Understanding package manager and systemctl

Day 7 : 90Days of DevOps Challenge

SS Shivraj Salunkhe · Mar 30, 2023 · 6 min read

What is a package manager in Linux?

1. In **Linux**, a package manager is a software tool that helps users to find, install, update, and remove **software packages** on their system. It simplifies the process of software installation and management by providing a centralized database of available packages and their dependencies, along with tools to **download** and **install** them automatically.
2. **Package managers** typically work with software repositories, which are collections of packages maintained by the distribution or **third-party** developers. They use a package format that includes the software files along with metadata, such as the package name, version number, and dependencies.
3. Some common Linux package managers include APT (Advanced Packaging Tool) used by **Debian** and **Ubuntu**, **YUM** (Yellowdog Updater, Modified) used by Red Hat and CentOS, and Pacman used by Arch Linux. Each package manager has its own set of commands and syntax, but they all serve the same basic purpose of simplifying software installation and management on Linux systems.

What is a package?

1. In the context of **software** and **Linux**, a package is a collection of files and software components that are bundled together to provide a specific **functionality** or **application**.
2. A **package** typically contains the software's executable files, libraries, configuration files, and other necessary components required for the software to **run** properly. It may also include documentation, examples, and other resources that are useful for users.

3. Packages are usually distributed in a **compressed format**, such as **tarball** or **zip**, and are often managed by a package manager, which handles tasks like **installation**, **updating**, and **removing packages**.
4. By bundling software components into **packages**, developers and system administrators can ensure that software is **installed** and **configured** correctly, and that any **dependencies** are met. Additionally, packages can be easily distributed and shared, making it simpler to **deploy software** across different **systems** and **environments.**

Different kinds of package managers

There are various kinds of package managers available for Linux and other operating systems, each with its own set of features and capabilities. Some common types of package managers include:

- **System package managers -** These are the default package managers that come with the operating system, and are used to install, update, and remove software packages system-wide. Examples include APT, YUM, and Pacman.
- **Language-specific package managers -** These are package managers that are designed to manage software packages for specific programming languages, such as Python's pip, Ruby's gem, and Node.js' npm.
- **Application-specific package managers -** Some software applications come with their own package managers that are used to manage packages specific to that application. Examples include Composer for PHP, and Docker's package manager for container images.

- **Source-based package managers -** These are package managers that install software packages from source code, rather than pre-compiled binaries. Examples include Portage for Gentoo, and Nix for NixOS.
- **Package managers for container images -** These are package managers that are used to manage software packages and dependencies within container images, such as Docker's package manager.

Each type of package manager has its own **strengths** and **weaknesses**, and is suited to different use **cases** and **scenarios**. Ultimately, the choice of package manager will depend on the specific requirements of the **user or organization**.

Task

-> here are the steps to install Docker and Jenkins on Linux using package managers:

Installing Docker:

1. Open a terminal on your Linux system.
2. Update the package index by running the command:

```
sudo apt update
```

3. Install the Docker package dependencies by running the command:

```
sudo apt install apt-transport-https ca-certificates curl gnupg lsb
```

4. Add the Docker GPG key by running the command:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gp
```

5. Add the Docker repository by running the command:

```
 echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

6. Update the package index again by running the command:

```
 sudo apt update
```

7. Install Docker by running the command:

```
sudo apt install docker-ce docker-ce-cli containerd.io
```

8. Verify that Docker is installed and running by running the command:

```
sudo systemctl status docker
```

   You should see a message indicating that the Docker service is **active** and **running**.

Installing Jenkins:

1. Open a terminal on your Linux system.
2. Add the Jenkins repository key by running the command:

```
sudo wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.k
sudo apt-key add -
```

3. Add the Jenkins repository by running the command:

```
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ >
```

4. Update the package index by running the command:

```
sudo apt update
```

5. Install Jenkins by running the command:

```
sudo apt install jenkins
```

What is systemctl and systemd ?

1. `systemctl` is a command-line utility in Linux that is used to control the systemd system and service manager. `systemd` is a system and service manager that is responsible for starting and stopping system services and managing system resources.
2. `systemd` is a replacement for the traditional System V init system that was used in many Linux distributions. It provides a number of advantages over the System V init system, such as faster boot times, parallel service startup, and better dependency tracking.

3. **`systemctl` allows users to control and manage the systemd system and services. Some common `systemctl` commands include:**
    - `systemctl start <service>`: Starts a service
    - `systemctl stop <service>`: Stops a service
    - `systemctl restart <service>`: Restarts a service
    - `systemctl enable <service>`: Enables a service to start automatically at boot time
    - `systemctl disable <service>`: Disables a service from starting automatically at boot time
    - `systemctl status <service>`: Displays the status of a service
4. `systemd` uses a configuration file format called "unit files" to define services, targets, and other system resources. Unit files are stored in the `/etc/systemd/system` directory and have a `.service`, `.target`, or `.socket` extension, depending on the type of resource they define. Users can create and modify unit files to customize system behavior and configure system services.

Difference Between systemctl vs service

-> `systemctl` is used to control and manage the systemd system and services. It is the preferred method for managing services on modern Linux distributions that use systemd as their system and service manager. `systemctl` provides more fine-grained

control over system services than the `service` command and supports additional features like socket activation and resource management.

-> `service`, on the other hand, is a legacy command-line utility used in older Linux distributions that used the System V init system. It is still included in many modern Linux distributions for compatibility with older scripts and service management tools. `service` provides basic control over system services, such as starting, stopping, and restarting them, but it does not support the more advanced features of `systemctl`.

In **summary**, `systemctl` is the preferred utility for managing system services in modern Linux distributions that use systemd, while `service` is a legacy utility that is still included for compatibility with older Linux distributions and scripts.

Thank You for Reading the Blog.. 👍

connect with me : linkedin.com/in/shivraj-salunkhe-5881141a4

follow my blog channel : **shivrajofficial.hashnode.dev**

Enter your email address SUBSCRIBE

Devops Linux Learning Journey learning shell script

WRITTEN BY

Shivraj Salunkhe

Follow

Dedicated and hardworking undergraduate student pursuing a degree in Information Technology with a passion for learning, leadership experience, and real-world skills through internships and part-time jobs in IT sector. Actively looking for new Opportunities and committed to do personal and professional growth.

MORE ARTICLES

Shivraj Salunkhe

File Permissions and Access Control Lists

Shivraj Salunkhe

Advanced Linux Shell Scripting for DevOps

Shivraj Salunkhe

Basic Linux Shell Scripting for DevOps Engineers

In Linux, file permissions refer to the settings that determine who can read, write, or execute a fi...

Engineers with User management

Advanced Linux Shell Scripting for DevOps Engineers covers a wide range of topics, including : Auto...

What is Kernel ? In computing, a kernel is the core component of an operating system that manages s...