ChatGPT

CI/CD and Environment Strategy

A lightweight continuous integration and deployment pipeline ensures code quality, enforces standards and automates deployments across development, staging and production environments.

Workflow Overview (GitHub Actions example)

1. Build & Test

- **Trigger**: Pull requests and pushes to any branch.
- **Steps**:
- **Checkout code**.
- **Set up Node** (use matrix for multiple Node versions if needed).
- **Install dependencies** (`npm ci` for both `frontend` and `backend`).
- **Run linters** (`npm run lint`) to enforce ESLint and Prettier rules.
- **Run tests** (`npm test`) for backend and frontend.
- **Generate build artifacts** if tests pass (e.g., `npm run build` for Next.js).

2. Publish Docker Images

- **Trigger**: Push to `main` or tagged release.
- **Steps**:
- Build Docker images for backend and frontend using Dockerfiles in `infra/docker`.
- Tag images with commit SHA and optionally `latest`.
- Push images to a container registry (e.g., GitHub Container Registry or Docker Hub).

3. Deploy

- **Trigger**: Push to `main` (staging) or create a release tag (production).
- **Steps**:
- Retrieve environment-specific secrets (database URL, JWT secret, SMTP credentials) from the secrets store.
- Deploy the backend and frontend containers to the target environment:
    - **Development**: Use `docker-compose` to start services locally or on a dev server.
    - **Staging**: Deploy to a managed platform like Heroku, Render or AWS Elastic Beanstalk using a single container per service to minimise cost.
    - **Production**: Deploy to AWS ECS Fargate or a small Kubernetes cluster when scaling requirements grow.
- Run database migrations before starting the new version.
- Notify stakeholders on deployment completion.

Environment Strategy

- **Development**: Each developer can run the full stack locally using `docker-compose`. Environment variables are loaded from `.env.development`. Databases can run in Docker or a shared dev database.
- **Staging**: A near-production mirror where new features are deployed after passing CI. Use `.env.staging` values and separate cloud resources (database, object storage). Staging is used for QA and user acceptance testing.
- **Production**: The live environment accessed by end users. Use `.env.production` values; resources are sized for expected load. Access is restricted and deployments require manual approval.

Secrets Management

- Use the CI/CD platform's secret storage (e.g., GitHub Secrets) for build-time secrets (container registry credentials). Only actions with appropriate permissions can read these secrets.
- For runtime secrets, use a managed secret store (AWS Secrets Manager, HashiCorp Vault). The deployment platform injects secrets into containers as environment variables or mounts them as files.
- Local development uses the `.env.*` files that are excluded from version control.

Deployment Flow to Minimise Costs

- Start with minimal infrastructure (single container for backend and another for frontend) on a low-cost PaaS. Scale vertically by increasing container size before moving to Kubernetes.
- Use auto-scaling on CPU or memory metrics to add containers only when needed.
- Clean up unused resources (old images, unused volumes) via scheduled jobs in the CI pipeline.

This CI/CD strategy ensures code quality, reduces manual overhead and supports environment parity, all while keeping operational costs low.
