Grafana

This is some note, nothing serious. **DO NOT** cite from this article.

1. Grafana stack

Grafana offers many tools, which we've grouped into the following categories:

- The core Grafana stack: **LGTM** and the **Grafana Agent**
- Grafana enterprise plugins
- Incident response tools
- Other Grafana tools

1.1. The core Grafana stack

The core Grafana stack consists of **Mimir**, **Loki**, **Tempo**, and **Grafana**; the acronym **LGTM** is often used to refer to this tech stack.

Mimir

Mimir is a **Time Series Database** (**TSDB**) for the storage of metric data. It uses low-cost object storage such as S3, GCS, or Azure Blob Storage. Mimir is a fully Prometheus-compatible solution that addresses the common scalability problems encountered with storing and searching huge quantities of metric data.

> An active time series is a metric with a value and unique labels that has reported a sample in the last 20 minutes.

Loki

Loki is a set of components that offer a full feature logging stack. Loki uses lower-cost object storage such as S3 or GCS, and only indexes label metadata.

Log aggregation tools typically use two data structures to store log data. An index that contains references to the location of the raw data paired with searchable metadata, and the raw data itself stored in a compressed form.

> Loki differs from a lot of other log aggregation tools by keeping the index data relatively small and scaling the search functionality by using horizontal scaling of the querying component.

Tempo

Tempo is a storage backend for high-scale distributed trace telemetry, with the aim of sampling 100% of the read path. Like Loki and Mimir, it leverages lower-cost object storage such as S3, GCS, or Azure Blob Storage.

Tempo also offers the ability to generate metrics from spans as they are ingested; these metrics can be written to any backend that supports Prometheus remote write.

Grafana

Grafana has been a staple for fantastic visualization of data since 2014. It has targeted the ability to connect

to a huge variety of data sources from TSDBs to relational databases and even other observability tools. Grafana has over 150 data source plugins available. Grafana has a huge community using it for many different purposes. This community supports over 6,000 dashboards, which means there is a starting place for most available technologies with minimal time to value.

Grafana Agent

Grafana Agent is a collection of tools for collecting logs, metrics, and traces. There are many other collection tools that Grafana integrates well with.

1.2. Grafana Enterprise plugins

These are part of any paid subscription to Grafana.

The Enterprise data source plugins allow organizations to read data from many other storage tools they may use, from software development tools such as **GitLab** and **Azure DevOps** to business intelligence tools such as **Snowflake**, **Databricks**, and **Looker**. Grafana also offers tools to read data from many other observability tools, which enables organizations to build comprehensive operational coverage while offering individual teams a choice of the tools they use.

1.3. Grafana incident response and management

Grafana offers three products in the **incident response and management** (**IRM**) space:

- At the foundation of IRM are **alerting rules**, which can notify via messaging apps, email, or Grafana OnCall
- **Grafana OnCall** offers an on-call schedule management system that centralizes alert grouping and escalation routing
- **Grafana Incident** offers a chatbot functionality that can set up necessary incident spaces, collect timelines for a post-incident review process, and even manage the incident directly from a messaging service

1.4. Other Grafana tools

Grafana Labs continues to be a leader in observability and has acquired several companies in this space to release new products that complement the tools we've already discussed.

Faro

Grafana Faro is a JavaScript agent that can be added to frontend web applications. The project allows for **real user monitoring** (**RUM**) by collecting telemetry from a browser.

By adding RUM into an environment where backend applications and infrastructure are instrumented, observers gain the ability to traverse data from the full application stack.

k6

k6 is a load testing tool that provides both a packaged tool to run in your own infrastructure and a cloud **Software as a Service** (**SaaS**) offering. Load testing, especially as part of a CI/CD pipeline, really enables teams to see how their application will perform under load, and evaluate optimizations and refactoring. Paired with other detailed analysis tools such as Pyroscope, the level of visibility and accessibility to non-technical members of the team can be astounding.

Pyroscope

Pyroscope is a tool that enable teams to engage in the continuous profiling of system resource use by

applications (CPU, memory, etc.). Pyroscope advertises that with a minimal overhead of ~2-5% of performance, they can collect samples as frequently as every 10 seconds. **Phlare** is a Grafana Labs project started in 2022, and the two projects have now merged.

2. Common log formats

Logs support performance and capacity monitoring in infrastructure, bug detection in software, root cause analysis, user behavior tracking, and more. Following certain guidelines will help your future self when you need to analyze logs.

Log formats usually identify if they are structured or unstructured, the data types used in them, and if any encoding or delimitation is being used.

2.1. Structured, semi-structured, and unstructured logging

It does not matter what your logs look like and they can come in structured, semi-structured, or unstructured formats.

when designing and building observability solutions, it's important to understand the log formats you are working with. This ensures that you can ingest, parse, and store the data in a way that it can be used effectively.

If you familiarized yourself with the *personas*, you have an awareness of who they will be used by and for what purpose.

2.1.1. Structured logging

Structured logs have a predetermined message format that allows them to be treated as datasets rather than text. The idea of structured logging is to present data with a defined pattern that can be easily understood by humans and efficiently processed by machines. The log entries are often delimited with characters such as a comma, space, or hyphen. Data fields may also be joined using an equals sign or colon for key-value pairs, such as `name=Diego` or `city=Berlin`.

A structured log format

```
{
"timestamp": "2023-04-25T12:15:03.006Z",
"message": "User Diego.Developer has logged in",
"log": {
"level": "info",
"file": "auth.py",
"line": 77
},
"user": {
"name": "diego.developer",
"id": 123
},
"event": {
"success": true
}
}
```

An additional benefit of structured logging is that you can validate the conformation of the data to a schema

with tools such as JSON schema. This opens up the possibility of making version control changes to the schema, which is where logs and event bus technology overlap.

2.1.2. Semi-structured logging

Semi-structured logs aim to bridge the gap between unstructured and structured and, as a result, can be quite complicated. They are designed to be easy for humans to read but also have a schema that makes it possible for machines to process them too. They have complex field and event separators and usually come with a defined pattern to aid with ingesting and parsing. Parsing is usually done using regular expressions or other code.

2.1.3. Unstructured logging

Unstructured logging typically refers to log entries that are presented in a textual format that can easily be read by humans but is difficult for machines to process. They are often color-coded with blank spaces to improve presentation and readability.

Parsing and splitting the data correctly creates a disassociation between events and their identifying metadata. An unstructured log will require some custom parsing, requiring intimate knowledge of the data and often creating additional work for the engineer when ingesting data. This also creates technical liability; the dependency on the log remaining the same restricts developers from changing logs or runs the risk of parsing and reporting on unstructured logs prone to breaking.

To aid the ability of machines to process unstructured logs, encapsulation prevents entries such as stack traces from splitting at an inappropriate location.

The following is an example of a multiline log, with a naive encapsulation that looks for line breaks; this will appear in logging systems as four distinct events:

```
2023-04-25 12:15:03,006 INFO [SVR042] UserMembershipsIterable Found 4
children for 4 groups in 3 ms
Begin Transaction update record.
Process started.
Process completed.
```

With encapsulation based on the timestamp at the start of the event, this will be stored correctly for searching.

2.2. Sample log formats

Many log formats have been used in computer systems. All of these formats have a common goal of presenting a standard structure or set of fields for recording important information about the activity of a computer system.

2.2.1. Common Event Format (CEF)

Developed by ArcSight to fulfill the **Security Information and Event Management** (SIEM) use case, the CEF is a structured text-based log format. Using UTF-8 encoding, the format contains a prefix, a CEF header, and a body containing additional enrichment data.

Log Section	Description
Prefix	It combines the event timestamp and source hostname

Log Section	Description
CEF header	It combines the following pieces of metadata: • Software version • Vendor name • Product name • Product version • Event name • Product event class identification code • Event severity
Body	It contains a list of key-value pairs

Example

```
CEF:0|Security Provider|Security Product|Version|123|User
Authenticated|3|src=10.51.113.149 suser=diego target=diego msg=User
authenticated from 1001:db7::5
```

2.2.2. NCSA Common Log Format (CLF)

As one of the oldest log formats used by web servers, the NCSA CLF has for a long time been the most common and well-known log formats. It has a fixed format text-based structure and therefore cannot be customized at all.

Here is the NCSA CLF field list:

- Remote host address
- Remote log name
- Username
- Timestamp
- Request and protocol version
- HTTP status code
- Bytes sento

Where data is missing from the log, a hyphen acts as a placeholder. Unsupported characters are replaced with the + symbol.

Here is an example NCSA CLF log:

```
127.0.0.1 user-identifier diego [25/Apr/2023:12:15:03 -0000] "GET /
apache_pb.gif HTTP/1.1" 200 2326
```

2.2.3. W3C Extended Log File Format

The Microsoft Internet Information Server log format known as W3C is a structured yet configurable format. Full control over the included fields ensures log files contain the most relevant data. Identification of the information or direction of flow is denoted using a string prefix: server (S), client ©, server to client (SC), and client to server (CS).

Here is the W3C Extended Log File Format field list:

- Timestamp
- Client IP
- Server IP
- URI-stem
- HTTP status code
- Bytes sent
- Bytes received
- Time taken
- Version

Here is an example W3C log:

```
#Software: Internet Information Services 10.0
#Version: 1.0
#Date: 2023-04-25 12:15:03
#Fields: time c-ip cs-method cs-uri-stem sc-status cs-version
12:15:03 10.51.113.149 GET /home.htm 200 HTTP/1.0
```

2.2.4. Windows Event Log

The Microsoft Windows operating system comes with a built-in complex structured logging system that captures data related to specific events on the operating system. There are four common Windows event log categories - system, application, security, and setup - and an additional special category for forwarded events.

Each event log is also one of five different types: information, warning, error, success audit, and failure audit. Windows Event Log is one of the most verbose log formats in use. It usually includes details such as timestamp, event ID, username, hostname, message, and category, making it invaluable in diagnosing problems. Windows event IDs are documented and searchable, so you can easily get detailed information regarding the log event; they are grouped into categories, narrowing down the area where the event occurred, which makes debugging very accurate.

Here is a trimmed example of Microsoft Windows Event Log:

```
An account was successfully logged on.
Subject:
Security ID: SYSTEM
Account Name: DESKTOP-TMC369$
Account Domain: WORKGROUP
Logon ID: 0xE37
Logon Information:
New Logon:
Security ID: AD\DiegoDeveloper
Account Name: diego.developer@themelt.cafe
Account Domain: AD
Logon ID: 0xEC4093F
Network Information:
Workstation Name: DESKTOP-TMC369
```

2.2.5. JavaScript Object Notation (JSON)

As one of the newer yet most commonly used log formats today, JSON is a structured format constructed from multiple key-value pairs. Using JSON, data can be nested into different layers while keeping the format easy to read. Additionally, different data types can be represented, such as string, number, Boolean, null, object, and array.

Here is an example JSON log file:

```
{
"timestamp": "2023-04-25T12:15:03.006Z",
"message": "User Diego.Developer has logged in",
"log": {
"level": "info",
"file": "auth.py",
"line": 77
},
"user": {
"name": "diego.developer",
"id": 123
},
"event": {
"success": true
}
}
```

2.2.6. Syslog

The go-to log format for many years and still widely used, Syslog is a defined standard for creating and transmitting logs. The **Syslog transport protocol** specifies how log transmission takes place, as well as the data format. The default network ports for the protocol are `514` and `6514`, with the latter being used for encryption.

The Syslog message format combines a standardized header and message holding the body of the log.

Here is an example Syslog log:

```
Apr 25 12:15:03 server1 sshd[41458] : Failed password for  diego from
10.51.113.149 port 22 ssh2
```

2.2.7. Logfmt

Logfmt is a widely used log format that fits as human readable and structured so that computers and people can both read it. A Logfmt-formatted log line consists of any number of key-value pairs that can be easily parsed. As there are no standards, it is easy to extend and perfect for developers to simply add more key-value pairs to the output.

Here is an example Logfmt log:

```
level=info method=GET path=/ host=myserver.me fwd="10.51.113.149"
service=4ms status=200
```

3. Exploring metric types and best practices

Metrics, along with logs, are an essential tool for software developers and operators, providing them with indicators regarding the state of applications and systems. Resource usage data is great for monitoring a metric that captures numerical data over time. There are many different types of resources but some good examples would be CPU or RAM usage, the number of messages in a queue, and the number of received HTTP requests. Metrics are frequently generated and easily enriched with labels, attributes, or dimensions, making them efficient to search and ideal in determining if something is wrong, or different from usual.

A metric commonly has the following fields:

- **Name**: This uniquely identifies the metric
- **Data point value(s)**: The data that’s stored varies by metric type
- **Dimensions**: Additional enrichment labels or attributes that support analysis

3.1. WIP