Rocky Linux Web Services Guide (English version)

A book from the Documentation Team

Version : 2025/07/12

Rocky Documentation Team

Table of contents

1. Foreword

Rocky Linux is part of the Enterprise Linux family, making it particularly well suited to hosting web services such as file servers (FTP, sFTP), web servers (apache, nginx), application servers (PHP, Python), database servers (MariaDB, MySQL, PostgreSQL) or more specific services such as load balancing, caching, proxy or reverse proxy (HAProxy, Varnish, Squid).

The web would not exist without email. Web services generally make extensive use of mail servers (Postfix).

Sometimes, these services are extremely busy or require highly available services. Other services can be implemented in these cases to guarantee optimal service performance (Heartbeat, PCS).

Each chapter of this book can be consulted independently, according to your needs, and reading the chapters in order is not compulsory.

This book is also part of a series of books dedicated to system administration under Linux (Admin Guide, Learning Bash, Learning Ansible). Where necessary, you will be invited to review the concepts you may be missing in the corresponding chapters of the books mentioned above.

1.1 Public

The target audience for this book is system administrators already trained in the use of system administration commands (see our book Admin Guide) who want to install, configure, and secure their web services.

1.2 How to use this book

This book has been designed as a training manual, and you can use it in several ways. It can be used as a training aid for trainers or as a self-training aid for administrators wishing to acquire new skills or reinforce their existing knowledge.

To implement some of the services presented in this book, you may need two (or more) servers to put the theory into practice.

2. Licence

RockyLinux offers Linux courseware for trainers or people wishing to learn how to administer a Linux system on their own.

RockyLinux materials are published under Creative Commons-BY-SA. This means you are free to share and transform the material, while respecting the author's rights.

BY : **Attribution**. You must cite the name of the original author.

SA : **Share Alike**.

- Creative Commons-BY-SA licence : https://creativecommons.org/licenses/by-sa/4.0/

The documents and their sources are freely downloadable from:

- https://docs.rockylinux.org
- https://github.com/rocky-linux/documentation

Our media sources are hosted at github.com. You'll find the source code repository where the version of this document was created.

From these sources, you can generate your own personalized training material using mkdocs. You will find instructions for generating your document here.

How can I contribute to the documentation project?

You'll find all the information you need to join us on our git project home page.

We wish you all a pleasant reading and hope you enjoy the content.

3. Part 1. Files Servers

Info

The content for this page has yet to be written.

4. Part 2. Web Servers Introduction

4.1 Introduction

4.1.1 HTTP protocol

HTTP (**H**yper**T**ext **T**ransfer **P**rotocol) has been the most widely used protocol on the Internet since 1990.

This protocol enables the transfer of files (mainly in HTML format but also in CSS, JS, AVI, etc.) localized by a character string called **URL** between a browser (the client) and a Web server (called `httpd` on UNIX machines).

HTTP is a "request-response" protocol operating on top of **TCP** (**T**ransmission **C**ontrol **P**rotocol).

1. The client opens a TCP connection to the server and sends a request.
2. The server analyzes the request and responds according to its configuration.

The HTTP protocol is "**STATELESS**": it does not retain any information about the client's state from one request to the next. Dynamic languages such as PHP, Python, or Java store client session information in memory (as on an e-commerce site).

The current HTTP protocols are version 1.1, which is widely used, and versions 2 and 3, which are gaining adoption.

An HTTP response is a set of lines sent to the browser by the server. It includes:

- A **status line**: specifies the protocol version and the request's processing status using a code and explanatory text. The line comprises three elements separated by a space:
- The protocol version used
- The status code
- The meaning of the code
- **Response header fields**: these optional lines provide additional information about the response and/or the server. Each line consists of a name qualifying the header type, followed by a colon (:) and the header value.
- **The response body**: contains the requested document.

Here is an example of an HTTP response:

```
$ curl --head --location https://docs.rockylinux.org
HTTP/2 200
accept-ranges: bytes
access-control-allow-origin: *
age: 109725
cache-control: public, max-age=0, must-revalidate
content-disposition: inline
content-type: text/html; charset=utf-8
date: Fri, 21 Jun 2024 12:05:24 GMT
etag: "cba6b533f892339d3818dc59c3a5a69a"
server: Vercel
strict-transport-security: max-age=63072000
x-vercel-cache: HIT
x-vercel-id: cdg1::pdqbh-1718971524213-4892bf82d7b2
content-length: 154696
```

Note

Learning how to use the `curl` command will be very helpful for troubleshooting your servers in the future.

The role of the web server is to translate a URL into a local resource. Consulting the https://docs.rockylinux.org/ page is like sending an HTTP request to this machine. The DNS service plays an essential role.

4.1.2 URLs

A **URL** (**U**niform **R**esource **L**ocator) is an ASCII character string used to designate resources on the Internet. It is informally referred to as a web address.

A URL has three parts:

```
<protocol>://<host>:<port>/<path>
```

- **Protocol name**: This is the language used to communicate over the network, such as HTTP, HTTPS, FTP, etc. The most widely used protocols are HTTP (HyperText Transfer Protocol) and its secure version, HTTPS, which is used to exchange Web pages in HTML format.
- **Login** and **password**: This option allows you to specify access parameters to a secure server. It is not recommended, as the password is visible in the URL (for security purposes).
- **Host**: This is the computer's name hosting the requested resource. Note that the server's IP address can also be used, but it makes the URL less readable.
- **Port number**: This is associated with a service that enables the server to know the requested resource type. The HTTP protocol's default port is port 80 and 443 with HTTPS. So, the port number is optional when the protocol is HTTP or HTTPS.
- **Resource path**: This part lets the server know the location of the resource. Generally, it is the location (directory) and name of the requested file. If nothing in the address specifies a location, it indicates the host's first page. Otherwise, it indicates the path to the page to display.

4.1.3 Ports

An HTTP request will arrive on port 80 (the default port for HTTP) of the server running on the host. However, the administrator is free to choose the server's listening port.

The HTTP protocol is available in a secure version: the HTTP protocol (port 443). Implement this encrypted protocol with the `mod_ssl` module.

Using other ports is also possible, such as port `8080` (Java EE application servers).

4.2 Apache and Nginx

The two most common web servers for Linux are Apache and Nginx. We will discuss this in the following chapters.

5. Part 2.1 Web Servers Apache

5.1 Apache

In this chapter, you will learn about the web server Apache.

Objectives: You will learn how to:

✓ install and configure Apache

🏁 **apache**, **http**, **httpd**

Knowledge: ⭐⭐
Complexity: ⭐⭐

Reading time: 30 minutes

5.1.1 Generalities

The Apache HTTP server is the work of a group of volunteers: The Apache Group. This group set out to build a Web server on the same level as commercial products but as free software (its source code is available).

Hundreds of users joined the original team and contributed ideas, tests, and lines of code to making Apache the most widely used Web server in the world.

Apache's ancestor is the accessible server developed by the National Center for Supercomputing Applications at the University of Illinois. The evolution of this server came to a halt when the person in charge left the NCSA in 1994. Users continued to fix bugs and create extensions, which they distributed as "patches", hence the name "a patchee server".

The release of Apache version 1.0 was on December 1, 1995 (over 30 years ago!).

The development team coordinates its work through a mailing list, where discussions regarding proposals and changes to the software occur. Changes are

voted on before incorporation into the project. Anyone can join the development team. To become a member of The Apache Group, you must actively contribute to the project.

The Apache server has a robust Internet presence, accounting for around 50% of the market share for all active sites.

Apache often loses market share to its biggest challenger, the Nginx server. The latter is faster at delivering web pages but less functionally complete than the giant Apache.

5.1.2 Installation

Apache is **cross-platform**. It is usable on Linux, Windows, Mac...

The administrator will have to choose between two installation methods:

- **Package installation**: the distribution vendor supplies **stable, supported** (but sometimes older) versions
- **Installation from source**: This involves the administrator compiling the software, who can specify the options that interest him or her, thus optimizing the service. Since Apache has a modular architecture, it is generally unnecessary to re-compile the Apache software to add or remove additional functionalities (add or remove modules).

The package-based installation method is strongly recommended. Additional repositories are available to install more recent versions of Apache on older distributions, but nobody will provide support in case of problems.

On Enterprise Linux distributions, the `httpd` package provides the Apache server.

In the future, you might have to install some extra modules. Here are some examples of modules and their roles:

- **mod_access**: filters client access by hostname, IP address, or other characteristic
- **mod_alias**: enables the creation of aliases or virtual directories
- **mod_auth**: authenticates clients
- **mod_cgi**: executes CGI scripts
- **mod_info**: provides information on server status
- **mod_mime**: associates file types with the corresponding action
- **mod_proxy**: proposes a proxy server
- **mod_rewrite**: rewrites URLs
- Others

```
sudo dnf install httpd
```

The version installed on Rocky Linux 9 is 2.4.

Installing the package creates an `apache` system user and a corresponding `apache` system group.

```
$ grep apache /etc/passwd
apache:x:48:48:Apache:/usr/share/httpd:/sbin/nologin
$ grep apache /etc/group
apache:x:48:
```

Enable and start the service:

```
$ sudo systemctl enable httpd --now
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /
usr/lib/systemd/system/httpd.service.
```

You can check the service's status:

```
$ sudo systemctl status httpd
● httpd.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; preset:
disabl>     Active: active (running) since Fri 2024-06-21 14:22:34 CEST; 8s ago
```

```
        Docs: man:httpd.service(8)
    Main PID: 4387 (httpd)
      Status: "Started, listening on: port 80"
       Tasks: 177 (limit: 11110)
      Memory: 24.0M
         CPU: 68ms
      CGroup: /system.slice/httpd.service
              ├─4387 /usr/sbin/httpd -DFOREGROUND
              ├─4389 /usr/sbin/httpd -DFOREGROUND
              ├─4390 /usr/sbin/httpd -DFOREGROUND
              ├─4391 /usr/sbin/httpd -DFOREGROUND
```

Do not forget to open your firewall (see Security section).

You can now check the availability of the service:

- From any web browser providing the IP address of your server (for example, http://192.168.1.100/).
- Directly from your server.

To do so, you must install a text browser, such as elinks.

```
sudo dnf install elinks
```

Browse your server and check the default page:

```
elinks http://localhost
```

Installing the `httpd` package generates a complete tree structure that needs to be fully understood:

```
/etc/httpd/
├── conf
│   ├── httpd.conf
│   └── magic
├── conf.d
│   ├── README
│   ├── autoindex.conf
│   ├── userdir.conf
│   └── welcome.conf
├── conf.modules.d
│   ├── 00-base.conf
│   ├── 00-brotli.conf
```

```
│   ├── 00-dav.conf
│   ├── 00-lua.conf
│   ├── 00-mpm.conf
│   ├── 00-optional.conf
│   ├── 00-proxy.conf
│   ├── 00-systemd.conf
│   ├── 01-cgi.conf
│   ├── 10-h2.conf
│   ├── 10-proxy_h2.conf
│   └── README
├── logs -> ../../var/log/httpd
├── modules -> ../../usr/lib64/httpd/modules
├── run -> /run/httpd
└── state -> ../../var/lib/httpd
/var/log/httpd/
├── access_log
└── error_log
/var/www/
├── cgi-bin
└── html
```

You will notice that the `/etc/httpd/logs` folder is a symbolic link to the `/var/log/httpd` directory. Similarly, you will notice that the files making up the default site are in the `/var/www/html` folder.

5.1.3 Configuration

Initially, the Apache server's configuration was in a single `/etc/httpd/conf/httpd.conf` file. Over time, this file has become increasingly prominent and less readable.

Modern distributions, therefore, tend to distribute Apache configuration over a series of `*.conf` files in the directories `/etc/httpd/conf.d` and `/etc/httpd/conf.modules.d`, attached to the main `/etc/httpd/conf/httpd.conf` file by the Include directive.

```
$ sudo grep "^Include" /etc/httpd/conf/httpd.conf
Include conf.modules.d/*.conf
IncludeOptional conf.d/*.conf
```

The `/etc/httpd/conf/httpd.conf` file is amply documented. In general, these comments are sufficient to clarify the administrator's options.

Global server configuration is in `/etc/httpd/conf/httpd.conf`.

This file has three sections for configuring:

- in **section 1**, the global environment;
- in **section 2**, the default site and default virtual site parameters;
- in **section 3**, the virtual hosts.

Virtual hosting lets you put **several virtual sites online** on the same server. The sites are then differentiated according to their domain names, IP addresses, etc.

Modifying a value in section 1 or 2 affects all hosted sites.

In a shared environment, modifications are, therefore, in section 3.

To facilitate future updates, creating a section 3 configuration file for each virtual site is strongly recommended.

Here is a minimal version of the `httpd.conf` file:

```
ServerRoot "/etc/httpd"
Listen 80
Include conf.modules.d/*.conf
User apache
Group apache
ServerAdmin root@localhost
<Directory />
    AllowOverride none
    Require all denied
</Directory>
DocumentRoot "/var/www/html"
<Directory "/var/www">
    AllowOverride None
    Require all granted
</Directory>
<Directory "/var/www/html">
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
<IfModule dir_module>
    DirectoryIndex index.html
</IfModule>
<Files ".ht*">
```

```
    Require all denied
</Files>
ErrorLog "logs/error_log"
LogLevel warn
<IfModule log_config_module>
    LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\""
combined
    LogFormat "%h %l %u %t \"%r\" %>s %b" common
    <IfModule logio_module>
      LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"
%I %O" combinedio
    </IfModule>
    CustomLog "logs/access_log" combined
</IfModule>
<IfModule alias_module>
    ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"
</IfModule>
<Directory "/var/www/cgi-bin">
    AllowOverride None
    Options None
    Require all granted
</Directory>
<IfModule mime_module>
    TypesConfig /etc/mime.types
    AddType application/x-compress .Z
    AddType application/x-gzip .gz .tgz
    AddType text/html .shtml
    AddOutputFilter INCLUDES .shtml
</IfModule>
AddDefaultCharset UTF-8
<IfModule mime_magic_module>
    MIMEMagicFile conf/magic
</IfModule>
EnableSendfile on
IncludeOptional conf.d/*.conf
```

Section 1

The various directives encountered in Section 1 are :

Option	Information
`ServerTokens`	This directive will be in a future chapter.
`ServertRoot`	Indicates the path to the directory containing all the files making up the Apache server.
`Timeout`	The number of seconds before the expiration time of a request that is too long (incoming or outgoing).
`KeepAlive`	Persistent connection (several requests per TCP connection).
`MaxKeepAliveRequests`	Maximum number of persistent connections.
`KeepAliveTimeout`	Number of seconds to wait for the next client request before closing the TCP connection.
`Listen`	Allows Apache to listen to specific addresses or ports.
`LoadModule`	Loads add-on modules (fewer modules = greater security).
`Include`	Includes other server configuration files.
`ExtendedStatus`	Displays more information about the server in the server status module.
`User` and `Group`	Allows the launching of Apache processes with different users. Apache always starts as root, then changes its owner and group.

MULTI-PROCESS MODULES (MPM)

The Apache server was designed to be powerful and flexible, capable of running on various platforms.

Different platforms and environments often mean different functionality or the use of other methods to implement the same functionality as efficiently as possible.

Apache's modular design allows the administrator to choose which features to include in the server by selecting which modules to load, either at compile or run-time.

This modularity also includes the most rudimentary web server functions.

The Multi-Process Modules (MPM) modules are responsible for associating with the machine's network ports, accepting requests, and distributing them among the various child processes.

Configuring MPM modules is in the `/etc/httpd/conf.modules.d/00-mpm.conf` configuration file:

```
# Select the MPM module which should be used by uncommenting exactly
# one of the following LoadModule lines.  See the httpd.conf(5) man
# page for more information on changing the MPM.

# prefork MPM: Implements a non-threaded, pre-forking web server
# See: http://httpd.apache.org/docs/2.4/mod/prefork.html
#
# NOTE: If enabling prefork, the httpd_graceful_shutdown SELinux
# boolean should be enabled, to allow graceful stop/shutdown.
#
#LoadModule mpm_prefork_module modules/mod_mpm_prefork.so

# worker MPM: Multi-Processing Module implementing a hybrid
# multi-threaded multi-process web server
# See: http://httpd.apache.org/docs/2.4/mod/worker.html
#
#LoadModule mpm_worker_module modules/mod_mpm_worker.so

# event MPM: A variant of the worker MPM with the goal of consuming
# threads only for connections with active processing
# See: http://httpd.apache.org/docs/2.4/mod/event.html
#
LoadModule mpm_event_module modules/mod_mpm_event.so
```

As you can see, the default MPM is the `mpm_event`.

The performance and capabilities of your web server depend heavily on the choice of MPM.

Choosing one module over another is a complex task, as is optimizing the chosen MPM module (number of clients, queries, etc.).

The Apache configuration assumes a moderately busy service (256 clients max) by default.

ABOUT KEEPALIVE DIRECTIVES

With the `KeepAlive` directive disabled, every resource request on the server requires opening a TCP connection, which is time-consuming from a network point of view and requires a lot of system resources.

With the `KeepAlive` directive set to `On`, the server keeps the connection open with the client for the duration of the `KeepAlive`.

This strategy is a quick winner because a web page contains several files (images, stylesheets, Javascript, etc.).

However, it is important to set this value as precisely as possible:

- Too short a value penalizes the customer,
- Too long a value penalizes server resources.

`KeepAlive` values for individual customer virtual hosts allow more granularity per customer. In this case, setting `KeepAlive` values happens directly in the customer's VirtualHost or at the proxy level (`ProxyKeepalive` and `ProxyKeepaliveTimeout`).

Section 2

Section 2 sets the values used by the main server. The main server responds to all requests not handled by one of the Virtualhosts in section 3.

The values are also used as default values for virtual sites.

Option	Information
`ServerAdmin`	specifies an e-mail address appearing on certain auto-generated pages, such as error pages.
`ServerName`	specifies the name identifying the server. It can happen automatically, but the recommendation is to specify it explicitly (IP address or DNS name).
`DocumentRoot`	specifies the directory containing files to serve to clients. Default /var/www/html/.
`ErrorLog`	specifies the path to the error file.
`LogLevel`	debug, info, notice, warn, error, crit, alert, emerg.
`LogFormat`	defines a specific log format. Done with the CustomLog directive.
`CustomLog`	specifies the path to access the file.
`ServerSignature`	seen in the security part.
`Alias`	specifies a directory outside the tree and makes it accessible by context. The presence or absence of the last slash in the context is important.
`Directory`	specifies behaviors and access rights by directory.
`AddDefaultCharset`	specifies the encoding format for pages sent (accented characters can be replaced by ?...).
`ErrorDocument`	customizes error pages.
`server-status`	report on server status.
`server-info`	report on server configuration.

THE `ErrorLog` DIRECTIVE

The `ErrorLog` directive defines the error log to use.

This directive defines the file name in which the server logs all errors it encounters. If the file path is not absolute, the assumption is to be relative to ServerRoot.

THE `DirectoryIndex` DIRECTIVE

The DirectoryIndex directive defines the site's home page.

This directive specifies the file's name loaded first, which will act as the site index or home page.

Syntax:

```
DirectoryIndex display-page
```

The full path is not specified. Searching for the file happens in the directory specified by DocumentRoot.

Example:

```
DocumentRoot /var/www/html
DirectoryIndex index.php index.htm
```

This directive specifies the name of the website index file. The index is the default page that opens when the client types the site URL (without having to type the index name). This file must be in the directory specified by the `DocumentRoot` directive.

The `DirectoryIndex` directive can specify several index file names separated by spaces. For example, a default index page with dynamic content and, as a second choice, a static page.

THE `Directory` DIRECTIVE

The Directory tag is used to define directory-specific directives.

This tag applies rights to one or more directories. The directory path is entered as an absolute.

Syntax:

```
<Directory directory-path>
Defining user rights
</Directory>
```

Example:

```
<Directory /var/www/html/public>
    Require all granted   # we allow everyone
</Directory>
```

The `Directory` section defines a block of directives applying to a part of the server's file system. The directives here will only apply to the specified directory (and sub-directories).

The syntax of this block accepts wildcards, but it is preferable to use the DirectoryMatch block.

In the following example, we'll deny access to the server's local hard disk, regardless of the client. The "/" directory represents the root of the hard disk.

```
<Directory />
    Require all denied
</Directory>
```

The following example shows authorizing access to all clients'/var/www/html publishing directory.

```
<Directory /var/www/html>
    Require all granted
</Directory>
```

When the server finds an `.htaccess` file, it needs to know whether directives placed in the file have authorization to modify the pre-existing configuration. The `AllowOverride` directive controls the authorization in `Directory` directives. When set to `none`, `.htaccess` files are completely ignored.

THE `mod_status`

The `mod_status` displays a `/server-status` or `/server-info` page summarizing server status:

```
<Location /server-status>
    SetHandler server-status
    Require local
</Location>

<Location /server-info>
    SetHandler server-info
    Require local
</Location>
```

Please note that this module provides information that should not be accessible to your users.

Shared hosting (section 3)

With shared hosting, the customer thinks they are visiting several servers. In reality, there is just one server and several virtual sites.

To set up shared hosting, you need to set up virtual hosts:

- declaring multiple listening ports
- declaring multiple listening IP addresses (virtual hosting by IP)
- declaring multiple server names (virtual hosting by name)

Each virtual site corresponds to a different tree structure.

Section 3 of the `httpd.conf` file declares these virtual hosts.

It is strongly recommended that you create a section 3 configuration file for each virtual site to facilitate future updates.

Choose virtual hosting "by IP" or "by name." Mixing the two solutions is not advisable for production use.

- Configuring each virtual site in an independent configuration file
- VirtualHosts are stored in `/etc/httpd/conf.d/`
- The file extension is `.conf`

THE `VirtualHost` DIRECTIVE

The `VirtualHost` directive defines virtual hosts.

```
<VirtualHost IP-address[:port]>
    # if the "NameVirtualHost" directive is present
    # then "address-IP" must match the one entered
    # under "NameVirtualHost" as well as for "port".
 ...
 </VirtualHost>
```

If you configure the Apache server with the basic directives seen above, you can only publish one site. Indeed, you can not publish multiple sites with the default settings: the same IP address, the same TCP port, and no hostname or unique hostname.

Virtual sites will enable us to publish several websites on the same Apache server. You will define blocks, each describing a website. In this way, each site will have its own configuration.

For ease of understanding, a website is often associated with a single machine. Virtual sites or hosts are so-called because they dematerialize the link between machines and websites.

Example 1:

```
Listen 192.168.0.10:8080
<VirtualHost 192.168.0.10:8080>
  DocumentRoot /var/www/site1/
  ErrorLog /var/log/httpd/site1-error.log
</VirtualHost>

Listen 192.168.0.11:9090
```

```
<VirtualHost 192.168.0.11:9090>
  DocumentRoot /var/www/site2/
  ErrorLog /var/log/httpd/site2-error.log
</VirtualHost>
```

IP-based virtual hosting applies specific guidelines based on the IP address and port on which the request is received. This generally means serving different websites on different ports or interfaces.

THE `NameVirtualHost` DIRECTIVE

The `NameVirtualHost` directive defines name-based virtual hosts.

This directive is mandatory for setting up name-based virtual hosts. With this directive, you specify the IP address on which the server will receive requests from name-based virtual hosts.

Syntax:

```
NameVirtualHost adresse-IP[:port]
```

Example:

```
NameVirtualHost 160.210.169.6:80
```

The directive must come before the virtual site description blocks. It designates the IP addresses used to listen to client requests for virtual sites.

To listen for requests on all the server's IP addresses, use the * character.

Taking changes into account

For each configuration change, it is necessary to reload the configuration with the following command:

```
sudo systemctl reload httpd
```

Manual

A package called' httpd-manual' contains a site that acts as an Apache user manual.

```
sudo dnf install httpd-manual
sudo systemctl reload httpd
```

You can access the manual with a web browser at http://127.0.0.1/manual when installed.

```
$ elinks http://127.0.0.1/manual
```

The `apachectl` command

The `apachectl` is the server control interface for the Apache `httpd` server.

It is a very useful command with the `-t` or `configtest`, which runs a configuration file syntax test.

> **Note**
>
> It is very useful when used with Ansible handlers to test the configuration.

5.1.4 Security

When protecting your server with a firewall (which is a good thing), you might need to consider opening it.

```
sudo firewall-cmd --zone=public --add-service=http
sudo firewall-cmd --zone=public --add-service=https
sudo firewall-cmd --reload
```

SELinux

By default, if SELinux security is active, it prevents the reading of a site from a directory other than `/var/www/`.

The directory containing the site must have the security context `httpd_sys_content_t`.

You can check the current context with the command:

```
* ls -Z /dir
```

Add context with the following command:

```
sudo chcon -vR --type=httpd_sys_content_t /dir
```

It also prevents the opening of a non-standard port. Opening the port is a manual operation using the `semanage` command (not installed by default).

```
sudo semanage port -a -t http_port_t -p tcp 1664
```

User and Group directives

The `User` and `Group` directives define an Apache management account and group.

Historically, root ran Apache, which caused security problems. The root always runs Apache, but then its identity is changed. Generally User `apache` and Group `apache`.

Never ROOT!

The Apache server (`httpd` process) starts with the `root` superuser account. Each client request triggers the creation of a "child" process. To limit risks, these child processes are launched from a less privileged account.

The User and Group directives declare the account and group used to create child processes.

This account and group must exist in the system (by default, this happens during installation).

File permissions

As a general security rule, web server content must not belong to the process running the server. In our case, the files should not belong to the `apache` user and group since it has written access to the folders.

You assign the contents to the unprivileged user, the root user, and the associated group. Incidentally, you also take the opportunity to restrict the group's access rights.

```
cd /var/www/html
sudo chown -R root:root ./*
sudo find ./ -type d -exec chmod 0755 "{}" \;
sudo find ./ -type f -exec chmod 0644 "{}" \;
```

6. Part 2.2 Web Servers Nginx

6.1 Nginx web server

In this chapter, you will learn about the web server Nginx.

Objectives: You will learn how to:

✓ install and configure Nginx

🏁 **nginx**, **http**

Knowledge: ⭐⭐
Complexity: ⭐⭐

Reading time: 15 minutes

6.1.1 Generalities

Nginx is a **free HTTP web server under BSD license**. It was first developed in Russia in 2002 by Igor Sysoev. In addition to the standard features of a web server, Nginx provides a **reverse proxy** for the **HTTP** protocol, and a proxy for the **POP** and **IMAP** messaging protocols.

The development of the Nginx server is a response to the **C10K** problem, which supports ten thousand concurrent connections (standard on the modern web). This is a real challenge for web servers.

Commercial support is available from Nginx Inc.

The server's internal architecture enables **very high performance** with **low memory consumption** compared to the Apache web server.

Modules complementing the essential functions of the Nginx kernel are compile-time bound. This means activation or deactivation cannot happen dynamically.

A master process controls server processes, making it possible to **modify configuration or update software without stopping the service**.

Nginx has a significant % market share of 28% on the busiest sites, just behind Apache (41%).

Features

Nginx offers the following basic functions:

- Hosting of static web pages
- Automatic index page generation
- Accelerated reverse proxy with cache
- Load balancing
- Fault tolerance
- Cached support for FastCGI, uWSGI, SCGI, and Memcached cache server
- Various filters for gzip, xslt, ssi, image transformation, and more
- Support for SSL/TLS and SNI
- HTTP/2 support

Other features:

- Hosting by name or IP address
- Keepalive management of client connections
- Log management: syslog, rotation, buffer
- URI rewriting
- Access control: by IP, password, and more
- FLV and MP4 streaming

6.1.2 Installation

Nginx is available directly from the app stream repository, and more recent versions are available as a dnf module.

```
sudo dnf install nginx
sudo systemctl enable nginx --now
```

6.1.3 Configuration

The location of the Nginx configuration is in `/etc/nginx/nginx.conf`.

This configuration file is a global server configuration file. Settings affect the entire server.

> **Note**
>
> The .htaccess file functionality known to Apache administrators does not exist in Nginx!

Provided here is the `nginx.conf` file, stripped of all comments:

```
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;
include /usr/share/nginx/modules/*.conf;
events {
    worker_connections 1024;
}
http {
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';
    access_log  /var/log/nginx/access.log  main;
    sendfile            on;
    tcp_nopush          on;
    tcp_nodelay         on;
    keepalive_timeout   65;
    types_hash_max_size 4096;
    include             /etc/nginx/mime.types;
    default_type        application/octet-stream;
    include /etc/nginx/conf.d/*.conf;
    server {
        listen       80;
        listen       [::]:80;
        server_name  _;
        root         /usr/share/nginx/html;
        include /etc/nginx/default.d/*.conf;
        error_page 404 /404.html;
        location = /404.html {
```

```
        }
        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }
    }
}
```

Default configuration guidelines:

Directive	Description
`user`	Defines the process owner `user` and `group`. If the group is not specified, the group with the same name as the user is used.
`worker_processes`	Defines the number of processes. The optimum value depends on many factors, such as the number of CPU cores and hard disk specifications. In case of doubt, the Nginx documentation suggests a starting value equivalent to the number of CPU cores available (the auto value will try to determine this).
`pid`	Defines a file to store the PID value.
`worker_connections`	Sets the maximum number of simultaneous connections a worker process can open (to the client and mandated servers).
`tcp_nopush`	`tcp_nopush` is inseparable from the sendfile option. It is used to optimize the quantity of information sent simultaneously. Packets are only sent when they have reached their maximum size.
`tcp_nodelay`	Activating `tcp_nodelay` forces data in the socket to be sent immediately, regardless of packet size, which is the opposite of what `tcp_nopush` does.
`sendfile`	Optimizes the sending of static files (this option is not required for a proxy-inverse configuration). If sendfile is enabled, Nginx ensures that all packets are completed before they are sent to the client (thanks to `tcp_nopush`). When the last packet arrives, Nginx disables `tcp_nopush` and forces data to be sent using `tcp_nodelay`.
`keepalive_timeout`	The maximum time before closing an inactive connection.
`types_hash_max_size`	Nginx maintains hash tables containing static information. Set the maximum size of the hash table.
`include`	Includes another file or files that match the template provided in the configuration.
`default_type`	Default MIME type of a request.
`ssl_protocols`	Accepted TLS protocol versions.
`ssl_prefer_server_ciphers`	Prefers server cipher suite to client cipher suite.
`access_log`	Configures access logs (see “log management” paragraph).
`error_log`	Configures error logs (see “log management” paragraph).
`gzip`	The ngx_http_gzip_module is a filter that compresses data transmitted in gzip format.
`gzip_disable`	Disables gzip based on a regular expression.

The structure of the Nginx configuration is:

```
# global directives

events {
    # worker configuration
}

http {
    # http service configuration

    # Configure the first server listening on port 80
    server {
        listen 80 default_server;
        listen [::]:80 default_server;
        root /var/www/html;
        index index.html index.htm;
        server_name _;
        location / {
            try_files $uri $uri/ =404;
        }
    }
}

mail {
    # mail service configuration

# global mail service directives
   server {
        # A first server listening on the pop protocol
        listen     localhost:110;
        protocol   pop3;
        proxy      on;
   }


   server {
        # A second server listening on the imap protocol
       listen     localhost:143;
       protocol   imap;
       proxy      on;
   }
}
```

6.1.4 https configuration

To configure an HTTPS service, you must add a server block or modify an existing one. A server block can listen on both port 443 and port 80.

You can add this block, for example, to the new `/etc/nginx/conf.d/default_https.conf` file:

```
server {
    listen              443 ssl default_server;
    ssl_protocols       TLSv1.3 TLSv1.2 TLSv1.1
    ssl_certificate     /path/to/cert.pem;
    ssl_certificate_key /path/to/key.key;
    root                /var/www/html;
    index               index.html index.htm;
    server_name         _;
    location / {
        try_files       $uri $uri/ =404;
    }
}
```

or you can modify the default server to support HTTPS:

```
server {
    listen              80;
    listen              443 ssl;
    server_name         _;
    ssl_protocols       TLSv1.3 TLSv1.2 TLSv1.1
    ssl_certificate     /path/to/cert.pem;
    ssl_certificate_key /path/to/key.key;
    ...
}
```

6.1.5 Log management

You can configure the `error_log` directive for error logs.

Syntax of the `error_log` directive:

```
error_log file [level];
```

The first parameter defines a file to receive error logs.

The second parameter determines the log level: debug, info, notice, warn, error, crit, alert, or emerg (see syslog chapter of our admin guide).

The function of sending logs to syslog is with the “syslog:” prefix.

```
access_log syslog:server=192.168.1.100:5514,tag=nginx debug;
```

6.1.6 Nginx as a reverse proxy

Reverse proxy functionality is with the `ngx_http_upstream_module`. It lets you define groups of servers which are then called by the `proxy_pass` or `fastcgi_pass` directives, `memcached_pass`, and more.

Example of a basic configuration, which distributes the load 2/3 to the first server and 1/3 to the second application server:

```
    upstream frontservers {
        server front1.rockylinux.lan:8080       weight=2;
        server front2.rockylinux.lan:8080       weight=1;
    }

    server {
        location / {
            proxy_pass http://docs.rockylinux.lan;
        }
    }
```

You can declare servers as backups:

```
    upstream frontservers {
        ...
        server front3.rockylinux.lan:8080   backup;
        server front4.rockylinux.lan:8080   backup;
    }
```

The server directive accepts many arguments:

- `max_fails=numberofattempts`: sets the number of connection attempts that must fail during the time period defined by the `fail_timeout` parameter for the server to be considered unavailable. The default value is 1; 0 disables functionality.
- `fail_timeout=time`: sets the time during which a defined number of connections will cause the server to be unavailable, and sets the period of time during which the server will be considered unavailable. The default value is 10 seconds.

7. Part 3. Application servers

7.1 PHP and PHP-FPM

In this chapter, you will learn about PHP and PHP-FPM.

PHP (**P**HP **H**ypertext **P**reprocessor) is a source scripting language specially designed for web application development. In 2024, PHP represented a little less than 80% of the web pages generated in the world. PHP is open-source and is the core of the most famous CMS (WordPress, Drupal, Joomla!, Magento, and others.).

PHP-FPM (**F**astCGI **P**rocess **M**anager) is integrated to PHP since its version 5.3.3. The FastCGI version of PHP brings additional functionalities.

Objectives: You will learn how to:

✔ install a PHP application server
✔ configure PHP-FPM pool
✔ optimize a PHP-FPM application server

🏁 **PHP**, **PHP-FPM**, **Application server**

Knowledge: ★★★
Complexity: ★★★

Reading time: 30 minutes

7.1.1 Generalities

CGI (**C**ommon **G**ateway **I**nterface) and **FastCGI** allow communication between the web server (Apache or Nginx) and a development language (PHP, Python, Java):

- In the case of **CGI**, each request creates a **new process**, which is less efficient in performance.
- **FastCGI** relies on a **certain number of processes** to treat its client requests.

PHP-FPM, **in addition to better performances**, brings:

- The possibility of better **partitioning the applications**: launching processes with different uid/gid, with personalized `php.ini` files,
- The management of the statistics,
- Log management,
- Dynamic management of processes and restart without service interruption ('graceful').

> **Note**
>
> Since Apache has a PHP module, php-fpm is more commonly used on an Nginx server.

7.1.2 Choose a PHP version

Rocky Linux, like its upstream, offers many versions of the language. Some of them have reached the end of their life but are kept to continue hosting historical applications that are not yet compatible with new versions of PHP. Please refer to the supported versions page of the php.net website to choose a supported version.

To obtain a list of available versions, enter the following command:

9.3 PHP module list

```
$ sudo dnf module list php

Rocky Linux 9 - AppStream
Name
Stream
Profiles
Summary
php                                                 8.1
[d]                                                 common [d], devel, minimal

Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

The Remi repository offers more recent releases of PHP than the Appstream repository, including versions 8.2 and 8.3.

To install the Remi repository, run the following command:

```
sudo dnf install https://rpms.remirepo.net/enterprise/remi-release-9.rpm
```

Enable the Remi repository by running the following command:

```
sudo dnf config-manager --set-enabled remi
```

You can now activate a newer module (PHP 8.3) by entering the following command:

```
sudo dnf module enable php:remi-8.3
```

8.9 PHP module list

```
$ sudo dnf module list php

Rocky Linux 8 - AppStream
Name
Stream
Profiles
Summary
php                                                  7.2
[d]                                                 common [d], devel,
minimal                                                PHP scripting language
php                                                  7.
3                                                    common [d], devel,
minimal                                                PHP scripting language
php                                                  7.
4                                                    common [d], devel,
minimal                                                PHP scripting language
php                                                  8.
0                                                    common [d], devel,
minimal                                                PHP scripting language

Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

Rocky provides different PHP modules from its AppStream repository.

You will note that Rocky 8.9's default version is 7.2, which has already reached its end of life at the time of this writing.

You can activate a newer module by entering the following command:

```
sudo dnf module enable php:8.0
================================================================================
===============
Package               Architecture         Version
Repository             Size
================================================================================
===============
Enabling module streams:
httpd                                      2.4
nginx                                      1.14
php                                        8.0

Transaction Summary
================================================================================
===============

Is this ok [y/N]:

Transaction Summary
================================================================================
===============

Is this ok [y/N]: y
Complete!
```

You can now proceed to the installation of the PHP engine.

7.1.3 Installation of the PHP CGI mode

First, install and use PHP in CGI mode. It can only work with the Apache web server and its `mod_php` module. This document's FastCGI part (php-fpm) explains how to integrate PHP in Nginx (but also Apache).

The installation of PHP is relatively trivial. It consists of installing the main package and the few modules you will need.

The example below installs PHP with the modules usually installed with it.

9.3 install PHP

```
sudo dnf install php php-cli php-gd php-curl php-zip php-mbstring
```

During installation, you will be prompted to import GPG keys for the epel9 (Extra Packages for Enterprise Linux 9) and Remi repositories. Enter y to import the keys:

```
Extra Packages for Enterprise Linux 9 - x86_64
Importing GPG key 0x3228467C:
Userid     : "Fedora (epel9) <epel@fedoraproject.org>"
Fingerprint: FF8A D134 4597 106E CE81 3B91 8A38 72BF 3228 467C
From       : /etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-9
Is this ok [y/N]: y
Key imported successfully
Remi's RPM repository for Enterprise Linux 9 - x86_64
Importing GPG key 0x478F8947:
Userid     : "Remi's RPM repository (https://rpms.remirepo.net/)
<remi@remirepo.net>"
Fingerprint: B1AB F71E 14C9 D748 97E1 98A8 B195 27F1 478F 8947
From       : /etc/pki/rpm-gpg/RPM-GPG-KEY-remi.el9
Is this ok [y/N]: y
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.

Complete!
```

8.9 install PHP

```
sudo dnf install php php-cli php-gd php-curl php-zip php-mbstring
```

You can check that the installed version corresponds to the expected one:

9.3 check PHP version

```
$ php -v
PHP 8.3.2 (cli) (built: Jan 16 2024 13:46:41) (NTS gcc x86_64)
Copyright (c) The PHP Group
Zend Engine v4.3.2, Copyright (c) Zend Technologies
with Zend OPcache v8.3.2, Copyright (c), by Zend Technologies
```

8.9 check PHP version

```
$ php -v
PHP 7.4.19 (cli) (built: May  4 2021 11:06:37) ( NTS )
```

```
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
with Zend OPcache v7.4.19, Copyright (c), by Zend Technologies
```

7.1.4 Apache Integration

To serve PHP pages in CGI mode, you must install the Apache server, configure it, activate it, and start it.

- Installation:

```
sudo dnf install httpd
```

```
activation:
```

```
sudo systemctl enable --now httpd
sudo systemctl status httpd
```

- Do not forget to configure the firewall:

```
sudo firewall-cmd --add-service=http --permanent
sudo firewall-cmd --reload
```

The default vhost should work out of the box. PHP provides a `phpinfo()` function that generates a summary table of its configuration. It is useful to test whether PHP is working well. However, be careful not to leave such test files on your servers. They represent a huge security risk for your infrastructure.

Create the file `/var/www/html/info.php` (`/var/www/html` being the default vhost directory of the default Apache configuration):

```
<?php
phpinfo();
?>
```

Use a web browser to check that the server works properly by going to the page http://your-server-ip/info.php.

> **Warning**
>
> Do not leave the `info.php` file on your server!

7.1.5 Installation of the PHP cgi mode (PHP-FPM)

Noted earlier, many advantages exist for switching web hosting to PHP-FPM mode.

The installation entails only the php-fpm package:

```
sudo dnf install php-fpm
```

As php-fpm is a service from a system point of view, you must activate and start it:

```
sudo systemctl enable --now php-fpm
sudo systemctl status php-fpm
```

Configuration of the PHP cgi mode

The main configuration file is `/etc/php-fpm.conf`.

```
include=/etc/php-fpm.d/*.conf
[global]
pid = /run/php-fpm/php-fpm.pid
error_log = /var/log/php-fpm/error.log
daemonize = yes
```

> **Note**
>
> The php-fpm configuration files are widely commented on. Go and have a look!

As you can see, the files in the `/etc/php-fpm.d/` directory with the `.conf` extension are always included.

By default, a PHP process pool declaration named `www`, is in `/etc/php-fpm.d/www.conf`.

```
[www]
user = apache
group = apache
```

```
listen = /run/php-fpm/www.sock
listen.acl_users = apache,nginx
listen.allowed_clients = 127.0.0.1

pm = dynamic
pm.max_children = 50
pm.start_servers = 5
pm.min_spare_servers = 5
pm.max_spare_servers = 35

slowlog = /var/log/php-fpm/www-slow.log

php_admin_value[error_log] = /var/log/php-fpm/www-error.log
php_admin_flag[log_errors] = on
php_value[session.save_handler] = files
php_value[session.save_path]    = /var/lib/php/session
php_value[soap.wsdl_cache_dir]  = /var/lib/php/wsdlcache
```

Instructions	Description
`[pool]`	Process pool name. The configuration file can comprise several process pools (the pool's name in brackets starts a new section).
`listen`	Defines the listening interface or the Unix socket used.

Configuring the way to access php-fpm processes

Two ways exist for connecting.

With an `inet-interface` such as:

`listen = 127.0.0.1:9000`.

Or with a UNIX socket:

`listen = /run/php-fpm/www.sock`.

> **Note**
>
> Using a socket when the web server and PHP server are on the same machine removes the TCP/IP layer and optimizes the performance.

When working with an interface, you have to configure `listen.owner`, `listen.group`, `listen.mode` to specify the owner, the owner group, and the rights of the UNIX socket. **Warning:** Both servers (web and PHP) must have access rights on the socket.

When working with a socket, you must configure `listen.allowed_clients` to restrict access to the PHP server to certain IP addresses.

Example: `listen.allowed_clients = 127.0.0.1`

Static or dynamic configuration

You can manage PHP-FPM processes statically or dynamically.

In static mode, `pm.max_children` sets a limit to the number of child processes:

```
pm = static
pm.max_children = 10
```

This configuration starts with 10 processes.

In dynamic mode, PHP-FPM starts at *most* the number of processes specified by the `pm.max_children` value. It first starts some processes corresponding to `pm.start_servers`, keeping at least the value of `pm.min_spare_servers` of inactive processes and, at most, `pm.max_spare_servers` of inactive processes.

Example:

```
pm = dynamic
pm.max_children = 5
pm.start_servers = 2
pm.min_spare_servers = 1
pm.max_spare_servers = 3
```

PHP-FPM will create a new process to replace one that has processed several requests equivalent to `pm.max_requests`.

By default, the value of `pm.max_requests` is 0, meaning processes are never recycled. The `pm.max_requests` option can be attractive for applications with memory leaks.

A third mode of operation is the `ondemand` mode. This mode only starts a process when it receives a request. It is not an optimal mode for sites with strong influences and is reserved for specific needs (sites with feeble requests, management backend, etc.).

Note

The configuration of the operating mode of PHP-FPM is essential to ensure the optimal functioning of your web server.

Process status

Like Apache and its `mod_status` module, PHP-FPM offers a page indicating the process's status.

To activate the page, set its access path with the `pm.status_path` directive:

```
pm.status_path = /status
```

```
$ curl http://localhost/status_php
pool:                 www
process manager:      dynamic
start time:           03/Dec/2021:14:00:00 +0100
start since:          600
accepted conn:        548
listen queue:         0
max listen queue:     15
listen queue len:     128
idle processes:       3
active processes:     3
total processes:      5
max active processes: 5
max children reached: 0
slow requests:        0
```

Logging long requests

The `slowlog` directive specifies the file that receives logging requests that are too long (for instance, whose time exceeds the value of the `request_slowlog_timeout` directive).

The default location of the generated file is `/var/log/php-fpm/www-slow.log`.

```
request_slowlog_timeout = 5
slowlog = /var/log/php-fpm/www-slow.log
```

A value of 0 for `request_slowlog_timeout` disables logging.

7.1.6 NGinx integration

The default setting of nginx already includes the necessary configuration to make PHP work with PHP-FPM.

The configuration file `fastcgi.conf` (or `fastcgi_params`) is under `/etc/nginx/`:

```
fastcgi_param  SCRIPT_FILENAME    $document_root$fastcgi_script_name;
fastcgi_param  QUERY_STRING       $query_string;
fastcgi_param  REQUEST_METHOD     $request_method;
fastcgi_param  CONTENT_TYPE       $content_type;
fastcgi_param  CONTENT_LENGTH     $content_length;

fastcgi_param  SCRIPT_NAME        $fastcgi_script_name;
fastcgi_param  REQUEST_URI        $request_uri;
fastcgi_param  DOCUMENT_URI       $document_uri;
fastcgi_param  DOCUMENT_ROOT      $document_root;
fastcgi_param  SERVER_PROTOCOL    $server_protocol;
fastcgi_param  REQUEST_SCHEME     $scheme;
fastcgi_param  HTTPS              $https if_not_empty;

fastcgi_param  GATEWAY_INTERFACE  CGI/1.1;
fastcgi_param  SERVER_SOFTWARE    nginx/$nginx_version;

fastcgi_param  REMOTE_ADDR        $remote_addr;
fastcgi_param  REMOTE_PORT        $remote_port;
fastcgi_param  SERVER_ADDR        $server_addr;
fastcgi_param  SERVER_PORT        $server_port;
fastcgi_param  SERVER_NAME        $server_name;

# PHP only, required if PHP was built with --enable-force-cgi-redirect
fastcgi_param  REDIRECT_STATUS    200;
```

For nginx to process `.php` files, add the following directives to the site configuration file:

If PHP-FPM is listening on port 9000:

```
location ~ \.php$ {
  include /etc/nginx/fastcgi_params;
  fastcgi_pass 127.0.0.1:9000;
}
```

If php-fpm is listening on a UNIX socket:

```
location ~ \.php$ {
  include /etc/nginx/fastcgi_params;
  fastcgi_pass unix:/run/php-fpm/www.sock;
}
```

7.1.7 Apache integration

The configuration of Apache to use a PHP pool is quite simple. You have to use the proxy modules with a `ProxyPassMatch` directive, for example:

```
<VirtualHost *:80>
  ServerName web.rockylinux.org
  DocumentRoot "/var/www/html/current/public"

  <Directory "/var/www/html/current/public">
    AllowOverride All
    Options -Indexes +FollowSymLinks
    Require all granted
  </Directory>
  ProxyPassMatch ^/(.*\.php(/.*)?)$ "fcgi://127.0.0.1:9000/var/www/html/
current/public"

</VirtualHost>
```

7.1.8 Solid configuration of PHP pools

Optimizing the number of requests served and analyzing the memory used by the PHP scripts is necessary to maximize the number of launched threads.

First of all, you need to know the average amount of memory used by a PHP process with the command:

```
while true; do ps --no-headers -o "rss,cmd" -C php-fpm | grep "pool www" | awk
'{ sum+=$1 } END { printf ("%d%s\n", sum/NR/1024,"Mb") }' >> avg_php_proc;
sleep 60; done
```

This will give you a pretty accurate idea of the average memory footprint of a PHP process on this server.

The rest of this document results in a memory footprint of 120MB per process at full load.

On a server with 8Gb of RAM, keeping 1Gb for the system and 1Gb for the OPCache (see the rest of this document), is 6Gb left to process PHP requests from clients.

You can conclude that this server can accept at most **50 threads** `((6*1024) / 120)`.

An exemplary configuration of `php-fpm` specific to this use case is:

```
pm = dynamic
pm.max_children = 50
pm.start_servers = 12
pm.min_spare_servers = 12
pm.max_spare_servers = 36
pm.max_requests = 500
```

with:

- `pm.start_servers` = 25% of `max_children`
- `pm.min_spare_servers` = 25% of `max_children`
- `pm.max_spare_servers` = 75% of `max_children`

7.1.9 Opcache configuration

The `opcache` (Optimizer Plus Cache) is the first level of cache that you can influence.

It keeps the compiled PHP scripts in memory, which strongly impacts the execution of the web pages (removes the reading of the script on disk + the compilation time).

To configure it, you must work on:

- The size of the memory dedicated to the opcache according to the hit ratio, configuring it correctly
- The number of PHP scripts to cache (number of keys + maximum number of scripts)
- The number of strings to cache

To install it:

```
sudo dnf install php-opcache
```

To configure it, edit the `/etc/php.d/10-opcache.ini` configuration file:

```
opcache.memory_consumption=128
opcache.interned_strings_buffer=8
opcache.max_accelerated_files=4000
```

Where:

- `opcache.memory_consumption` corresponds to the amount of memory needed for the opcache (increase this until obtaining a correct hit ratio).
- `opcache.interned_strings_buffer` is the amount of strings to cache.
- `opcache.max_accelerated_files` is near to the result of the `find ./ -iname "*.php"|wc -l` command.

To configure the opcache, refer to an `info.php` page (including the `phpinfo();`) (see, for example, the values of `Cached scripts` and `Cached strings`).

Note

At each new deployment of new code, it will be necessary to empty the opcache (for example by restarting the php-fpm process).

Note

Do not underestimate the speed gain that can be achieved by setting up and configuring the opcache correctly.

8. Part 4.1 Database servers MariaDB

MySQL, MariaDB and PostgreSQL are open-source RDBMS (Relational DataBase Management System).

8.1 MariaDB and MySQL

In this chapter, you will learn about the RDBMS MariaDB and MySQL.

Objectives: You will learn how to:

✓ install, configure, and secure MariaDB server and MySQL server;
✓ perform some administrative actions on databases and users.

🏁 **RDBMS**, **database**, **MariaDB**, **MySQL**

Knowledge: ★ ★ ★
Complexity: ★ ★ ★

Reading time: 30 minutes

8.1.1 Generalities

MySQL was developed by Michael "Monty" Widenius (a Finnish computer scientist),, who founded MySQL AB in 1995. MySQL AB was acquired by SUN in 2008, which in turn was acquired by Oracle in 2009. Oracle still owns the MySQL software and distributes it under a dual GPL and proprietary license.

In 2009, Michael Widenius left SUN, founded Monty Program AB, and launched the development of his community fork of MySQL: MariaDB under a GPL license. The MariaDB Foundation governs the project and ensures that it remains free.

It was not long before most Linux distributions offered MariaDB packages instead of MySQL ones, and major accounts such as Wikipedia and Google also adopted the community fork.

MySQL and MariaDB are among the world's most widely used RDBMSs (professionally and by the general public), particularly for web applications (**LAMP**: Linux + Apache + Mysql-MariaDB + Php).

Mysql-MariaDB's main competitors are:

- PostgreSQL,
- OracleDB,
- Microsoft SQL Server.

Database services are multi-threaded and multi-user, run on most operating systems (Linux, Unix, BSD, Mac OSx, Windows), and are accessible from many programming languages (PHP, Java, Python, C, C++, Perl, others).

Support is offered for several engines, enabling the assignment of different engines to different tables within the same database, depending on requirements:

MyISAM

: the simplest, but does not support transactions or foreign keys. It is an indexed sequential engine. MyISAM is now deprecated.

InnoDB

: manages table integrity (foreign keys and transactions), but takes up more disk space. This has been the default engine since MySQL version 5.6. It is a transactional engine.

Memory

: tables are stored in memory.

Archive

: data compression on insertion saves disk space but slows down search queries (cold data).

It is a matter of adopting an engine according to need: Archive for log storage, Memory for temporary data, and so on.

MariaDB/MySQL uses port 3306/TCP for network communication.

This chapter will deal with this version as the default version supplied with Rocky is the MariaDB community version of the database. Only the differences between MySQL and MariaDB are specifically dealt with.

8.1.2 Installation

Use the `dnf` command to install the `mariadb-server` package:

```
sudo dnf install -y mariadb-server
```

By default, the version installed on a Rocky 9 is 10.5.

Activate the service at startup and start it:

```
sudo systemctl enable mariadb --now
```

You can check the status of the `mariadb` service:

```
sudo systemctl status mariadb
```

To install a more recent version, you'll need to use the `dnf` modules:

```
$ sudo dnf module list mariadb
Last metadata expiration check: 0:00:09 ago on Thu Jun 20 11:39:10 2024.
Rocky Linux 9 - AppStream
Name                          Stream
Profiles                                      Summary
mariadb                       10.11                       client, galera,
server [d]                      MariaDB Module

Hint: [d]efault, [e]nabled, [x]disabled, [i]nstalled
```

If you have not yet installed the mariadb server, activating the desired module version will suffice:

```
$ sudo dnf module enable mariadb:10.11
Last metadata expiration check: 0:02:23 ago on Thu Jun 20 11:39:10 2024.
Dependencies resolved.
================================================================================
```

```
================================================================
Package                          Architecture
Version                             Repository                          Size
================================================================================
================================================================
Enabling module streams:
 mariadb                                                          10.11

Transaction Summary
================================================================================
================================================================
Is this ok [y/N]: y
Complete!
```

You can now install the package. The desired version will be automatically installed:

```
sudo dnf install -y mariadb-server
```

About default users

Please note the logs provided by mariadb at first start (`/var/log/messages`):

```
mariadb-prepare-db-dir[6560]: Initializing MariaDB database
mariadb-prepare-db-dir[6599]: Two all-privilege accounts were created.
mariadb-prepare-db-dir[6599]: One is root@localhost, it has no password, but
you need to
mariadb-prepare-db-dir[6599]: be system 'root' user to connect. Use, for
example, sudo mysql
mariadb-prepare-db-dir[6599]: The second is mysql@localhost, it has no password
either, but
mariadb-prepare-db-dir[6599]: you need to be the system 'mysql' user to
connect.
mariadb-prepare-db-dir[6599]: After connecting you can set the password, if you
would need to be
mariadb-prepare-db-dir[6599]: able to connect as any of these users with a
password and without sudo
```

8.1.3 Configuration

Configuration files can are in `/etc/my.cnf` and `/etc/my.cnf.d/`.

Some important default options have been setup in the `/etc/my.cnf.d/mariadb-server.cnf`:

```
[server]

[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
log-error=/var/log/mariadb/mariadb.log
pid-file=/run/mariadb/mariadb.pid
...
```

As you can see, data is in the `/var/lib/mysql` per default. This folder can require a lot of storage space and recurring volume increases. It is therefore advisable to mount this folder on a dedicated partition.

8.1.4 Security

MariaDB and Mysql include a script to help you secure your server. It remove for example remote root logins and sample users, the less-secure default options.

Use the `mariadb-secure-installation` and secure your server:

```
sudo mariadb-secure-installation
```

The script will prompt you to provide a password for your root user.

Note

The `mysql_secure_installation` command is now a symlink to the `mariadb-secure-installation` command:

```
$ ll /usr/bin/mysql_secure_installation
lrwxrwxrwx. 1 root root 27 Oct 12  2023 /usr/bin/mysql_secure_installation -> mariadb-secure-installation
```

If providing a password each time you have to use mariadb's commands is a problem, you can set up a `~/.my.cnf` file with your credentials, that will be used per default by mariadb to connect to your server.

```
[client]
user="root"
password="#######"
```

Ensure the permissions are restrictive enough to only allow the current user can access:

```
chmod 600 ~/.my.cnf
```

Warning

This is not the best way. There is another solution more secure than storing a password in plain text. Since MySQL 5.6.6, it is now possible to store your credentials in an encrypted login `.mylogin.cnf`, thanks to the `mysql_config_editor` command.

If your server runs a firewall (which is a good thing), you might need to consider opening it, but only if you need your service accessible from the outside.

```
sudo firewall-cmd --zone=public --add-service=mysql
sudo firewall-cmd --reload
```

Note

The best security is not to open your database server to the outside world (if the application server is hosted on the same server), or to restrict access to authorized IPs only.

8.1.5 Administration

The `mariadb` command

The `mariadb` command is a simple SQL shell that supports interactive and non-interactive use.

```
mysql -u user -p [base]
```

Option	Information
`-u user`	Provides a username to connect with.
`-p`	Asks for a password.
`base`	The database to connect to.

Note

The `mysql` command is now a symlink to the `mariadb` command:

```
$ ll /usr/bin/mysql
lrwxrwxrwx. 1 root root 7 Oct 12  2023 /usr/bin/mysql -> mariadb
```

Example:

```
$ sudo mariadb -u root
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 15
Server version: 10.5.22-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
3 rows in set (0.003 sec)
```

The `mariadb-admin` command

The `mariadb-admin` command is a client for administering a MariaDB server.

```
mariadb-admin -u user -p command
```

Option	Information
`-u user`	Provides a username to connect with.
`-p`	Asks for a password.
`command`	A command to execute.

The `mariadb-admin` provides several commands as `version`, `variables`, `stop-slave` or `start-slaves`, `create databasename`, and so on.

Example:

```
mariadb-admin -u root -p version
```

Note

The `mysqladmin` command is now a symlink to the `mariadb-admin` command:

```
$ ll /usr/bin/mysqladmin
lrwxrwxrwx. 1 root root 13 Oct 12  2023 /usr/bin/mysqladmin -> mariadb-admin
```

8.1.6 About logs

MariaDB provides various logs:

- **Error log**: Contains messages generated at service startup and shutdown and important events (warnings and errors).
- **Binary log**: This log (in binary format) records all actions that modify database structure or data. If you need to restore a database, you will need to restore the backup AND replay the binary log to recover the state of the database before the crash.
- **Query log**: All client requests are logged here.
- **Slow requests log**: Slow queries, i.e., those that take longer than a set time to execute, are logged separately in this log. By analyzing this file, you may be able to take steps to reduce execution time (e.g., by setting up indexes or modifying the client application).

With the exception of the binary log, these logs are in text format, so they can be used directly!

To enable logging of long requests, edit the `my.cnf` configuration file to add the following lines:

```
slow_query_log      = 1
slow_query_log_file = /var/log/mysql/mysql-slow.log
long_query_time     = 2
```

The minimum value for the `long_query_time` variable is 0, and the default value is `10` seconds.

Restart the service so the changes take effect.

Once the log file is full, you can analyze it with the `mariadb-dumpslow` command.

```
mariadb-dumpslow [options] [log_file ...]
```

Option	Information
`-t n`	Displays only the first n queries.
`-s sort_type`	Sorts by number of queries.
`-r`	Inverts results display.

Sort types can be :

Option	Information
`c`	according to number of requests.
`t` or `at`	according to execution time or average execution time (a for average).
`l` or `al`	according to lock time or its average.
`r` or `aR`	as a function of the number of lines returned or its average.

8.1.7 About backup

As with any RDBMS, backing up a database is done while the data modification is offline. You can do this by the following:

- stopping the service, known as an offline backup;
- while the service runs by temporarily locking out updates (suspending all modifications). This is an online backup.
- using a snapshot of the LVM file system, enabling data backup with a cold file system.

The backup format can be an ASCII (text) file, representing the state of the database and its data in the form of SQL commands or a binary file corresponding to MySQL storage files.

While you can back up a binary file using common utilities such as tar or cpio, an ASCII file requires a utility such as `mariadb-dump`.

The `mariadb-dump` command can perform a dump of your database.

During the process, data access is locked.

```
mariadb-dump -u root -p DATABASE_NAME > backup.sql
```

> **Note**
>
> Do not forget that after restoring a full backup, restoring the binary files (binlogs) completes the reconstitution of the data.

The resulting file is usable to restore the database data. The database must still exist, or you must have recreated it beforehand!:

```
mariadb -u root -p DATABASE_NAME < backup.sql
```

8.1.8 Graphical tools

Graphical tools exist to facilitate the administration and management of database data. Here are a few examples:

- DBeaver

8.1.9 Workshop

In this workshop, you will install, configure, and secure your MariaDB server.

Task 1: Installation

Install the MariaDB-server package:

```
$ sudo dnf install mariadb-server
Last metadata expiration check: 0:10:05 ago on Thu Jun 20 11:26:03 2024.
Dependencies resolved.
================================================================================
================================================================
Package                                      Architecture
Version                                Repository                  Size
================================================================================
================================================================
Installing:
 mariadb-server                               x86_64                 3:
10.5.22-1.el9_2                    appstream                  9.6 M
Installing dependencies:
...
```

Installation adds a `mysql` user to the system, with `/var/lib/mysql` as home directory:

```
$ cat /etc/passwd
...
mysql:x:27:27:MySQL Server:/var/lib/mysql:/sbin/nologin
...
```

Enable and start the service:

```
$ sudo systemctl enable mariadb --now
Created symlink /etc/systemd/system/mysql.service → /usr/lib/systemd/system/
mariadb.service.
Created symlink /etc/systemd/system/mysqld.service → /usr/lib/systemd/system/
mariadb.service.
Created symlink /etc/systemd/system/multi-user.target.wants/mariadb.service → /
usr/lib/systemd/system/mariadb.service.
```

Check the installation:

```
$ sudo systemctl status mariadb
● mariadb.service - MariaDB 10.5 database server
     Loaded: loaded (/usr/lib/systemd/system/mariadb.service; enabled; preset:
disabled)
     Active: active (running) since Thu 2024-06-20 11:48:56 CEST; 1min 27s ago
       Docs: man:mariadbd(8)
             https://mariadb.com
    Process: 6538 ExecStartPre=/usr/libexec/mariadb-check-socket (code=exited,
status=0/SUCCESS)
    Process: 6560 ExecStartPre=/usr/libexec/mariadb-prepare-db-dir
mariadb.service (code=exited, status=0/SUCCESS)
    Process: 6658 ExecStartPost=/usr/libexec/mariadb-check-upgrade
(code=exited, status=0/SUCCESS)
   Main PID: 6643 (mariadbd)
     Status: "Taking your SQL requests now..."
      Tasks: 9 (limit: 11110)
     Memory: 79.5M
        CPU: 1.606s
     CGroup: /system.slice/mariadb.service
             └─6643 /usr/libexec/mariadbd --basedir=/usr

Jun 20 11:48:56 localhost.localdomain mariadb-prepare-db-dir[6599]: The second
is mysql@localhost, it has no password either, but
Jun 20 11:48:56 localhost.localdomain mariadb-prepare-db-dir[6599]: you need
to be the system 'mysql' user to connect.
```

```
Jun 20 11:48:56 localhost.localdomain mariadb-prepare-db-dir[6599]: After
connecting you can set the password, if you would need to be
Jun 20 11:48:56 localhost.localdomain mariadb-prepare-db-dir[6599]: able to
connect as any of these users with a password and without sudo
Jun 20 11:48:56 localhost.localdomain mariadb-prepare-db-dir[6599]: See the
MariaDB Knowledgebase at https://mariadb.com/kb
Jun 20 11:48:56 localhost.localdomain mariadb-prepare-db-dir[6599]: Please
report any problems at https://mariadb.org/jira
Jun 20 11:48:56 localhost.localdomain mariadb-prepare-db-dir[6599]: The latest
information about MariaDB is available at https://mariadb.org>Jun 20 11:48:56
localhost.localdomain mariadb-prepare-db-dir[6599]: Consider joining MariaDB's
strong and vibrant community:
Jun 20 11:48:56 localhost.localdomain mariadb-prepare-db-dir[6599]: https://
mariadb.org/get-involved/
Jun 20 11:48:56 localhost.localdomain systemd[1]: Started MariaDB 10.5
database server.
```

Try connecting to the server:

```
$ sudo mariadb
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 9
Server version: 10.5.22-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
3 rows in set (0.001 sec)

MariaDB [(none)]> exit
Bye
```

```
$ sudo mariadb-admin version
mysqladmin  Ver 9.1 Distrib 10.5.22-MariaDB, for Linux on x86_64
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Server version          10.5.22-MariaDB
```

```
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /var/lib/mysql/mysql.sock
Uptime:                 7 min 24 sec

Threads: 1  Questions: 9  Slow queries: 0  Opens: 17  Open tables: 10  Queries
per second avg: 0.020
```

As you can see, the `root` user does not need to provide a password. You will correct that during the next task.

Task 2: Secure your server

Launch the `mariadb-secure-installation` and follow the instructions:

```
$ sudo mariadb-secure-installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user. If you've just installed MariaDB, and
haven't set the root password yet, you should just press enter here.

Enter current password for root (enter for none):
OK, successfully used password, moving on...

Setting the root password or using the unix_socket ensures that nobody
can log into the MariaDB root user without the proper authorisation.

You already have your root account protected, so you can safely answer 'n'.

Switch to unix_socket authentication [Y/n] y
Enabled successfully!
Reloading privilege tables..
 ... Success!


You already have your root account protected, so you can safely answer 'n'.

Change the root password? [Y/n] y
New password:
Re-enter new password:
Password updated successfully!
Reloading privilege tables..
 ... Success!
```

```
By default, a MariaDB installation has an anonymous user, allowing anyone
to log into MariaDB without having to have a user account created for
them.  This is intended only for testing, and to make the installation
go a bit smoother.  You should remove them before moving into a
production environment.

Remove anonymous users? [Y/n] y
 ... Success!

Normally, root should only be allowed to connect from 'localhost'.  This
ensures that someone cannot guess at the root password from the network.

Disallow root login remotely? [Y/n] y
 ... Success!

By default, MariaDB comes with a database named 'test' that anyone can
access.  This is also intended only for testing, and should be removed
before moving into a production environment.

Remove test database and access to it? [Y/n] y
 - Dropping test database...
 ... Success!
 - Removing privileges on test database...
 ... Success!

Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.

Reload privilege tables now? [Y/n] y
 ... Success!

Cleaning up...

All done!  If you've completed all of the above steps, your MariaDB
installation should now be secure.

Thanks for using MariaDB!
```

Try connecting again, with and without a password to your server:

```
$ mariadb -u root
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password:
NO)

$ mariadb -u root -p
```

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 10.5.22-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Configure your firewall:

```
sudo firewall-cmd --zone=public --add-service=mysql --permanent
sudo firewall-cmd --reload
```

Task 3: Testing the installation

Verify your installation:

```
$ mysqladmin -u root -p version
Enter password:
mysqladmin  Ver 9.1 Distrib 10.5.22-MariaDB, for Linux on x86_64
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Server version          10.5.22-MariaDB
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /var/lib/mysql/mysql.sock
Uptime:                 29 min 18 sec

Threads: 1  Questions: 35  Slow queries: 0  Opens: 20  Open tables: 13
Queries per second avg: 0.019
```

The `version` gives you information about the server.

Task 4: Create a new database and a user

Create a new database:

```
MariaDB [(none)]> create database NEW_DATABASE_NAME;
```

Create a new user and give him all rights on all tables of that database:

```
MariaDB [(none)]> grant all privileges on NEW_DATABASE_NAME.* TO
'NEW_USER_NAME'@'localhost' identified by 'PASSWORD';
```

Replace `localhost` per `%` if you want to grant access from everywhere, or replace per IP address if possible.

You can restrict the privileges granted. There are different types of permissions to offer users:

- **SELECT**: read data
- **USAGE**: authorization to connect to the server (given by default when a new user is created)
- **INSERT**: add new tuples to a table.
- **UPDATE**: modify existing tuples
- **DELETE**: delete tuples
- **CREATE**: create new tables or databases
- **DROP**: delete existing tables or databases
- **ALL PRIVILEGES**: all rights
- **GRANT OPTION**: give or remove rights to other users

Do not forget to reload and apply the new rights:

```
MariaDB [(none)]> flush privileges;
```

Check:

```
$ mariadb -u NEW_USER_NAME -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 8
Server version: 10.5.22-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| NEW_DATABASE_NAME  |
| information_schema |
+--------------------+
2 rows in set (0.001 sec)
```

Add sample data into your database:

```
$ mariadb -u NEW_USER_NAME -p NEW_DATABASE_NAME
MariaDB [NEW_DATABASE_NAME]> CREATE TABLE users(
    id INT NOT NULL AUTO_INCREMENT,
    first_name VARCHAR(30) NOT NULL,
    last_name VARCHAR(30) NOT NULL,
    age INT DEFAULT NULL,
    PRIMARY KEY (id));
Query OK, 0 rows affected (0.017 sec)

MariaDB [NEW_DATABASE_NAME]> INSERT INTO users (first_name, last_name, age)
VALUES ("Antoine", "Le Morvan", 44);
Query OK, 1 row affected (0.004 sec)
```

Task 5: Create a remote user

In this task, you will create a new user, grant access from the remote, and test a connection with that user.

```
MariaDB [(none)]> grant all privileges on NEW_DATABASE_NAME.* TO
'NEW_USER_NAME'@'%' identified by 'PASSWORD';
Query OK, 0 rows affected (0.005 sec)

MariaDB [(none)]> flush privileges;
Query OK, 0 rows affected (0.004 sec)
```

Use this user and the `-h` option to connect remotely to your server:

```
$ mariadb -h YOUR_SERVER_IP -u NEW_USER_NAME -p NEW_DATABASE_NAME
Enter password:
...

MariaDB [NEW_DATABASE_NAME]>
```

Task 6: Perform an upgrade

Enable the module needed:

```
$ sudo dnf module enable mariadb:10.11
[sudo] password for antoine:
Last metadata expiration check: 2:00:16 ago on Thu Jun 20 11:50:27 2024.
Dependencies resolved.
================================================================================
================================================================
Package                          Architecture
Version                             Repository                          Size
================================================================================
================================================================Enabling module
streams:
 mariadb                                                              10.11

Transaction Summary
================================================================================
================================================================
Is this ok [y/N]: y
Complete!
```

Upgrade the packages:

```
$ sudo dnf update mariadb
Last metadata expiration check: 2:00:28 ago on Thu Jun 20 11:50:27 2024.
Dependencies resolved.
================================================================================
================================================================
Package                          Architecture
Version
Repository              Size
================================================================================
================================================================
Upgrading:
 mariadb                          x86_64              3:
10.11.6-1.module+el9.4.0+20012+a68bdff7              appstream              1.
7 M
 mariadb-backup                   x86_64              3:
10.11.6-1.module+el9.4.0+20012+a68bdff7              appstream              6.
7 M
 mariadb-common                   x86_64              3:
10.11.6-1.module+el9.4.0+20012+a68bdff7              appstream
28 k
 mariadb-errmsg                   x86_64              3:
```

```
10.11.6-1.module+el9.4.0+20012+a68bdff7                appstream
254 k
 mariadb-gssapi-server                x86_64              3:
10.11.6-1.module+el9.4.0+20012+a68bdff7                appstream
15 k
 mariadb-server                       x86_64              3:
10.11.6-1.module+el9.4.0+20012+a68bdff7                appstream
10 M
 mariadb-server-utils                 x86_64              3:
10.11.6-1.module+el9.4.0+20012+a68bdff7                appstream
261 k

Transaction Summary
================================================================================
================================================================
Upgrade  7 Packages

Total download size: 19 M
Is this ok [y/N]: y
Downloading Packages:
(1/7): mariadb-gssapi-
server-10.11.6-1.module+el9.4.0+20012+a68bdff7.x86_64.rpm
99 kB/s |  15 kB     00:00
(2/7): mariadb-server-
utils-10.11.6-1.module+el9.4.0+20012+a68bdff7.x86_64.rpm
1.1 MB/s | 261 kB     00:00
(3/7): mariadb-
errmsg-10.11.6-1.module+el9.4.0+20012+a68bdff7.x86_64.rpm
2.5 MB/s | 254 kB     00:00
(4/7): mariadb-
common-10.11.6-1.module+el9.4.0+20012+a68bdff7.x86_64.rpm
797 kB/s |  28 kB     00:00
(5/7):
mariadb-10.11.6-1.module+el9.4.0+20012+a68bdff7.x86_64.rpm
5.7 MB/s | 1.7 MB     00:00
(6/7): mariadb-
server-10.11.6-1.module+el9.4.0+20012+a68bdff7.x86_64.rpm
9.5 MB/s |  10 MB     00:01
(7/7): mariadb-
backup-10.11.6-1.module+el9.4.0+20012+a68bdff7.x86_64.rpm
7.7 MB/s | 6.7 MB     00:00
--------------------------------------------------------------------------------
----------------------------------------------------------------
Total
13 MB/s |  19 MB     00:01
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
```

```
Running transaction

...

Complete!
```

Your databases now need upgrading (check your `/var/log/messages` as the service complaints):

```
mariadb-check-upgrade[8832]: The datadir located at /var/lib/mysql needs to be
upgraded using 'mariadb-upgrade' tool. This can be done using the following
steps:
mariadb-check-upgrade[8832]:  1. Back-up your data before with 'mariadb-
upgrade'
mariadb-check-upgrade[8832]:  2. Start the database daemon using 'systemctl
start mariadb.service'
mariadb-check-upgrade[8832]:  3. Run 'mariadb-upgrade' with a database user
that has sufficient privileges
mariadb-check-upgrade[8832]: Read more about 'mariadb-upgrade' usage at:
mariadb-check-upgrade[8832]: https://mariadb.com/kb/en/mysql_upgrade/
```

Do not forget to execute the upgrade script provided by MariaDB:

```
sudo mariadb-upgrade
Major version upgrade detected from 10.5.22-MariaDB to 10.11.6-MariaDB. Check
required!
Phase 1/8: Checking and upgrading mysql database
Processing databases
mysql
mysql.column_stats                                 OK
mysql.columns_priv                                 OK
mysql.db                                           OK
...
Phase 2/8: Installing used storage engines... Skipped
Phase 3/8: Running 'mysql_fix_privilege_tables'
Phase 4/8: Fixing views
mysql.user                                         OK
...
Phase 5/8: Fixing table and database names
Phase 6/8: Checking and upgrading tables
Processing databases
NEW_DATABASE_NAME
information_schema
performance_schema
sys
sys.sys_config                                     OK
```

```
Phase 7/8: uninstalling plugins
Phase 8/8: Running 'FLUSH PRIVILEGES'
OK
```

Task 6: Perform a dump

The `mariadb-dump` command can perform a dump of your database.

```
mariadb-dump -u root -p NEW_DATABASE_NAME > backup.sql
```

Verify:

```
cat backup.sql
-- MariaDB dump 10.19  Distrib 10.11.6-MariaDB, for Linux (x86_64)
--
-- Host: localhost    Database: NEW_DATABASE_NAME
-- ------------------------------------------------------
-- Server version       10.11.6-MariaDB

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
...

--
-- Table structure for table `users`
--

DROP TABLE IF EXISTS `users`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(30) NOT NULL,
  `last_name` varchar(30) NOT NULL,
  `age` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=latin1
COLLATE=latin1_swedish_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `users`
--

LOCK TABLES `users` WRITE;
/*!40000 ALTER TABLE `users` DISABLE KEYS */;
```

```
INSERT INTO `users` VALUES
(1,'Antoine','Le Morvan',44);
/*!40000 ALTER TABLE `users` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

...
-- Dump completed on 2024-06-20 14:32:41
```

8.1.10 Check your Knowledge

✔ Which database version is installed by default?

MySQL 5.5

MariaDB 10.5

MariaDB 11.11

Mysql 8

✔ Which command do you use to apply rights changes?

flush rights

flush privileges

mariadb reload

apply

8.1.11 Conclusion

In this chapter, you have installed and secured a MariaDB database server and created a database and a dedicated user.

These skills are a prerequisite for the administration of your databases.

In the next section, you will see how to install the MySQL database instead of the MariaDB fork.

9. Part 4.2 Database Servers MySQL

9.1 MySQL

In this chapter, you will learn how to install MySQL server.

> **Note**
>
> Only notable differences between the MariaDB and MySQL versions are included.

Objectives: You will learn how to:

✔ install, configure, and secure the MariaDB server and MySQL server;

🏁 **RDBMS**, **database**, **MariaDB**, **MySQL**

Knowledge: ★★★
Complexity: ★★★

Reading time: 10 minutes

9.1.1 Installation of MySQL

By default, the installed version of MySQL is version 8.0.

This time, you have to install the `mysql-server` package:

```
sudo dnf install mysql-server
```

and start the `mysqld` service:

```
sudo systemctl enable mysqld.service --now
```

You can now follow the previous chapter by replacing the following commands:

- `mariadb` => `mysql`
- `mariadb-admin` => `mysql_admin`
- `mariadb-dump` => `mysql_dump`
- `mariadb-secure-installation` => `mysql_secure_installation`

You will have to use a different repository to install the latest version of MySQL server.

Visit this page: https://dev.mysql.com/downloads/repo/yum/ and copy the repository URL.

For example:

```
sudo dnf install -y https://dev.mysql.com/get/mysql84-community-release-
el9-1.noarch.rpm
```

When completed, you can perform the `dnf update`:

```
$ dnf update
Error: This command has to be run with superuser privileges (under the root
user on most systems).
[antoine@localhost ~]$ sudo dnf update
MySQL 8.4 LTS Community
Server
377 kB/s | 226 kB     00:00
MySQL Connectors
Community
110 kB/s |  53 kB     00:00
MySQL Tools 8.4 LTS
Community
170 kB/s |  97 kB     00:00
Dependencies resolved.
================================================================================
================================================================
Package                                 Architecture
Version                               Repository
                      Size
================================================================================
================================================================Installing:
 mysql-community-client                   x86_64           8.
4.0-1.el9                           mysql-8.4-lts-community           3.1 M
```

```
     replacing  mysql.x86_64 8.0.36-1.el9_3
 mysql-community-server                   x86_64           8.
4.0-1.el9                          mysql-8.4-lts-community            50 M
     replacing  mariadb-connector-c-config.noarch 3.2.6-1.el9_0
     replacing  mysql-server.x86_64 8.0.36-1.el9_3
Installing dependencies:
  ...

Transaction Summary
================================================================================
=============================================================Install  7
Packages

Total download size: 59 M
Is this ok [y/N]: y
Downloading Packages:
(1/7): mysql-community-client-
plugins-8.4.0-1.el9.x86_64.rpm
3.4 MB/s | 1.4 MB     00:00
(2/7): mysql-community-
common-8.4.0-1.el9.x86_64.rpm
1.3 MB/s | 576 kB     00:00
(3/7): mysql-community-icu-data-
files-8.4.0-1.el9.x86_64.rpm
30 MB/s | 2.3 MB     00:00
(4/7): mysql-community-
client-8.4.0-1.el9.x86_64.rpm
5.8 MB/s | 3.1 MB     00:00
(5/7): mysql-community-
libs-8.4.0-1.el9.x86_64.rpm
6.8 MB/s | 1.5 MB     00:00
(6/7): net-
tools-2.0-0.62.20160912git.el9.x86_64.rpm
1.1 MB/s | 292 kB     00:00
(7/7): mysql-community-
server-8.4.0-1.el9.x86_64.rpm
48 MB/s |  50 MB     00:01
--------------------------------------------------------------------------------
-------------------------------------------------------------
Total
30
MB/s |  59 MB     00:01
MySQL 8.4 LTS Community
Server
3.0 MB/s | 3.1 kB     00:00
Importing GPG key 0xA8D3785C:
 Userid     : "MySQL Release Engineering <mysql-build@oss.oracle.com>"
 Fingerprint: BCA4 3417 C3B4 85DD 128E C6D4 B7B3 B788 A8D3 785C
 From       : /etc/pki/rpm-gpg/RPM-GPG-KEY-mysql-2023
```

```
Is this ok [y/N]: y
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :
  ...
Installed:
  mysql-community-server-8.4.0-1.el9.x86_64
  ...

Complete!
```

Do not forget to re-enable and restart your server:

```
sudo systemctl enable mysqld.service --now
```

9.1.2 Check your Knowledge of MySQL

✔ Which MySQL database version is installed by default?

MySQL 5.5

MariaDB 10.5

MariaDB 11.11

Mysql 8

10. Part 4.3 MariaDB database replication

10.1 Secondary server with MariaDB

This chapter will teach you how to configure Primary/Secondary system servers with MariaDB.

Objectives: You will learn how to:

✔ activate the binlogs in your servers;
✔ set up a secondary server to replicate data from the primary server.

🏁 **MariaDB**, **Replication**, **Primary**, **Secondary**

Knowledge: ★★
Complexity: ★★★

Reading time: 10 minutes

10.1.1 Generalities secondary server with MariaDB

As soon as you start using your database more intensively, you must replicate your data on several servers.

This can be done in several ways:

- Distribute write requests to the primary server and read requests to the secondary server.
- Perform database backups on the secondary server, which avoids blocking writes to the primary server for the duration of the backups.

If your usage becomes even more demanding, you may consider switching to a primary/primary system: replications are then made crosswise, but beware of the risk of blocking the uniqueness of primary keys. Otherwise, you will need to switch to a more advanced clustering system.

10.1.2 Configuration of secondary server with MariaDB

How to activate the binlogs

Perform this action on the primary and secondary servers:

Add the following options to your `/etc/my.cnf.d/mariadb-server.cnf` file, under the `[mariadb]` key:

```
[mariadb]
log-bin
server_id=1
log-basename=server1
binlog-format=mixed
```

for the primary server and for the secondary server:

```
[mariadb]
log-bin
server_id=2
log-basename=server2
binlog-format=mixed
```

The `server_id` option must be unique on each server in the cluster, while the `log-basename` option allows you to specify a prefix to the binlog files. If you do not do this, you cannot rename your server.

You can now restart the MariaDB service on both servers:

```
sudo systemctl restart mariadb
```

You can check that binlogs files are well created:

```
$ ll /var/lib/mysql/
total 123332
...
-rw-rw----. 1 mysql mysql         0 Jun 21 11:07 multi-master.info
drwx------. 2 mysql mysql      4096 Jun 21 11:07 mysql
srwxrwxrwx. 1 mysql mysql         0 Jun 21 11:16 mysql.sock
-rw-rw----. 1 mysql mysql       330 Jun 21 11:16 server1-bin.000001
```

```
-rw-rw----. 1 mysql mysql        21 Jun 21 11:16 server1-bin.index
...
```

How to configure the replication

First of all, on the primary, you will need to create users authorized to replicate data (be careful to restrict the IPs authorized):

```
$ sudo mariadb

MariaDB [(none)]> CREATE USER 'replication'@'%' IDENTIFIED BY 'PASSWORD';
Query OK, 0 rows affected (0.002 sec)

MariaDB [(none)]> GRANT REPLICATION SLAVE ON *.* TO 'replication'@'%';
Query OK, 0 rows affected (0.002 sec)
```

or better for security (change '192.168.1.101' with your own secondary IP):

```
$ sudo mariadb

MariaDB [(none)]> CREATE USER 'replication'@'192.168.1.101' IDENTIFIED BY
'PASSWORD';
Query OK, 0 rows affected (0.002 sec)

MariaDB [(none)]> GRANT REPLICATION SLAVE ON *.* TO
'replication'@'192.168.1.101';
Query OK, 0 rows affected (0.002 sec)
```

You must lock in new transactions if your primary server already contains data. In contrast, the exporting or importing of data occurs to the secondary server(s) and tells the secondary servers when to start replication. If your server does not yet contain any data, the procedure is greatly simplified.

Prevent any changes to the data while you view the binary log position:

```
$ sudo mariadb

MariaDB [(none)]> FLUSH TABLES WITH READ LOCK;
Query OK, 0 rows affected (0.021 sec)

MariaDB [(none)]> SHOW MASTER STATUS;
+--------------------+----------+--------------+------------------+
| File               | Position | Binlog_Do_DB | Binlog_Ignore_DB |
```

```
+--------------------+----------+--------------+------------------+
| server1-bin.000001 |     1009 |              |                  |
+--------------------+----------+--------------+------------------+
1 row in set (0.000 sec)
```

Do not quit your session to keep the lock.

Record the File and Position details.

If your server contains data, it is time to create a backup and import it onto your secondary server(s). Keep the lock for the duration of the backup and release it as soon as the backup is complete. This reduces downtime (the time it takes to copy and import the data on the secondary servers).

You can remove the lock now:

```
$ sudo mariadb

MariaDB [(none)]> UNLOCK TABLES;
Query OK, 0 rows affected (0.000 sec)
```

On the secondary server, you can now set up the primary server to replicate with the following:

```
MariaDB [(none)]> CHANGE MASTER TO
  MASTER_HOST='192.168.1.100',
  MASTER_USER='replication',
  MASTER_PASSWORD='PASSWORD',
  MASTER_PORT=3306,
  MASTER_LOG_FILE='server1-bin.000001',
  MASTER_LOG_POS=1009,
  MASTER_CONNECT_RETRY=10;
Query OK, 0 rows affected, 1 warning (0.021 sec)

MariaDB [(none)]> START SLAVE;
Query OK, 0 rows affected (0.001 sec)
```

Replace the primary server IP with yours and the `MASTER_LOG_FILE` and `MASTER_LOG_POS` values with those you previously registered.

Check if the replication is ok:

```
MariaDB [(none)]> SHOW SLAVE STATUS \G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 192.168.1.100
                  Master_User: replication
              Master_Log_File: server1-bin.000001
          Read_Master_Log_Pos: 1009
...
        Seconds_Behind_Master: 0
      Slave_SQL_Running_State: Slave has read all relay log; waiting for more
updates
...
1 row in set (0.001 sec)
```

The `Seconds_Behind_Master` is an interesting value to monitor as it can help you see if there is a replication issue.

10.1.3 Workshop secondary server using MariaDB

For this workshop, you will need two servers with MariaDB services installed, configured, and secured, as described in the previous chapters.

You will configure replication on the secondary server, create a new database, insert data into it, and check that it is accessible on the secondary server.

Our two servers have the following IP addresses:

- server1: 192.168.1.100
- server2: 192.168.1.101

Remember to replace these values with your own.

Task 1: Create a dedicated replication user

On the primary server:

```
$ sudo mariadb

MariaDB [(none)]> CREATE USER 'replication'@'192.168.1.101' IDENTIFIED BY
'PASSWORD';
Query OK, 0 rows affected (0.002 sec)
```

```
MariaDB [(none)]> GRANT REPLICATION SLAVE ON *.* TO
'replication'@'192.168.1.101';
Query OK, 0 rows affected (0.002 sec)
```

Task 2: Record the primary server values

```
$ sudo mariadb

MariaDB [(none)]> FLUSH TABLES WITH READ LOCK;
Query OK, 0 rows affected (0.021 sec)

MariaDB [(none)]> SHOW MASTER STATUS;
+--------------------+----------+--------------+------------------+
| File               | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+--------------------+----------+--------------+------------------+
| server1-bin.000001 |     1009 |              |                  |
+--------------------+----------+--------------+------------------+
1 row in set (0.000 sec)

MariaDB [(none)]> UNLOCK TABLES;
Query OK, 0 rows affected (0.000 sec)
```

Task 3: Activate the replication

On the secondary server:

```
MariaDB [(none)]> CHANGE MASTER TO
  MASTER_HOST='192.168.1.100',
  MASTER_USER='replication',
  MASTER_PASSWORD='PASSWORD',
  MASTER_PORT=3306,
  MASTER_LOG_FILE='server1-bin.000001',
  MASTER_LOG_POS=1009,
  MASTER_CONNECT_RETRY=10;
Query OK, 0 rows affected, 1 warning (0.021 sec)

MariaDB [(none)]> START SLAVE;
Query OK, 0 rows affected (0.001 sec)
```

Check if the replication is ok:

```
MariaDB [(none)]> SHOW SLAVE STATUS \G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
```

```
                  Master_Host: 192.168.1.100
                  Master_User: replication
              Master_Log_File: server1-bin.000001
          Read_Master_Log_Pos: 1009
...
         Seconds_Behind_Master: 0
       Slave_SQL_Running_State: Slave has read all relay log; waiting for more
updates
...
1 row in set (0.001 sec)
```

Task 4: Create a new database and a user

On the primary server:

```
MariaDB [(none)]> create database NEW_DATABASE_NAME;
Query OK, 1 row affected (0.002 sec)

MariaDB [(none)]> grant all privileges on NEW_DATABASE_NAME.* TO
'NEW_USER_NAME'@'localhost' identified by 'PASSWORD';
Query OK, 0 rows affected (0.004 sec)
```

On the secondary server, check for the creation of the database:

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| NEW_DATABASE_NAME  |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

On the secondary server, try connecting the new user created on the primary:

```
$ mariadb -u NEW_USER_NAME -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
```

```
+--------------------+
| NEW_DATABASE_NAME  |
| information_schema |
+--------------------+
2 rows in set (0.000 sec)
```

Task 5: Insert new data

Insert new data on the primary server:

```
MariaDB [(none)]> use NEW_DATABASE_NAME
Database changed

MariaDB [(none)]>  CREATE TABLE users(
    ->     id INT NOT NULL AUTO_INCREMENT,
    ->     first_name VARCHAR(30) NOT NULL,
    ->     last_name VARCHAR(30) NOT NULL,
    ->     age INT DEFAULT NULL,
    ->     PRIMARY KEY (id));

MariaDB [NEW_DATABASE_NAME]> INSERT INTO users (first_name, last_name, age)
VALUES ("Antoine", "Le Morvan", 44);
Query OK, 1 row affected (0.004 sec)
```

On the secondary server, check that data are replicated:

```
MariaDB [(none)]> use NEW_DATABASE_NAME
Database changed

MariaDB [NEW_DATABASE_NAME]> show tables;
+----------------------------+
| Tables_in_NEW_DATABASE_NAME |
+----------------------------+
| users                      |
+----------------------------+
1 row in set (0.000 sec)

MariaDB [NEW_DATABASE_NAME]> SELECT * FROM users;
+----+------------+-----------+------+
| id | first_name | last_name | age  |
+----+------------+-----------+------+
|  1 | Antoine    | Le Morvan |   44 |
+----+------------+-----------+------+
1 row in set (0.000 sec)
```

10.1.4 Check your Knowledge of the secondary server with MariaDB

✓ Each server must have the same ID within a cluster.

True

False

✓ Binary logs must be enabled before replication is activated.

True

False

It depends

10.1.5 Conclusion about the secondary server with MariaDB

As you can see, creating one or more secondary servers is a relatively easy action, but it does require service interruption on the main server.

However, it offers many advantages: high data availability, load balancing, and simplified backup.

In a central server crash, one of the secondary servers can be promoted to the central server.

11. Part 5. Load balancing, caching and proxyfication

In this part, we will discuss existing solutions for improving traffic performance and accepting more and more client connections.

12. Part 5.1 HAProxy

Info

This content is not written yet.

13. Part 5.2 Varnish

13.1 Varnish

This chapter will teach you about the web accelerator proxy cache: Varnish.

Objectives: You will learn how to:

✔ Install and configure Varnish;
✔ Cache the content of a website.

🏁 **reverse-proxy**, **cache**

Knowledge: ⭐⭐
Complexity: ⭐⭐⭐

Reading time: 30 minutes

13.1.1 Generalities

Varnish is an HTTP reverse-proxy-cache service or a website accelerator.

Varnish receives HTTP requests from visitors:

- if the response to the cached request is available, it returns the response directly to the client from the server's memory,
- if it does not have the response, Varnish addresses the web server. Varnish then sends the request to the web server, retrieves the response, stores it in its cache, and responds to the client.

Responding from the in-memory cache improves response times for clients. In this case, there is no access to physical disks.

By default, Varnish listens on port **6081** and uses **VCL** (**V**arnish **C**onfiguration **L**anguage) for its configuration. Thanks to VCL, it is possible to:

- Decide the content the client receives by way of transmission
- What the cached content is
- From what site and how do modifications of the response occur?

Varnish is extensible with VMOD modules (Varnish Modules).

Ensuring high availability

The use of several mechanisms ensures high availability throughout a web chain:

- If Varnish is behind load balancers(LBs), they are already in HA mode, as the LBs are generally in cluster mode. A check from the LBs verifies varnish availability. If a varnish server no longer responds, it is automatically removed from the pool of available servers. In this case, the Varnish is in ACTIVE/ACTIVE mode.
- if varnish is not behind an LB cluster, clients address a VIP (see Heartbeat chapter) shared between the 2 varnishes. In this case, varnish is in ACTIVE/PASSIVE mode. The VIP switches to the second varnish node if the active server is unavailable.
- When a backend is no longer available, you can remove it from the varnish backend pool, either automatically (with a health check) or manually in CLI mode (useful for easing upgrades or updates).

Ensuring scalability

If the backends are no longer sufficient to support the workload:

- either add more resources to the backends and reconfigure the middleware
- or add another backend to the varnish backend pool

Facilitating scalability

A web page is often composed of HTML (often dynamically generated by PHP) and more static resources (JPG, gif, CSS, js, and so on) during creation. It quickly

becomes interesting to cache the cacheable resources (the static ones), which offloads many requests from the backends.

> **Note**
>
> Caching web pages (HTML, PHP, ASP, JSP, etc.) is possible but more complicated. You need to know the application and whether the pages are cacheable, which should be true with a REST API.

When a client accesses a web server directly, the server must return the same image as often as the client requests. Once the client has received the image for the first time, it is cached on the browser side, depending on the configuration of the site and the web application.

When accessing the server behind a properly configured cache server, the first client requesting the image will initiate an initial backend request. However, caching of the image will occur for a certain period of time, and subsequent delivery will be directed to other clients requesting the same resource.

Although a well-configured browser-side cache reduces the number of requests to the backend, it complements the use of a varnish proxy cache.

TLS certificate management

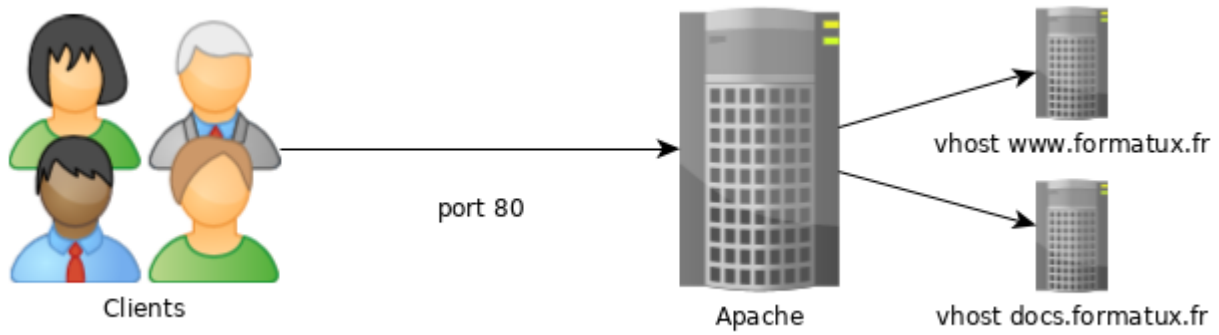
Varnish cannot communicate in HTTPS (and it is not its role to do so).

The certificate must, therefore, be either:

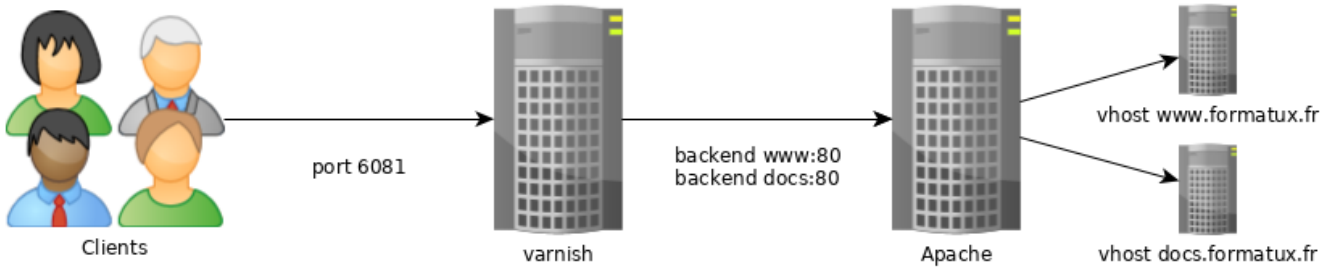
- carried by the LB when the flow passes through it (the recommended solution is to centralize the certificate, etc.). The flow then passes unencrypted through the data center.
- carried by an Apache, Nginx, or HAProxy service on the varnish server itself, which only acts as a proxy to the varnish (from port 443 to port 80). This solution is useful if accessing varnish directly.
- Similarly, Varnish cannot communicate with backends on port 443. When necessary, you need to use an Nginx or Apache reverse proxy to decrypt the request for varnish.

How it works

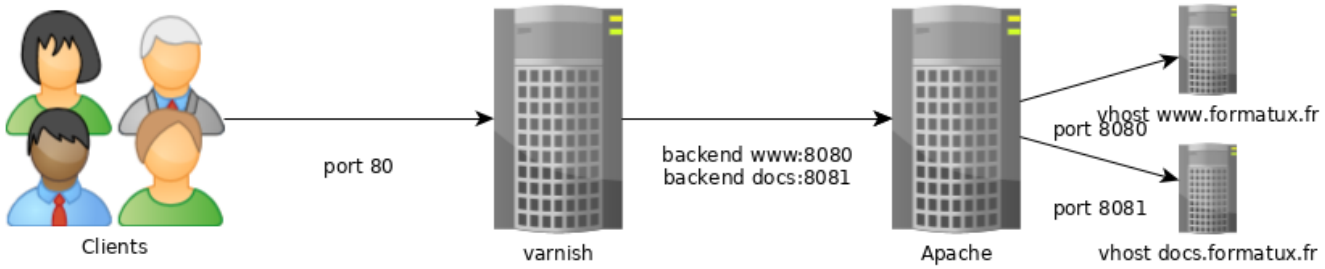
In a basic Web service, the client communicates directly with the service with TCP on port 80.



To use the cache, the client must communicate with the web service on the default Varnish port 6081.



To make the service transparent to the client, you must change the default listening port for Varnish and the web service vhosts.



To provide an HTTPS service, add either a load balancer upstream of the varnish service or a proxy service on the varnish server, such as Apache, Nginx, or HAProxy.

13.1.2 Configuration

Installation is simple:

```
dnf install -y varnish
systemctl enable varnish
systemctl start varnish
```

Configuring the varnish daemon

Since `systemctl`, varnish parameters are setup thanks to the service file `/usr/lib/systemd/system/varnish.service`:

```
[Unit]
Description=Varnish Cache, a high-performance HTTP accelerator
After=network-online.target

[Service]
Type=forking
KillMode=process

# Maximum number of open files (for ulimit -n)
LimitNOFILE=131072

# Locked shared memory - should suffice to lock the shared memory log
# (varnishd -l argument)
# Default log size is 80MB vsl + 1M vsm + header -> 82MB
# unit is bytes
LimitMEMLOCK=85983232

# Enable this to avoid "fork failed" on reload.
TasksMax=infinity

# Maximum size of the corefile.
LimitCORE=infinity

ExecStart=/usr/sbin/varnishd -a :6081 -f /etc/varnish/default.vcl -s malloc,
256m
ExecReload=/usr/sbin/varnishreload

[Install]
WantedBy=multi-user.target
```

Change the default values thanks to `systemctl edit varnish.service`: this will create the `/etc/systemd/system/varnish.service.d/override.conf` file:

```
$ sudo systemctl edit varnish.service
[Service]
ExecStart=/usr/sbin/varnishd -a :6081 -f /etc/varnish/default.vcl -s malloc,
512m
```

You can select the option several times to specify a cache storage backend. Possible storage types are `malloc` (cache in memory, then swap if needed), or `file` (create a file on disk, then map to memory). Sizes are expressed in K/M/G/T (kilobytes, megabytes, gigabytes, or terabytes).

Configuring the backends

Varnish uses a specific language called VCL for its configuration.

This involves compiling the VCL configuration file in C. If compilation is successful with no alarms, the service can be restarted.

You can test the varnish configuration with the following command:

```
varnishd -C -f /etc/varnish/default.vcl
```

Note

It is advisable to check the VCL syntax before restarting the `varnishd` daemon.

Reload the configuration with the command:

```
systemctl reload varnishd
```

Warning

A `systemctl restart varnishd` empties the varnish cache and causes a peak load on the backends. You should, therefore, avoid reloading `varnishd`.

> **Note**
>
> To configure Varnish, please follow the recommendations on this page: https://www.getpagespeed.com/server-setup/varnish/varnish-virtual-hosts.

13.1.3 VCL language

Subroutines

Varnish uses VCL files, segmented into subroutines containing the actions to run. These subroutines run only in the specific cases they define. The default `/etc/varnish/default.vcl` file contains the `vcl_recv`, `vcl_backend_response` and `vcl_deliver` routines:

```
#
# This is an example VCL file for Varnish.
#
# It does not do anything by default, delegating control to the
# builtin VCL. The builtin VCL is called when there is no explicit
# return statement.
#
# See the VCL chapters in the Users Guide at https://www.varnish-cache.org/
docs/
# and http://varnish-cache.org/trac/wiki/VCLExamples for more examples.

# Marker to tell the VCL compiler that this VCL has been adapted to the
# new 4.0 format.
vcl 4.0;

# Default backend definition. Set this to point to your content server.
backend default {
    .host = "127.0.0.1";
    .port = "8080";
}

sub vcl_recv {

}

sub vcl_backend_response {

}

sub vcl_deliver {
```

```
}
```

- **vcl_recv**: routine called before sending the request to the backend. In this routine, you can modify HTTP headers and cookies, choose the backend, etc. See actions `set req`.
- **vcl_backend_response**: routine called after reception of the backend response (`beresp` means BackEnd RESPonse). See `set bereq.` and `set beresp.` actions.
- **vcl_deliver**: This routine is useful for modifying Varnish output. If you need to modify the final object (e.g., add or remove a header), you can do so in `vcl_deliver`.

VCL operators

- `=`: assignment
- `==`: comparison
- `~`: comparison in combination with a regular expression and ACLs
- `!`: negation
- `&&`: and logic
- `||`: or logical

Varnish objects

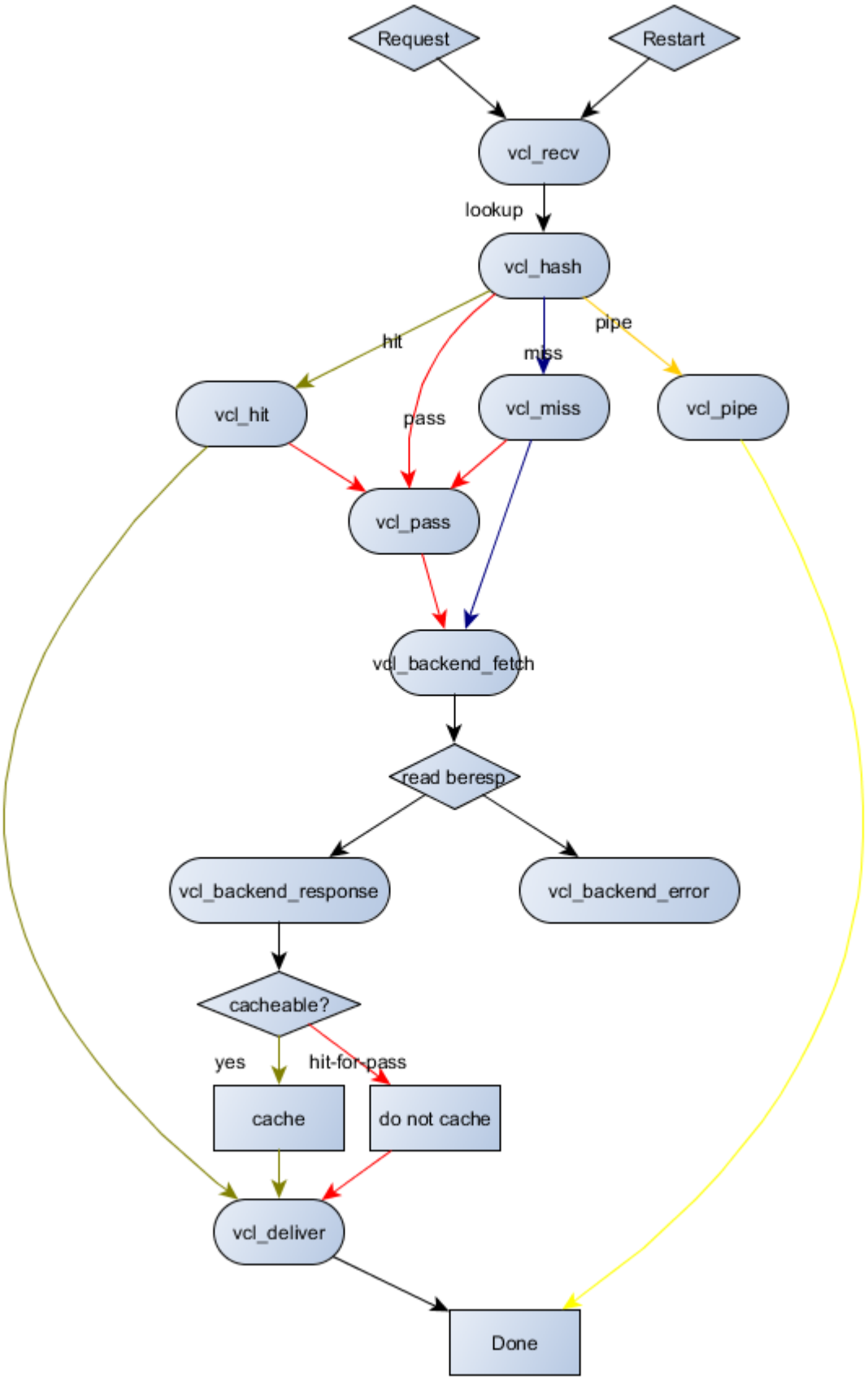
- **req**: the request object. Creates the `req` when Varnish receives the request. Most of the work in the `vcl_recv` subroutine concerns this object.
- **bereq**: the request object destined for the web server. Varnish creates this object from `req`.
- **beresp**: the web server response object. It contains the object headers from the application. You can modify the server response in the `vcl_backend_response` subroutine.
- **resp**: the HTTP response sent to the client. Modify this object with the `vcl_deliver` subroutine.
- **obj**: the cached object. Read-only.

Varnish actions

The most frequent actions:

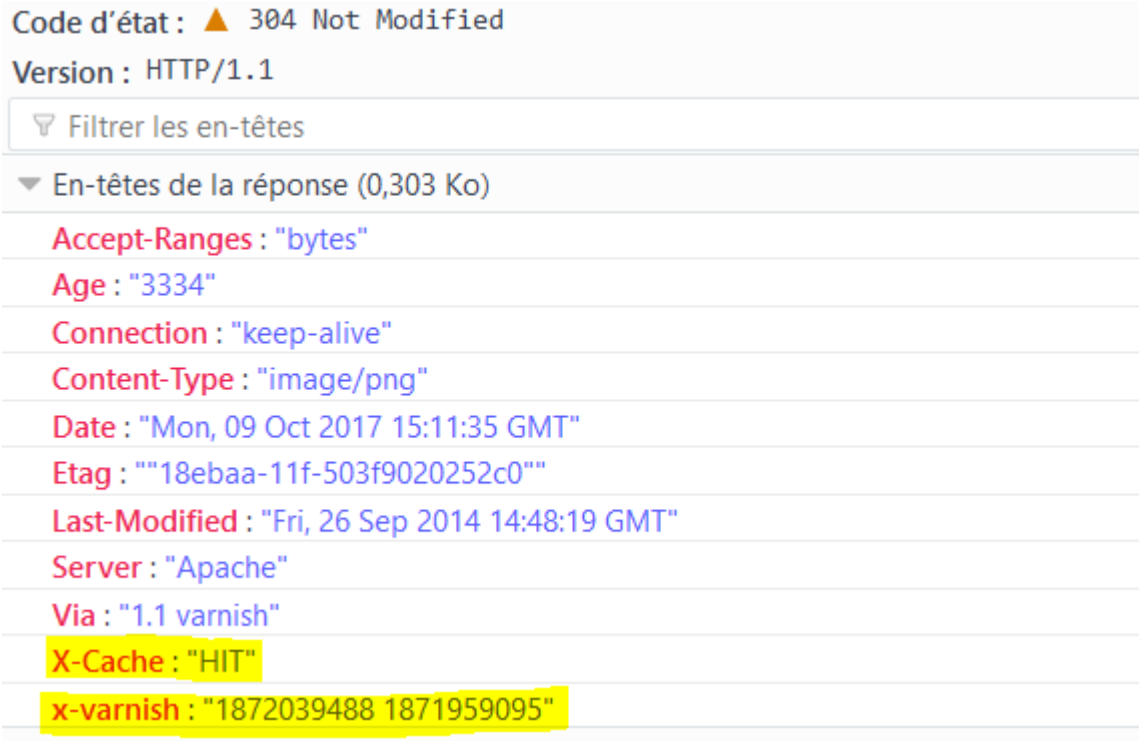
- **pass**: When returned, the request and subsequent response will come from the application server. No application of cache occurs. `pass` returns from the `vcl_recv` subroutine.
- **hash**: When returned from `vcl_recv`, Varnish will serve the content from the cache even if the request's configuration specifies passing without a cache.
- **pipe**: Used to manage flows. In this case, Varnish will no longer inspect each request but let all bytes pass to the server. Websockets or video stream management, for example, use `pipe`.
- **deliver**: Delivers the object to the client. Usually from the `vcl_backend_response` subroutine.
- **restart**: Restarts the request processing process. Retains modifications to the `req` object.
- **retry**: Transfers the request back to the application server. Used from `vcl_backend_response` or `vcl_backend_error` if the application response is unsatisfactory.

In summary, illustrated in the diagram below are the possible interactions between subroutines and actions:



13.1.4 Verification/Testing/Troubleshooting

It is possible to verify that a page comes from the varnish cache from the HTTP response headers:



13.1.5 Backends

Varnish uses the term `backend` for the vhosts it needs to proxy.

You can define several backends on the same Varnish server.

Configuring backends is through `/etc/varnish/default.vcl`.

ACL management

```
# Deny ACL
acl deny {
"10.10.0.10"/32;
"192.168.1.0"/24;
}
```

Apply ACL:

```
# Block ACL deny IPs
if (client.ip ~ forbidden) {
  error 403 "Access forbidden";
}
```

Do not cache certain pages:

```
# Do not cache login and admin pages
if (req.url ~ "/(login|admin)") {
  return (pass);
}
```

POST and cookies settings

Varnish never caches HTTP POST requests or requests containing cookies (whether from the client or the backend).

If the backend uses cookies, content caching will not occur.

To correct this behavior, you can unset the cookies in your requests:

```
sub vcl_recv {
    unset req.http.cookie;
}

sub vcl_backend_response {
    unset beresp.http.set-cookie;
}
```

Distribute requests to different backends

When hosting several sites, such as a document server () and a wiki (), it is possible to distribute requests to the right backend.

Backends declaration:

```
backend docs {
    .host = "127.0.0.1";
    .port = "8080";
}

backend blog {
```

```
    .host = "127.0.0.1";
    .port = "8081";
}
```

Modification of `req.backend` object occurs according to the host called in the HTTP request in the `vcl_recv` subroutine:

```
sub vcl_recv {
    if (req.http.host ~ "^doc.rockylinux.org$") {
        set req.backend = docs;
    }

    if (req.http.host ~ "^wiki.rockylinux.org$") {
        set req.backend = wiki;
    }
}
```

Load distribution

Varnish can handle load balancing with specific backends called directors.

The round-robin director distributes requests to the round-robin backends (alternately). You can assign a weight to each backend.

The client director distributes requests according to a sticky session affinity on any header element (that is, with a session cookie). In this case, a client is always returned to the same backend.

Backends declaration

```
backend docs1 {
    .host = "192.168.1.10";
    .port = "8080";
}

backend docs2 {
    .host = "192.168.1.11";
    .port = "8080";
}
```

The `director` allows you to associate the 2 defined backends.

Director declaration:

```
director docs_director round-robin {
    { .backend = docs1; }
    { .backend = docs2; }
}
```

All that remains is to define the director as a backend to the requests:

```
sub vcl_recv {
    set req.backend = docs_director;
}
```

Managing backends with CLI

Marking backends as **sick** or **healthy** is possible for administration or maintenance purposes. This action allows you to remove a node from the pool without modifying the Varnish server configuration (without restarting it) or stopping the backend service.

View backend status: The `backend.list` command displays all backends, even those without a health check (probe).

```
$ varnishadm backend.list
Backend name                   Admin      Probe
site.default                   probe      Healthy (no probe)
site.front01                   probe      Healthy 5/5
site.front02                   probe      Healthy 5/5
```

To switch from one state to another:

```
varnishadm backend.set_health site.front01 sick

varnishadm backend.list
Backend name                   Admin      Probe
site.default                   probe      Healthy (no probe)
site.front01                   sick       Sick 0/5
site.front02                   probe      Healthy 5/5

varnishadm backend.set_health site.front01 healthy

varnishadm backend.list
```

```
Backend name                   Admin      Probe
site.default                   probe      Healthy (no probe)
site.front01                   probe      Healthy 5/5
site.front02                   probe      Healthy 5/5
```

To let Varnish decide on the state of its backends, it is imperative to manually switch backends to sick or healthy backends and back to auto mode.

```
varnishadm backend.set_health site.front01 auto
```

Declaring the backends is done by following: https://github.com/mattiasgeniar/varnish-6.0-configuration-templates.

13.1.6 Apache logs

As the HTTP service is reverse proxied, the web server will no longer have access to the client's IP address but to the Varnish service.

To take reverse proxy into account in Apache logs, change the format of the event log in the server configuration file:

```
LogFormat "%{X-Forwarded-For}i %l %u %t "%r" %>s %b "%{Referer}i" "%{User-
Agent}i"" varnishcombined
```

and take this new format into account in the website vhost:

```
CustomLog /var/log/httpd/www-access.log.formatux.fr varnishcombined
```

and make it Varnish compatible:

```
if (req.restarts == 0) {
  if (req.http.x-forwarded-for) {
    set req.http.X-Forwarded-For = req.http.X-Forwarded-For + ", " + client.ip;
  } else {
   set req.http.X-Forwarded-For = client.ip;
  }
}
```

13.1.7 Cache purge

A few requests to purge the cache:

on the command line:

```
varnishadm 'ban req.url ~ .'
```

using a secret and a port other than the default:

```
varnishadm -S /etc/varnish/secret -T 127.0.0.1:6082 'ban req.url ~ .'
```

on the CLI:

```
varnishadm

varnish> ban req.url ~ ".css$"
200

varnish> ban req.http.host == example.com
200

varnish> ban req.http.host ~ .
200
```

via an HTTP PURGE request:

```
curl -X PURGE http://example.com/foo.txt
```

Configuring Varnish to accept this request is done with:

```
acl local {
    "localhost";
    "10.10.1.50";
}

sub vcl_recv {
    # directive to be placed first,
    # otherwise another directive may match first
    # and the purge will never be performed
    if (req.method == "PURGE") {
        if (client.ip ~ local) {
            return(purge);
```

```
        }
    }
}
```

13.1.8 Log management

Varnish writes its logs in memory and binary to not penalize its performance. When it runs out of memory space, it rewrites new records on top of old ones, starting from the beginning of its memory space.

It is possible to consult the logs with the `varnishstat` (statistics), `varnishtop` (top for Varnish), `varnishlog` (verbose logging), or `varnishnsca` (logs in NCSA format, like Apache) tools:

```
varnishstat
varnishtop -i ReqURL
varnishlog
varnishnsca
```

Using the `-q` option to apply filters to logs is done using:

```
varnishlog -q 'TxHeader eq MISS' -q "ReqHeader ~ '^Host: rockylinux\.org$'"
varnishncsa -q "ReqHeader eq 'X-Cache: MISS'"
```

`varnishlog` and `varnishnsca` daemons logs to disk independently of the `varnishd` daemon. The `varnishd` daemon continues to populate its logs in memory without penalizing performance towards clients; then, the other daemons copy the logs to disk.

13.1.9 Workshop

For this workshop, you will need one server with Apache services installed, configured, and secured, as described in the previous chapters.

You will configure a reverse proxy cache in front of it.

Your server has the following IP addresses:

- server1: 192.168.1.10

If you do not have a service to resolve names, fill the `/etc/hosts` file with content like the following:

```
$ cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6

192.168.1.10 server1 server1.rockylinux.lan
```

Task 1: Installation and configuration of Apache

```
sudo dnf install -y httpd mod_ssl
sudo systemctl enable httpd  --now
sudo firewall-cmd --permanent --add-service=http
sudo firewall-cmd --permanent --add-service=https
sudo firewall-cmd --reload
echo "<html><body>Node $(hostname -f)</body></html>" | sudo tee "/var/www/html/
index.html"
```

Verify:

```
$ curl http://server1.rockylinux.lan
<html><body>Node server1.rockylinux.lan</body></html>

$ curl -I http://server1.rockylinux.lan
HTTP/1.1 200 OK
Date: Mon, 12 Aug 2024 13:16:18 GMT
Server: Apache/2.4.57 (Rocky Linux) OpenSSL/3.0.7
Last-Modified: Mon, 12 Aug 2024 13:11:54 GMT
ETag: "36-61f7c3ca9f29c"
Accept-Ranges: bytes
Content-Length: 54
Content-Type: text/html; charset=UTF-8
```

Task 2: Install varnish

```
sudo dnf install -y varnish
sudo systemctl enable varnishd --now
sudo firewall-cmd --permanent --add-port=6081/tcp --permanent
sudo firewall-cmd --reload
```

Task 3: Configure Apache as a backend

Modify `/etc/varnish/default.vcl` to use apache (port 80) as backend:

```
# Default backend definition. Set this to point to your content server.
backend default {
    .host = "127.0.0.1";
    .port = "80";
}
```

Reload Varnish

```
sudo systemctl reload varnish
```

Check if varnish works:

```
$ curl -I http://server1.rockylinux.lan:6081
HTTP/1.1 200 OK
Server: Apache/2.4.57 (Rocky Linux) OpenSSL/3.0.7
X-Varnish: 32770 6
Age: 8
Via: 1.1 varnish (Varnish/6.6)

$ curl http://server1.rockylinux.lan:6081
<html><body>Node server1.rockylinux.lan</body></html>
```

As you can see, Apache serves the index page.

Some headers have been added, giving us information that our request was handled by varnish (header `Via`) and the cached time of the page (header `Age`), which tells us that our page was served directly from the varnish memory instead of from the disk with Apache.

Task 4: Remove some headers

We will remove some headers that can give unneeded information to hackers.

In the sub `vcl_deliver`, add the following:

```
sub vcl_deliver {
    unset resp.http.Server;
```

```
    unset resp.http.X-Varnish;
    unset resp.http.Via;
    set resp.http.node = "F01";
    set resp.http.X-Cache-Hits = obj.hits;
    if (obj.hits > 0) { # Add debug header to see if it is a HIT/MISS and the
number of hits, disable when not needed
      set resp.http.X-Cache = "HIT";
    } else {
      set resp.http.X-Cache = "MISS";
    }
}
```

Test your config and reload varnish:

```
$ sudo varnishd -C -f /etc/varnish/default.vcl
...
$ sudo systemctl reload varnish
```

Check the differences:

```
$ curl -I http://server1.rockylinux.lan:6081
HTTP/1.1 200 OK
Age: 4
node: F01
X-Cache-Hits: 1
X-Cache: HIT
Accept-Ranges: bytes
Connection: keep-alive
```

As you can see, removing the unwanted headers occurs while adding the necessary ones (to troubleshoot).

13.1.10 Conclusion

You now have all the knowledge you need to set up a primary cache server and add functionality.

Having a varnish server in your infrastructure can be very useful for many things besides caching: for backend server security, for handling headers, for facilitating updates (blue/green or canary mode, for example), etc.

13.1.11 Check your Knowledge

✓ Can Varnish host static files?

True

False

✓ Does the varnish cache have to be stored in memory?

True

False

14. Part 5.3 Squid

14.1 Squid

This chapter will teach you about Squid, the HTTP proxy cache.

Objectives: You will learn how to:

✔ install squid
✔ configure it to be a proxy and cache HTTP content.

🏁 **squid**, **proxy**, **HTTP**

Knowledge: ⭐⭐
Complexity: ⭐⭐

Reading time: 20 minutes

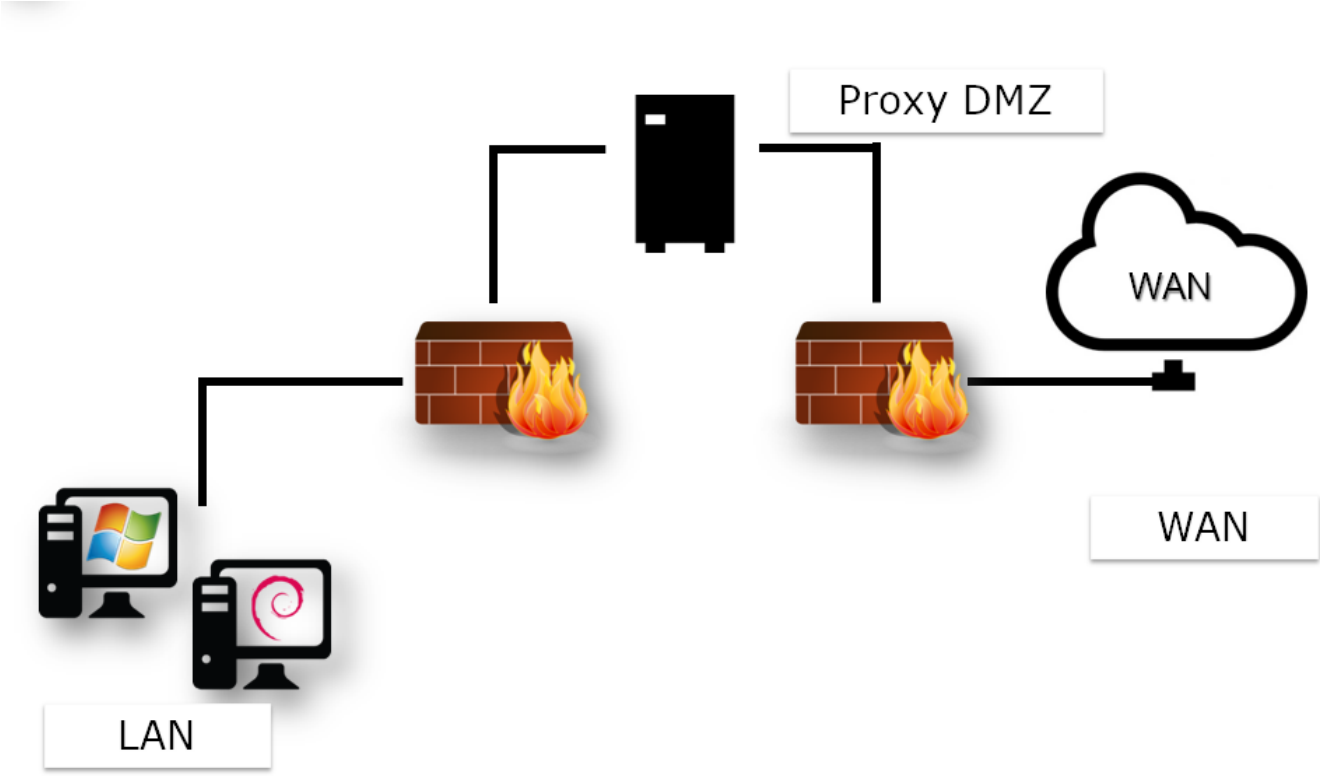
14.1.1 Generalities

Setting up a proxy server involves choosing between two types of architecture:

- A standard proxy architecture requiring specific configuration of each client and their web browsers
- Captive proxy architecture, which involves intercepting the frames sent by the client and rewriting them to the proxy server

In either case, a break in the network occurs: A client can no longer physically address a remote server directly without going through a proxy server.

Two firewalls protect the client workstation but never communicate directly with the outside network.

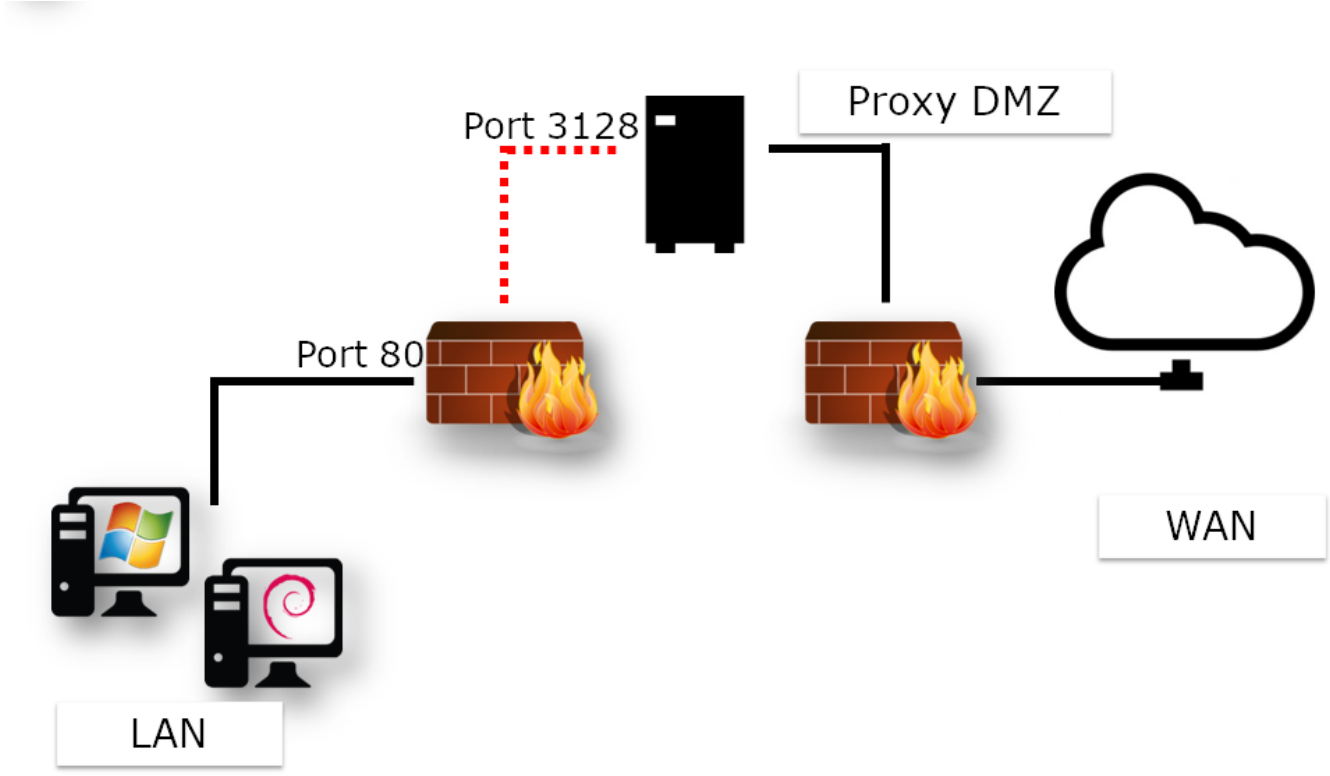


Note

This architecture requires browser configuration on the client workstation.

You don't need to configure all client workstations with a captive proxy.

The configuration occurs at the gateway level, where it receives client requests and transparently rewrites the frames to send them to the proxy.



Note

This architecture requires a specific configuration on the router.

In the case of standard proxy or captive proxy architecture, one of the primary interests of this type of service is to act as a cache.

In this way, a file downloaded once from the WAN (potentially from a slower link than the LAN) stores itself in memory in the proxy cache for subsequent clients to use. This optimizes bandwidth on the slow link.

As you will see later, there are other uses for a proxy.

Deploying a proxy can:

- Deny access to specific resources based on various parameters
- Set up authentication and monitoring of clients' Internet activities
- Set up a hierarchy of distributed caches
- Hide the LAN architecture from a WAN point of view (how many clients are there on the LAN?)

Among the advantages are the following:

- Anonymity on the Internet
- Authentication
- Client activity logging
- Filtering
- Limiting access
- Bandwidth optimization
- Security

Note

Implementing authentication blocks many of the malicious effects of viruses on the LAN.

Warning

The proxy service becomes a critical service requiring high availability.

When operating a Squid Proxy server, the administrator must exploit the logs. Therefore, it is essential to know the main HTTP response codes.

Code	Categories
1XX	Info
2XX	Success
3XX	Redirection
4XX	Customer request error
5XX	Server error

Examples:

- 200: ok
- 301: Moved Permanently
- 302: Moved Temporarily
- 304: Not modified
- 400: Bad request
- 401: Unauthorized
- 404: Not found

About Squid

Squid supports HTTP and FTP protocols.

The advantages of installing a solution based on the Squid server:

- Hardware solutions are expensive
- Developed since 1996
- Released under GNU/GPL license

SIZING

- Ensure high availability
- Use fast hard disks for caching
- RAM and CPU should be correctly sized

> **Note**
>
> Allow 14MB of RAM per GB of disk cache.

14.1.2 Installation

Install the `squid` package:

```
sudo dnf install squid
```

> **Warning**
>
> Take care not to start the service until the cache has been initialized!

Squid server tree and files

The single configuration file is `/etc/squid/squid.conf`.

Service logs (stop and restart) are in `/var/log/squid.cache.log`, while client requests are in `/var/log/squid/access.log`. By default, cache files are in `/var/spool/squid/`.

The `squid` command

The `squid` command controls the squid server.

Syntax of the command:

```
squid [-z|-s|-k parse|-k rotate]
```

Option	Description
`-z`	Initializes cache directories
`-s`	Enables syslog logging
`-k parse`	Test configuration file
`-k rotate`	Rotates logs

Logging client requests can quickly lead to storing large amounts of data.

It is a good idea to regularly create a new log file and archive the old one in compressed format.

You can do this either manually, with the `-k rotate` option of the `squid` command, or automatically with the dedicated Linux service `logrotate`.

14.1.3 Configuration

Configure Squid in `/etc/squid/squid.conf`.

- Proxy port number (listening port) `http_port`

```
http_port num_port
```

> **Note**
>
> The port number is set to 3128 by default but frequently changes to 8080. Remember to open the corresponding firewall port!

When the service restarts, the Squid server will listen on the port defined by the `http_port` directive.

- RAM reservation `cache_mem`

```
cache_mem taille KB|taille MB|taille GB
```

For example:

```
cache_mem 1 GB
```

> **Tip**
>
> Best practice: 1/3 of total RAM allocated

- Internet Cache Protocol (ICP) `icp_port`

The Internet Cache Protocol (ICP) enables neighboring Squid servers to exchange requests. It is common practice to propose a hierarchy of proxies that share their information bases.

The `icp_port` directive defines the port number Squid uses to send and receive ICP requests from neighboring Squid servers.

> **Tip**
>
> Set to 0 to deactivate.

- Anonymous FTP user `ftp_user`

The `ftp_user` directive associates an FTP user with anonymous FTP connections. The user must have a valid e-mail address.

```
ftp_user bob@rockylinux.lan
```

- Set up Access Control Lists

ACL syntax:

```
acl name type argument
http_access allow|deny aclname
```

Example:

```
acl LUNCHTIME time 12:00-14:00
http_access deny LUNCHTIME
```

A more extensive discussion of ACLs is in the "Advanced configuration" section.

- Maximum size of a cached object `maximum_object_size`

`maximum_object_size` directive syntax:

```
maximum_object_size size
```

Example:

```
maximum_object_size 32 MB
```

The object is not cached if the object size is greater than the `maximum_object_size` limit.

- Proxy server name `visible_hostname`

Syntax of `visible_hostname` directive:

```
visible_hostname name
```

Example:

```
visible_hostname proxysquid
```

> **Note**
>
> The value supplied may be different from the hostname.

- Define a cache for squid `cache_ufs`

```
cache_ufs format path size nbFolderNiv1 nbFolderNiv2
```

IDefining multiple caches on different file systems to optimize access times is possible.

Example:

```
cache_dir ufs /var/spool/squid/ 100 16 256
```

Option	Description
ufs	Unix File System
100	Size in mega
16	16 top-level folders
256	256 second-level folders

When the service launches for the first time, it generates the cache directory:

```
sudo squid -z
sudo systemctl start squid
```

14.1.4 Advanced configuration

Les Access Control List (ACL)

Syntax of the `http_access` directive

```
http_access allow|deny [!]acl_name
```

Example:

```
http_access allow LUNCHTIME
http_access deny !LUNCHTIME
```

The `!acl_name` ACL is the opposite of the `acl_name` ACL.

Syntax of the `acl` directive:

```
acl name type argument
```

The order of ACLs is cumulative. Several ACLs with the same name represent a single ACL.

Examples:

Lunchtime authorization:

```
acl LUNCHTIME time 12:00-14:00
http_access allow LUNCHTIME
```

Ban videos:

```
acl VIDEOS rep_mime_type video/mpeg
acl VIDEOS rep_mime_type video/avi
http_access deny VIDEOS
```

Managing IP addresses:

```
acl XXX src 192.168.0.0/255.255.255.0
acl XXX dst 10.10.10.1
```

FQDN management:

```
acl XXX srcdomain .rockylinux.org
acl XXX dstdomain .linux.org
```

Port management:

```
acl XXX port 80 21
```

Protocol management:

```
acl XXX proto HTTP FTP
```

Caching algorithms

Different cache algorithms exist with different characteristics:

- LRU - *Least Recently Used*: removes the oldest objects from the RAM
- LRU-THOLD: copies an object to the cache according to its size
- MRU: *Most Recently Used*: deletes the least requested data
- GDSF: *Greedy Dual Size Frequency*: deletes according to original size and access time with the smallest retained
- LFUDA: *Least Frequently Used With Dynamic Aging*: same as GDSF, but without the notion of size. Useful for caches with large files

Client authentication

Squid relies on external programs to manage authentication. It can be based on a simple flat file such as `htpasswd` or on LDAP, SMB, PAM, or other services.

Authentication can also be a legal necessity. Remember to get your users to sign a usage charter!

14.1.5 Tools

The `squidclient` command

Use the `squidclient` command to test a request to the squid server.

`squidclient` command syntax:

```
squidclient [-s] [-h target] [-p port] url
```

Example:

```
squidclient -s -h localhost -p 8080 http://localhost/
```

Option	Description
`-s`	Silent mode (displays nothing in the console)
`-h`	Defines target proxy
`-p`	Listening port (default 3128)
`-r`	Forces the server to reload the object

Analyze logs

You can monitor Squid's log records with the command:

```
tail -f /var/log/squid/access.log
```

Decomposition of a log line:

Option	Description
Date	Log timestamp
Response time	Response time for request
@client	Client IP address
Status code	HTTP response code
Size	Transfer size
Method	HTTP method (Put / Get / Post / etc.)
URL	Request URL
Peer Code	Inter-proxy response code
File type	Mime type of request target

14.1.6 Security

The firewall should be open for the listening port:

```
sudo firewall-cmd --add-port=3128/tcp --permanent
sudo firewall-cmd --reload
```

14.1.7 Workshop

In this workshop, you will install Squid on your server and use it to download updates.

Task 1: Install and configure Squid

Install Squid:

```
sudo dnf install squid
sudo systemctl enable squid
sudo firewall-cmd --add-port=3128/tcp --permanent
sudo firewall-cmd --reload
```

Remove the comment in this line of the `/etc/squid/squid.conf` file to create a cache directory on disk:

```
cache_dir ufs /var/spool/squid 100 16 512
```

Adjust the cache size as required.

Create the cache directories and start the service.

```
sudo squid -z
sudo systemctl start squid
```

Task 2: Use your proxy with curl

Open a new terminal on your proxy server to follow the proxy's access.

```
sudo tail -f /var/log/squid/access.log
```

On the second terminal, use `curl` to access a web page through the proxy:

```
$ curl -I --proxy "http://192.168.1.10:3128" https://docs.rockylinux.org
HTTP/1.1 200 Connection established

HTTP/2 200
content-type: text/html
...
```

As you can see, two HTTP connections exist. The first is with the proxy, and the second is from the proxy to the remote server.

You can see the trace on your second terminal:

```
1723793294.548     77 192.168.1.10 TCP_TUNNEL/200 3725 CONNECT
docs.rockylinux.org:443 - HIER_DIRECT/151.101.122.132 -
```

The content is not cached here as you request an `https` connection to the remote server.

Task 3: Configure DNS to use your proxy server

Edit the `/etc/dnf/dnf.conf` file to use the proxy squid:

```
[main]
gpgcheck=1
installonly_limit=3
clean_requirements_on_remove=True
best=True
skip_if_unavailable=False
proxy=http://192.168.1.10:3128
```

Clean your `dnf` cache and try an update:

```
sudo dnf clean all
sudo dnf update
```

Verify on your terminal that the `dnf` connection uses your proxy to download its update. Note that the "URL of repository" in the line that follows will be replaced with the actual mirror URL:

```
1723793986.725     20 192.168.1.10 TCP_MISS/200 5238 GET "URL of repository"/
9.4/extras/x86_64/os/repodata/7d78a729-8e9a-4066-96d4-ab8ed8f06ee8-
FILELISTS.xml.gz - HIER_DIRECT/193.106.119.144 application/x-gzip
...
1723794176.255      1 192.168.1.10 TCP_HIT/200 655447 GET "URL of repository"/
9.4/AppStream/x86_64/os/repodata/1af312c9-7139-43ed-8761-90ba3cd55461-
UPDATEINFO.xml.gz - HIER_NONE/- application/x-gzip
```

In this example, you can see one connection with a TCP_MISS (not present in the cache) and another with TCP_HIT (use the cache to answer the client).

14.1.8 Conclusion

You now know how to install Squid on your local network. This will enable you to centralize your outgoing connections to the Internet and secure your local network.

14.1.9 Check your Knowledge

✔ What is the port listened to by a squid server per default?

- 8080
- 1234
- 443
- 3128

✔ What Squid is?

- A reverse proxy cache
- A proxy cache

15. Part 6. Mail servers

16. Part 7. High availability

16.1 Clustering under Linux

High availability is a term often used in IT, in connection with system architecture or a service, to designate the fact that this architecture or service has a suitable rate of availability. ~ wikipedia

This availability is a performance measure expressed as a percentage obtained by the ratio **Operating time / Total desired operating time**.

Rates	Annual downtime
90%	876 hours
95%	438 hours
99%	87 hours 36 minutes
99,9%	8 hours 45 minutes 36 seconds
99,99%	52 minutes 33 seconds
99,999%	5 minutes 15 seconds
99,9999%	31,68 seconds

"High Availability" (**HA**) refers to all measures taken to guarantee a service's highest possible availability—that is, correct operation 24 hours a day.

16.1.1 Overview

A cluster is a "computer cluster", a group of two or more machines.

A cluster allows:

- distributed computing by using the computing power of all the nodes
- high availability: service continuity and automatic service failover in the event of a node failure

Types of services

- Active/passive services

Installing a cluster with two active/passive nodes using Pacemaker and DRBD is a low-cost solution for many situations requiring a high-availability system.

- N+1 services

  With multiple nodes, Pacemaker can reduce hardware costs by allowing several active/passive clusters to combine and share a backup node.

- N TO N services

  With shared storage, every node can potentially be used for fault tolerance. Pacemaker can also run multiple copies of services to spread the workload.

- Remote site services

  Pacemaker includes enhancements to simplify the creation of multisite clusters.

VIP

The VIP is a virtual IP address assigned to an Active/Passive cluster. Assign the VIP to an active cluster node. If a service failure occurs, the VIP is deactivated on the failed node, while activation occurs on the node taking over. This is known as failover.

Clients always address the cluster using VIP, making active server failovers transparent.

Split-brain

Split-brain is the leading risk a cluster may encounter. This condition occurs when several nodes in a cluster think their neighbor is inactive. The node then tries to start the redundant service, and several nodes provide the same service, which can lead to annoying side effects (duplicate VIPs on the network, competing data access, and so on).

Possible technical solutions to avoid this problem are:

- Separate public network traffic from cluster network traffic
- using network bonding

16.2 Pacemaker (PCS)

In this chapter, you will learn about Pacemaker, a clustering solution.

Objectives: You will learn how to:

✔ install and configure a Pacemaker cluster; ✔ administer a Pacemaker cluster.

🏁 **clustering**, **ha**, **high availability**, **pcs**, **pacemaker**

Knowledge: ★ ★ ★ **Complexity**: ★ ★

Reading time: 20 minutes

16.2.1 Generalities

Pacemaker is the software part of the cluster that manages its resources (VIPs, services, data). It is responsible for starting, stopping and, supervising cluster resources. It guarantees high node availability.

Pacemaker uses the message layer provided by **corosync** (default) or **Heartbeat**.

Pacemaker consists of **5 key components**:

- Cluster Information Base (**CIB**)
- Cluster Resource Management daemon (**CRMd**)
- Local Resource Management daemon (**LRMd**)
- Policy Engine (**PEngine** or **PE**)
- Fencing daemon (**STONITHd**)

The CIB represents the cluster configuration and the current state of all cluster resources. Its contents are automatically synchronized across the entire cluster and used by the PEngine to calculate how to achieve the ideal cluster state.

The list of instructions is then provided to the Designated Controller (DC). Pacemaker centralizes all cluster decisions by electing one of the CRMd instances as master.

The DC executes the PEngine's instructions in the required order, transmitting them to the local LRMd or the CRMd of the other nodes via Corosync or Heartbeat.

Sometimes, stopping nodes to protect shared data or enable recovery may be necessary. Pacemaker comes with STONITHd for this purpose.

Stonith

Stonith is a component of Pacemaker. It stands for Shoot-The-Other-Node-In-The-Head, a recommended practice for ensuring the isolation of the malfunctioning node as quickly as possible (shut down or at least disconnected from shared resources), thus avoiding data corruption.

An unresponsive node does not mean that it can no longer access data. The only way to ensure that a node is no longer accessing data before handing it over to another node is to use STONITH, which will shut down or restart the failed server.

STONITH also has a role if a clustered service fails to shut down. In this case, Pacemaker uses STONITH to force the entire node to stop.

Quorum management

The quorum represents the minimum number of nodes in operation to validate a decision, such as deciding which backup node should take over when one of the nodes is in error. By default, Pacemaker requires more than half the nodes to be online.

When communication problems split a cluster into several group nodes, quorum prevents resources from starting up on more nodes than expected. A cluster is quorate when more than half of all nodes known to be online are in its group (active_nodes_group > active_total_nodes / 2).

When a quorum is not reached, the default decision is to turn off all resources.

Case study:

- On a **two-node cluster**, since reaching quorum **is not possible**, a node failure must be ignored, or the entire cluster will be shut down.
- If a 5-node cluster is split into 2 groups of 3 and 2 nodes, the 3-node group will have a quorum and continue to manage resources.
- If a 6-node cluster is split into 2 groups of 3 nodes, no group will have a quorum. In this case, the pacemaker's default behavior is to stop all resources to avoid data corruption.

Cluster communication

A pacemaker uses either **Corosync** or **Heartbeat** (from the Linux-ha project) for node-to-node communication and cluster management.

COROSYNC

Corosync Cluster Engine is a messaging layer between cluster members that integrates additional functionalities to implement high availability within applications. The Corosync derives from the OpenAIS project.

Nodes communicate in Client/Server mode with the UDP protocol.

It can manage clusters of more than 16 Active/Passive or Active/Active modes.

HEARTBEAT

Heartbeat technology is more limited than Corosync. It is impossible to create a cluster of more than two nodes, and its management rules are less sophisticated than those of its competitor.

> **Note**
>
> The choice of pacemaker/corosync today seems more appropriate, as it is the default choice for RedHat, Debian and Ubuntu distributions.

Data management

THE DRDB NETWORK RAID

DRDB is a block-type device driver enabling RAID 1 (mirroring) implementation over the network.

DRDB can be useful when NAS or SAN technologies are unavailable, but data synchronization is needed.

16.2.2 Installation

To install Pacemaker, first enable the `highavailability` repository:

```
sudo dnf config-manager --set-enabled highavailability
```

Some information about the pacemaker package:

```
$ dnf info pacemaker
Rocky Linux 9 - High
Availability
289 kB/s | 250 kB     00:00
Available Packages
Name         : pacemaker
Version      : 2.1.7
Release      : 5.el9_4
Architecture : x86_64
Size         : 465 k
Source       : pacemaker-2.1.7-5.el9_4.src.rpm
Repository   : highavailability
Summary      : Scalable High-Availability cluster resource manager
URL          : https://www.clusterlabs.org/
License      : GPL-2.0-or-later AND LGPL-2.1-or-later
Description  : Pacemaker is an advanced, scalable High-Availability cluster
resource
             : manager.
             :
             : It supports more than 16 node clusters with significant
capabilities
             : for managing resources and dependencies.
             :
             : It will run scripts at initialization, when machines go up or
down,
```

```
             : when related resources fail and can be configured to
periodically check
             : resource health.
             :
             : Available rpmbuild rebuild options:
             :   --with(out) : cibsecrets hardening nls pre_release profiling
             :                 stonithd
```

Using the `repoquery` command, you can find out the dependencies of the pacemaker package:

```
$ repoquery --requires pacemaker
corosync >= 3.1.1
pacemaker-cli = 2.1.7-5.el9_4
resource-agents
systemd
...
```

The pacemaker installation will, therefore, automatically install corosync and a CLI interface for a pacemaker.

Some information about the corosync package:

```
$ dnf info corosync
Available Packages
Name         : corosync
Version      : 3.1.8
Release      : 1.el9
Architecture : x86_64
Size         : 262 k
Source       : corosync-3.1.8-1.el9.src.rpm
Repository   : highavailability
Summary      : The Corosync Cluster Engine and Application Programming
Interfaces
URL          : http://corosync.github.io/corosync/
License      : BSD
Description  : This package contains the Corosync Cluster Engine Executive,
several default
             : APIs and libraries, default configuration files, and an init
script.
```

Install now the required packets:

```
sudo dnf install pacemaker
```

Open your firewall if you have one:

```
sudo firewall-cmd --permanent --add-service=high-availability
sudo firewall-cmd --reload
```

Note

Do not start the services now, as they are not configured and will not work.

16.2.3 Cluster management

The `pcs` package provides cluster management tools. The `pcs` command is a command-line interface for managing the **Pacemaker high-availability stack**.

Cluster configuration could be done by hand, but the pcs package makes managing (creating, configuring, and troubleshooting) a cluster much easier!

Note

There are alternatives to pcs.

Install the package on all nodes and activate the daemon:

```
sudo dnf install pcs
sudo systemctl enable pcsd --now
```

The package installation created a `hacluster` user with an empty password. To perform tasks such as synchronizing corosync configuration files or rebooting remote nodes. Assigning a password to this user is necessary.

```
hacluster:x:189:189:cluster user:/var/lib/pacemaker:/sbin/nologin
```

On all nodes, assign an identical password to the hacluster user:

```
echo "pwdhacluster" | sudo passwd --stdin hacluster
```

Note

Please replace "pwdhacluster" with a more secure password.

From any node, it is possible to authenticate as a hacluster user on all nodes then use the `pcs` commands on them:

```
$ sudo pcs host auth server1 server2
Username: hacluster
Password:
server1: Authorized
server2: Authorized
```

From the node on which pcs authentication occurs, launch the cluster configuration:

```
$ sudo pcs cluster setup mycluster server1 server2
No addresses specified for host 'server1', using 'server1'
No addresses specified for host 'server2', using 'server2'
Destroying cluster on hosts: 'server1', 'server2'...
server2: Successfully destroyed cluster
server1: Successfully destroyed cluster
Requesting remove 'pcsd settings' from 'server1', 'server2'
server1: successful removal of the file 'pcsd settings'
server2: successful removal of the file 'pcsd settings'
Sending 'corosync authkey', 'pacemaker authkey' to 'server1', 'server2'
server1: successful distribution of the file 'corosync authkey'
server1: successful distribution of the file 'pacemaker authkey'
server2: successful distribution of the file 'corosync authkey'
server2: successful distribution of the file 'pacemaker authkey'
Sending 'corosync.conf' to 'server1', 'server2'
server1: successful distribution of the file 'corosync.conf'
server2: successful distribution of the file 'corosync.conf'
Cluster has been successfully set up.
```

Note

The pcs cluster setup command handles the quorum problem for two-node clusters. Such a cluster will, therefore, function correctly in the event of the failure of one of the two nodes. If you manually configure Corosync or use another cluster management shell, you must configure Corosync correctly.

You can now start the cluster:

```
$ sudo pcs cluster start --all
server1: Starting Cluster...
server2: Starting Cluster...
```

Enable the cluster service to start on boot:

```
sudo pcs cluster enable --all
```

Check the service status:

```
$ sudo pcs status
Cluster name: mycluster

WARNINGS:
No stonith devices and stonith-enabled is not false

Cluster Summary:
  * Stack: corosync (Pacemaker is running)
  * Current DC: server1 (version 2.1.7-5.el9_4-0f7f88312) - partition with
quorum
  * Last updated: Mon Jul  8 17:50:14 2024 on server1
  * Last change:  Mon Jul  8 17:50:00 2024 by hacluster via hacluster on
server1
  * 2 nodes configured
  * 0 resource instances configured

Node List:
  * Online: [ server1 server2 ]

Full List of Resources:
  * No resources

Daemon Status:
  corosync: active/disabled
  pacemaker: active/disabled
  pcsd: active/enabled
```

Adding resources

Before you can configure the resources, you will need to deal with the alert message:

```
WARNINGS:
No stonith devices and stonith-enabled is not false
```

In this state, Pacemaker will refuse to start your new resources.

You have two choices:

- disable `stonith`
- configure it

First, you will disable `stonith` until you learn how to configure it:

```
sudo pcs property set stonith-enabled=false
```

> **Warning**
>
> Be careful not to leave `stonith` disabled in a production environment!

VIP CONFIGURATION

The first resource you will create on your cluster is a VIP.

List the standard resources available with the `pcs resource standards` command:

```
$ pcs resource standards
lsb
ocf
service
systemd
```

This VIP corresponds to customers' IP addresses so they can access future cluster services. You must assign it to one of the nodes. Then, if a failure occurs, the cluster will switch this resource from one node to another to ensure continuity of service.

```
pcs resource create myclusterVIP ocf:heartbeat:IPaddr2 ip=192.168.1.12
cidr_netmask=24 op monitor interval=30s
```

The `ocf:heartbeat:IPaddr2` argument contains three fields that provide Pacemaker with the following:

- the standard (here `ocf`)
- the script namespace (here `heartbeat`)
- the resource script name

The result is the addition of a virtual IP address to the list of managed resources:

```
$ sudo pcs status
Cluster name: mycluster

...
Cluster name: mycluster
Cluster Summary:
  * Stack: corosync (Pacemaker is running)
  ...
  * 2 nodes configured
  * 1 resource instance configured

Full List of Resources:
  * myclusterVIP        (ocf:heartbeat:IPaddr2):         Started server1
...
```

In this case, VIP is active on server1. Verification with the `ip` command is possible:

```
$ ip add show dev enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 08:00:27:df:29:09 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.10/24 brd 192.168.1.255 scope global noprefixroute enp0s3
       valid_lft forever preferred_lft forever
    inet 192.168.1.12/24 brd 192.168.1.255 scope global secondary enp0s3
       valid_lft forever preferred_lft forever
```

Toggle tests

From anywhere on the network, run the ping command on the VIP:

```
ping 192.168.1.12
```

Put the active node on standby:

```
sudo pcs node standby server1
```

Check that all pings succeed during the operation (no missing `icmp_seq`):

```
64 bytes from 192.168.1.12: icmp_seq=39 ttl=64 time=0.419 ms
64 bytes from 192.168.1.12: icmp_seq=40 ttl=64 time=0.043 ms
```

```
64 bytes from 192.168.1.12: icmp_seq=41 ttl=64 time=0.129 ms
64 bytes from 192.168.1.12: icmp_seq=42 ttl=64 time=0.074 ms
64 bytes from 192.168.1.12: icmp_seq=43 ttl=64 time=0.099 ms
64 bytes from 192.168.1.12: icmp_seq=44 ttl=64 time=0.044 ms
64 bytes from 192.168.1.12: icmp_seq=45 ttl=64 time=0.021 ms
64 bytes from 192.168.1.12: icmp_seq=46 ttl=64 time=0.058 ms
```

Check the cluster status:

```
$ sudo pcs status
Cluster name: mycluster
Cluster Summary:
...
  * 2 nodes configured
  * 1 resource instance configured

Node List:
  * Node server1: standby
  * Online: [ server2 ]

Full List of Resources:
  * myclusterVIP        (ocf:heartbeat:IPaddr2):         Started server2
```

The VIP has moved to server2. Check with the `ip add` command as before.

Return server1 to the pool:

```
sudo pcs node unstandby server1
```

Note

Once server1 has been `unstandby`, the cluster returns to its normal state, but the resource is not transferred back to server1: it remains on server2.

SERVICE CONFIGURATION

You will install the Apache service on both nodes of your cluster. This service is only started on the active node and will switch nodes at the same time as the VIP if the active node fails.

Refer to the Apache chapter for detailed installation instructions.

You must install `httpd` on both nodes:

```
sudo dnf install -y httpd
sudo firewall-cmd --permanent --add-service=http
sudo firewall-cmd --reload
```

Warning

Do not start or activate the service yourself. The Pacemaker will take care of it.

An HTML page containing the server name will show by default:

```
echo "<html><body>Node $(hostname -f)</body></html>" | sudo tee "/var/www/html/
index.html"
```

The Pacemaker resource agent will use the `/server-status` page (see Apache chapter) to determine its health status. You must activate it by creating the file `/etc/httpd/conf.d/status.conf` on both servers:

```
sudo vim /etc/httpd/conf.d/status.conf
<Location /server-status>
    SetHandler server-status
    Require local
</Location>
```

To create a resource, you will call "WebSite"; you will call the Apache script of the OCF resource and in the heartbeat namespace.

```
sudo pcs resource create WebSite ocf:heartbeat:apache configfile=/etc/httpd/
conf/httpd.conf statusurl="http://localhost/server-status" op monitor
interval=1min
```

The cluster will check Apache's health every minute (`op monitor interval=1min`).

Finally, to ensure that the Apache service starts on the same node as the VIP address, you must add a constraint to the cluster:

```
sudo pcs constraint colocation add WebSite with myclusterVIP INFINITY
```

Configuring the Apache service to start after the VIP is also possible. This can be useful if Apache has VHost configurations to listen to the VIP address (`Listen 192.168.1.12`):

```
$ sudo pcs constraint order myclusterVIP then WebSite
Adding myclusterVIP WebSite (kind: Mandatory) (Options: first-action=start
then-action=start)
```

Testing the failover

You will perform a failover and test that your web server is still available:

```
$ sudo pcs status
Cluster name: mycluster
Cluster Summary:
  * Stack: corosync (Pacemaker is running)
  * Current DC: server1 (version 2.1.7-5.el9_4-0f7f88312) - partition with
quorum
  ...

Node List:
  * Online: [ server1 server2 ]

Full List of Resources:
  * myclusterVIP        (ocf:heartbeat:IPaddr2):         Started server1
  * WebSite     (ocf:heartbeat:apache):  Started server1
```

You are currently working on server1.

```
$ curl http://192.168.1.12/
<html><body>Node server1</body></html>
```

Simulate a failure on server1:

```
sudo pcs node standby server1
```

```
$ curl http://192.168.1.12/
<html><body>Node server2</body></html>
```

As you can see, your web service is still working, but it is on server2 now.

```
sudo pcs node unstandby server1
```

Note that the service was only interrupted for a few seconds while the VIP switched over, and the services restarted.

16.2.4 Cluster troubleshooting

The `pcs status` command

The `pcs status` command provides information about the overall status of the cluster:

```
$ sudo pcs status
Cluster name: mycluster
Cluster Summary:
  * Stack: corosync (Pacemaker is running)
  * Current DC: server1 (version 2.1.7-5.el9_4-0f7f88312) - partition with
quorum
  * Last updated: Tue Jul  9 12:25:42 2024 on server1
  * Last change:  Tue Jul  9 12:10:55 2024 by root via root on server1
  * 2 nodes configured
  * 2 resource instances configured

Node List:
  * Online: [ server1 ]
  * OFFLINE: [ server2 ]

Full List of Resources:
  * myclusterVIP        (ocf:heartbeat:IPaddr2):         Started server1
  * WebSite     (ocf:heartbeat:apache):  Started server1

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

As you can see, one of the two servers is offline.

The `pcs status corosync` command

The `pcs status corosync` command provides information about the status of `corosync` nodes:

```
$ sudo pcs status corosync

Membership information
----------------------
    Nodeid      Votes Name
         1          1 server1 (local)
```

and once the server2 is back:

```
$ sudo pcs status corosync

Membership information
----------------------
    Nodeid      Votes Name
         1          1 server1 (local)
         2          1 server2
```

The `crm_mon` command

The `crm_mon` command returns cluster status information. Use the `-1` option to display the cluster status once and exit.

```
$ sudo crm_mon -1
Cluster Summary:
  * Stack: corosync (Pacemaker is running)
  * Current DC: server1 (version 2.1.7-5.el9_4-0f7f88312) - partition with
quorum
  * Last updated: Tue Jul  9 12:30:21 2024 on server1
  * Last change:  Tue Jul  9 12:10:55 2024 by root via root on server1
  * 2 nodes configured
  * 2 resource instances configured

Node List:
  * Online: [ server1 server2 ]

Active Resources:
  * myclusterVIP        (ocf:heartbeat:IPaddr2):         Started server1
  * WebSite     (ocf:heartbeat:apache):  Started server1
```

The `corosync-*cfgtool*` commands

The `corosync-cfgtool` command checks that the configuration is correct and that communication with the cluster is working properly:

```
$ sudo corosync-cfgtool -s
Local node ID 1, transport knet
LINK ID 0 udp
        addr    = 192.168.1.10
        status:
                nodeid:          1:     localhost
                nodeid:          2:     connected
```

The `corosync-cmapctl` command is a tool for accessing the object database. For example, you can use it to check the status of cluster member nodes:

```
$ sudo corosync-cmapctl  | grep members
runtime.members.1.config_version (u64) = 0
runtime.members.1.ip (str) = r(0) ip(192.168.1.10)
runtime.members.1.join_count (u32) = 1
runtime.members.1.status (str) = joined
runtime.members.2.config_version (u64) = 0
runtime.members.2.ip (str) = r(0) ip(192.168.1.11)
runtime.members.2.join_count (u32) = 2
runtime.members.2.status (str) = joined
```

16.2.5 Workshop

For this workshop, you will need two servers with Pacemaker services installed, configured, and secured, as described in the previous chapters.

You will configure a highly available Apache cluster.

Your two servers have the following IP addresses:

- server1: 192.168.1.10
- server2: 192.168.1.11

If you do not have a service to resolve names, fill the `/etc/hosts` file with content like the following:

```
$ cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6

192.168.1.10 server1 server1.rockylinux.lan
192.168.1.11 server2 server2.rockylinux.lan
```

You will use the VIP address of `192.168.1.12`.

Task 1: Installation and configuration

To install Pacemaker, enable the `highavailability` repository.

On both nodes:

```
sudo dnf config-manager --set-enabled highavailability
sudo dnf install pacemaker pcs
sudo firewall-cmd --permanent --add-service=high-availability
sudo firewall-cmd --reload
sudo systemctl enable pcsd --now
echo "pwdhacluster" | sudo passwd --stdin hacluster
```

On server1:

```
$ sudo pcs host auth server1 server2
Username: hacluster
Password:
server1: Authorized
server2: Authorized
$ sudo pcs cluster setup mycluster server1 server2
$ sudo pcs cluster start --all
$ sudo pcs cluster enable --all
$ sudo pcs property set stonith-enabled=false
```

Task 2: Adding a VIP

The first resource you will create on your cluster is a VIP.

```
pcs resource create myclusterVIP ocf:heartbeat:IPaddr2 ip=192.168.1.12
cidr_netmask=24 op monitor interval=30s
```

Check the cluster status:

```
$ sudo pcs status
Cluster name: mycluster
Cluster Summary:
...
  * 2 nodes configured
  * 1 resource instance configured

Node List:
  * Node server1: standby
  * Online: [ server2 ]

Full List of Resources:
  * myclusterVIP        (ocf:heartbeat:IPaddr2):         Started server2
```

Task 3: Installing the Apache server

Perform this installation on both nodes:

```
$ sudo dnf install -y httpd
$ sudo firewall-cmd --permanent --add-service=http
$ sudo firewall-cmd --reload
echo "<html><body>Node $(hostname -f)</body></html>" | sudo tee "/var/www/html/
index.html"
sudo vim /etc/httpd/conf.d/status.conf
<Location /server-status>
    SetHandler server-status
    Require local
</Location>
```

Task 4: Adding the `httpd` resource

Only on server1, add the new resource to the cluster with the needed constraints:

```
sudo pcs resource create WebSite ocf:heartbeat:apache configfile=/etc/httpd/
conf/httpd.conf statusurl="http://localhost/server-status" op monitor
interval=1min
sudo pcs constraint colocation add WebSite with myclusterVIP INFINITY
sudo pcs constraint order myclusterVIP then WebSite
```

Task 5: Test your cluster

You will perform a failover and test that your web server is still available:

```
$ sudo pcs status
Cluster name: mycluster
Cluster Summary:
  * Stack: corosync (Pacemaker is running)
  * Current DC: server1 (version 2.1.7-5.el9_4-0f7f88312) - partition with
quorum
  ...

Node List:
  * Online: [ server1 server2 ]

Full List of Resources:
  * myclusterVIP        (ocf:heartbeat:IPaddr2):         Started server1
  * WebSite     (ocf:heartbeat:apache):  Started server1
```

You are currently working on server1.

```
$ curl http://192.168.1.12/
<html><body>Node server1</body></html>
```

Simulate a failure on server1:

```
sudo pcs node standby server1
```

```
$ curl http://192.168.1.12/
<html><body>Node server2</body></html>
```

As you can see, your webservice is still working but on server2 now.

```
sudo pcs node unstandby server1
```

Note that the service was only interrupted for a few seconds while the VIP switched over and the services restarted.

16.2.6 Check your knowledge

✔ Is the `pcs` command the only one to control a pacemaker cluster?

✔ Which command returns the cluster state?

`sudo pcs status`

`systemctl status pcs`

`sudo crm_mon -1`

`sudo pacemaker -t`