Introduction to Linux: **Services and Remote Access**

Philipp Fruck & Pius Walter

System Services and Remote Systems

Goal of this Lecture

- How to run persistent, scheduled and background services
- How to inspect logs, analyze failures & perform troubleshooting
- How to securely connect to remote systems
- How Linux manages networking and DNS
- How to sandbox services with systemd

Why?

- 90 % of infrastructure runs on headless Linux servers
- You must control systems without a GUI, even during incidents
- Understanding services & logs is key for:
  - intrusion detection
  - forensic analysis
  - disaster recovery

systemd: Service Management

Start/Stop/Enable

```
sudo systemctl start nginx
sudo systemctl enable nginx
# hard restart (drop connections)
sudo systemctl restart nginx
# or soft reload (keep connections)
sudo systemctl reload nginx
```

Shortcuts:

```
# combine start and enable
systemctl [enable|disable] --now ...
# show logs (new since systemd 258)
systemctl [start|stop|restart] -v ...
```

Check status

```
systemctl status sshd
systemctl status --user pipewire
```

List failed services

```
systemctl list-units --failed
```

Analyze boot time

```
systemd-analyze
systemd-analyze blame
systemd-analyze critical-chain
```

systemd: Service Unit Basics

Example: `/etc/systemd/system/example.service`

```
[Unit]
Description=Example App

[Service]
ExecStart=/usr/local/bin/app
Restart=always
User=appuser

[Install]
#WantedBy=graphical.target    # GUI ready
#WantedBy=multi-user.target   # CLI ready
WantedBy=default.target       # GUI or CLI
#WantedBy=network.target      # network ready
```

User vs System scope

- system: `/etc/systemd/system/`
- user: `~/.config/systemd/user/`

Enable and run

```
sudo systemctl [--user] daemon-reload
sudo systemctl [--user] enable --now example.service
```

Dependencies

```
[Unit]
# start serviceA when starting serviceB.
Requires=serviceA.service
# ensure serviceA is fully started.
After=serviceA.service
```

journalctl: Viewing Logs

System log viewer (replaces many legacy tools like `dmesg` and `/var/log/*`)

Basic usage

```
journalctl -u sshd.service
journalctl -b    # since boot
journalctl -b  1 # since first boot
journalctl -b -1 # since last boot
journalctl -f    # live follow
# commonly used: eXplain, End, Follow
journalctl [--user] -xefu <unit>
```

Filter by user

```
journalctl _UID=1000
```

```
# equivalent to sudo dmesg --ctime
sudo journalctl -ke
# equivalent to
# sudo cat /var/log/audit/audit.log
sudo journalctl -t audit \
    --no-pager --output=cat
```

SSH: Server, Keys and Permissions

Server config: `/etc/ssh/sshd_config`

```
PasswordAuthentication no
PermitRootLogin no
Port 22
```

Apply changes

```
sudo sshd -t # ensure valid config!
sudo systemctl reload sshd  # keep sessions
sudo systemctl restart sshd # drop sessions
```

Generate and copy key pair

```
ssh-keygen -t ed25519 -C "you@hostname"
ssh-copy-id user@host
```

Permissions matter!

File	Permissions
`~/.ssh`	700
`~/.ssh/authorized_keys`	600
`~/.ssh/id_*`	600

```
chmod 700 ~/.ssh
chmod 600 ~/.ssh/*
```

`~/.ssh/authorized_keys` holds the public keys that are authorized to log in as the given user.

`ssh-copy-id` is a shortcut for running e.g.

```
echo "ssh-ed25519 AAAA.... mypubkey" \
    >> ~/.ssh/authorized_keys
```

> Use SSH key authentication over passwords!

SSH: Jump Hosts and Port Forwarding

Jump Hosts (ProxyJump)

For hosts only reachable via another host

```
ssh -J user@jump.example.com user@target
```

Permanent config: `~/.ssh/config`

```
Host internal
    HostName 10.0.0.5
    ProxyJump jump.example.com
```

```
ssh internal
```

Port Forwarding

Local: expose a remote service locally

```
ssh -L 8080:localhost:80 user@host
```

Remote: expose your local service remotely

```
ssh -R 2222:localhost:22 user@host
```

SOCKS5 proxy: full encrypted tunnel

```
ssh -D 1080 user@host
```

tmux: Terminal Multiplexing

Persistent sessions which survive disconnects

- Split terminal into panes
- Shared sessions for teamwork

Basic usage

```
tmux                # start
tmux ls             # list sessions
tmux attach -t <id>  # reattach
```

Basic keys (press `Ctrl+B` first)

Action	Keys
Split vertically	`%`
Split horizontally	`"`
New window	`c`
Switch windows	`0-9`
Detach session	`d`

Activate mouse mode by pressing `Ctrl+B` and `:` then entering `set -g mouse on`.

Networking

Modern

- `NetworkManager.service` (e.g. on Fedora)
  - `nmcli`/`nmtui` on the CLI
- `systemd-networkd.service`
  - Relies on config files
  - More suitable for servers

Traditional

- `networking.service` (e.g. Debian)
  - Uses `/etc/network/interfaces`

Ubuntu

- Netplan abstraction layer
  - NetworkManager for GUI
  - systemd-networkd for server

Universal

IP command is universal, but not persistent

```
# Address Add
ip a a 192.168.7.42 dev enp0s1
# Address Delete
ip a d 192.168.7.42 dev enp0s1
```

Naming

- Traditional names: `eth0`, `eth1`, ...
- `systemd` naming scheme (enp0s1):
  - `en` = Ethernet, `wl` = Wi-Fi
  - `s` = slot, `u` = port
  - `man systemd.net-naming-scheme`

DNS

Traditional `/etc/resolv.conf`

Plain text file storing DNS servers and search domains:

```
nameserver 1.1.1.1
search example.com
```

- Managed manually or by scripts
  - e.g. `ifup`/`dhclient`
- Applications read it directly

`systemd-resolved`

- `systemd` service providing DNS resolution
- `resolvectl status`
- Maintains its own dynamic DNS configuration
- `/etc/resolv.conf` is usually a **symlink** to
  - `/run/systemd/resolve/stub-resolv.conf`
    - uses 127.0.0.53 as local DNS stub
  - Or `/run/systemd/resolve/resolv.conf`
    - full configuration
- Benefits:
  - Support multiple interfaces with different DNS
  - Provides caching

Key Difference:
`/etc/resolv.conf` is **static**, whereas `systemd-resolved` is a **dynamic DNS resolver** that updates `/etc/resolv.conf` or provides a local stub for applications.

Scheduled Tasks

Traditional: Cron

```
crontab -e # edit
crontab -l # list
```

```
┌──────── minute (0 - 59)
│ ┌────── hour (0 - 23)
│ │ ┌──── day of month (1 - 31)
│ │ │ ┌── month (1 - 12)
│ │ │ │ ┌ day of week (0 - 7)
│ │ │ │ │   (Sunday=0 or 7)
* * * * *  command_to_run
```

Q: How do we define a nightly backup job at 4 AM?

A: `0 4 * * * backup.sh`

Modern: systemd timers

- Name indicates which service to start
  - `foo.timer` triggers `foo.service`
- e.g. `/etc/systemd/system/myservice.timer`

```
[Unit]
Description=Runs myservice.service
[Timer]
# Run 5min after system has booted
OnBootSec=5m
# Run every hour
OnCalendar=hourly
[Install]
WantedBy=timers.target
```

```
systemctl list-timers
```

Preference: systemd timers -> better logging, dependency handling, boot triggers

Hardening Services: systemd-analyze security

Services often run as root with full system access. systemd can sandbox them without changing the application.

```
# exposure overview of all running services
systemd-analyze security
# detailed report for a single unit
systemd-analyze security sshd.service
```

- Each setting contributes to an **exposure level** from 0.0 (safe) to 10.0 (UNSAFE)
- ✓ = restricted, ✗ = exposed
- The score is a hint, not a guarantee: e.g. sshd legitimately needs a lot of access

An analysis of NowNewPrivileges for Debian packages:

> https://media.ccc.de/v/all-systems-go-2026-400-the-state-of-systemd-sandboxing-in-debian

Hardening Services: Directives

Add to the `[Service]` section

```
[Service]
DynamicUser=yes
NoNewPrivileges=yes
ProtectSystem=strict
ProtectHome=yes
PrivateTmp=yes
PrivateDevices=yes
CapabilityBoundingSet=
RestrictAddressFamilies=AF_INET AF_INET6
SystemCallFilter=@system-service
```

Directive	Effect
`DynamicUser`	Temporary unprivileged user
`NoNewPrivileges`	Blocks `setuid` escalation
`ProtectSystem=strict`	File system is read-only
`ProtectHome`	`/home` is inaccessible
`PrivateTmp`	Own, isolated `/tmp`
`CapabilityBoundingSet=`	Drops all capabilities
`RestrictAddressFamilies`	Limits socket types
`SystemCallFilter`	Allow-list of syscalls (seccomp)

Use systemctl edit <unit> to add hardening via a drop-in without touching the vendor unit file.

Exercise

1. List the exposure of all services: `systemd-analyze security`
2. Find the most exposed service and inspect it: `systemd-analyze security <unit>`
3. Create a service `/etc/systemd/system/hello.service` which runs `/usr/bin/sleep 600`
4. Check its score with `systemd-analyze security hello.service`
5. Add hardening directives with `systemctl edit hello.service` until the score drops below 3.0
6. Restart the unit, then check `systemctl status hello` and `journalctl -u hello`

Bonus: Try the directives ad hoc without a unit file

```
systemd-run -p ProtectSystem=strict -p PrivateTmp=yes --pty bash
```

Thank you!

Thank you for your attention!

Don't forget the feedback in Moodle please!