Max’s DevOps Deep-Dive Roadmap — No Cheat Sheets

You said you want to dig deep and become unstoppable. This plan is hands-on, outcome-driven, and built around **production-grade skills**. The rhythm is: **Read fast → Build → Break → Fix → Explain**.

How to use this roadmap

- **Daily cadence (90–150 mins):** 20% reading, 60% building, 20% journaling + explaining.
- **Proof of work:** One repo: `realred-devops/` with subfolders per track. Every day ends with a commit + README note.
- **TDD for DevOps:** Before touching keys, write a tiny *checklist* and a *success metric*.
- **Explain like SRE:** Close each day with a 5-line postmortem: *What I did, Risks, Observability, Rollback, Next*.

Repo scaffold:

```
realred-devops/
  00-notes/
  10-linux-git/
  20-docker/
  30-kubernetes/
  40-terraform/
  50-ansible/
  60-cicd/
  70-observability/
  80-chaos/
  90-capstone-realred/
```

Day 0 – Set up the arena (today)

Goal: A reproducible local lab that mirrors production primitives.

1) **Install & verify** - Docker, `kubectl`, `kind` or minikube, Helm - Terraform, Ansible, Git, `gh` (GitHub CLI)

2) **Create a local K8s cluster**

```
kind create cluster --name realred --config - <<'YAML'
kind: Cluster
```

```
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
- role: worker
- role: worker
YAML
kubectl get nodes
```

3) **Bootstrap repo**

```
mkdir -p realred-devops/{00-notes,10-linux-git,20-docker,30-kubernetes,40-
terraform,50-ansible,60-cicd,70-observability,80-chaos,90-capstone-realred}
cd realred-devops && git init
```

4) **Success criteria** - `kubectl get nodes` shows 3 nodes. - Repo initialized with README.

Commit your Day 0 README with the success checks.

Days 1–3 — Linux & Git that never fail

Outcomes: You can trace a prod issue from process → socket → file → container.

Build/Break/Fix drills - Write a `troubleshoot.sh` that, given a PID/port, prints owning process, open files, remote endpoints, and top CPU/mem. - Git: create a branch protection simulation locally (no force push, required reviews via hooks). Add a pre-commit hook that lints YAML and forbids `latest` tags.

Commands to master - `ss -tulpen`, `lsof -p`, `strace -p`, `journalctl -u`, `top/htop`, `iostat`, `vmstat`, `free -m`, `dig`, `curl -v`, `ip route`. - Git: `reflog`, `bisect`, `worktree`, `cherry-pick`, `rebase -i`.

Explain: Write a 10-line note on how you'd debug a 99th percentile latency spike after a deploy.

Days 4–6 — Docker, images, and supply-chain safety

Outcomes: Deterministic builds, tiny images, fast rollbacks.

Labs 1) **Multi-stage build** for a simple FastAPI service. Target final image <120MB. 2) **Image provenance**: Tag images `realred/api:v1.0.0`, sign with `cosign` (optional), push to registry (local `registry:2` container or Docker Hub). 3) **Rollback drill**: Deploy `v1.0.1` with a deliberate bug, detect via healthcheck failure, and redeploy `v1.0.0`.

Snippets

```
# Dockerfile
FROM python:3.12-slim AS build
WORKDIR /app
COPY pyproject.toml poetry.lock ./
RUN pip install --no-cache-dir uv && uv pip install -r <(uv pip compile -q
pyproject.toml) --system
COPY . .
RUN python -m compileall .

FROM python:3.12-slim
WORKDIR /app
COPY --from=build /app /app
ENV PORT=8000
HEALTHCHECK CMD curl -f http://localhost:$PORT/health || exit 1
CMD ["python","-m","app"]
```

Checks - `docker history` shows small layers. - `docker run` healthcheck flips `healthy → unhealthy` when you break the app.

Days 7–10 — Kubernetes that you can trust

Outcomes: Safe rollouts, instant rollbacks, zero-downtime switches.

Core Labs 1) **Rolling updates & undo**

```
kubectl create deploy rr-api --image=realred/api:v1.0.0 --port=8000
kubectl set image deploy/rr-api rr-api=realred/api:v1.0.1
kubectl rollout status deploy/rr-api
# simulate failure → then
kubectl rollout undo deploy/rr-api
kubectl rollout history deploy/rr-api
```

2) **Blue-Green** (two Deployments + one Service switch)

```
# service.yaml
apiVersion: v1
kind: Service
metadata: { name: rr-api }
spec:
```

```
  selector: { app: rr-api, track: blue }
  ports: [{ port: 80, targetPort: 8000 }]
```

```
# deploy-blue.yaml
apiVersion: apps/v1
kind: Deployment
metadata: { name: rr-api-blue }
spec:
  selector: { matchLabels: { app: rr-api, track: blue } }
  template:
    metadata: { labels: { app: rr-api, track: blue } }
    spec:
      containers: [{ name: api, image: realred/api:v1.0.0, ports:
[{containerPort:8000}] }]
```

```
# deploy-green.yaml (candidate)
apiVersion: apps/v1
kind: Deployment
metadata: { name: rr-api-green }
spec:
  selector: { matchLabels: { app: rr-api, track: green } }
  template:
    metadata: { labels: { app: rr-api, track: green } }
    spec:
      containers: [{ name: api, image: realred/api:v1.0.1, ports:
[{containerPort:8000}] }]
```

Switch: Patch the Service selector from `track: blue` → `track: green` and observe instant cutover + easy revert.

3) **Poor-man's Canary** (approximate split by replica counts) - Run `green` with 1 replica, `blue` with 9. Increase `green` gradually while watching errors/latency.

4) **Health, readiness, liveness** - Add probes, PodDisruptionBudget, and `maxUnavailable=0` for strict availability.

Deliverables: `30-kubernetes/blue-green/` with manifests + a README explaining tradeoffs.

Days 11–13 — Terraform: state is everything

Outcomes: Safe plans, module hygiene, bulletproof state.

Labs 1) **Remote backend with locking**: S3 + DynamoDB (or localstack). Enable versioning. 2) **Modules**: Create `vpc/`, `eks/`, `rds/` modules with input validation and outputs. Pin provider versions. 3) **Drift detection & rollback** - Drift: change a tag in console → `terraform plan` shows diff. - Rollback: revert by checking out previous commit and `terraform apply` (roll **forward** to known good).

Golden rules - Never `terraform apply` without a saved plan file: `terraform plan -out tf.plan && terraform apply tf.plan`. - **Rollback ≠ state surgery.** Prefer re-applying a previous Git tag.

Days 14–16 — Ansible: idempotence & safe changes

Outcomes: Predictable runs, controlled blast radius.

Labs 1) Roles + `site.yml` with `serial`, `max_fail_percentage`, and `strategy: free`. 2) Use `--check` + `--diff` in CI for dry-run gates. 3) Inventory: static vs dynamic; group_vars; vault secrets. 4) **Rollback pattern**: versioned artifacts + `releases/` symlink strategy (like Capistrano) to flip back instantly.

Snippet

```
- hosts: api
  serial: 2
  strategy: free
  tasks:
    - name: Deploy release
      unarchive:
        src: /artifacts/realred-api-{{ version }}.tar.gz
        dest: /opt/realred/releases/{{ version }}
        remote_src: yes
    - name: Switch current
      file:
        src: /opt/realred/releases/{{ version }}
        dest: /opt/realred/current
        state: link
      notify: restart
  handlers:
    - name: restart
      systemd:
        name: realred-api
        state: restarted
```

Days 17–18 — CI/CD that enforces quality

Outcomes: Build once, promote across stages, fast rollback.

Pipeline sketch (GitHub Actions) - Jobs: `build` → `scan` → `deploy-staging` → `smoke` → `promote-prod` (manual approval) → `deploy-prod`. - Artifacts: Docker image `realred/api:${{ github.sha }}` + Helm chart versioned. - **Rollback job**: input `revision_or_tag`; redeploy previous artifact; auto-comment with links to logs and metrics.

Key gates: - Unit tests, Dockerfile lint, trivy scan, `terraform plan` comment, Ansible `--check`, `kubectl diff`, canary success threshold.

Day 19 — Observability baseline

Outcomes: See issues before users do.

Stack: Prometheus + Alertmanager + Grafana; Loki (or filebeat + Elasticsearch).

Labs - Export app metrics (requests, errors, p99 latency). Create dashboards for RED (Rate, Errors, Duration) and SLI/SLO. - Alerts: high error rate on new deploy; alert links to the exact pipeline run + commit.

Day 20 — Chaos & incident response

Outcomes: Calm under fire.

Labs - Kill pods, add latency, fill disk (in a sandbox). Verify SLO alerts fire and rollback jobs trigger. - **Postmortem template (blameless)** - *Impact, Timeline, Root cause(s), Contributing factors, What went well, Action items (with owners & dates).*

Day 21 — Interview combat day

- 60-min **mock interview**: whiteboard “design a zero-downtime deploy pipeline with rollback”.
- 30-min **troubleshooting**: you fix a failing rollout using `kubectl rollout undo` and Service selector switches.
- 30-min **behavioral** using STAR with real incidents from your labs.

Deliver a 1-page portfolio README linking to your labs and the capstone.

Capstone — RealRed end-to-end

Goal: Ship a tiny service with a production-like pipeline and a rock-solid rollback.

Minimal scope - **App**: FastAPI `GET /health`, `GET /version` returning git SHA. - **Infra (Terraform)**: VPC + EKS (or local kind in CI as substitute). Remote backend with locking. - **Config (Ansible)**: OS hardening + runtime deps for a worker node or VM path. - **K8s**: Helm chart with values for blue/green, probes, HPA, PDB. - **CI/CD**: Build → scan → deploy-staging → promote → deploy-prod with rollback job. - **Observability**: Dashboards + alerts tied to releases.

What to show in interviews - GIF/screencast switching Service selector blue→green and back. - A runbook: "Rollback in ≤ 2 minutes".

Rollback strategy — deep dive (Docker, K8s, Terraform, Ansible)

Kubernetes - **Rolling back**: `kubectl rollout undo deployment/rr-api --to-revision=<n>` - **Blue/Green**: Service selector flip is O(seconds); rollback is the reverse patch. - **Readiness first**: only route to `Ready` pods; set `maxUnavailable: 0` for strict uptime.

Docker - Keep previous images: `realred/api:1.0.0`, `1.0.1`. Rollback = redeploy the last good tag. Never use `latest`.

Terraform - Rollback = **re-apply a previous git tag**. Do not edit state. Guard with saved plan + backend locking.

Ansible - Versioned artifacts + symlink `current → releases/<version>` switch. Rollback = repoint symlink + restart.

Daily "kata" (10–15 mins)

- Write one `kubectl` one-liner, one `jq` filter, and one `bash` function you didn't know yesterday.
- Translate a manual step into code (Makefile or script) every day.

Quick self-tests

1) Explain readiness vs liveness and how each prevents bad rollouts. 2) How do you recover from a bad Terraform apply that changed a security group? (Hint: re-apply previous tag; don't hand-edit state.) 3) Show a blue-green switch using only Service selector changes. 4) Prove your Docker image is reproducible and small. 5) Where do you add gates in CI to stop a risky deploy?

What “unstoppable” looks like

- You can ship a change and roll it back in < 2 minutes.
- You can describe your pipeline, observability, and rollback with logs/metrics links.
- You have a repo of labs that *prove* all of the above.

Now go build. Commit Day 0 in the next 60 minutes.