# ***HPC***

# ***MONITORING***

TEAM MEMBERS:

AMOGH VASISHTA

BHARGAVI RAO

BRINDA S

H A VISHWADATTA

SAKSHYA M M

## **INTRODUCTION**

The HPC Monitoring project aims to develop a comprehensive monitoring system for High-Performance Computing (HPC) environments. HPC systems play a crucial role in scientific research, data analysis, and computational simulations. Monitoring these systems is essential to ensure their optimal performance, identify bottlenecks, and detect potential issues.

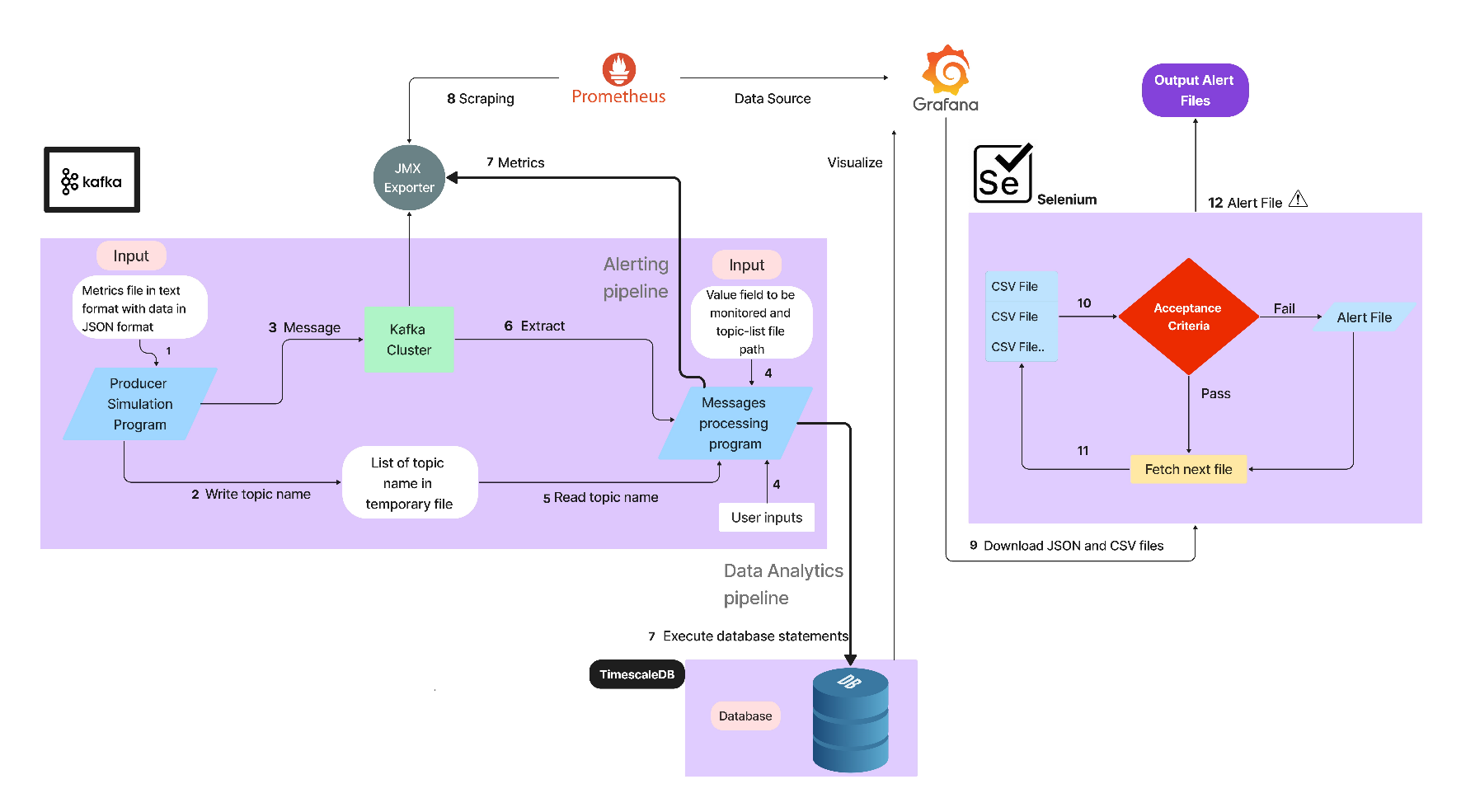
This project leverages the power of Kafka, Prometheus, JMX Exporter, TimescaleDB, Selenium, and Grafana to collect, analyze, and visualize metrics from HPC systems. Kafka is utilized as a real-time messaging system for data ingestion, allowing seamless streaming of metrics from various sources. Prometheus serves as a metrics collection and storage solution, providing a robust infrastructure for data analysis. JMX Exporter enables the monitoring of Java applications by exposing JMX (Java Management Extensions) metrics to Prometheus. TimescaleDB is utilized as a time-series database for efficient storage and retrieval of time-stamped metrics data. Selenium, an automated testing framework, enables the automation of web interactions with Grafana, a popular visualization tool for displaying metrics.

Grafana plays a critical role in this project by providing a user-friendly interface for visualizing and exploring the collected metrics. It offers a variety of panels, each suited for different types of metrics and data representations. The integration with Selenium allows for automated interaction with Grafana, enabling the extraction of metrics from different panel types. The retrieved metrics are then stored, analyzed, and visualized using Prometheus, JMX Exporter, and TimescaleDB.

By integrating these technologies, the HPC monitoring system enables real-time data streaming, automated metric retrieval from Grafana panels, comparative analysis against acceptance criteria, monitoring of Java applications through JMX metrics, and visual representation of metrics through Grafana. The system facilitates the detection of performance anomalies, proactive alerting, and assists in optimizing the overall performance of HPC environments.

This report provides an overview of the project, detailing the integration of Kafka, Prometheus, JMX Exporter, TimescaleDB, Selenium, and Grafana. It discusses the functionalities and interactions of these technologies and highlights the significance of monitoring in HPC systems. Furthermore, it presents the methodology, results, and achievements of the project, along with potential future enhancements for the HPC monitoring system.

## **SYSTEM ARCHITECTURE AND DESIGN**



The system architecture of the HPC Monitoring project showcases the integration of various technologies to enable efficient monitoring and retrieval of metrics. The architecture diagram provides a high-level representation of the system's components and their interactions, offering a comprehensive view of how data flows through the monitoring ecosystem.

At the core of the architecture, we have Kafka, a distributed messaging system, which serves as the central communication channel. Kafka facilitates the seamless exchange of metrics data between data producers and data consumers. Data producers include the HPC systems and applications that generate metrics, while data consumers include the components responsible for data storage, analysis, and visualization.

Prometheus, a powerful monitoring and alerting toolkit, is a key component of the architecture. It collects metrics from various sources and stores them in a time-series database called TimescaleDB. TimescaleDB is a high-performance relational database optimized for time-series data, ensuring efficient storage and retrieval of metrics.

To expand the monitoring capabilities to Java applications, the architecture incorporates JMX Exporter. JMX Exporter provides a bridge between Java Management Extensions (JMX) and Prometheus, enabling the collection of metrics from Java applications and exposing them in a format consumable by Prometheus.

The architecture also leverages Selenium, an automated testing framework, to interact with the web interface of Grafana. Selenium scripts automate the process of logging into Grafana and extracting the metrics being displayed. These metrics are then stored for further analysis and comparison.

Grafana, a popular visualization and monitoring platform, plays a crucial role in the architecture. It provides a user-friendly interface for visualizing the collected metrics and creating dashboards and alerts. Grafana enables users to gain valuable insights from the monitored metrics and supports the generation of reports and visualizations for analysis and decision-making.

The architecture diagram not only illustrates the flow of metrics data through the system but also highlights the automation capabilities and the integration of specialized components for monitoring Java applications and interacting with the Grafana interface.

## **REQUIREMENTS**

Software Requirements:

1. Kafka: Apache Kafka 2.8.0 or more
2. Prometheus: Prometheus 2.30.3 or more
3. TimescaleDB: TimescaleDB 2.4.1 with PostgreSQL 13.4 or more
4. JMX Exporter: JMX Exporter 1.16.0 or more
5. Selenium: Selenium WebDriver with ChromeDriver
6. Grafana: Grafana 8.3.2 or more

Disk Space Requirements:

1. Kafka: Allocate at least 10 GB of disk space for Kafka logs and data.
2. Prometheus: Estimate disk space based on the data volume and retention policy. Allocate at least 100 GB for moderate-scale deployments.
3. TimescaleDB: Estimate disk space based on the data volume and retention policy.
4. Selenium: Allocate disk space for storing Selenium scripts and any required dependencies. A few hundred megabytes should be sufficient.

Memory Requirements:

1. Kafka: Allocate at least 4 GB of memory for Kafka to handle incoming messages and data processing.
2. Prometheus: Allocate at least 4 GB of memory for Prometheus, but adjust based on the scale of your monitoring system and data volume.
3. TimescaleDB: Allocate memory for TimescaleDB's data caching and indexing operations. Allocate at least 4 GB of memory, but adjust based on the data volume and query requirements.
4. Selenium: Provide sufficient memory for Selenium to run the automated web interface interactions smoothly. Allocate at least 2 GB of memory, but adjust based on the complexity of the web pages and the browser being used.
5. Grafana: Allocate at least 4 GB of memory for Grafana, but adjust based on the usage and number of concurrent users.

These are just example versions and recommended values. Please ensure to refer to the official documentation of each software component for the latest versions and detailed system requirements specific to your environment and use case.

## **SETUP AND CONFIGURATION**

1. Kafka:
   * Install Kafka by downloading the Kafka binaries from the Apache Kafka website.
   * Extract the downloaded archive and configure the Kafka broker properties, such as `server.properties`, to specify the broker-specific settings like port, log directory, and replication factor.
   * Start the Kafka broker using the `bin/kafka-server-start.sh` script.
   * Create Kafka topics using the `bin/kafka-topics.sh` script, specifying the topic name, number of partitions, and replication factor.
2. Prometheus:
   * Download Prometheus from the Prometheus website and extract the archive.
   * Configure the Prometheus server by editing the `prometheus.yml` file, specifying scrape targets (i.e., endpoints from which to collect metrics), retention policy, and other settings.
   * Start the Prometheus server using the `prometheus` executable.
   * Access the Prometheus UI at `http://localhost:9090` to explore and query the collected metrics.
3. TimescaleDB:
   * Install PostgreSQL on your system, which is the underlying database for TimescaleDB.
   * Install TimescaleDB extension by following the installation instructions provided in the TimescaleDB documentation specific to your PostgreSQL version.
   * Create a TimescaleDB hypertable to store the metrics data, defining the time column and other relevant dimensions.
   * Configure TimescaleDB settings, such as retention policy, chunk size, and compression, based on your requirements.
   * Connect to TimescaleDB using a PostgreSQL client and interact with the hypertable to insert, query, and manage the metrics data.
4. JMX Exporter:
   * Download the JMX Exporter JAR file from the official repository.
   * Configure the JMX Exporter by creating a YAML configuration file that specifies the target Java applications to monitor and the desired metrics to collect.
   * Start the JMX Exporter using the command `java -javaagent:/path/to/jmx\_exporter.jar -javaagent:/path/to/config.yaml`.
   * Verify that the JMX Exporter is scraping metrics from the target Java applications by accessing the `/metrics` endpoint exposed by the exporter.
5. Selenium:
   * Install Selenium WebDriver for your preferred programming language (e.g., Python, Java).
   * Set up a web browser driver (e.g., ChromeDriver, GeckoDriver) compatible with the chosen browser.
   * Write Selenium scripts to automate the interactions with the Grafana frontend, such as logging in, navigating to specific pages, and extracting metrics data.
   * Execute the Selenium scripts at the desired intervals using a scheduling mechanism (e.g., cron jobs, task scheduler).
6. Grafana:
   * Download and install Grafana from the Grafana website.
   * Start the Grafana server using the appropriate executable for your operating system.
   * Access the Grafana web interface at `http://localhost:3000` and log in using the default credentials (admin/admin).
   * Configure data sources in Grafana to connect to TimescaleDB and Prometheus to fetch metrics data.
   * Create dashboards in Grafana to visualize the metrics, selecting appropriate panels and configuring queries, thresholds, and visualizations.

NOTE: These are general steps for setting up and configuring each tool. For more detailed steps with commands, refer the GitHub repository whose link has been provided at the end of the report.

These tools can also be easily installed by running the shell scripts provided in the GitHub repository.

## **PROJECT WORKFLOW**

The workflow of this project is given below which is based on the code provided in the GitHub repository:

1. Metrics Collection:
   1. The code sets up a Kafka producer using the `KafkaProducer` class from the `kafka` library.
   2. It reads a list of monitored metrics from a file.
   3. The producer sends the metrics data to a Kafka topic using the `send()` method.
   4. The code includes two simulation modes: `latency\_simulation()` and `frequency\_simulation()`.
   5. In `latency\_simulation()`, metrics data is sent to the topic with a delay, simulating latency.
   6. In `frequency\_simulation()`, metrics data is sent to the topic with varying frequencies.

b. JMX Exporter:

* 1. The JMX Exporter is a separate component used to extract application-specific metrics from Java applications.
  2. It connects to Java applications via JMX to retrieve metrics data exposed by the applications.
  3. The JMX Exporter transforms the retrieved metrics data into a format compatible with Prometheus.
  4. The JMX Exporter exposes the transformed metrics data through an HTTP endpoint.

c. Prometheus:

* 1. Prometheus is a monitoring and alerting toolkit.
  2. It scrapes metrics data from various sources, including the JMX Exporter's HTTP endpoint.
  3. Prometheus collects, stores, and processes the metrics data for further analysis and visualization.

1. Metrics Storage:

a.TimescaleDB:

* 1. TimescaleDB is used as the database solution for storing the collected metrics data.
  2. The code in Code Snippet 2 inserts the metrics data into the TimescaleDB database.
  3. TimescaleDB provides scalability and optimizations for time-series data storage.

1. Metrics Visualization:
   1. Grafana:
   2. Grafana is a popular visualization tool used for creating dashboards and displaying metrics.
   3. While not explicitly shown in the provided code snippets, Grafana can be integrated with Prometheus and TimescaleDB to visualize the collected metrics data.
   4. Grafana can connect to Prometheus as a data source and create dashboards to display the metrics stored in TimescaleDB.
2. Regular Metrics Collection and Comparison:
   * 1. The provided Selenium script, scheduled at regular intervals (e.g., every hour), re-collects metrics from Grafana.
     2. The newly collected metrics are compared with the metrics from the previous run to identify any changes or discrepancies.
     3. If there are failed tests or metrics outside the expected range, alerts can be triggered for further investigation or action.
     4. The code also includes functionality for logging into Grafana, fetching panel data from specific URLs, and performing alerting based on acceptance criteria and deviation analysis.

This workflow illustrates the sequence of steps involved in the HPC Monitoring project, starting from metrics collection through Kafka, JMX Exporter, and Prometheus, to storage in TimescaleDB. It then covers metrics visualization using Grafana with the assistance of Selenium WebDriver for interacting with the Grafana panels. Finally, it highlights the reporting of the collected metrics and the regular re-collection and comparison of metrics for monitoring and alerting purposes.

## **PROCEDURE**

1. Firstly, start all the services from the terminal.
2. Verify all the services are up through Prometheus targets menu in 9090 port.
3. Start the producer simulation program by passing the file name in the command line.(Note:the file should be in the same directory).
4. Note down the temporary file path.
5. Start the consumer simulation program .
6. Give the necessary inputs in the terminal.
7. Verify that the data is being exported in 8080 port(JMX Exporter).
8. Also Verify that data is inserted to the respective Hyper-table in TimescaleDB administrator (DBeaver is the tool used here).
9. Visualize the data in the dashboard setup for the respective metrices in Grafana.

10.Run Selenium tests for these panels which collects the panel data in the form of csv file.

**CHALLENGES**

It is essential to carefully plan and design the simulation and validation framework, ensuring proper data formatting, synchronization, and integration between the tools like Kafka, Prometheus, TimescaleDB and Grafana. We may encounter some challenges due to the differences in these technologies and their intended purposes. Some of the challenges we faced while working with these tools are listed below:

1. Prometheus:
2. Compatibility issues: Use a version of Prometheus that is compatible with the operating system and architecture. Installing an incompatible version can lead to installation or runtime errors.
3. Configuration errors: Prometheus requires proper configuration to collect metrics from targets and define alerting rules. Mistakes or typos in the configuration file can cause issues during installation or runtime. Verify the correctness of the configuration file, paying attention to the syntax and necessary parameters by referring to the Prometheus documentation for guidance on configuration options.
4. Resource limitations: Prometheus consumes CPU, memory, and disk space, especially when collecting and storing large amounts of metrics. Ensure that the server has sufficient resources available to accommodate Prometheus' requirements.
5. While exploring different tools for our project, we came across an alerting system called Node Exporter. The challenges we faced while working with this are:
6. Permission issues: We faced permission-related problems while installing or running Node Exporter. We made sure we have the necessary permissions to install and run the software. Running the installation commands with appropriate administrative privileges or using the "sudo" command if required.
7. Compatibility issues: Using an incompatible version may lead to installation or runtime errors. Use a version of Node Exporter that is compatible with your server's operating system and architecture.
8. Configuration errors: Incorrect configuration settings can cause issues with Node Exporter, such as incorrect metric collection or failure to communicate with the Prometheus server. Double-check the configuration file for Node Exporter and ensure that the necessary parameters are correctly set.
9. System resource limitations: If the server is under heavy load or has limited resources, it may affect the performance of Node Exporter or the accuracy of the collected metrics. Monitor the system resources during installation and ensure that the server has enough CPU, memory, and disk space available for Node Exporter to operate optimally.
10. TimescaleDB:
11. When setting up TimescaleDB in a VM, we encountered challenges with the Confluent Kafka approach. The installation of Confluent Kafka caused significant slowdowns in both the VM and the main machine, and it even led to occasional crashes. As a result, we had to explore alternative solutions and deviate from the Confluent Kafka approach.
12. Installation Issues: Installing Confluent Kafka and its JDBC connector can be complex, requiring specific dependencies and configurations. Compatibility issues with the operating system or other software components can arise.
13. Connectivity Problems: Establishing a reliable connection between Kafka and TimescaleDB can be challenging. Configuring network settings, ensuring proper firewall rules, and configuring authentication credentials need careful attention to ensure successful connectivity.
14. Configuration Complexity: Configuring the JDBC connector and setting up the necessary properties to establish a synchronized connection between Kafka and TimescaleDB can be intricate. Understanding the required configuration parameters and their values, as well as integrating them correctly, can be time-consuming and prone to errors.
15. Troubleshooting and Debugging: Debugging issues with the JDBC connector and resolving connectivity problems can be challenging. Identifying and interpreting error messages, analyzing logs, and troubleshooting network or configuration issues require a systematic and meticulous approach.
16. OpenSearch is an another open-source, community driven search engine tool we came across during our project. But due to its extremely specific hardware and software requirements, our systems didn’t support the functioning of OpenSearch. Some of the issues we encountered are:
17. Insufficient memory in our standalone system's VMs: When the virtual machine has insufficient memory allocated to it, we encountered issues during installation or startup. For example, if the allocated memory is below the minimum requirements, such as less than 2 GB, OpenSearch fails to start or operate properly.
18. Slow Performance: Insufficient memory can result in slow performance as OpenSearch struggles to handle the data and query loads. For example, if the allocated memory is significantly lower than the recommended guidelines, such as less than 8 GB for a sizable dataset, we experience prolonged response times and sluggish behaviour which can even lead to crashing of Linux in our VMs.
19. Resource Contention: When multiple processes or applications are running in the virtual machine and competing for limited memory resources, resource contention issues can arise. For instance, if the virtual machine has only 6 GB of memory allocated, but other processes like Kafka producer and consumer consume a significant portion of it, OpenSearch may experience performance degradation or instability.
20. Access to cloud VMs: Access to GCP or Azure requires credit card for billing based on the resources consumed, including CPU, memory, storage and any additional services or features associated with the VMs.

## **EPILOGUE**

In conclusion, the HPC Monitoring project successfully implemented a comprehensive monitoring solution for High-Performance Computing (HPC) systems. The project combined various tools and technologies to collect, store, visualize, and analyze metrics from distributed applications running on the HPC infrastructure.

By leveraging Kafka, JMX Exporter, Prometheus, TimescaleDB, and Grafana, the project established a robust and scalable monitoring architecture. Kafka facilitated the reliable and efficient collection of metrics data from distributed sources, while JMX Exporter enabled the extraction of application-specific metrics from Java applications. Prometheus acted as the central data repository, collecting and storing the metrics data. TimescaleDB provided a high-performance time-series database for efficient storage and retrieval of the metrics.

The integration of Grafana with Selenium WebDriver enabled the automation of metrics retrieval from the Grafana visualization tool. Selenium interacted with the Grafana web interface, allowing for seamless login, navigation through different panels, and extraction of the displayed metrics. This automation facilitated regular re-collection and comparison of metrics, aiding in the identification of any changes or anomalies.

The HPC Monitoring project offered several benefits to the monitoring and management of HPC systems. It provided real-time insights into the performance and behavior of distributed applications, allowing for proactive monitoring and optimization. The ability to visualize and analyze metrics in Grafana empowered administrators and stakeholders to make data-driven decisions and detect potential issues promptly.

Additionally, the project offered extensibility and flexibility, allowing for the integration of additional data sources, customization of dashboards, and the application of advanced analytics techniques. The use of open-source tools ensured cost-effectiveness and the availability of a vibrant community for support and enhancements.

Moving forward, the HPC Monitoring project can be further expanded to incorporate machine learning and anomaly detection algorithms for automated alerting and predictive analytics. It can also integrate with additional monitoring tools and frameworks to capture a more comprehensive view of the HPC system's performance.