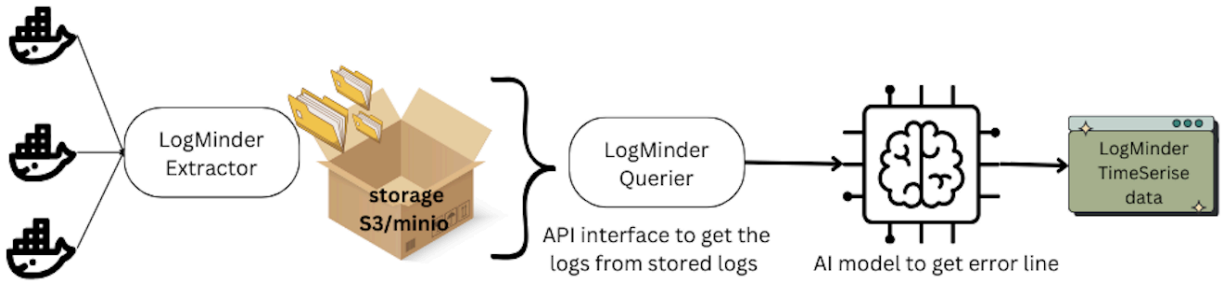
LogMinder

The **Logminder** project aims to analyze logs produced by Docker containers and extract information, particularly around error occurrences in the logs. The project consists of five primary components: log extraction, storage, querying, AI-driven error prediction, and time-series data analysis. By leveraging modern file storage solutions (e.g., MinIO, S3) and AI, Logminder offers a comprehensive system for log management, error prediction, and real-time analysis.

Use Case

- **Objective**: The goal is to identify error trends and provide analysis of container logs over time. This can help in identifying application issues, debugging, performance optimization, and preventing future problems.
- **User Scenario**: An engineer wants to monitor logs from multiple Docker containers, detect patterns, and analyze errors over time. By utilizing Logminder, they can visualize error trends, retrieve logs by time range, and use AI to predict if a log contains an error or not.



1. logminder-extractor

Description: The **logminder-extractor** is responsible for collecting logs from each container and storing them in a specified file storage system, such as S3, MinIO, etc.

How It Works:

- Periodically polls or subscribes to Docker events to collect logs from running containers.
- Logs are collected in real-time and are structured in a format like container-name/date/time.log.
- The extractor sends these logs to the configured storage (MinIO/S3).

Key Features:

- Real-time or batch log extraction.
- Support for multiple containers and efficient parallel extraction.
- Logs are stored based on container name and timestamp, allowing for easy retrieval.

Use Case Example: If there is a container running for a microservice, the logs for the last 24 hours can be fetched by **logminder-extractor** and saved to MinIO for future use in analysis or querying.

2. Storage

Description: The **Storage** component is where the logs collected by logminder-extractor are stored. This can be any file-storage system (S3, MinIO, etc.) configured to store logs in an organized manner based on container names and timestamps.

Key Features:

- Scalable storage, supporting a large number of containers and logs.
- Efficient directory structure: container-name/YYYY-MM-DD/HH-MM-SS.log for easy retrieval.
- Supports S3, MinIO, or other compatible storage solutions.

Implementation:

- MinIO or AWS S3 buckets store the logs using a structured path format.
- Ensure backup and redundancy for long-term storage.

Use Case Example: Logs generated by app-backend container on 2024-09-23 are stored under app-backend/2024-09-23/.

3. logminder-querier

Description: The **logminder-querier** is an API service responsible for retrieving logs based on queries like container name and time range. The API will allow users to access logs for a particular container within a specific time window.

How It Works:

- Accepts requests with parameters such as container name, start-time, and end-time.
- Fetches logs from the storage system based on the provided parameters.
- Returns the logs as a response to the user.

Key Features:

- Time-based log querying for specific containers.

- Efficient retrieval based on structured log paths.
- REST API for ease of integration with other tools or services.

Implementation:

- Use Python/Go to implement a REST API service.
- Integrate MinIO/S3 SDK to read logs from file storage based on container name and timestamp.

Use Case Example: A request for logs from app-backend container between 2024-09-23 00:00:00 and 2024-09-23 01:00:00 will fetch and return logs within this time frame.

4. AI Model

Description: The **AI model** is used to predict whether a particular log line is an error or not. This component helps in automating the detection of issues within logs using machine learning techniques.

How It Works:

- The model is trained using historical log data labeled as "error" or "non-error."
- For each log line retrieved by the querier, the AI model makes predictions to classify the line as error or non-error.

Key Features:

- Predicts log errors in real-time or batch processing.
- Can be integrated with logminder-querier to flag errors directly in query responses.
- Supports various machine learning models like SVM, Random Forest, or deep learning models trained for log anomaly detection.

Implementation:

- Use TensorFlow, PyTorch, or Scikit-learn for building the model.
- Train the model on historical log data that includes both error and non-error entries.
- Export the trained model and integrate it with the querying service to classify logs in real-time.

Use Case Example: When a engineer queries logs, the AI model flags certain log lines as potential errors based on patterns it has learned from previous log data.

5. logminder

Description: The **logminder** component provides an interface for interacting with the system. It offers the ability to perform data analysis on logs, generate CSV reports, and present time-series data visualizations.

How It Works:

- Fetches processed logs from logminder-querier and visualizes the data.
- Displays trends in error rates over time for each container.
- Allows the user to export data as CSV for offline analysis.

Key Features:

- UI for real-time analysis of logs and error trends.
- Time-series visualizations and dashboards.
- Export functionality to download logs or error analysis as CSV files.

Implementation:

- Use React, Angular, or similar for building the frontend.
- Backend integrates with logminder-querier to retrieve logs and error predictions.

Use Case Example: The DevOps engineer can use the UI to generate a time-series graph of error occurrences in the app-backend container over the past week.

Deployment

Pre-requisites:

- Docker installed on the system.
- MinIO/S3 setup for log storage.
- Kubernetes or Docker Swarm (optional) for managing containers.
- Python environment (for AI model training).

Steps:

1. **Setup MinIO/S3 Storage**:
    - Configure your file storage solution, such as MinIO or AWS S3, to store container logs.
    - Create bucket in minio/s3
    - Need access key and secret key
2. **Install and Run logminder-extractor**:

    - Pull logs from each container and push them to your storage using MinIO or S3.
    - https://github.com/jatin-jangir/logminder-extractor
    - Use above github to run logminder-extractor microservice
3. **Deploy logminder-querier**:
    - Deploy the API service to serve log requests.
    - https://github.com/jatin-jangir/logminder-querier
    - Use this repo to start the micro-service
4. **Train and Deploy AI Model**:
    - Train the AI model on historical logs and deploy it alongside the querier to classify log lines.

5. **Integrate logminder**:
    - We have all the microservices ready but this will be the last screw that holds everything together. Here we have to generate time series data for each container.

Future Improvements

- Integration with more advanced alerting systems for real-time error detection and notification.
- Deploy the UI dashboard that connects to the logminder-querier API for real-time log analysis and visualization.
- Advanced AI models for better log classification and anomaly detection.
- Support for more log sources beyond Docker and Kubernetes (system logs, etc.).