

DevOps Foundations Handbook — From Linux Basics to Job-Ready DevOps (2025 Edition)

Audience: IT graduates/newcomers with minimal Linux/DevOps exposure.

Goal: Learn Linux fundamentals end-to-end, then connect them to DevOps workflows (Docker/Kubernetes/CI-CD).

Style: Plain English, step-by-step labs, copy-paste commands, quick checks, and interview angles.

How to Use This Book

- **Practice-first:** Every chapter includes a **Lab** you can run on Ubuntu 22.04 (or Rocky/RHEL 9) VM or WSL2. If a command differs by distro, both variants are shown.
- **Safe sandbox:** Prefer a VM or cloud instance (e.g., free tier) to avoid breaking your main OS.
- **Copy/paste blocks:** Anything in a code block can be pasted into the terminal. Read the comment lines (starting with `#`).
- **Symbols:**
 - ✓ **Quick Check** — validate learning
 - 📖 **Lab** — step-by-step practice
 - ! **Pitfall** — common mistakes
 - 🗨️ **Interview** — how to explain in interviews

Minimum Lab Setup 1. Create **two Linux VMs** (Ubuntu Server 22.04 recommended): `vm-linux1` (server), `vm-linux2` (client).

2. Ensure outbound internet works.

3. Create a non-root user with sudo: `sudo adduser devops && sudo usermod -aG sudo devops`.

Part I — Linux Fundamentals

Chapter 1 — Kernel (The Brain of Linux)

Concept: The kernel mediates between hardware and user apps. It manages CPU, memory, devices, filesystems, and networking.

Why it matters (DevOps): Performance tuning, debugging crashes, container isolation, and security rely on kernel features.

🔧 **Lab: Inspect Kernel & Tune a Runtime Parameter**

```
# Show kernel version and flavor
uname -a

# List kernel modules (drivers/features)
lsmod | head

# View kernel boot messages (recent)
dmesg | tail -n 50

# Read a runtime kernel parameter (example: IP forwarding)
cat /proc/sys/net/ipv4/ip_forward

# Enable IPv4 forwarding temporarily (until reboot)
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward

# Persist it across reboots (Ubuntu/Debian)
echo 'net.ipv4.ip_forward=1' | sudo tee /etc/sysctl.d/99-ipforward.conf
sudo sysctl --system
```

✓ **Quick Check:** Explain the difference between changing `/proc/sys/...` and setting `/etc/sysctl*.conf`.

■ **Interview:** “The kernel is a monolithic kernel that exposes tunables via `sysctl` and `/proc`; we avoid permanent changes in `/proc` and persist via `sysctl` configs.”

Chapter 2 — Boot Process (Power-on to Login)

Stages: Firmware (BIOS/UEFI) → Bootloader (GRUB) → Kernel + initramfs → `systemd` → login.

Lab: Read Boot Logs & Default Target

```
# Last boot messages (systemd journal)
sudo journalctl -b -0 --no-pager | head -n 40

# Previous boot
sudo journalctl -b -1 -n 50

# Default boot target (runlevel equivalent)
systemctl get-default
```

✓ **Quick Check:** What’s the difference between `multi-user.target` and `graphical.target`?

! **Pitfall:** Editing GRUB without backup. Always `sudo cp /etc/default/grub{,.bak}` before changes; then `sudo update-grub` (Debian/Ubuntu) or `sudo grub2-mkconfig -o /boot/grub2/grub.cfg` (RHEL-family).

Chapter 3 — User Space vs Kernel Space

Concept: Apps run in **user space**; kernel operates in **privileged space**. System calls bridge them.

 **Lab: Watch System Calls with** `strace`

```
# Trace syscalls of a simple command
strace -o /tmp/ls.trace -f ls >/dev/null 2>&1
wc -l /tmp/ls.trace # count syscalls captured
head -n 10 /tmp/ls.trace
```

✓ **Quick Check:** Why can't user space touch hardware directly?

Chapter 4 — Run Levels → Systemd Targets

Concept: Legacy runlevels map to systemd "targets".

 **Lab: Switch Targets Safely**

```
systemctl list-units --type=target
systemctl get-default
sudo systemctl set-default multi-user.target # non-GUI
# Reboot when convenient to test
```

■ **Interview:** "Runlevels are abstracted as systemd targets; we manage them via `systemctl set-default`."

Chapter 5 — Linux Processes (Lifecycle & Control)

States: running (R), sleeping (S), uninterruptible (D), stopped (T), zombie (Z).

Tools: `ps`, `top` / `htop`, `nice` / `renice`, `kill`.

 **Lab: Observe & Manage Processes**

```
ps aux --sort=-%cpu | head

# Start a CPU burner in background (safe)
yes > /dev/null & # press Enter to get shell back
jobs -l

# Lower its priority (higher niceness)
renice +10 <PID>

# Send SIGTERM, then SIGKILL if needed
kill -15 <PID>
kill -9 <PID>
```

✓ **Quick Check:** When do you prefer SIGTERM over SIGKILL?

Chapter 6 — Inter-Process Communication (IPC)

Mechanisms: pipes, FIFOs, shared memory, message queues, sockets.

Lab: Named Pipe (FIFO)

```
mkfifo /tmp/demo.fifo
# Terminal A
cat /tmp/demo.fifo
# Terminal B
echo "hello via fifo" > /tmp/demo.fifo
```

✓ **Quick Check:** How do FIFOs differ from regular files?

Chapter 7 — Signals (SIGTERM, SIGKILL, ...)

Lab: Trap Signals in a Bash Script

```
cat > /tmp/signal_demo.sh <<'EOF'
#!/usr/bin/env bash
trap 'echo "Caught SIGTERM, cleaning up..."; exit 0' SIGTERM
trap 'echo "Caught SIGINT (Ctrl+C)'" SIGINT

echo "PID $$ running; send me SIGTERM or press Ctrl+C"
while true; do sleep 2; done
EOF
```

```
chmod +x /tmp/signal_demo.sh
/tmp/signal_demo.sh &
# In another terminal: kill -15 <PID>
```

✓ **Quick Check:** Why can't a process catch SIGKILL?

Chapter 8 — Zombie Processes

Concept: A zombie has exited but its parent hasn't read its exit status.

Lab: Detect Zombies

```
ps -eo pid,ppid,stat,cmd | awk '$3 ~ /Z/ {print}'
```

Fix: Restart the parent or ensure parent calls `wait()` (app bug).

■ **Interview:** "Zombies are not consuming CPU; they indicate parent mismanagement; the kernel holds a process table entry until `wait()`."

Chapter 9 — cgroups (Resource Limits)

Concept: Control CPU/memory/IO per process or service. Systemd exposes cgroup controls.

Lab: Limit Memory for a Scope (systemd)

```
# Run a command in its own cgroup limited to 200MB RAM
sudo systemd-run --scope -p MemoryMax=200M --unit memscope-200m stress --vm 1 --
vm-bytes 300M --timeout 20
# If 'stress' missing: sudo apt -y install stress || sudo dnf -y install stress
```


✓ **Quick Check:** What happens when a process exceeds `MemoryMax`?

Chapter 10 — Linux Namespaces (Isolation)

Concept: Namespaces isolate views of resources (PID, NET, MNT, UTS, IPC, USER). Containers rely on them.

Lab: Create a Private Network Namespace

```
# Create a shell with a new network namespace
sudo unshare -n bash -c 'ip link; ping -c1 8.8.8.8 || echo "no network in new ns"'
```

 **Interview:** “Docker isolation combines namespaces + cgroups; overlay filesystems provide layered images.”

Chapter 11 — Sockets & Ports

Concept: A port is a numbered door on an IP address. TCP is reliable; UDP is faster but best-effort.

Lab: List Listening Ports & Owing Processes


```
sudo ss -lntp | head
# or
sudo lsof -i -P -n | head
```

Chapter 12 — Protocols (SSH, SCP, SFTP, FTP)

Lab: Key-based SSH & File Copy

```
# On client (vm-linux2)
ssh-keygen -t ed25519 -C "devops@lab" -N '' -f ~/.ssh/id_ed25519
ssh-copy-id devops@<server-ip>
ssh devops@<server-ip>

# Copy files
scp /etc/hosts devops@<server-ip>:/tmp/
# SFTP interactive
sftp devops@<server-ip>
```

 **Pitfall:** Prefer **SFTP/SCP over FTP**; FTP is plaintext unless wrapped with TLS (FTPS).

Chapter 13 — Shell (bash/zsh) & Environment

- Profile files: `/etc/profile`, `~/.bashrc`, `~/.profile`.
- Useful shortcuts: Ctrl-R (reverse search), `!!` (last command), `!$` (last arg).

Lab: Create a Handy Alias & Prompt

```
echo "alias ll='ls -alF'" >> ~/.bashrc
source ~/.bashrc
```

Chapter 14 — Command Line Mastery

Essential toolbox: `ls`, `cd`, `pwd`, `cat`, `less`, `tail -f`, `head`, `grep`, `sed`, `awk`, `cut`, `sort`, `uniq`, `tr`, `wc`, `xargs`, `find`, `du`, `df`, `tar`, `zip/unzip`.

Lab: Find Big Files under /var

```
sudo du -ah /var | sort -hr | head -n 20
```

Chapter 15 — Editors: Vim & Nano

Vim basics: `i` insert, `Esc` normal, `:w` write, `:q` quit, `:wq` save+quit, `dd` delete line, `yy` yank copy, `p` paste.

Lab: Edit a Config

```
sudo nano /etc/motd
# or
sudo vim /etc/motd
```

Part II — Storage

Chapter 16 — Filesystem Hierarchy (FHS)

- `/etc` configs, `/var` variable data/logs, `/home` users, `/opt` add-on packages, `/tmp` temp files, `/usr` user programs.

Lab: Explore Top Level

```
tree -L 1 /
```

Chapter 17 — Filesystems & Inodes

Inode: metadata (owner, perms, timestamps, blocks) — not the filename.

Lab: Show Inodes & Types

```
ls -li /etc | head
stat /etc/passwd
```

Chapter 18 — Volumes & Partitions

Tools: `lsblk`, `blkid`, `fdisk/parted`, `mkfs`, `mount`.

Lab: Create & Mount a New Partition (VM Disk)

```
# 1) Add a new virtual disk (e.g., /dev/sdb) in your hypervisor first.
lsblk
sudo fdisk /dev/sdb    # n=new, w=write (accept defaults for a demo)
sudo mkfs.ext4 /dev/sdb1
sudo mkdir -p /mnt/data
sudo mount /dev/sdb1 /mnt/data
# Persist across reboots:
echo '/dev/sdb1 /mnt/data ext4 defaults 0 2' | sudo tee -a /etc/fstab
```

! Pitfall: Wrong `/etc/fstab` entries can break boot. Keep a rescue ISO handy.

Chapter 19 — LVM (Logical Volume Management)

Flow: PV (disk) → VG (pool) → LV (slice) → filesystem.

Lab: Create & Extend an LV

```
# Install tools
sudo apt -y install lvm2 || sudo dnf -y install lvm2

# Assume /dev/sdb is free
sudo pvcreate /dev/sdb
sudo vgcreate vgdata /dev/sdb
sudo lvcreate -L 2G -n lvlogs vgdata
sudo mkfs.xfs /dev/vgdata/lvlogs    # or ext4
```



```
sudo mkdir -p /var/log/extra
sudo mount /dev/vgdata/lvlogs /var/log/extra

# Extend by 1G
sudo lvextend -L +1G /dev/vgdata/lvlogs
# Grow filesystem (XFS):
sudo xfs_growfs /var/log/extra
# For ext4: sudo resize2fs /dev/vgdata/lvlogs
```

Chapter 20 — NFS (Network File System)

Lab: Share a Folder via NFS (Server: vm-linux1, Client: vm-linux2)

```
# Server
sudo apt -y install nfs-kernel-server || sudo dnf -y install nfs-utils
sudo mkdir -p /srv/share
sudo chown nobody:nogroup /srv/share
echo '/srv/share *(rw,sync,no_subtree_check)' | sudo tee -a /etc/exports
sudo exportfs -ra
sudo systemctl restart nfs-server || sudo systemctl restart nfs-kernel-server

# Client
sudo apt -y install nfs-common || sudo dnf -y install nfs-utils
sudo mkdir -p /mnt/nfsshare
sudo mount <server-ip>:/srv/share /mnt/nfsshare
```

 **Quick Check:** How is NFS different from SCP?

Chapter 21 — Backup & Restore

Tools: `tar`, `rsync`, snapshots (LVM/ZFS), cloud storage.

Lab: Compressed Backup of /etc & Restore

```
# Backup
sudo tar -czf /tmp/etc-backup-$(date +%F).tar.gz /etc

# Restore (extract to /tmp/restore)
mkdir -p /tmp/restore
sudo tar -xzf /tmp/etc-backup-*.tar.gz -C /tmp/restore
```

Part III — System Management & Automation

Chapter 22 — Users & Groups

Lab: Create Users, Groups, and Set Ownership

```
sudo groupadd webops
sudo useradd -m -s /bin/bash -G webops alice
sudo passwd alice
sudo mkdir -p /opt/app && sudo chown alice:webops /opt/app
sudo chmod 770 /opt/app
```

Chapter 23 — SSH Management

Lab: Key-Only Login & Disable Passwords (Server)

```
# Generate key on client and copy (see Ch.12)
# Harden sshd
sudo cp /etc/ssh/sshd_config{,.bak}
echo 'PasswordAuthentication no' | sudo tee -a /etc/ssh/sshd_config
sudo systemctl restart sshd || sudo systemctl restart ssh
```

! Pitfall: Always keep one active session while testing SSH changes to avoid lockout.

Chapter 24 — Package Management (apt/dnf)

```
# Debian/Ubuntu
sudo apt update && sudo apt -y upgrade
sudo apt search nginx
sudo apt -y install nginx

# RHEL/Rocky
sudo dnf check-update || true
sudo dnf -y install nginx
```

Chapter 25 — Systemd Management

Unit types: service, socket, timer, target.

Lab: Create a Simple Service + Timer

```
# Service
sudo tee /etc/systemd/system/hello.service >/dev/null <<'EOF'
[Unit]
Description=Hello Logger

[Service]
Type=simple
ExecStart=/usr/bin/bash -c 'echo "$(date) hello" >> /var/log/hello.log; sleep 10'
EOF

# Timer (runs every minute)
sudo tee /etc/systemd/system/hello.timer >/dev/null <<'EOF'
[Unit]
Description=Run Hello Logger every minute

[Timer]
OnCalendar=*:0/1
Persistent=true

[Install]
WantedBy=timers.target
EOF

sudo systemctl daemon-reload
sudo systemctl enable --now hello.timer
systemctl list-timers --all | grep hello
```

Chapter 26 — Cronjobs

```
crontab -e      # add:  */5 * * * * /usr/bin/date >> /tmp/cronproof.log
```

 **Quick Check:** Difference between **systemd timers** and **cron**?

Chapter 27 — Shell Scripting (Automation)

Lab: Disk Usage Alert Script

```
cat > ~/disk_alert.sh <<'EOF'
#!/usr/bin/env bash
THRESH=80
for M in $(df -hP | awk 'NR>1 {print $6}'); do
    USE=$(df -hP "$M" | awk 'NR==2 {gsub("%", "", $5); print $5}')
    if [ "$USE" -ge "$THRESH" ]; then
        echo "[$(date)] WARNING: $M at ${USE}%" | tee -a ~/disk_alert.log
    fi
done
EOF
chmod +x ~/disk_alert.sh
# Test run
~/disk_alert.sh
# Schedule via cron or systemd timer
```

Chapter 28 — Process Management (Advanced)

- Foreground/background: `&`, `jobs`, `fg`, `bg`.
- Priorities: `nice`, `renice`.
- File descriptors: `lsof`.

Lab: Who Owns Port 80?

```
sudo ss -lntp | grep ':80 '
# or
sudo lsof -i :80
```

Part IV — Logging & Troubleshooting

Chapter 29 — Syslog & Rsyslog

Lab: Send a Test Syslog Message

```
logger -p user.info "hello from $(hostname)"
sudo tail -n 3 /var/log/syslog # Debian/Ubuntu
# RHEL/Rocky: check /var/log/messages
```

Chapter 30 — journalctl (Systemd Journal)

```
# Recent errors
sudo journalctl -p err -n 50 --no-pager
# Filter by unit
sudo journalctl -u ssh --since "-1h" --no-pager
# Follow live
sudo journalctl -f
```

Chapter 31 — Log Rotation (logrotate)

Lab: Custom Rotation Rule

```
sudo tee /etc/logrotate.d/hello.log >/dev/null <<'EOF'
/var/log/hello.log {
    size 10k
    rotate 5
    compress
    missingok
    copytruncate
}
EOF
sudo logrotate -d /etc/logrotate.conf # dry run
```

Chapter 32 — Resource Troubleshooting

CPU/Memory/IO Toolkit

- CPU/mem: `top`, `htop`, `vmstat 1`, `free -h`
- IO: `iostat -xz 1` (install `sysstat`)
- Open files/sockets: `lsof`

Lab: Simulate Load & Investigate

```
sudo apt -y install stress || sudo dnf -y install stress
stress --cpu 2 --vm 1 --vm-bytes 256M --timeout 20
vmstat 1 5
```

Part V — Security

Chapter 33 — File Permissions & ACLs

```
# rwx for user/group/others
ls -l /etc/passwd
chmod 640 /opt/app/secret.txt
chown alice:webops /opt/app/secret.txt

# ACLs (fine-grained)
sudo apt -y install acl || sudo dnf -y install acl
sudo setfacl -m u:bob:r /opt/app/secret.txt
getfacl /opt/app/secret.txt
```


Chapter 34 — Firewalls (UFW, iptables/nftables)

UFW (Ubuntu)

```
sudo ufw allow OpenSSH
sudo ufw allow 80/tcp
sudo ufw enable
sudo ufw status verbose
```

iptables (generic; note many distros default to nftables backend)

```
# List
sudo iptables -L -n -v
# Allow SSH
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
# Drop all else (demo; be careful on remote hosts!)
sudo iptables -P INPUT DROP
```


 **Pitfall:** Remote lockouts. Keep a console session open while testing.

Chapter 35 — SELinux (RHEL-family) & AppArmor (Ubuntu)

Check & Mode

```
# SELinux
sestatus || getenforce
# Set permissive (temporary)
sudo setenforce 0

# AppArmor status (Ubuntu)
sudo aa-status
```

 **Interview:** “SELinux enforces label-based policy; AppArmor uses path-based profiles. Start in permissive mode to learn, then enforce.”

Chapter 36 — seccomp (Syscall Filtering)

Concept: Limit which system calls a process may invoke.

Lab: Run a Container with a Strict Profile (Docker)

```
# If Docker installed:
docker run --rm --security-opt no-new-privileges --pids-limit=100 alpine sh -c
'echo ok && sleep 1'
```

Chapter 37 — auditd (Who did what?)

Lab: Watch Changes to /etc/passwd

```
sudo apt -y install auditd || sudo dnf -y install auditd
sudo systemctl enable --now auditd
sudo auditctl -w /etc/passwd -p wa -k passwd_changes
# Make a harmless read
sudo tail /etc/passwd > /dev/null
sudo ausearch -k passwd_changes | tail
```

Chapter 38 — System Hardening Checklist

- Disable password SSH; use keys.
- Minimal packages; remove unused services.
- Firewall default-deny, allow only needed ports.
- Auto security updates (`unattended-upgrades` / `dnf-automatic`).
- Enforce sudo, no direct root SSH.
- Centralized logs and time sync (chrony/systemd-timesyncd).
- Regular backups & restore testing.

Part VI — Networking

Chapter 39 — eBPF (Observability in Kernel)

Concept: Safely run tiny programs in kernel for tracing/monitoring.

Lab: Use bcc-tools (if available)

```
sudo apt -y install bpfcc-tools || sudo dnf -y install bcc-tools
sudo execsnoop -n 5 # trace new process execs (Ctrl+C to stop)
```

Chapter 40 — iptables Recipes

```
# 1) Allow inbound SSH/HTTP, drop others
sudo iptables -F
sudo iptables -P INPUT DROP
sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
```



```
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT

# 2) NAT (source NAT) for outbound from 192.168.56.0/24 via eth0
sudo iptables -t nat -A POSTROUTING -s 192.168.56.0/24 -o eth0 -j MASQUERADE
```

Chapter 41 — hosts File & Basic DNS

```
# Map a name to an IP locally (testing only)
echo '10.10.10.10 internal-api.local' | sudo tee -a /etc/hosts
getent hosts internal-api.local
```

Chapter 42 — Proxy Configurations (CLI & System)

```
# Temporary for one session
export http_proxy=http://proxy.example:3128
export https_proxy=$http_proxy
export no_proxy=127.0.0.1,localhost,.example.local

# apt (Ubuntu)
echo 'Acquire::http::Proxy "http://proxy.example:3128";' | sudo tee /etc/apt/
apt.conf.d/01proxy

# git
git config --global http.proxy http://proxy.example:3128

# Docker daemon (systemd drop-in)
sudo mkdir -p /etc/systemd/system/docker.service.d
sudo tee /etc/systemd/system/docker.service.d/proxy.conf >/dev/null <<'EOF'
[Service]
Environment="HTTP_PROXY=http://proxy.example:3128"
Environment="HTTPS_PROXY=http://proxy.example:3128"
Environment="NO_PROXY=localhost,127.0.0.1"
EOF
sudo systemctl daemon-reload && sudo systemctl restart docker
```

Part VII — Appendices & Accelerators

Appendix A — 15-Day Crash Plan (Timetable)

Daily Structure: Morning (theory + cheatsheet), Evening (lab), Night (review + notes).

Days 1–3: Linux essentials (top 50 cmds, processes/signals) → Labs: process control, `kill`, `trap`.

Days 4–5: Storage & users → Labs: `lsblk`, `df`, LVM mini, SSH keys.

Days 6–7: Packages, services, cron → Labs: Nginx install, systemd unit, cron script.

Days 8–9: Logs & troubleshooting → Labs: `journalctl`, `vmstat`, stress test.

Days 10–11: Networking → Labs: SSH/SCP, `ss`, ufw/iptables basics.

Days 12–13: Security → Labs: perms/ACL, SELinux/AppArmor, auditd.

Days 14–15: Docker + CI quick demo → Labs: Dockerfile build/push, GH Actions sample.

Appendix B — Interview Checkpoints (Sample Q → Talking Points)

- **Boot process?** Firmware → GRUB → kernel+initramfs → `systemd` → targets. Mention `journalctl -b`.
- **Zombie process?** Exited child awaiting `wait()`; detected via `ps STAT=Z`.
- **Inode?** Metadata holder; names map to inode; hard link shares inode.
- **cgroups & namespaces?** Limits + isolation; containers rely on both.
- **ext4 vs XFS?** ext4 general-purpose; XFS great for large files/concurrency; XFS must use `xfs_growfs` to expand.
- **SELinux vs AppArmor?** Label vs path-based.
- **TCP vs UDP?** Reliability vs speed; examples: SSH (TCP), DNS (UDP), with exceptions.

Appendix C — Portfolio Projects (Step-by-Step)

Project 1 — Linux Health Monitor (Bash)

Outcome: Script logs CPU/mem/disk thresholds and rotates logs.

```
mkdir -p ~/projects/health && cd ~/projects/health
cat > health.sh <<'EOF'
#!/usr/bin/env bash
set -euo pipefail
CPU_LIM=80 MEM_LIM=80 DISK_LIM=80
LOG=${1:-health.log}
while true; do
    CPU=$(top -bn1 | awk '/Cpu\(s\)/{print 100-$8}')
    MEM=$(free | awk '/Mem/{printf("%.0f", $3/$2*100)}')
    DSK=$(df -hP / | awk 'NR==2{gsub("%", "", $5);print $5}')

```

```

    TS=$(date +%F_%T)
    echo "$TS cpu=$CPU mem=$MEM disk=$DSK" | tee -a "$LOG"
    sleep 30
done
EOF
chmod +x health.sh

```

Add `logrotate` rule (see Chapter 31). Push to GitHub with README.

Project 2 — Web Server with Firewall

- Install Nginx (Ch.24).
- Allow only 22/80 (Ch.34).
- Serve custom `/var/www/html/index.html`.
- Document steps & verification (`curl -I http://<ip>`).

Project 3 — Backup & Restore (rsync + cron)

```

sudo apt -y install rsync || sudo dnf -y install rsync
mkdir -p ~/backups
cat > ~/backup_etc.sh <<'EOF'
#!/usr/bin/env bash
set -e
DEST=~/.backups/etc_$(date +%F)
mkdir -p "$DEST"
sudo rsync -a /etc/ "$DEST"/
EOF
chmod +x ~/backup_etc.sh
(crontab -l; echo '0 2 * * * ~/backup_etc.sh') | crontab -

```

Project 4 — Dockerized App + Compose

```

mkdir -p ~/projects/hello-docker && cd ~/projects/hello-docker
cat > app.sh <<'EOF'
#!/usr/bin/env bash
echo "Hello from $(hostname) on $(date)"
EOF
chmod +x app.sh
cat > Dockerfile <<'EOF'
FROM alpine:3.20
COPY app.sh /usr/local/bin/app
RUN chmod +x /usr/local/bin/app
CMD ["/usr/local/bin/app"]
EOF
cat > docker-compose.yml <<'EOF'

```

```

services:
  hello:
    build: .
    container_name: hello
    restart: unless-stopped
EOF
# docker compose up --build -d

```

Project 5 — CI/CD with GitHub Actions (build + push)

- Create Docker Hub repo.
- Add `DOCKERHUB_USERNAME` / `DOCKERHUB_TOKEN` secrets in GitHub.
- `.github/workflows/build.yml`:

```

name: build-push
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: docker/setup-buildx-action@v3
      - uses: docker/login-action@v3
        with:
          username: ${ secrets.DOCKERHUB_USERNAME }
          password: ${ secrets.DOCKERHUB_TOKEN }
      - uses: docker/build-push-action@v6
        with:
          context: .
          push: true
          tags: ${ secrets.DOCKERHUB_USERNAME }/hello:latest

```

Project 6 — Kubernetes Mini (minikube)

- Install minikube + kubectl.
- `deployment.yaml` and `service.yaml` for the `hello` image.
- Verify with `kubectl get pods,svc` and port-forward.

Appendix D — Cheatsheets

- **Processes:** `ps aux`, `top`, `htop`, `kill -15/-9`, `nice/renice`, `lsof`
- **Storage:** `lsblk`, `fdisk`, `mkfs.ext4`, `mount`, `/etc/fstab`, LVM flow
- **Logs:** `journalctl -xe`, `/var/log/*`, `logger`, `logrotate`
- **Network:** `ip a`, `ip r`, `ss -lntp`, `ping`, `traceroute`, `dig`, `curl -v`

• **Security:** `chmod/chown`, `setfacl`, `ufw`, `iptables`, `sestatus` / `aa-status`, `auditctl`

Final Words

Practice daily. Break servers (in a lab), then fix them. Keep notes. Push projects to GitHub. Tie every Linux concept to a DevOps outcome (containers, orchestration, CI/CD). That's how you become a polished DevOps engineer.