SYSTEMS LAB DOCUMENTATION // CASE STUDY

POST-MORTEM: BUILDING MASWILDLABS.COM

Target System	maswildlabs.com	**Incident Type**	Data Loss / File Corruption
Environment	Proxmox VE Cluster (Isolated LXCs)	**Recovery Resolution**	Automated GitOps Infrastructure

PROJECT BACKGROUND AND VISION

The objective of this project was to build a clean, professional technical writing portfolio designed to showcase my work. Initially, I chose MkDocs for the building. While MkDocs is an excellent tool for standard documentation, its layout constraints made custom branding and portfolio personalization highly frustrating. After researching alternatives (Hugo, Astro, Docusaurus), I pivoted to Astro for its layout flexibility, speed, and exceptional Markdown parsing capabilities.

THE TECH STACK AND METHODOLOGY

The infrastructure is a self-hosted microservices architecture running on a Proxmox virtualized environment. To ensure strict security and service isolation, each component runs within its own dedicated, unprivileged LXC container: one for the Astro application server, one for the Nginx reverse proxy, and a separate instance hosting Forgejo for local version control. Public traffic is routed securely through Cloudflare's edge network to mask the host IP and prevent automated scanning.

My role on this project was closer to being a Product Owner or System Architect rather than a traditional coder. Because I am not a web designer, I translated the design visions from my head into descriptive prompts for an AI assistant to generate the site code. The compiled assets were then uploaded manually to the server using WinSCP.

THE TECHNICAL HURDLES AND AI WORKFLOW

Working with AI on a comprehensive web project introduced significant file management challenges. Because the AI could only see the code fragments pasted into the chat window, it lacked the broader context of my local folder structure. This led to frequent confusion regarding where specific files belonged, particularly when balancing assets in the public directory versus components in the source folder.

To combat this, I developed a specific strategy. Instead of copying and pasting disjointed code snippets into existing files—which routinely cause syntax errors, I used the AI as a global refactoring engine. Whether updating a complex Astro component or modifying Markdown frontmatter metadata, I passed the entire file context to the AI and had it rewrite the code completely. This top-down rewrite strategy kept the codebase clean, preventing mismatched brackets and broken imports before the code ever left the chat interface.

THE CRITICAL INCIDENT: DATA LOSS

Despite running a local instance of Forgejo/Gitea inside my Proxmox cluster specifically for version control, I fell into a hasty rhythm of modifying files locally and immediately uploading them via WinSCP. Because I bypassed my repository tracking, I had no safety net when a major file corruption occurred during a complex design update. I forgot to back up the working configuration, lost the file integrity, and was forced to revert the site layout and start over from scratch.

> **Root Cause Analysis:** The site configuration corrupted due to an unverified code placement. This could not be rolled back because the saved local file immediately overwrote the server copy via WinSCP. No secondary backup existed because version control tracking had not been initialized for the directory, caused by prioritizing rapid local iteration over established infrastructure workflows.

LESSONS LEARNED AND NEXT ACTIONS

This incident highlighted the irony of underutilizing my own home lab infrastructure. Moving forward, the project workflow is being completely restructured with the following safeguards:

First: Manual WinSCP drag-and-drop deployments are banned. Every single code alteration must be committed locally to the project repository.

Second: I am establishing an automated local GitOps pipeline. By utilizing native webhooks inside my local Forgejo instance, pushing code to the repository will automatically trigger a lightweight background daemon on the web server container. This daemon will execute a secure shell script to pull the latest main branch, install dependencies, and run the Astro build command locally.

This transformation turns a painful data loss incident into a highly resilient, automated, and secure self-hosted architecture.