ISBN: 978-3-95450-249-3 ISSN: 2673-5350 doi: 10.18429/JACoW-IBIC2024-WEP50

RESEARCH ON VISUALIZATION AND INDEXING OF DATA BASED ON THE ELK STACK

Yukun Li[1†], Jianshe Cao[1], Qiang Ye, Yaoyao Du
Institute of High Energy Physics, Chinese Academy of Sciences, Beijing 100049, China
[1]also at University of Chinese Academy of Sciences, Beijing 100049, China

Abstract

This paper proposes a comprehensive solution for real-time collection and analysis of Beam Position Monitor (BPM) telemetry data using Kafka and the ELK stack [1]. It involves transmitting PV variables from BPM electronic devices to the Kafka messaging queue, enabling powerful and scalable data streaming. By retrieving JSON formatted data from Kafka using the ELK stack, efficient data indexing and visualization in Kibana are achieved. The paper provides detailed explanations of the architectural design, implementation details, and the advantages of using Kafka as a central hub for BPM data dissemination. This integration not only enhances the performance and reliability of the data processing pipeline but also offers a powerful tool for physicists and engineers for real-time visualization and monitoring of BPM data.

DISTRIBUTED MESSAGING SYSTEM

Kafka, in conjunction with Zookeeper, forms a distributed messaging system that ensures high reliability and consistency within the Kafka cluster. Zookeeper serves as Kafka's configuration center, storing all crucial information about Topics, Partitions, and Brokers. When Brokers in the Kafka cluster join or leave, Zookeeper dynamically adjusts the system configuration to ensure stable operation of the cluster. Moreover, Zookeeper is responsible for the fault recovery process when a Broker fails, by re-electing new leaders to maintain the continuity of the messaging system.

Kafka relies on Zookeeper to handle synchronization issues within the cluster, where the stability and reliability of Zookeeper directly impact the quality and efficiency of Kafka services. Together, they support building an efficient, scalable, and reliable real-time data processing system.

Kafka Message Queue

Apache Kafka is a distributed streaming platform capable of efficiently handling vast data streams [2]. Its core is a publish-subscribe messaging system designed for scenarios requiring high throughput and low latency data transmission.

The Architecture of Kafka is illustrated in Figure 1. The operational mechanics of Kafka include multiple Producers, Servers (Brokers), Consumers, Consumer Groups, and a Zookeeper cluster. Producers publish messages to Kafka, where messages are stably stored on Brokers' log files in chronological order. Consumers read messages from the Brokers and support message re-reads. Kafka enables message broadcasting (where each message is read by multiple consumers) and load balancing (where each message is processed by only one consumer in a consumer group) through Consumer Groups.

The operational mechanics of Kafka include multiple Producers, Servers (Brokers), Consumers, Consumer Groups, and a Zookeeper cluster. Producers publish messages to Kafka, where messages are stably stored on Brokers' log files in chronological order. Consumers read messages from the Brokers and support message re-reads. Kafka enables message broadcasting (where each message is read by multiple consumers) and load balancing (where each message is processed by only one consumer in a consumer group) through Consumer Groups.

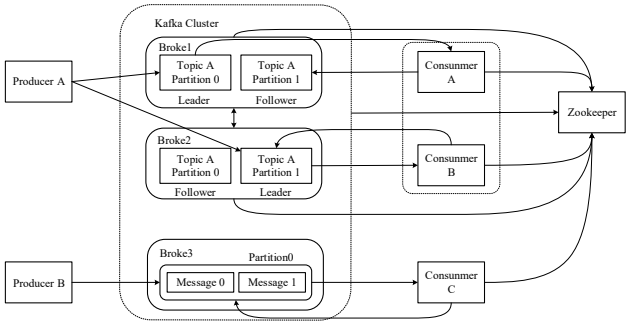

Figure 1: Kafka architecture diagram.

Zookeeper Cluster

Zookeeper is software that provides consistency services for distributed systems, maintaining configuration information, name registration, and distributed synchronization through a centralized service [3]. As a centralized coordination system for distributed applications, Zookeeper records all critical system status information, essential for the system's reliability and consistency.

In Kafka, Zookeeper manages cluster metadata and ensures synchronization among Brokers. Kafka utilizes Zookeeper for leader elections among Brokers, managing metadata of Topics, and monitoring cluster states, ensuring message availability and consistency in case of Broker failures. Each Kafka cluster includes a Zookeeper cluster to maintain consistent cluster status.

ELK STACK

The ELK technology stack, composed of Elasticsearch, Logstash, and Kibana, is a collection of three open-source tools mainly used for efficiently handling large data volumes through search, analysis, and visualization. The ELK

†Li Yukun, working on accelerator beam measurement and control.
E-mail: liyukun@ihep.ac.cn.

ISBN: 978-3-95450-249-3 ISSN: 2673-5350 doi: 10.18429/JACoW-IBIC2024-WEP50

stack offers a powerful and flexible solution from data collection to processing, and visualization, where each component plays a crucial role in the data handling process [4].

Logstash

Logstash is a powerful data processing pipeline capable of simultaneously collecting data from multiple sources, transforming it, and sending it to designated repositories [5]. It supports various input, filter, and output plugins, adapting to nearly any type of log format and data source. Any type of event stream can be processed and transformed through Input, Filter, and Output plugins in Logstash. Its flexibility and robust data transformation capabilities make it an ideal choice for data preprocessing and cleansing.

Logstash can couple with various external systems, collecting data from sources including log files, system monitoring tools, and other data streams. It can gather data from multiple sources at the same time. In this paper, Kafka is chosen as the data source for Logstash. As a high-performance message queue, Kafka can cache data in the Kafka cluster to prevent data loss. Deploying Kafka as a data source for Logstash instead of directly transmitting data through Logstash aims to relieve the pressure on Logstash as a data pipeline and avoid data loss.

Logstash supports various Filters, such as grok, json, mutate, etc. By using filters, Logstash can parse and transform data into structured and query-friendly formats. For instance, this paper uses the json plugin to convert array-type json format data into list types. Data processed by Logstash can also be outputted to multiple destinations through output plugins. This paper chooses to output data to the Elasticsearch cluster, although Logstash also supports other storage systems like Amazon S3, MongoDB, etc.

Elasticsearch

Elasticsearch is a highly scalable open-source full-text search and analytics engine. It operates in real-time, providing quick and efficient storage, search, and analysis of large-scale data [6]. Built on Apache Lucene, Elasticsearch supports various types of search capabilities, including full-text search, structured search, and geospatial search.

As a distributed system, Elasticsearch automatically distributes data and query loads across multiple nodes, handling massive amounts of data. It provides near real-time search capabilities, enabling quick retrieval of data. Elasticsearch also supports automatic sharding and replication, which facilitates handling hardware failures and supports horizontal scaling of clusters.

For deployment, this paper opts for a three-node Elasticsearch cluster. In an Elasticsearch cluster, multiple nodes work together, sharing data and processing tasks. The cluster elects a master node through an election process, which manages the cluster's metadata and controls cluster operations. If a network failure or other issue causes some nodes to lose connection with others, it could lead to multiple nodes or groups of nodes believing they are the master node, conducting operations independently. This split-brain scenario can severely affect data integrity and availability.

Kibana

Kibana is a data visualization front-end platform for Elasticsearch, used for searching, viewing, and interacting with data stored in Elasticsearch indices [7]. Through Kibana, users can create data queries and charts, visually presenting data in a straightforward manner.

Kibana's dashboard interface supports various chart types, including bar charts, line charts, pie charts, and scatter plots, allowing for diverse styles of data visualization. Kibana provides a web-based interface, enabling users to analyze and query data in Elasticsearch in real-time.

SYSTEM ARCHITECTURE AND IMPLEMENTATION

Data Transmission Process

The data transmission architecture is illustrated as Figure 2 [8]. Data acquisition programs retrieve PV variables already written to the database from Archiver and convert these process variables into JSON formatted data. This data is then written to a pv_list as a data cache. At this stage, the data in the cache is array-type, which is then converted to list-type before being written into the designated Kafka message topic. After confirming successful message transmission, the topic, partition, and offset of the data written to Kafka are printed. Once data is written to Kafka, it is sent to the Elasticsearch cluster through the Logstash pipeline, where it generates a data index, ultimately visualized on the Kibana interface.

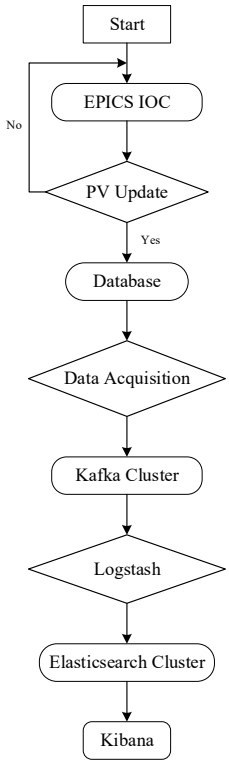

Figure 2: Data transmission flowchart.

ISBN: 978-3-95450-249-3 ISSN: 2673-5350 doi: 10.18429/JACoW-IBIC2024-WEP50

System Architecture

The overall system architecture is designed as shown in the Figure 3. The EPICS-IOC represents a hardware BPM electronics, supplying PV variable data for the entire system. The end-users interact with a data visualization interface and monitor the operation of various system modules. Deploying traditional Kafka and ELK components requires extensive and complex environment configuration. To enhance compatibility across different environments and improve the system's portability and scalability, the entire system adopts a microservices architecture. It uses the Docker container engine to deploy Kafka, Zookeeper, Logstash, Elasticsearch, and Kibana within a three-node Kubernetes high-availability cluster [9]. The data acquisition program is implemented using a Python script.

Docker container technology primarily addresses compatibility issues across different environments by providing a virtual Linux system environment, allowing container applications to be compatible with various host environments. Kubernetes cluster technology offers a more advanced and flexible way to orchestrate, schedule, and manage containers in larger and more complex systems.

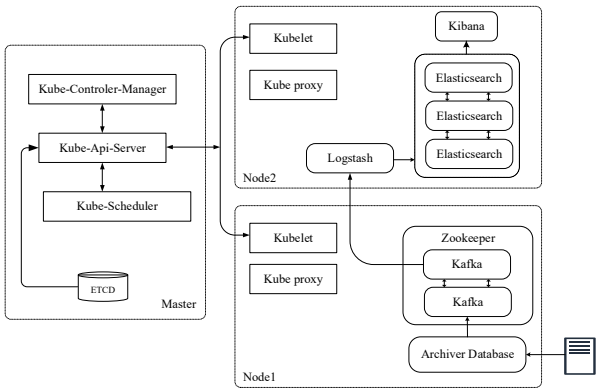

Figure 3: System architecture diagram.

System Implementation

The configuration for services such as Kafka, Zookeeper, and the ELK stack is defined through YAML files within the Kubernetes cluster. Configuration files allow defining services including Service, Deployment, StatefulSet, ConfigMap, PersistentVolume, and PersistentVolumeClaim [10].

Service.yaml enables port configuration for each service. The port configuration for each service is shown in Table 1. For services like Zookeeper and Logstash that do not require external access, their service ports only need to be exposed to other services within the system, in this case, Kafka and Elasticsearch. The default ClusterIP mode can satisfy these requirements. However, Kafka requires data input from external systems into the message queue, and Kibana needs terminal access to the data visualization dashboards, thus for Kafka and Kibana services, the NodePort mode is selected. Elasticsearch is a special case as it needs a web interface to confirm its operational status and internal ports to receive data transmitted from the Logstash pipeline. For these dual types of port exposure requirements, both NodePort and ClusterIP modes are set accordingly.

Table 1: Port configuration table

Service Type	Port Type	Default Port	External Port
Kafka	NodePort	9092	30092
Zookeeper	ClusterIP	2181	-
Logstash	ClusterIP	9600	-
Elasticsearch	NodePort/ ClusterIP	9200/9300	30920/-
Kibana	NodePort	5601	30561

In Kubernetes, container orchestration involves various resource object types. This document deals with Deployment and StatefulSet types. For general services, Deployment is typically chosen as it is used to manage stateless applications where each Pod's replica can replace another without retaining any specific state. Deployment ensures that multiple replicas of an application are evenly distributed across the cluster, accessed through load balancers, which helps achieve high availability and load balancing of the application. Deployment supports declarative update strategies, allowing application versions to be updated without downtime. In this document, Kafka, Zookeeper, Logstash, and Kibana are all of the Deployment type, as these services generally do not involve data storage issues. Kafka, as a message queue, usually does not persist data continuously.

StatefulSet is designed for applications that need to maintain persistent state. It guarantees a fixed, persistent identifier for each Pod's replica, even when rescheduled to other nodes. StatefulSet provides stronger guarantees on creating and terminating Pods. They are always created and terminated in order, ensuring the current Pod is running before starting the creation of the next one. This is particularly important for applications requiring strict startup sequences, like Elasticsearch as a distributed database mentioned in this document. When configuring the Elasticsearch service, three replicas are created, and persistent volumes are also created for each Pod, ensuring each Pod has its storage which is remounted to the corresponding Pod if rescheduled to another node.

EXPERIMENTS AND RESULTS

The underlying platform for the system is the Proxmox VE virtualization platform, which includes a Kubernetes cluster environment composed of three virtual machine nodes. YAML files for various services are deployed within the cluster. The dashboard interface like Figure 4 allows monitoring of service conditions. After starting Kafka and the ELK components, the data acquisition program is executed, entering Kafka to view the JSON array-type data already written into the message queue. Entering the Logstash backend to execute the conf file transmits the data from the Kafka message queue to Elasticsearch, where Elasticsearch automatically generates a data index. As

ISBN: 978-3-95450-249-3 ISSN: 2673-5350 doi: 10.18429/JACoW-IBIC2024-WEP50

shown in the Figure 5, the Kibana dashboard interface offers a visualization interface, which can be added to query the data in Elasticsearch.

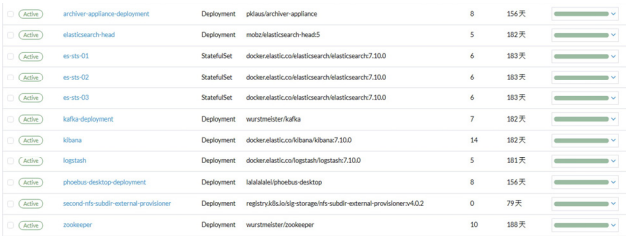

Figure 4: Kubernetes dashboard.

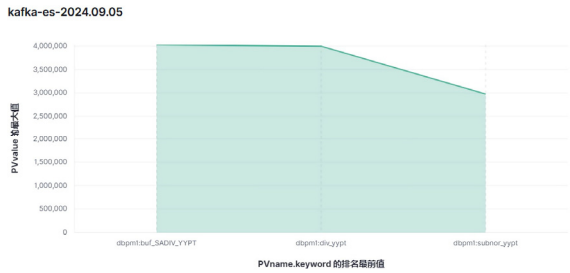

Figure 5: Kibana visualization dashboard.

CONCLUSION

This paper proposes a method for real-time analysis and visualization of PV data within BPM using the Kafka message queue and ELK stack. Given the extensive data querying and processing requirements, this method has significant practical value. The research approach combines Docker container technology and a highly available Kubernetes cluster with a microservices architecture, enhancing system stability and scalability for handling large-scale data in major scientific installations. The system has undergone prolonged testing in a production environment, demonstrating good operational performance and validating the technical feasibility of this approach. Future research will integrate different system data collection, querying, and processing needs to further enhance system performance and reliability.

REFERENCES

[1] Elastic stack, https://www.elastic.co/elastic-stack/

[2] Apache Kafka, https://kafka.apache.org/

[3] Apache Zookeeper, https://zookeeper.apache.org/

[4] M. Bajer, "Building an IoT Data Hub with Elasticsearch, Logstash and Kibana", *IEEE Xplore*, Aug. 01, 2017. doi: 10.1109/FiCloudW.2017.101

[5] Logstash, https://www.elastic.co/logstash

[6] Elasticsearch, https://www.elastic.co/elasticsearch

[7] Kibana, https://www.elastic.co/kibana

[8] Sasaki, Shinya, T. T. Nakamura, and M. Hirose, "Monitoring system for IT infrastructure and EPICS control system at SuperKEKB", *17th international conference on accelerator and large experimental physics control systems (ICALEPCS'19)*, New York, USA, 2019, pp. 05-11. doi:10.18429/JACoW-ICALEPCS2019-WEPHA134

[9] E. Kristiani, C.-T. Yang, C.-Y. Huang, Y.-T. Wang, and P.-C. Ko, "The Implementation of a Cloud-Edge Computing Architecture Using OpenStack and Kubernetes for Air Quality Monitoring Application", *Mobile Networks Appl.*, vol. 26, pp. 1070-1092, Jul. 2020. doi: 10.1007/s11036-020-01620-5

[10] R. Wang, Y. Guo, N. Xie, R. Gu, and Z. Li, "A new deployment method of the archiver application with Kubernetes for the CAFe facility", *Radiat. Detect. Technol. Methods*, vol. 6, pp. 508-518, Oct. 2022. doi:10.1007/s41605-022-00356-y