Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Best Practices in Linux/63d76dd127750352bc2e8f86

TUTORIAL

Best Practices in Linux

Topics

Virtualization with VMware and VirtualBox

RHEL 9 provides the *virtualization* functionality, which enables a machine running RHEL 9 to *host* multiple virtual machines (VMs), also referred to as *guests*. VMs use the host's physical hardware and computing resources to run a separate, virtualized operating system (*guest OS*) as a user-space process on the host's operating system.

Advantages of virtualization

- Flexible and fine-grained allocation of resources
- Software-controlled configurations
- Separation from the host
- Space and cost efficiency
- Software compatibility

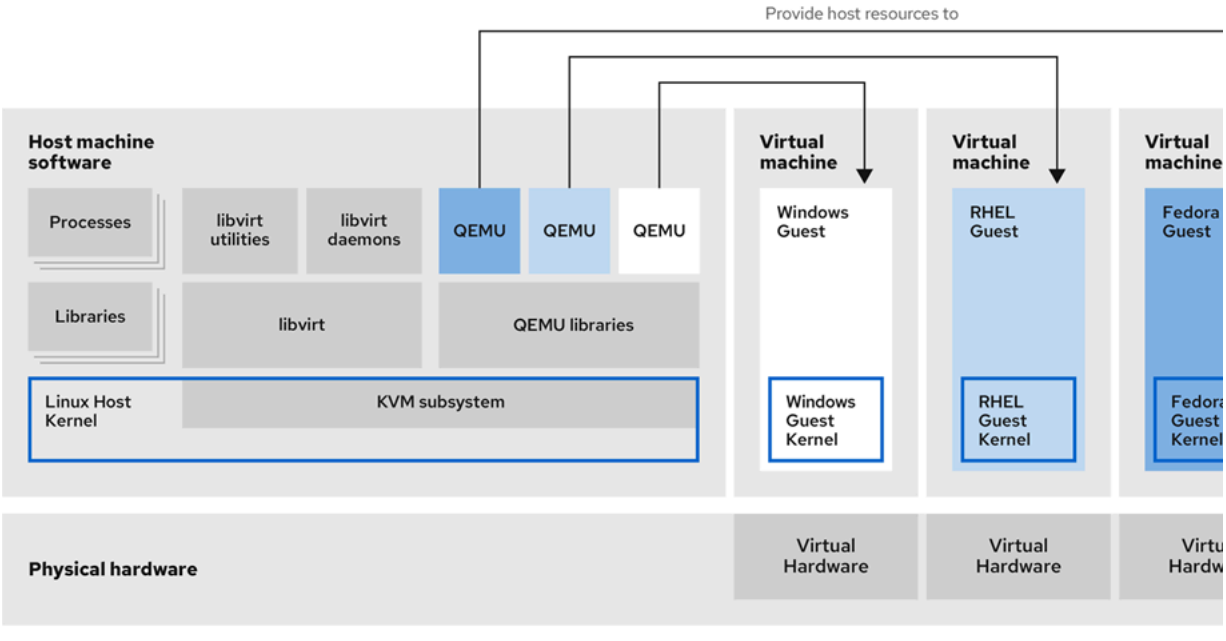

Image Source: https://access.redhat.com

Install KVM on RHEL 9

Kernel-based Virtual Machine, or KVM in short, is an opensource virtualization solution for the Linux kernel. It supports both Intel and AMD CPUs and allows users to create and manage virtual machines on a Linux system. The kernel functions as a hypervisor and allows you to virtualize your entire dedicated server and create multiple VMs.

Prerequisites

- Minimal Installed RHEL 9 with Desktop Environment
- Sudo user with admin rights
- Local Yum Repository or Red Hat Subscription

1.First, check whether virtualization is enabled or not on your system.

For Intel CPUs

```
$ grep -e 'vmx' /proc/cpuinfo
```

Or

```
$ lscpu | grep Virtualization
```

For AMD CPUs

```
$ grep -e 'svm' /proc/cpuinfo
```

2) Install Virtualization Packages

First of all, refresh the repositories and install all available updates

```
$ sudo dnf update -y
```

Once all the updates are installed successfully then reboot the system once

Next, install the virt-install and virt-viewer packages using the following command.

```
$ sudo dnf install virt-install virt-viewer -y
```

virt-install is a command-line tool for creating virtual machines from the command line.

The virt-viewer application is a lightweight UI interface that enables you to interact with the KVM virtual machine using VNC or SPICE remote desktop protocol.

Next, install the libvirt virtualization daemon.

```
$ sudo dnf install -y libvirt
```

Once the virtualization daemon has been installed, proceed and install virt-manager. This is a Qt-based graphical interface for managing virtual machines using the libvirt daemon.

```
$ sudo dnf install virt-manager -y
```

Then install additional virtualization tools to provide a seamless user experience.

```
$ sudo dnf install -y virt-top libguestfs-tools
```

3) Start and Enable Libvirtd Virtualization Daemon

Once you have installed all the required virtualization packages, be sure to start and enable the virtualization daemon as follows.

```
$ sudo systemctl start libvirtd
$ sudo systemctl enable libvirtd
```

Then verify if the daemon is running.

```
$ sudo systemctl status libvirtd
```

4) Configure Network Bridge for KVM

If you want to access your kvm virtual machines outside of your KVM hypervisor then you must configure a network bridge (kvmbr0) and attach physical interface to it.

Note: Virtual Bridge 'vbr0' automatically created when we install KVM packages. But this is used only for testing purpose. VMs will get the nated IP address via this bridge.

To create a network bridge kvmbr0, run following commands from the terminal,

```
$ nmcli connection show
$ sudo nmcli connection add type bridge autoconnect yes con-name kvmbr0 ifname kvmbr0
$ sudo nmcli connection modify kvmbr0 ipv4.addresses 192.168.1.179/24 gw4 192.168.1.1 ipv4.method
manual
$ sudo nmcli connection modify kvmbr0 ipv4.dns 192.168.1.1
$ sudo nmcli connection del enp0s3
$ sudo nmcli connection add type bridge-slave autoconnect yes con-name enp0s3 ifname enp0s3 master
kvmbr0
$ sudo nmcli connection up kvmbr0
```

Note: Replace the interface name and ip address details as per you setup.

Check network bridge (kvmbro) status using ip command,

```
$ ip addr show
```

5) Create Virtual Machine using Virt-Manager GUI

With all the packages required by KVM already installed along with network bridge configuration. we will now launch a virtual machine using the Virtual Machine Manager GUI utility.

Using the GNOME search tool, search and launch the Virtual Machine Manager.

Using version control (Git)

Version control is very useful to **keep track of changes you made to your scripts**. It allows you to choose when you have reached a stage in your code that you think is worth keeping track of, like a new function that makes your data analysis so much better.

What is git:

Git is a *free* and *open source* distributed *version control system*. It has many functionalities and was originally geared towards software development and production environment. In fact, Git was initially designed and developed in 2005 by Linux kernel developers (including Linus Torvalds) to track the development of the Linux kernel.

How does it work?

Git can be enabled on a specific folder/directory on your file system to version files within that directory (including sub-directories). In git (and other version control systems) terms, this "tracked folder" is called a **repository** (which formally is a specific data structure storing versioning information).

- Git stores snapshot of your files (that have changes)
- Git is distributed, meaning that all the copies of your repository are of equal value, containing all your codes and its history. There is no central repository
- Git has integrity, meaning each file is checked (summed) to be sure there was no bit loss during any file manipulation by git. Each snapshot (also called commit) has a unique identifier.

The Git workflow Overview

1. You modify files in your working directory and save them as usual
2. You **add** snapshots of your changes files to your staging area
3. You do a **commit**, which takes the files as they are in the staging area and permanently stores them as snapshots to your Git directory.

Repeat this process, every time you create a new snapshot, you add the new version of the file to the database, while keeping all the previous versions in the database. It creates an history that is like a graph that you can navigate:

Getting started with Git

Before you start using git on any computer, you will have to set your identity on your system, as every snapshot of files is associated with the user who implemented the modifications to the files.

1. Setting up your identity

You need to do this step the first time you use git on a computer.

Setup your profile:

Your name and email:

```
git config --global user.name "yourName"
git config --global user.email "yourEmail"
```

Optional:

Check that everything is correct:

```
git config --global --list
```

Modify everything at the same time:

```
git config --global --edit
```

Set your text editor:

```
git config --system core.editor vim
```

Cache your GitHub credentials (for 1 hour):

```
git config --global credential.helper 'cache --timeout=3600'
```

2. Starting a git repository

As we mentioned earlier, a **git repository** is a folder/directory on your machine in which content is monitored for any changes by git.

A.) Start versioning an existing folder on your system

`git init` is the command to start the tracking in a specific directory and transform it into a git repository:

```
mkdir oss
cd oss
mkdir dessert
cd dessert
pwd
git init
```

B.) Copying an existing repository to your system

`git clone` is the git command to copy an existing repository to your machine, more precisely adding the repository to the directory you are in.

```
cd
mkdir oss
cd oss
git clone https://github.com/your_username/your_reponame.git
```

The cloning process will automatically create a directory on your machine named like the online repository.

Tracking your changes

Let us have a closer look at the git workflow. It is important to stress that almost all of the git workflow happens on your local machine:

Example

Navigate in the `dessert` repository you just created.

1.) Let us create a csv file containing our name and favorite dessert:

```
vim favorite_desserts.csv
```

2.) Edit the new file adding headers and your info:

Note: hit the key `i` to switch to *insert mode* in vim. My file would look like this

```
Name, Dessert
Julien, Ice cream
```

Exit the insert mode `esc`

Exit vim and save `:wq`

3.) Add the new file to git:

```
git status
git add favorite_desserts.csv
git status
git commit -m "Julien's favorite dessert"
git status
```

4.) Add a friend to the csv:

```
vim favorite_desserts.csv

My name, My desert
Julien, Ice cream
Eliott, Crepes
```

Save and exit vim by typing `:wq`

5.) Add (stage) and commit the new version of the file

```
git status
git add favorite_desserts.csv
git status
git commit -m "Adding Eliott's favorite dessert"
git status
```

6.) Check the differences between the two last commits:

```
git diff HEAD~1
```

Note: hit q to exit

7.) We can also look at the log of commits to look a the commit sequence

```
git log
```

```
git log -1
```

Monitoring system performance

top Command

top command is used to show the Linux processes. It provides a dynamic real-time view of the running system. When you run top command, it will open an interactive command mode where the top half

portion will contain the statistics of processes and resource usage and Lower half contains a list of the currently running processes.

It will show the following fields:

- **PID:** Shows task's unique process id.
- **PR:** The process's priority. The lower the number, the higher the priority.
- **VIRT:** Total virtual memory used by the task.
- **USER:** User name of owner of task.
- **%CPU:** Represents the CPU usage.
- **TIME+:** CPU Time, the same as 'TIME', but reflecting more granularity through hundredths of a second.
- **SHR:** Represents the Shared Memory size (kb) used by a task.
- **NI:** Represents a Nice Value of task. A Negative nice value implies higher priority, and positive Nice value means lower priority.
- **%MEM:** Shows the Memory usage of task.
- **RES:** How much physical RAM the process is using, measured in kilobytes.
- **COMMAND:** The name of the command that started the process.

sar command

It can be used to monitor Linux system's resources like CPU usage, Memory utilization, I/O devices consumption, Network monitoring, Disk usage, process and thread allocation, battery performance, Plug and play devices, Processor performance, file system and more. Linux system Monitoring and analyzing aids understanding system resource usage which can help to improve system performance to handle more requests.

`If this command is not running, then install sysstat and enable the data collection by editing the value as true in the file /etc/default/sysstat.`

```
Syntax :
          $ sar -[ options ] time_interval number_of_tines_to_display
```

To start the service

```
 # systemctl start sysstat.service
```

To verify the sar version :

```
# sar -V
```

- To report CPU details total 5 times with the interval of 2 seconds

```
# sar -u 2 5
To report about the amount of memory used, amount of memory free, available cache, available buffers
total 3 times with the interval of 1 second
# sar -r 1 3
To report about file systems mounted on the device total 5 times with the interval of 2 seconds.
```

```
# sar -F 2 5
To report cpu usage for given core :
# sar -P 1 1 3
To report about network interface, network speed, IPV4, TCPV4, ICMPV4 network traffic and errors
# sar -n DEV 1 3 | egrep -v lo
To report statistics about swapping
# sar -S 1 3
To report details about I/O operations like transaction per second, read per second, write per
second
# sar -b 1 3
To report paging statistics (KBs paged-in/sec, KBs paged-out/sec, pagefault/sec etc.)
# sar -B 2 5
```