La Salle Computer Society
DevOps Standards Manual

V1.1.01 -August, 2025 - 40th LSCS

The DevOps Standards Manual (BSM) outlines the **standard tools, practices, conventions, and processes** that every project must follow. The manual standards is a requirement that serves as the foundation and base for future projects as of August 2025.

What is DevOps and its Philosophy?

DevOps is a combination of cultural philosophies, practices, and tools designed to increase an organization's ability to deliver applications and services at high velocity. It's not just a role, but a way of working that breaks down the traditional silos between software development (Dev) and IT operations (Ops).

The core philosophy for the **La Salle Computer Society (LSCS) Research and Development Committee** revolves around three key principles:

1. **Collaboration & Ownership**
    - Developers, designers, and project managers must work together throughout the entire project lifecycle.
    - Teams are expected to own their services from the initial commit to production deployment and maintenance.
    - This will lead to a deeper understanding of the project and to higher-quality, more reliable applications.
2. **Automation**
    - Automate everything possible to reduce manual effort and minimize human error.
    - This is central to our workflow, from testing and building to deployment.
    - We achieve this by requiring all repositories to have a **CI/CD pipeline** using **GitHub Actions** and containerizing all applications with **Docker**.
3. **Continuous Improvement**
    - We continuously monitor, test, and gather feedback to make our development and deployment pipelines faster, more secure, and more resilient.
    - Every project is an opportunity to improve our standards.
    - Suggestions and recommendations for improvement are always welcome no matter what position you hold.

Tools to be Used

- Essentials

    - Git → for version control (obviously).
    - Github → for storing repositories.
        - Github Projects → for seamless project management and integration with Github Issues, Pull Requests, etc.
    - Github Actions → for creating CI/CD pipelines.
    - Docker → for containerization.
    - Coolify (self-hosted) → for deploying projects.
        - Mostly for development and staging
        - Production for internal projects (and sometimes external projects based on the requirements)
- DevOps/Infrastructure-specific
    - Ansible → for automating configuration management
        - Ex. *You are **tasked to migrate the entire existing LSCS server to another server**, you use this to backup volumes/databases/applications, setup new server, setup SSH keys, and restore the same configuration of the old server to the new server.*
    - Terraform → for provisioning/deploying resources (Infrastructure-as-Code or IaC).
        - Ex. *You are **tasked to deploy an application with a database in Digital Ocean**, you use this to write and define the application configuration (name, region, RAM, etc.) and "apply" the file to automatically deploy it.*
    - Netdata → for observability and metrics\
    - Falco → for runtime security to detect malicious actors in the server (ex. Crypto miners)
    - Trivy → for vulnerability scanning (mostly used in CI/CD pipelines).
    - Jaeger/OpenTelemetry (Optional) → for distributed tracing to understand the "why" of a service using too much RAM or CPU on the application-level
        - Netdata and Prometheus tell you *that* a service is slow or has high CPU usage, distributed tracing tells you *why*.
        - For this to work, developers must also use OpenTelemetry metrics (in the codebase) and send to the Jaeger service.
    - Discord Alerts (Webhook) → the main alerting platform for all LSCS applications.

Standard Rules and Conventions

1. **EVERYTHING MUST BE IN GITHUB (VERSION-CONTROLLED USING GIT)**
2. **ALL PROJECTS MUST BE CONTAINERIZED (VIA DOCKER)**
    - All necessary repositories (for frontend and backend) must include a `Dockerfile` for easier, consistent, and faster deployment of applications.
    - ***Benefit: Equal and standard environments for development, staging, and production.***
3. **ALL REPOSITORIES MUST HAVE A CI/CD PIPELINE, PROPER GITHUB ISSUES AND PRs TEMPLATES**
    - Github Issues, Pull Request Templates, etc. (refer to [LSCS RND] Code Contribution Guidelines)
4. **THE `main` BRANCH MUST BE BRANCH-PROTECTED**
    - No one should be able to push directly to the `main` branch
    - Use rulesets → at the very least, protect the `main` branch
5. **ENSURE ALL PROJECTS MUST HAVE WELL-DEFINED SPEC AND BEST PRACTICES**
    - Clear scope and requirements per project.
    - Conventional commits (refer to [LSCS RND] Code Contribution Guidelines)

- Github Issues, Pull Request Templates, etc. (refer to [LSCS RND] Code Contribution Guidelines)

Branches and Environments

- `main` **Branch → Production Environment**
  - **Purpose**: This branch represents the official, stable, user-facing code. It should **always** be in a deployable state.
  - **Workflow**: Code is only merged into `main` from the `staging` branch (if used) or the `dev` branch after a successful Pull Request (PR) that has been reviewed and approved. All PRs merged into `main` **must use "Squash and Merge"** to maintain a clean and readable commit history.
  - **Deployment**
    - A merge to `main` triggers the final deployment pipeline to the production environment.

- `staging` **Branch → Staging Environment**
  - **Purpose**
    - This branch serves as a pre-production environment for final testing, quality assurance (QA), and User Acceptance Testing (UAT). The staging environment should mirror the production environment as closely as possible.
  - **Workflow**
    - Created from the `dev` branch when a set of features is ready for release testing. No new features should be added to this branch; only bug fixes are permitted.
  - **Deployment**
    - A push to the `staging` branch deploys the code to the staging server, which is hosted in the same environment as production to ensure consistency.
  - ❗ **Budget Constraint Note** ❗
    - The `staging` branch and its corresponding environment are highly recommended but are **not a hard requirement** for all projects.
    - For external projects with budget limitations (e.g., hosted on Digital Ocean), maintaining a separate staging server may not be feasible.
    - In these cases, thorough testing in the `dev` environment and a rigorous PR review process are critical before merging directly from `dev` to `main`.

Production Checklist

Projects

- Most applications are ***"3-tier applications"***, meaning a (1) frontend/client, (2) backend/API, and a (3) database

- [ ] Setup repository for frontend and backend
    - [ ] Protect `main` branch → ruleset
- [ ] Proper environments setup
    - `main` branch - for production environment
    - `dev` branch - for development environment
    - `staging` branch - for staging environment (optional)
- [ ] Proper CI/CD setup for necessary repositories
- [ ] Setup templates for GitHub Issues, Pull Requests (see [LSCS RND] Code Contribution Guidelines)
- [ ] Protected `main` branch
    - Do not allow force pushes
    - Configure "Squash and Merge" for Pull Requests
- [ ] Domain name (DNS) setup
    - If project is **INTERNAL**, domain names should be:
        - For frontend/web clients: **project.app.dlsu-lscs.org**
        - For backend/web APIs: **project.api.dlsu-lscs.org**
    - If project is **EXTERNAL**:
        - If client requires a **CUSTOM DOMAIN NAME** then coordinate with project manager
            - bought via a DNS provider (e.g. Cloudflare, Porkbun, Namecheap, GoDaddy, etc.)
            - then map custom DNS to the IPv4 or IPv6 address and/or the nameservers of the deployment platform (coordinate with project manager)
- [ ] Hosting / Deployment Platform (e.g. AWS, Digital Ocean, Render, Kubernetes, Selfhosted, etc.)
    - [ ] Frontend Client deployment
    - [ ] Backend API deployment
    - [ ] Database deployment (either Managed or Selfhosted)