DevOps Projects

[2 projects for final year and 2 for pre-final year]

1. **End to End DevOps Implementation on an E-Commerce Project**

**Background**  
E-commerce teams need fast, reliable, repeatable delivery pipelines to ship features, scale infrastructure, and maintain high availability. Many small teams rely on ad-hoc scripts and manual steps for infra provisioning, containerization, CI/CD and deployments — which increases errors, slows release cadence and makes onboarding new engineers harder.

**Problem statement**  
Organisations lack a single, hands-on training path that covers the full DevOps lifecycle (cloud setup, IaC, containerisation, Kubernetes, CI/CD, GitOps, domain/config setup, and production verification) while also preparing students for real world tasks and interviews. Students need practised, project-based learning that mirrors industry workflows.

**DevOps solution**  
A practical, end-to-end project where students implement an e-commerce application from local development to production on Kubernetes using Terraform and AWS, set up CI/CD (GitHub Actions + ArgoCD), and configure networking and custom domain.

**Implementation approach**

1. Introduce the e-commerce project: architecture, components and repo layout.
2. Hands-on AWS account setup and IAM role/user creation for secure access.
3. Provision EC2 and networking basics (security groups, inbound rules).
4. Install and configure Docker and Kubernetes (minikube / EKS) locally and on cloud.
5. Containerise the application, build and push images, run locally with docker-compose.
6. Use Terraform to provision AWS resources, manage state and locking, and implement backend and VPC/EKS infra.
7. Write Kubernetes manifests (deployments, services, ingress, storage) and deploy on cluster.
8. Configure Route53 and TLS for a custom domain; set up ingress controller.
9. Implement GitOps with ArgoCD, and CI with GitHub Actions (end-to-end pipeline).

**Impact / Benefits**

* Students learn the full lifecycle from infra provisioning to production deployment.
* They gain practical skills in Terraform, Docker, Kubernetes, GitOps and CI/CD.
* Project artifacts (repos, Terraform state, Kubernetes manifests, CI pipelines) become portfolio material for interviews.
* Teaches real world operational practices (state management, RBAC, ingress, domain config, ArgoCD workflows).

1. **Learning Platform’s Speed Up with End-to-End CI/CD and GitOps on AWS**

**Background**  
A growing online learning platform runs mixed legacy services and new microservices across environments with manual, error-prone releases, inconsistent quality checks, and limited observability. Engineering teams need a single, repeatable pipeline to build, test, deploy and monitor reliably.

**Problem statement**  
How do we teach a production-grade workflow that automates builds, enforces quality gates, stores immutable artefacts, deploys consistently to Kubernetes via GitOps, and provides operational visibility — while producing resume-ready project work?

**DevOps solution**  
Deliver a hands-on project using Terraform for infra, Jenkins/GitHub Actions for CI, SonarQube for quality, Artifactory for artifacts, Docker and Helm for packaging, and ArgoCD (GitOps) to deploy on EKS with Prometheus/Grafana monitoring and Route53/Ingress for DNS and TLS.

**Implementation approach**

1. Define app architecture, repo layout and CI/CD goals.
2. Provision AWS infra (VPC, EKS, IAM) with Terraform and remote state.
3. Install CI tooling (Jenkins/GitHub Actions), SonarQube and Artifactory; author pipelines.
4. Containerise services, push images, write Helm charts and Kubernetes manifests.
5. Enable GitOps with ArgoCD, configure Ingress Route53 and TLS.
6. Deploy Prometheus and Grafana for dashboards, add RBAC, backups and runbooks; include hands-on labs.

**Impact / Benefits**

* Learners will acquire a practical, portfolio-grade project demonstrating infrastructure as code, containerisation, CI/CD, GitOps and observability.
* The organisation benefits from faster, safer, and repeatable releases, stronger code quality through automated gates, and improved operational visibility that reduces incident time-to-resolution.
* Learners leave with concrete artefacts (Terraform modules, Jenkinsfiles, Helm charts, ArgoCD apps) and interview-ready explanations, making them immediately more effective in production engineering roles.

1. **Automated CI/CD Pipeline for Scalable Application**

**Background**  
A growing online learning startup has a Java-based platform with thousands of students logging in daily. Initially, the team deployed updates manually: developers copied WAR files to servers, ops teams restarted Tomcat, and testers verified changes directly in production. This worked when users were few, but as traffic grew, downtime became frequent, rollbacks were messy, and debugging was slow. Releases stretched from hours to days.

**Problem statement**  
The company faces mounting pressure from customers demanding new features faster. Manual deployments cause errors, environments drift apart, and teams waste time fixing failed builds instead of shipping value. The lack of automation means every release feels risky. How can the company streamline this into a repeatable, reliable, and observable delivery process?

**DevOps solution**  
Implement a structured CI/CD pipeline that automates the journey from code commit to production. Use version control for collaboration, automate builds and testing, containerise applications for consistency, configure infrastructure automatically, and deploy on scalable cloud-native infrastructure. Add monitoring and dashboards so issues are visible before customers complain.

**Implementation approach**

1. **Source control integration** – manage all code in Git with feature branching.
2. **Continuous integration** – set up pipelines to build and test code automatically with every commit.
3. **Application packaging** – create Docker images and push them to a central registry.
4. **Automated configuration** – use tools to set up servers and environments consistently.
5. **Continuous delivery** – deploy to Kubernetes clusters on AWS with manifests and Helm charts.
6. **Observability** – add monitoring and dashboards for health, performance, and error tracking.
7. **Learning labs** – simulate failures, rollbacks, and scaling scenarios to mirror production challenges.

**Impact / Benefits**

* **Faster, reliable releases**: updates move from days to minutes.
* **Error reduction**: automation removes manual, repetitive mistakes.
* **Consistency across environments**: containers ensure dev, test, and prod match.
* **Improved visibility**: monitoring alerts teams before customers are impacted.
* **Career advantage**: learners can showcase an end-to-end, real-world DevOps project on their resumes and confidently explain it in interviews.

1. **Securing and Accelerating Releases with Continuous Delivery Pipelines**

**Background**  
A growing SaaS company frequently pushes updates to its cloud application. Initially, developers manually tested, built, and deployed code to production. While this worked at a small scale, the lack of automation soon caused delays, inconsistent builds, and insecure deployments. With customer growth accelerating, the company must adopt a robust CI/CD pipeline that ensures speed, reliability, and security.

**Problem statement**  
Manual deployment practices lead to missed quality checks, security gaps, and unstable releases. Teams spend too much time fixing deployment failures instead of delivering features. The organisation needs a system that integrates code commits, automated testing, containerisation, security scans, and production deployments into a single, repeatable pipeline.

**DevOps solution**  
Set up a production-grade CI/CD pipeline using GitHub Actions, the same approach adopted by leading tech companies. The pipeline will automate every stage: code integration, testing, container image builds, vulnerability scanning, and deployment to production. With built-in compliance, secrets management, and monitoring, the system guarantees that every release is both fast and secure.

**Implementation approach**

1. **Pipeline setup** – configure GitHub Actions workflows for code integration and branching.
2. **Secrets management** – store and use environment secrets securely within workflows.
3. **Automated testing** – add unit, integration, and quality checks in the pipeline.
4. **Containerisation** – build and push Docker images for consistent runtime environments.
5. **Security scanning** – integrate vulnerability and compliance scans before release.
6. **Production deployment** – automate deployment to cloud infrastructure with zero-downtime strategies.
7. **Best practices** – optimise pipeline for scalability, speed, and reliability with reusable workflows.

**Impact / Benefits**

* **Faster releases** – automated workflows reduce delivery time from hours to minutes.
* **Higher quality** – integrated testing ensures code reliability before production.
* **Improved security** – secrets management and vulnerability scans protect against breaches.
* **Consistency** – containerised builds guarantee the same results across environments.
* **Industry relevance** – learners master GitHub Actions, one of the most widely adopted CI/CD tools and gain skills directly applicable in real-world production pipelines.