USEFUL COMMANDS USED IN LINUX

YOU WILL COME ACROSS THESE COMMANDS IN UBUNTU WHILE INSTALLING GRAFANA.

Why does Docker offer installation packages in two common formats, .deb and .rpm, for different types of Linux systems?

Docker offers installation packages in two common formats, .deb and .rpm, for different types of Linux systems

Installation Packages: Docker is a software tool used to create and manage containers. To make it easy for people to install Docker on their Linux computers, Docker provides special packages.

.deb and .rpm: These are two common formats for packaging software on Linux. Think of them like different types of gift wrapping paper. Some Linux systems use .deb (like Ubuntu), while others use .rpm (like Fedora). Docker provides packages in both formats, so it can be installed on various Linux systems.

Linux Distributions and Architectures: Linux comes in many versions, called distributions (e.g., Ubuntu, Fedora, CentOS). Each distribution can run on different types of computer hardware, known as architectures (e.g., x86, ARM). Docker makes sure to offer packages that work with different distributions and architectures, so more people can use it on their Linux computers.

So, in simple words, Docker provides different types of packages to make it easy for people with various Linux systems to install and use Docker on their computers.

How is .deb different from .rpm?

deb and .rpm are both package file formats that are used to install software on Linux systems. However, there are some key differences between the two formats.

.deb packages are used by Debian-based Linux distributions, such as Ubuntu, Debian, and Kali Linux. .rpm packages are used by Red Hat-based Linux distributions, such as Fedora, CentOS, and Red Hat Enterprise Linux.

Another difference between .deb and .rpm packages is the way they are installed. .deb packages are installed using the dpkg command, while .rpm packages are installed using the rpm command.

Finally, .deb and .rpm packages also differ in terms of their structure. .deb packages contain a number of control files and data files, while .rpm packages contain a database of information about the package.

Here is a simple analogy to explain the difference between .deb and .rpm packages:

Imagine you are downloading a software application for your computer. The application may be available in either a .deb or .rpm package format.

The .deb package is like a ZIP file for Debian-based Linux distributions. The .rpm package is like a ZIP file for Red Hat-based Linux distributions.

To install the software application, you need to use the appropriate package manager for your Linux distribution. If you are using a Debian-based Linux distribution, you need to use the dpkg command to install the .deb package. If you are using a Red Hat-based Linux distribution, you need to use the rpm command to install the .rpm package.

Once the package is installed, you can start using the software application.

Which package format you should use depends on the Linux distribution you are using. If you are not sure which package format to use, you can consult the documentation for your Linux distribution.

How do I know if I am installing Docker on a Gnome Desktop environment?

To check if you're installing Docker on a GNOME Desktop environment in Ubuntu, follow these simple steps:

Open the Terminal: Click on the "Activities" menu at the top-left corner of your screen (it looks like a grid of squares) to open the GNOME Applications menu. In the search bar, type "Terminal" and click on the Terminal application that appears. It's like opening a command prompt on Windows.

Check for GNOME Desktop: In the terminal, you can verify if you're using the GNOME Desktop environment by running the following command:

echo $XDG_CURRENT_DESKTOP

Then press Enter. If it displays "GNOME," it means you are indeed using the GNOME Desktop environment.

If it shows "GNOME," it's like confirming that you're in a room with the lights turned on.

If it shows something else, it means you might be using a different desktop environment.

This check helps ensure that you're installing Docker on a GNOME Desktop environment in Ubuntu.

Explain below terms in Dockers.

Docker image

Docker Client

Docker hub

Docker Daemon

Docker image: A Docker image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. Docker images are built from a Dockerfile, which is a text file that contains instructions on how to build the image.

Docker client: The Docker client is a command-line tool that you use to interact with the Docker daemon. The Docker client allows you to build, run, and manage Docker images and containers.

Docker Hub: Docker Hub is a registry for Docker images. Docker Hub allows you to store and share your Docker images with others.

Docker daemon: The Docker daemon is a background service that manages Docker images and containers. The Docker daemon is responsible for building, running, and stopping Docker containers.

Here is a simple analogy to explain the above terms:

Imagine you are building a house. The Docker image is like the blueprint for the house. The Docker client is like the architect who uses the blueprint to build the house. The Docker daemon is like the construction workers who build the house. And Docker Hub is like a library of blueprints that you can use to build your house.

How Docker works:

You build a Docker image from a Dockerfile.

You push the Docker image to Docker Hub.

You pull the Docker image from Docker Hub to your local machine.

You run the Docker image to create a container.

You interact with the container using the Docker client.

Docker containers are isolated from each other, which means that they do not share resources or affect each other. This makes Docker containers ideal for running multiple applications on the same server.

Docker is a powerful tool that can be used to improve the way you develop, deploy, and manage your applications.

What is difference between Docker Image and Docker Container?

The difference between Docker Image and Docker Container as below:

Docker Image:

Docker Image is like a frozen pizza recipe.

It's a package that contains all the ingredients and instructions needed to make a pizza, but it's not the pizza itself.

You can have many copies of this recipe (image) to make as many pizzas (containers) as you want.

It's a static, unchanging file.

Docker Container:

Docker Container is like a freshly baked pizza.

It's a running instance of the image, like having a hot, ready-to-eat pizza.

Containers are where your application actually runs and can be active, like enjoying your delicious pizza.

It's dynamic and can change as your application runs.

In simple terms, a Docker Image is the recipe or blueprint, while a Docker Container is the actual, running instance created from that recipe. Images are static, and containers are dynamic and active, like the difference between a pizza recipe and a pizza you're eating.

What happens when you install Docker Desktop and Docker Engine on same machine?

When Docker Desktop starts, it creates a dedicated context for itself. This context is used to isolate Docker Desktop from any other Docker installations that may be running on the system, such as a local Docker Engine.

The dedicated context is also set as the current context in use. This means that when you run Docker commands from the Docker Desktop terminal, the commands will target the Docker Desktop context.

This is done to avoid a clash with a local Docker Engine that may be running on the Linux host and using the default context. If Docker Desktop did not use its own dedicated context, Docker commands would target the local Docker Engine instead.

When Docker Desktop shuts down, it resets the current context to the previous one. This means that Docker commands will start targeting the local Docker Engine again.

Here is a simple analogy to explain the above:

Imagine you have two different versions of a software application installed on your computer. You want to use one version of the application for work, and the other version for personal use.

To avoid accidentally using the wrong version of the application, you could create two different user accounts on your computer. One user account would be used for work, and the other user account would be used for personal use.

Docker Desktop uses a similar approach to isolate itself from any other Docker installations that may be running on the system. It creates its own dedicated context, which is like a separate user account for Docker.

This allows you to use Docker Desktop without having to worry about interfering with any other Docker installations on the system.