APEX IT SOLUTIONS

INFRASTRUCTURE · SECURITY · DEVOPS

15

CHAPTERS

FIELD GUIDE

The DevOps Field Guide

From Linux Commands to AI-Powered Infrastructure

Written by **Sohaib Aftab** · Senior DevOps Engineer & Co-Founder, Apex IT Solutions

EDITION
2026 Edition

CHAPTERS
15 + Epilogue

AUDIENCE
Students & Practitioners

WEB
apexitsolutions.co

FOREWORD · A NOTE FROM THE AUTHOR

A Note From The Author

Let me be honest with you right from the start.

When I graduated with a Software Engineering degree, I had no idea what DevOps was. None. I was thrown into an application support role, which is a very polite way of saying I spent my days staring at log files and pretending I understood what they meant.

Then one day, a senior engineer walked past my desk, glanced at my screen, and said — 'You should look into DevOps.' He walked away before I could ask a follow-up question. That was it. That was my introduction.

Three months later, I was promoted to DevOps Engineer. I want to be clear: this was not because I had mastered everything. It was because I refused to stop Googling. I broke things constantly. I once rebooted a production server by accident and had to sit in silence for eleven very long minutes while it came back up, sitting completely still, because I had a theory that any sudden movement on my end might somehow make things worse on the server's end. It made no sense. I did it anyway.

But here's what I learned: the people who succeed in DevOps are not necessarily the smartest ones. They are the ones who are not afraid to break things in test environments, who document their mistakes, and who are deeply, almost irrationally, curious about how systems work.

I have since worked in the data centers of every major telecom in Pakistan. I have racked servers at 2 AM. I have built Kubernetes clusters from bare metal in environments where the air conditioning was optional. I have written Terraform that worked perfectly, and Ansible playbooks that made absolutely no sense in retrospect. I have deployed LLMs on GPU clusters and explained to business stakeholders why 'just restart it' is not always the answer.

And now I am here, writing a guide for you — the students who are about to enter one of the most exciting fields in technology.

This guide is not just a list of tools. It is the roadmap I wish someone had handed me when I started. It will be honest with you about what actually matters, what you can safely ignore for now, and what is genuinely changing the industry in 2026.

Read it once for overview. Come back to specific sections when you need them. And when you break something in production one day — and you will — remember that the person who wrote this book has been there, and survived.

Good luck. You will not need as much of it as you think.

Sohaib Aftab

Senior DevOps Engineer & Co-Founder Apex IT Solutions

CONTENTS · WHAT'S INSIDE

Fifteen Chapters — and a Closing Word

From Linux fundamentals to AI-assisted infrastructure, the guide is structured as a single path you can read straight through, or dip into chapter-by-chapter as you need it.

CHAPTER 01

01

What Even Is DevOps?

Defining DevOps — culture, practice, the infinite loop, and what it is not.

IN THIS CHAPTER

7 sections, lists & callouts

Before we dive into commands and YAML files, let's settle something that confuses a lot of people, including many who have been working in tech for years.

DevOps is not a job title. It is not a tool. It is not a department you create by renaming your sysadmin team and giving them a free t-shirt.

DevOps is a culture and a set of practices that bridges the gap between software development and IT operations. The idea is simple: the people who write code and the people who run that code in production should talk to each other, share responsibility, and work in a way that allows software to be delivered reliably and at speed.

> **THE CLASSIC PROBLEM**
>
> Developer pushes code. Operations team deploys it. Something breaks. Developer says 'it worked on my machine.' Operations team quietly considers changing careers. This is the problem DevOps solves.

The Core Pillars

- **Continuous Integration** — developers merge code frequently into a shared repository
- **Continuous Delivery** — that code is automatically tested and ready to deploy at any time
- **Infrastructure as Code** — your servers and cloud resources are defined in files, not clicked together in a console
- **Monitoring and Observability** — you know what is happening in your systems at all times
- **Collaboration and Shared Responsibility** — devs and ops own the product together

The DevOps Lifecycle (The Infinite Loop You Will Live In)

The DevOps lifecycle is often shown as an infinite loop — and that is genuinely accurate, because the work never truly stops. Here is how it flows:

- **Plan** — Define what needs to be built or changed
- **Code** — Developers write it
- **Build** — Automated systems compile and package it
- **Test** — Automated tests verify it works
- **Release** — Approve it for deployment
- **Deploy** — Push it to production
- **Operate** — Keep it running, scale it, patch it
- **Monitor** — Watch what it is doing
- **Back to Plan** — Use what you learned to improve the next cycle

In 2026, AI is starting to assist at almost every step of this loop. We will cover that in its own chapter, because it deserves serious attention.

CHAPTER 02

02

Linux — The Foundation of Everything

Linux as foundation — navigation, processes, networking, permissions, shell scripting.

IN THIS CHAPTER

19 sections, lists & callouts

APEX IT SOLUTIONS

I am going to tell you something that might sound dramatic: every server you will ever work on in your DevOps career runs Linux. Every container you will ever deploy runs on a Linux kernel. Kubernetes runs on Linux. AWS runs on Linux. Your CI/CD pipelines execute on Linux.

If you do not understand Linux, you will spend your entire career pressing buttons without understanding what they do. That is a frightening position to be in at 2 AM when something is on fire.

> **STORY FROM THE FIELD**
>
> My first month in a real data center, I was asked to 'just check the disk usage' on a server. I SSH'd in, typed du -sh / and hit Enter. On a system with millions of small files. I watched in horror as the command crawled through every directory. The senior engineer saw my face and came over. He typed Ctrl+C, looked at me, and said 'df -h.' That is the day I started actually learning Linux.

The Absolute Essentials

Navigation and File Management

- **ls -la** — List everything including hidden files and permissions
- **cd, pwd, mkdir, rm -rf** — The basics you will use every single day
- **find /path -name 'file*'** — Search for files without losing your mind
- **grep -r 'pattern' /logs/** — Search inside files
- **tail -f /var/log/app.log** — Watch logs in real time (you will use this constantly)

Process Management

- **ps aux | grep process** — Find what is running
- **kill -9 PID** — The nuclear option
- **top / htop** — See system resources at a glance
- **systemctl start/stop/status service** — Manage services
- **journalctl -u service -f** — Follow service logs

Networking

- **curl -I https://example.com** — Check HTTP response headers
- **netstat -tlnp** — See what ports are listening
- **ss -tlnp** — Modern replacement for netstat
- **ping, traceroute, nslookup, dig** — Diagnose connectivity
- **iptables -L** — View firewall rules (and occasionally panic at them)

Disk and Memory

- **df -h** — Disk space per partition
- **du -sh /path** — Size of a directory

- **free -m** — Memory usage
- **lsblk** — View block devices

Permissions (This Will Save You From Yourself)

File permissions in Linux follow a pattern: Owner, Group, Others. Each can Read (4), Write (2), Execute (1).

- **chmod 755 file** — Owner can do everything, others can read and execute
- **chmod 600 keyfile.pem** — Only owner can read (required for SSH keys)
- **chown user:group file** — Change ownership

Shell Scripting — Automation Starts Here

You do not need to be a programmer to work in DevOps. But you do need to be able to write shell scripts. A shell script is just a file full of Linux commands that you can run all at once.

Start simple. Automate the things you do manually more than three times. A good rule: if you run the same five commands every morning, put them in a script.

> PRO TIP
>
> Learn to pipe commands together. cat access.log | grep 'ERROR' | awk '{print $1}' | sort | uniq -c | sort -rn will give you the top IPs generating errors. This single pipeline has saved me hours of manual analysis.

What to Learn in Linux for 2026

- **eBPF** — The modern way to do networking, security, and performance tracing in the kernel. Tools like Cilium use it for Kubernetes networking.
- **Systemd** — Modern Linux service management. Understand unit files.
- **cgroups and namespaces** — The underlying technology that makes containers work. You do not need to master this, but knowing it exists changes how you think about containers.

CHAPTER 03

03

Networking — Because Packets Have to Get Somewhere

Networking essentials every DevOps engineer must know — fundamentals, CIDRs, Kubernetes.

IN THIS CHAPTER

8 sections, lists & callouts

APEX IT SOLUTIONS

Here is a confession: I avoided networking for the first year of my career. It seemed complicated, full of acronyms, and the diagrams in textbooks always made me want to take a nap.

Then I started working in telecom data centers, where networking is literally the entire point of the business. You learn fast when the alternative is looking incompetent in front of people who have been doing this since before you were born.

What Every DevOps Engineer Must Know

The Fundamentals

- **OSI Model** — Know what each layer does. You will reference it when debugging.
- **TCP/IP** — Understand handshakes, ports, and how connections are established
- **DNS** — How names resolve to IP addresses. More outages are caused by DNS than people admit.
- **HTTP/HTTPS** — Methods, status codes, headers. This is the language of APIs.
- **Load Balancing** — Round-robin, least connections, session persistence
- **Firewalls and Security Groups** — How traffic is allowed or blocked

Subnetting and CIDRs

When you are working with cloud VPCs and Kubernetes pod networks, you will encounter CIDR notation constantly. 10.0.0.0/16 means a network of 65,536 addresses. 10.0.1.0/24 means 256 addresses. Know this. It will come up in interviews and in production incidents.

Kubernetes Networking Specifically

- Every pod gets its own IP address
- Services abstract pods behind a stable IP and DNS name
- Ingress controllers route external traffic into the cluster
- Network policies control which pods can talk to each other
- CNI plugins (Calico, Flannel, Cilium) handle the actual networking layer

> **REAL INCIDENT STORY**
>
> Early in my career I was troubleshooting why two microservices could not communicate inside a Kubernetes cluster. I spent an hour checking application logs. The issue? A network policy I had applied earlier was blocking the traffic. The application was fine. The network said no. Always check the network before blaming the app.

CHAPTER 04

04

Git and Version Control — If It's Not in Git, It Doesn't Exist

Git and version control — daily commands, branching, and modern GitOps workflows.

IN THIS CHAPTER

9 sections, lists & callouts

APEX IT SOLUTIONS

There is a saying in modern DevOps: if it is not in Git, it does not exist. This applies to application code, infrastructure definitions, CI/CD pipelines, Kubernetes manifests, and configuration files.

Git is non-negotiable. Learn it properly.

The Commands You Will Use Daily

- **git clone** — Get a copy of a repository
- **git add / git commit -m 'message'** — Stage and save changes
- **git push / git pull** — Send and receive from remote
- **git branch / git checkout -b feature/name** — Create and switch branches
- **git merge / git rebase** — Combine branches (and occasionally cause drama)
- **git log --oneline** — See history quickly
- **git stash** — Save work temporarily without committing
- **git revert / git reset** — Undo things (know the difference)

Branching Strategies

For teams, having a branching strategy matters a lot. The most common ones you will encounter:

- **GitFlow** — main, develop, feature, release, hotfix branches. Structured but complex.
- **Trunk-Based Development** — Everyone commits to main frequently. Requires good CI. Preferred in 2026.
- **GitHub Flow** — Simple: main plus short-lived feature branches. Widely used.

GitOps — The Modern Way to Deploy

GitOps is the practice of using Git as the single source of truth for your infrastructure and deployments. When you push a change to Git, an automated system (like Argo CD or Flux) detects the change and applies it to your cluster.

This is not just convenient. It gives you an automatic audit trail, easy rollbacks, and means no one is SSHing into production to make manual changes. In 2026, GitOps is considered a best practice for Kubernetes deployments.

> **GITOPS IN ONE SENTENCE**
>
> You want 3 replicas? Put it in Git. Argo CD sees it. Argo CD makes it so. You want to roll back? Revert the commit. Done.

CHAPTER 05

05

Containers and Docker — The Box That Changed Everything

Containers and Docker — core concepts, Dockerfiles, and 2026 best practices.

IN THIS CHAPTER

8 sections, lists & callouts

APEX IT SOLUTIONS

Before containers, deploying an application meant: make sure the right version of Python is installed, make sure these environment variables are set, make sure this library is present, and pray that the production server is configured the same way as the development laptop.

It usually was not.

Docker changed this by packaging the application and everything it needs to run into a single portable unit called a container. The same container that works on your laptop will work on the server. This sounds simple. It is revolutionary.

Core Docker Concepts

- **Image** — A blueprint for a container. Think of it like a class in programming.
- **Container** — A running instance of an image. Think of it like an object.
- **Dockerfile** — The recipe for building an image
- **Docker Hub / Container Registry** — Where images are stored and shared
- **Volume** — Persistent storage that survives container restarts
- **Network** — How containers communicate with each other

A Dockerfile That Makes Sense

A basic Dockerfile for a Node.js application looks like this:

FROM node:20-alpine WORKDIR /app COPY package*.json ./ RUN npm install COPY . . EXPOSE 3000 CMD ["node", "app.js"]

Docker Best Practices in 2026

- Use multi-stage builds to keep image sizes small
- Never run containers as root in production
- Scan your images for vulnerabilities using tools like Trivy
- Use specific image tags, never just 'latest' in production
- **Keep images minimal** — alpine-based images are much smaller
- Store secrets in environment variables or secret managers, never baked into the image

> **STORY FROM THE FIELD**
>
> At one of the telecom clients, we had a container image that was 4.2 GB. It included build tools, development dependencies, and frankly some things nobody could explain. By switching to multi-stage builds, we got it to 180 MB. Deployments that took 12 minutes started completing in under 2 minutes. The developers were suspicious that something had gone wrong.

CHAPTER 06

06

Kubernetes — The Operating System of the Cloud

Kubernetes — workloads, networking, configuration, scaling, Helm, and senior insights.

IN THIS CHAPTER

16 sections, lists & callouts

APEX IT SOLUTIONS

Let me be direct: Kubernetes has a steep learning curve. It will feel overwhelming at first. This is normal and experienced. Every DevOps engineer who tells you Kubernetes was easy at first is either lying or has a very short memory.

But once it clicks — and it will click — you will understand why it has become the standard for running applications at scale. I have built Kubernetes clusters from bare metal in telecom data centers. I have provisioned EKS clusters on AWS for fintech applications. I have woken up at 3 AM to debug pod scheduling issues. Kubernetes and I have a complicated relationship. But I would not trade it.

What Kubernetes Does

Kubernetes orchestrates containers. It decides where to run them, restarts them when they crash, scales them when load increases, updates them without downtime, and provides networking and storage for them. It turns a pool of servers into a single, managed platform.

The Key Concepts

Workloads

- **Pod** — The smallest deployable unit. One or more containers that share networking and storage.
- **Deployment** — Manages a set of identical pods. Handles rolling updates and rollbacks.
- **StatefulSet** — Like a Deployment but for stateful applications that need stable identities (databases).
- **DaemonSet** — Runs one pod on every node (used for monitoring agents, log collectors).
- **Job / CronJob** — Run pods once or on a schedule.

Networking

- **Service** — A stable endpoint that routes traffic to pods (ClusterIP, NodePort, LoadBalancer).
- **Ingress** — Routes external HTTP traffic into the cluster based on rules.
- **NetworkPolicy** — Controls which pods can communicate with each other.

Configuration

- **ConfigMap** — Store non-sensitive configuration data.
- **Secret** — Store sensitive data like passwords and API keys (encrypted at rest).
- **Namespace** — Virtual cluster isolation within a cluster.

Scaling and Availability

- **HorizontalPodAutoscaler (HPA)** — Scales pods based on CPU/memory or custom metrics.
- **VerticalPodAutoscaler (VPA)** — Adjusts resource requests/limits automatically.
- **Karpenter** — Modern node autoscaler that provisions nodes just-in-time. A big deal in 2026.
- **PodDisruptionBudget** — Ensures minimum availability during maintenance.

Helm — Package Manager for Kubernetes

Helm lets you package, share, and deploy Kubernetes applications as charts. Instead of managing dozens of YAML files manually, you use a Helm chart that can be configured with a values file. Most popular open-source tools (Prometheus, Grafana, cert-manager) are distributed as Helm charts.

What Senior Engineers Know That Juniors Often Miss

- **Resource requests and limits** — Always set them. Pods without limits can eat all the node's memory.
- **Liveness and readiness probes** — Kubernetes needs to know when your app is healthy.
- **RBAC** — Role-Based Access Control. Who can do what in your cluster.
- **Cluster upgrade strategy** — Nodes must be drained carefully before upgrading.
- **etcd** — The brain of Kubernetes. Back it up. Regularly.

> **HARD LESSON LEARNED**
>
> I once deployed a new version of an application to a production Kubernetes cluster without readiness probes configured. Kubernetes saw the pods starting and started sending traffic to them before the app was ready. Users got errors for two minutes. Two minutes does not sound long until it is your production environment. Always configure your probes.

CHAPTER 07

07

CI/CD Pipelines — Making Deployment Boring (In a Good Way)

CI/CD pipelines — the stages, the tools landscape in 2026, and what makes one good.

IN THIS CHAPTER

14 sections, lists & callouts

APEX IT SOLUTIONS

The goal of a CI/CD pipeline is to make deploying software so automated, so reliable, and so well-tested that it becomes boring. A boring deployment is a successful deployment.

CI stands for Continuous Integration: developers merge their code frequently and automated systems verify that it works. CD stands for Continuous Delivery or Continuous Deployment: the tested code is automatically prepared for and sometimes automatically pushed to production.

The Stages of a Pipeline

- **Source** — Code is pushed to Git. This triggers the pipeline.
- **Build** — The code is compiled or packaged into a deployable artifact.
- **Test** — Automated tests run: unit tests, integration tests, security scans.
- **Scan** — Containers are scanned for vulnerabilities. SAST tools analyze the code.
- **Push** — The built artifact or container image is pushed to a registry.
- **Deploy to Staging** — The application is deployed to a non-production environment.
- **Integration Tests** — End-to-end tests run against the staging environment.
- **Deploy to Production** — Manual or automatic promotion to production.

The Tools Landscape in 2026

GitHub Actions — The Current Standard

GitHub Actions is the dominant choice for teams using GitHub. Pipelines are defined in YAML files inside your repository. It integrates naturally with the rest of your codebase and has a massive marketplace of pre-built actions.

GitLab CI/CD — Powerful and Self-Contained

If your team uses GitLab, its built-in CI/CD is excellent. One platform for code, issues, pipelines, and container registry.

Argo CD — GitOps for Kubernetes

Argo CD watches your Git repository and automatically syncs your Kubernetes cluster to match. It provides a beautiful UI showing the difference between what is in Git and what is running. If they differ, it will reconcile them.

Jenkins — The Veteran

Jenkins has been around for a long time and is installed everywhere. In 2026, it is considered legacy for new projects, but you will encounter it because it is so widespread. Know the basics.

> **THE JENKINS EXPERIENCE**
>
> My first Jenkins setup was a self-hosted instance on a server that also ran three other services, had not been updated in two years, and had plugins that conflicted with each other in ways nobody fully understood. We migrated to GitHub Actions. The pipelines ran twice as fast. Nobody missed Jenkins. Almost nobody.

What Makes a Good Pipeline in 2026

- **Pipeline as Code** — Your pipeline definition lives in the repository alongside the application code
- **Fail Fast** — Run quick tests first. Do not spend 20 minutes on heavy tests only to fail on a syntax error.
- **Parallel Stages** — Run independent stages at the same time
- **Immutable Artifacts** — Build once, deploy the same artifact everywhere
- **Secrets Management** — Never hardcode credentials. Use vault or cloud secret managers.
- **Rollback Strategy** — If production breaks, you need to be able to roll back in under 5 minutes

CHAPTER 08

08

Infrastructure as Code — Your Data Center in a Text File

Infrastructure as Code — Terraform, Ansible, OpenTofu, and the modern IaC discipline.

IN THIS CHAPTER

13 sections, lists & callouts

APEX IT SOLUTIONS

Before Infrastructure as Code, provisioning a server meant: submit a ticket, wait three days, receive access, manually configure everything, hope you remember what you did six months later when something breaks.

Infrastructure as Code (IaC) means you define your infrastructure — servers, databases, networks, load balancers — in code files that can be version controlled, reviewed, tested, and applied automatically.

Terraform — The Industry Standard

Terraform by HashiCorp is the most widely used IaC tool. It uses a declarative language called HCL (HashiCorp Configuration Language). You describe what you want, and Terraform figures out how to get there.

A simple example: declaring an AWS EC2 instance in Terraform is just a few lines of code. Terraform will create it, track its state, and know what changes to apply if you modify the configuration.

Core Terraform Workflow

- **terraform init** — Initialize the working directory
- **terraform plan** — Preview changes before applying
- **terraform apply** — Apply the changes to the real infrastructure
- **terraform destroy** — Remove all resources (do this carefully)

Terraform Best Practices

- **Use remote state** — Store state in S3 or Terraform Cloud, not locally
- **State locking** — Prevent multiple people from running Terraform simultaneously
- **Modules** — Reusable components for common patterns (VPC, EKS cluster, etc.)
- **Workspaces** — Manage multiple environments (dev, staging, production)
- **Use Infracost** — See the cost of your infrastructure changes before applying them

Ansible — Configuration Management and Automation

While Terraform provisions infrastructure, Ansible configures it. Ansible connects to servers over SSH and executes playbooks — YAML files that describe the desired state of the system.

I have used Ansible extensively across telecom data centers to configure bare-metal servers, install software, manage users, and deploy applications consistently across dozens of machines simultaneously.

- **Agentless** — No software needs to be installed on target servers
- **Idempotent** — Running the same playbook twice gives the same result
- **Large module library** — Thousands of pre-built modules for almost anything

OpenTofu and What is Coming

In 2024, Terraform changed its license in a way that concerned the open-source community. OpenTofu is the open-source fork of Terraform that maintains the original MPL license. The syntax is identical. In 2026, organizations are evaluating both. Know that OpenTofu exists.

CHAPTER 09

09

Cloud Platforms — AWS, Azure, and GCP

Cloud platforms — what you must know on any cloud, plus AWS for DevOps in 2026.

IN THIS CHAPTER

13 sections, lists & callouts

I have worked across all three major cloud platforms. They are more similar than their marketing materials suggest. Each has its strengths, its quirks, and its way of charging you more money than you expected.

The truth is: if you understand the concepts — virtual machines, managed Kubernetes, object storage, networking, IAM — on one cloud, you can transfer that understanding to any cloud within a few weeks. The tools differ. The fundamentals do not.

What You Need to Know on Any Cloud

Compute

- **Virtual Machines** — EC2 on AWS, VMs on Azure, Compute Engine on GCP
- **Managed Kubernetes** — EKS (AWS), AKS (Azure), GKE (GCP). These are the most important managed services for DevOps.
- **Serverless** — Lambda, Azure Functions, Cloud Run. Run code without managing servers.

Storage

- **Object Storage** — S3 (AWS), Blob Storage (Azure), GCS (GCP). Store files, backups, artifacts.
- **Block Storage** — EBS (AWS), Managed Disks (Azure). Attach to VMs like a hard drive.
- **Managed Databases** — RDS, Aurora, Cloud SQL. Databases without managing the server.

Networking

- **VPC** — Virtual Private Cloud. Your isolated network in the cloud.
- **Security Groups / NSGs** — Firewall rules for your cloud resources.
- **Load Balancers** — ALB, NLB on AWS. Distribute traffic across instances.
- **CDN** — CloudFront, Azure CDN. Deliver content close to users globally.

Identity and Access (Critical)

IAM — Identity and Access Management — is where security begins in the cloud. Always apply the principle of least privilege: give each service and person the minimum permissions they need to do their job and nothing more.

- **Roles over Users** — Assign permissions to roles, not individual users
- **Service Accounts** — For applications that need cloud permissions
- **MFA** — Multi-factor authentication for all human accounts
- **Audit Logs** — CloudTrail (AWS), Activity Log (Azure). Know what happened and who did it.

AWS in 2026 — The Services That Matter Most for DevOps

- **EKS** — Elastic Kubernetes Service. The main Kubernetes offering.
- **EC2** — Virtual machines. You will still use these.
- **RDS / Aurora** — Managed relational databases.
- **S3** — Object storage. Also used for Terraform state, artifact storage.
- **IAM** — Access control. Get this wrong and you have security problems.

- **VPC** — Network isolation. Understand subnets, route tables, NAT gateways.
- **CloudWatch** — Logs and basic metrics. Often paired with a more capable solution.
- **ECR** — Elastic Container Registry. Where your Docker images live.
- **Secrets Manager** — Store and rotate secrets securely.

> **CLOUD COST HORROR STORY**
>
> A developer at a fintech project I worked on accidentally left a test cluster running over a long weekend. With NAT gateway traffic and data transfer charges, the weekend cost more than the entire month's budget had planned for. Always set billing alerts. Always.

CHAPTER 10

10

Monitoring and Observability — You Can't Fix What You Can't See

Monitoring and observability — the three pillars and what to actually alert on.

IN THIS CHAPTER

14 sections, lists & callouts

Monitoring is not just about knowing when things break. It is about understanding your system deeply enough to predict when things will break before they do. That shift — from reactive to proactive — is what separates average operations from excellent ones.

In 2026, observability has expanded beyond monitoring. The three pillars of observability are metrics, logs, and traces. Together, they give you a complete picture of your system.

The Three Pillars

Metrics — What is Happening Right Now

Metrics are numerical measurements over time: CPU usage, memory consumption, request rate, error rate, latency. They are aggregated and efficient to store. They tell you that something is wrong but often not exactly why.

- **Prometheus** — Open-source metrics collection and storage. Industry standard.
- **Grafana** — Visualize metrics from Prometheus and other sources. Dashboard heaven.
- **CloudWatch, Azure Monitor, Google Cloud Monitoring** — Cloud-native metric services.

Logs — What Actually Happened

Logs are timestamped records of discrete events. They give you context and detail about specific occurrences. The challenge with logs is volume — modern applications generate enormous amounts of log data.

- **ELK Stack** — Elasticsearch, Logstash, Kibana. Powerful and widely used.
- **Loki** — Lighter-weight log aggregation from Grafana Labs. Works well with Kubernetes.
- **Fluentd / Fluent Bit** — Log collectors and forwarders.

Traces — How a Request Moved Through Your System

Distributed tracing follows a single request as it moves through multiple services. When a request fails or is slow, traces show you exactly which service caused the problem and how long each step took. This is essential for microservices architectures.

- **OpenTelemetry (OTel)** — The emerging standard for instrumentation. Learn this in 2026.
- **Jaeger, Zipkin** — Open-source tracing backends.
- **Datadog, Dynatrace, New Relic** — Commercial observability platforms with all three pillars.

What to Alert On

Alert fatigue is real. If every minor fluctuation triggers a 3 AM page, engineers start ignoring alerts. This is dangerous. Alert on symptoms, not causes.

- Error rate above threshold (e.g., more than 1% of requests returning 5xx errors)
- Latency above SLA (e.g., p99 response time above 2 seconds)
- Availability drop (e.g., service health check failing)
- **Disk usage above 80%** — This one bites you if you ignore it
- **Pod restart loops in Kubernetes** — Something is crashing repeatedly

THE ALERT THAT WOKE EVERYONE UP

At a production environment I managed, we had an alert for disk usage set to 95%. That sounds safe. But the disk was filling at 2% per day and nobody noticed the trend on the graph. At 94.8%, the database could not write. Brief system downtime. After that, I set a predictive alert: 'if current trend continues, disk will be full in 7 days.' That kind of alert gives you time to fix things during business hours.

CHAPTER 11

11

DevSecOps — Security Is Not a Phase, It's the Whole Thing

DevSecOps — security in the pipeline, Kubernetes, and the software supply chain.

IN THIS CHAPTER

7 sections, lists & callouts

There used to be a practice where developers built software, operations deployed it, and then security came in at the end and said no. This did not work well for anyone.

DevSecOps integrates security into every phase of the software development and deployment lifecycle. 'Shift left' means moving security checks earlier in the process — during development and CI/CD — rather than waiting until deployment.

Security in Your Pipeline

- **SAST** — Static Application Security Testing. Scan code for vulnerabilities before it runs. Tools: SonarQube, Semgrep.
- **DAST** — Dynamic Application Security Testing. Test the running application for vulnerabilities.
- **Container Scanning** — Scan Docker images for known CVEs. Tools: Trivy, Snyk, Grype.
- **Secret Scanning** — Detect accidentally committed passwords and API keys. GitHub has this built in now.
- **Dependency Scanning** — Check for vulnerable third-party libraries.

Kubernetes Security

- **RBAC** — Least privilege for all service accounts and users
- **Network Policies** — Control which pods can talk to each other
- **Pod Security Standards** — Prevent privileged containers and other risky configurations
- **Secrets Encryption at Rest** — Encrypt Kubernetes secrets in etcd
- **Image signing with Cosign** — Verify the authenticity of container images
- **Kyverno or OPA Gatekeeper** — Policy enforcement in Kubernetes

Supply Chain Security — The Emerging Priority

In 2026, software supply chain security has become a major focus. The question is: can you trust all the code and dependencies that go into your application? SBOMs — Software Bills of Materials — enumerate every component in your software, making it possible to quickly identify whether a new vulnerability affects you.

- **Sigstore / Cosign** — Sign and verify container images
- **SBOM generation** — Tools like Syft can generate a full inventory of your container's contents
- **Snyk** — Commercial tool for scanning dependencies, containers, and IaC

CHAPTER 12

12

AI in DevOps — The Biggest Shift Since Containerization

AI in DevOps — AIOps, LLMs in pipelines, LLMOps, platform engineering, and the skills that matter.

IN THIS CHAPTER

21 sections, lists & callouts

I want to spend real time on this chapter because I believe it represents the most significant change to the DevOps profession in a decade. Not because AI is going to replace engineers — it is not — but because engineers who use AI effectively will increasingly out-perform those who do not.

When I started my career, the idea of a system automatically diagnosing its own failures and writing the remediation steps was science fiction. In 2026, it is a product you can buy. Understanding what these tools do, and where they still need human judgment, is now a core competency.

AIOps — Artificial Intelligence for IT Operations

AIOps platforms use machine learning to analyze the enormous volumes of telemetry data — logs, metrics, traces, events — that modern infrastructure generates. The goal is to detect anomalies, correlate related signals, identify root causes, and increasingly, automate remediation.

- **Anomaly Detection** — AI learns what 'normal' looks like and alerts you to deviations, even subtle ones that rule-based systems would miss
- **Alert Correlation** — Instead of 200 individual alerts during an incident, get one grouped incident showing the cause and its downstream effects
- **Root Cause Analysis** — AI analyzes the pattern of failures to identify the most likely source
- **Predictive Maintenance** — Forecast when disks will fill, when traffic spikes will overwhelm capacity, before they happen

Tools in this space include Dynatrace (with its Davis AI engine), New Relic, Datadog's Watchdog, PagerDuty AIOps, and IBM Watson AIOps.

> **WHY THIS MATTERS FOR YOU**
>
> I have seen AIOps systems reduce mean time to resolution on incidents by 60-70% compared to manual triage. The system does not replace the engineer. It hands the engineer a diagnosis instead of a stack of alerts. The engineer still decides what to do. But they spend minutes understanding the problem instead of hours.

LLMs in Your Pipelines

Large Language Models are being integrated into CI/CD pipelines in increasingly practical ways:

- **Automated Code Review** — LLMs review pull requests, suggest improvements, flag potential bugs
- **IaC Generation** — Describe what you want in plain English, get Terraform or Kubernetes YAML back
- **Runbook Generation** — Automatically draft operational runbooks from incident data
- **Incident Summarization** — During an outage, an LLM can ingest logs, metrics, and alerts and produce a concise summary for stakeholders
- **Pipeline Optimization** — Analyze historical CI run data to identify bottlenecks and suggest improvements

AI-Assisted Infrastructure as Code

Tools like GitHub Copilot and Amazon Q have made writing Terraform, Ansible, and Kubernetes manifests significantly faster. In my experience, they are most valuable for boilerplate and for helping you remember the exact syntax of something you use occasionally. They still need review — they hallucinate, they suggest outdated patterns, and they do not know your specific environment.

The practical advice: use them as a starting point. Always review what they produce. Understand what you are deploying.

LLMOps — Running AI in Production

If your organization deploys AI applications — chatbots, recommendation systems, AI assistants — you need to think about how to operate them in production. This is LLMOps.

- **Model versioning** — Track which version of a model is running in production
- **Prompt management** — Version control your prompts the way you version control code
- **Inference serving** — Deploy models efficiently using tools like TorchServe, Triton, or vLLM on Kubernetes
- **Cost management** — LLM inference is expensive. Track token usage, optimize prompts, use caching.
- **Quality monitoring** — Unlike traditional software, you need to monitor the quality of model outputs, not just latency and errors

Platform Engineering and Internal Developer Platforms

Platform engineering is the practice of building internal tools and platforms that help developers deploy and operate applications without needing deep DevOps expertise. In 2026, this is one of the hottest areas of the field.

Backstage (from Spotify, now a CNCF project) is the most popular framework for building Internal Developer Platforms. It provides a service catalog, documentation portal, and plugin system that teams use to create self-service infrastructure provisioning.

- Developers deploy applications without writing Kubernetes YAML
- Platform teams build the guardrails that ensure security and compliance
- AI is being integrated into IDPs to generate configurations and answer developer questions

The Skills That Matter in an AI World

Here is my honest assessment of what the AI wave means for your career:

- **Understanding the fundamentals matters more, not less** — AI tools produce output you cannot evaluate if you do not understand the underlying systems
- **Prompt engineering for infrastructure** — Learn to describe what you want precisely enough that AI tools produce useful results
- **Evaluation and validation** — The ability to critically assess AI-generated code and configuration is now a core skill

- **System thinking** — AI handles individual tasks. You need to understand how systems fit together.
- **Cost awareness** — AI inference, cloud resources, and modern tooling are expensive. FinOps thinking is increasingly valued.

CHAPTER 13

13

Your DevOps Roadmap for 2026 — Where to Start and What to Learn

A DevOps roadmap for 2026 — a phased plan from fundamentals to specialisation.

IN THIS CHAPTER

11 sections, lists & callouts

Everything we have covered is a lot. Let me now give you a practical, ordered path.

Phase 1: The Foundation (Months 1-3)

Do not touch Kubernetes yet. Build the foundation.

- **Linux fundamentals** — File system, permissions, process management, networking commands
- **Shell scripting** — Write scripts that automate repetitive tasks
- **Git** — Commit, branch, merge, rebase. Actually understand them, do not just copy commands.
- **Networking basics** — TCP/IP, DNS, HTTP, subnets
- **At least one cloud** — Create an AWS free tier account. Build things. Break things. Understand the billing dashboard.

Phase 2: Automation and Containers (Months 3-6)

- **Docker** — Build images, run containers, write Dockerfiles, use Docker Compose
- **CI/CD basics** — Set up a GitHub Actions pipeline that builds and tests your code
- **Terraform basics** — Provision cloud resources with code
- **Ansible basics** — Configure servers with playbooks

Phase 3: Kubernetes and Advanced Pipelines (Months 6-12)

- **Kubernetes fundamentals** — Pods, Deployments, Services, ConfigMaps, Secrets
- **Helm** — Deploy applications using charts
- **Advanced CI/CD** — Multi-stage pipelines, artifact management, deployment strategies
- **Monitoring** — Set up Prometheus and Grafana. Write alerts.
- **GitOps** — Deploy a simple application using Argo CD

Phase 4: Advanced Topics and Specialization (Year 2 and Beyond)

- **DevSecOps** — Integrate security scanning into your pipelines
- **Service mesh** — Istio or Linkerd for mTLS and observability
- **Platform engineering** — Understand the Internal Developer Platform concept
- **AIOps tools** — Explore Dynatrace, Datadog, or New Relic. Understand what AI monitoring looks like.
- **LLM deployment** — Deploy an open-source model on Kubernetes. Understand the infrastructure requirements.
- **FinOps** — Learn to track and optimize cloud costs

The 2026 Tech Radar at a Glance

CATEGORY	LEARN NOW	MASTER THIS	DE-PRIORITIZE
CI/CD	Dagger, GitOps	GitHub Actions	Jenkins (self-hosted)
Cloud IaC	Crossplane	Terraform / OpenTofu	Manual console clicks
Compute	WebAssembly Edge	Kubernetes	Bare metal management
Observability	OpenTelemetry	Prometheus / Grafana	Proprietary agents only
Security	Cosign / Kyverno	Trivy / Snyk	Manual security audits
Scaling	Karpenter	Cluster Autoscaler	Fixed server fleets

CHAPTER 14

14

Certifications — Worth It, or Just Expensive Paper?

Certifications — which ones are worth your time in 2026, and how to study effectively.

IN THIS CHAPTER

7 sections, lists & callouts

Honest answer: it depends on which ones.

Certifications do not make you a good engineer. Hands-on experience does. But certifications signal to employers that you have studied a domain seriously, and they open doors — especially when you are earlier in your career and do not yet have the work history to speak for itself.

Certifications Worth Your Time in 2026

Cloud Provider Certifications

- **AWS Certified Solutions Architect** — Associate — The most recognized cloud certification globally. A solid starting point.
- **AWS Certified DevOps Engineer** — Professional — More advanced. Valuable once you have real experience.
- **CKA** — Certified Kubernetes Administrator — Hands-on exam, no multiple choice. Highly respected. Difficult. Worth every rupee.
- **CKAD** — Certified Kubernetes Application Developer — Developer perspective on Kubernetes.
- **Azure Administrator Associate / DevOps Expert** — If your target employers use Azure.

Security Certifications

- **CKS** — Certified Kubernetes Security Specialist — Advanced. Do the CKA first.
- **AWS Security Specialty** — If you want to specialize in cloud security.

How to Study Effectively

- **Build labs, not just watch videos** — Set up real environments on your laptop or cloud free tier
- **Killer.sh for CKA/CKAD** — The best practice exam simulator. Harder than the real thing on purpose.
- **KodeKloud, LinuxFoundation, A Cloud Guru** — Good course platforms
- **Read the official documentation** — Not exciting, but it is authoritative

CHAPTER 15

15

Getting Hired — What Employers Actually Look For

Getting hired — what your resume should show, what to build, and interview topics.

IN THIS CHAPTER

7 sections, lists & callouts

This chapter exists because I have interviewed candidates and I have been interviewed. Both sides of the table have taught me things.

What Your Resume Should Show

- **Specific technologies with context** — Not just 'Kubernetes' but 'Managed Kubernetes clusters serving 500k daily requests'
- **Automation impact** — 'Reduced deployment time from 45 minutes to 8 minutes by implementing CI/CD pipeline'
- **Incidents resolved** — 'Diagnosed and resolved production database connection pool exhaustion affecting 12k users'
- **Scale** — Numbers. How many servers, how much traffic, how many teams did your platform serve

What You Should Build to Show Skills

- **A personal project with a full CI/CD pipeline** — Code on GitHub, automated pipeline, deployed to a cloud
- **A Kubernetes cluster** — Even a simple one on your laptop with kind or minikube. Know how to deploy, scale, and troubleshoot it.
- **A monitoring stack** — Set up Prometheus and Grafana for something. Show the dashboards.
- **Infrastructure as Code** — Your AWS or GCP setup should be in Terraform, not clicked together

Common Interview Topics

- Explain how a request flows from a user's browser to a Kubernetes pod
- How would you troubleshoot a pod that is stuck in CrashLoopBackOff?
- Describe a CI/CD pipeline you would design for a microservices application
- What is the difference between a Deployment and a StatefulSet?
- How do you handle secrets in Kubernetes?
- What does blue/green deployment mean, and when would you use it over canary?

> **INTERVIEW ADVICE**
>
> When you get a troubleshooting question, think out loud. Interviewers are not just testing whether you know the answer. They are watching how you approach problems. Say 'I would first check the pod logs, then look at events, then check if the image is pulling correctly, then look at resource limits.' Show your thinking.

EPILOGUE

Final Word

The Mindset That Matters Most

Closing reflections — the mindset that makes a great DevOps engineer.

CLOSING REFLECTION

The mindset that matters most

You can memorize every command in this guide and still not become a good DevOps engineer. The technical knowledge is necessary but not sufficient.

What separates the engineers I respect most from the ones who struggle is a specific mindset:

- **Automation instinct** — When you do something manually twice, you start thinking about how to automate it
- **Systems thinking** — You ask 'why' not just 'how.' You want to understand how pieces fit together.
- **Comfort with ambiguity** — Production issues are rarely clear. You need to investigate without panicking.
- **Documentation habit** — You write things down. Future you and your teammates will thank you.
- **Continuous learning** — This field changes faster than almost any other. You never fully arrive.

When I started my journey, I had none of these. I developed them through years of mistakes, late nights, and genuinely caring about the systems I was responsible for.

You are starting with something I did not have: a clear map of the territory. Use it well.

The terminals are waiting.

Need Guidance? Let's Talk.

Look, I know this guide covers a lot of ground. If you are sitting there wondering where to actually begin, what certification makes sense for your situation, how to structure a lab environment, or whether your resume is going to get past the first filter — those are exactly the kinds of conversations I enjoy.

I run Apex IT Solutions with a team that does real DevOps work for real companies. We also spend time helping engineers at the beginning of their careers figure out their path. Not as a charity — as a genuine interest in the quality of the people entering this field.

If you want to talk, reach out. No complicated process. Just a conversation.

Email: sohaib@apexitsolutions.co

Website: www.apexitsolutions.co

DevOps Resources: devops.apexitsolutions.co

The DevOps Field Guide 2026 | Apex IT Solutions