please update it with devops prerequisites which i need to learn before leatning devops and then update whole thing include what isdevops why is devops and all those things

Here's a complete, beginner-friendly guide that adds DevOps prerequisites and the "what/why" of DevOps, then walks through the same 6-week AWS-only plan with clear reasons, steps, and artifacts to keep. The tone is explanatory and step-by-step to match a preference for simple language and visual documentation.[1] [2]

What is DevOps?

- DevOps is a way of building and running software where development and operations work as one team, using automation, version control, CI/CD, infrastructure as code, and feedback loops to deliver reliable changes quickly.[2] [3]
- Instead of "big releases" with surprises, DevOps promotes small, frequent, tested deployments with strong monitoring and the ability to roll back safely.[3] [2]

Why DevOps matters

- It reduces risk and downtime by automating builds, tests, deployments, and infrastructure, so releases are repeatable and traceable.[2] [3]
- It improves reliability and cost control by measuring real behavior in production and reacting with scaling, alerts, and guardrails.[3] [2]

Prerequisites before DevOps

- Computer science basics
  - Know how programs run (processes, memory, files), how networks connect (IP, TCP/UDP, DNS), and how HTTP APIs work (methods, status codes, headers).[2] [3]
- Linux fundamentals
  - Files/permissions, users/groups, processes/services with systemd, key networking tools (ip, ss, dig), SSH keys and hardening.[3] [2]
- Scripting fundamentals
  - Bash for automation and logs; Python for CLIs, REST calls, and cloud SDKs; write small, tested scripts.[2] [3]
- Git and collaboration

    - Branching, pull requests, code reviews, and tagging; understand CI triggers on push and pull requests.[1] [3]
- Containers
    - Basic Dockerfile, images vs containers, multi-stage builds, environment variables, health checks.[4] [3]
- Cloud concepts
    - Regions/AZs, compute/storage/network, identity/permissions, high availability vs disaster recovery.[5] [3]
- Mindset
    - Automate repeatable steps, document decisions, measure outcomes, and design for rollback and failure.[1] [2]

6-week AWS-only roadmap (expressive, step-by-step)

- Learning format: each day has Build (do), Understand (read/diagram), Prove (artifact saved). This fits a clear, documented style.[1] [2]

Week 1: Linux, Bash, Git, Python

- Goal: be able to troubleshoot a server and write small automations.[3] [2]
- Build
    - Bash toolkit: log parser (grep/awk to count 4xx/5xx), port check (ss -ltn), network view (ip a, ip r), quick disk/mem checks (df, free). Add comments explaining each command.[6] [3]
    - Python CLI: argparse with “--logfile”; requests to call a public REST API; retry on 500s with backoff; two pytest tests.[2] [3]
- Understand
    - Services and logs: systemctl to start/enable, journalctl -u service; SSH key-only login and disable root login. Draw a simple request flow: client → DNS → server:port → service.[7] [2]
- Prove
    - GitHub Actions workflow runs lint and tests; README with screenshots and a one-line “why this matters” under each command.[1] [3]

Week 2: Containerization and image security

- Goal: ship a container image that starts fast and scans cleanly.[3] [2]
- Build
    - Wrap a FastAPI or Node app with a “/healthz” route; multi-stage Dockerfile to keep it small; docker-compose for local run; push to Amazon ECR.[8] [3]
    - Scan with Trivy; export SBOM; fix one high vulnerability by pinning a safer base image or updating packages.[8] [3]

- Understand
  - Health vs readiness checks; why container logs matter for quick root-cause; environment variables and secrets patterns.[2] [3]
- Prove
  - Publish the image tag, Trivy report, and a “before/after” note on the fix.[1] [3]

Week 3: AWS IaC + blue/green delivery

- Goal: create a secure, repeatable environment and deploy with near-zero downtime.[3] [2]
- Build
  - Terraform modules: VPC with two AZs; public subnets for ALB; private subnets for EC2; route tables with IGW and NAT; SGs for 80/443 to ALB and app port on instances.[5] [3]
  - Launch Template + Auto Scaling Group (min 2) to span AZs; user-data pulls the container and starts it; ALB with ACM TLS; Route 53 record to ALB.[5] [8]
  - CodePipeline: CodeBuild builds/pushes image; CodeDeploy blue/green replaces instances behind ALB; Manual Approval before traffic shift.[8] [3]
- Understand
  - Why private subnets protect instances; how ALB health checks drain bad targets; what ACM does for TLS.[5] [2]
- Prove
  - Architecture diagram, successful pipeline run with green checks, ALB target group health screenshot.[1] [3]

Week 4: Observability, security, and stable scaling

- Goal: detect issues early, recover fast, and keep access safe.[2] [3]
- Build
  - CloudWatch logs/metrics/dashboards for CPU, ALB 5xx, latency, and error count; alarm on high 5xx; test alarm by hitting a failing endpoint.[8] [3]
  - IAM least-privilege instance role (ECR pull, SSM read for params); Secrets Manager for API/DB secrets; rotation script scheduled by a simple cron/SSM.[8] [3]
  - Auto Scaling: target tracking on ALB request count; set warm-up and cooldown to stop thrashing; note how wrong values cause scale up/down loops.[9] [3]
- Understand
  - “Observe → Alert → Act” loop; where to look first for 502/504 (target health, app port, logs).[3] [2]
- Prove
  - Dashboard and alarm screenshots; a one-page troubleshooting playbook with commands and rollback steps.[1] [3]

Week 5: Testing, API quality, and traffic strategies

- Goal: prevent bad releases and roll out changes safely.[2] [3]
- Build
  - Postman collection with negative tests; run with Newman in CI to gate deployment; fail fast on contract breaks.[3] [1]
  - k6 smoke test to validate core endpoints at low RPS; publish HTML/JSON report as pipeline artifact.[1] [3]
  - Route 53 weighted records for canary (10/90); promote to 100% when dashboards and tests stay green; pair with health checks for auto-drain.[8] [3]
- Understand
  - Blue/green vs rolling vs canary; DNS TTL and rollback time; how health checks and weights work together.[2] [3]
- Prove
  - Screenshots of weights changing and health checks; a two-step rollback runbook: set weight back and redeploy previous artifact.[3] [1]

Week 6: Hardening, cost, DR, and interview polish

- Goal: show ownership: security posture, costs, recovery, and crisp explanations.[2] [3]
- Build
  - WAF rules on ALB/CloudFront to block common attacks and rate-limit bursts; verify a rule fired in logs.[8] [3]
  - S3 lifecycle policies for logs; enforce encryption at rest (bucket defaults) and in transit (HTTPS only).[8] [3]
  - Budgets and cost allocation tags; add a “cost” panel to the dashboard; document the biggest cost levers (instance type, data transfer, NAT).[10] [3]
- Understand
  - DR: backups/snapshots/AMIs, multi-AZ vs multi-Region, RTO/RPO in plain language; how alarms connect to rollback/runbooks.[3] [2]
- Prove
  - Final portfolio README with diagrams, pipeline links, alarm screenshots, test reports, cost/budget screenshot, and a 20-question Q&A sheet.[1] [3]

Interview talking points

- What DevOps is and why it helps: continuous, automated, observable delivery that reduces risk and speeds learning.[2] [3]
- Design story: two-AZ VPC with private EC2 behind ALB, blue/green via CodePipeline/CodeDeploy, Route 53 canary, CloudWatch dashboards/alarms, IAM least-privilege, Secrets Manager.[5] [8]

- Troubleshooting story: ALB 502—confirm app port with ss, check target health, view logs, and rollback by shifting Route 53 weights while investigating.[6] [8]
- Cost/reliability story: lifecycle and budgets, scaling policies with warm-up/cooldown, backup/DR plan with clear RTO/RPO.[9] [10]

Final artifacts checklist

- Repo containing Terraform modules, Bash utilities, Python CLI + tests, Dockerfile/compose, CodePipeline/CodeBuild/CodeDeploy configs, Postman and k6 tests, CloudWatch dashboards/alarms screenshots, WAF and lifecycle policies, runbooks, and diagrams.[1] [3]

If helpful, a printable pre-DevOps prerequisites checklist and day-by-day commands/ Terraform skeletons can be generated next with short “what this does” explanations and small diagrams to fit the preferred learning style.[7] [1]

⁂

1. productivity.documentation
2. learning.technical_concepts
3. learning.technical_skills
4. tools.docker_workflow
5. tools.aws_networking
6. productivity.communication
7. learning.visual_learning
8. tools.aws_infrastructure
9. tools.aws_scaling
10. tools.cloud_infrastructure