Grafana (<https://grafana.com/oss/loki/>)

Objective:

The objective of this Proof of Concept (POC) is to evaluate Grafana, a log processing, analysis, and monitoring tool, and compare it with other tools in terms of license cost, configuration effort, stability and support, and integration cost.

The POC aims to assess the tool's capabilities to filter logs, group logs from different sources by context, and provide analytics.

Scope:

The POC will focus on the following aspects:

* Analyzing license cost: Evaluate the licensing model and associated costs of Grafana.
* Assessing configuration effort: Document the steps required to configure Grafana for log processing, analysis, and monitoring.
* Evaluating stability and support: Examine the stability of Grafana and assess the support available from the community or the tool's developers.
* Investigating integration cost: Determine if any modifications are needed in the monitored applications to work with Grafana and identify the specific changes required.

Deliverables:

* Documentation: Prepare a comprehensive document outlining the steps to configure Grafana for log processing, analysis, monitoring, and a communication schema between the tool and the monitored systems.
* Demo: Develop a demo showcasing the functionality of Grafana, including log filtering, context-based log grouping, and analytics.

Evaluation Criteria:

* License Cost: Evaluate the affordability and suitability of Grafana's licensing model.
* Configuration Effort: Assess the complexity and ease of configuring Grafana for log processing, analysis, and monitoring.
* Stability and Support: Evaluate the stability of the tool and the availability of community support or official technical assistance.
* Integration Cost: Determine if any modifications are required in the monitored applications and identify the specific changes needed.

Desired log structure:

{

  "type": "error",

  "severity": "high",

  "message": "An unexpected error occurred while processing the request.",

  "machine": "server-01",

  "user": "john.doe@example.com",

  "timestamp": "2023-07-13T10:30:00Z",

  "process": "payment-service",

  "thread": "1234",

  "context": "payment-12345",

  "source": "payment-processor",

  "requestId": "abcd1234",

  "correlationId": "xyz789",

  "exception": {

    "message": "Division by zero",

    "code": "DIV\_ZERO",

    "stackTrace": "..."

  },

  "logContext": {

    "orderId": "54321",

    "amount": 100.0

  }

}

1. type: Indicates the type of log entry, such as "error," "warning," "info," etc.
2. message: Describes the log message, providing relevant details about the event or issue.
3. machine: Specifies the machine or server where the log originates.
4. user: Represents the user associated with the logged activity, if applicable.
5. timestamp: Records the timestamp when the log entry was generated.
6. process: Identifies the specific process or application component generating the log.
7. thread: Indicates the thread or execution context within the process.
8. context identification: In scenarios where multiple microservices or components are involved in an operation, include a context identification field to correlate logs across services.
9. severity: Specifies the severity level of the log entry, such as "critical," "error," "warning," "info," or "debug." This helps prioritize and filter logs based on their criticality.
10. source: Indicates the specific component or module within the process that generated the log entry. This can be useful in large-scale applications with multiple subsystems.
11. request ID: If the log entry is related to a specific user request or transaction, including a unique identifier can help track and correlate logs associated with that particular request.
12. correlation ID: In distributed systems or microservices architectures, a correlation ID can be used to correlate logs across different services that are part of the same operation or transaction.
13. exception details: If the log entry is an error or exception, including additional details such as the stack trace or error code can assist in troubleshooting and debugging.
14. log context: Any additional contextual information that is relevant to the log entry. This can include specific parameters, data, or states associated with the logged event.

Timeline: 6 days to complete the POC, including documentation and the demo. The timeline will be as follows:

* 1 day: Research and familiarize with Grafana and its features.
* 2-4 days: Conduct testing, configuration, and evaluation of Grafana.
* 2 days: Prepare the documentation and demo based on the findings.

Documentation: Configuring Grafana for Log Processing, Analysis, Monitoring, and Communication

**Set-up using Docker:**

**Reference**: [Grafana Documentation - Docker Installation](https://grafana.com/docs/loki/latest/installation/docker/)

**Steps:**

1. Download Loki Configuration File:
   * Visit the following URL to download the Loki configuration file:
   * Replace "version" in the URL with the desired version you plan to use.
   * Save the content of this file as "loki-config.yml".
   * URL: [Loki Configuration File](https://raw.githubusercontent.com/grafana/loki/v2.8.0/cmd/loki/loki-local-config.yaml)
2. Download Promtail Configuration File:
   * Visit the following URL to download the Promtail configuration file:
   * Replace "version" in the URL with the desired version we plan to use.
   * Save the content of this file as "promtail-config.yml".
   * URL: [Promtail Configuration File](https://raw.githubusercontent.com/grafana/loki/v2.8.0/clients/cmd/promtail/promtail-docker-config.yaml)
3. Create a Docker Compose File:
   * Create a new Docker Compose file and define the services as follows:

version: "3"

networks:

  loki:

services:

  loki:

    image: grafana/loki:2.4.0

    volumes:

      - ./loki/loki-config.yml:/etc/loki/loki-config.yml

    ports:

      - "3100:3100"

    restart: unless-stopped

    command: -config.file=/etc/loki/loki-config.yml

    networks:

      - loki

  promtail:

    image: grafana/promtail:2.4.0

    volumes:

      - ./promtail/promtail-config.yml:/etc/promtail/promtail-config.yml

      - ./promtail:/var/log

    restart: unless-stopped

    command: -config.file=/etc/promtail/promtail-config.yml

    networks:

      - loki

  grafana:

    image: grafana/grafana:latest

    user: "1000"

    volumes:

      - ./grafana:/var/lib/grafana

    ports:

      - "3000:3000"

    restart: unless-stopped

    networks:

      - loki

* + In the above configuration, we define three services: loki, promtail, and grafana.
  + The services are kept under the same network named loki.
  + Volumes are set for promtail and loki to link the configuration files by specifying their locations inside the Docker containers.
  + The location of the logs folder is added under promtail's volumes and mapped to Docker's /var/log.
  + The command is set to run the config file from the Docker volume.
  + Appropriate ports are exposed for loki and grafana.

1. Compose the Docker File.
2. Check Loki Status:
   * Access [**http://localhost:3100/ready**](http://localhost:3100/ready)in your browser and wait until Loki is ready.
3. Check Loki Metrics:
   * Access [**http://localhost:3100/metrics**](http://localhost:3100/metrics)in your browser to verify the availability of metrics.
4. Login to Grafana:
   * Access [**http://localhost:3000/login**](http://localhost:3000/login)in your browser.
   * Use "admin" as the username and password.
5. Add Datasource in Grafana:
   * Click on "Add Datasource" in Grafana.
   * Select "Loki" as the datasource type.
   * Set the URL as **http://loki:3100/** or use the URL mentioned in the "clients" section inside promtail-config.yml.
6. Explore Data in Grafana:
   * Open the Data Explorer in Grafana.
   * Select "filename" as the label and choose your log file.
   * Apply filters as needed and click on "Run Query" to analyze the data.