< Go to the original

Exposing JMX Metrics from Kafka to Prometheus and Grafana

Situation

Bill WANG

Follow

androidstudio · July 15, 2024 (Updated: July 15, 2024) · Free: No

Situation

Our goal was to monitor Kafka and its components (Kafka Connect, Schema Registry, and Zookeeper) by exposing their JMX metrics to Prometheus and visualizing them in Grafana. This required setting up a JMX exporter for Kafka and configuring the system to ensure seamless metric collection and visualization.

Task

I was responsible for configuring the JMX exporter for Kafka and its components, integrating it with Prometheus for metric scraping, and setting up Grafana dashboards to visualize these metrics. The tasks included:

- Setting up Kafka by Confluent for Kubernetes (CFK)
- Configuring Prometheus to scrape JMX metrics
- Visualizing the metrics in Grafana

Action

1. **Setup Prerequisites:**

- Installed Kafka by Confluent for Kubernetes (CFK) using Confluent Operator.
- Installed Prometheus and Grafana using their official Helm charts.

2. **Configure JMX Exporter:**

- Confirmed that the JMX exporter was running as a Java agent in Confluent's Kafka and other components by checking the Kafka pod configuration.
- Verified the JMX exporter setup with HTTP port 7778.
- Retrieved JMX configuration examples from the JMX Exporter GitHub repository for Kafka, Kafka Connect, and Zookeeper.

3. **Update Configurations:**

- Noted that direct updates to the ConfigMap reverted to the old configuration. Decided to use Custom Resource Definitions (CRDs) for updates.
- Edited the CRD specifications to include JMX metrics configurations and saved the changes, which updated the ConfigMap.
- Ensured similar updates for Kafka Connect, Schema Registry, and Zookeeper.

4. **Expose JMX Metrics:**

- Listed service names in the Confluent namespace and set up port forwarding for Kafka Connect service.

5. Configure Prometheus:

- Edited Prometheus' ConfigMap to add scrape configurations for Kafka and other components.
- Provided detailed DNS names due to different namespaces for Prometheus and Kafka services.

6. Visualize Metrics in Grafana:

- Configured Prometheus as a data source in Grafana.
- Imported a sample Kafka dashboard (ID 18276) to visualize Kafka metrics.
- Customized Grafana dashboards as needed using Grafana's documentation.

Result

The integration was successful, achieving the following:

- JMX metrics for Kafka and its components were exposed and accessible via Prometheus.
- Prometheus was configured to scrape these metrics effectively.
- Grafana dashboards provided real-time visualization of Kafka metrics, aiding in monitoring and analysis.

By methodically setting up and configuring the JMX exporter, Prometheus, and Grafana, we successfully created a robust monitoring solution for Kafka and its components, enabling better visibility and management of the system's performance and health.

Follow-Up

Decisions:

- Use CRDs for updating JMX configurations to ensure persistence.
- Set up Prometheus and Kafka in different namespaces for better organization.

Next Steps:

- Continuously monitor and refine the configurations to ensure optimal performance.
- Explore additional metrics and custom dashboards based on user feedback and operational needs.

References:

- Confluent for Kubernetes Quickstart
- JMX Exporter GitHub
- Kafka Dashboard on Grafana Labs

Learning is fun

DevOps # Kubernetes # Kafka # Prometheus # Grafana # STAR

#devops #kubernetes #kafka #interview #best-practices