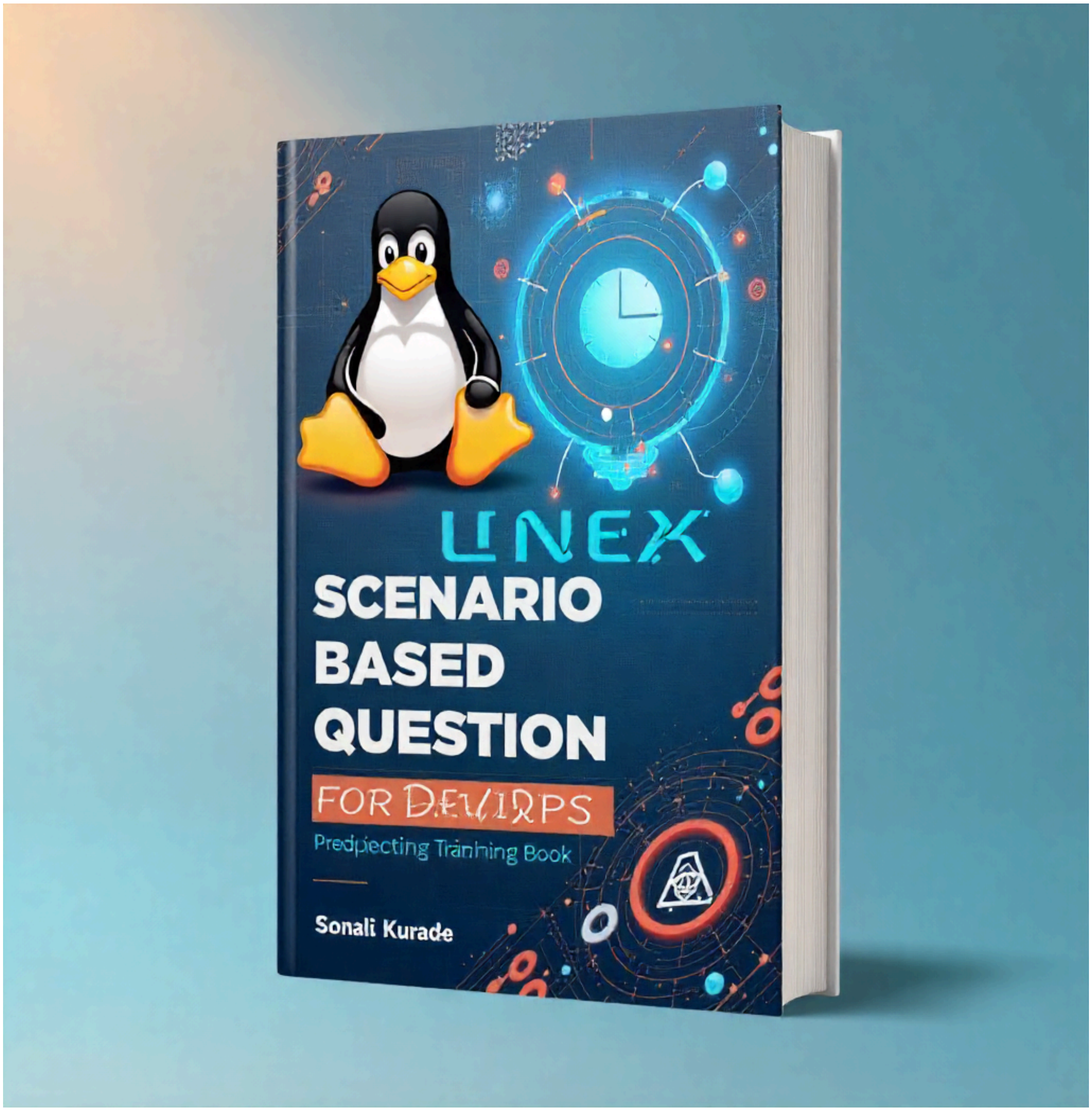
LINEX
SCENARIO
BASED
QUESTION
FOR DEVIRPS
Predpiecting Tranhing Book
Sonali Kurade

Real-Time Linux + DevOps Scenarios

🔹 Scenario 1: Debugging a Failed Deployment (Linux Architecture + File System + File Commands)

👉 **Situation:**
You deployed a new microservice on an application server, but the application is failing to start. You need to quickly debug.

👉 **Steps (Concept Mapping):**

1. **Kernel & User Space**
   - You run `dmesg | tail -20` to check if the kernel is throwing memory or process-related errors.
   - You also verify process states with `ps -ef` (running in user space).
2. **File System Hierarchy**
   - Check **configuration files** in `/etc/myapp/config.yaml`.
   - Review **logs** in `/var/log/myapp/error.log`.
   - Look at **temporary files** in `/tmp` which may cause conflicts if left behind.
3. **File/Directory Commands**
   - Use `cd /etc/myapp/` and `ls -l` to verify the config file exists.
   - Copy a backup config using `cp config.yaml config.yaml.bak`.
   - Edit the file with `vi config.yaml`.

4. **File Viewing & Editing**
    - Run `tail -f /var/log/myapp/error.log` to watch real-time logs while restarting the service.

👉 **Outcome:**
You found a wrong database path in `/etc/myapp/config.yaml`, fixed it, restarted the service, and the application came up successfully.

🔹 Scenario 2: Optimizing Build Pipeline (Links + File Commands + File System)

👉 **Situation:**
Your CI/CD pipeline in Jenkins is failing because the build agent is running out of disk space while compiling a large application.

👉 **Steps (Concept Mapping):**

1. **File System Hierarchy**
    - Build artifacts are stored in `/var/lib/jenkins/workspace/`.
    - Older builds were archived in `/opt/builds/`.
2. **File/Directory Commands**
    - Check space usage with `du -sh /var/lib/jenkins/workspace/*`.
    - Move older logs/artifacts to another partition with `mv`.

3. **Links (Hard vs Soft Links)**

   - Instead of duplicating large artifacts, create **soft links** from `/var/lib/jenkins/workspace/build-latest` → `/opt/builds/build-123`.

   - This saves space and ensures Jenkins always points to the latest successful build.

4. **File Viewing & Editing**

   - Update the Jenkins build script using `nano build.sh` to point to the symlink instead of the full build path.

👉 **Outcome:**
Disk space was optimized using symbolic links, and the Jenkins pipeline successfully completed without failing due to low storage.

---

1. Linux Architecture (Kernel, Shell, User space)

Scenario 1:
A Kubernetes node crashes frequently. You run `dmesg` and see **OOM (Out of Memory)** errors from the **kernel**. You decide to set pod resource limits in YAML and monitor with `top` from user space.

Scenario 2:
During an automated deployment, a Jenkins job hangs. You use the **shell** to run `ps -ef` and kill the stuck process with `kill -9 <PID>`. This clears the issue and the pipeline proceeds.

---

2. File System Hierarchy (/etc, /var, /home, /opt, /tmp, /proc)

Scenario 3:
An Nginx web server deployed in a VM fails to start. You check `/etc/nginx/nginx.conf` for syntax errors, then verify error logs in `/var/log/nginx/error.log`.

Scenario 4:
A build agent is consuming too much `/tmp` space. You clean stale files with `rm -rf /tmp/*` and restart the agent, freeing space for new builds.

Scenario 5:
Your CI/CD tool is installed in `/opt/jenkins`. You update its configuration in `/etc/systemd/system/jenkins.service` and reload the daemon with `systemctl daemon-reload`.

Scenario 6:
To check CPU usage issues in production, you read `/proc/cpuinfo` and `/proc/meminfo`. Based on this, you scale pods vertically in Kubernetes.

---

3. File/Directory Commands (ls, cd, pwd, mkdir, touch, rm, cp, mv, find, locate)

Scenario 7:
A developer asks for access to logs of their app. You `cd /var/log/myapp/` and use `ls -lrt` to find the latest logs.

Scenario 8:
You need to create a new workspace for a project. You run `mkdir /home/devops/projectX && cd /home/devops/projectX`.

Scenario 9:
While troubleshooting, you must quickly find a missing `config.yaml`. You use `find / -name config.yaml 2>/dev/null` to locate it.

Scenario 10:
An application build needs a backup before upgrading. You `cp -r /opt/myapp /opt/myapp_bkp` before applying the patch.

Scenario 11:
You want to archive old deployment artifacts. Move them using `mv build-2024.tar.gz /opt/archives/`.

4. File Viewing & Editing (cat, less, head, tail -f, nano, vi)

Scenario 12:
Your microservice crashes after deployment. You use `tail -f /var/log/myservice/error.log` to monitor real-time logs while restarting.

Scenario 13:
You need to check only the first 20 lines of a Kubernetes manifest file. Use `head -20 deployment.yaml`.

Scenario 14:
While investigating a huge log file, you use `less +G /var/log/syslog` to jump to the end without crashing the terminal.

Scenario 15:
A pipeline script has a typo. You open it in `vi build.sh`, correct it, save, and rerun the pipeline.

Scenario 16:
A team member forgot to update credentials. You quickly edit `nano /etc/myapp/credentials.env` to add the right values.

5. Links (Hard link vs Soft link)

Scenario 17:
Your build server generates large artifacts in `/var/lib/jenkins/workspace/`. To save space, you create a **soft link** (`ln -s /opt/builds/latest /var/lib/jenkins/workspace/latest`) pointing to the latest artifact.

Scenario 18:
A monitoring tool requires log access from multiple locations. Instead of copying logs, you create a **hard link** from `/var/log/app1.log` to `/var/monitor/app1.log`.

Scenario 19:
You migrate a config file to `/opt/configs/`, but old scripts still expect it in `/etc/app/config.yaml`. You solve it with a soft link:
`ln -s /opt/configs/config.yaml /etc/app/config.yaml`.

Scenario 20:
A Jenkins job deletes an artifact accidentally. Since a **hard link** existed in `/opt/artifacts/`, the file was preserved and pipeline recovery was possible.

Perfect 🙌 Let's build the **50 real-time Linux + DevOps scenarios** set.
I'll keep the **20 you already have** (architecture, filesystem, commands, editing, links) and expand with **more categories**:

- Permissions & Ownership
- Processes & Services
- Networking & Firewalls

- Package Management
- Scripting & Automation
- Monitoring & Performance
- Troubleshooting

🔥 50 Real-Time Linux + DevOps Scenarios

✅ Already Covered (20 Scenarios Recap)

- Linux Architecture (2)
- File System Hierarchy (4)
- File/Directory Commands (5)
- File Viewing & Editing (5)
- Links (4)

🔹 Permissions & Ownership (6 Scenarios)

Scenario 21:
A deployment script fails because it doesn't have permission to write logs in `/var/log/app/`. You fix it with:

```
chown jenkins:jenkins /var/log/app/

chmod 755 /var/log/app/
```

Scenario 22:
A developer accidentally commits sensitive credentials. You secure the file with:

```
chmod 600 secrets.env
```

Scenario 23:
You onboard a new DevOps engineer. Create a user and give sudo privileges:

```
useradd devops1

passwd devops1

usermod -aG sudo devops1
```

Scenario 24:
A Docker build fails due to permission issues on mounted volumes. You run:

```
ls -ld /data/volume

chown 1000:1000 /data/volume
```

Scenario 25:
You need to share logs with a QA team but prevent modifications. Set directory as **read-only**:

```
chmod -R 444 /var/log/app/
```

Scenario 26:
In a shared server, you restrict one user from accessing another's files by checking `/home/*` permissions.

🔹 Processes & Services (5 Scenarios)

Scenario 27:
An application keeps crashing. You run `ps aux | grep app` to check the running process and restart with `systemctl restart app.service`.

Scenario 28:
Jenkins service stops after reboot. You enable auto-start:

```
systemctl enable jenkins
```

Scenario 29:
A CPU spike is observed. You use `top` to identify a rogue process and kill it.

Scenario 30:
To debug memory leaks, you run `htop` and sort by memory usage.

Scenario 31:
You need to stop all Python processes for a failed pipeline run:

```
pkill -f python
```

🔹 Networking & Firewalls (6 Scenarios)

Scenario 32:
Pods cannot reach a VM. You test connectivity with `ping <ip>` and `curl http://<ip>:8080`.

Scenario 33:
A Jenkins webhook from GitHub fails. You open firewall port:

```
ufw allow 8080/tcp
```

Scenario 34:
DNS resolution fails in a container. You check `/etc/resolv.conf`.

Scenario 35:
SSH login is slow. You disable reverse DNS lookups by editing `/etc/ssh/sshd_config`.

Scenario 36:
App communication between two regions is blocked. You verify routing with:

```
traceroute <ip>
```

Scenario 37:
You monitor network usage with:

```
iftop
```

🔹 Package Management (5 Scenarios)

Scenario 38:
An app requires `curl` but it’s missing. You install with:

```
apt-get install curl -y
```

Scenario 39:
Your pipeline fails because Python is outdated. You upgrade:

```
yum update python3
```

Scenario 40:
A CI/CD tool requires Node.js. You add repo and install:

```
curl -fsSL https://deb.nodesource.com/setup_18.x | bash -

apt-get install -y nodejs
```

Scenario 41:
You need to check installed packages:

```
dpkg -l | grep nginx
```

Scenario 42:
Roll back a broken update by removing the latest version:

```
yum downgrade package_name
```

🔹 Scripting & Automation (5 Scenarios)

Scenario 43:
Write a script to back up `/etc/` daily into `/opt/backups/`.

Scenario 44:
Automate cleaning `/tmp` every night with a cron job.

Scenario 45:
Script to check if a service (nginx) is running; if not, restart it.

Scenario 46:
Automate Jenkins workspace cleanup older than 10 days:

```
find /var/lib/jenkins/workspace/ -mtime +10 -exec rm -rf {} \;
```

Scenario 47:
Send an email alert if disk usage > 80%.

🔹 Monitoring & Performance (5 Scenarios)

Scenario 48:
App is slow. You run `vmstat 5` to monitor CPU/IO.

Scenario 49:
Check disk performance with:

```
iostat -x 5
```

Scenario 50:
Monitor real-time logs for multiple apps using `multitail`.

Scenario 51:
Measure network latency using `ping` to the DB server.

Scenario 52:
Set up `sar` to collect CPU utilization every 5 minutes.

🔹 Troubleshooting (6 Scenarios)

Scenario 53:
A pipeline fails with "permission denied". You use `ls -l` to check file ownership.

Scenario 54:
A server is unresponsive. You check system logs:

```
journalctl -xe
```

Scenario 55:
Deployment failed because of missing library. You locate it with `ldd binary_name`.

Scenario 56:
A pod fails due to “read-only filesystem”. You check mount options in `/etc/fstab`.

Scenario 57:
SSH key authentication fails. You verify file permissions:

```
chmod 600 ~/.ssh/id_rsa

chmod 700 ~/.ssh
```

Scenario 58:
Disk full error during deployment. You find large files:

```
du -sh * | sort -h
```