Custom Metrics Driven Horizontal Pod Autoscaling (HPA) in Kubernetes

This project shows how to leverage custom metrics to dynamically manage resource scaling in Kubernetes, although traditional scaling focuses on CPU and memory usage. Custom metrics, such as request count, message queue length, etc., provide a more accurate representation of application-specific factors. This approach allows Kubernetes to ensure tailored scaling as per specific business needs, improving resource efficiency and enhancing the performance of the application with exclusive workloads. Employing custom metrics requires a Horizontal Pod Autoscaler (HPA) for automatic scaling, a metrics monitoring tool, and a metrics adapter to connect custom metrics with the Kubernetes environment for dynamical scaling.

Implementation: Orchestrator Application

The setup in this project serves as a simple demonstration of how Custom Metrics HPA can be effectively used to dynamically scale applications based on real-time workload measurements. The orchestrator application, developed using **Spring Boot**, acts as a task-processing system, that tracks tasks with various statuses. A custom metric, **queue_length**, represents the number of tasks in the READY status, waiting to be processed. This metric plays an important role in determining when scaling is necessary, as it directly indicates the application's workload. The HPA of the Kubernetes deployment uses queue_length to automatically scale in/out the number of replicas. The **ReactJS** application interface allows users to add, view, reschedule, and delete tasks, and real-time information regarding the tasks can be viewed in the work queue.

Architecture Overview: Orchestrator Deployment, Prometheus, Prometheus Adapter, and HPA

The above-mentioned implementation is deployed in a Kubernetes cluster environment, and the custom metrics architecture consists of the following components:

- **Orchestrator Deployment**: The orchestrator application is deployed in Kubernetes, with environment variables that provide pod-specific information and the frequency in which the custom metrics must be checked. It includes two services: one for the application API and another for exposing Prometheus metrics.
- **Prometheus**: Prometheus scrapes and monitors the queue_length custom metric at regular intervals from the orchestrator metrics service of the orchestrator deployment. This metric is essential for the HPA to make guided scaling decisions.
- **Prometheus Adapter**: The adapter maps the queue_length metric to Kubernetes resources such as scraping jobs, pods, and namespaces, allowing Kubernetes to use this metric for autoscaling. It also manages the custom metrics API, which is used to retrieve and act on the custom metric data from HPA.
- **Horizontal Pod Autoscaler (HPA)**: The HPA automatically adjusts the number of replicas for the orchestrator deployment based on the queue_length. It increases or decreases the number of replicas as the task load fluctuates, confirming efficient resource utilization.

Conclusion

This process underlines the effectiveness of custom metrics-driven autoscaling in Kubernetes, offering flexible scaling solutions for complex applications. By leveraging queue_length as a custom metric, the system dynamically scales to meet real-time demands. This approach enhances resource efficiency and performance, providing a suitable option for applications that require specific scaling based on definitive workloads, such as task processing, request handling, or user activity.