

# **ClusterGit**

## Vision Document

Version 1.0

Team Members:

Lauren Robison

Conner Reiter

Sean Sikora

Brandon Springer

Sasha Tapinsh

Title: Preliminary Vision Document - Team#12	
Version: 1.0	Date:10/10/2025

## Revision History

Date	Version	Description	Name(s)
9/25/2025	0.1	Document creation	Lauren Robison
10/3/25	0.2	Audit	Brandon Springer, Conner Reiter, Sasha Tapinsh
10/03/25	0.3	Added Glossary and References	Sean Sikora

## Table of Contents

<b>1. Introduction.....</b>	<b>3</b>
1.1. Overview.....	3
1.2. Definitions, Acronyms, and Abbreviations.....	3
1.3. References.....	5
<b>2. Problem Statement.....</b>	<b>6</b>
Current Situation:.....	6
Problem:.....	6
Proposal:.....	6
Predicted Outcome:.....	6
<b>3. Stakeholder and User Descriptions.....</b>	<b>7</b>
3.1. Stakeholder Summary.....	7
3.2. User Summary.....	8
3.3. User Environment.....	8
Student User.....	8
Instructor User.....	8
Admin User.....	9
3.4. Operating Environment.....	9
Hardware Environment.....	9
Software Environment.....	9
Networking Environment.....	9
3.5. Key Stakeholder or User Needs.....	10
<b>4. Product Overview.....</b>	<b>11</b>
4.1. Overview & Scope.....	11
4.2. Assumptions & Dependencies.....	11
4.3. Product Features.....	12
Critical Features.....	12
Optional Features.....	12
4.4. SWOT Analysis:.....	12
Strengths.....	12
Weaknesses.....	13
Opportunities.....	13
Threats.....	13

Title: Preliminary Vision Document - Team#12	
Version: 1.0	Date:10/10/2025

# 1. Introduction

## 1.1. Overview

ClusterGit is a self-hosted, distributed file submission application built on a Raspberry Pi 5 cluster. The system addresses the challenges of handling very large coursework files (2-8 GB per submission, totaling several terabytes per course), which cannot be managed effectively on platforms like GitHub due to strict size and encryption limitations.

By layering Git with Git-annex and deploying containerized services with K3s and Longhorn, the platform supports multi-gigabyte uploads, encrypted storage and transfers, replication, rollback, and long-term accessibility.

The project will be implemented in two phases:

- Phase 1 (Infrastructure): Establish cluster, distributed storage, Git-annex workflows, and prove support for real-world file sizes.
- Phase 2 (Application Layer): Add user authentication, encryption, a web-based UI, and usability features.

The final deliverable will be a low-cost, open-source submission platform that allows instructors and students to manage coursework reliably without cloud dependency.

The following document explores the vision of the project, namely:

- Statement of the problem and the proposed solution
- Description of the project stakeholders and the application's users
- Major features and optional requirements
- Project dependencies and priorities

## 1.2. Definitions, Acronyms, and Abbreviations

- **Back End** of a web application includes the server-side logic of the website that powers it from behind the scenes. It communicates information between the front-end and database.
- **Bare Metal Programming** is software that runs directly on the hardware without an operating system, giving full control over the processor, memory, and peripherals, commonly used in embedded systems and low-level Raspberry Pi setups.
- **Branch** is an independent line of development in a Git repository, allowing new features or fixes to be worked on without affecting the main codebase.
- **CI/CD Pipeline** is a sequence of steps that automates the building, testing, and deployment of software, ensuring reliable delivery.

Title: Preliminary Vision Document - Team#12	
Version: 1.0	Date:10/10/2025

- **Cluster** refers to a group of interconnected Raspberry Pi's (or other computers) that work together as a single system to improve performance, availability, and scalability.
- **Commit** is a recorded snapshot of changes in a Git repository, forming part of the project's history.
- **Container** is a lightweight, portable unit that packages an application and its dependencies, ensuring consistency across environments.
- **Continuous Deployment (CD)** is the automated release of code changes to production after passing tests and reviews.
- **Continuous Integration (CI)** is a development practice where code changes are automatically built and tested to detect issues early.
- **Database** is an organized collection of data that can be accessed, managed, and updated electronically.
- **Docker** is a platform for developing, shipping, and running applications inside lightweight containers that package code and dependencies together.
- **Front End** of a web application is the part of the website that a user interacts with, including the UI.
- **Git** is a distributed version control system that allows developers to track changes in source code, collaborate on projects, and manage different versions of files across multiple contributors.
- **Git-annex** is a tool that extends Git by allowing large files or binary data to be managed outside the main repository while still tracking them with Git.
- **Helm** is a package manager for Kubernetes that simplifies the deployment and management of applications in clusters.
- **Ingress** in Kubernetes refers to an API object that manages external access to services in a cluster, typically via HTTP or HTTPS.
- **K3s** is a lightweight Kubernetes distribution optimized for resource-constrained environments such as Raspberry Pi clusters.
- **Kubernetes orchestration** is the automated management of containerized applications across clusters, handling deployment, scaling, and maintenance tasks.
- **Load Balancer** distributes network traffic across multiple servers to improve reliability and performance.
- **Longhorn** is a cloud-native distributed block storage system for Kubernetes that provides persistent storage for containers.
- **Merge** is the process of integrating changes from one branch into another within a Git repository.
- **MetalLB** is a load-balancer implementation for Kubernetes clusters that allows services running inside the cluster to be accessed from outside networks.
- **Persistent Volume (PV)** is a storage resource in Kubernetes that allows data to persist beyond the lifecycle of a container or pod.

- **Pod** is the smallest deployable unit in Kubernetes, containing one or more containers that share resources.
- **Pull Request** is a request to merge changes from one branch into another, often used in collaborative projects for code review.
- **Raspberry Pi** is a low-cost, credit-card-sized computer designed for educational use, prototyping, and lightweight computing projects.
- **Repositories** are storage locations for code, documentation, and files that track versions of a project, often managed with version control systems like Git.
- **Repos** is a common shorthand for repositories, typically referring to code storage locations managed with Git.
- **SSH** (Secure Shell) is a protocol for securely accessing and managing devices over a network through encrypted communication.
- **Supabase** is an open-source backend-as-a-service platform that provides database, authentication, storage, and real-time subscriptions on top of PostgreSQL.
- **UI** stands for User Interface and is the point of contact with which users interact with a system.
- **Ubuntu** is a popular Linux-based operating system known for its ease of use, stability, and wide community support.
- **Version Control** is the practice of tracking and managing changes to software code, enabling multiple developers to collaborate and revert to previous versions when necessary.
- **Web Portal** is a website that collects information from other sources and presents the most relevant information to users in a consistent manner.

### 1.3. References

- Docker Documentation: <https://docs.docker.com>
- k3s Documentation: <https://docs.k3s.io>
- Raspberry Pi Documentation: <https://www.raspberrypi.com/documentation>
- Longhorn Documentation: <https://longhorn.io/docs/1.10.0>
- MetalLB documentation: <https://metallb.io>
- Ubuntu Documentation: <https://help.ubuntu.com>

## 2. Problem Statement

### Current Situation:

In game development and similar technical courses, students will regularly produce massive projects using tools like Unity or Blender. A single assignment can range from 2-8 GB, and across a class of 45 students with six assignments, total data

Title: Preliminary Vision Document - Team#12	
Version: 1.0	Date:10/10/2025

volume can exceed 3 TB. When replication for fault tolerance is factored in, requirements can climb over 6 TB. Currently, many instructors rely on GitHub or cloud-based platforms for submissions, but these solutions still have significant limitations.

### Problem:

Existing submission platforms such as GitHub restrict file size to 100 MB per file and repository capacity to about 10 GB per repo. Cloud services, while capable of handling large files, often come with high costs, potential security concerns, and restrictions from institutional IT policies. As a result, instructors struggle to manage large-scale coursework submissions, while students face unreliable or overly complicated workflows for submitting their assignments.

### Proposal:

To address these issues, ClusterGit will implement a self-hosted, distributed file submission system built on a Raspberry Pi cluster. The platform layers Git with Git-annex for large file support and uses K3s with Longhorn for container orchestration, replication, and rollback functionality. This ensures secure, reliable handling of multi-gigabyte files while remaining cost-effective and compliant with academic needs.

### Predicted Outcome:

When completed, ClusterGit will provide a low-cost, scalable, and open-source submission platform that:

- Supports multi-gigabyte file uploads and downloads.
- Ensures redundancy and recovery across Raspberry Pi nodes.
- Secures files through encryption at rest and in transit.
- Provides long-term accessibility for both students and instructors.

This solution will eliminate GitHub's size restrictions, reduce reliance on costly cloud services, and empower instructors with a reliable, self-sufficient platform for managing coursework at scale.

## 3. Stakeholder and User Descriptions

### 3.1. Stakeholder Summary

*Everyone with an interest in or influence on the project.*

Stakeholder	Position	Responsibilities
Professor Jay Johns (Sponsor/	Faculty sponsor and advisor overseeing project scope, progress,	<ul style="list-style-type: none"> <li>• Provide project requirements and scope.</li> <li>• Aid team in completing the project through mentorship and feedback.</li> </ul>

Title: Preliminary Vision Document - Team#12	
Version: 1.0	Date:10/10/2025

Advisor)	and deliverables.	<ul style="list-style-type: none"> <li>● Evaluate project deliverables and prototype performance.</li> <li>● Monitor progress and ensure alignment with academic expectations.</li> <li>● Provide resources (lab space, hardware access, course alignment).</li> </ul>
Project Team (Developers/ Admins)	Student developers and maintainers of the ClusterGit system.	<ul style="list-style-type: none"> <li>● Build, configure, and deploy the Raspberry Pi cluster.</li> <li>● Implement Git-annex workflows, containerization, and distributed storage.</li> <li>● Monitor performance, troubleshoot errors, and apply upgrades.</li> </ul>
Instructors (Primary)	Faculty member assigning coursework and reviewing submissions.	<ul style="list-style-type: none"> <li>● Receive, review, and archive large student project files.</li> <li>● Ensure grading integrity by having reliable access to all submissions.</li> <li>● Provide feedback to improve the system's usability.</li> </ul>
Students (Primary)	Course participants submitting assignments, often multi-GB Unity or Blender projects.	<ul style="list-style-type: none"> <li>● Upload large projects through CLI or web UI.</li> <li>● Follow submission procedures reliably.</li> <li>● Ensure files are submitted before deadlines.</li> </ul>
Institutional Information Technology (Secondary)	University IT department responsible for compliance, networking, and security.	<ul style="list-style-type: none"> <li>● Approve and enforce network/security policies.</li> <li>● Provide guidance on compliance and infrastructure standards.</li> <li>● Monitor integration with campus systems to avoid security risks.</li> </ul>

## 3.2. User Summary

*Those who directly interact with the system's interface or workflows.*

Name	Description	Responsibilities
Students	End users who submit projects via command line (Git-annex) or a user-friendly web interface.	<ul style="list-style-type: none"> <li>● Upload multi-gigabyte project files reliably.</li> <li>● Verify submissions are complete and correct.</li> <li>● Report issues encountered during submission.</li> <li>● Follow deadlines and submission guidelines.</li> </ul>

Title: Preliminary Vision Document - Team#12	
Version: 1.0	Date:10/10/2025

Instructors	Faculty users who collect, review, and archive assignments from the system.	<ul style="list-style-type: none"> <li>● Bulk download submissions for grading.</li> <li>● Access submission history and metadata (who submitted, when).</li> <li>● Provide feedback to the project team for improvements.</li> <li>● Archive submissions for long-term access.</li> </ul>
Admins	System administrators (initially the project team, later possibly faculty advisor).	<ul style="list-style-type: none"> <li>● Oversee system operation and cluster health.</li> <li>● Manage replication and redundancy settings.</li> <li>● Troubleshoot technical issues and apply software/hardware updates.</li> <li>● Ensure uptime and reliability of the system.</li> <li>● Implement and monitor user authentication and security.</li> </ul>

### 3.3. User Environment

#### Student User

Students can create an account and log in with their credentials to the web interface. They will submit large project files through either a command-line Git-annex workflow or a web-based submission interface. To avoid overwhelming local storage, students can configure their setup so that large files are uploaded to the cluster and removed from their laptops after confirmations, keeping only Git metadata. Students typically work on personal laptops, personal desktops, or school desktops and may upload from either campus networks or home networks. The environment must provide an intuitive, reliable, and secure submission process with progress indicators and confirmation that submissions are complete.

#### Instructor User

Instructors create an account and log in with their credentials through the web interface. They will access submissions via the web interface or by downloading zip archives of student work. The system provides options for per-student or per-assignment zip downloads, enabling instructors to efficiently retrieve the data they need for grading. Instructors may also use the web interface to view submission metadata such as student name, time stamps, and confirm submission status. The environment should allow for organized retrieval, batch download options, and reliable archiving to support grading workflows.

#### Admin User

The admins can log in with elevated credentials. They manage the system through SSH, K3s dashboards, Git-annex commands, and monitoring tools. Admins could oversee

Title: Preliminary Vision Document - Team#12	
Version: 1.0	Date:10/10/2025

replication, encryption, and rollback settings, ensuring redundancy and availability. They primarily operate in the on-campus lab environment where the Raspberry Pi 5 cluster is physically located, but may need secure remote access for maintenance and monitoring. Their environment would support troubleshooting, performance monitoring, and user management to keep the system reliable at scale.

### 3.4. Operating Environment

The ClusterGit system operates in a hybrid environment due to creating an infrastructure then applying an application to utilize.

#### Hardware Environment

- **Cluster Nodes:** Five Raspberry Pi 5 boards with 8GB RAM each, with power supply.
- **Storage:** M.2 Hats for SSDs. Currently have 2 x 2TB, expandable for future needs.
- **Networking:** Local network switch to interconnect the cluster nodes.
- **Client Devices:** Students and instructors will use laptops or desktops for accessing.

#### Software Environment

- **Operating System:** Ubuntu Server 24.04 running on Raspberry Pi 5 nodes.
- **Orchestration:** K3s (lightweight Kubernetes) for containerized service management.
- **Storage Layer:** Longhorn for distributed storage, replication, and rollback.
- **Version Control:** Git layered with Git-annex for large file handling.
- **Authentication & Metadata:** Supabase (PostgreSQL backend) for login, user roles, and submission metadata.
- **User Interface:** Web portal for students and instructors, integrated with Supabase authentication. Git/CLI tools are also available for advanced users.
- **Admin Tools:** SSH access to nodes, K3s dashboard, and monitoring utilities.

#### Networking Environment

- **Cluster Connectivity:** All nodes connected via a private on-campus LAN, ensuring high-speed replication and scheduling.
- **User Access (Students & Instructors):** Through CLI, available on-campus or remotely via secure VPN or HTTPS.
- **Admin Access:** Through SSH, restricted to project team members.

### 3.5. Key Stakeholder or User Needs

Need	Priority	Concerns	Proposed Solution
Demonstrate a functioning distributed	Critical	Must prove system can handle real-world large file submissions.	Deploy and test a Raspberry Pi 5 cluster with K3s + Longhorn + Git-annex using

Title: Preliminary Vision Document - Team#12	
Version: 1.0	Date:10/10/2025

submission prototype to sponsor			multi-GB uploads and downloads.
Students can upload large files reliably	<b>Critical</b>	Uploads may fail on slower networks; interruptions could corrupt files.	Provide resumable uploads and Git-annex integration with status/confirmation messages.
Instructors can download and grade submissions efficiently	<b>Critical</b>	Downloading many large projects one-by-one is inefficient.	Provide ZIP download bundles (per student or per assignment).
Students and instructors can access the system from both on-campus and remote locations	<b>High</b>	Campus IT may restrict remote access or ports.	Provide a secure web portal with HTTPS; support VPN-based access if required by IT.
Authentication and secure access for all users	<b>High</b>	External dependencies (Supabase) may cause downtime; weak auth risks data leaks.	Use Supabase for hosted authentication and metadata; plan a local Postgres fallback if the internet is unavailable.
System ensures redundancy and fault tolerance	<b>High</b>	Node or storage failure could result in data loss.	Use Longhorn replication and rollback features to keep multiple copies and allow quick recovery.
Storage can scale with class size and future use	<b>Medium</b>	Assignments across multiple courses could exceed initial SSD capacity.	Support adding more SSDs or integrating with an external NAS.
Admins can securely maintain and monitor the system	<b>Medium</b>	Without secure access, cluster health and troubleshooting could be compromised.	Provide SSH access via VPN; use K3s dashboard and monitoring tools.
<b>Optional: Web-based student IDE for low-resource devices</b>	<b>Optional - Low</b>	Developing an in-browser IDE could require significant additional setup and maintenance	If time permits, integrate a lightweight, browser-based IDE hosted on the cluster so students can edit/test projects without requiring powerful personal laptops.

Title: Preliminary Vision Document - Team#12	
Version: 1.0	Date:10/10/2025

## 4. Product Overview

### 4.1. Overview & Scope

The goal of the project is to develop a self-hosted, distributed file submission platform built on a Raspberry Pi 5 cluster. The system addresses the challenges of coursework submissions in game development and other technical courses where project files can run to multiple GB in size, far exceeding GitHub's limits.

The system will provide two main user views:

- Student View: Students can log in to a web portal, submit large assignments, and verify successful uploads.
- Instructor View: Instructors can log in, view student submissions, and download ZIP archives for grading or long-term archiving.

At the infrastructure level, the system uses K3s for container orchestration, Longhorn for distributed storage with replication and rollback, and Git-annex for large file handling. This development provides a scalable, low-cost, open-source solution for managing coursework submissions that require terabytes of storage, ensuring reliability, accessibility, and academic usability.

### 4.2. Assumptions & Dependencies

#### Assumptions

- Raspberry Pi 5 devices and associated storage hardware will remain available and operational throughout the project.
- Supabase will remain supported and provide stable service during the project.
- Institutional IT will approve the required networking setup for student and instructor access.

#### Dependencies

- Raspberry Pi 5 cluster hardware (Pis, SSDs, networking).
- K3s orchestration layer for container deployment.
- Longhorn for distributed storage and redundancy.
- Git-annex for large file storage and versioning.
- Supabase/PostgreSQL for authentication and metadata.
- Internet connectivity for Supabase access, container pulls, and remote student/instructor use.

## 4.3. Product Features

### Critical Features

- Secure login systems
- Student Features
  - Create an account and log in.
  - Submit large assignments (multi-GB files).
  - Verify submission completion and history.
- Instructor Features
  - Create an account and log in.
  - View students' submissions and metadata.
  - Download submissions as ZIP archives (per student or per assignment)
- Admin Features
  - SSH access for cluster management.
  - Monitor replication, storage usage, and cluster health.
  - Troubleshoot and apply upgrades.

### Optional Features

- 3D-printed frame for Raspberry Pi nodes and intranet setup for local access.
- Virtual IDE for distributed development on low-cost devices.
- Quality assurance and maintainability tools to ensure long-term file integrity and system scalability, like NAS integration.
- Replication monitoring and auto-healing for lost nodes to improve resilience.

## 4.4. SWOT Analysis:

### Strengths

- Low-cost, scalable infrastructure using Raspberry Pi 5 hardware.
- Open-source stack (Git-annex, K3s, Longhorn, Supabase).
- Fault tolerance through Longhorn replication and rollback.
- Supports multi-gigabyte files that exceed GitHub and Brightspace limits.
- Self-hosted design ensures long-term access without cloud dependency.

### Weaknesses

- Limited processing and memory compared to enterprise servers.
- Dependence on Supabase for authentication may cause downtime.
- Steep learning curve: team's limited experience with K3s and distributed systems.
- Complexity of setup may be difficult for non-technical instructors.

Title: Preliminary Vision Document - Team#12	
Version: 1.0	Date:10/10/2025

## **Opportunities**

- Collect direct feedback from students and instructors to refine usability.
- Expand usage beyond game development to other large-file courses (design, media, research datasets).
- Integrate external NAS or hybrid cloud for scalable long-term storage.
- Add optional Virtual IDE for students with low-resource devices.
- Enhance resilience with optional replication monitoring
- Open-source release could benefit other institutions.

## **Threats**

- Institutional IT restrictions on VPNs or external network access.
- Hardware or SSD failures could cause downtime without proactive planning.
- Internet or Supabase outages may disrupt logins and metadata access.
- Scope creep from optional features (IDE, auto-healing, extra security) could delay core functionality.