# DevOps Shack

# Linux Best Practices

## 1. Security Best Practices

Security is crucial for protecting Linux systems from unauthorized access, malware, and vulnerabilities.

### 1.1 User and Access Management

- Use Least Privilege Principle: Create users with only the necessary privileges. Avoid using `root` for daily tasks.
- Disable Root Login: Prevent direct root S login by setting `PermitRootLogin no` in `/etc/s/sd_config`.
- Implement sudo: Grant admin privileges via `sudo` instead of using `su`. Use `/etc/sudoers` for precise control.
- Set Strong Password Policies: Configure `/etc/login.defs` to enforce password complexity rules.
- Use S Key-Based Authentication: Disable password authentication (`PasswordAuthentication no`) and use S keys.

### 1.2 Firewall and Network Security

- Use Firewalls: Configure `iptables`, `nftables`, or `ufw` (for Ubuntu) to allow only necessary traffic.
- Disable Unused Services: Stop and disable unnecessary services with `systemctl disable service-name`.
- Limit Open Ports: Use `netstat -tulnp` or `ss -tulnp` to check open ports and close unnecessary ones.
- Implement Fail2ban: Protect against brute force attacks by setting up Fail2ban to block repeated failed logins.

### 1.3 Secure File System & Permissions

- Set Correct File Permissions: Use `chmod` and `chown` to restrict access (`chmod 600` for sensitive files).
- Use `nosuid`, `noexec`, and `nodev` for Partitions:
  - Add these options in `/etc/fstab` for `/tmp`, `/var`, and `/home`.
- Encrypt Sensitive Data: Use LUKS for full disk encryption and `gpg` for encrypting individual files.
- Use SELinux or AppArmor: Enforce additional security policies for process and file access.

### 1.4 Regular Security Auditing

- Enable Auditd: Configure `auditd` to log system activities (`auditctl -w /etc/passwd -p wa -k passwd_changes`).
- Scan for Malware: Use ClamAV or `rkhunter` to scan for malware and rootkits.
- Monitor System Logs: Regularly check `/var/log/auth.log`, `/var/log/syslog`, and `/var/log/messages`.

## 2. Performance Optimization

A well-optimized Linux system runs efficiently and reduces resource consumption.

### 2.1 Kernel and System Tuning

- Adjust Kernel Parameters: Modify `/etc/sysctl.conf` for optimized memory, networking, and process handling.
- Limit Swappiness: Reduce swap usage (`sysctl vm.swappiness=10`) to prefer RAM over swap.

### 2.2 Resource Management

- Use cgroups and systemd: Restrict CPU and memory usage using cgroups.

- Limit Background Processes: Use `nice` and `ionice` to prioritize processes.
- Monitor with Tools:
  - `htop`, `top` for CPU and memory usage.
  - `iotop` for disk I/O monitoring.
  - `netstat` and `ss` for network connections.

### 2.3 Disk and Filesystem Optimization

- Use Modern Filesystems: Prefer `ext4`, `XFS`, or `btrfs` over `ext3`.
- Schedule TRIM for SSDs: Use `fstrim -av` for SSD performance.
- Check Disk Health: Use `smartctl` to monitor drive health.
- Enable Write Caching: Use `hdparm` for HDD performance tuning.

## 3. System Maintenance and Monitoring

Keeping a Linux system well-maintained ensures long-term stability.

### 3.1 Regular Updates and Patching

- Enable Automatic Updates:
  - Debian/Ubuntu: Use `unattended-upgrades`.
  - RHEL/CentOS: Use `yum-cron` or `dnf-automatic`.
- Check CVE Reports: Use `lynis audit system` or `osquery` to identify vulnerabilities.

### 3.2 Log Management

- Rotate Logs Automatically: Configure `logrotate` in `/etc/logrotate.conf`.
- Centralized Logging: Forward logs to a SIEM system like ELK, Splunk, or Graylog.
- Monitor Logs in Real-time: Use `tail -f /var/log/syslog`.

### 3.3 Backup and Disaster Recovery

- Automate Backups: Use `rsync`, `borg`, or `restic` for backups.
- Use Snapots: Utilize LVM snapots or Btrfs/ZFS snapots.
- Test Recovery Plans: Regularly restore backups to test their integrity.

## 4. Networking Best Practices

Optimizing network performance is critical for servers and production systems.

### 4.1 Network Configuration

- Use Static IPs for Servers: Avoid DHCP for production servers.
- Optimize MTU Size: Adjust MTU settings based on network requirements.
- Reduce DNS Resolution Time: Use a local DNS cache (`systemd-resolved`, `dnsmasq`).

### 4.2 Network Performance Tuning

- Enable TCP BBR Congestion Control: `sysctl net.ipv4.tcp_congestion_control=bbr`.
- Disable IPv6 (if not needed): `sysctl net.ipv6.conf.all.disable_ipv6=1`.

## 5. Software & Package Management

Proper package management reduces the risk of outdated or vulnerable software.

### 5.1 Use Official Repositories

- Avoid Third-Party Repositories: Use only trusted sources.
- Enable Security Repositories: Example: `yum install yum-plugin-security`.

### 5.2 Keep Packages Minimal

- Uninstall Unused Packages: Use `apt autoremove` or `dnf remove` for unnecessary packages.
- Use Containerization: Deploy applications in Docker, Podman, or Kubernetes to avoid system bloat.

## 6. Automation and Scripting

Automation helps streamline repetitive administrative tasks.

### 6.1 Use Configuration Management Tools

- Ansible, Puppet, Chef, or SaltStack for managing configurations.
- IaC (Infrastructure as Code): Use Terraform for provisioning.

### 6.2 ell Scripting Best Practices

- Follow ell Script Best Practices:
  - Use `#!/bin/ba -e` to stop execution on error.
  - Write clean, modular, and well-documented scripts.
  - Store sensitive data in environment variables.

## 7. Hardening Containers & Virtual Machines

Security for virtualized and containerized environments is critical.

### 7.1 Best Practices for Containers

- Use Minimal Base Images: Use Alpine or Debian Slim.
- Limit Container Privileges: Avoid running containers as root.
- Use Read-Only Filesystems: `docker run --read-only`.

### 7.2 Best Practices for VMs

- Use Thin Provisioning: Optimize disk usage.

- Disable Unnecessary Virtual Interfaces: Only enable essential network interfaces.

## 8. High Availability & Scalability

Ensuring availability and scalability is key in production environments.

### 8.1 Load Balancing

- Use HAProxy, Nginx, or Envoy for load balancing.
- Implement Failover Mechanisms: Use Pacemaker and Corosync.

### 8.2 Monitoring & Alerting

- Use Prometheus + Grafana: For real-time monitoring.
- Set Up Alerts: Use `alertmanager` to notify based on defined threolds.

## 9. Debugging & Troubleooting Best Practices

Effective debugging ensures stability and quick issue resolution.

### 9.1 System Logs & Diagnostics

- Check System Logs Efficiently:
  - Use `journalctl -xe` for detailed logs.
  - View kernel logs with `dmesg | tail -50`.

Monitor logs in real time:

```
tail -f /var/log/syslog
```

- Centralized Logging:
  - Use syslog, Fluentd, Graylog, or ELK (Elasticsearch, Logsta, Kibana) for centralized logging.

Example: Forward logs using rsyslog (`/etc/rsyslog.conf`):

```
*.* @logserver:514
```

9.2 Performance Debugging

- CPU Usage Analysis:
  - `htop` (interactive)
  - `top -o %CPU` (sort by CPU usage)
- Memory & Swap Debugging:
  - Check memory usage: `free -m`
  - Analyze swap: `swapon -s`
  - Detect memory leaks: `valgrind --leak-check=full ./app`
- Disk I/O Performance Monitoring:

Identify heavy disk activity:

```
iotop -o
```

Check file system usage:

```
df -h
```

- Network Performance Issues:

Detect high traffic usage:

```
iftop -i eth0
```

Measure network latency:

```
ping -c 10 google.com
```

9.3 Debugging Services & Processes

Check Open Ports:

```
ss -tulnp
```

**Identify Running Services:**

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

**Troubleshoot Failing Services:**

```
systemctl status apache2
journalctl -u apache2 --no-pager
```

## 10. Compliance & Audit Best Practices

**Compliance is crucial for regulatory requirements (HIPAA, PCI-DSS, ISO 27001).**

10.1 File Integrity Monitoring (FIM)

Use AIDE (Advanced Intrusion Detection Environment):

```
apt install aide
aideinit
mv /var/lib/aide/aide.db.new /var/lib/aide/aide.db
```

Run periodic checks:

```
aide --check
```

Audit File Access:

```
auditctl -w /etc/passwd -p wa -k passwd_changes
```

## 10.2 User & Authentication Compliance

**Enforce Strong Password Policies: Edit `/etc/security/pwquality.conf`:**

```
minlen = 12
dcredit = -1
ucredit = -1
ocredit = -1
lcredit = -1
```

**Set Automatic Account Lockout for Failed Logins:**

```
auth required pam_tally2.so deny=5 unlock_time=600
```

## 10.3 System Auditing

**Use Lynis for security auditing:**

```
apt install lynis
lynis audit system
```

- **Automate Compliance Checks:**

**OpenSCAP for PCI-DSS, CIS benchmarks:**

```
oscap xccdf eval --profile

xccdf_org.ssgproject.content_profile_pci-dss
/usr/are/xml/scap/ssg/content/ssg-ubuntu1804-ds.xml
```

## 11. Cloud & Virtualization Best Practices

Cloud environments require additional security and performance optimizations.

### 11.1 Secure Cloud Instances

Disable password authentication & use key-based login:

```
sed -i 's/#PasswordAuthentication yes/PasswordAuthentication
no/' /etc/s/sd_config
systemctl restart sd
```

- Enable Cloud Security Tools:
  - AWS: AWS Config, GuardDuty
  - Azure: Azure Security Center
  - GCP: Google Cloud Security Command Center

### 11.2 Optimize Cloud Resources

- Use Auto-Scaling Groups: Automatically scale instances as needed.
- Implement Spot Instances & Reserved Instances: Cost-saving strategies.
- Enable Instance Metadata Service (IMDSv2) for enhanced security.

## 12. DevOps & CI/CD Best Practices

For modern DevOps pipelines, Linux plays a key role in automation and deployment.

## 12.1 Secure CI/CD Pipelines

- Limit Privileges in Pipelines: Run builds with the lowest necessary permissions.
- Use Secrets Management: Store credentials in Vault, AWS Secrets Manager, or Kubernetes Secrets.

Scan Containers for Vulnerabilities:

```
trivy image my-app:latest
```

## 12.2 Infrastructure as Code (IaC) Best Practices

Use Terraform or Ansible for provisioning:

```
terraform init
terraform apply
```

- Enforce Security Policies: Use `tflint` for Terraform linting.

## 12.3 Kubernetes & Container Best Practices

Limit Container Privileges:
yaml

```
securityContext:
  allowPrivilegeEscalation: false
  runAsNonRoot: true
```

**Use Read-Only Filesystems in Containers:**

```
docker run --read-only my-secure-app
```

- **Restrict Network Access with Kubernetes Network Policies.**

## 13. Linux Hardening Best Practices

**System hardening protects against exploits and unauthorized access.**

### 13.1 Remove Unused Software & Services

**List Installed Packages:**

```
dpkg --list | less
```

**Remove Unused Packages:**

```
apt autoremove -y
```

### 13.2 Restrict Kernel Modules

**Blacklist Dangerous Modules:**

```
echo "blacklist usb-storage" >>
/etc/modprobe.d/blacklist.conf
```

### 13.3 Restrict USB Devices & External Media

**Disable USB Storage:**

```
echo "install usb-storage /bin/true" >
/etc/modprobe.d/usb-storage.conf
```

## 14. Disaster Recovery & Backup Strategies

**A robust backup and recovery strategy is crucial for business continuity.**

### 14.1 Backup Automation

**Use Rsync for Incremental Backups:**

```
rsync -av --delete /var/www /backup/
```

**Schedule Backups with Cron:**

```
0 2 * * * rsync -av --delete /var/www /backup/
```

### 14.2 Disaster Recovery Plan

- **Maintain Cold & Hot Standby Servers: Replicate production environments for quick failover.**
- **Perform Regular DR Drills: Test recovery procedures frequently.**