Shasta Log Aggregation, Monitoring and Alerting in HPC Environments with Grafana Loki and ServiceNow

1st Elizabeth Bautista
NERSC
Lawrence Berkeley National Lab
Berkeley, USA
ejbautista@lbl.gov

2nd Nitin Sukhija
Department of Computer Science
Slippery Rock University of Pennsylvania
Slippery Rock, USA
nitin.sukhija@sru.edu

3rd Siqi Deng
NERSC
Lawrence Berkeley National Lab
Berkeley, USA
siqideng@lbl.gov

2022 IEEE International Conference on Cluster Computing (CLUSTER) | 978-1-6654-9856-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/CLUSTER51413.2022.00079

***Abstract*—The ongoing deployments of post-petascale computing systems has led to the proliferation in the hybrid computing models and orchestration of many complex services leading to the growth in operational challenges. It becomes increasingly important to deploy new integrated and comprehensive event management and monitoring solutions to collect computing infrastructures health logs and metrics data, to correlate and analyze the gathered data for reducing response time and downtime in face of computational center critical issues caused due to the physical and the digital threats. To address the above mentioned challenges, in this paper we present an automated log aggregation, monitoring and alerting framework that leverages the Operations Monitoring and Networking Infrastructure (OMNI) when used with Hewlett Packard Enterprise (HPE) Shasta, Grafana Loki and ServiceNow platforms for enabling a comprehensive proactive event response management and monitoring. Moreover, herein we also present two case studies involving automated remediation workflows employing the proposed framework at the National Energy Research Scientific Computing Center (NERSC) at Lawrence Berkeley National Laboratory (LBNL) using the Perlmutter computational system for real-time collection, aggregation, correlation, analysis, management and visualization of the system health metrics and logs in a single pane of glass for enhancing proactive monitoring and operational efficiency.**

***Index Terms*—Event Response, Shasta, Data Monitoring, ServiceNow, Prometheus, Grafana Loki, Visualization**

I. Introduction

The rapid move to the realm of exascale computing delivering computing performance of 10^{18} floating point operations per second has led to the proliferation of opportunities involving integration of numerous convergent technologies. This eruption of disruptive technologies and the convergence of HPC, artificial intelligence, analytics and modeling and simulation in the Exascale Era necessitates research and development of transformative and comprehensive technologies to support and scale with the complexity of escalating data, HPC and AI workloads [1] [2]. Currently all major supercomputer vendors are working with many organizations to implement and deliver exascale computing systems and technologies. One of the global supercomputer leaders, Hewlet Packard Enterprise (HPE) after its acquisition of Cray Inc announced in 2021 their plans to deliver the United States' first exascale system and technologies to the U.S. Department of Energy's (DOE) Argonne National Lab as a subcontractor to Intel [3] [4]. This next era of supercomputing systems employs HPEs innovative Shasta operating system, an architecture announced in 2018 [5], that enables flexible design, heterogeneous computing and new data-centric workloads to cope with the extreme scale and evolving demands of current and future processing capabilities.

The Shasta system architecture is currently employed by the pre-exascale computing system, Perlmutter at the NERSC [6]. Shasta software underpins cloudlike flexibility and full data center interoperability leading to extreme scale efficiency and extensibility required for more realized performance and accelerated discovery in diverse application domains.

In the exascale era, the advent in complexity of applications and services executing on the ever-larger-scale data centers catalyzing significant new research entails significant alternatives and support to overcome challenges pertaining to power consumption, data movement, fault tolerance, and extreme parallelism. Thus, minimizing costs of deploying, operating, and managing such heterogeneous computing infrastructures is of paramount importance. In addition to the above challenges, it becomes increasingly important for data centers to support high performance, management, flexibility, and reliability for the new applications utilizing such complex computing systems. Complicating the situation is the inexorable amount of data produced from all facility components, creating a challenge for storing, managing, and processing more generated logs and metrics to gather plausible insights required for future facility planning, decision making and user support [7]. HPE also has a policy of keeping event information for no more than two months, therefore, there is no historical data.

To address these challenges, HPC data centers are develop-

Authorized licensed use limited to: Southeast University. Downloaded on May 05,2023 at 03:12:04 UTC from IEEE Xplore. Restrictions apply.

ing and implementing next-generation data center monitoring and event management infrastructures that provide significantly improved insights and accurate operational diagnostics for enhanced decisions, availability, reliability, and user experiences at dramatically lower costs [8], because historical data can be examined and is readily available. So even if HPE doesn't keep the data, we stream the same data to another system, the Operations Monitoring and Notification System (OMNI) in order for our facility to consistently have access to at least two years of data immediately or be able to restore prior data that is more than two years old.

The new monitoring solutions that are currently deployed and proposed for the post peta-scale era must consider scalability, low-overhead and management efficiency as important design characteristics in addition to mandating the use of automation and orchestration systems for anticipating real-time resolutions while operating under dynamically changing production environments.

The sheer complexity and volume of heterogeneous logs and metrics gathered from these complex systems and sources that encompass a facility infrastructure means achieving operational efficiency becomes extremely challenging. Computational center staff must be able to analyze all the data available then provide a plan of action as quickly as possible to minimize downtime. Operational competence in such rapidly evolving high performance computing environments mandates the integration of log aggregation, event management and alerting components to the current scale monitoring solutions. This integration enables the reduction in noise caused by multiple alerts from the same events, the seamless analysis of logs and metrics, as well as the correlation of all events to accelerate actionable alerts and incidents with minimal response time. Recently, to support post-petascale generation computing operations, OMNI was implemented and deployed at NERSC to gather data for real-time data analysis and archiving diverse data and metrics related to the health of the heterogeneous computing components of the facility and the environment of the facility itself [9].

In this paper, we proposed a framework that facilitates a unified aggregated log and metric system for real-time root cause analysis of various critical applications and services issues, thus reducing Mean Time to Repair (MTTR) and enhancing the troubleshooting efficiency of the computational center by proactively alerting operational issues at NERSC. The framework is designed to meet the demanding and evolving monitoring and management requirements of the current and future scale of heterogeneous computing systems, such as, Perlmutter [10].

The rest of the paper is organized as follows. A review of related work and extreme-scale computational center management challenges is presented in Section 2. The design and organization of the comprehensive log aggregation, monitoring and alerting framework is described in Section 3. The use cases of the implemented event management and monitoring workflow are summarized in Section 4 followed by the conclusions and possible future directions in Section 5.

II. Background and Related Work

With every new generation of high performance computing, monitoring system health becomes a challenge. The exascale generation encompasses bigger HPC compute systems and the integration of the hybrid computing models and orchestration of many complex services, adding to the scale of the dynamicity and unpredictability of these high performance environments. Therefore, given the scale and complexity of these newer generation computing infrastructure, the existing monitoring approaches involving Nagios [11], Spiceworks [12], Icinga [13], or Zabbix [14] will grow obsolete. An emergent monitoring solution for the modern data centers requires a mix of tools and technologies that facilitates a holistic view of monitoring the computation center, hyperscale innovations like the capability to gather all data available, providing a proactive real-time alerting and a view of dashboards in one visible location. Some of the key design characteristics that can be considered should focus on the following:

- **Scalability:** implement scale-out approaches for supporting greater application volumes and reducing expenses.
- **Automation and Integration:** enable employment of application programming interfaces (API) for integrating facilities and IT operational platforms.
- **Real-Time Event Collection:** capable of seamlessly collecting and aggregating extreme-scale event logs and metrics stemming from various computational infrastructure components that provides current as well as historical data.
- **A single location to view all relevant dashboards:** implement a consolidated view of all tools that correlate, analyze and trigger automated remediation solutions from interactive dashboards.

Many tools have been researched and implemented for monitoring and event management in high performance computing environments. One of the leading monitoring tools employed by many organizations is Nagios. However, the static version of the tool has limited scalability into exascale systems and is extremely difficult from a management perspective when it comes to heterogeneous environments. Like Nagios, other popular monitoring tools that support a good web user interface and API but are limited by the scalability include, Spiceworks Network Monitoring Tool (SNMT) [12], Paessler PRTG Network Monitor [15], Zabbix [14] and Icinga [13]. Monitoring solutions, such as, New Relic [16], Honeycomb [17] and DataDog [18] which are service based systems and may fit in large scale environments are licensed based on the number of generated events and might not be a feasible cost effective solution for an exascale computing infrastructure that generates billions of events and metrics. There are platforms that integrates monitoring and event management and are proving scalable solutions, such as, the Service Management Project at CERN IT Computer Centre [19], Trumpet [20], Trinity Monitoring infrastructure [21], OpenLorenz [22], monitoring and automated recovery system at the Glasgow Tier$-$2 cluster [23] and others [24]. And of course, it would be HPE's

Authorized licensed use limited to: Southeast University. Downloaded on May 05,2023 at 03:12:04 UTC from IEEE Xplore. Restrictions apply.

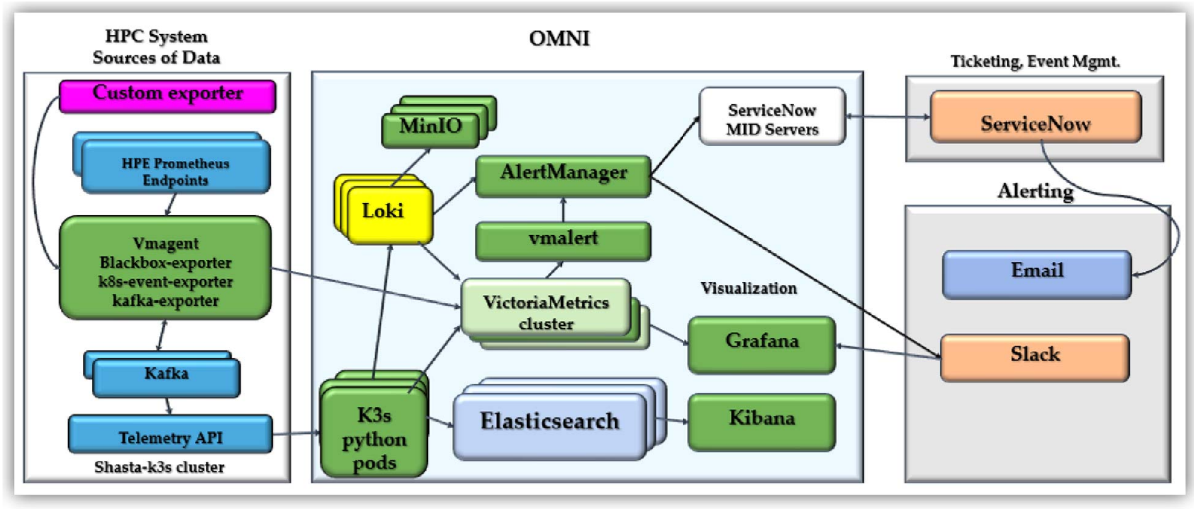


Fig. 1. Overview of Design and Architecture of the Log Aggregation, Monitoring and Alerting Framework.

preference for us to use the monitoring system that they have provided.

In our previous research work, we have developed a scalable monitoring and event management framework [25] [26] [27]. To best of our knowledge, in comparison to the above-mentioned related solutions, this present work is the first log aggregation, monitoring and alerting framework that enables the seamless integration of Shasta logs and data from a Prometheus data source, which is then collected in a single data warehouse, and where events are managed with alerts and creating incident management tickets for tracking in ServiceNow. This framework enables operational intelligence by collecting diverse events from both logs and metrics data, correlating events and employing machine learning methods for proactive incident response. And because we have collected all the data in OMNI, we have up to two years of operational data immediately available as historical data and further if we need to restore older data.

III. Log Aggregation, Monitoring and Alerting Framework

The exponential growth in the number of platforms and components comprising the post-petascale computing systems infrastructure translates in increasing scale of event generation and reporting leading to operational challenges related to alerting and remediation. In this section we provide an overview of the design of our scalable framework that integrates the Shasta, Grafana Loki, ServiceNow and Loki to provide the log aggregation, monitoring and alert management for Permutter and can be a template for future heterogeneous computing systems, network, and applications at the NERSC computational facility.

The Figure 1 illustrates the design and architecture of the comprehensive monitoring and event management framework proposed in this paper that utilizes automatic log and metric events aggregation, event correlation, root cause analysis, alerting prioritizing, prediction, and reporting via single pane view dashboards, thus enhancing insight analysis and resolution for resolving critical issues in real-time at NERSC facility.

The monitoring data comes from the following multiple sources as seen on Figure 1.

- Prometheus-style exporters and endpoints that are installed by HPE (e.g. node-exporter).
- Prometheus-style exporters from the Opensource community and installed by NERSC (e.g. blackbox-exporter and kafka-exporter).
- Custom Prometheus-style exporters that are written and installed by NERSC (e.g. aruba-exporter).
- Environmental and facility metrics, LDMS metrics, and network performance metrics that are stored in Kafka and available via the Telemetry API.
- Syslog, container logs, and redfish events that stored in Kafka and available via the Telemetry API.

This design aims to put all data from different sources into two categories: metrics and logs, but handle them differently at first. As a rule, we send metrics to Victoriametrics, the time series database and logs to Loki, the log aggregation tool. For example, Vmagent collects metrics from all the Prometheus-style exporters and sends data to Victoriametrics. K3s python pods (green box in center larger box) are python-written clients running in a Kubernetes environment. They read data in different Kafka topics via the Telemetry API and send them to either Victoriametrics or Loki.Both Victoriametrics and Loki support Grafana and Alertmanager natively. Therefore, even though metrics and logs are stored separately, they are unified in the stage of visualization and alerting.

Alerting is handled using vmalert for metrics, a component of VictoriaMetrics, that queries the database based on predefined rules. When the return value matches, vmalert sends an

Authorized licensed use limited to: Southeast University. Downloaded on May 05,2023 at 03:12:04 UTC from IEEE Xplore. Restrictions apply.

event to AlertManager. The same process happens using Ruler, a component of Loki, for logs.

A. Grafana Loki:

Inspired by Prometheus [28], the Grafana Loki platform consists of a fully featured logging stack that facilitates storing and querying logs from all applications and infrastructure. Loki enables seamless integration with Prometheus, Grafana [29] and K8s platform, thus aids in moving and building metrics, logs and generating alerts. Loki provides a log collector, PromTail [30] that aids to label, transform and filter logs. In contrast with other logging platforms, Loki does not index the text of the logs but allows indexing the metadata about the logs by creating labels. Log data is compressed and stored in chunks, thus a small index and compressed chunks significantly reduce the costs for storage and the log query times.

The log query is performed using LogQL [31], a powerful query language that uses labels to filter and return the content of the log lines. The queries can be executed and visualized using Grafana or a command line interface, LogCLI. Loki also includes Ruler, a component that enables assessment of a collection of configurable queries and execute an action based on the outcome, thus aids in setting alerting rules along with configuring routing of the resulting alerts from a Prometheus Alertmanager [32].

B. HPE Shasta System for Exascale Era:

Currently, the development and deployment of the exascale systems such as Aurora at Argonne National Laboratory, Frontier at Oak Ridge National Laboratory and El Capitan at Lawrence Livermore National Laboratory is in progress. These three extreme scale supercomputers will employ the HPE Shasta architecture, its Slingshot interconnect [33] and a new software platform in the 2023 timeframe [3] [4]. However, NERSC has already deployed Shasta in Perlmutter [6].

Shasta is a transformative platform designed with a vision of convergence of the HPC-cloud and HPC-enterprise and underpins the novel and extensible software platform to meet the increasing performance and scalability demands of the exascale era. Shasta unlocks the full potential of exascale computing by facilitating management and reliability of end to end hybrid HPC and AI workloads with the flexibility of cloud and full data center interoperability

The Shasta platform enables resiliency and serviceability by utilizing diagnostic capabilities using various system health metrics. Shasta facilitates flexibility to support multiple generations of a wide array of processors, GPU's, node configuration, system interconnects, storage and software, enabling easy upgradability over time. Moreover, the platform enables extensibility of the traditional HPC batch workflow with Kubernetes (K8s) [34] container orchestration for convergent HPC and AI workflows.

Shasta also provides diagnostic components and a telemetry API for the system as well as user application level monitoring, where the log generation sources are configurable via REST API to identify, repair, and recover from failures, thus aid in minimizing downtime of the system.This promises a lot, however, in the last year, NERSC has been in deep collaboration with HPE to complete the development of various components of Shasta in order to provide the diagnostic capabilities and monitoring data as previously mentioned.

C. Operations Monitoring and Notification Infrastructure (OMNI):

OMNI is a data warehouse to collect, manage and analyze data related to monitoring of extreme scale computing systems [9]. This infrastructure facilitates a single location for storing the heterogeneous datasets and is backed by a scalable and parallel time-series database, Elasticsearch [35] and VictoriaMetrics. Examples of operational data include time series data from the environment (e.g., temperature, power, humidity levels, and particle levels), monitoring data (e.g., network speeds, latency, packet loss, utilization or those that monitor the filesystem for disk write speeds, I/O, CRC errors), and event data (e.g., system logs, console logs, hardware failure events, power events essentially anything that has a start and end time).

OMNI is able to ingest at a rate of up to 400,000 messages per second from heterogeneous and distributed sources within the data center. Once ingested, the data is indexed for near real-time retrieval and querying, up to two years of operational data is immediately available and more can be restored. Data may be directly queried using the native RESTful API or using visualization and data discovery tools, such as Kibana [36] or Grafana. OMNI is highly available with minimal maintenance operations disrupting the data collection process. As systems become larger, more heterogeneous, and more complex, data can become unmanageable. Phase 1 of Perlmutter is projected to produce over 400 gigabytes of data per day. As more data is released by the different monitoring components, this could potentially become 10x per day. This is in addition to the existing data already being collected from the entire facility. The advantage of having OMNI is the central location for keeping heterogeneous data that can be correlated but also OMNI's ability to archive data, analyze data using various tools and have immediate operational historical data.

D. ServiceNow:

ServiceNow, a Platform-as-a-service provider (PaaS) facilitates the management of the IT infrastructure and services incidents, problem and change events [37]. ServiceNow focuses on fostering a unified cloud platform to automate and transform processes pertaining to the IT services, asset and event management, security and business operations and more. ServiceNow employs a configuration management database (CMDB), that maintains accurate and up-to-date records of the IT assets of an organization for achieving asset, compliance and configuration management efficiency. The service impact analysis is enabled by the CMBD database by storing current and accurate information pertaining to all the technical services in a Configuration item (CI). Moreover, the service

Authorized licensed use limited to: Southeast University. Downloaded on May 05,2023 at 03:12:04 UTC from IEEE Xplore. Restrictions apply.

mapping is made service-aware by tightly coupling it with CMDB database, where service maps employ discovery and infrastructure information in CMDB for creating an accurate and complete tag based map of all applications, virtual systems, underlying network, databases, servers and other IT components that supports the service. Furthermore, the automation of the service mapping facilitates not only a user interface illustrating an accurate service-level relationship but also adaptation of the service maps in real-time, enabling faster access to the insights and history of service topology.

The event management platform empowered by ServiceNow aids in providing an efficient integration with various monitoring tools, such as Prometheus for receiving events data for analysis and response. ServiceNow performs root cause analysis, defines alert rules, thresholds, and remediation actions by utilizing the gathered operational metrics and employing machine learning to reduce the Mean Time to Resolution (MTTR). Moreover, visualization of services facilitated by the ServiceNow dashboards helps to prioritize service incidents based on the impact and criticality. NERSC does not use all of ServiceNow's features and currently only use their incident management module, and event management module. Using event management, CMDB and CI still needed to be configured using Perlmutter assets only. ServiceNow is the incident management platform adopted by NERSC as well as LBNL, and other Department of Energy (DOE) national laboratories even prior to the installation of the Perlmutter system.

IV. Log aggregation, Monitoring and Alerting Workflow

Given that the scale and heterogeneity of the number of computing components and its supporting platforms is expanding rapidly at NERSC, the increase in alert remediation has become a critical element of the Perlmutter operational monitoring and event management infrastructure. In this section we present two case studies that employ automated remediation workflows that integrate the Shasta, Prometheus, OMNI, Grafana and ServiceNow platforms to aid in operational intelligence and troubleshooting efficiency. The automated log aggregation, monitoring and alerting management workflow implements are described in the following stages:

- Kafka or Prometheus endpoints [38], Telemetry API servers, rsyslogd aggregators, and other services that used for managing and monitoring Perlmutter are running as containers in the Perlmutter Kubernetes environment. The health and performance metrics of these services are collected by VMagent which is also a container in the same cluster.
- VMagent directly pushes metrics to the VictoriaMetrics (Open source time-series database) [39] cluster in OMNI.
- Sensors in each cabinet, chassis, node, switch, cooling unit collect data like temperature, humidity, power, fan speed.
- Redfish (RESTful interface for the infrastructure management) [40] endpoint on each controller push metrics and events(e.g. power down) to an HMS(hardware management service) collector.
- The HMS collector pushes data to Kafka, where Kafka stores data in different topics by categories and serves them to possible consumers.
- The telemetry API server acts as a middleman between Kafka and data consumers and is responsible for authentication and balancing income requests.
- The telemetry API client then sends a request to the API server and creates a subscription to a Kafka topic. Kafka pushes data to the client via the API. The client converts the data format before sending it to VictoriaMetrics or Loki.
- For metrics, vmalert, a component of the VictoriaMetrics cluster, queries the database continuously with predefined alerting rules created by NERSC. If the return value is true, vmalert sends an event to AlertManager. For logs, Ruler, a component of the Loki cluster evaluates logs by querying the database continuously with predefined alerting rules. If the return value is true, Ruler sends an event to Alertmanager.
- Alertmanager receives events, groups them by priority, category, source, etc. and sends alert messages to Slack or ServiceNow.
- Alerts are transformed into ServiceNow (SN) "Events", which are correlated and grouped into SN "Alerts", which then trigger automated response actions (incidents, notifications, etc.)

Leveraging all the open source products listed in Figure 1 for the development and deployment stages of Perlmutter's comprehensive monitoring and alerting infrastructure, we are planning to employ the above described workflow to automate many use cases for enhancing operational efficiency as described in the following sections.

The use cases described in the following sections were investigated by employing the Loki cluster, where the cluster has 8 server nodes (that work as Kubernetes worker nodes) and 4 virtual machines (that work as Kubernetes master nodes and gateway node) using Kubernetes ENV: v1.21.10+k3s1 and Loki version: 2.4.2. Moreover, each server node comprises of 64 Processors (AMD EPYC 7302) with 128GB memory and a 14TB NVME disk drive.

A. Leak Detection and and Alerting:

Perlmutter compute nodes feature direct liquid cooling. The process of diirect liquid cooling greatly enhances thermal transfer efficiency, but has a risk of damaging parts resulting from leaks. Multiple liquid leak detection sensors are installed in each compute node cabinet. If a leak is detected, those sensors will send out an event alert with location information to the Shasta Monitoring Framework via the redfish protocol, the protocol provided by HPE.These events can be retrieved via the Shasta Telemetry API along with other numerical monitoring data. While HPE would like us to use their tools, it has a very steep learning curve and we are not able to manipulate the visual to provide additional information. This

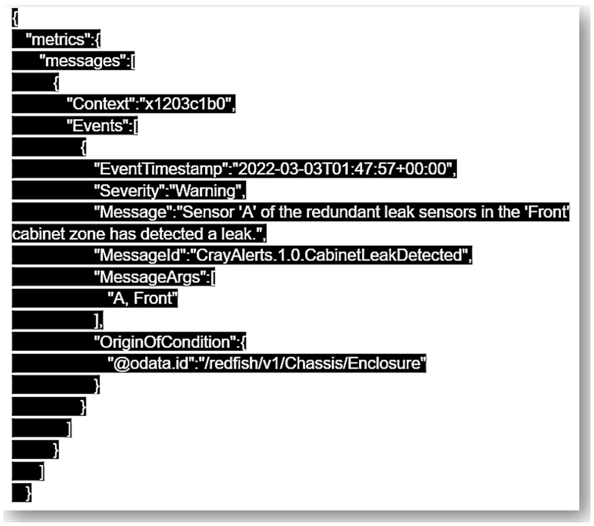
Authorized licensed use limited to: Southeast University. Downloaded on May 05,2023 at 03:12:04 UTC from IEEE Xplore. Restrictions apply.

is why we capture the raw data instead and provide our own visual or tools. Their Prometheus monitoring model is designed for monitoring numerical metrics, not events or logs. Therefore, we wouldn't even be able to get an alert unless we specifically look at their tools. This is very manual and can be missed with so many other areas we are monitoring and watching.

Loki is like the Prometheus tool mentioned above but for logs. It constantly evaluates Shasta events and logs, including Redfish events in this use case, and turns the result into Prometheus-style metrics. It is compatible with AlertManager which provides our alerting. The Telemetry API publishes Redfish events in a nested JSON format. Figure 2 illustrate a sample raw data pulled from the Telemetry API using Python3. The context information indicates where the event happened, where x1203c1b0 is the location of a Perlmutter chassis. This nested JSON format data with timestamp and strings doesn't fit in the Prometheus monitoring model. However, Loki is designed for storing strings and converting them into Prometheus-style metrics, thus implementing Loki is an innovative idea when applied to these type of metrics.

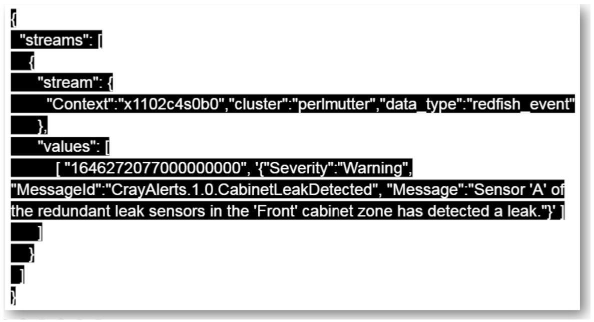
Every log in Loki has three parts. They are timestamp, labels, and log content (called as string interchangeably) and are handled differently. The timestamp in Loki is an unix epoch in nanoseconds and the labels are the key-value pairs. Similar to Prometheus that uses a combination of labels to identify a particular dimensional instantiation of a certain metric, Loki utilizes labels in the same way to filter and aggregate logs.

Loki indexes the timestamp and labels only, and the log contents are compressed and stored in chunks, where a chunk is a concept that Loki uses to describe how it stores logs in small buckets. Every log has one or more labels. If logs share the same combination of unique labels, they are called a log stream. Each log stream fills a separate chunk. So logs with the same combination of labels are stored in the same chunk, and sorted in timestamp order. When a chunk is full, Loki creates a new chunk. Chunks are first stored in memory, and then moved to disk. Before sending the Redfish event shown in Figure 2 to Loki, there is a need to clean up the data, and to extract all valuable information. Obviously, there is one timestamp in ISO 8601 format only. It should be converted into an unix epoch in nanoseconds. The OriginOfCondition field contains a link to the Redfish endpoint which is not useful and the MessageArgs field has duplicate information in the Message field. So both fields should be removed. Meanwhile, we would like to enrich the log with two extra key-value pairs: "cluster":"perlmutter" and "data_type":"redfish_event" because there is more than one cluster at NERSC, and we store multiple types of string data (e.g. Syslog, container logs) in Loki. Therefore, we will send a log with the data illustrated in Figure 3 to Loki: The timestamp is obvious. However, the problem is which parts of the rest of the data should be sent as labels, or log content. Since labels are indexed, more labels creates more index entries and each log stream fills a chunk. The overuse of labels will create a huge amount of small chunks in memory



```
{
  "metrics":{
    "messages":[
      {
        "Context":"x1203c1b0",
        "Events":[
          {
            "EventTimestamp":"2022-03-03T01:47:57+00:00",
            "Severity":"Warning",
            "Message":"Sensor 'A' of the redundant leak sensors in the 'Front'
cabinet zone has detected a leak.",
            "MessageId":"CrayAlerts.1.0.CabinetLeakDetected",
            "MessageArgs":[
              "A, Front"
            ],
            "OriginOfCondition":{
              "@odata.id":"/redfish/v1/Chassis/Enclosure"
            }
          }
        ]
      }
    ]
  }
}
```

Fig. 2. A sample raw data pulled from the Telemetry API.



```
{
  "streams": [
    {
      "stream": {
        "Context":"x1102c4s0b0","cluster":"perlmutter","data_type":"redfish_event"
      },
      "values": [
          [ "1646272077000000000", '{"Severity":"Warning",
"MessageId":"CrayAlerts.1.0.CabinetLeakDetected", "Message":"Sensor 'A' of
the redundant leak sensors in the 'Front' cabinet zone has detected a leak."}' ]
      ]
    }
  ]
}
```

Fig. 3. A sample log data input to Loki platform.

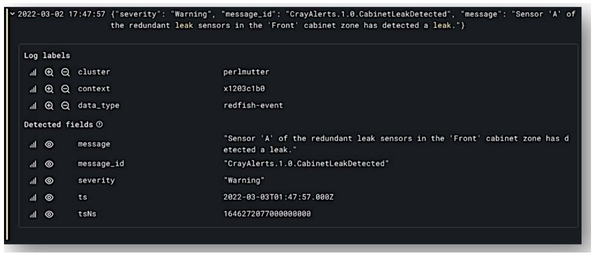
and on disk. Moreover, Loki prefers handling bigger but fewer chunks. Thus, to achieve better performance, there is need to limit the number of labels in logs, and use key-value pairs with less variation as labels if possible.

The Context field is critical for filtering events from a specific location, so it should be sent as a label. Both the cluster and data_type fields have little variation and are useful in filtering logs from other clusters and data types. Thus, are sent as labels as well. The rest fields are Severity, MessageId, and Message, which describe what the event was and should be sent as log content. Moreover, Grafana can further extract information if a log string is structured in JSON. Thus, all three Severity, MessageId, and Message fields can be wrapped in a string as:

'{"Severity":"Warning","MessageId":"CrayAlerts.1.0. CabinetLeakDetected","Message":"Sensor 'A' of the redundant leak sensors in the 'Front' cabinet zone has detected a leak."}'

Eventually, clients that are written in Python3 are utilized

Authorized licensed use limited to: Southeast University. Downloaded on May 05,2023 at 03:12:04 UTC from IEEE Xplore. Restrictions apply.



```
2022-03-02 17:47:57 {"severity": "Warning", "message_id": "CrayAlerts.1.0.CabinetLeakDetected", "message": "Sensor 'A' of
                     the redundant leak sensors in the 'Front' cabinet zone has detected a leak."}

Log labels
    cluster                         perlmutter
    context                         x1203c1b0
    data_type                       redfish-event
Detected fields
    message                         "Sensor 'A' of the redundant leak sensors in the 'Front' cabinet zone has d
                                    etected a leak."
    message_id                      "CrayAlerts.1.0.CabinetLeakDetected"
    severity                        "Warning"
    ts                              2022-03-02T01:47:57.000Z
    tsNs                            1646272077000000000
```

Fig. 4. Redfish Event visualization using Grafana.

to subscribe events from the Telemetry API, and convert each nested JSON event into a combination of timestamp, labels and string data, and send it to Loki.

Figure 4 illustrate the Redfish event in Grafana (Loki has no UI, thus data is visualized in Grafana). LogQL is used to convert logs from string to numerical metrics that fit in the Prometheus monitoring model. LogQL is Grafana Loki's PromQL inspired query language, where queries act as if they are a distributed grep to aggregate log sources. LogQL uses labels and operators for filtering. The LeakDetected event shown in Figure 4 can be converted to a metric in Grafana using a LogQL query as shown below:

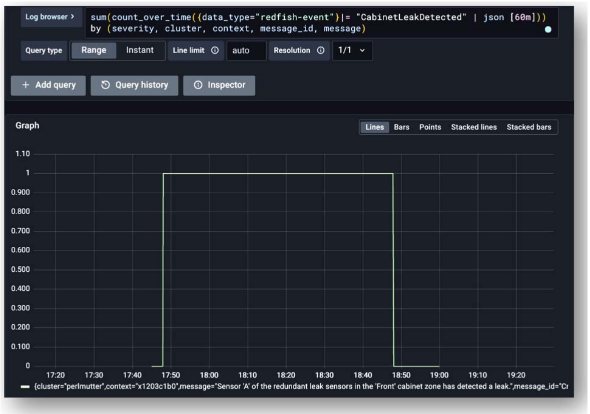
sum(count_over_time(data_type="redfish-event"|= "CabinetLeakDetected" | json [60m])) by (severity, cluster, context, message_id, message)

The result of the query increases from zero to one at round 17:47:57 PST in the graph as shown in Figure 5. We first find all Redfish events that contain the string of CabinetLeakDetected.

data_type= "redfishevent"|="CabinetLeakDetected"

Then use a JSON parser to extract all key-value pairs in strings by JSON and a LogQL function called count_over_time counts the number of such events in the last 60 minutes. The leak event happened at 17:47:47 PST, so the query can find one event between 17:47:47 PST and 18:47:47 PST, and Loki returns 1 in the 60 minute time window. The sum(...) by (...) function groups the return value by labels. Thus, if multiple leak events from different location are found, Loki returns multiple vectors with different labels instead of one vector without labels.

The LogQL turns strings into metrics which then can be used for generating alerts. Loki includes a component called the Ruler which is responsible for continually evaluating a set of configurable queries and performing an action based on the result. In the configuration of Ruler, we can define alerting rule for remediation plans. Loki Ruler alerting rules share the same format as Prometheus alerting rules. The alerting rule also utilizes the LogQL query. If the return value is greater than zero and it lasts more than one minutes, an alert will be generated.

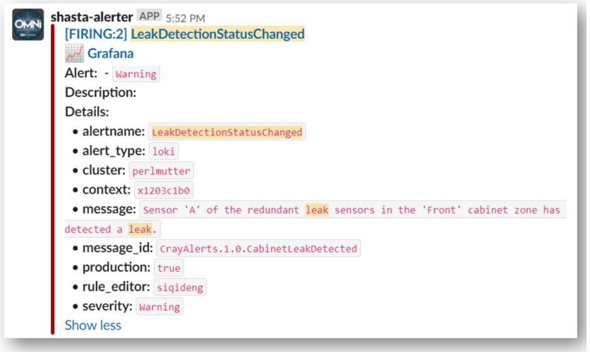


```
sum(count_over_time({data_type="redfish-event"}|= "CabinetLeakDetected" | json [60m]))
by (severity, cluster, context, message_id, message)
```

Query type: Range | Line limit: auto | Resolution: 1/1

{cluster="perlmutter",context="x1203c1b0",message="Sensor 'A' of the redundant leak sensors in the 'Front' cabinet zone has detected a leak.",message_id="Cr

Fig. 5. LeakDetected event converted to a metric in Grafana using a LogQL query.



shasta-alerter APP 5:52 PM
[FIRING:2] LeakDetectionStatusChanged
Grafana
Alert: - Warning
Description:
Details:
- alertname: LeakDetectionStatusChanged
- alert_type: loki
- cluster: perlmutter
- context: x1203c1b0
- message: Sensor 'A' of the redundant leak sensors in the 'Front' cabinet zone has detected a leak.
- message_id: CrayAlerts.1.0.CabinetLeakDetected
- production: true
- rule_editor: siqideng
- severity: Warning

Show less

Fig. 6. Slack alert generated from the Redfish leak event.

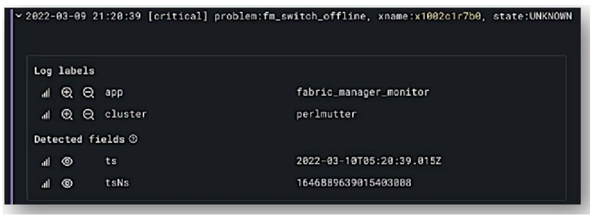
The Ruler is compatible with the Alertmanager and both can be connected by adding an Alertmanager webhook in the configuration of the Ruler. In Alertmanager, a Slack webhook is added in order for Alertmanager to send alerts to Slack. Further, the Slack alert is enriched with different types of fonts and bullet points. Figure 6 illustrates the Slack alert generated from the Redfish leak event example.

Without this implementation, there would not be an automatic way of being alerted to leaks in the Perlmutter system. A person would be spending their time physically looking through the HPE tools and this would be their job for the whole day. Because these tools looks like lines without any color differentiation, a person would have to read it line by line to actually "see" the event they are looking for. And, this is only one event interspersed with other events. By receiving the alerts from Loki automatically, we minimize downtime by being able to mitigate the leak problem quicker.

B. Switch Offline Detection and Alerting:

Each Rosetta switch connects eight compute nodes. If one switch goes offline, the connection of the group of eight compute nodes goes down. There is a Slingshot Fabric Manager

Authorized licensed use limited to: Southeast University. Downloaded on May 05,2023 at 03:12:04 UTC from IEEE Xplore. Restrictions apply.



```
2022-03-09 21:20:39 [critical] problem:fm_switch_offline, xname:x1002c1r7b0, state:UNKNOWN

Log labels
    app                 fabric_manager_monitor
    cluster             perlmutter
Detected fields
    ts                  2022-03-10T05:20:39.015Z
    tsNs                1646889639015403008
```

Fig. 7. Sample switch event illustration in Grafana.

in Shasta, provided by HPE, that manages all switches. It provides an API for querying the state of each switch. NERSC uses a python program to query the API periodically, and send out an event to Loki if any switch stage change is found. A sample event that indicates the state of the x1002c1r7b0 switch became unknown is shown below.

As with the leak detection, without the query going through the API and the event sent to Loki for processing, it would be a staff person's job to look through these events provided by HPE in order to check if a switch stage has changed or not. What is likely to occur, is the Shasta system will notify that the fabric has gone down. We like to be more proactive than that and be alerted before the possibility of the network fabric going down so we can mitigate the problem to minimize downtime.

[critical] problem:fm_switch_offline, xname:x1002c 1r7b0, state:UNKNOWN

Figure 7 illustrates the sample switch event in Grafana. It has two labels: app and cluster which notify about the source of this event that are generated by the fabric manager monitor program and from the Perlmutter system.

There is little control over the format of the event message. However, all switch offline events can be easily queried by the utilizing the following Loki's query:

app="fabric_manager_monitor" |= "fm_switch_offline"

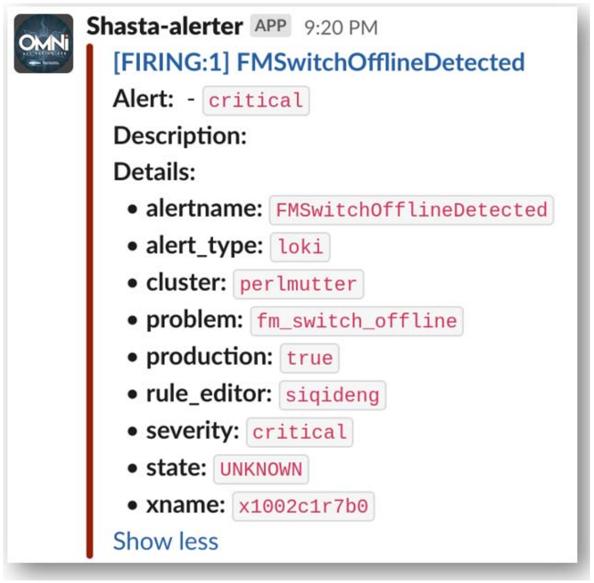
Furthermore, we extract more information from the message by leveraging a pattern function in Loki as shown below:

|pattern"[<severity>]problem:<problem>, xname:<xname>,state:<state>"

Just like the previous use case, logs are converted into metrics by countering the number of matched events, and grouped by different labels and attributes. The alerting rule shown in Figure 8 searches for the switch offline events in Loki database, and sends out an event to AlertManager. Following the notification from the Loki, the AlertManager sends out a Slack notification shown in Figure 9 to inform the on-call staff.

```
    - alert: FMSwitchOfflineDetected
      expr: sum(count_over_time({app="fabric_manager_monitor"} |=
"fm_switch_offline" | pattern "[<severity>] problem:<problem>, xname:<xname>,
state:<state>"[60m])) by (severity,problem,xname,cluster,state) > 0
      for: 3m
      labels:
        production: "true"
        alert_type: loki
```

Fig. 8. Alerting rule querying offline switch events in Loki's database.



Shasta-alerter APP 9:20 PM

[FIRING:1] FMSwitchOfflineDetected

Alert: - critical

Description:

Details:

- alertname: FMSwitchOfflineDetected
- alert_type: loki
- cluster: perlmutter
- problem: fm_switch_offline
- production: true
- rule_editor: siqideng
- severity: critical
- state: UNKNOWN
- xname: x1002c1r7b0

Show less

Fig. 9. Offline switch slack notification by AlertManager.

V. Conclusions and Future Work

The automated log aggregation, monitoring and alerting framework presented in this paper integrates the OMNI infrastructure with HPE's Shasta, Grafana Loki and ServiceNow platforms for log and metrics collections, correlation and visualizations enabling proactive monitoring and event response management in single pane of glass view. Moreover, alert remediation and real-time automated root cause analysis enabled via the seamless analysis of logs from various critical applications and services aids in reducing the number of incidents requiring troubleshooting from operational staff, thus decreasing staff interventions, and increasing efficiency.

The immediate future work will be to employ Loki for syslog monitoring and creating a mechanism for monitoring the health status and performance for the General Parallel File System (GPFS) which is one of Perlmutter's storage components. We hope to create more informative slack notifications by linking dashboards with Slack with more illustrated automated remediation workflows that will aid in speeding the time to resolve exponentially increasing number of computational center incidents, as well as for other issues that requires the data and the logs to be gathered and aggregated seamlessly.

Authorized licensed use limited to: Southeast University. Downloaded on May 05,2023 at 03:12:04 UTC from IEEE Xplore. Restrictions apply.

VI. Acknowledgments

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DEAC0205CH11231.

References

[1] G. Iuhasz and D. Petcu, "Monitoring of exascale data processing," in *2019 IEEE International Conference on Advanced Scientific Computing (ICASC)*. IEEE, 2019, pp. 1–5.

[2] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, vol. 15, p. 181, 2008.

[3] "First us exascale supercomputer to be a cray shasta system," *https://www.hpe.com/us/en/newsroom/press-release/2019/03/first-us-exascale-supercomputer-to-be-a-cray-shasta-system.html*.

[4] "Hpe cray exascale 2019," *https://www.hpe.com/us/en/compute/hpc/cray/doe-el-capitan-press-release.html*.

[5] C. Lundin and S. Fisher, "Significant advances in cray system architecture for diagnostics availability resiliency and health," in *Cray User Group Meeting*, 2019.

[6] "Berkeley lab first in line for cray "shasta" supercomputers," *https://www.nextplatform.com/2018/10/30/berkeley-lab-first-in-line-for-cray-shasta-supercomputers/*.

[7] K. Djemame and H. Carr, "Exascale computing deployment challenges," in *International Conference on the Economics of Grids, Clouds, Systems, and Services*. Springer, 2020, pp. 211–216.

[8] J. Shalf, S. Dosanjh, and J. Morrison, "Exascale computing technology challenges," in *International Conference on High Performance Computing for Computational Science*. Springer, 2010, pp. 1–25.

[9] E. Bautista, C. Whitney, and T. Davis, "Big data behind big data," in *Conquering Big Data with High Performance Computing*. Springer, 2016, pp. 163–189.

[10] Perlmutter, "https://www.nersc.gov/systems/perlmutter/."

[11] "Nagios," *https://www.nagios.org/*.

[12] "Spiceworks network monitoring management software," *https://www.spiceworks.com/*.

[13] "Icinga," *https://icinga.com/*.

[14] "Zabbix," *https://www.zabbix.com/*.

[15] "Paessler prtg network monitor," *https://www.paessler.com/prtg*.

[16] "New relic," *https://newrelic.com/*.

[17] "Honeycomb," *https://www.honeycomb.io/*.

[18] "Datadog," *https://www.datadoghq.com/*.

[19] R. A. Alonso, G. Arneodo, O. Barring, E. Bonfillou *et al.*, "Migration of the cern it data centre support system to servicenow," in *Journal of Physics: Conference Series*, vol. 513, 2014, p. 062032.

[20] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Trumpet: Timely and precise triggers in data centers," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 129–143.

[21] A. DeConinck, A. Bonnie, K. Kelly, S. Sanchez, C. Martin, M. Mason, J. M. Brandt, A. C. Gentile, B. A. Allan, A. M. Agelastos *et al.*, "Design and implementation of a scalable monitoring system for trinity." Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2016.

[22] "Openlorenz: Web-based hpc dashboard and more," *https://software.llnl.gov/repo//hpc/OpenLorenz*.

[23] E. Simili, G. Stewart, G. Roy, S. Skipsey, and D. Britton, "A hybrid system for monitoring and automated recovery at the glasgow tier-2 cluster," in *EPJ Web of Conferences*, vol. 251. EDP Sciences, 2021, p. 02047.

[24] D. A. Nikitenko, S. A. Zhumatiy, and P. A. Shvets, "Making large-scale systems observable-another inescapable step towards exascale," *Supercomputing Frontiers and Innovations*, vol. 3, no. 2, pp. 72–79, 2016.

[25] N. Sukhija, E. Bautista, O. James, D. Gens, S. Deng, Y. Lam, T. Quan, and B. Lalli, "Event management and monitoring framework for hpc environments using servicenow and prometheus," in *Proceedings of the 12th International Conference on Management of Digital EcoSystems*, 2020, pp. 149–156.

[26] N. Sukhija and E. Bautista, "Towards a framework for monitoring and analyzing high performance computing environments using kubernetes and prometheus," in *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 2019, pp. 257–262.

[27] E. Bautista, N. Sukhija, M. Romanus, T. Davis, and C. Whitney, "Omni at the edge," in *Cybersecurity and High-Performance Computing Environments*. Chapman and Hall/CRC, 2022, pp. 63–84.

[28] "Prometheus: Monitoring system and time series database," *https://prometheus.io/*.

[29] "Grafana," *https://grafana.com/*.

[30] "Promtail," *https://grafana.com/docs/loki/latest/clients/promtail/*.

[31] "Logql: Log query language," *https://grafana.com/docs/loki/latest/logql/*.

[32] "Alertmanager," *https://prometheus.io/docs/alerting/latest/alertmanager/*.

[33] D. De Sensi, S. Di Girolamo, K. H. McMahon, D. Roweth, and T. Hoefler, "An in-depth analysis of the slingshot interconnect," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–14.

[34] "Kubernetes," *https://kubernetes.io/*.

[35] "Elasticsearch: Distributed, restful engine," *https://www.elastic.co/products/elasticsearch*.

[36] "Kibana," *https://www.elastic.co/kibana/*.

[37] ServiceNow, "https://www.servicenow.com/."

[38] "Apache kafka," *https://kafka.apache.org/*.

[39] "Victoriametrics," *https://victoriametrics.com/*.

[40] "Redfish," *https://www.dmtf.org/standards/redfish*.

Authorized licensed use limited to: Southeast University. Downloaded on May 05,2023 at 03:12:04 UTC from IEEE Xplore. Restrictions apply.