

# How to Create and Manage Systemd Services in Linux

Written By: *Saikat Goswami*

---

This article is a complete, in-depth guide—suitable for beginners, system administrators, DevOps engineers, and developers—on creating, enabling, managing, debugging, and optimizing **systemd services**.

## Contents

How to Create and Manage Systemd Services in Linux .....	1
1.    What is the <i>init</i> process in Linux ?.....	3
2.    Introduction to systemd.....	3
2. Understanding Systemd Unit Files .....	3
3.    What is a unit file? .....	5
3.1 Common unit file types.....	5
3.2 Sections of a unit file.....	5
<b>Explanation</b> .....	5
3.3 A sample systemd Service File .....	6
Explanation of Sections .....	6
[Unit] Section .....	6
[Service] Section.....	7
<b>Common Directives:</b> .....	7
<b>Service Types:</b> .....	7
[Install] Section.....	7
Systemd follows dependencies. In the file if you have: .....	8
Another option in the file is ‘ <b>Requires</b> =’ : .....	8
Systemd services often need environment variables for need for things like: .....	8
Adding Inline in the Service File. Add the following under [Service] section:.....	8
Load from File .....	8

Running as root is unsafe. To run as a non-root user, add:.....	9
Restrict File Access For restricting file access, add the following entries in your service file: .....	9
Disable Networking For enabling and disabling networking in your service, add the following entry in your service file:.....	9
Restrict Read/Write Access For restricting read/write access, add the following entries in your service file:.....	9
Disable Capability Escalation.....	9
Use systemd sandboxing.....	9
Using systemd for Automatic Restarts and Resilience .....	10
<b>4 Managing Systemd Services.....</b>	<b>10</b>
4.1 To <b>start</b> service, type:.....	10
4.2 To <b>stop</b> service, type: .....	10
4.3 To <b>restart</b> service type: .....	10
4.4 To <b>reload</b> service type:.....	11
4.5 If you want your service to startup at boot, type: .....	11
4.6 If you do not want your service to startup at boot, type:.....	11
4.7 If you want to check if your service is configured to startup at boot: .....	11
<b>5. Creating a Custom Systemd Service.....</b>	<b>11</b>
5.1 Let's create a real service for a simple shell script.....	11
5.2 Viewing Logs with Journald.....	13
<b>6. Using systemd Timers Instead of Cron Jobs .....</b>	<b>15</b>
<b>7. Debugging and Troubleshooting systemd Services.....</b>	<b>15</b>
7.1 Check Logs.....	15
7.2 Check Startup Time and Dependencies .....	16
<b>8. Real-World Examples .....</b>	<b>18</b>
8.1 Docker Container Auto-Start Using systemd.....	18
<b>9. Best Practices for Writing Systemd Service Files.....</b>	<b>18</b>
<b>10.Conclusion.....</b>	<b>18</b>

## 1. What is the *init* process in Linux ?

The **init** process in Linux is the *first process* started by the kernel after booting. Always runs as process ID 1, making it the root of the process tree. It is responsible for booting the system, managing services, tracking logs, supervising processes, and ensuring that the system remains stable and predictable.

## 2. Introduction to systemd

**Systemd** has become the standard init system on most modern Linux distributions, including Ubuntu, Debian, CentOS, Fedora, RHEL, AlmaLinux, Rocky Linux and many others. Systemd is an init system and service manager designed to overcome the limitations of traditional init scripts (SysVinit). Earlier Linux systems relied on `/etc/init.d/` shell scripts that executed sequentially. As systems grew more complex and boot speeds became a priority, a more modern, parallelized, event-driven system was required.

Systemd provides:

- Faster boot times through parallelization
- A standardized service configuration format
- Built-in logging via **journald**
- Automatic service restarts and supervision
- Dependency-based startup
- Integration with **cgroups** for resource management

Most importantly, systemd introduces **unit files**—small declarative configuration files that define services, mount points, timers, devices, and more.

This article focuses specifically on **systemd service units** and teaches you how to write, install, run, monitor, troubleshoot, and optimize them.

## 2. Understanding Systemd Unit Files

A **systemd ‘unit file’** acts as a blueprint that tells systemd what to start, when to start it, and under what conditions it should run. For `.service` files specifically, the unit defines details such as the service’s executable command, its startup behavior, dependencies, restart policies, and logging configuration. These files make service management predictable and consistent because all operational rules are declared in one structured place.

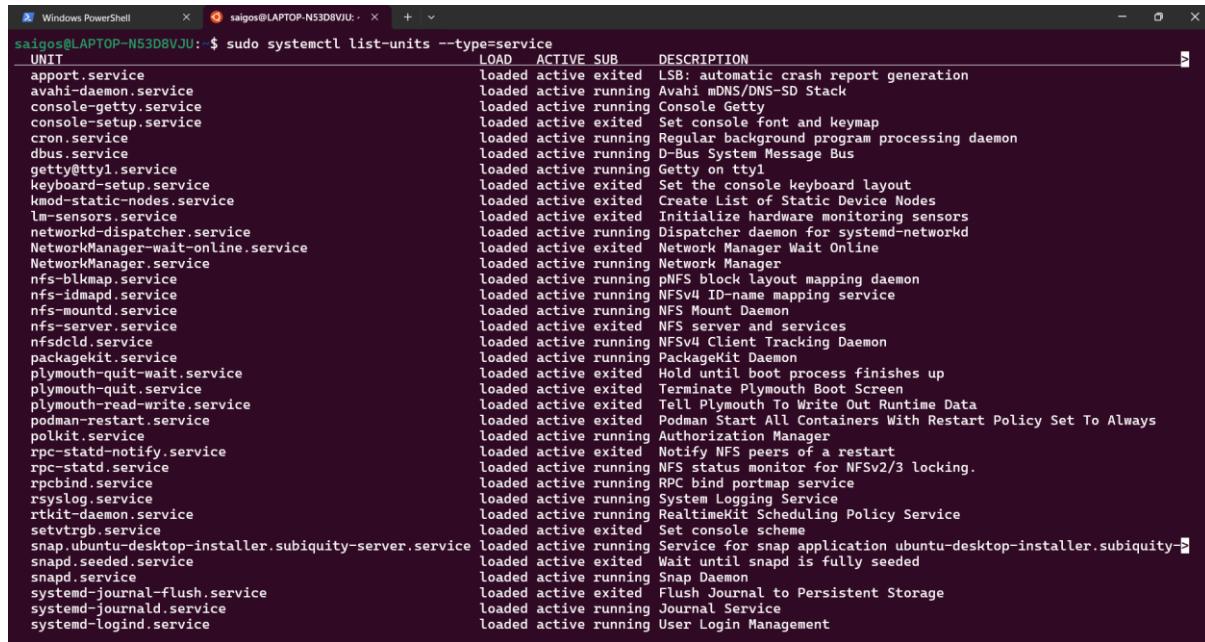
A unit file is a simple text file describing how systemd should manage a resource. For services, the file extension is `.service`.

Unit files typically reside in:

Directory	Purpose
<code>/usr/lib/systemd/system/</code>	Distribution-provided services
<code>/etc/systemd/system/</code>	Administrator-created or overridden services
<code>/run/systemd/system/</code>	Runtime-generated units

To list all active services, type:

```
systemctl list-units --type=service
```



```
saigos@LAPTOP-N53DBVJU: ~$ sudo systemctl list-units --type=service
UNIT                                     LOAD   ACTIVE SUB   DESCRIPTION
apport.service                           loaded  active exited  LSB: automatic crash report generation
avahi-daemon.service                    loaded  active running Avahi mDNS/DNS-SD Stack
console-getty.service                  loaded  active running Console Getty
console-setup.service                  loaded  active exited  Set console font and keymap
cron.service                            loaded  active running Regular background program processing daemon
dbus.service                            loaded  active running D-Bus System Message Bus
getty@tty1.service                     loaded  active running Getty on tty1
keyboard-setup.service                 loaded  active exited  Set the console keyboard layout
kmod-static-nodes.service              loaded  active exited  Create List of Static Device Nodes
lm-sensors.service                     loaded  active exited  Initialize hardware monitoring sensors
networkd-dispatcher.service           loaded  active running Dispatcher daemon for systemd-network
NetworkManager-wait-online.service    loaded  active exited  Network Manager Wait Online
NetworkManager.service                 loaded  active running Network Manager
nfs-blkmap.service                    loaded  active running pNFS block layout mapping daemon
nfs-idmapd.service                   loaded  active running NFSv4 ID-name mapping service
nfs-mountd.service                   loaded  active running NFS Mount Daemon
nfs-server.service                   loaded  active exited  NFS server and services
nfsdcl.d.service                     loaded  active running NFSv4 Client Tracking Daemon
packagekit.service                   loaded  active running PackageKit Daemon
plymouth-quit-wait.service            loaded  active exited  Hold until boot process finishes up
plymouth-quit.service                loaded  active exited  Terminate Plymouth Boot Screen
plymouth-read-write.service          loaded  active exited  Tell Plymouth To Write Out Runtime Data
podman-restart.service               loaded  active exited  Podman Start All Containers With Restart Policy Set To Always
polkit.service                        loaded  active running Authorization Manager
rpc-statd-notify.service             loaded  active exited  Notify NFS peers of a restart
rpc-statd.service                   loaded  active running NFS status monitor for NFSv2/3 locking.
rpcbind.service                      loaded  active running RPC bind portmap service
rsyslog.service                      loaded  active running System Logging Service
rtkit-daemon.service                loaded  active running RealtimeKit Scheduling Policy Service
setvtrgb.service                    loaded  active exited  Set console scheme
snap.ubuntu-desktop-installer.subiquity-server.service loaded  active running Service for snap application ubuntu-desktop-installer.subiquity->
snapd.seeded.service                loaded  active exited  Wait until snapd is fully seeded
snapd.service                        loaded  active running Snap Daemon
systemd-journal-flush.service       loaded  active exited  Flush Journal to Persistent Storage
systemd-journald.service            loaded  active running Journal Service
systemd-logind.service              loaded  active running User Login Management
```

To list all installed services:

```
systemctl list-unit-files --type=service
```

UNIT FILE	STATE	VENDOR_PRESET
accounts-daemon.service	enabled	enabled
apparmor.service	enabled	enabled
apport-autoreport.service	static	-
apport-forward@.service	static	-
apport.service	generated	-
apt-daily-upgrade.service	static	-
apt-daily.service	static	-
apt-news.service	static	-
auth-rpcgss-module.service	static	-
autovt@.service	alias	-
avahi-daemon.service	enabled	enabled
bluetooth.service	enabled	enabled
bolt.service	static	-
cloud-config.service	enabled	enabled
cloud-final.service	enabled	enabled
cloud-init-hotplugd.service	static	-
cloud-init-local.service	enabled	enabled
cloud-init.service	enabled	enabled
cni-dhcp.service	disabled	enabled
colord.service	static	-
configure-printer@.service	static	-
console-getty.service	enabled-runtime	disabled
console-setup.service	enabled	enabled
container-getty@.service	static	-
cron.service	enabled	enabled
cryptdisks-early.service	masked	enabled
cryptdisks.service	masked	enabled
dbus-fi.wl.wpa_supplicant1.service	alias	-
dbus-org.bluez.service	alias	-
dbus-org.freedesktop.Avahi.service	alias	-
dbus-org.freedesktop.hostname1.service	alias	-
dbus-org.freedesktop.locale1.service	alias	-
dbus-org.freedesktop.login1.service	alias	-
dbus-org.freedesktop.ModemManager1.service	alias	-
dbus-org.freedesktop.nm-dispatcher.service	alias	-
dbus-org.freedesktop.resolve.service	alias	-

### 3. What is a unit file?

- A **unit file** is a configuration file that tells systemd how to manage a resource (service, socket, device, mount, etc.).
- All unit files live under `/etc/systemd/system/`, `/lib/systemd/system/`, or `/usr/lib/systemd/system/`.
- Each unit file has a **type**, indicated by its suffix.

#### 3.1 Common unit file types

- `.service` → defines how to start/stop/manage a service (e.g., `nginx.service`).
- `.socket` → describes a network socket or IPC socket.
- `.target` → groups units together (like runlevels).
- `.mount` → controls filesystem mounts.
- `.timer` → schedules tasks (like cron).
- `.device`, `.path`, etc. → specialized resources.

#### 3.2 Sections of a unit file

A unit file contains sections such as:

#### Explanation

- **[Unit] section** → metadata (description, dependencies).

- **[Service] section** → runtime configuration for the service process (command, environment variables, working directory, etc.).
- **[Install] section** → installation details (wanted-by targets).

### 3.3 A sample systemd Service File

A sample service file looks like this:

```
[Unit]
Description=My Sample Service
After=network.target

[Service]
Type=simple
ExecStart=/usr/bin/python3 /home/user/app.py

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

#### Explanation of Sections

##### [Unit] Section

The [Unit] section describes:

- A short description
- Dependencies
- Behavior before starting the service

Common directives:

<b>Directive</b>	<b>Description</b>
Description=	Human-readable description
After=	Start service after another unit
Before=	Opposite of After
Requires=	Hard dependency; fails if missing
Wants=	Soft dependency

Example:

```
After=network-online.target
Requires=docker.service
```

## [Service] Section

The heart of the service.

### Common Directives:

Directive	Purpose
Type=	How the process behaves
ExecStart=	The command to start the service
ExecReload=	Command to reload
ExecStop=	Command to stop
Restart=	Restart policies
Environment=	Set environment variables
User= / Group=	Non-root service execution
WorkingDirectory=	Set app directory

### Restart policies examples:

```
Restart=on-failure
RestartSec=3
```

## Service Types:

Type	Meaning
Simple	Default; runs a foreground process
Forking	Background/daemonized processes
Oneshot	Runs to completion (e.g., scripts)
Notify	App notifies systemd when ready
Idle	Starts after all jobs are loaded

## [Install] Section

Under this section, consider an example entry:

```
WantedBy=multi-user.target
```

It means when enabled, start this service whenever the system reaches multi-user mode (normal boot).

### Common targets:

Target	Description
multi-user.target	Normal CLI system

Target	Description
graphical.target	Desktop
network.target	Networking available
local-fs.target	Filesystems ready

Systemd follows dependencies. In the file if you have:

```
After=network-online.target
```

This means your service will start *after* network-online.target has reached “active”.

Another option in the file is ‘`Requires=`’:

```
Requires=network-online.target
```

If network-online.target service fails, your service won't start. Think of `Requires=` as a mandatory dependency. Like saying: “Start with me, and if you fail, I fail too.”

Systemd services often need environment variables for need for things like:

- application mode (`APP_ENV=production`)
- ports (`APP_PORT=9000`)
- database URLs  
(`DB_URL=mysql://user:password@localhost:3306/mydatabase`)

Adding Inline in the Service File. Add the following under [Service] section:

```
Environment="APP_ENV=production"
```

## Load from File

Create a file:

```
vi/etc/myapp/env
```

Add variables:

```
[Service]
APP_ENV=production
APP_PORT=9000
```

Reference it in your unit file:

```
EnvironmentFile=/etc/myapp/env
```

The content looks like this:

```
[Service]
Environment="APP_ENV=production"
Environment="APP_PORT=9000"
```

**Running as root is unsafe. To run as a non-root user, add:**

```
User=appuser
Group=appuser
```

## Restrict File Access

For restricting file access, add the following entries in your service file:

```
ProtectSystem=full
ProtectHome=true
```

## Disable Networking

For enabling and disabling networking in your service, add the following entry in your service file:

```
PrivateNetwork=true
```

## Restrict Read/Write Access

For restricting read/write access, add the following entries in your service file:

```
ReadOnlyPaths=/usr
ReadWritePaths=/var/log/myapp
```

## Disable Capability Escalation

```
NoNewPrivileges=true
```

## Use systemd sandboxing

```
ProtectKernelTunables=yes
ProtectProc=invisible
RestrictSUIDSGID=yes
```

These features significantly harden services in production environments.

## Using systemd for Automatic Restarts and Resilience

Systemd can restart your app automatically under many circumstances.

Examples:

```
Restart=on-failure  
RestartSec=5
```

Other types:

Policy	Meaning
No	Never restart
on-success	Restart on exit code 0
on-failure	Restart on non-zero exit
on-abnormal	Restart on signal
Always	Always restart

Example for high reliability:

```
Restart=always  
RestartSec=1  
StartLimitInterval=400  
StartLimitBurst=10
```

## 4 Managing Systemd Services

### 4.1 To **start** service, type:

```
sudo systemctl start myapp.service
```

### 4.2 To **stop** service, type:

```
sudo systemctl stop myapp.service
```

### 4.3 To **restart** service type:

```
sudo systemctl restart myapp.service
```

Stops and then starts the service. Required when:

- You've changed binaries or environment variables.
- The service is stuck or misbehaving.
- You want a clean slate.

#### 4.4 To **reload** service type:

```
sudo systemctl reload myapp.service
```

Sends a reload signal (usually `SIGHUP`) to the running process. The service keeps running but re-reads its configuration files. Required when:

- You've updated config files
- You want changes applied without downtime.

**Limitation:** Reloading only works if `ExecReload` is defined in the Unit file. If not, the command may fail or do nothing.

#### 4.5 If you want your service to startup at boot, type:

```
sudo systemctl enable myapp.service
```

#### 4.6 If you do not want your service to startup at boot, type:

```
sudo systemctl disable myapp.service
```

#### 4.7 If you want to check if your service is configured to startup at boot:

```
systemctl is-enabled myapp.service
```

## 5. Creating a Custom Systemd Service

### 5.1 Let's create a real service for a simple shell script.

#### Step 1: Write Your Application

Example script `/home/saigos/hello/hello.sh`:

```
#!/bin/bash
while true; do
    echo "Hello from systemd service!" >> /home/saigos/hello/output.log
    sleep 5
done
```

Make it executable:

```
chmod a+x /home/saigos/hello/hello.sh
```

#### Step 2: Create the Service File

Create a file:

```
sudo nano /etc/systemd/system/hello.service
```

Paste:

```
[Unit]
Description=Hello Test Service
After=network.target

[Service]
ExecStart=/home/saigos/hello/hello.sh
Restart=always
RestartSec=2
User=root

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

### Step 3: Reload systemd

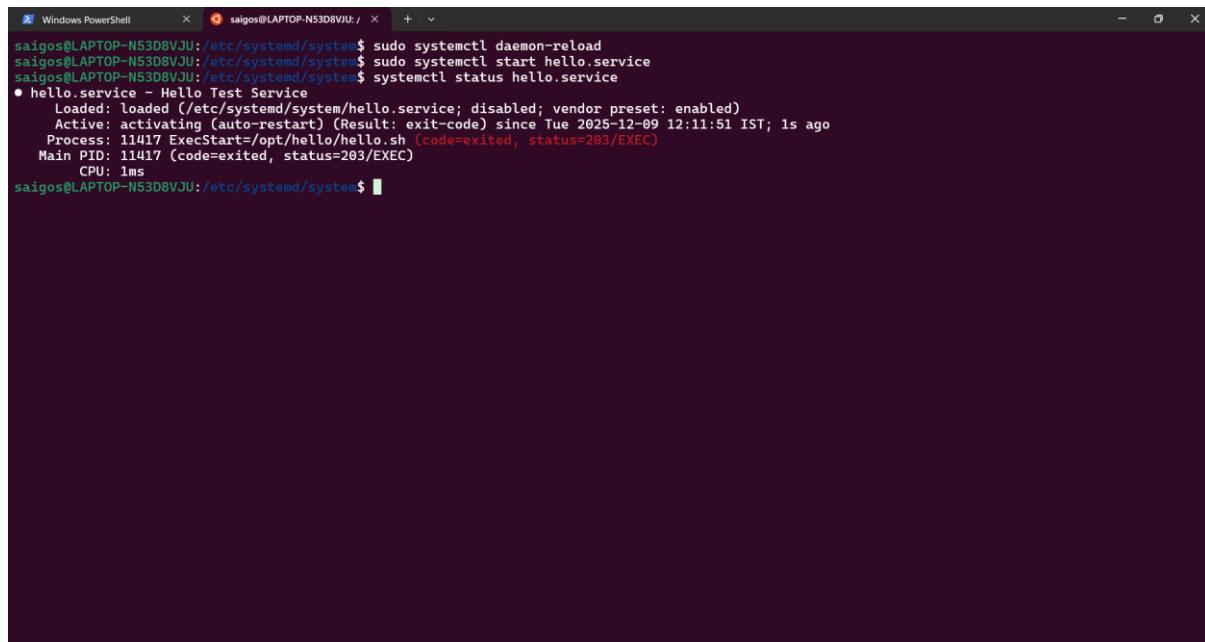
```
sudo systemctl daemon-reload
```

### Step 4: Start the Service

```
sudo systemctl start hello.service
```

To check the status, check status:

```
systemctl status hello.service
```

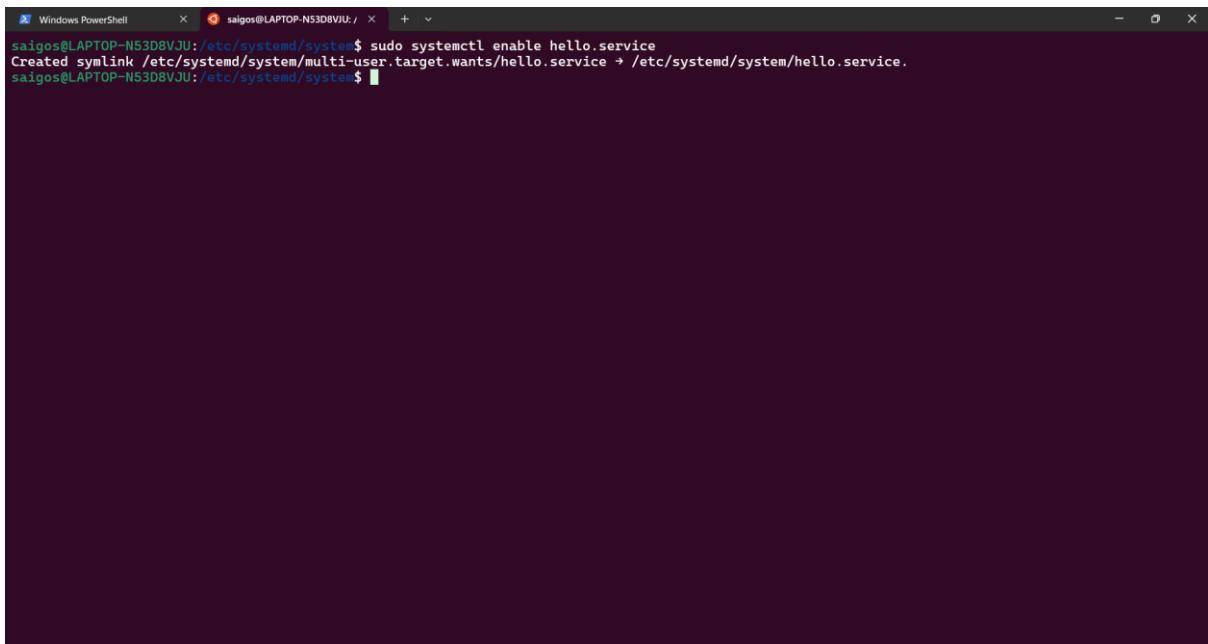


The screenshot shows a Windows PowerShell window with the title bar "Windows PowerShell". The command history and output are as follows:

```
saigos@LAPTOP-N53D8VJU:/etc/systemd/system$ sudo systemctl daemon-reload
saigos@LAPTOP-N53D8VJU:/etc/systemd/system$ sudo systemctl start hello.service
saigos@LAPTOP-N53D8VJU:/etc/systemd/system$ systemctl status hello.service
● hello.service - Hello Test Service
   Loaded: loaded (/etc/systemd/system/hello.service; disabled; vendor preset: enabled)
     Active: activating (auto-restart) (Result: exit-code) since Tue 2025-12-09 12:11:51 IST; 1s ago
       Process: 11417 ExecStart=/opt/hello/hello.sh (code=exited, status=203/EXEC)
    Main PID: 11417 (code=exited, status=203/EXEC)
      CPU: 1ms
saigos@LAPTOP-N53D8VJU:/etc/systemd/system$
```

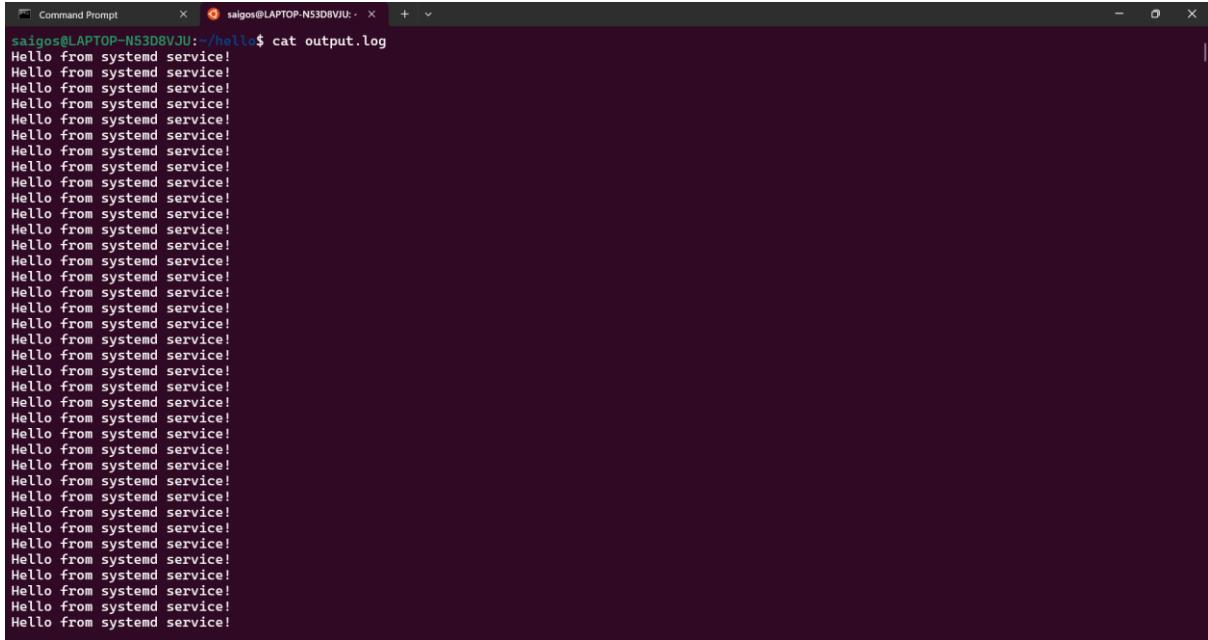
### Step 5: If you want to start hello.service at boot, type

```
sudo systemctl enable hello.service
```



```
Windows PowerShell saigos@LAPTOP-N53D8VJU: /etc/systemd/system$ sudo systemctl enable hello.service
Created symlink /etc/systemd/system/multi-user.target.wants/hello.service → /etc/systemd/system/hello.service.
saigos@LAPTOP-N53D8VJU: /etc/systemd/system$
```

See output at /home/saigos/hello/output.log:

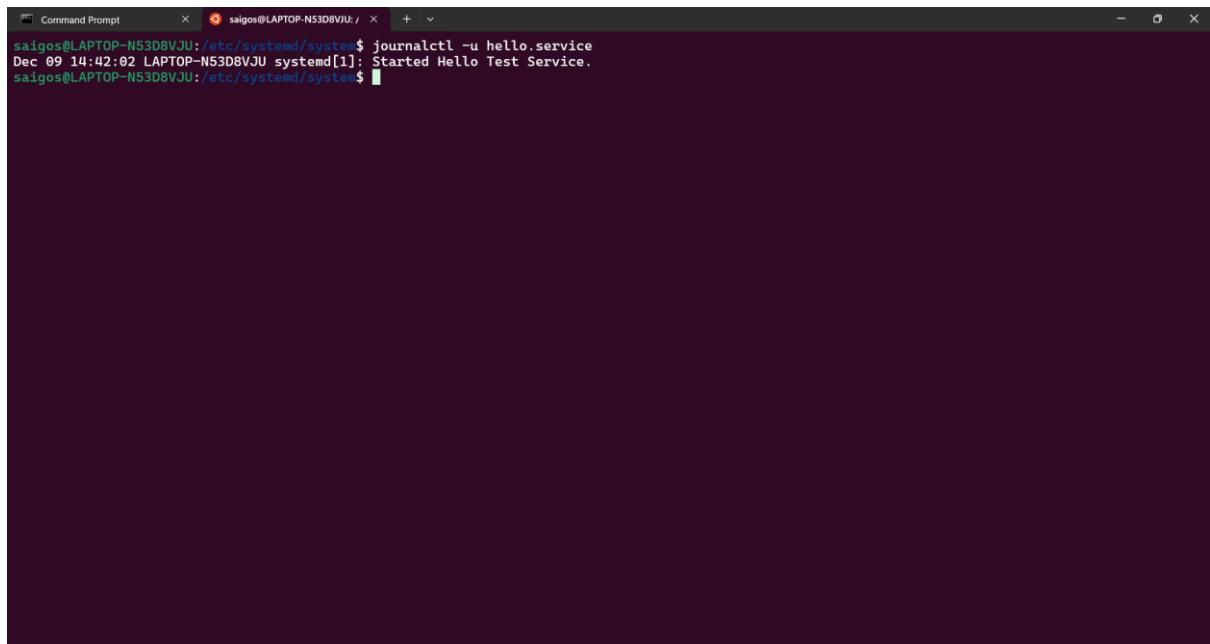


```
Command Prompt saigos@LAPTOP-N53D8VJU:~/hello$ cat output.log
Hello from systemd service!
```

## 5.2 Viewing Logs with Journald

Systemd integrates with journald:

```
journalctl -u hello.service
```

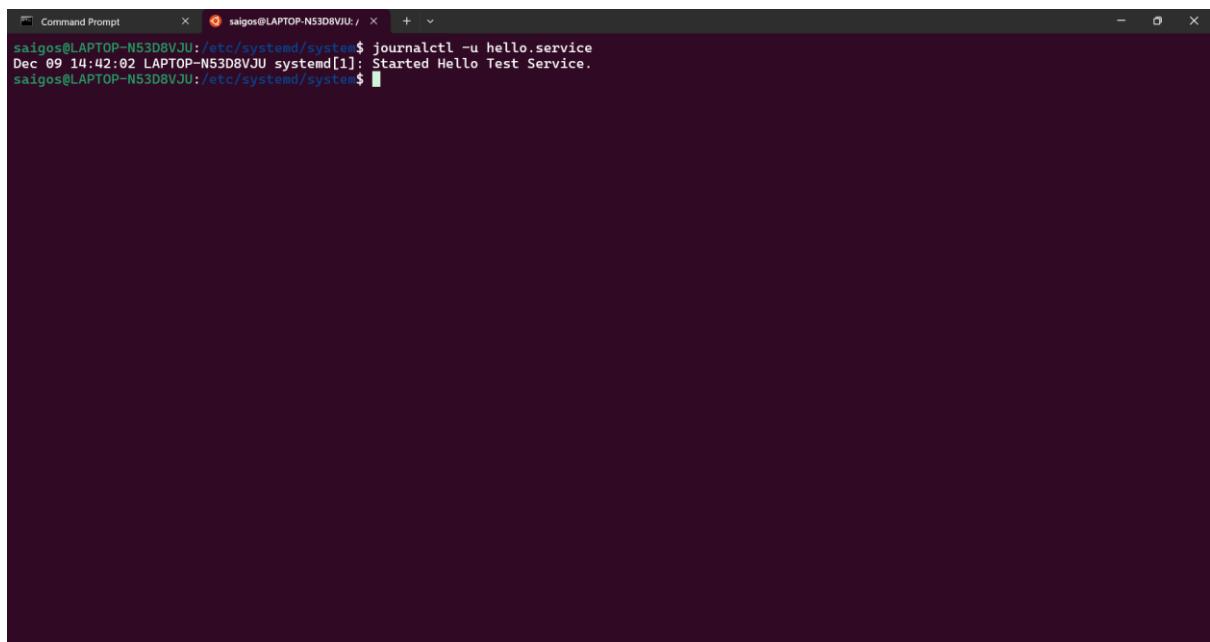


```
saigos@LAPTOP-N53DBVJU:/etc/systemd/system$ journalctl -u hello.service
Dec 09 14:42:02 LAPTOP-N53DBVJU systemd[1]: Started Hello Test Service.
saigos@LAPTOP-N53DBVJU:/etc/systemd/system$
```

You can view logs. Systemd integrates with **journald**. Type:

```
journalctl -u hello.service
```

You will see the output:



```
saigos@LAPTOP-N53DBVJU:/etc/systemd/system$ journalctl -u hello.service
Dec 09 14:42:02 LAPTOP-N53DBVJU systemd[1]: Started Hello Test Service.
saigos@LAPTOP-N53DBVJU:/etc/systemd/system$
```

Follow logs live:

```
journalctl -u hello.service -f
```

Show only today's logs:

```
journalctl -u hello.service --since=today
```

## 6. Using systemd Timers Instead of Cron Jobs

Systemd timers replace cron jobs and integrate better with logs, dependencies, and failures.

Example timer unit (backup.timer):

```
[Unit]
Description=Run backup every day

[Timer]
OnCalendar=daily
Persistent=true

[Install]
WantedBy=timers.target
```

Timer calls backup.service.

Enable:

```
systemctl enable --now backup.timer
```

List timers:

```
systemctl list-timers
```

---

## 7. Debugging and Troubleshooting systemd Services

### 7.1 Check Logs

```
journalctl -xe
```

```

Command Prompt saigos@LAPTOP-N53DBVJU: / + -
Support: http://www.ubuntu.com/support

System Journal (/var/log/journal/dbb74d415e5f4bd8a92bf24dd926d22e) is currently using 840.1M.
Maximum allowed usage is set to 4.0G.
Leaving at least 4.0G free (of currently available 947.8G of disk space).
Enforced usage limit is thus 4.0G, of which 3.1G are still available.

The limits controlling how much disk space is used by the journal may
be configured with SystemMaxUse=, SystemKeepFree=, SystemMaxFileSize=,
RuntimeMaxUse=, RuntimeKeepFree=. RuntimeMaxFileSize= settings in
/etc/systemd/journald.conf. See journald.conf(5) for details.

Dec 09 14:38:20 LAPTOP-N53DBVJU sudo[1287]: pam_unix(sudo:session): session closed for user root
Dec 09 14:38:55 LAPTOP-N53DBVJU sudo[1294]: saigos : TTY=pts/0 ; PWD=/etc/systemd/system ; USER=root ; COMMAND=/usr/bin/journalctl --vacuum-t
Dec 09 14:38:55 LAPTOP-N53DBVJU sudo[1294]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Dec 09 14:38:55 LAPTOP-N53DBVJU sudo[1294]: pam_unix(sudo:session): session closed for user root
Dec 09 14:39:56 LAPTOP-N53DBVJU sudo[1306]: saigos : TTY=pts/0 ; PWD=/etc/systemd/system ; USER=root ; COMMAND=/usr/bin/systemctl start hello
Dec 09 14:39:56 LAPTOP-N53DBVJU sudo[1306]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Dec 09 14:39:56 LAPTOP-N53DBVJU sudo[1306]: pam_unix(sudo:session): session closed for user root
Dec 09 14:41:56 LAPTOP-N53DBVJU sudo[1317]: saigos : TTY=pts/0 ; PWD=/etc/systemd/system ; USER=root ; COMMAND=/usr/bin/vi hello.service
Dec 09 14:41:56 LAPTOP-N53DBVJU sudo[1317]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Dec 09 14:41:56 LAPTOP-N53DBVJU sudo[1317]: pam_unix(sudo:session): session closed for user root
Dec 09 14:42:02 LAPTOP-N53DBVJU sudo[1320]: saigos : TTY=pts/0 ; PWD=/etc/systemd/system ; USER=root ; COMMAND=/usr/bin/systemctl start hello
Dec 09 14:42:02 LAPTOP-N53DBVJU sudo[1320]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Dec 09 14:42:02 LAPTOP-N53DBVJU systemd[1]: Started Hello Test Service.
Subject: A start job for unit hello.service has finished successfully
Defined-By: systemd
Support: http://www.ubuntu.com/support

A start job for unit hello.service has finished successfully.

The job identifier is 579.
Dec 09 14:42:02 LAPTOP-N53DBVJU sudo[1320]: pam_unix(sudo:session): session closed for user root
Dec 09 15:17:02 LAPTOP-N53DBVJU CRON[1915]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Dec 09 15:17:02 LAPTOP-N53DBVJU CRON[1916]: (root) CMD ( cd / && run-parts --report /etc/cron.hourly)
Dec 09 15:17:02 LAPTOP-N53DBVJU CRON[1915]: pam_unix(cron:session): session closed for user root
Dec 09 15:27:42 LAPTOP-N53DBVJU sudo[2065]: saigos : TTY=pts/0 ; PWD=/etc/systemd/system ; USER=root ; COMMAND=/usr/bin/journalctl -xe
Dec 09 15:27:42 LAPTOP-N53DBVJU sudo[2065]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
lines 4-40/40 (END)
```

## 7.2 Check Startup Time and Dependencies

systemd-analyze blame

```

Command Prompt saigos@LAPTOP-N53DBVJU: /etc/systemd/system$ systemd-analyze blame
Command 'system-analyze' not found, did you mean:
  command 'systemd-analyze' from deb systemd (249.11-0ubuntu3.17)
Try: sudo apt install <deb name>
saigos@LAPTOP-N53DBVJU: /etc/systemd/system$ systemd-analyze blame
3.897s snapd.seed.service
3.646s snapd.service
2.128s landscape-client.service
1.248s podman-restart.service
701ms dev-sdd.device
692ms networkd-dispatcher.service
629ms NetworkManager-wait-online.service
526ms podman-auto-update.service
347ms systemd-resolved.service
252ms systemd-udev-trigger.service
227ms systemd-timesyncd.service
226ms e2scrub_reap.service
213ms user@1000.service
206ms nfs-server.service
204ms systemd-journal-flush.service
190ms systemd-logind.service
168ms rpcbind.service
156ms nfs-mountd.service
154ms NetworkManager.service
148ms avahi-daemon.service
133ms apport.service
130ms systemd-udevd.service
126ms polkit.service
106ms rpc-statd.service
100ms systemd-tmpfiles-clean.service
95ms run-rpc-pipefs.mount
88ms systemd-journald.service
75ms lm-sensors.service
73ms nfs-idmapd.service
67ms systemd-update-utmp.service
65ms wpa_supplicant.service
64ms rsyslog.service
59ms podman.service
```

Graph the boot process:

systemd-analyze plot > boot.svg

Ubuntu 22.04 LTS LAPTOP-NS5D6V3U (Linux 6.6.7-2-microsoft-standard-WSL2 #1 SMP PREEMPT\_DYNAMIC Thu Jun 5 18:30:46 UTC 2025) x86\_64



## 8. Real-World Examples

### 8.1 Docker Container Auto-Start Using systemd

```
[Unit]
Description=Run My Docker Container
After=docker.service
Requires=docker.service

[Service]
ExecStart=/usr/bin/docker run --rm --name myapp -p 8080:8080 myimage
ExecStop=/usr/bin/docker stop myapp
Restart=always

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

## 9. Best Practices for Writing Systemd Service Files

1. Never run apps as root unless necessary
2. Use `Restart=on-failure` to ensure reliability
3. Keep unit files in `/etc/systemd/system/`
4. It is recommended to use logs via `journald` instead of custom logging
5. Use `EnvironmentFile` instead of hardcoding values
6. Prefer absolute paths in unit files
7. Use `systemd-analyze verify` before enabling

Following such practices ensures stable, maintainable services.

## 10. Conclusion

Creating and managing systemd services is one of the most essential skills for working with modern Linux systems. Systemd is not just a replacement for SysVinit—it is a powerful framework that handles service lifecycles, supervision, logging, security, sandboxing, scheduling, and system orchestration.

By understanding unit file structures, dependencies, environment variables, service types, logging, and security features, you can build robust and reliable services tailored to your system or production environment.

Whether you're deploying a Python web app, managing Docker containers, setting up automated backups, or writing a lightweight daemon, **systemd provides the speed, stability, and control you need.**