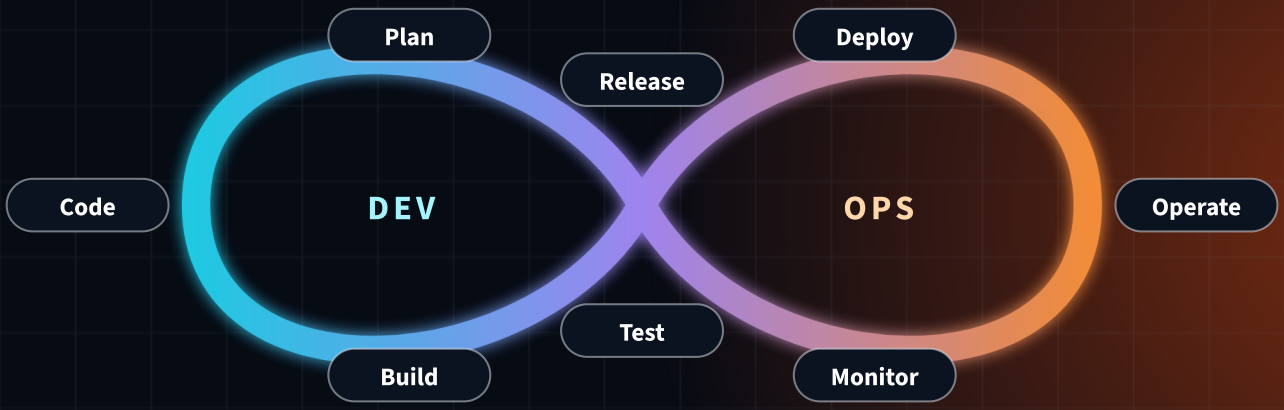
JOB-READY SERIES · JOB-SWITCH EDITION

CODE TO CLOUD

DevOps

The Job-Switch Guide · From Linux to Kubernetes

13 tools every DevOps interview asks about, in simple language: commands, real configs, hands-on labs, production incidents, portfolio projects and 120+ interview questions with answers.




```
shreyash@prod-cluster: ~

$ terraform apply -auto-approve  Apply complete! 12 added.
$ kubectl rollout status deploy/career  successfully rolled out
$ curl -s offer.yournextjob.com/status  {"status": "HIRED"}
```

Linux · Bash · Python · Networking · Git · Docker · GitHub Actions · AWS · Terraform · Ansible · Kubernetes · Prometheus · Grafana

WRITTEN BY

Shreyash Tripathi

17 Chapters · Labs · Incidents · Interview Questions

Edition 2026

JOB-READY SERIES · JOB-SWITCH EDITION



DevOps

The Job-Switch Guide, from Linux to Kubernetes

Learn the tool | Run the lab | Break it, fix it | Build the project | Ace the interview

WRITTEN BY

Shreyash Tripathi

Job-Switch Edition, 2026

© Shreyash Tripathi. All rights reserved.

Preface

Switching into DevOps feels overwhelming because the tool list never ends. The good news: interviews keep asking about the same thirteen tools, and they all connect into one story. Code goes into Git, a pipeline builds and tests it, Docker packages it, Terraform creates the cloud, Ansible configures it, Kubernetes runs it, and Prometheus and Grafana tell you when it breaks. This book walks that story from start to finish.

It is written like a school textbook on purpose: numbered chapters, simple language, a check with answers after every topic, and exercises with a full answer key. Because DevOps is learned with your hands, every chapter also has hands-on labs, real config files, a troubleshooting section, Interview Corners with the questions you will actually be asked, At Work boxes from real on-call life, and a resume line you can use once you have done the work.

Do the labs. Break things on purpose and fix them. Push every mini project to GitHub. By the last chapter you will have a portfolio, a story for every interview question, and the confidence to switch.

Shreyash Tripathi

Features of Every Chapter

🎯 **INTERVIEW CORNER**

Questions interviewers really ask, with model answers.

💼 **AT WORK**

What really happens on the job and on call.

🧪 **HANDS-ON LAB**

Do it yourself on a laptop or the AWS free tier.

⚠️ **COMMON MISTAKE**

Traps that break production and fail interviews.

💡 **DO YOU KNOW?**

Real facts from the industry.

😄 **FUNNY SIDE**

Because 3 a.m. pages need jokes.

✅ **CHECK YOUR PROGRESS**

Short questions with answers after every topic, plus exercises, scenario questions, a mini project and a full answer key at the end of every chapter.

Contents

PART I: FOUNDATIONS

1 DevOps and the Job Switch Roadmap 6
2 Linux for DevOps 24
3 Bash Scripting 41
4 Python for DevOps 60
5 Networking for DevOps 81

PART II: BUILD AND SHIP

6 Git and GitHub for DevOps 100
7 Docker 117
8 CI/CD with GitHub Actions 135

PART III: CLOUD AND INFRASTRUCTURE AS CODE

9 AWS for DevOps 159
10 Terraform 179
11 Ansible 205

PART IV: ORCHESTRATION AND OBSERVABILITY

12 Kubernetes I: Core Concepts 231
13 Kubernetes II: Production 250
14 Monitoring with Prometheus 268
15 Dashboards and Alerting with Grafana 287

PART V: GET THE JOB

16 Real-World Projects and Incident Scenarios 308
17 DevOps Interview Question Bank 327

APPENDICES

A Command Cheat Sheet 344
B Config Templates 350
C Glossary 358
D Job Switch Toolkit 362

PART I

Foundations

The DevOps mindset, Linux, Bash, Python and networking.

CHAPTER 1

DevOps and the Job Switch Roadmap

"It works on my machine." "Great. Then we will ship your machine." That is how Docker was born. Almost.

After studying this chapter, you will be able to

- Explain what DevOps really is (a culture, not a job title) using the CALMS model and the four DORA metrics.
- Place DevOps inside the software development life cycle and tell CI, continuous delivery and continuous deployment apart.
- Compare DevOps, SRE, Platform and Cloud engineer roles and know what each one interviews for.
- Draw the full toolchain map of this book, from a code commit to a Grafana dashboard.
- Follow a 90-day plan to switch jobs, with three portfolio projects and a resume that gets shortlisted.
- Prepare for the typical interview rounds of a DevOps hiring process.

Think of a big Indian wedding. The caterers cook, the decorators decorate, the band plays, and the family runs around. If the caterers finish the food at 9 PM but the hall is booked only from 10 PM, the food goes cold. If the band arrives before the stage is ready, they stand outside sweating. Every team did its own job well, but the wedding still went badly. What was missing? A shared plan, shared timing and somebody watching the whole show.

Software teams had the same problem for decades. Developers wrote code and "threw it over the wall" to operations. Operations got a zip file, a half-page README and a deadline. When things broke at 2 AM, each side blamed the other. DevOps is the idea that the people who build software and the people who run it should work as one team, with shared goals, shared tools and lots of automation. This chapter explains that idea, the job roles around it and a concrete plan to switch into one of them.

1.1 What DevOps Is (and Is Not)

DevOps is a set of practices and a culture that shortens the time from "a developer wrote a line of code" to "a customer is using it safely in production". It joins **Dev**elopment and **Op**erations. The

term became popular around 2009, after Patrick Debois organised the first DevOpsDays conference in Ghent, Belgium.

DevOps is **not** a single tool. Installing Jenkins does not make you a DevOps company, just like buying a gym membership does not make you fit. It is also not "the ops person who now also writes YAML". It is a way of working where teams own their software from laptop to production.

1.1.1 The CALMS Model

CALMS is a simple checklist to judge whether a team is really doing DevOps. It was coined by Jez Humble, building on John Willis and Damon Edwards' earlier "CAMS".

Letter	Stands for	What it looks like in practice
C	Culture	Devs and ops share on-call, blameless postmortems, no "not my problem".
A	Automation	Builds, tests, infra and deploys run from code, not from someone's memory.
L	Lean	Small batches, short queues, remove waiting time (like waiting 3 days for a server).
M	Measurement	Track deploy frequency, failures, latency, cost. Decisions use data.
S	Sharing	Runbooks, dashboards and knowledge are open to everyone.

1.1.2 DORA Metrics

The DevOps Research and Assessment (DORA) team, now part of Google Cloud, studied thousands of organisations and found four metrics that separate high performers from low performers. Interviewers love these because they show you think about outcomes, not only tools.

Metric	Question it answers	Elite teams (roughly)
Deployment frequency	How often do we ship to production?	On demand, many times a day
Lead time for changes	Commit to production takes how long?	Less than a day
Change failure rate	What percent of deploys cause a problem?	Low, around 0 to 15 percent
Time to restore service	When it breaks, how fast do we recover?	Less than an hour

The first two measure **speed**. The last two measure **stability**. The big finding is that good teams get both. Speed and stability are not enemies; small, frequent, automated changes are both faster and safer. Recent DORA reports also discuss a fifth measure around reliability and rework; check the latest State of DevOps report for the current wording.

> 💡 **DO YOU KNOW?**
>
> The book "Accelerate" (2018) by Nicole Forsgren, Jez Humble and Gene Kim explains the research behind the DORA metrics. Quoting it in an interview makes you sound like someone who reads beyond tutorials.

🎯 **INTERVIEW CORNER**

Q: What is DevOps in one sentence? **A:** A culture and set of practices where development and operations share ownership of software, using automation and measurement to deliver changes quickly and safely.

Q: Name the DORA metrics. **A:** Deployment frequency, lead time for changes, change failure rate and time to restore service. Two measure speed, two measure stability.

Q: How would you improve lead time in a team that deploys once a month? **A:** Find where time is spent (often manual testing, approvals, environment setup), automate tests in CI, make batches smaller, use feature flags and automate deployment so releasing is a button, not a project.

✅ **CHECK YOUR PROGRESS 1.1**

1. **Is DevOps a tool? Explain.**

   Answer: No. It is a culture and set of practices. Tools like Git, Docker and Kubernetes support it, but without shared ownership and automation, tools alone do not create DevOps.

2. **Which two DORA metrics measure stability?**

   Answer: Change failure rate and time to restore service.

3. **What does the "L" in CALMS mean?**

   Answer: Lean: work in small batches and remove waiting and waste from the delivery process.

1.2 The SDLC and Where DevOps Fits

The Software Development Life Cycle (SDLC) is the journey of a feature: plan, code, build, test, release, deploy, operate and monitor. In the old waterfall model each stage happened once, in order, often over months. In agile, teams do short sprints, but deployment was often still manual and rare. DevOps closes the loop: what you learn from monitoring goes straight back into planning.



Figure 1.1: The DevOps loop. Highlighted stages are where automation gives the biggest gains.

You may have seen the "infinity loop" drawing. The left loop is Dev (plan, code, build, test). The right loop is Ops (release, deploy, operate, monitor). The point of the drawing is that there is no end; feedback flows forever.

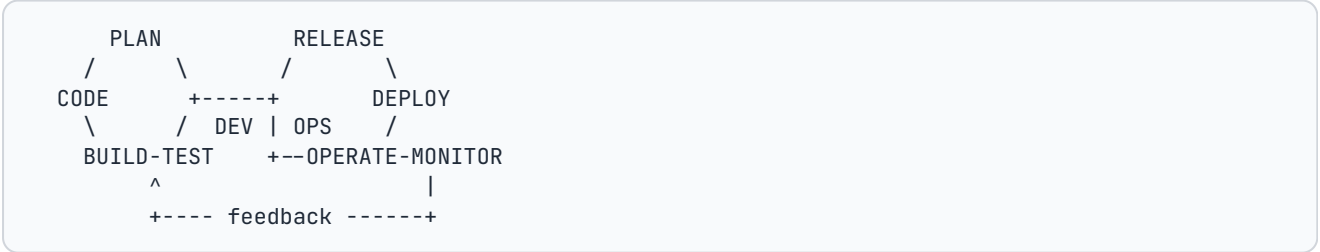


Figure 1.2: The DevOps infinity loop in simple ASCII form.

1.2.1 CI vs Continuous Delivery vs Continuous Deployment

These three terms are asked in nearly every DevOps interview. People mix them up, so learn them exactly.

Term	What happens automatically	Human step
Continuous Integration (CI)	Every push is built and tested; developers merge to main often (at least daily).	Deploy is still manual and separate.
Continuous Delivery (CD)	CI plus every passing build is packaged and ready to deploy to production at any time.	A human clicks "approve" to go to prod.
Continuous Deployment	Every change that passes all checks goes to production with no human approval.	None. Tests and monitoring are the gate.



Figure 1.3: Remove the "approve?" box and continuous delivery becomes continuous deployment.

> ⚠️ **COMMON MISTAKE**
>
> Saying "we have CI/CD" when the pipeline only runs unit tests and someone still SSHs into a server to deploy. That is CI only. Be honest in interviews; interviewers can tell, and honesty plus a plan to improve scores better than buzzwords.

> 😄 **FUNNY SIDE**
>
> Before CI, teams had "integration week": everyone merged their branches on Friday, and the build was broken until Thursday. It was like ten cousins each cooking one part of the biryani at home and mixing it in the hall. Somebody always brought the rice without salt.

> ✅ **CHECK YOUR PROGRESS 1.2**
>
> 1. **What is the one difference between continuous delivery and continuous deployment?**
>
>   **Answer:** Continuous delivery needs a human approval before production; continuous deployment releases every passing change automatically.
>
> 2. **Which SDLC stages benefit most from automation?**
>
>   **Answer:** Build, test, release and deploy, plus monitoring and alerting during operate.

1.3 DevOps vs SRE vs Platform vs Cloud Engineer

Job titles in this space overlap a lot. One company's "DevOps Engineer" is another company's "Cloud Engineer". Still, each title has a centre of gravity, and knowing it helps you target your preparation.

Role	Main focus	What they interview for
DevOps Engineer	CI/CD pipelines, containers, IaC, automation, release process	Linux, scripting, Git, Docker, a CI tool, Terraform, Kubernetes basics, one cloud, troubleshooting
Site Reliability Engineer (SRE)	Reliability of production: SLOs, error budgets, incident response, capacity	Coding (often a real DSA round), Linux internals, networking, system design, monitoring, incident stories
Platform Engineer	Building an internal developer platform: golden paths, self-service infra, Kubernetes platform	Kubernetes depth, Terraform modules, API/tool design, developer experience, Go or Python
Cloud Engineer	Designing and running cloud infra: networking, IAM, cost, migrations	AWS/Azure/GCP services, VPC design, IAM, security, cost, certifications often valued

The SRE idea came from Google, where Ben Treynor Sloss described it as "what happens when you ask a software engineer to design an operations team". SREs set a **Service Level Objective** (SLO), for example "99.9 percent of requests succeed in 30 days". The allowed 0.1 percent failure is the **error budget**. If the budget is used up, the team slows down feature releases and fixes reliability.

```
99.9% availability over 30 days
30 days x 24 h x 60 min = 43,200 minutes
Error budget = 0.1% x 43,200 = 43.2 minutes of downtime allowed
```

🎯 **INTERVIEW CORNER**

Q: What is the difference between DevOps and SRE? **A:** DevOps is a broad culture and set of practices. SRE is one concrete way to implement it, focused on reliability, using SLOs, error budgets and software engineering to run operations. Google's line: "class SRE implements DevOps".

Q: What is an SLI, SLO and SLA? **A:** SLI is the measured indicator (for example, percent of successful requests). SLO is the internal target for that SLI (99.9 percent). SLA is the contract with customers, usually looser than the SLO, with penalties if broken.

🔑 **KEY POINT**

For a first switch, target "DevOps Engineer" or "Cloud Engineer" roles. They have the widest hiring and test the tools in this book directly. SRE and Platform roles often expect some production experience first, so they make a great second step.

CHECK YOUR PROGRESS 1.3

1. **Calculate the monthly error budget for a 99.95 percent SLO over 30 days.**
   Answer: 0.05 percent of 43,200 minutes = 21.6 minutes.
2. **Which role is most likely to have a coding (DSA) round?**
   Answer: SRE, especially at large product companies.
3. **What does a Platform Engineer build?**
   Answer: An internal developer platform: self-service tools, templates and paved roads so product teams can build, deploy and run services without filing tickets.

1.4 The Toolchain Map of This Book

This book covers 13 tools and skills. Each one solves one step of the journey from code to customer. Learn them as a chain, not as separate subjects. When an interviewer asks "walk me through your pipeline", they want this chain.

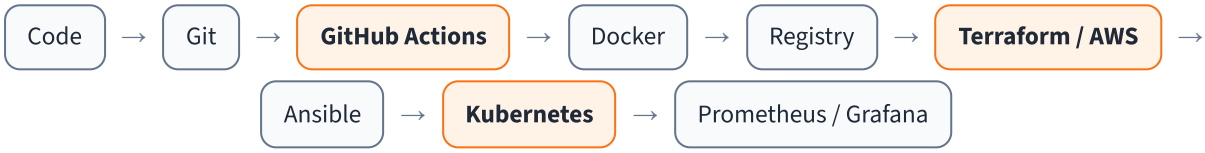


Figure 1.4: The toolchain map. One commit travels left to right; alerts travel right to left back to the developer.

#	Tool / skill	Job in the chain
1	Linux	The operating system almost every server and container runs on.
2	Bash	Glue scripts, quick automation, pipeline steps.
3	Python	Bigger automation, API calls, cloud scripts with boto3.
4	Git	Version control; every change starts as a commit.
5	GitHub Actions	CI/CD engine: runs tests, builds images, deploys on every push.
6	Docker	Packages the app and its dependencies into an image.
7	Container registry	Stores images (Docker Hub, Amazon ECR, GitHub Container Registry).
8	Terraform	Creates cloud infra (VPC, EC2, EKS, RDS) from code.
9	AWS	The cloud where the infra lives.
10	Ansible	Configures servers: packages, users, files, services.
11	Kubernetes	Runs and scales containers, self-heals, rolls out updates.
12	Prometheus	Collects metrics and fires alerts.
13	Grafana	Dashboards to see what metrics are saying.

1.4.1 One Commit, End to End

```
Developer laptop
  | git push
  v
GitHub repo ---> GitHub Actions workflow
                   | 1. run tests
                   | 2. docker build
                   | 3. docker push ---> Registry (ECR)
                   | 4. kubectl / helm deploy
                   v
AWS (built by Terraform, nodes set up by Ansible)
  +-- Kubernetes cluster --- pulls image from ECR
         |
         +-- Prometheus scrapes metrics ⟶ Grafana
         +-- Alertmanager ⟶ Slack / PagerDuty ⟶ on-call
```

Figure 1.5: The life of a commit through the tools in this book.

Here is a tiny taste of what each step looks like in real files. You will learn each one properly in later chapters.

```yaml
# .github/workflows/ci.yml  (GitHub Actions)
name: ci
on:
  push:
    branches: [main]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: make test
      - run: docker build -t myapp:${{ github.sha }} .
```

```
# main.tf  (Terraform)
resource "aws_instance" "web" {
  ami           = var.ami_id
  instance_type = "t3.micro"
  tags = { Name = "web-1", Owner = "team-a" }
}
```

```
# PromQL: requests per second that returned 5xx, last 5 minutes
sum(rate(http_requests_total{status=~"5.."}[5m]))
```

> 💼 **AT WORK**
>
> Monday 10:40 AM. Grafana shows checkout latency jumped from 200 ms to 2 s right after the 10:30 deploy. The on-call engineer checks the GitHub Actions run, sees which commit went out, and runs `kubectl rollout undo deployment/checkout`. Latency drops back in two minutes. Later, a blameless postmortem finds a missing database index. The fix ships with a new test. This is the full toolchain earning its salary: fast detection (Prometheus/Grafana), fast traceability (Git and CI), fast recovery (Kubernetes rollback).

🎯 **INTERVIEW CORNER**

Q: Walk me through what happens after a developer pushes code in your ideal pipeline. **A:** CI triggers, runs lint and unit tests, builds a Docker image tagged with the commit SHA, scans it, pushes to the registry, deploys to staging, runs smoke tests, then promotes to production with a rolling or canary rollout. Monitoring watches error rate and latency; if they spike, we roll back automatically or manually.

Q: Why tag images with the Git SHA instead of "latest"? **A:** The SHA is unique and traceable to an exact commit, so rollbacks are precise. "latest" changes over time, so you cannot tell what is actually running.

Q: Terraform and Ansible both "automate infra". Why use both? **A:** Terraform is best at provisioning (creating cloud resources and tracking them in state). Ansible is best at configuring what is inside servers. With Kubernetes and images, the Ansible part shrinks, but it is still common for VMs and bare metal.

✅ **CHECK YOUR PROGRESS 1.4**

1. **Which tool stores Docker images between build and deploy?**
   Answer: A container registry such as Amazon ECR, Docker Hub or GitHub Container Registry.
2. **What is the difference between Prometheus and Grafana?**
   Answer: Prometheus collects and stores metrics and evaluates alert rules. Grafana visualises metrics from Prometheus and other sources on dashboards.

1.5 The 90-Day Switch Plan

Ninety days is enough to become interview-ready for a junior or mid DevOps role if you study about 2 hours on weekdays and 5 to 6 hours on weekends. It is not enough to "master" anything. Nobody masters Kubernetes in 90 days; even Kubernetes maintainers keep reading the release notes.

Weeks	Focus	Done when you can...
1-2	Linux + Bash	Debug a full disk, manage a systemd service, write a health-check script.
3	Git + Python basics	Rebase, resolve conflicts, write a log parser and an API script.
4-5	Docker + GitHub Actions	Write a multi-stage Dockerfile and a pipeline that builds and pushes it.
6-7	AWS core + Terraform	Build VPC, subnets, EC2, security groups and S3 from Terraform with remote state.
8	Ansible	Configure a fresh server (nginx, users, firewall) with a role.
9-10	Kubernetes	Deploy an app with Deployment, Service, Ingress, ConfigMap, probes; debug CrashLoopBackOff.
11	Prometheus + Grafana	Write PromQL for rate, error ratio and p95 latency; build one dashboard and one alert.

Weeks	Focus	Done when you can...
12	Capstone + mock interviews	Finish portfolio, polish resume, do 3 mock interviews, start applying.
13	Apply + revise	Apply daily, revise weak areas from interview feedback.

Start applying from week 9, not week 13. Early interviews are practice; they show you what the market actually asks. Keep a simple spreadsheet: company, role, date applied, round reached, questions asked. After ten interviews this sheet becomes your best study guide.

1.5.1 Daily Routine

- **30 minutes**: concept reading (this book, official docs).
- **60 minutes**: hands-on lab. Type every command. Break things on purpose.
- **20 minutes**: write notes in your own words in a public GitHub repo ("til" or "devops-notes").
- **10 minutes**: answer 3 interview questions aloud, as if to an interviewer.

> ⚠️ **COMMON MISTAKE**
>
> Watching 40 hours of video courses at 1.5x speed and never typing a command. Watching someone swim does not teach swimming. If your terminal history is empty, your preparation is empty.

> 🧪 **HANDS-ON LAB 1.1: SET UP YOUR DEVOPS WORKBENCH**
>
> 1. Install WSL2 with Ubuntu (Windows) or use your Mac/Linux terminal.
> 2. Install Git, Docker (Docker Desktop or Docker Engine), Python 3.11+, the AWS CLI v2, Terraform, kubectl and kind or minikube.
> 3. Verify each tool:
> `git --version; docker version; python3 --version; aws --version; terraform version; kubectl version --client`
> 4. Create an AWS free tier account. Immediately enable MFA on the root user, create an IAM user or use IAM Identity Center for daily work, and set a billing budget alert of a small amount (for example 5 USD).
> 5. Create a public GitHub repo called `devops-90-days` and commit a README with your plan table.

> 💡 **DO YOU KNOW?**
>
> Forgotten cloud resources are the most common way learners get surprise bills. A NAT gateway or a load balancer left running for a month can cost more than a nice dinner. Always run `terraform destroy` after a lab and keep a billing alert on.

CHECK YOUR PROGRESS 1.5

1. **Why should you start applying in week 9?**
   Answer: Early interviews reveal what the market really asks, give practice, and the hiring process takes weeks anyway.
2. **Name two safety steps right after creating an AWS account.**
   Answer: Enable MFA on root and set a billing budget alert. Also stop using root for daily work.

1.6 Building a Portfolio: Three Projects

Recruiters see hundreds of resumes that list "Docker, Kubernetes, Terraform". A link to a clean GitHub repo with a working pipeline and a README with a diagram is what separates you. Build three projects, each bigger than the last.

1.6.1 Project 1: Containerised App with CI

A small API (Python Flask/FastAPI or Node) with tests, a multi-stage Dockerfile and a GitHub Actions pipeline that tests, builds and pushes the image to GitHub Container Registry. Add a status badge to the README.

1.6.2 Project 2: Infrastructure as Code on AWS

Terraform code that builds a VPC with public and private subnets, an EC2 instance or Auto Scaling group behind a load balancer, and an S3 backend for state with locking. Ansible configures nginx on the instance. Add a `Makefile` with `make up` and `make down`.

1.6.3 Project 3: The Capstone

Deploy the Project 1 app to Kubernetes (kind locally, or EKS if your budget allows; EKS has a control plane charge, so destroy it after demos). Use a Helm chart or Kustomize, GitHub Actions for deploy, Prometheus and Grafana for monitoring, and one alert that fires on high error rate. Write a postmortem for a failure you created on purpose.

```
capstone/
  app/                # source + tests
  Dockerfile
  .github/workflows/
    ci.yml            # test, build, push
    deploy.yml        # deploy to cluster
  infra/terraform/    # VPC, EKS or EC2
  infra/ansible/      # server config (if VMs)
  k8s/                # manifests or helm chart
  monitoring/         # alert rules, dashboard JSON
  docs/architecture.png
  docs/postmortem-001.md
  README.md           # what, why, how to run, diagram
```

> 🔑 **KEY POINT**
>
> A README is your silent interviewer. It should answer in 30 seconds: what does this do, what does the architecture look like, how do I run it, and what did you learn. Include a screenshot of the Grafana dashboard and the green pipeline.

> ✅ **CHECK YOUR PROGRESS 1.6**
>
> 1. **Why write a postmortem in a personal project?**
>
>   **Answer:** It shows incident thinking: detection, root cause, fix and prevention. That is exactly what on-call interviews test.
>
> 2. **What must every portfolio README contain?**
>
>   **Answer:** Purpose, architecture diagram, how to run and tear down, tools used and lessons learnt.

1.7 Resume Tips for a DevOps Switch

Your resume has about 6 to 10 seconds of a recruiter's attention on first pass. Make those seconds count.

- **One page** if you have under 5 years of experience. Two pages maximum after that.
- **Title line** matching the job: "DevOps Engineer | AWS, Terraform, Kubernetes", not "Software Developer seeking opportunities".
- **Skills block** grouped: Cloud, IaC, Containers, CI/CD, Monitoring, Scripting. Only list what you can defend for 10 minutes.
- **Bullets with numbers**: action verb + what + tool + result. "Cut build time from 18 to 6 minutes by caching dependencies in GitHub Actions."
- **Reframe your current job**: if you wrote a deploy script, set up Jenkins, or fixed a production issue as a developer, that is DevOps experience. Put it first.
- **Projects section** with GitHub links if your job has little DevOps work.
- **Certifications** help for cloud roles: AWS Certified Cloud Practitioner is a start; AWS Solutions Architect Associate or CKA (Certified Kubernetes Administrator) carry more weight.

Weak bullet	Strong bullet
Worked on Docker and Kubernetes.	Containerised 4 Java services and migrated them to Kubernetes, reducing deploy time from 40 minutes to 8.
Responsible for monitoring.	Built Prometheus alerts and Grafana dashboards for 12 services, cutting mean time to detect from about 20 to 3 minutes.
Used Terraform.	Wrote reusable Terraform modules for VPC and EKS used by 3 teams; new environment setup went from 2 days to 30 minutes.

😄 **FUNNY SIDE**

A resume listing 47 tools including "Kubernetes (expert)" got the question "What is a Pod?" The candidate said "a group of Kubernetes". Technically creative. Not hired.

✅ **CHECK YOUR PROGRESS 1.7**

1. **Rewrite "Handled CI/CD" as a strong bullet.**

   Answer: Example: "Built a GitHub Actions pipeline for 6 microservices with tests, image scanning and staged deploys, increasing release frequency from monthly to daily."

2. **Why should you list only tools you can defend?**

   Answer: Interviewers pick items from your resume and go deep. One bluff found ruins trust in everything else.

1.8 Typical Interview Rounds

The process varies, but most DevOps hiring in India and abroad follows a similar shape.



Figure 1.6: A typical DevOps interview funnel.

Round	What happens	How to prepare
Recruiter screen	Experience, notice period, salary, tools match	A crisp 60-second intro; know your numbers.
Technical screen	Rapid questions: Linux, networking, Docker, K8s, CI/CD, cloud	Check Your Progress and Interview Corner boxes in this book.
Hands-on / live debug	Fix a broken Dockerfile, write a Bash/Python script, debug a pod, write Terraform	Labs. Speak your thinking aloud while typing.
System design	Design CI/CD for 50 microservices, a highly available web app on AWS, a logging system	Practise drawing diagrams; discuss trade-offs and failure modes.
Behavioural / manager	Incidents you handled, conflicts, ownership	Prepare 5 STAR stories (Situation, Task, Action, Result).

🎯 **INTERVIEW CORNER**

Q: Tell me about a production incident you handled. **A:** Use STAR. Situation: payment API errors at 9 PM. Task: restore service. Action: checked dashboards, saw database connections maxed, found a new deploy with a connection leak, rolled back, then added a connection pool limit and an alert. Result: restored in 15 minutes; leak never recurred.

Q: Why do you want to move into DevOps? **A:** Give a real, specific reason: "I enjoyed automating our team's deployment and fixing production issues more than feature work, and I want to own reliability end to end." Avoid "it pays more", even if it does.

1.8.1 Troubleshooting Your Job Search

Symptom	Likely cause	Fix
No recruiter calls after 50 applications	Resume keywords and title do not match; no proof of skills	Match title to role, add numbers and GitHub links, ask for referrals.
Calls come, but you fail technical screens	Theory gaps	Revise fundamentals: Linux, networking, Docker, K8s objects. Practise aloud.
Pass screens, fail hands-on rounds	Not enough typing practice	Do timed labs; rebuild projects from scratch without notes.
Reach final round, no offer	Weak stories or communication	Prepare STAR stories, show ownership, ask good questions at the end.

💼 **AT WORK**

A backend developer at a Bengaluru fintech noticed builds took 25 minutes. She added dependency caching and parallel test jobs, bringing it to 9 minutes, and wrote a one-page note. Her manager let her own the pipeline. Six months later she moved internally to the platform team. Internal switches are often the fastest path: solve a visible ops pain in your current job first.

✅ **CHECK YOUR PROGRESS 1.8**

1. **What does STAR stand for?**
   Answer: Situation, Task, Action, Result.
2. **What should you do in a live-debug round while typing?**
   Answer: Explain your thinking aloud, state hypotheses and how you will check them, so the interviewer sees your method even if you do not finish.

What Have We Learnt?

- DevOps is a culture of shared ownership supported by automation, not a tool or a single person.
- CALMS (Culture, Automation, Lean, Measurement, Sharing) is a quick health check for DevOps adoption.
- The DORA metrics are deployment frequency, lead time, change failure rate and time to restore; good teams are fast and stable.
- CI integrates and tests every change; continuous delivery keeps every build releasable; continuous deployment releases automatically.
- DevOps, SRE, Platform and Cloud roles overlap but interview for different depths; DevOps and Cloud are the easiest first switch.
- SLOs and error budgets turn reliability into a number teams can manage.
- The toolchain flows: code, Git, GitHub Actions, Docker, registry, Terraform/AWS, Ansible, Kubernetes, Prometheus/Grafana.
- A 90-day plan with daily hands-on labs and early applications makes a switch realistic.
- Three portfolio projects with strong READMEs and numbered resume bullets prove your skills.
- Interviews usually include screens, a hands-on round, system design and behavioural STAR stories.

Key terms: DevOps, CALMS, DORA metrics, lead time, change failure rate, MTTR, SDLC, continuous integration, continuous delivery, continuous deployment, SRE, SLI, SLO, SLA, error budget, platform engineering, toolchain, container registry, infrastructure as code, STAR method.

🔑 **RESUME LINE**

"Designed and documented an end-to-end CI/CD pipeline (GitHub Actions, Docker, Terraform, Kubernetes) for a 3-service app, enabling 10+ deploys per day with automatic rollback on failed health checks."
"Introduced DORA metric tracking for 5 teams; lead time dropped from 6 days to under 1 day within a quarter."

Exercises

A. Multiple Choice Questions

1. Which of these is NOT a DORA metric?
   a. Deployment frequency
   b. Lines of code per day
   c. Change failure rate
   d. Time to restore service

2. In continuous deployment, production releases happen:
   a. Once a sprint
   b. After manager approval
   c. Automatically after all checks pass
   d. Only on weekends

3. A 99.9 percent monthly SLO (30 days) allows about how much downtime?
   a. 4.3 minutes
   b. 43 minutes
   c. 4.3 hours
   d. 43 hours

4. Which tool is mainly used to provision cloud resources from code?
   a. Grafana
   b. Ansible
   c. Terraform
   d. Git

5. The "M" in CALMS stands for:
   a. Monitoring
   b. Measurement
   c. Microservices
   d. Management

6. Which role most often focuses on SLOs and error budgets?
   a. SRE
   b. QA engineer
   c. Data analyst
   d. Frontend developer

B. Short Answer Questions

1. Explain the difference between continuous delivery and continuous deployment with an example.
2. Why are speed and stability not opposites according to DORA research?
3. List the toolchain of this book in order and give one line on each tool's job.
4. Write two strong resume bullets for a developer who automated deployments with a shell script.

C. Scenario Questions

1. Production error rate jumped to 20 percent right after a deploy. You are on call. What do you do?
2. Your team deploys once a month and every release causes a weekend outage. The manager asks you for a plan. What do you propose?
3. You have used 80 percent of this month's error budget by day 10. A big feature launch is planned for day 15. What do you recommend?

D. Mini Project

Create a public GitHub repo `devops-roadmap` containing: your 90-day plan as a Markdown table with checkboxes, an ASCII or image toolchain diagram, and a `tracker.csv` for job applications. Update it weekly with what you learnt. This becomes a visible streak of learning recruiters can see.

Answer Key: Chapter 1

A. MCQ

1. (b) Lines of code is a vanity metric, not a DORA metric.
2. (c) No human approval; passing checks release automatically.
3. (b) 0.1 percent of 43,200 minutes is 43.2 minutes.
4. (c) Terraform provisions; Ansible configures.
5. (b) Measurement.
6. (a) SRE uses SLOs and error budgets as core tools.

B. Short Answers

1. Continuous delivery keeps every build ready and a human approves the prod release (for example, a release manager clicks deploy at 4 PM). Continuous deployment releases each passing commit automatically (a merged fix is live in 15 minutes with no click).
2. Small, frequent, automated changes are easier to test, review and roll back, so they cause fewer and shorter failures. High performers score well on both speed and stability metrics.
3. Linux (server OS), Bash (glue scripts), Python (automation), Git (version control), GitHub Actions (CI/CD), Docker (packaging), registry (image storage), Terraform (provisioning), AWS (cloud), Ansible (configuration), Kubernetes (running containers), Prometheus (metrics and alerts), Grafana (dashboards).
4. "Automated release of 3 services with a Bash deploy script, reducing manual deploy time from 45 to 5 minutes." "Added pre-deploy health checks and rollback to the script, cutting failed releases by about half."

C. Scenario Questions

1. 1) Acknowledge the alert and tell the team in the incident channel. 2) Confirm on dashboards that errors began at deploy time. 3) Roll back immediately (for example `kubectl rollout undo` or redeploy the previous image tag); restore first, debug later. 4) Verify error rate returns to normal. 5) Investigate logs and the diff. 6) Write a blameless postmortem with action items such as a canary step or a missing test.
2. Measure current DORA metrics. Add CI with automated tests on every pull request. Move to smaller, more frequent releases behind feature flags. Automate deploys with a pipeline and add staging plus smoke tests. Use rolling or canary deploys with automatic rollback. Review metrics monthly to show improvement.
3. Follow the error budget policy: pause risky launches, focus on reliability fixes for the causes of the burn, and launch day 15 only if burn rate is under control, perhaps with a canary and feature flag. Discuss openly with product; the budget is a shared agreement, not a veto by ops alone.

D. Mini Project: Model Solution

Repo layout: `README.md` (goal, plan table with `- [ ]` checkboxes per week, toolchain diagram in a code block), `tracker.csv` with columns `company,role,date,source,round,result,questions`, and

a `notes/` folder with one Markdown file per week. Commit at least weekly so the contribution graph shows steady progress. Pin the repo on your GitHub profile.

CHAPTER 2

Linux for DevOps

"There is no cloud. It is just someone else's Linux computer."

After studying this chapter, you will be able to

- Navigate the Linux filesystem and use the essential commands without looking them up.
- Manage permissions, ownership, users, groups and sudo safely.
- Control processes, signals and systemd services, and read logs with journalctl.
- Check disk, memory and network, and use SSH keys, cron and environment variables.
- Troubleshoot a slow or full server step by step, the way on-call engineers do.

Linux is like the kitchen of a busy restaurant. Customers (users) only see the food on the table (your website). Behind the door, there are cooks (processes), a pantry (the filesystem), a head chef who decides who can touch the knives (permissions and sudo), and a notebook where every mistake is written down (logs). As a DevOps engineer, you work in this kitchen every day.

Almost every server in the cloud, every Docker container and every Kubernetes node runs Linux. When an interviewer says "the server is slow, what do you do?", they are testing this chapter. Learn it well, and every other tool in this book becomes easier.

2.1 The Filesystem Hierarchy

In Linux, everything starts at the root directory `/`. There are no drive letters like `C:`. Disks are attached (mounted) somewhere inside this single tree. The layout follows the Filesystem Hierarchy Standard (FHS).

Path	What lives there	DevOps relevance
/etc	System configuration files	nginx, ssh, systemd unit overrides, hosts, fstab
/var	Variable data: logs, caches, spool	/var/log fills disks; /var/lib/docker holds images
/home	Users' home directories	~/.ssh keys, dotfiles
/root	Home of the root user	Should be rarely used
/usr	Installed programs and libraries	/usr/bin, /usr/local/bin for custom tools
/opt	Optional third-party software	Vendor apps, agents
/tmp	Temporary files, often cleared on reboot	Scratch space; world-writable

Path	What lives there	DevOps relevance
/proc, /sys	Virtual files exposing kernel and process info	/proc/cpuinfo, /proc/meminfo, /proc/<pid>/
/dev	Device files	/dev/nvme0n1, /dev/null
/boot	Kernel and bootloader	A full /boot can break kernel upgrades

> 💡 **DO YOU KNOW?**
>
> In Linux "everything is a file". Even your running process is visible as a folder under `/proc`. Try `cat /proc/$$/status` to see details of your own shell. `$$` is the current shell's process ID.

2.1.1 Essential Commands

```
pwd                     # where am I?
ls -lah                 # long list, all files, human sizes
cd /var/log             # change directory; cd - goes back
mkdir -p app/config     # create nested dirs
cp -r src/ backup/      # copy recursively
mv old.txt new.txt      # move or rename
rm -rf build/           # delete recursively (careful!)
touch app.log           # create empty file / update time
cat, less, head -n 20, tail -n 50   # read files
find /var/log -name "*.gz" -mtime +7 # files older than 7 days
which nginx             # where is the binary?
ln -s /opt/app/v2 /opt/app/current  # symbolic link
```

```
$ ls -lah /etc/nginx
total 72K
drwxr-xr-x  8 root root 4.0K Mar 10 09:12 .
drwxr-xr-x 98 root root 4.0K Mar 12 11:40 ..
-rw-r--r--  1 root root 1.5K Mar 10 09:12 nginx.conf
drwxr-xr-x  2 root root 4.0K Mar 10 09:12 sites-enabled
```

> ⚠️ **COMMON MISTAKE**
>
> `rm -rf $DIR/*` in a script where `DIR` is empty becomes `rm -rf /*`. Always quote variables and use `${DIR:?}` in scripts so they fail if the variable is empty. More on this in Chapter 3.

> ✅ **CHECK YOUR PROGRESS 2.1**
>
> 1. **Where would you look for nginx configuration and nginx logs?**
>
>   **Answer:** Config in `/etc/nginx`; logs in `/var/log/nginx`.
>
> 2. **How do you find files ending in .log larger than 100 MB under /var?**
>
>   **Answer:** `find /var -name "*.log" -size +100M`

2.2 Permissions, Ownership and sudo

Every file has an owner, a group and three permission sets: user (u), group (g) and others (o). Each set has read (r=4), write (w=2) and execute (x=1).

```
-rwxr-x---  1 deploy www-data  2048 Mar 12 10:00 deploy.sh
 |  |  |
 |  |  +-- others: --- (0)
 |  +----- group:  r-x (4+1=5)
 +-------- user:   rwx (4+2+1=7)      ⇒ mode 750
```

```
chmod 750 deploy.sh          # numeric
chmod u+x,g-w script.sh      # symbolic
chown deploy:www-data app/   # owner and group
chown -R deploy app/         # recursive
umask                        # default mask, often 0022
```

For directories, **x** means "can enter", and **r** means "can list". A web server needs `x` on every parent directory to reach a file. Special bits: **setuid** (runs as the file owner, like `passwd`), **setgid** on directories (new files inherit the group) and the **sticky bit** on `/tmp` (only owners can delete their files).

2.2.1 sudo

`sudo` lets an allowed user run a command as root. Rules live in `/etc/sudoers` and `/etc/sudoers.d/`. Always edit with `visudo`, which checks syntax; a broken sudoers file can lock everyone out of root.

```
# /etc/sudoers.d/deploy  (edit with: sudo visudo -f /etc/sudoers.d/deploy)
deploy ALL=(root) NOPASSWD: /usr/bin/systemctl restart myapp
```

This gives the `deploy` user only one power: restart `myapp`. This is the principle of least privilege.

2.3 Users and Groups

```
sudo useradd -m -s /bin/bash -G docker ravi   # create user with home, shell, extra group
sudo passwd ravi                              # set password
sudo usermod -aG sudo ravi                    # append to group (-a is vital!)
id ravi
uid=1001(ravi) gid=1001(ravi) groups=1001(ravi),27(sudo),998(docker)
getent passwd ravi                            # entry from /etc/passwd
sudo userdel -r ravi                          # delete user and home
```

User info is in `/etc/passwd`, hashed passwords in `/etc/shadow`, groups in `/etc/group`. On Debian/Ubuntu, `adduser` is a friendlier interactive wrapper. Group changes apply at the next login; use `newgrp docker` or log out and in.

⚠️ **COMMON MISTAKE**

`usermod -G docker ravi` without `-a` replaces all of Ravi's extra groups with just `docker`. If one of those was `sudo`, Ravi just lost admin rights. Always use `-aG`.

🎯 **INTERVIEW CORNER**

Q: What does chmod 644 mean? **A:** Owner read/write, group read, others read. Typical for config files.

Q: A user gets "Permission denied" running ./script.sh. What do you check? **A:** Execute bit (`ls -l`), ownership, execute permission on parent directories, whether the filesystem is mounted `noexec`, and the shebang interpreter path.

Q: Why use sudo instead of logging in as root? **A:** Least privilege, per-user audit trail (sudo logs who ran what), and no shared root password.

✅ **CHECK YOUR PROGRESS 2.2 AND 2.3**

1. **Write the numeric mode for rwxr-xr-x.**
   Answer: 755.
2. **Why use visudo instead of a normal editor?**
   Answer: It locks the file and validates syntax before saving, preventing a broken sudoers file.
3. **Which file stores password hashes?**
   Answer: `/etc/shadow`, readable only by root.

2.4 Processes and Signals

A process is a running program with a PID (process ID). PID 1 is the init system, which is `systemd` on most modern distros.

```
$ ps aux --sort=-%cpu | head -5
USER   PID %CPU %MEM    VSZ   RSS TTY STAT START TIME COMMAND
app   2314 87.5 12.1 2154320 495312 ?  Sl  09:02 41:10 java -jar app.jar
root   812  2.1  0.4  72340  16420 ?  Ss  08:55  0:44 /usr/sbin/nginx
$ pgrep -a nginx
812 nginx: master process /usr/sbin/nginx
$ pstree -p | head
```

Process states in `STAT`: **R** running, **S** sleeping, **D** uninterruptible sleep (usually waiting for disk or network storage), **Z** zombie (finished, but parent has not collected its exit code), **T** stopped.

Signal	Number	Meaning
SIGTERM	15	Polite "please stop". Default for `kill`. App can clean up.
SIGKILL	9	Forced stop by the kernel. Cannot be caught. No cleanup.

Signal	Number	Meaning
SIGHUP	1	Terminal closed; many daemons reload config on it.
SIGINT	2	Ctrl+C.
SIGSTOP / SIGCONT	19 / 18 (x86)	Pause and resume.

```
kill 2314            # SIGTERM
kill -9 2314         # SIGKILL, last resort
kill -HUP 812        # ask nginx to reload
pkill -f "app.jar"   # by command line pattern
nohup ./job.sh &     # keep running after logout
jobs; fg %1; bg %1   # shell job control
```

😄 **FUNNY SIDE**

Using `kill -9` first is like switching off the mixer grinder at the mains while it still has chutney inside. It stops, yes. But someone will be cleaning the walls. Try SIGTERM, wait a few seconds, then SIGKILL.

🎯 **INTERVIEW CORNER**

Q: What is a zombie process and how do you remove it? **A:** A child that exited but whose parent did not call wait(). It uses no CPU or memory, only a PID slot. You cannot kill a zombie; fix or restart the parent, and init adopts and reaps it.

Q: Difference between SIGTERM and SIGKILL? **A:** SIGTERM can be caught for graceful shutdown; SIGKILL cannot be caught and stops the process immediately. Kubernetes sends SIGTERM, waits the grace period (30 s default), then SIGKILL.

✅ **CHECK YOUR PROGRESS 2.4**

1. **A process shows state D for a long time. What does that suggest?**

   Answer: It is stuck waiting on I/O, often a slow disk or hung NFS mount. Even SIGKILL may not end it until the I/O returns.

2. **Which signal do many daemons use to reload config?**

   Answer: SIGHUP.

2.5 systemd Services and journalctl

systemd starts services at boot, restarts them on failure and collects their logs in the journal. You will manage services with `systemctl` daily.

```
sudo systemctl status nginx
sudo systemctl start|stop|restart|reload nginx
sudo systemctl enable --now nginx      # start now and at boot
systemctl is-active nginx; systemctl is-enabled nginx
systemctl list-units --type=service --state=failed
sudo systemctl daemon-reload           # after editing unit files
```

```
$ systemctl status myapp
● myapp.service - My Python API
     Loaded: loaded (/etc/systemd/system/myapp.service; enabled; preset: enabled)
     Active: failed (Result: exit-code) since Tue 2026-03-12 10:14:02 UTC; 3min ago
    Process: 4120 ExecStart=/opt/myapp/venv/bin/gunicorn app:app (code=exited,
status=1/FAILURE)
```

2.5.1 Writing a Unit File

```
# /etc/systemd/system/myapp.service
[Unit]
Description=My Python API
After=network-online.target
Wants=network-online.target

[Service]
User=app
Group=app
WorkingDirectory=/opt/myapp
EnvironmentFile=/etc/myapp/env
ExecStart=/opt/myapp/venv/bin/gunicorn -b 0.0.0.0:8000 app:app
Restart=on-failure
RestartSec=5

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

2.5.2 journalctl

```
journalctl -u myapp                # logs for one unit
journalctl -u myapp -f             # follow live
journalctl -u myapp --since "10 min ago"
journalctl -p err -b               # errors since this boot
journalctl --disk-usage
sudo journalctl --vacuum-size=500M  # shrink journal
```

✅ CHECK YOUR PROGRESS 2.5

1. **You edited a unit file but the change has no effect. What did you forget?**
   Answer: `sudo systemctl daemon-reload`, then restart the service.
2. **What does Restart=on-failure do?**
   Answer: systemd restarts the service when it exits with a non-zero code or is killed by a signal, but not when it exits cleanly.

2.6 Package Managers

Task	Debian/Ubuntu (apt)	RHEL/Fedora/Amazon Linux 2023 (dnf)
Refresh index	`sudo apt update`	`sudo dnf makecache`
Install	`sudo apt install -y nginx`	`sudo dnf install -y nginx`
Upgrade all	`sudo apt upgrade -y`	`sudo dnf upgrade -y`
Remove	`sudo apt remove nginx`	`sudo dnf remove nginx`
Search	`apt search redis`	`dnf search redis`
Which package owns a file	`dpkg -S /usr/sbin/nginx`	`rpm -qf /usr/sbin/nginx`

In Dockerfiles, combine `apt-get update && apt-get install -y --no-install-recommends ...` in one `RUN` and clean `/var/lib/apt/lists/*` to keep images small. Use `apt-get` (stable interface) in scripts rather than `apt`.

2.7 Disk and Memory

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme0n1p1   30G   29G  1.0G  97% /
/dev/nvme1n1    100G   12G   88G  12% /data

$ df -i            # inodes; can be 100% even with free space
$ sudo du -xh --max-depth=1 /var | sort -h | tail -5
1.2G  /var/cache
3.8G  /var/lib
18G   /var/log
23G   /var

$ free -h
               total   used   free  shared  buff/cache  available
Mem:           7.7Gi  5.1Gi  310Mi   12Mi       2.3Gi      2.3Gi
Swap:             0B     0B     0B

$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
nvme0n1     259:0    0   30G  0 disk
└─nvme0n1p1 259:1    0   30G  0 part /
nvme1n1     259:2    0  100G  0 disk /data
```

Read `free` correctly: the important column is **available**, not **free**. Linux uses spare RAM as disk cache and gives it back when apps need it. "Free memory is wasted memory" is the kernel's motto.

```
# Attach and mount a new EBS volume
sudo mkfs.ext4 /dev/nvme1n1
sudo mkdir -p /data
sudo mount /dev/nvme1n1 /data
sudo blkid /dev/nvme1n1         # get UUID
# /etc/fstab entry so it survives reboot (nofail avoids boot hang)
UUID=3f1c...  /data  ext4  defaults,nofail  0  2
sudo mount -a                   # test fstab before rebooting!
```

✅ **CHECK YOUR PROGRESS 2.6 AND 2.7**

1. **df shows 40 percent used but you cannot create files. Why?**
   Answer: Inodes are exhausted, usually by millions of tiny files. Check with `df -i`.
2. **Why test fstab with mount -a?**
   Answer: A wrong fstab entry can stop the server from booting. `mount -a` shows errors while you still have a shell.

2.8 Logs, SSH, cron and Environment Variables

2.8.1 Logs

```
tail -f /var/log/nginx/access.log
tail -n 200 /var/log/syslog | grep -i error      # Ubuntu
grep -c " 500 " /var/log/nginx/access.log        # count 500s
grep -i "out of memory" /var/log/kern.log
zgrep "timeout" /var/log/nginx/error.log.2.gz    # search gzipped logs
dmesg -T | tail                                   # kernel messages, readable time
```

On RHEL-family systems the main log is `/var/log/messages` instead of `/var/log/syslog`. Rotation is handled by `logrotate` with configs in `/etc/logrotate.d/`.

2.8.2 SSH Keys and Config

```
ssh-keygen -t ed25519 -C "ravi@laptop"         # creates ~/.ssh/id_ed25519(.pub)
ssh-copy-id -i ~/.ssh/id_ed25519.pub ubuntu@10.0.1.15
chmod 700 ~/.ssh; chmod 600 ~/.ssh/id_ed25519 ~/.ssh/authorized_keys
ssh -i ~/.ssh/key.pem ec2-user@3.110.20.5
ssh -v web1                                     # verbose for debugging
```

```
# ~/.ssh/config
Host bastion
  HostName 3.110.20.5
  User ubuntu
  IdentityFile ~/.ssh/id_ed25519

Host web1
  HostName 10.0.1.15
  User ubuntu
  ProxyJump bastion
```

Now `ssh web1` hops through the bastion automatically. Hardening in `/etc/ssh/sshd_config`: `PasswordAuthentication no`, `PermitRootLogin no`, then `sudo sshd -t` to test and reload the service (named `ssh` on Ubuntu, `sshd` on RHEL). Keep your current session open while testing so you do not lock yourself out.

2.8.3 cron

```
# crontab -e   (min hour day-of-month month day-of-week command)
*/5 * * * *  /opt/scripts/healthcheck.sh >> /var/log/health.log 2>&1
30 2 * * *   /opt/scripts/backup.sh
0 9 * * 1-5  /opt/scripts/report.sh        # weekdays 9 AM
crontab -l                                  # list
```

cron runs with a tiny `PATH` and no login profile. Use absolute paths, set variables at the top of the crontab, and always redirect output so failures are visible. On systemd systems, **timers** are a modern alternative with journal logging.

2.8.4 Environment Variables

```
echo $PATH
export APP_ENV=prod              # current shell and children
env | grep APP_                  # list
APP_ENV=dev ./run.sh             # only for this command
echo 'export EDITOR=vim' >> ~/.bashrc
cat /proc/2314/environ | tr '\0' '\n'   # env of a running process (as root/owner)
```

> 🎯 **INTERVIEW CORNER**
>
> **Q:** SSH says "Permission denied (publickey)". What do you check? **A:** Correct user (ubuntu vs ec2-user), correct key with `-i`, key permissions 600, `~/.ssh` 700 and `authorized_keys` 600 on the server, the public key is actually in `authorized_keys`, and `ssh -v` output. On the server, check `/var/log/auth.log` or `journalctl -u ssh`.
>
> **Q:** A cron job works manually but not from cron. Why? **A:** Different environment: minimal PATH, no profile, different working directory, different user. Use absolute paths and log output to a file.

✅ **CHECK YOUR PROGRESS 2.8**

1. **Write a cron schedule for "every day at 11:45 PM".**
   Answer: `45 23 * * *`
2. **What does ProxyJump do?**
   Answer: Connects to the target through an intermediate host (bastion) in one command.

2.9 Vim Basics

On a minimal server, vim (or vi) may be the only editor. You need about ten keys to survive.

Key	Action
`i`	Insert mode (start typing)
`Esc`	Back to normal mode
`:w` / `:q` / `:wq` / `:q!`	Save / quit / save and quit / quit without saving
`/text` , `n`	Search, next match
`dd` , `yy` , `p`	Delete line, copy line, paste
`u` , `Ctrl+r`	Undo, redo
`gg` , `G` , `:42`	Top, bottom, go to line 42
`:%s/old/new/g`	Replace all
`:set number`	Show line numbers

💡 **DO YOU KNOW?**

"How to exit vim" is one of the most viewed questions in Stack Overflow history, with millions of views. The answer is `Esc` then `:q!` . You are now ahead of millions of developers.

✅ **CHECK YOUR PROGRESS 2.9**

1. **How do you replace every "8080" with "9090" in vim?**
   Answer: `:%s/8080/9090/g`
2. **How do you quit without saving?**
   Answer: `Esc` , then `:q!` .

2.10 Performance Troubleshooting

When a server is slow, check the four resources in order: **CPU, memory, disk I/O, network**. Brendan Gregg's USE method says: for each resource, check Utilisation, Saturation and Errors.

```
$ uptime
 10:21:44 up 12 days,  3:02,  1 user,  load average: 7.85, 6.10, 3.02
```

Load average is the number of processes running or waiting (including D state) averaged over 1, 5 and 15 minutes. Compare with CPU count (`nproc`). On a 2-vCPU box, 7.85 means heavy queuing. Rising numbers (3.02 to 7.85) mean it is getting worse.

```
top        # press P (sort CPU), M (sort memory), 1 (per-CPU), q (quit)
htop       # friendlier, colour, tree view (F5)

$ vmstat 1 5
procs -----memory------ ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd  free  buff cache  si  so   bi    bo   in   cs us sy id wa st
 6  3      0 310m  80m  2.2g   0   0  120  9800 2100 4300 35 10 15 40  0
```

In vmstat: **r** = runnable processes (CPU queue), **b** = blocked on I/O, **wa** = percent CPU time waiting for I/O, **si/so** = swap in/out, **st** = time stolen by the hypervisor. Here `wa 40` and `bo 9800` say the disk is the bottleneck, not the CPU.

```
$ iostat -xz 1        # from sysstat package
Device   r/s   w/s  rkB/s   wkB/s  r_await w_await aqu-sz  %util
nvme0n1  5.0 820.0   40.0 98000.0     1.20   45.30  37.10  99.80

$ ss -tulpn           # listening sockets and owning process
Netid State  Local Address:Port  Process
tcp   LISTEN 0.0.0.0:80          users:(("nginx",pid=812,fd=6))
tcp   LISTEN 127.0.0.1:5432      users:(("postgres",pid=901,fd=5))
$ ss -s               # socket summary, many TIME-WAIT? many ESTAB?
$ sudo lsof -i :8000  # who uses port 8000
$ sudo lsof -p 2314 | wc -l   # open files of a process
```

> 💼 **AT WORK**
>
> An alert says "API p95 latency 4 s". `top` shows CPU only 20 percent busy, but load average is 12 on 4 cores. `vmstat` shows high `wa` . `iostat` shows the root disk at 100 percent util with high `w_await` . `iotop` points to a debug logger writing gigabytes per minute after someone set `LOG_LEVEL=DEBUG` . Setting it back to INFO and restarting fixed latency in a minute. The follow-up: alert on disk util and keep log level in config managed by Git.

2.11 Troubleshooting a Slow or Full Server, Step by Step

2.11.1 Disk Full

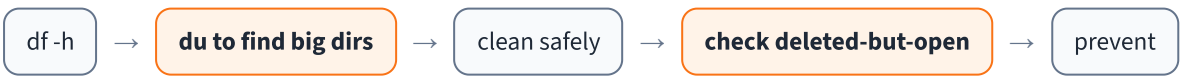


Figure 2.1: The disk-full playbook.

```
df -h /                                      # 1. which filesystem?
df -i /                                      # 2. inodes too?
sudo du -xh --max-depth=1 / 2>/dev/null | sort -h | tail   # 3. biggest dirs
sudo find /var/log -type f -size +500M -exec ls -lh {} \;
sudo journalctl --vacuum-size=500M           # 4. clean journal
docker system df; docker system prune        # 5. Docker leftovers (careful)
sudo lsof +L1                                 # 6. deleted files still held open
# 7. Fix a held log: truncate instead of rm, or restart the process
sudo truncate -s 0 /var/log/myapp/app.log
```

> 🔑 **KEY POINT**
>
> If you `rm` a huge log file that a process still has open, the space is **not** freed until the process closes it. `df` stays at 100 percent while `du` looks fine. Find it with `lsof +L1` and restart the process, or truncate files instead of deleting them.

2.11.2 Slow Server

1. `uptime` : load vs `nproc` . Is it getting worse?
2. `dmesg -T | tail` : OOM killer, disk errors, network drops?
3. `top` / `htop` : which process uses CPU or memory?
4. `free -h` , `vmstat 1` : low available memory, swapping, high wa?
5. `iostat -xz 1` : disk saturated?
6. `ss -s` , `ss -tnp` : connection pile-up, many CLOSE-WAIT (app not closing sockets)?
7. App logs and `journalctl -u app` : errors, timeouts to a database?
8. Fix the immediate issue, then add monitoring and an alert so you see it earlier next time.

2.11.3 Common Errors

Symptom	Cause	Fix
`No space left on device`	Disk or inodes full	df -h, df -i, du, clean logs, lsof +L1
`Out of memory: Killed process 2314 (java)` in dmesg	OOM killer	Find leak, lower heap, add RAM or swap, set limits
`bind: Address already in use`	Port taken	`ss -tulpn | grep :8000` , stop old process
`Too many open files`	File descriptor limit	Raise `LimitNOFILE` in unit file; check for fd leaks
`sudo: unable to resolve host`	Hostname missing in /etc/hosts	Add `127.0.1.1 hostname`
Service "active (running)" but site down	Listening on 127.0.0.1 only, or firewall/security group	`ss -tlnp` , check ufw/firewalld and cloud security group
`Read-only file system`	Kernel remounted after disk errors	Check dmesg, fsck, replace volume

💉 HANDS-ON LAB 2.1: BREAK AND FIX A SERVER

1. Launch an Ubuntu VM (multipass, VirtualBox, WSL2 or a t3.micro EC2 on free tier).
2. Install nginx: `sudo apt update && sudo apt install -y nginx sysstat`. Check `systemctl status nginx` and `curl -I localhost`.
3. Fill the disk: `fallocate -l 2G /tmp/big.img` (adjust size). Run `df -h` and find it with `du`.
4. Simulate the deleted-open-file trap: `python3 -c "f=open('/tmp/held.log','w'); f.write('x'*500000000); f.flush(); input()"` in one terminal; in another `rm /tmp/held.log`, then `df -h` and `sudo lsof +L1`. Press Enter in the first terminal and watch space return.
5. Create load: `yes > /dev/null &` twice. Watch `uptime`, `top`, `vmstat 1`. Kill with `pkill yes`.
6. Write the myapp.service unit from 2.5.1 for a simple `python3 -m http.server 8000`, enable it, kill the process with `kill -9` and watch systemd restart it in `journalctl -u myapp -f`.
7. Clean up: delete files and stop the VM or terminate the EC2 instance.

🎯 INTERVIEW CORNER

Q: df shows the disk 100 percent full but du shows only 10 GB used of 30 GB. Why? **A:** Deleted files still held open by a process. Use `lsof +L1`, then restart or signal that process. Less often, files hidden under a mount point.

Q: What is load average and what is "high"? **A:** Average number of runnable plus uninterruptible processes over 1, 5, 15 minutes. Compare to CPU cores; consistently above core count means queuing.

Q: How do you find which process listens on port 443? **A:** `sudo ss -tlnp | grep :443` or `sudo lsof -i :443`.

✅ CHECK YOUR PROGRESS 2.10 AND 2.11

1. **vmstat shows high wa and high b. What is the likely bottleneck?**
   Answer: Disk I/O. Confirm with `iostat -xz 1` and find the writer with `iotop`.
2. **How do you free space from a huge log held by a running process without restarting it?**
   Answer: `truncate -s 0 file` (or `: > file`) instead of deleting it.
3. **Many sockets are in CLOSE-WAIT. What does that mean?**
   Answer: The remote side closed the connection but your application has not closed its socket, usually an app bug or connection leak.

What Have We Learnt?

- Linux has a single tree from `/`; config is in /etc, logs and data in /var, kernel info in /proc.
- Permissions are rwx for user, group and others; chmod, chown and least-privilege sudo rules keep systems safe.
- Use `usermod -aG`, never plain `-G`, when adding users to groups.
- Processes receive signals; prefer SIGTERM and use SIGKILL only as a last resort.
- systemd manages services through unit files; journalctl reads their logs.
- apt and dnf install software; combine update and install in one Docker RUN line.
- df, du, lsblk and free show disk and memory; "available" memory is what matters.
- SSH keys plus ~/.ssh/config and ProxyJump make secure access easy; cron needs absolute paths.
- Troubleshoot with the USE method: CPU, memory, disk, network, using top, vmstat, iostat, ss and lsof.
- A full disk with "missing" space usually means deleted files still held open.

Key terms: FHS, root filesystem, inode, chmod, chown, umask, sudoers, visudo, PID, signal, SIGTERM, SIGKILL, zombie, systemd, unit file, journalctl, apt, dnf, fstab, mount, load average, iowait, USE method, ss, lsof, cron, ProxyJump, environment variable.

🔑 **RESUME LINE**

"Administered 40+ Ubuntu and Amazon Linux servers; wrote systemd units and logrotate policies that cut disk-full incidents from 6 per month to zero."
"Diagnosed a production latency spike to disk I/O saturation using vmstat and iostat, restoring p95 latency from 4 s to 300 ms within 15 minutes."

Exercises

A. Multiple Choice Questions

1. Which directory holds most system configuration?
   a. /var b. /etc
   c. /usr d. /opt
2. chmod 640 gives others:
   a. read b. read and write
   c. no access d. execute
3. Which signal cannot be caught by a process?
   a. SIGTERM b. SIGHUP
   c. SIGINT d. SIGKILL
4. After editing a systemd unit file you must run:
   a. systemctl daemon-reload b. systemctl reboot
   c. journalctl --flush d. systemctl mask
5. Which command lists listening TCP ports with process names?
   a. df -h b. ss -tlnp
   c. free -h d. lsblk
6. In vmstat output, a high "wa" value means:
   a. CPU waiting on I/O b. Swap is full
   c. Network is down d. Too many users logged in

B. Short Answer Questions

1. Explain the difference between "free" and "available" in `free -h`.
2. Write a sudoers rule allowing user `ci` to restart only the `api` service without a password.
3. List four reasons a cron job might work in your shell but fail in cron.
4. What is the difference between `systemctl enable` and `systemctl start`?

C. Scenario Questions

1. At 3 AM you get "disk 98 percent full on db-1". What do you do?
2. Users report the website is very slow. CPU in top is 15 percent. What next?
3. After a deploy, `systemctl status api` shows "activating (auto-restart)" in a loop. What do you do?

D. Mini Project

Build `linux-server-baseline`: a repo with a README runbook for disk-full, high-CPU and service-down incidents, a hardened `sshd_config` snippet, a sample systemd unit, a logrotate config for your app and a `bootstrap.sh` that creates a deploy user, installs packages and enables the service. Test it on a fresh VM.

Answer Key: Chapter 2

A. MCQ

1. (b) /etc holds configuration.
2. (c) The last digit 0 means no permissions for others.
3. (d) SIGKILL is handled by the kernel and cannot be caught.
4. (a) daemon-reload makes systemd re-read unit files.
5. (b) ss -tlnp shows TCP listeners with processes.
6. (a) wa is the percentage of time CPU is idle waiting for I/O.

B. Short Answers

1. "free" is RAM not used for anything. "available" estimates how much memory apps can get, including reclaimable cache. Use "available" to judge memory pressure.
2. In `/etc/sudoers.d/ci` via visudo: `ci ALL=(root) NOPASSWD: /usr/bin/systemctl restart api`
3. Minimal PATH, missing environment variables, different working directory, different user or permissions, output not logged so errors are invisible, and `%` characters needing escaping in crontab.
4. `enable` makes the service start at boot (creates symlinks); `start` runs it now. `enable --now` does both.

C. Scenario Questions

1. 1) `df -h` and `df -i` to confirm which filesystem. 2) `du -xh --max-depth=1` to find the culprit; on DB servers check logs, WAL or binlogs, and backups. 3) Free space safely: compress or move old logs, vacuum journal, never delete live database files. 4) `lsof +L1` for deleted-open files. 5) If still critical, grow the EBS volume and filesystem (`growpart` , `resize2fs` or `xfs_growfs`). 6) Add logrotate, retention and an alert at 80 percent.
2. Check `uptime` load vs cores, `vmstat 1` for wa and swapping, `iostat` for disk, `free -h` , `ss -s` for connection build-up, and app logs for slow downstream calls such as the database. Check the load balancer and database metrics too; the slowness may not be on this host.
3. `journalctl -u api -n 100` to read the error (missing env var, port in use, bad path, permission denied). Compare the new release with the old one. Roll back to the previous version to restore service, fix the cause, and add a pre-deploy check or smoke test.

D. Mini Project: Model Solution

Files: `README.md` (runbooks with exact commands from 2.10 and 2.11), `sshd_config.d/hardening.conf` (`PasswordAuthentication no` , `PermitRootLogin no`), `systemd/myapp.service` , `logrotate/myapp` (daily, rotate 7, compress, missingok, copytruncate), and `bootstrap.sh` using `set -euo pipefail` , `useradd -m` , `apt-get install` , copying configs and `systemctl enable --now myapp` . Show a test run log in the README.