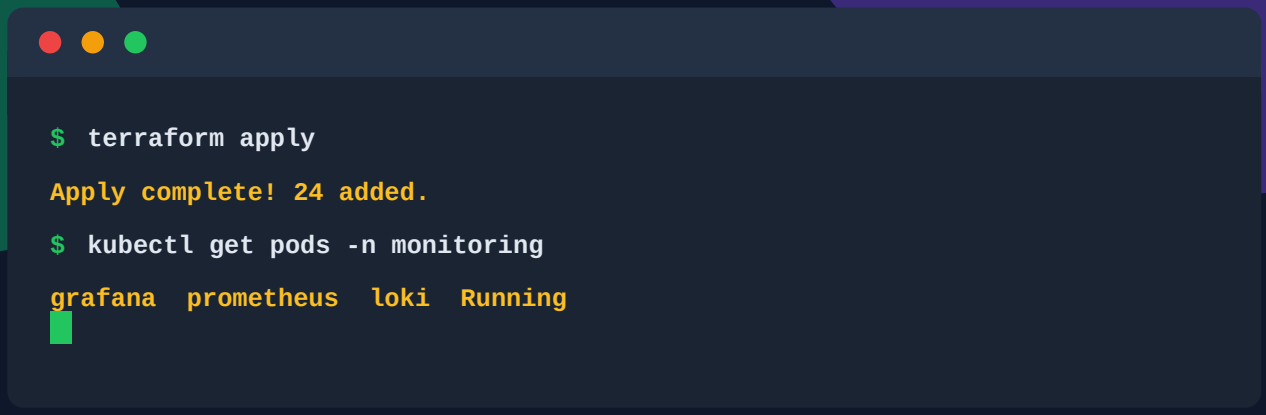


DevOps Training Program

Syllabus and Roadmap

Himali.Tech • 5 weeks • 27 sessions • Learning outcomes and assignments

WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 5 • NEW
Linux + Git	Docker + CI/CD	Kubernetes	AWS + IaC	Observability

Linux > Networking > Git > Docker > CI/CD > Kubernetes > AWS > Terraform > Monitoring

Lab platform: Iximiuz Labs | 1.5 hours per session | Daily LinkedIn updates

Understand > Demonstrate > Practice > Troubleshoot > Document

Tip: the checkboxes in this PDF are clickable. Open it in a PDF reader to tick them off as you learn.

DevOps Training Program

Himali.Tech: Training Roadmap, Learning Outcomes, and Assignments

This program helps beginners build a strong foundation in DevOps through structured learning, hands-on labs, real-world scenarios and project-based assignments. Students progress from Linux fundamentals to networking, Git, web servers, Docker, CI/CD, Kubernetes, cloud computing, Infrastructure as Code (IaC) with Terraform and, in Week 5, **monitoring and observability**. The program emphasizes practical implementation, troubleshooting, automation, collaboration and professional documentation.

0.1 Training structure

Item	Detail
Duration	5 weeks (4 weeks of core DevOps plus 1 week of monitoring and observability)
Schedule	5 training days per week, 1.5 hours per day
Total sessions	27 sessions (25 weekday plus 2 weekend), 40.5 hours of guided training
Learning approach	Theory, live demonstrations, hands-on labs, assignments and project-based learning
Weekends and labs	Weekend: programming fundamentals and DevOps automation. Lab platform: Iximiuz Labs

0.2 Roadmap at a glance

Week	Focus	Days	Assignment
1	Linux, Networking and Git	Days 1-5 + weekend	Linux and networking practical labs
2	Nginx, Docker and CI/CD	Days 6-10	Full-stack app deployed with Docker and CI/CD
3	Kubernetes and Iximiuz Labs	Days 11-16	Three-tier application on Kubernetes
4	AWS Cloud and Infrastructure as Code	Days 17-20	AWS three-tier app with Terraform
5 (new)	Monitoring and Observability	Days 21-25	Observability stack with Grafana, Prometheus, Loki and CloudWatch

0.3 Daily learning and LinkedIn guidelines

Students share their learning progress on LinkedIn every day to show consistency, document growth and build a professional presence.

Each daily post must:

- Summarize the topics learned, practical exercises completed and challenges encountered.
- Include relevant screenshots, terminal output, code snippets, diagrams or short demonstrations.
- Explain at least one important concept or problem-solving experience in your own words.
- Tag **Himali.Tech** in every post so mentors can track progress.
- Use relevant hashtags, such as #DevOps, #Linux, #Docker, #Kubernetes, #Grafana and #CloudComputing.
- Keep a consistent learning record throughout the program.

SAFETY FIRST

Posts must reflect your actual work and understanding. Never share passwords, access tokens, private keys, confidential project information or other sensitive data. This includes Grafana admin passwords, AWS keys and Slack webhook URLs.

WEEK
1

Linux, Networking and Git

Systems administration, networking and version control.

QUEST Administer Linux | **QUEST** Diagnose networks | **QUEST** Collaborate with Git

OBJECTIVE

Build a strong foundation in Linux system administration, networking concepts and version control.

Day 1 Linux Fundamentals

- Understand Linux architecture and the filesystem hierarchy.
- Explore `/`, `/home`, `/etc`, `/var`, `/tmp`, `/usr` and `/proc`.
- Learn essential commands for file and directory management.
- Work with files, directories, paths, redirection, pipes and command-line utilities.
- Understand absolute and relative paths.
- Read command manuals and access help.

Day 2 Linux Users, Permissions and Processes

- Understand users, groups and ownership; create, modify and manage them.
- Understand file permissions, ownership and permission modes.
- Use `chmod`, `chown`, `chgrp`, `id`, `groups` and `sudo`.
- Explore `/etc/passwd`, `/etc/shadow` and `/etc/group`: purposes and access restrictions.
- Inspect and manage processes with `ps`, `top`, `pgrep`, `kill` and related commands.
- Understand foreground and background processes, signals and process termination.

Day 3 Bash Scripting and Automation

- Understand shell scripting fundamentals and script execution.
- Work with variables, user input, command-line arguments and exit codes.
- Use conditional statements, loops and functions.
- Automate repetitive administrative tasks using Bash.
- Check command success and handle basic errors.
- Create a simple system-monitoring or health-check script.

Day 4 Networking Fundamentals

- Understand the OSI model and the TCP/IP protocol stack.
- Learn IP addressing, subnetting fundamentals, ports, DNS and gateways.
- Understand TCP versus UDP and how clients communicate with servers.
- Learn how HTTP and HTTPS requests work end to end.
- Inspect interfaces, routes, sockets and listening ports.
- Use `ip`, `ping`, `curl`, `ss`, `dig` and `traceroute`.
- Use SSH to securely access remote servers.
- Troubleshoot basic DNS, connectivity and service-access issues.

Day 5 Git and Version Control

- Understand why version control matters and how Git tracks changes.
- Learn repositories, commits, branches and the staging area.
- Use `git init`, `clone`, `status`, `add`, `commit`, `log` and `diff`.
- Work with remote repositories on GitHub or GitLab; use `push`, `pull` and `fetch`.
- Create and merge branches with a feature-branch workflow.
- Resolve merge conflicts.
- Use `.gitignore` to exclude unnecessary or sensitive files.
- Understand pull requests (PRs) and collaborative code review.

Weekend

Programming Fundamentals for DevOps

- Variables, data types, functions, conditionals and loops.
- Accepting user input and handling basic errors.
- Executing HTTP requests with `curl`; checking website availability and status codes.
- Performing service health checks and looking up DNS records.
- Checking running processes and terminating unwanted ones.
- Detecting suspicious processes with rule-based checks and alerting Slack through an incoming webhook.
- Checking SSL/TLS certificate expiration dates.
- Automating Git commands with Python input and `subprocess`.

WEEKEND OUTCOME

You can write simple scripts that automate common DevOps checks and report their results.

Platform: Iximiuz Labs. Complete the assigned Linux and networking exercises within one week after the final Week 1 class.

WEEK 1 ASSIGNMENT: LINUX AND NETWORKING PRACTICAL LABS	tick each box as you finish
☐	Create a GitHub repository with scripts, notes and relevant lab documentation.
☐	Add the assigned mentor as a repository collaborator.
☐	Create a feature branch and commit the completed work.
☐	Push the branch and open a pull request against `main`.
☐	Write a clear PR description with evidence of the completed exercises.
☐	Address mentor feedback until the PR is reviewed and merged.

WEEK
2

Nginx, Docker and CI/CD

Serve, package and automatically deliver applications.

QUEST Serve with Nginx | **QUEST** Containerize with Docker | **QUEST** Automate delivery

OBJECTIVE

Learn how to serve applications, package them into containers and automate application delivery.

Day 6 Web Servers and Nginx

- Understand the role of web servers and reverse proxies.
- Install and start Nginx.
- Understand static content serving versus reverse proxying.
- Configure server blocks (virtual hosts) to serve multiple websites.
- Deploy a static HTML website using Nginx.
- Understand how Nginx forwards requests to backend services.
- Read access and error logs.
- Identify HTTP status codes 200, 301, 400, 403, 404, 500, 502 and 504.
- Troubleshoot basic web server configuration issues.

Day 7 Docker Fundamentals

- Understand containers and how they differ from virtual machines.
- Learn Docker architecture, the daemon and the container runtime.
- Pull, run, inspect, stop and remove containers.
- Understand images, containers, registries and image layers.
- Build images using Dockerfiles.
- Inspect logs, execute commands and troubleshoot containers.
- Tag and push images to a container registry.

Day 8 Docker Compose, Networking and Storage

- Create and manage multi-container applications with Docker Compose.
- Configure services, environment variables and dependencies.
- Understand Docker bridge networks and service-to-service communication.
- Configure port mappings and expose application services.
- Use bind mounts and named volumes for persistent data.
- Deploy a full-stack app with a frontend, backend and database.
- Troubleshoot common container networking and storage problems.

Day 9 CI/CD Fundamentals and Pipeline Creation

- Understand Continuous Integration and Continuous Delivery/Deployment.
- Learn the stages of a CI/CD pipeline.
- Configure a pipeline using GitHub Actions.
- Trigger workflows on code pushes and pull requests.
- Automate code checks and Docker image builds.
- Manage pipeline variables and secrets securely.
- Understand pipeline logs, failures and debugging.

Day 10 Automated Deployment

- Configure a remote Linux virtual machine for deployment.
- Automate application deployment through a CI/CD pipeline.
- Build and publish Docker images to a registry.
- Deploy containers to the target server.
- Configure environment variables, networking and persistent storage.
- Verify application health after deployment.
- Understand basic rollback and recovery strategies.

Deploy a full-stack application on a Linux virtual machine using Docker and CI/CD.

WEEK 2 ASSIGNMENT: FULL-STACK APPLICATION DEPLOYMENT	tick each box as you finish
☐	A frontend, a backend and a database.
☐	Dockerfiles and a Docker Compose configuration.
☐	A GitHub Actions or GitLab CI pipeline.
☐	Automated image building and publishing.
☐	Automated deployment to the virtual machine.
☐	Persistent storage and environment-based configuration.
☐	Application health checks and deployment verification.
☐	README explaining the architecture, setup, pipeline and troubleshooting steps.

WEEK
3

Kubernetes and Iximiuz Labs

Container orchestration and multi-tier deployments.

QUEST Understand the cluster	**QUEST** Run workloads and Services	**QUEST** Deploy three tiers

OBJECTIVE

Understand container orchestration and deploy a multi-tier application on Kubernetes.

Day 11 — Kubernetes Fundamentals

- Understand why Kubernetes exists and how it compares with Docker Swarm.
- Learn Kubernetes architecture and cluster components.
- Control plane: API Server, etcd, Scheduler and Controller Manager.
- Worker node: kubelet, kube-proxy and container runtime.
- Set up a local cluster using Minikube; configure and use `kubectl`.
- Understand Pods as the smallest deployable workload unit.
- Create, inspect, describe, view logs from and delete Pods.
- Write and apply basic Kubernetes YAML manifests.

Day 12 — Workloads, Networking and Configuration

- Understand Deployments and ReplicaSets for stateless applications.
- Scale Deployments and inspect ReplicaSet behavior.
- Perform rolling updates and rollbacks.
- Expose applications using ClusterIP and NodePort Services.
- Understand LoadBalancer Services and their infrastructure requirements.
- Learn the purpose of Ingress and an Ingress Controller.
- Configure ConfigMaps and Secrets.
- Understand PersistentVolumes and PersistentVolumeClaims.
- Deploy and access a sample application through Services.

Day 13 — Advanced Kubernetes and Multi-Tier Deployment

- Use Namespaces to organise and isolate resources.
- Configure the Horizontal Pod Autoscaler (HPA) and understand its CPU/memory metrics requirements.
- Understand StatefulSets for stateful workloads such as databases.
- Create Jobs and CronJobs for batch and scheduled workloads.
- Deploy a three-tier application using Flask and PostgreSQL.
- Configure Deployments, Services, storage and ingress routing.
- Inspect resource usage using `kubectl top`.
- Troubleshoot using logs, events and `kubectl describe`.
- Understand resource requests, limits and ResourceQuotas.

Day 14-16 — Hands-On Labs with Iximiuz Labs

- Linux networking and network troubleshooting.
- Container fundamentals and Docker workflows.
- Container networking and storage.
- Kubernetes architecture and core resources.
- Kubernetes workloads, networking and configuration.
- Troubleshooting container and Kubernetes deployment issues.

The instructor assigns labs based on student progress and the platform's available exercises.

Deploy a three-tier application on Kubernetes.

WEEK 3 ASSIGNMENT: THREE-TIER KUBERNETES APPLICATION	tick each box as you finish
☐	Frontend, backend and database components.
☐	Deployments and Services.
☐	Ingress routing through a configured Ingress Controller.
☐	ConfigMaps and Secrets for configuration.
☐	Persistent storage for the database, where applicable.
☐	HPA configured with suitable resource requests and working metrics.
☐	Health checks and basic resource monitoring.
☐	Kubernetes YAML manifests stored in Git.
☐	README with architecture diagrams, deployment steps, verification commands and troubleshooting notes.

WEEK
4

AWS Cloud and Infrastructure as Code

Core AWS services, provisioned with Terraform.

QUEST Design a cloud network | **QUEST** Run apps on AWS | **QUEST** Provision with Terraform

OBJECTIVE

Understand core AWS services and provision cloud infrastructure using Terraform.

Day 17 AWS Fundamentals and Networking

- Understand AWS global infrastructure, Regions and Availability Zones.
- Learn Amazon VPC and its role in isolating cloud resources.
- Understand public and private subnets.
- Configure route tables and Internet Gateways.
- Understand NAT Gateways and private-subnet outbound connectivity.
- Learn the basics of security groups and network access control.
- Design a secure, multi-tier cloud network.

Day 18 Compute, Storage, Databases and Containers

- Amazon EC2 for virtual machine workloads.
- Amazon S3 for object storage.
- Amazon RDS for managed relational databases.
- Amazon ECR for storing container images.
- Amazon ECS for running and managing containers.
- AWS Lambda fundamentals for serverless computing.
- Load balancing and high availability.
- How these services fit together in a production-style architecture.

Day 19 Delivery Services and Terraform Fundamentals

- Amazon CloudFront for content delivery.
- Amazon Route 53 for DNS and routing.
- Why integrate S3, CloudFront and Route 53.
- Infrastructure as Code and its benefits.
- Terraform providers, resources, variables, outputs and state.
- Use `terraform init`, `validate`, `plan`, `apply` and `destroy`.
- Organise code into reusable files and modules.
- Manage state securely; protect credentials and state files.

Day 20 Infrastructure Provisioning and Deployment

- Provision AWS resources using Terraform.
- Create a VPC, subnets and the required routing resources.
- Deploy an application on EC2 or ECS.
- Configure an Application Load Balancer where appropriate.
- Integrate S3 and CloudFront for static content delivery.
- Configure Route 53 if an eligible domain and hosted zone are available.
- Understand where Lambda fits and implement a suitable function.
- Verify connectivity, security, application health and deployment behavior.
- Estimate costs and clean up resources no longer required.

Design, provision, deploy and document a three-tier application on AWS.

Deployment option	Notes
Containerised app on EC2	Simplest path; closest to your Week 2 Docker deployment
Containerised app on ECS	Managed container orchestration with ECR and a load balancer
Kubernetes-based deployment	Only if you have sufficient time and AWS experience

WEEK 4 ASSIGNMENT: AWS THREE-TIER APPLICATION WITH TERRAFORM

tick each box as you finish

- [ ] VPC with appropriately configured subnets and routing.
- [ ] Compute resources for the application, and load balancing.
- [ ] S3 integration and a CloudFront distribution.
- [ ] Route 53 configuration, where domain access is available.
- [ ] A suitable Lambda function with a clearly defined purpose.
- [ ] Terraform code that provisions the required infrastructure.
- [ ] An architecture diagram showing the request flow and AWS services.
- [ ] README on architecture, deployment process, security considerations and cost management.
- [ ] A LinkedIn blog post documenting the project and tagging **Himali.Tech**.

REMEMBER

Implement services as a coherent architecture, not as a checklist with no use case. Secure every AWS resource and clean it up after evaluation.

WEEK
5

Monitoring and Observability

See inside running systems: metrics, logs, traces and alerts.

QUEST Collect metrics with Prometheus	**QUEST** Visualize in Grafana + CloudWatch	**QUEST** Alert and respond to incidents

OBJECTIVE

Learn to see inside running systems: collect metrics, logs and traces, build dashboards, and alert on problems before users notice. Everything is applied to the applications you deployed in Weeks 2 to 4.

Your app /metrics + logs > **Prometheus** metrics > **Loki** logs > **Grafana** dashboards > **Alerts** Slack / SNS

Tool	Purpose	Where used
Prometheus + exporters	Collect and store metrics (Node Exporter, cAdvisor, `/metrics`)	Days 21, 22, 25
Grafana	Dashboards, exploration and alerting UI	Days 22 to 25
Alertmanager	Group, route and silence alerts (Slack)	Day 22
Loki + Grafana Alloy	Centralised logs and LogQL queries	Day 23
AWS CloudWatch + SNS	AWS metrics, logs, dashboards and alarms	Day 24
OpenTelemetry, Tempo or Jaeger	Distributed tracing	Day 25
Blackbox Exporter, Uptime Kuma	Uptime and endpoint probes	Day 25

Day 21 Observability Fundamentals and Prometheus

- Monitoring versus observability, and the three pillars: metrics, logs, traces.
- The four golden signals (latency, traffic, errors, saturation) plus the RED and USE methods.
- SLIs, SLOs, SLAs and error budgets at a basic level.
- Prometheus architecture: scraping, targets, labels and the time-series database.
- Run Prometheus with Docker Compose and write `prometheus.yml` scrape configs.
- Expose host metrics with Node Exporter and container metrics with cAdvisor.
- Instrument a Flask app with a `/metrics` endpoint using `prometheus_client`.
- Write basic PromQL: `rate()`, `sum by`, `histogram_quantile()`.

Day 22 Grafana Dashboards and Alerting

- Install Grafana and add Prometheus as a data source.
- Build dashboards with time series, stat, gauge and table panels.
- Use variables and templating; import and adapt community dashboards.
- Provision data sources and dashboards as code, stored in Git.
- Understand alert rules: thresholds, evaluation, pending and firing states.
- Use Alertmanager: grouping, severity labels, routing and silences.
- Send alerts to Slack with an incoming webhook (the weekend skill, reused).
- Run Prometheus, Grafana and the app together in one Compose file.

Day 23 Centralised Logging with Loki

- Why centralise logs; structured (JSON) logging and log levels.
- Loki architecture and how it differs from Elasticsearch (labels, not full-text indexing).
- Collect container and host logs with Grafana Alloy (the successor to Promtail).
- Query with LogQL: label filters, line filters and log-based rates.
- Correlate logs and metrics on one Grafana dashboard.
- Alert on log patterns such as repeated errors.
- Retention, label cardinality and storage cost.
- Overview of ELK/OpenSearch as an alternative stack.

Day 24 AWS CloudWatch

- Metrics: namespaces, dimensions, default EC2/ALB/RDS/ECS metrics and custom metrics.
- Install the CloudWatch agent on EC2 for memory, disk and log-file collection (with an IAM role).
- Log groups, streams and retention; send ECS container logs with the `awslogs` driver.
- Logs Insights queries and metric filters.
- Alarms: thresholds, evaluation periods and composite alarms, notifying through SNS.
- CloudWatch dashboards and a first look at Container Insights.
- Add CloudWatch as a Grafana data source and combine it with Prometheus.
- Provision log groups and alarms with Terraform; control cost with retention.
- Overview of CloudTrail (who did what) and AWS X-Ray (tracing).

Day 25 Kubernetes Monitoring, Tracing and Incident Response

- Deploy `kube-prometheus-stack` with Helm; understand ServiceMonitors.
- Monitor nodes, pods and namespaces; connect metrics-server and HPA (Week 3) with dashboards.
- Distributed tracing: spans, traces, context propagation, OpenTelemetry basics, Tempo or Jaeger.
- Uptime probes with Blackbox Exporter, Uptime Kuma or CloudWatch Synthetics.
- Reduce alert noise: alert on symptoms, use `for:` durations, link a runbook.
- Incident basics: severity, on-call, runbooks and blameless postmortems.
- How commercial tools (Datadog, New Relic) map to the same concepts.

A sketch of what you will configure (illustrative; metric names depend on your instrumentation):

```
# prometheus.yml
scrape_configs:
  - job_name: backend
    static_configs: [{targets: ["backend:5000"]}]
  - job_name: node
    static_configs: [{targets: ["node-exporter:9100"]}]

# alert rule
- alert: HighErrorRate
  expr: sum(rate(http_requests_total{status=~"5.."}[5m])) / sum(rate(http_requests_total[5m])) > 0.05
  for: 5m
  labels: {severity: critical}
  annotations: {summary: "More than 5% of requests are failing"}
```

Add monitoring and alerting to a three-tier application from Weeks 2 to 4 (Docker, Kubernetes or AWS).

WEEK 5 ASSIGNMENT: OBSERVABILITY STACK FOR THE THREE-TIER APP	tick each box as you finish
☐	Prometheus and Grafana running (Docker Compose or Helm) and scraping the application.
☐	The app exposes `/metrics` (requests, latency, errors); host and container metrics via exporters.
☐	A Grafana dashboard covering the four golden signals, provisioned from files stored in Git.
☐	Centralised logs in Loki through Grafana Alloy, with a saved LogQL query for errors.
☐	At least 3 alert rules (service down, high error rate, high CPU or memory) routed to Slack with a runbook link.
☐	AWS CloudWatch: logs from EC2 or ECS, a Logs Insights query, a dashboard and 2+ alarms notifying by SNS email.
☐	Incident drill: break a service, show the alert firing and recovering, then write a short postmortem.
☐	README with architecture diagram, setup steps, key queries, screenshots, troubleshooting and cost notes.
☐	A LinkedIn post documenting the project and tagging **Himali.Tech**.

PRO TIP

Bonus: add CloudWatch as a Grafana data source on the same dashboard, add OpenTelemetry traces with Tempo or Jaeger, or create the CloudWatch alarms with Terraform.

Assessment and Completion Criteria

Evaluated on practical implementation, consistency, collaboration, troubleshooting and documentation.

Area	What mentors look for
Daily learning updates	Consistent LinkedIn posts tagging Himali.Tech
Hands-on practice	Completion of assigned labs and practical exercises
Git workflow	Meaningful commits, feature branches and mentor-reviewed pull requests
Weekly assignments	Working implementations that meet the stated requirements
Troubleshooting	Ability to explain problems encountered and how they were resolved
Monitoring and incident response	Dashboards, alerts and a postmortem that show you can detect and explain a failure
Documentation	Clear READMEs, commands, configuration explanations and architecture diagrams
Final project	A deployed cloud application with Terraform code, monitoring and a public project write-up

EXPECTED LEARNING OUTCOMES: BY THE END I CAN...	tick each box as you finish
☐	Use Linux confidently for basic system administration and troubleshooting.
☐	Understand networking concepts and diagnose common connectivity issues.
☐	Collaborate using Git and GitHub pull requests.
☐	Automate routine operational checks using Bash and Python.
☐	Configure Nginx and deploy containerised applications using Docker.
☐	Build basic CI/CD pipelines for automated application delivery.
☐	Deploy and troubleshoot multi-tier applications on Kubernetes.
☐	Understand core AWS networking, compute, storage and delivery services.
☐	Provision cloud infrastructure using Terraform.
☐	Monitor applications with Prometheus, Grafana, Loki and CloudWatch, and respond to alerts.
☐	Document technical work and show progress through professional LinkedIn posts.

PROGRAM GOAL

Help beginners move from foundational concepts to practical DevOps implementation through consistent learning, hands-on experimentation and real-world project delivery.