La Salle Computer Society
LSCS Infrastructure: DevOps Blueprint

V1.1.01 -August, 2025 - 40th LSCS

This document outlines the architectural components and strategic choices for the LSCS Infrastructure, a modern, fully automated platform designed for resilience, security, and operational excellence. By integrating best-in-class open-source tools within a cohesive Infrastructure as Code (IaC) framework, this blueprint establishes a robust foundation for deploying, managing, and scaling applications efficiently.

---

1. Core Philosophy: Automation and Immutability

The entire infrastructure is managed as code, stored in version-controlled Git repositories. This approach ensures that the system is repeatable, auditable, and easily recoverable. The platform is built upon a hybrid model, leveraging the convenience of the Coolify PaaS for its user interface and deployment automation, while using foundational DevOps tools to manage the entire lifecycle declaratively.

---

2. Technology Stack Overview

The LSCS stack is composed of carefully selected tools, each serving a specific purpose within the DevOps lifecycle.

2.1. Configuration & Infrastructure as Code (IaC)

This is the bedrock of the platform, ensuring every component is defined and versioned.

- **Ansible**
  - Used for initial server configuration and hardening. An Ansible playbook transforms a bare VPS into a production-ready host with all necessary dependencies and security configurations, including the installation of Coolify itself. This makes server setup a repeatable, one-step process.
- **Terraform**
  - Manages all resources *within* the Coolify platform. While Ansible sets up the server, Terraform defines the projects, applications, databases, and their configurations inside Coolify. This ensures that the PaaS layer itself is managed as code, making the entire application landscape portable and recoverable.

2.2. Deployment & Containerization

The platform is built around container technology for consistency and isolation.

- **Docker and Docker Compose**

    - All applications and services are containerized using Docker. Multi-container applications are defined using docker-compose.yml files, which serve as the single source of truth for an application's architecture.
- **Coolify**
    - Serves as the central deployment engine and user interface. It leverages its Docker Compose build pack to interpret the application definitions from Git and manage the container lifecycle. It provides a user-friendly abstraction over the underlying Docker engine.
- **Traefik**
    - Functions as the reverse proxy and is managed by Coolify. It automatically handles routing, load balancing, and SSL certificate generation (via Let's Encrypt) for all deployed services, simplifying how applications are exposed to the internet.

2.3. CI/CD Automation

The pipeline from code commit to production deployment is fully automated.

- **GitHub Actions:** Orchestrates the entire CI/CD workflow. When code is pushed to the main branch, a GitHub Actions pipeline is triggered to:
    1. Build the application's Docker image.
    2. Run automated tests.
    3. Perform a vulnerability scan on the image using Trivy.
    4. Push the final, scanned image to a container registry.
    5. Trigger a deployment in Coolify via its API and webhooks.

---

3. Observability: Gaining Insight into the System

A multi-layered approach to observability ensures deep visibility into both the infrastructure and the applications running on it.

3.1. Monitoring

- **Primary Tool: Netdata**
    - **Role:** Netdata is the primary tool for real-time, high-granularity monitoring of the VPS and all running containers. It provides instant, per-second metrics on CPU, memory, disk I/O, and network usage with zero configuration.
    - **Benefits:** Its extremely low resource footprint and auto-discovery capabilities make it the perfect "out-of-the-box" solution for immediate system health visualization and anomaly alerting.
- **Advanced Option: Prometheus & Grafana Stack**
    - The Prometheus/Grafana stack is the industry standard for long-term metric storage, powerful querying (PromQL), and creating highly customized, business-specific dashboards.
    - **Role:** This stack is deployed *as another service within Coolify* when deeper, long-term analysis is required. It complements Netdata by providing historical trend analysis and a more powerful platform for correlating diverse metrics over time. Coolify has built-in monitoring for basic resource health, but this stack provides a path to observability maturity.

3.2. Logging

- For a single-server environment, a robust yet lightweight logging stack is essential. The recommended solution is **Loki** combined with **Promtail**.
- **Strategy:**

1. **Docker Daemon Configuration**
    - The Docker daemon is configured to use the local logging driver with built-in log rotation. This is a critical first step to prevent log files from growing indefinitely and consuming all disk space.
2. **Log Aggregation with Loki & Promtail**
    - Promtail is deployed as a lightweight agent to collect logs from all containers and forward them to Loki.
    - Loki is a highly efficient log aggregation system that, unlike traditional solutions, only indexes the metadata (labels like application name, container ID, etc.) rather than the full text of the logs. This makes it extremely fast and resource-efficient.
3. **Visualization**
    - Loki integrates seamlessly with Grafana, allowing the team to query, visualize, and alert on logs from the same interface used for advanced metrics.

3.3. Tracing & Application-Level Observability

- This is entirely **application-level** and serves a different purpose than infrastructure monitoring. While tools like Netdata and Prometheus tell you *that* a service is slow or has high CPU usage, distributed tracing tells you *why*.
- **Role of Jaeger / OpenTelemetry**
    - OpenTelemetry is a set of tools and libraries used to instrument your application code to generate "traces."
        - A *trace* follows a single user request as it travels through all the different microservices in your architecture.
    - Jaeger is the backend that collects and visualizes these traces.
- **When to Use It:** For the initial deployment of monolithic internal applications, distributed tracing is likely not necessary. However, as the organization's architecture evolves towards microservices, it becomes an indispensable tool for debugging performance bottlenecks and understanding complex service interactions. It should be considered a key component for future application development.

---

4. Security: A Multi-Layered Defense

Security is integrated throughout the lifecycle, from development to runtime.

- **Vulnerability Scanning: Trivy**
    - **Role**
        - Trivy is a static analysis tool that scans for known vulnerabilities (CVEs) in OS packages and application dependencies.
    - **Implementation**
        - It is integrated directly into the **GitHub Actions CI/CD pipeline**. Before an image is pushed to the registry, Trivy scans it. If high or critical severity vulnerabilities are found, the pipeline fails, preventing insecure code from being deployed. A second Trivy scan is run as a scheduled task on the server itself to detect vulnerabilities in already-running applications that may have been discovered after deployment.
- **Runtime Threat Detection: Falco**
    - **Role**
        - Falco provides runtime security. It continuously monitors the server's kernel and container activity in real-time to detect suspicious behavior that indicates a potential breach, such as a crypto miner, reverse shell execution, or unexpected network connections.

    - **Implementation**
        - Falco runs as a persistent background service on the VPS, providing a constant layer of threat detection.

4.1. Alerting & Notifications

- **Discord Webhooks**
    - Serves as the central notification channel. Alerts from Coolify (deployments), Trivy (vulnerability findings), Falco (runtime threats), and Netdata (performance anomalies) are all routed to a dedicated Discord channel, providing a single pane of glass for important events.

---

5. Supporting Infrastructure

5.1. Cron Job Management

- **GoCron or Cronicle**
    - To avoid the unmanageable nature of system crontab, scheduled tasks are managed using a dedicated service deployed within Coolify. Both GoCron and Cronicle provide a web UI, robust logging, and allow job definitions to be stored as code in a Git repository, adhering to IaC principles.

5.2. Optional Components for Future Growth

- **Cloud Providers (Digital Ocean, AWS)**
    - These serve as the underlying IaaS layer where the VPS is provisioned. The IaC approach ensures the platform is cloud-agnostic and can be migrated if needed.
- **Analytics & Data Warehousing (Google Analytics, BigQuery)**
    - These are application-level services for business intelligence and data engineering, separate from the core infrastructure but can be integrated with the deployed applications.
- **Secrets Management (HashiCorp Vault or Bitwarden)**
    - While GitHub Actions secrets are used for CI/CD, a dedicated secrets manager is the recommended best practice for managing application secrets in production. Deploying Vault or using Bitwarden Secrets Manager provides a centralized, secure, and auditable way to handle sensitive credentials for all applications and services.