ceph admin keyring name must match with the storage ID `rbd-store-01` and stored as `/etc/pve/priv/ceph/rbd-store-01.keyring`.

# Changing the storage type from ide to virtio

If IDE was used during the initial VM setup and if it needs to be changed to virtio later, it can be done through the Proxmox GUI without reinstalling the OS. The VM will need to be powered off first and then the virtual disk needs to be removed through the Proxmox GUI. Clicking on **Remove** will cause the virtual disk to become unused, as shown in the following screenshot:




Double-click on the unused virtual disk to add it back to the VM. Select **VIRTIO** from the dialog box as shown in the following screenshot:




# The 'pveceph configuration not initialized (500)' error for the Ceph tab

Non-initialization of the `pveceph` configuration is an error that occurs when we click on the **Ceph** tab in the Proxmox GUI without initializing the Ceph storage. If Ceph is not going to be used along with Proxmox on the same cluster on the same Proxmox node, then this error should be simply ignored. Refer to *Chapter 7*, *High Availability Storage for High Availability Cluster*, to initialize the Ceph server on the same Proxmox node.

# Ceph FS storage disappears after a Proxmox node reboots

Ceph FS needs to be mounted in order to make it available for storage service. If the mount point is not set in `/etc/fstab`, it will need to be remounted after each reboot. The following format is used to enter Ceph FS in `/etc/fstab`:

```
id={user-ID}[,conf={path/to/conf.conf}] /mount/path  fuse.ceph
  defaults 0 0
```

Based on this format, we can add the following line:

```
id=admin,conf=/etc/ceph/conf.conf /mnt/<path>  fuse.ceph defaults
  0 0
```

# VM cloning does not parse in Ceph storage

When full cloning is performed on a virtual machine stored on Ceph storage, the virtual disk looses parsing. This means that when parsing for VM cloning, Proxmox uses the `qemu-img` method instead of rbd flattening. Until flattening is implemented in later versions of Proxmox, VM clones will lose parsing when stored on Ceph storage.

# Network connectivity issues

This section contains issues related to virtual or physical network connectivity within Proxmox.

# No connectivity on Realtek RTL8111/8411 Rev. 06 NIC

A network interface based on Realtek RTL8111/8411 revision 06 seems to be up but with no connectivity. The issue is that some newer Realtek chipsets don't get compiled with the right drivers. This causes the interface to be up without any network traffic. In order to fix this issue, the older driver needs to be downloaded from the Realtek site and compiled manually. The driver can be downloaded from `http://www.realtek.com.tw/Downloads/`.

Since this driver is manually installed, during a kernel update it will get updated automatically. To prevent this and ensure the driver builds itself automatically when a new kernel is installed, run the following commands and then reboot the node:

```
# apt-get install dkms build-essentials pve-headers-2.6.32-25-pve
# cat <<EOF > /usr/src/r8168-8.037.00/dkms.conf
PACKAGE_NAME=r8168
PACKAGE_VERSION=8.037.00
MAKE[0]="'make'"
BUILT_MODULE_NAME[0]=r8168
BUILT_MODULE_LOCATION[0]="src/"
DEST_MODULE_LOCATION[0]="/kernel/updates/dkms"
AUTOINSTALL="YES"
EOF
```

## Network performance is slower with e1000 vNIC

The performance of e1000 virtual network interfaces is about 30-35 percent lesser as compared to virtio virtual network interfaces. Changing vNICs to virtio will increase the overall network bandwidth of a virtual machine. The virtio drivers are included in all major Linux flavors. For Windows machines, an ISO file with virtio drivers can be downloaded from `http://www.linux-kvm.org/page/WindowsGuestDrivers/Download_Drivers`.

# KVM virtual machine issues

This section contains issues related to KVM virtual machines only.

## Windows 7/XP machine converted to Proxmox KVM hangs during boot

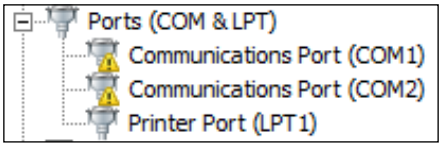
The Windows operating system can be unforgiving when you convert or migrate from one type of hardware to another. It is certainly possible to convert/migrate just about any Windows OS as long as proper procedure is followed. For in-depth information on the proper procedure to migrate Windows machines to a virtual machine, visit the Proxmox Wiki at `http://pve.proxmox.com/wiki/Migration_of_servers_to_Proxmox_VE#mergeide`.

## Windows 7 VM only boots when rebooted manually

This issue causes a Windows 7 virtual machine to shut down when initiated reboot from within OS. A manual power on through the Proxmox GUI is required to power up the VM. This is an issue caused by the installation of Windows itself, especially a VM that is configured with a standard video. Changing the display to SPICE solves the issue for a Windows 7 virtual machine. This is not a common occurrence and causes an issue in some Windows 7 VMs, while others just run fine.

# The Proxmox 3.2 upgrade adds two com ports and one parallel port to the Windows VM

After you upgrade a node to Proxmox VE 3.2, all Windows-based VMs add two com ports and one parallel port as shown in the following screenshot:




The issue seems to originate due to the new `pve-qemu-kvm` package. Downgrading it to `pve-qemu-kvm-1.4.-17` will fix this issue. As of this writing, there is no updated `pve-qemu-kvm` package to fix the issue permanently.

# The qemu-img command does not convert the .vmdk image files created with the .ova template in Proxmox VE 3.2

We can usually convert any `vmdk` disk image file with the following command:

```
root@pve:~# qemu-img convert -f vmdk disk1.vmdk -O qcow2 vm-101-disk-
  1.qcow2
```

However, the `.vmdk` image files created with VMware's `.ova` template may present one or more of the following error messages while converting it with the `qemu-img` command:

```
qemu-img: 'image' uses a vmdk feature which is not supported by this
  qemu version: VMDK version 3
qemu-img: Could not open 'disk1.vmdk': Could not open 'disk1.vmdk':
  Wrong medium type
qemu-img: Could not open 'disk1.vmdk'
```

The `.vmdk3` format is only supported in `pve-qemu-kvm 2.0`. Enter the following command to check the version installed in the Proxmox node:

```
# pveversion -v
```

Look for the version number of `pve-qemu-kvm`. A `.vmdk3` file can still be converted to `qcow2` by following the instructions given at `http://carlos-spitzer.com/2013/12/26/how-to-convert-vmdk-3-images-to-qcow2-even-when-qemu-img-fails-o/`.

# Online migration of a virtual machine fails with a 'Failed to sync data' error

In order to migrate virtual machines online without powering them off, the virtual disk of the VM must be on a shared storage system. Any VM with a virtual disk on local storage cannot be migrated live. The error will look as follows:

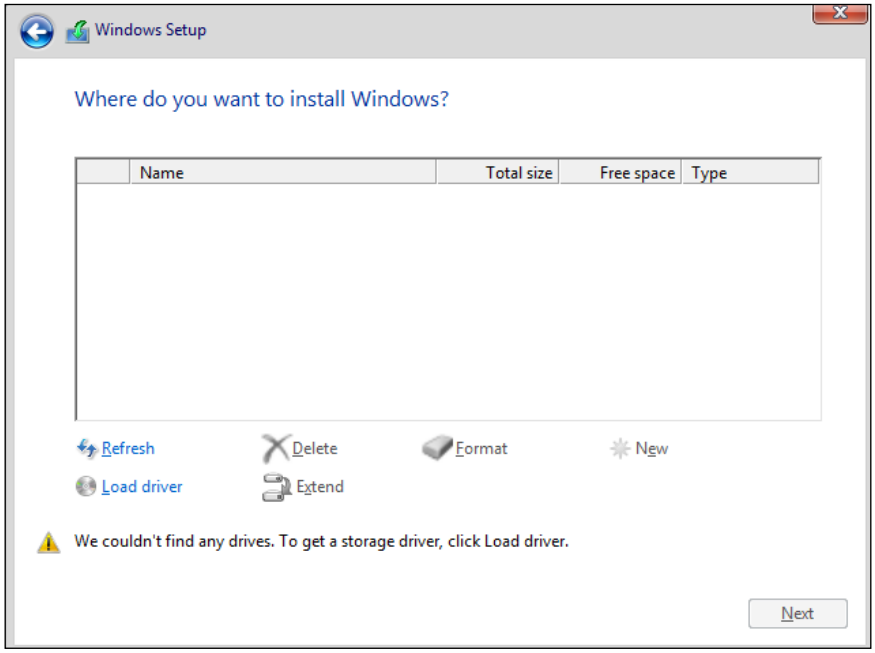
```
Mar 18 19:54:37 starting migration of VM 5134 to node 'titan'
  (192.168.10.1)
Mar 18 19:54:37 copying disk images
Mar 18 19:54:37 ERROR: Failed to sync data - can't do online
  migration - VM uses local disks
Mar 18 19:54:37 aborting phase 1 - cleanup resources
Mar 18 19:54:37 ERROR: migration aborted (duration 00:00:00): Failed
  to  sync data - can't do online migration - VM uses local disks
TASK ERROR: migration aborted
```

# Change in memory allocation is not initialized after a VM is rebooted

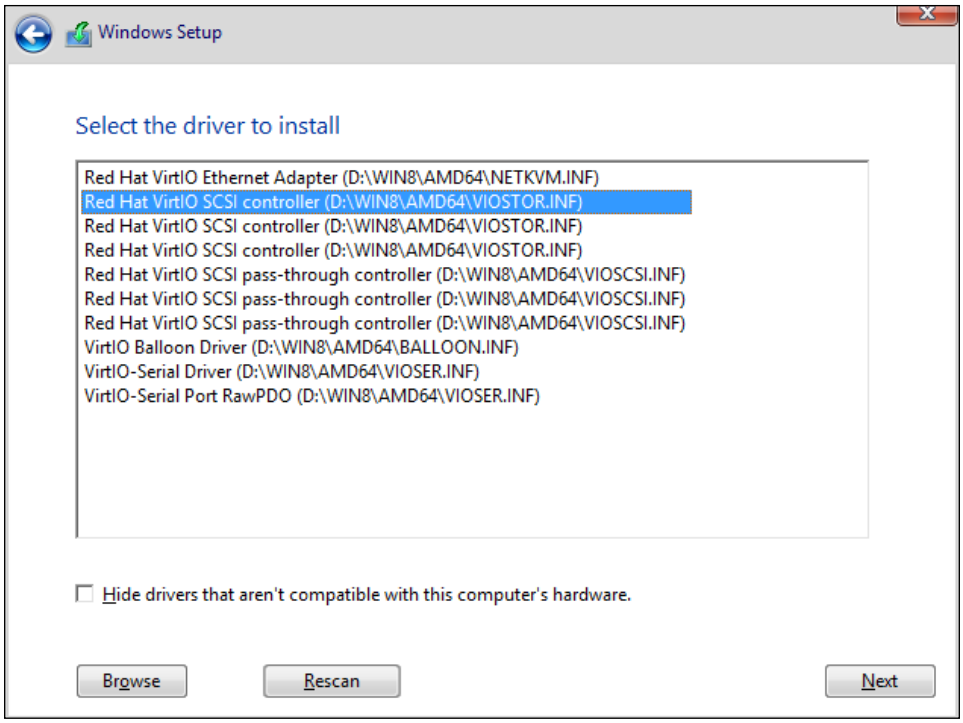
A virtual machine must be powered off to make changes to the memory, video, or virtual network interface. No changes will take effect if a VM is simply restarted.

# The virtio virtual disk is not available during the Windows Server installation

The virtio drivers are not included in the Windows Server installation. During the installation, the Windows setup will not see any virtio virtual disks attached with the virtual machine as shown in the following screenshot:




A virtio driver must be downloaded and loaded during the installation in order to activate the virtio virtual disk with the Windows operating system. The ISO image of virtio drivers can be downloaded from `http://www.linux-kvm.org/page/WindowsGuestDrivers/Download_Drivers`. The driver installation window is shown in the following screenshot:




# OpenVZ container issues

This section contains issues related to OpenVZ containers only.

## The creation of OpenVZ container takes a long time on NFS or GlusterFS storage

OpenVZ parameters depend on BeanCounters. Settings such as numproc, numtcpsock, kmemsize, numfile, tcpsndbuf, and tcprcvbuf in UBC need to be tweaked to increase the performance. Changes can be made to the container configuration file under `/etc/pve/openvz`. Visit the following links to get more details on the parameters used by OpenVZ containers:

- `http://wiki.openvz.org/UBC_primary_parameters`
- `https://wiki.openvz.org/UBC_secondary_parameters`

# OpenVZ containers are no longer shown after a cluster is created

This is a common case when a single node Proxmox was set up with an OpenVZ container, and then later when the actual cluster was created and the node in question was added to the cluster. Very likely, all the CT configuration files are in `/var/lib/vz/private`. Simply copying the CT configuration files from `/var/lib/vz/private` to `/etc/pve/openvz` will make them visible from the Proxmox GUI. The directory location `/etc/pve/` is synced over multiple nodes in the cluster to ensure all the nodes have identical configuration. Refer to *The Proxmox cluster directory structure* section in *Chapter 2*, *Proxmox Under the Hood*, for explanation on `/etc/pve`.

# Header error during the installation of PF_RING in Proxmox

While installing PF_RING to be used with an OpenVZ container, the installation is interrupted with the following error messages:

```
Configuring  securityonion-pfring-ld (20120827-
  0ubuntu0securityonion4)...
Configuring  securityonion-pfring-userland (20130828-
  0ubuntu0securityonion1)...
Configuring  securityonion-pfring-module (20121107-
  0ubuntu0securityonion10)...

Creating symlink /var/lib/dkms/pf_ring/5/source -> /usr/src/pf_ring-5

DKMS: add completed.
Error! Your kernel headers for kernel 2.6.32-26-pve cannot be found.
Please install the linux-headers-2.6.32-26-pve package,
or use the --kernelsourcedir option to tell DKMS where it's located
WARNING: Deprecated config file /etc/modprobe.conf, all config files
  belong into /etc/modprobe.d/.
FATAL: Module pf_ring not found.
dpkg: error processing securityonion-pfring-module (--configure):
subprocess installed post-installation script returned error exit
  status 1
```

The error occurs due to the lack of development headers. Simply installing them using the following command will fix the issue:

```
# apt-get install pve-headers-$(uname –r)
```

# Backup/restore issues

This section contains issues related to backing up and restoring Proxmox.

# A Proxmox VM is locked after backup crashes unexpectedly

A Proxmox VM getting locked is commonly caused after a VM backup is interrupted or crashed. Simply unlocking the VM through SSH using the following command will fix this issue:

```
# qm unlock <vm_id>
```

# Backing up only the primary OS virtual disk

By default, Proxmox backup will back up all the virtual disks assigned to a VM. If we want to exclude certain virtual disks from the backup process, we only need to add the option `backup=no` at the end of a virtual disk line item in `<vm_id>.conf`, as follows.

```
virtio0: rbd-hdd-01:vm-101-disk1,size=80G
virtio0: rbd-hdd-01:vm-101-disk2,size=200G,backup=no
```

In the previous example, the virtual machine has two virtual disks. Disk 1 is for the primary OS and disk 2 is for the secondary. By adding `backup=no`, Proxmox will skip this disk during the backup process and only back up the primary disk.

# Backup of VMs stops prematurely with an 'Operation Not Permitted' error

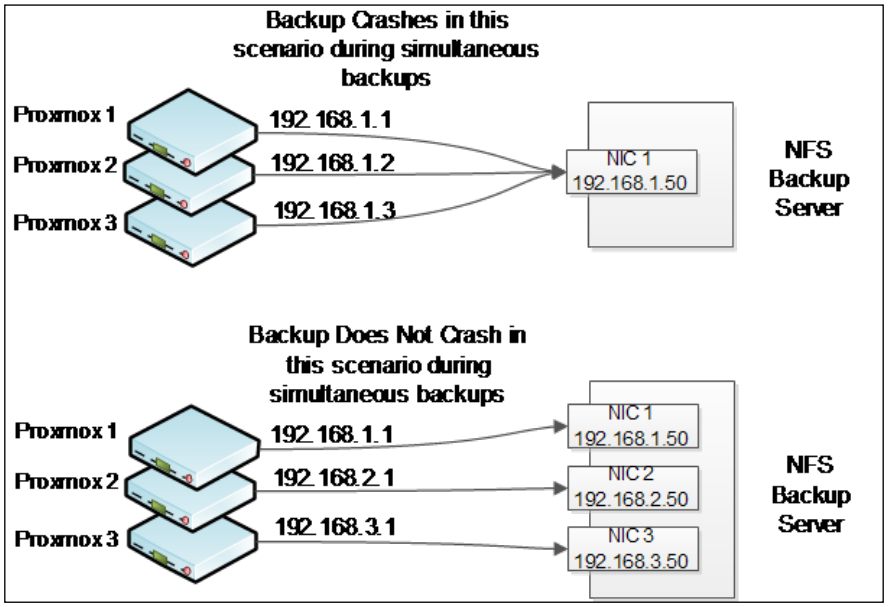
The premature stopping of VMs' backup usually looks like the following message that is taken from the syslog of a Proxmox node:

```
ERROR: job failed with err -1 - Operation not permitted
INFO: aborting backup job
INFO: stopping kvm after backup task
ERROR: Backup of VM 101 failed - job failed with err -1 - Operation not
permitted
```

The primary cause of this issue is that the backup storage has less space than the total storage required to perform a backup task. Verify the total storage space that is required for backing up the selected virtual machines.

# A backup task takes a very long time to complete, or it crashes when multiple nodes are backing up to the same backup storage

When multiple Proxmox nodes are backing up to same the backup storage simultaneously, it tends to take a very long time or the backup is crashed. This is a common occurrence when backup traffic coexists with the main cluster traffic on a gigabit network and the backup node only has one network interface. By separating backups in multiple subnets with multiple network interface cards, we can prevent this issue. Linking the aggregation of multiple network interfaces to form a bigger bandwidth, as shown in the following screenshot, can also help to avoid backup traffic congestion.




# Backup of virtual machines aborts a backup task prematurely

During a VM backup, the following error message appears in the backup log after it aborts a running backup task:

```
101: INFO: status: 1% (129309081/4294967296), sparse 0% (886784),
  duration 91, 33/33 MB/s
[...]
```

```
107: INFO: status: 80% (2706263244/4294967296), sparse 16%
  (698703462), duration 1950, 5/4 MB/s
107: ERROR: interrupted by signal
107: INFO: aborting backup job
```

This error usually occurs when there is a version mismatch for the `pve-qemu-kvm` package in Proxmox. As of this writing, the available `pve-qemu--kvm` package version is 1.7-8. Check for the version that is installed when you get this error during a backup using the following command:

```
# pveversion -v
```

If you're using an older version, then upgrade to the latest version using the following command to fix the issue:

```
# apt-get install pve-qemu-kvm
```

# Backup storage has a lot of .dat files and .tmp directories using the storage space

Due to a backup crash or unfinished backups, there may be backup leftover files in the backup storages such as the `.dat` files and `.tmp` directories. These files and folders can be easily deleted to reclaim storage space. Run the following command from the backup storage to find these files and directories:

```
root@pve:/ # find -name *tmp
root@pve:/ # find -name *dat
```

# VNC/SPICE console issues

This section contains issues related to the VNC and SPICE consoles in Proxmox.

# The mouse pointer is not shared with SPICE-VIEWER on Windows 8 VM

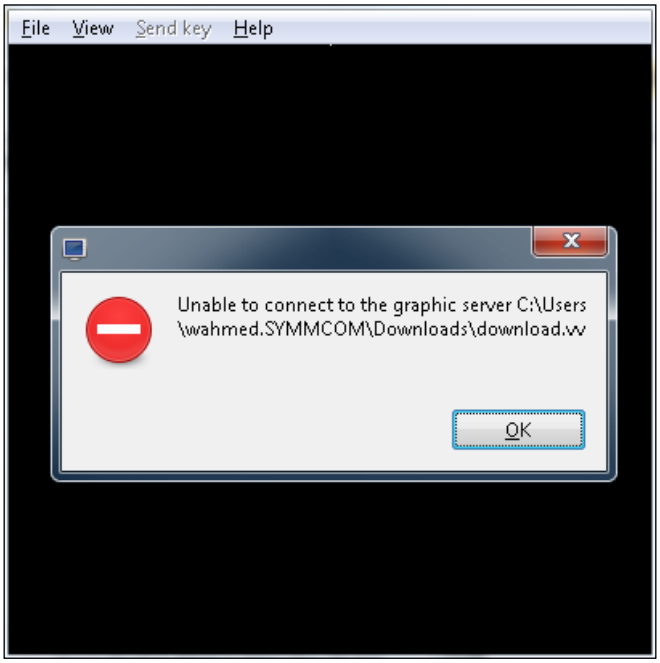
In order to have a seamless mouse point between the VM and host machine, SPICE Guest Tools must be installed inside the VM. The Guest Tools package contains full driver support for Windows 7 and Windows 2008 R2. However, the support for Windows 8 or 8.1 is close to nonexistent.

# The SPICE console has become unstable after the Proxmox VE 3.2 update

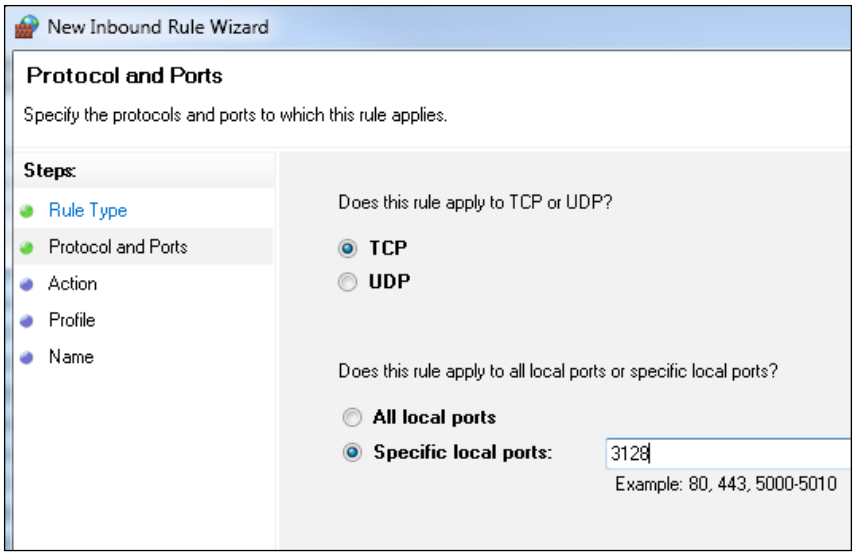
After clicking on the **SPICE** button for a SPICE-enabled VM, the browser gives an empty ticket error. This is a common occurrence if the browser cache is not updated after a Proxmox upgrade. Just refreshing the browser will fix this issue.

# Remote Viewer is unable to connect to a SPICE-enabled virtual machine on Windows OS

This issue is caused by a firewall that blocks the SPICE port, which prevents SPICE-enabled virtual machines to be connected to SPICE. The error message is shown in the following screenshot:




Open port 3128 from Windows firewall, as shown in the following screenshot, to allow Remote Viewer to connect to a SPICE virtual machine.




# Summary

We hope this troubleshooting chapter has provided you with some insight into some of the common issues that are most likely to surface in a Proxmox cluster. As mentioned earlier in this chapter, this is by no means a complete list of all the possible issues. If at all possible, always hold off major Proxmox upgrades for a production cluster. Give it some time to work out the bugs. This way your cluster will have a very little chance to go down due to any unforeseen bugs.

Acquiring Proxmox's subscription is the best way to ensure that there are fewer bugs in the repositories since Proxmox Enterprise repositories goes through an additional layer of scrutiny and testing. For information on Proxmox subscriptions, refer to `https://www.proxmox.com/proxmox-ve/pricing`.

The Proxmox forum is also a great place to ask for help or share issues with the community. There are many forum users who are ready to provide their expertise. Visit the forum at `http://forum.proxmox.com`.

In the next chapter, we are going to put our knowledge together through some networking scenarios where Proxmox can be used. The first part of the chapter will have the scenarios described, while the second part of the chapter will have network diagrams that show the scenarios in action.

# 10
# Putting It All Together

Equipped with all the knowledge we have gathered from the previous chapters in this book, we are now ready to put all the pieces together to form an advanced virtual environment. We can do this for just about any scenarios that we will be called for by proposing an adequate solution for a virtual environment. In the first part of this chapter, you are given a set of scenarios to build networks using Proxmox for various industries. In the second part of the chapter, you will find complete network diagrams of each scenario given in the first part of the chapter so that you can try to formulate the diagram on your own first, and then match it with the solutions provided.

Some scenarios have been taken from real-life production environments, and some are theoretical to show how you can build advanced networks with Proxmox. You can refer to these network models and use them as they are, or modify them to suit your needs better.

We hope that through these network scenarios and models, you will start seeing Proxmox from a whole new point of view and be fully prepared to face any level of virtual infrastructure you are going to be challenged with.

While analyzing these scenarios, keep in mind that the solutions and diagrams provided in this chapter are one of the many ways in which the network infrastructure could be set up. To fit the diagrams within the confinement of the book, some non-vital components such as physical switches, cabling, and so on may have been omitted.

These network diagrams show the relationship between components within an infrastructure, such as virtual environment, cluster of nodes, and overall network connectivity. They also represent virtual network components such as how bridges relate to each other, network segmentation, and so on.

The Ceph cluster has been used as a primary storage backend in all these scenarios. The latest Proxmox VE Version 3.2 supports co-existence of Proxmox and Ceph on the same node. This is a Proxmox technology preview in this version. This preview is solely the inclusion of Ceph in the Proxmox hypervisor and not the Ceph itself. Ceph is very much well established and stable. For the purpose of stability and any unknown bugs, we are going to use a separate Ceph cluster in network diagrams.

All IP addresses and hostnames used in this chapter are fictitious. Any resemblances to real networks or hostnames are purely coincidental.

# Scenario #1 – academic institution

This scenario is for a typical academic institution with multiple networks, multiple campuses, and a multibuilding setup with both private and public networks.

The following are the key requirements for this scenario:

- Network isolation to protect sensitive data.
- Ability to have centralized management for network infrastructure.
- Professors should be on a separate Wi-Fi network that only they have access to. This network should give them the ability to log in to the campus main server to retrieve their files for a lecture.
- Students should have on-campus Wi-Fi access and a wired Internet connection to their dormitories. These subnets must be separated from the main campus network.
- Library should be on a separate subnet with its own servers.
- Classrooms, admin offices, and professors should be on the main isolated network. Professors should have the ability to retrieve their files from the file server connected to the classroom computers during lectures.

This is a scenario of a typical campus network of an academic institution. Thanks to Proxmox, we can have all the main server equipments and virtual environments in one place to have centralized management. There are the following five subnets in this network:

| Subnet | Description |
|---|---|
| 192.170.10.0 | This is the wired network for the dormitory. Firewall provides DHCP. This subnet does not need to go through the main network. |
| 192.180.10.0 | This denotes the student and public Wi-Fi on campus. Firewall provides DHCP. This subnet does not need to go through the main network. |

| Subnet | Description |
|---|---|
| 192.160.10.0 | This is the main administrative and professors' network. Private Wi-Fi for professors' usage is an extension of this network to allow professors to retrieve their files in a wireless fashion. All classrooms are also on this network to provide in-class access of files for professors. |
| 192.110.10.0 | This is the storage cluster. |
| 192.190.10.0 | This is the library subnet. DHCP is provided by the virtualized library server. This server is for library usage only. Separate LAN (eth2) is used to connect the virtual machine with the library building. |

# Scenario #2 – multitier storage cluster using Proxmox cluster

The following are the key requirements for this scenario:

- Separate storage clusters for SSD, hybrid HDD, and HDD
- Storage clusters should be on separate subnets
- Storage should be distributed with High Availability and high scalability

For this scenario, each Proxmox node must have at least four network interface cards: three to connect three storage cluster subnets, and one to connect to Proxmox virtual environment. This example is for six virtual machines to have access to three different performing storages. The following table shows three Ceph clusters and their performance categories:

| Subnet | Description |
|---|---|
| 192.168.10.0:6789 | This is for Ceph cluster #1 with SSDs for all OSDs. This subnet is connected with Proxmox nodes through eth1. This storage is used by VM6. |
| 192.168.20.0:6790 | This is for Ceph cluster #2 with hybrid HDDs for all OSDs. This subnet is connected with Proxmox nodes through eth2. This storage is used by VM5. |
| 192.168.30.0:6791 | This is for Ceph cluster #3 with HDDs for all OSDs. This subnet is connected with Proxmox nodes through eth3. This storage is used by VM1, VM2, VM3, and VM4. |
| 192.160.10.0 | This is the main subnet for all virtual machines. |

If you are going to use the latest Proxmox VE 3.2 to create a multitier Ceph cluster, keep in mind that as of now, the Proxmox GUI does not support multi-Ceph clusters. You can only manage one cluster through the Proxmox GUI and additional Ceph clusters will have to be managed through the CLI.

Multitiered infrastructure is very typical for data centers, where there is a different level of SLA-based clients with various requirements for storage performance.

# Scenario #3 – virtual infrastructure for multitenant cloud service provider

The following are the key requirements for this scenario:

- There should be a firewall cluster for edge firewalls
- Each client network must be fully isolated from each other
- A separate storage cluster for backup
- Client users must be able to access their company virtual desktops via RDP
- There must be bandwidth control ability for client networks' Internet connectivity
- Ability to replicate all data to another data center

In this scenario, the virtualized firewall and virtual bridges are used to separate traffic between each client network. Virtual firewall has seven virtual network interfaces to connect six client networks within a virtual environment and to provide WAN connectivity. Internet bandwidth is controlled through a virtual firewall for each vNIC. A virtual firewall is connected to WAN through the main virtual bridge vmbr0. Proxmox cluster has virtual bridges listed in the following table:

| Virtual bridge | Description |
|---|---|
| vmbr0 | This is the main virtual bridge to provide WAN connection to virtual firewalls |
| vmbr1 | This connects the main storage cluster |
| vmbr5 | This connects the storage cluster for backup |

| Virtual bridge | Description |
|---|---|
| vmbr10 | This is the bridge for the subnet 192.10.10.0 of the company ABC |
| vmbr20 | This is the bridge for the subnet 192.20.20 of the company XYZ.0 |
| vmbr30 | This is the bridge for the OpenVZ containers for web hosting instances |
| vmbr40 | This is the bridge for the Object Storage instances to be used by a software developer |
| vmbr50 | This is the bridge for the subnet 192.50.50.0 of the company 123 |
| vmbr60 | This is the bridge for a small business virtual cluster |

Each bridge connects the client company's virtual machines together and creates an isolated internal network for respective clients.

# Scenario #4 – a nested virtual environment for a software development company

The key requirements for this scenario are as follows:

- The developer must have a nested virtual environment to test software
- Outsourced developers should have access to nested virtual environments using RDP
- The developer must have control to create or delete a virtual cluster
- A tested virtual environment must be fully isolated from the main company's network

In this scenario, a nested Proxmox virtual cluster is created inside the main cluster mainly for software testing purposes for a software development company. Since virtual clusters can be created at any time and taken down, it reduces the cost and time by not setting up the entire hardware and setup process. A virtual firewall is used to direct traffic between nested and main virtual environments. All developers access their nested virtual machines through RDP port forwarding. Outsourced developers also connect to nested virtual environments using RDP. The main firewall does port forwarding to the virtual firewall. Then, the virtual firewall does port forwarding to nested virtual machines.

Four subnets used in this example are listed in the following table:

| Subnet | Description |
|---|---|
| 192.160.10.0 | This is the main company subnet. All staff, including developers, are on this subnet. |
| 192.160.20.0 | This is the main storage cluster subnet. It is connected to the main cluster with vmbr1. |
| 194.160.10.0 | This is the nested cluster subnet. It is isolated from the main cluster with vmbr2, which is only connected to the virtual firewall. |
| 194.160.20.0 | This is the nested storage cluster subnet. |

Virtual machines VM Proxmox 1, VM Proxmox 2, and VM Proxmox 3 are used to create the nested Proxmox cluster while VM Storage 1, VM Storage 2, and VM Storage 3 virtual machines are used to create the nested storage cluster.

# Scenario #5 – a virtual infrastructure for the public library

The key requirements for this scenario are as follows:

- Catalog consoles or terminals should be on a separate subnet along with the main admin subnet
- Public Wi-Fi and consoles or terminals for public Internet usage should be on the same separate subnet
- Kiosks for self check-in/check-out books and media
- Access to online library catalog
- Public Internet traffic must be monitored for any Internet usage policy violation
- Public computers should have printer access

This is a typical scenario for a public library network. Since a public library is a public place with access to computers for public usage, it is very important to isolate sensitive networks such as a library member database and library office administration. In this example, the network is isolated by using two subnets listed in the following table:

| Subnet | Description |
|---|---|
| 192.160.10.0 | This is the main network only for the library staff and protected consoles such as catalog, Kiosks, staff printers, and self check-in/check-out. |
| 192.200.200.0 | This is the public subnet for public Wi-Fi, internet consoles, and printers with the payment system. |

The network 192.200.200.0 is controlled, managed, and isolated using a virtual firewall VM5. The virtual firewall has two vNICs, one for a WAN connection through vmbr3, and the other to connect to the dedicated NIC on the Proxmox node through vmbr4. The eth2 network interface card of Proxmox node is connected to a separate LAN switch to only connect public devices. The virtual firewall provides the ability to monitor Internet traffic to keep in line with any violation of the library's Internet usage policy.

Each Proxmox nodes has four network interface cards, eth0, eth1, eth2, and eth3, and the cluster has three virtual bridges, vmbr0, vmbr2, and vmbr4. The main storage cluster is connected to the Proxmox node through eth1 and the backup cluster is connected to eth3.

# Scenario #6 – multifloor office virtual infrastructure with virtual desktops

The key requirements for this scenario are as follows:

- All staff should be on virtual desktops
- Redundant Internet connectivity
- Each department should have their own remote desktop server
- The accounting department network traffic should only be directed to their department

This is a common scenario for an office building where departments are on different floors of the building. Since the accounting department requires data isolation, we are going to use vLAN to isolate their data. Administrative offices and the copy room main server room are on the fourth floor. The Human Resource department is on the fifth floor, the Marketing department is on the sixth, and the Accounting department is on the seventh floor. The fifth, sixth, and seventh floors have their own LAN switches. So, we can easily use vLAN for other floors if it was required. We only need to set up vLAN on the switch on the fourth floor.

Each Proxmox node has two network interfaces. The eth1 card is used to connect storage cluster and eth0 is used to connect all virtual machines to their departments. Vlan0.10 is used to separate only the accounting traffic directly to the seventh floor.

All department staff use virtual desktops through RDP. Each department virtual server acts as remote desktop servers and the department's main server.

# Scenario #7 – virtual infrastructure for hotel industry

The key requirements for this scenario are as follows:

- Centralized IT infrastructure management.
- Dedicated secured Wi-Fi access for guests.
- Secured private Wi-Fi access in restaurants and bars for menu tablets only. The Wi-Fi needs to talk to the restaurant and bar servers.
- All staff must have remote desktops for day-to-day work.
- The video surveillance system should be integrated with the virtual environment.

This is a scenario for a typical hotel establishment with an in-house restaurant. This example uses a central virtualized database server to store all information. Although it is an unconventional way to connect all departments with a single database, including the surveillance system, it is possible to use an all-in-one single solution to reduce the cost and management overhead. In a typical scenario, separate software is used to handle different departments without data portability. In this example, unified management software connects all departments with a single database and a customized user interface for each department.

Secured non-filtered Wi-Fi connectivity is provided to all guests. DHCP is provided directly by the firewall. Secured private Wi-Fi is set up for restaurant menu tablets only. All menu tablets only connect to the restaurant/bar virtual server with IP 192.190.1.5. All department thin clients and IP-based surveillance cameras are connected to the main network subnet 192.190.1.0.

# Scenario #8 – virtual infrastructure for a geological survey organization

The key requirements for this scenario are as follows:

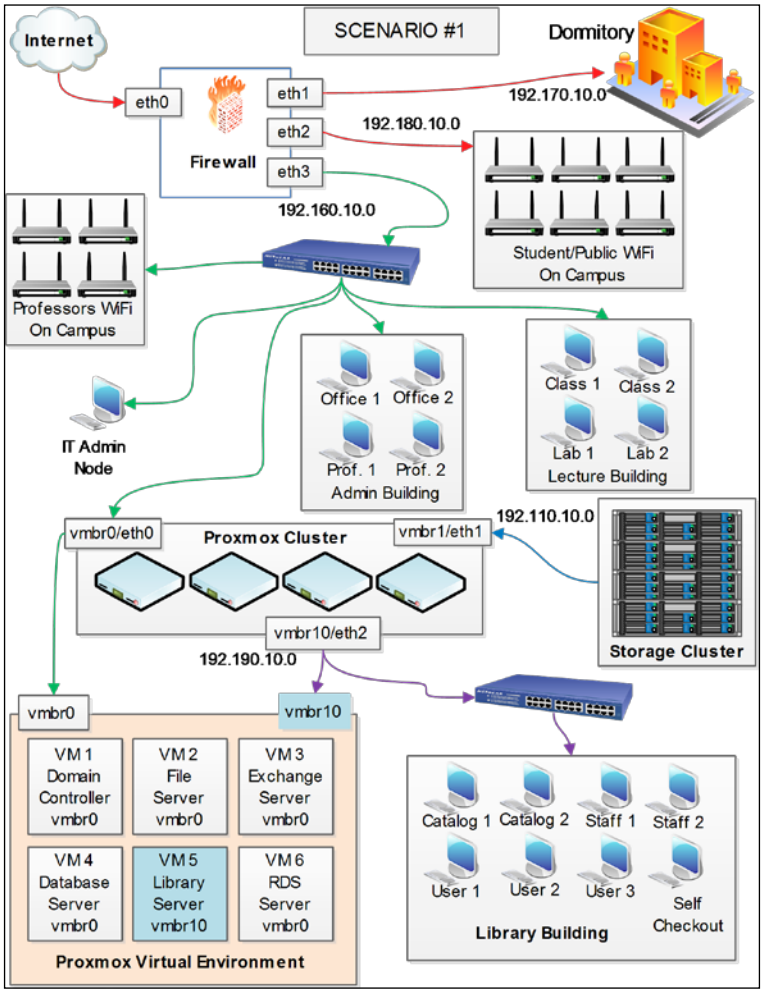
- Field surveyors should submit their work orders from their mobile devices through a VPN connection
- There must be a failover infrastructure in multisite network topologies

In this scenario, a geographical survey company has a main office and a branch office connected with a 1+GBps hardlink network connectivity. Each office has an identical infrastructure setup. All surveyors use mobile devices such as tablets for their survey work. The survey software autodetects which office IP is live and sends data to the infrastructure of that office. All data is replicated on a block level in real-time between two offices.

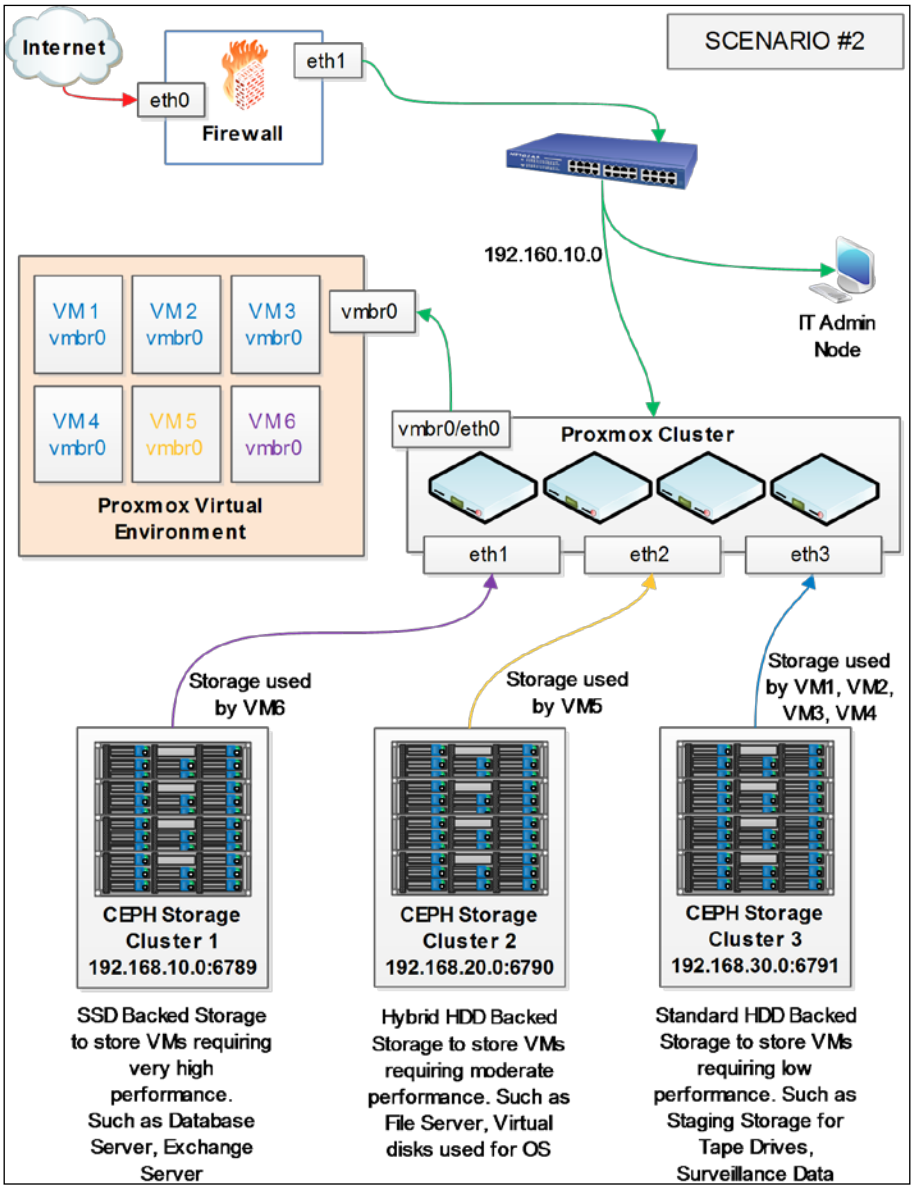
If the infrastructure of one office becomes unavailable, the staff can simply continue to work out of infrastructure from the other office.

# Network diagrams for scenarios

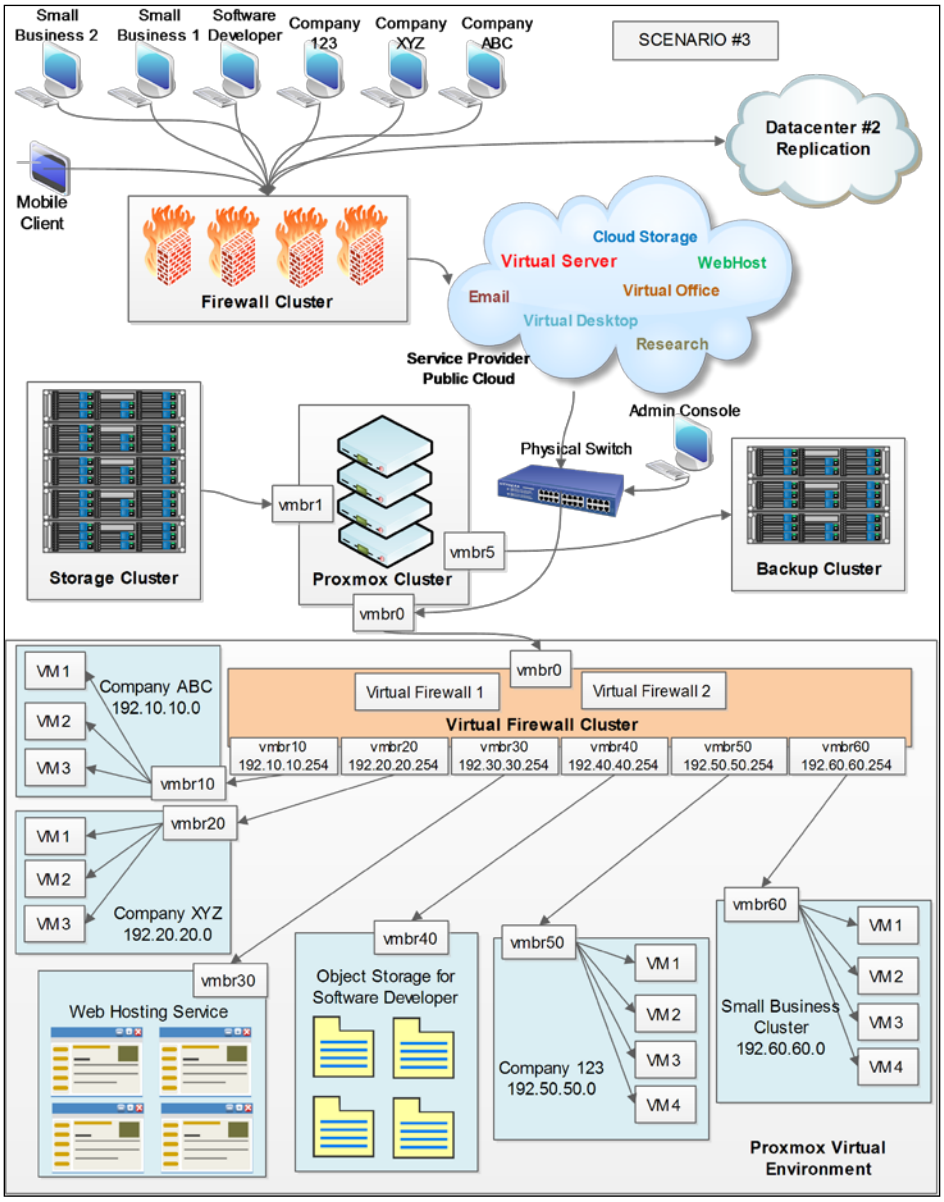
The following is the network diagram for *Scenario #1 – academic institution*:



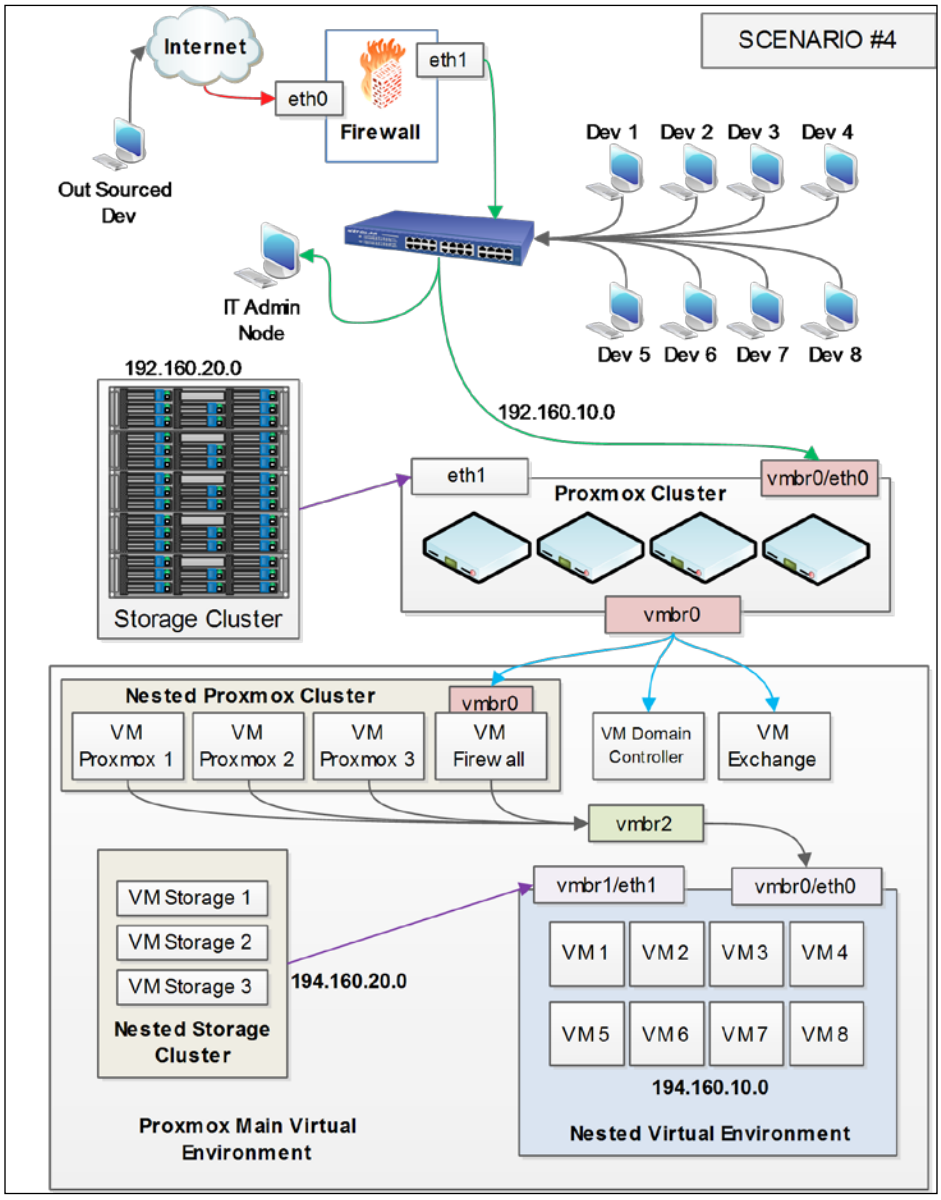
The following is the network diagram for *Scenario #2 – multitier storage cluster with Proxmox cluster*:



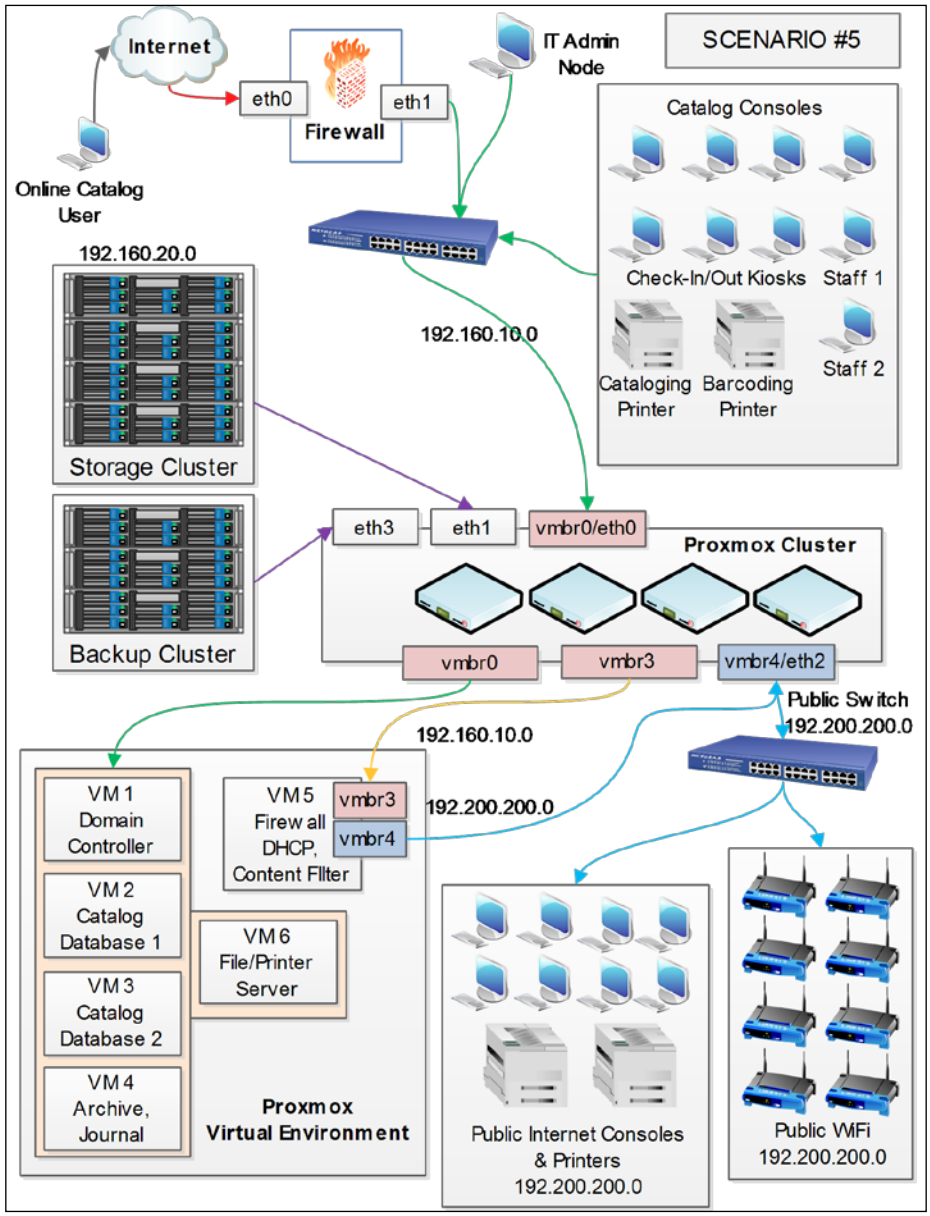
The following is the network diagram for *Scenario #3 – virtual infrastructure for multitenant cloud service provider*:



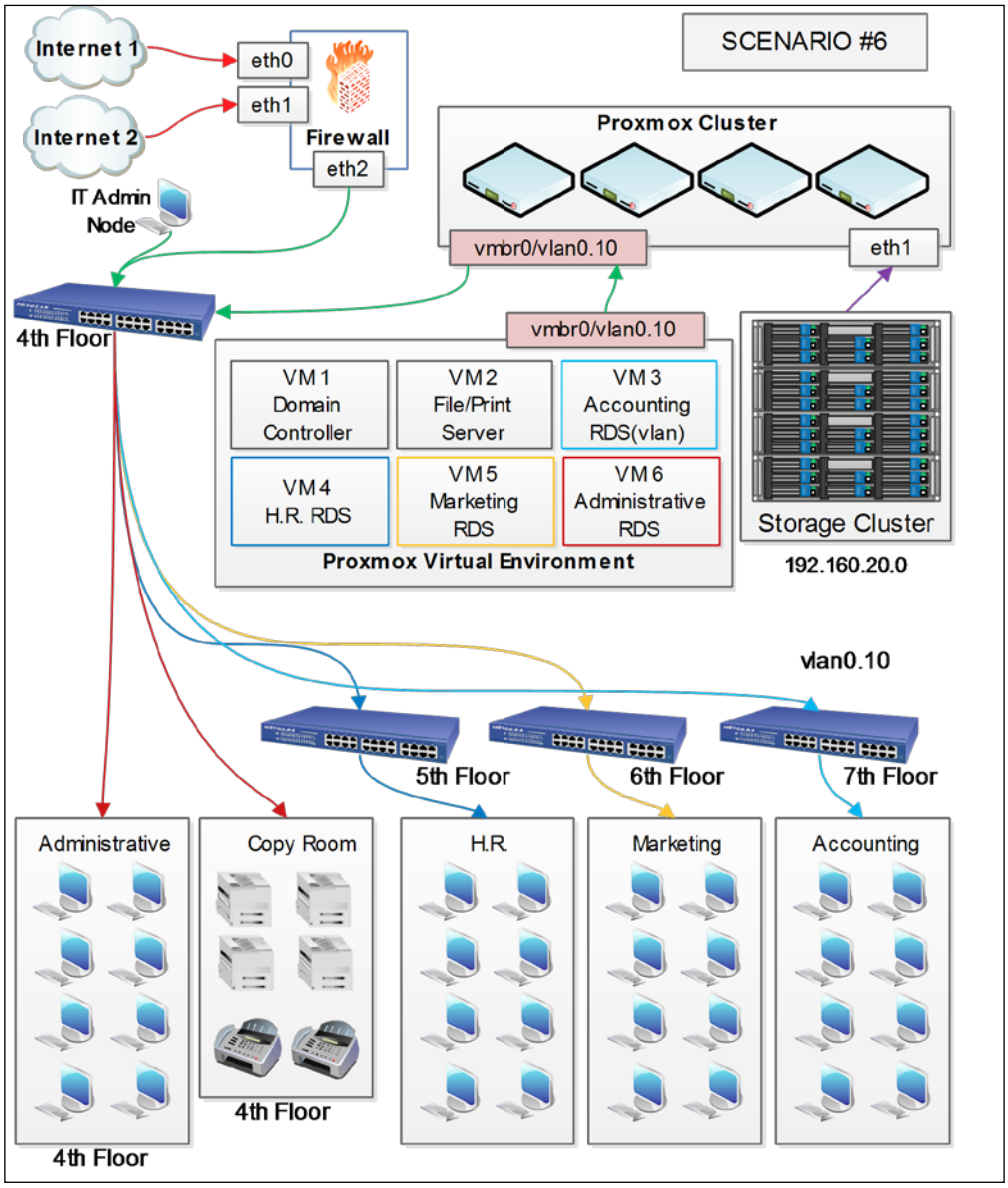
The following is the network diagram for *Scenario #4 – a nested virtual environment for a software development company*:



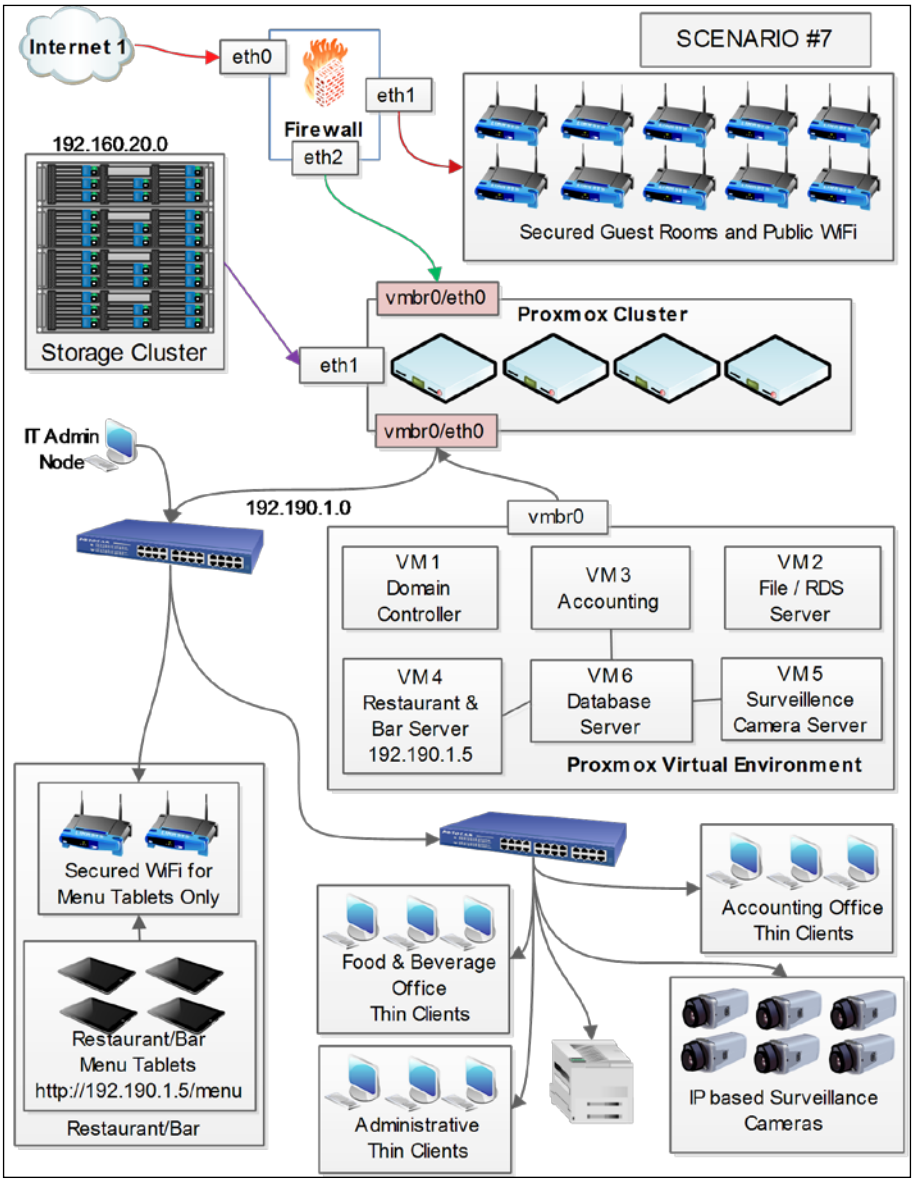
The following is the network diagram for *Scenario #5 – a virtual infrastructure for the public library*:



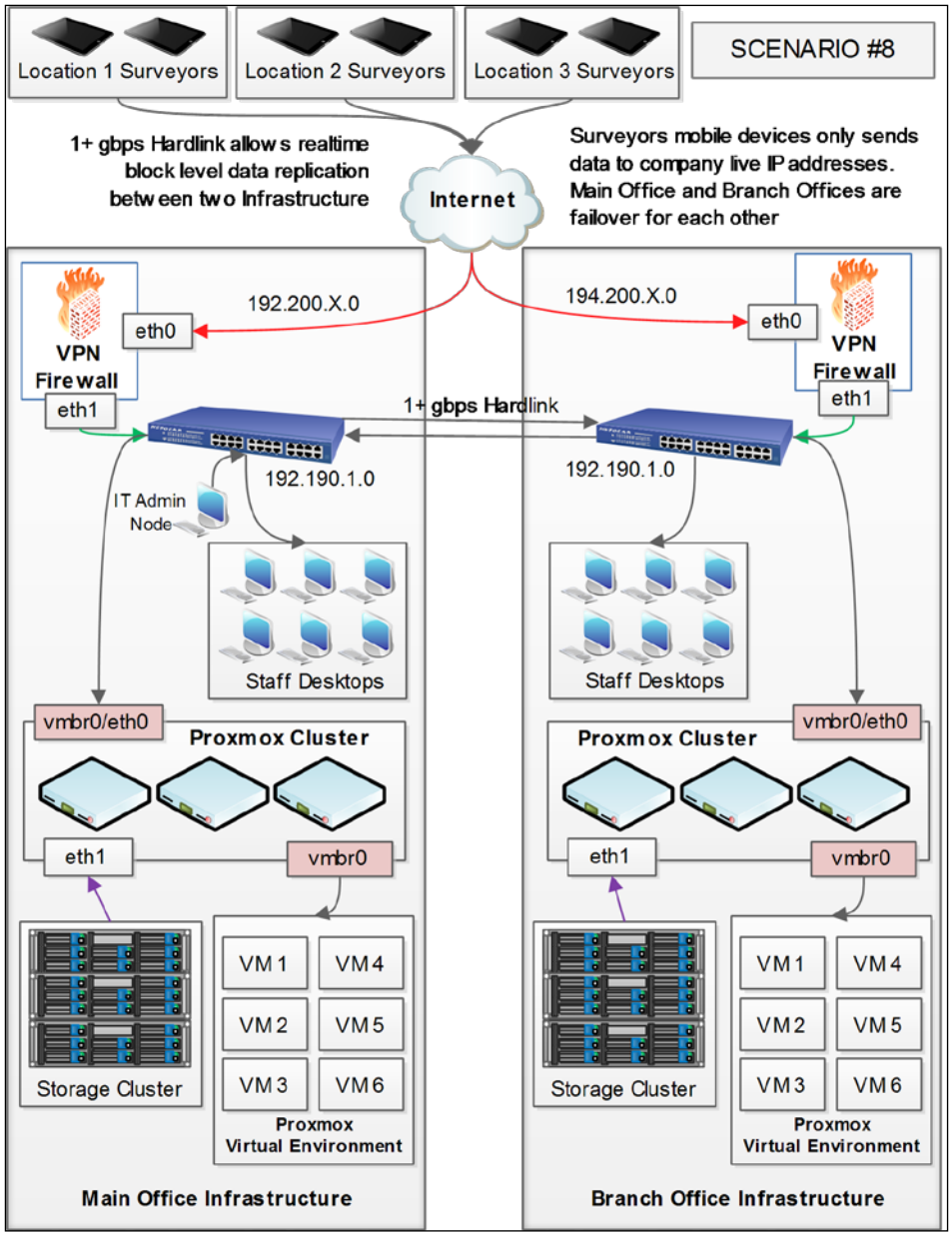
The following is the network diagram for *Scenario #6 – multifloor office virtual infrastructure with virtual desktops*:



The following is the network diagram for *Scenario #7 – virtual infrastructure for hotel industry*:



The following is the network diagram for *Scenario #8, virtual infrastructure for a geological survey organization*:



# Summary

We do hope that these network diagrams were helpful to understand the flexibility and customizability of Proxmox. These are just very few scenarios out of hundreds that you might face in the real world. The concept of network diagraming will also help you visualize a network before you start implementing it. A network diagram on paper helps you to figure out data path or connectivity before implementing or documenting an existing network to see how the network environment is set up.

Although we are at the end of this book, your journey into the world of virtualization using Proxmox is just beginning. If the knowledge from this book has helped you to understand Proxmox better and you actually have leaped forward many steps, then this book has fulfilled its purpose. I would personally like to thank you for going through the journey together and wish you all the best in your virtualization career.

# Index

## Symbols



## A

## B



## C

## D



## E

## F



## G



## H



## I

## K



## L



## M

# N

## O



## P

## Q



## R



## S

## T



## U



## V

## W



## X



## Y



## Z


## Thank you for buying
# Mastering Proxmox

## About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

## About Packt Open Source

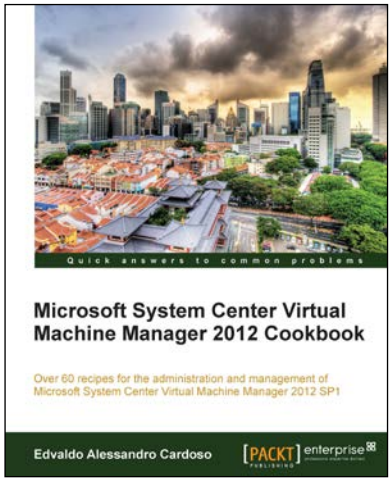
In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licenses, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

## Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

[PACKT] open source
community experience distilled
PUBLISHING



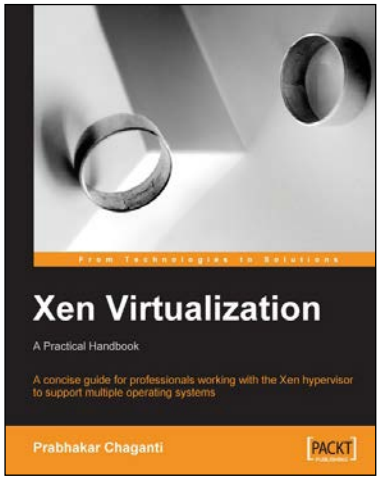

## Microsoft System Center Virtual Machine Manager 2012 Cookbook

ISBN: 978-1-84968-632-7 Paperback: 342 pages

Over 60 recipes for the administration and management of Microsoft System Center Virtual Machine Manager 2012 SP1

1. Create, deploy, and manage Datacentres, Private and Hybrid Clouds with hybrid hypervisors by using VMM 2012 SP1, App Controller, and Operations Manager.
2. Integrate and manage fabric (compute, storages, gateways, networking) services and resources. Deploy Clusters from bare metal servers.

---

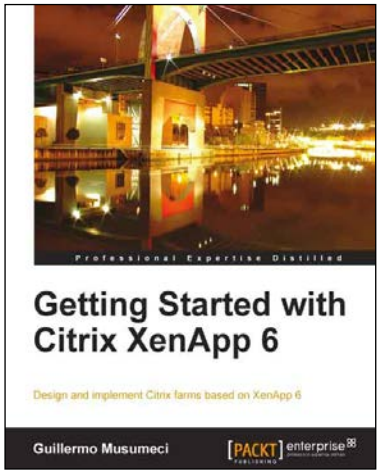



## Xen Virtualization

ISBN: 978-1-84719-248-6 Paperback: 148 pages

A concise guide for professionals working with the Xen hypervisor to support multiple operating systems

1. Installing and configuring Xen.
2. Managing and administering Xen servers and virtual machines.
3. Setting up networking, storage, and encryption.
4. Backup and migration.



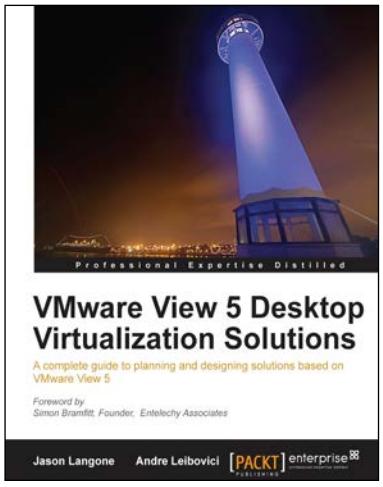

## Getting Started with Citrix XenApp 6

ISBN: 978-1-84968-128-5 Paperback: 444 pages

Design and implement Citrix farms based on XenApp 6

1. Use Citrix management tools to publish applications and resources on client devices with this book and eBook.
2. Deploy and optimize XenApp 6 on Citrix XenServer, VMware ESX, and Microsoft Hyper-V virtual machines and physical servers.
3. Understand new features included in XenApp 6 and review Citrix farms terminology and concepts.

---




## VMware View 5 Desktop Virtualization Solutions

ISBN: 978-1-84968-112-4 Paperback: 288 pages

A complete guide to planning and designing solutions based on VMware View 5

1. Written by VMware experts Jason Langone and Andre Leibovici, this book is a complete guide to planning and designing a solution based on VMware View 5.
2. Secure your Visual Desktop Infrastructure (VDI) by having firewalls, antivirus, virtual enclaves, USB redirection and filtering, and smart card authentication.