FERMILAB-TM-2735-SCD

GlideinMonitor

Marco Mambelli, Fermilab
Thomas Hein, UIC-Fermilab

1 Introduction

1.1 Purpose

This document describes GlideinMonitor. GlideinMonitor is a Web application that allows to view GlideinWMS's Glidein log files: it provides a user interface, tools to do quick searches and to decode the log content; it provides an efficient managed archive of the log files and a framework to add log processing, e.g. log sanitation.

The system does not provide extensive log analysis, e.g. like Elastic Search. The implementation of log pre-processing (e.g. log sanitation) is not covered in this description. The system is designed with security in mind but the choice and implementation of a user authentication and authorization layer is not covered.

1.2 Scope

This document will describe GlideinMonitor. GlideinMonitor is a Web application that allows to view GlideinWMS's Glidein log files: it provides a user interface, tools to do quick searches and to decode the log content; it provides an efficient managed archive of the log files and a framework to add log processing, e.g. log sanitation.

The system does not provide extensive log analysis, e.g. like Elastic Search. The implementation of log pre-processing (e.g. log sanitation) is not covered in this description. The system is designed with security in mind but the choice and implementation of a user authentication and authorization layer is not covered.

1.3 Rationale

GlideinMonitor will help greatly troubleshooting operations of GlideinWMS operators and developers. This includes Factory and VO Frontend operators, VO managers, the software development team. GlideinMonitor simplifies the distribution of the log files and the access to the correct logs needed to understand and fix problems with the GlideinWMS system. The access to this information is currently very labor intensive:

- The interested party has to request a copy of a specific log files to the Factory operators
- Log files have to be searched with Unix command line tools (grep, …)

This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

- Embedded components have to be extracted and decoded manually on a host where the GlideinWMS Factory tools are installed
- The relevant information has to be isolated and if other log files are needed the whole process needs to be repeated

1.4 Terminology

Glideins are pilot jobs sent to distributed computing nodes to test and setup the nodes so that user jobs will run as desired. These also collect information about the nodes and the jobs running on them.

GlideinWMS is the Glidein based Workload Management System [https://glideinwms.fnal.gov/], a system to provide simple access to distributed computing resources. It has three main components: the Glideins, the Factory producing Glideins and the Frontend calculating the Glideins required to satisfy the user's needs.

GlideinMonitor is the Web application described in this document

Indexer is the GlideinMonitor component that collects, prepares and archives new log files

Webserver is a Web application serving and decoding the archived log files. It provides simple log search capabilities and client-side applications to decode and search the log files [https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/]. It provides also a RESTful [https://en.wikipedia.org/wiki/Representational_state_transfer] interface to the log files.

Pre-processing in this document refers to any operation that is performed on the log files before archiving them and making them available via the Webserver. This may include for example log anonymization and sanitation, e.g. to remove from the log files Personally Identifiable Information (PII [https://en.wikipedia.org/wiki/Personal_data])

2 Overview

GlideinMonitor automatically indexes job files, currently those files include ".err" and ".out", and saves them in a compressed archive for long term storage. While searching and compressing them, basic information within these files such as their size, and date generated within the file is stored in a database.

GldeinMonitor contains an authenticated website dashboard for users to interact with. This dashboard allows for searching for jobs in a table based format. Specific queries can be performed such as providing a datetime range and selecting which entries the user is interested in.

Once a particular job has been selected by the user, the website will provide general information about that job such as the entry, factory, time created, and more. Links are also provided to either view or download the raw files themselves, an archive of the file, or even compressed logs within them such as Condor logs.

The web dashboard itself is powered by a REST based API that other applications can use as well. The API provides various endpoints providing JSON responses. These endpoints provide information such as the output from the table found in the homepage of the dashboard, general job view information, ability to download the archived job file itself, and more.

The server and indexing system powering it is made using Python with typical installations only requiring at minimum a Flask installation. More powerful installations can make use of external database support such as MariaDB, separate job file indexing and web server instances, and more robust web servers such a NginX powering Flask.

3 Requirements

This server requires a few basic read and write permissions to folders and to the desired database. A more in depth explanation is stated in Major Inputs and Outputs

The web server will only require read level access but the index script requires write level access.

The index script also requires write level access to a database. This database can either be a MariaDB instance or a SQLite database. The web server will only require read level access to this same database.

Both the index script and the web server requires a Python v3.7.x installation. The web server further needs the Flask package to be installed as well.

3.1 Actors

External actors depends primarily on the setup. For any setup, the Factory log files must be generated and published for the GlideinMontitor indexing script to access. Some setups may utilize external databases such as an offsite MariaDB instance that would also pose as an external actor. User's accessing the dashboard would be additional external actors as well.

3.2 The Major Inputs and Outputs

Read access is required for a folder containing factory logs. This folder can be a mount, a synced folder, or the actual folder of the factory logs. The index script will access this folder each time it is run, however, the web server will never access this folder.

The index script also requires a place to save logs and a long term storage folder for archives. The long term archive folder must be in a place that the web server can access directory.

3.3 Behavioral requirements (use cases)

3.3.1 Index System

Task	Index Job Files
Level	GlideinMonitor archiving job files for long term storage and writing basic information about each job in a database
Goal	Stated as a short active verb phrase
Actor	Cron
Trigger	Scheduled Interval to find new job files
Preconditions	Job file accessible, database & archive location writable, and index system not already running.
Post-conditions	N/A
Description	1. Cron launches index system 2. Index system searches for new/modified job files and archives them accordingly 3. Index system updates database
Nonstandard Flow	- Lock file present, index system does not run - Missing out or err file to a job, index system writes to log and continues - Index system cannot write to disk, index system writes to log and exits - Index system cannot write to database, index system writes to log and exists
Comments	The index system should be placed on the cron and run automatically. The schedule should be less than the time of the lifecycle of job files available on the factory (job files from being added to the factory to being deleted).

3.3.2 Web Query

Task	Querying and viewing job files
Level	A user searching for job files and viewing/downloading their contents

Goal	Allow a user to find job files quickly and get specific information within their contents quickly
Actor	User viewing information about jobs
Trigger	User using the web interface
Preconditions	GlideinMonitor running, web server having access to job file archives & database, and user w/ proper credentials to dashboard
Post-conditions	N/A
Description	1. User types in credentials into dashboard http login 2. User creates query in table to find job(s) wanted 3. Selects wanted job id, loads job view page 4. User searches within content or opens/downloads individual logs
Nonstandard Flow	- User cannot find job - Incorrect credentials - Job file missing content/archive
Comments	The dashboard is a quick way to search for job files given a proper query. User's must have credentials to the web server before performing searches or using the API. The job view page allows for viewing of the condor extracted logs as well.

3.4 Constraints

Indexing speed is limited on how fast the server can open each .out/.err file for the basic information. This process starts with getting general information about the file before the index script opens the file. This general information is then checked against the database that will respond with whether or not the file needs to be updated.

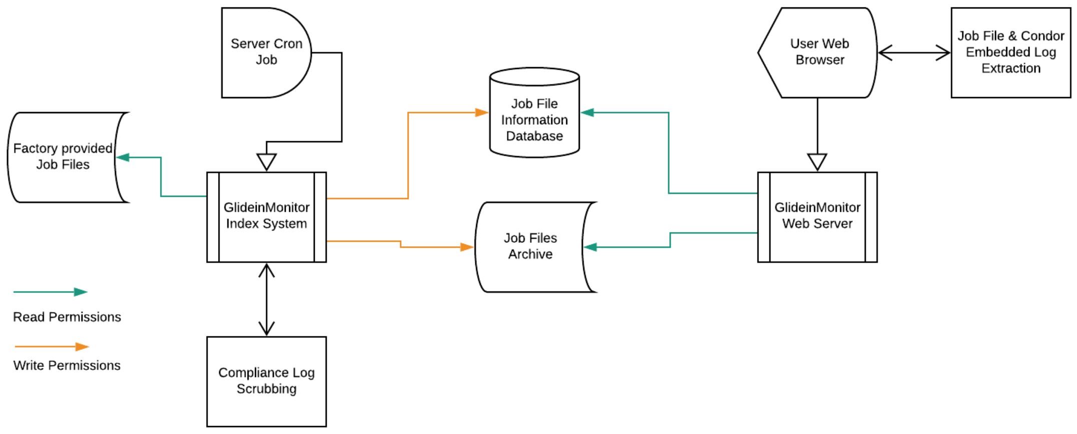
If the file needs to be updated (or created for the first time) in the archive, the index script will proceed to open the file accessing general information like creation time and searching for condor logs.

Web server constraints are dependent on the speed at which the database can be queried, the archives can be accessed and sent over to the client, and the number of clients using the server at once.

It is recommended for production use that NginX is used if there are a large number of clients separate from the factory operations staff. The package powering the web pages is called Flask, which includes a very simple web server that, according to their documentation, should only be used for development purposes.

4 Architectural Overview

4.1 General Overview


Server Cron Job
Factory provided Job Files
GlideinMonitor Index System
Job File Information Database
Job Files Archive
User Web Browser
Job File & Condor Embedded Log Extraction
GlideinMonitor Web Server
Compliance Log Scrubbing
Read Permissions
Write Permissions

The design of this project allows the web server and the index system to be completely isolated assuming they can share a file system for archived job files and a database.

While the index system is designed to run as lean as possible, it does contain modularity including the ability to send uncompressed logs in their human readable text form to check for sensitive information.

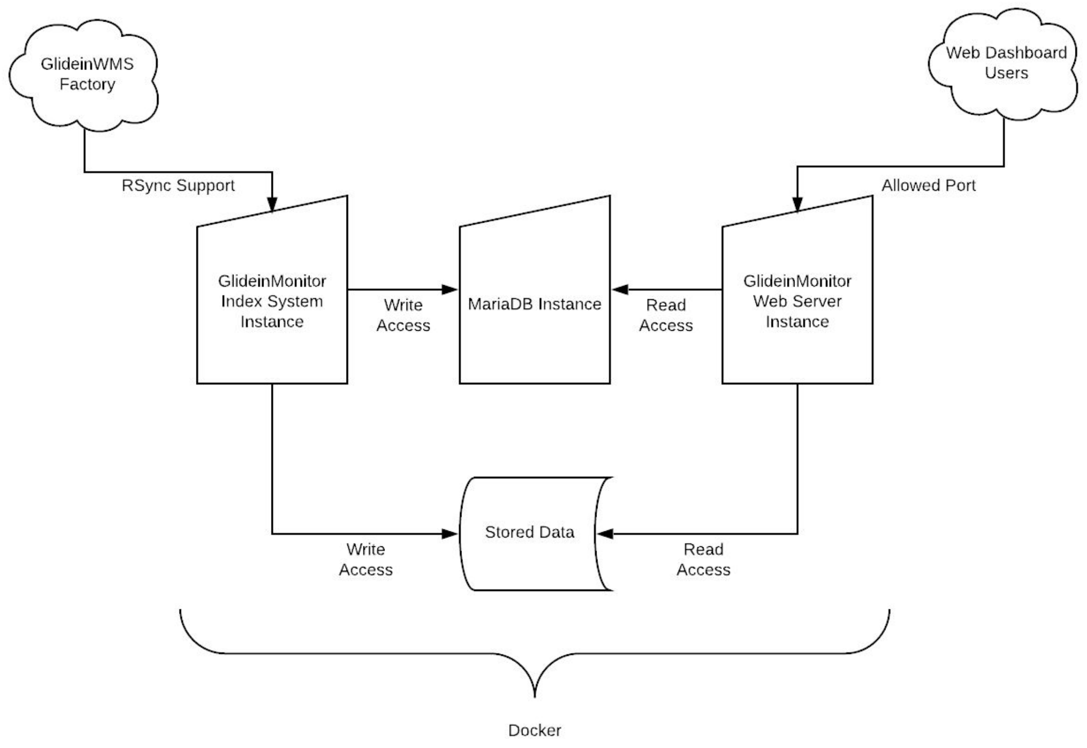
While the index system does need write access to the database and the location of where it sends archived job files, the web server and index system when fetching job files from the factory only requires read level access.

In order to lessen the load on the web server, the server itself only sends the archived job file as it sits on the disk. The web browser will go through the process locally of extracting the data, parsing the extracted files, and even parsing the hashed condor logs within them.

4.2 Functional Unit Process

4.2.1 GlideinMonitor Index System

4.3 Deployment scenario



5 Component Interfaces

Expand the interfaces of the components shown in the previous section. Include relationships to other components here.

How they talk together, provide SQL schema

6 Protocols

Describe known elements of any protocol involved in data exchanges, external or internal to this subsystem, and the types of messages or data that may be exchanged. This is distinguished from component interfaces right now – should it be?

Show important invocation or message exchange timing sequences here. Show what parts of the interfaces are used by other components.

rest api

7 Discussion

This section captures discussions and information that lead to the current architecture view and component organization.

7.1 Decisions and Choices

The GlideinMonitor contains two systems under the same repository, a web dashboard and an indexing script. The indexing script and the dashboard have been designed as lean as possible while giving options for added features. The project also is allowed to run side by side or on totally separate systems as well. The web dashboard itself, especially the job view page, has been created to handle zip extraction, log extraction, and condor log decompilation.

7.2 Rationale

The index system and web dashboard have different purposes and executions but do share many functions such as connecting to the database, configuration parsing, and logging. For

7.3 Implications resulting from Choices

Additional constraints that are imposed on the whole system or this system as a result of choices made here.

7.4 Resulting rules

Include all things that must be true in the system and rules that must be followed while the system is in operation. An example is that one worker node will only be assigned to one partition.

7.5 Constraints imposed on other systems

List what constraints this system imposed on other systems.

8 Testing considerations

Explain any load testing that must be performed to evaluate the performance of this system as a whole or parts within it. Suggestions for how to test (verification and validation) this system should be included. Ways of evaluating the performance of this system should be included here.