GSoC 2020 Apache Proposal

Apache RocketMQ Scaler for KEDA

Application

Name : Hien Nguyen

University : Haaga-Helia University of Applied Sciences - Bachelor of Information Technology - (Location: Helsinki, Finland)(expect graduation : August 2020)

Email : hienminhnguyen711@gmail.com / Phone : +358 469 335 071

Github: https://github.com/hiejulia

Related skills : Java, Golang, SQL, NoSQL, Rest, gPRC, Microservices, Test, DevOps, Distributed system, Cloud(AWS, Azure) , Golang, Maven, Docker, Kubernetes

GSoC - Apache RocketMQ Scaler for KEDA proposal

Context

KEDA allows for fine-grained autoscaling (including to/from zero) for event-driven Kubernetes workloads. KEDA serves as a Kubernetes Metrics Server and allows users to define autoscaling rules using a dedicated Kubernetes custom resource definition. KEDA has a number of “scalers” that can both detect if a deployment should be activated or deactivated, and feed custom metrics for a specific event source. In this topic, you need to implement the RocketMQ scalers.

Motivation

- Implement RocketMQ scalers with KEDA

Intuition

- KEDA supports scaling Event driven application, built int scalers for vendors : Azure, AWS, Kafka, GCP, MySQL, RocketMQ,etc; multiple workloads type(jobs,deployments,trigger)
- KEDA does not support Apache RocketMQ now. So we need to create PR in KEDA repo for new support for RocketMQ
- KEDA has event-driven scaling which means it scales based on events instead of RAM or CPU usage, also can be combined with other options like virtual nodes

Solution

- Implement Scalers for RocketMQ with KEDA

Install tools/ softwares

- JDK, Maven, Git, 4g+ free disk for Broker server, terminal, Apache RocketMQ
- Helm, Kubernetes, kubectl,Docker, KEDA, Golang

Basic Configuration

- RocketMQ
  - Build and start RocketMQ locally(with Docker container or extract source release and build binary artifact)
  - Test and research DLedgerRoleChangeHandler.java, SlaveSynchronize.java, algorithms class : AllocateMessageQueueAveragelyByCircle,AllocateMachineRoomNearby, AllocateMessageQueueConsistentHash, etc, Rebalance logic consumer, MQFaultStrategy, ConsistentHash,etc
- KEDA
  - After deployment, we could validate the deployment scales, checking the pods and start process queue message, if message length increase, more pods will be pro-actively added
  - We can also get the number of messages vs the target per pod
  - *After the queue is empty and the specified cooldown period (a property of the `ScaledObject`, default of 300 seconds) the last replica will scale back down to zero.*
  - Benchmark the ScaledObject will be deployed when there is change in the scaling as the number of message increases

```
pollingInterval: 30  # Optional. Default: 30 seconds
  cooldownPeriod:  300 # Optional. Default: 300 seconds
  minReplicaCount: 0   # Optional. Default: 0
  maxReplicaCount: 100 # Optional. Default: 100
```

  - Visualize with metrics

Research:

- Helm
- Apache RocketMQ operator(https://github.com/apache/rocketmq-operator),Docker image
- **Apache RocketMQ multi-replica mechanism** (research about DLedger & OpenMessaging)

- KEDA architectures

How to implement

I think we have to create a PR for KEDA project for new feature request for scaler for Apache RocketMQ.

KEDA is written in Go (gPRC). KEDA has ScaledObject for the service, Metrics adapter, Scaler, Controller.

RocketMQ operator deployment on Kubernetes also support Horizontal scale with Name server cluster scale, Broker Cluster Scale

RocketMQ has Client SDK for Go, Java

More in detailed (more detailed will added)

- create new RocketMQ scaler with metadata, get connection,etc
- check if there is message pending from RocketMQ queue to be processed
- get queue message
- Research all methods for event driven trigger from RocketMQ, implement
- Get metrics spec for scaling for horizontal pod autoscaler
- write test

Key modules implementation architecture design

Core component	**Description**
Runtime	RocketMQ, KEDA Runtime environment
Open Messaging Connect	- OpenMessaging Connect API can be loaded by RocketMQ connection
Main scale handler Authentication	- Add NewRocketMQScaler - Secure patterns & config for authentication -Pod authentication provider
Main Apache RocketMQ scaler	-Main logic implementation to handle scaler for RocketMQ
Apache RocketMQ scaler test	-Test scaler logic

Main Scale Handler

- Create Custom Resources `scaledobjects.keda.k8s.io` and `triggerauthentications.keda.k8s.io` to map event source (RocketMQ, metadata of RocketMQ) to a deployment. Authentication config from `TriggerAuthentication`
- Monitor event source
- return scalers.NewRocketMQScaler(resolvedEnv, triggerMetadata, authParams)

Main Apache RocketMQ scaler

Research about main logic :

- KEDA monitor event source, feed that data to Kubernetes and HPA and drive rapid scale of deployment. Each replica of deployment is actively puling items from event source
- Scale based on events while preserve rich connection & processing semantics with event source (in order processing, retries, deadletter, checkpointing)
- Some main logic
    - No message: scale to 0
    - Message arrives : detect event,activate deployment
        - container connect to RocketMQ abd start pull message
        - more messages : KEDA feed to HPA to drive scale out
        - each replica of deployment is actively processing messages, or each replica is processing a batch of messages in a distributed manner
- `ScaledObject` is used to define how KEDA scale and triggers
- Logic to handle long running executions
    - Run as jobs: run event driven code in kubernetes jobs instead of deployment
        - Custom resource for the cases : no job created, KEDA create a job, when jon starts running, it pulls a single message and processes it to completion
    - leverage the container lifecyle `SIGTERM`from lifecycle hooks
- Logic for authentication
    - Config auth per ScaledObject
    - Re use creds or delegate authentication with TriggerAuthentication

- Create a new RocketMQ scaler passed metadata
- Config authentication
- Get RocketMQ connection
- Function to check if there are pending messages to be processed from RocketMQ
- Function to get message from RocketMQ

- Function to get metrics spec for scaling returns the MetricSpec for the Horizontal Pod autoscaler
- Function to get metrics return value for a supported metric and an error if there is a problem getting the metric
- Close RocketMQ connection

Main Apache RocketMQ test

- RocketMQ metadata, resolved env for test : properly formed metadata, malformed queue length, missing host, missing queue name, etc

Main Apache RocketMQ test

- Set up local env for test
- Start RocketMQ Name Server
- Start RocketMQ Broker
- Send & Receive message test case(add dependencies rocketmq-client), send async/sync/ 1way mode, consume message
- Broadcast with consumer set to broadcast mode, register message listener and consume messge concurrently
- Schedule example: send scheduled message and register message listener
- Batch example : send messages in batches, split into lists
- Filter example: RocketMQ offer some SQL expression to filter out message
- Logappender example:
- Open Message example: OMS Producer & OMS Pull consumer, OMS Push Consumer
- Transaction example: send transactional message and implement transaction listener interface
- Test for parallel producing & consuming in RocketMQ, research more
- Shutdown server

Main Apache RocketMQ for KEDA test

- Install KEDA
- Deploy ScaledObjects, check HPA
- Implement some RocketMQ usage examples, deploy to KEDA(virtual nodes can be test also), generate deployment yaml, deploy container image, apply the deployment to the cluster
- Validate the scale with KEDA and benchmark, profil with event trigger, for example, publish message to queue and then validate the deployment scales(check to see how more pods will be pro-actively added when for example message length continues to increase)
- Clean up the resource

Deliverables & Timetable

All the decision, methods, tasks, implementation will be discussed with mentor beforehand

Date	Time	Description
Now - 18 May	6 weeks	- Install, config the environment(64bit OS Mac), JDK, Maven, Git, 4g+ free disk for Broker server.

		For KEDA: Helm, Kubernetes, . Docker, - Clone source code of RocketMQ, RocketMQ Operator, build and test it locally. - Clone source code of KEDA, build and test it locally - Community:contact with 2 mentors and discuss in depth about RocketMQ.In particular with Apache $Project, including project mailing lists, wikis, issue trackers, test systems. - RocketMQ : read docs, read issues opened on github, read code,research architecture, make some modifications, test all examples with rocketmq queue(order, broadcast, schedule, batch, filter, logappender, openmessaging, transaction). Research about rocketmq multi-replica algorithms(based on DLedger). Download, test&run OpenMessaging Connect API and research about it algorithms - KEDA: research mainly about architecture
18 May - 1 June	2 week - Checkpoint	-Test KEDA some available examples like Kafka, Azure SQS queue,service bus,Event hub, Kinesis stream, RocketMQ(most closer with RocketMQ queue),read docs & examples. Test examples how KEDA works with RocketMQ, Azure queue, service bus, Redis, MySQL, NATS streaming, Kafka, GCP pub/sub - Test RocketMQ examples : Pub/sub, Broadcast, Schedule, Batch, Filter, Logappender, OpenMessaging, Transaction -Discuss with mentor & community
1 June - 15 June	2 week - Checkpoint-Kickstart	- Divide into subtasks, create backlog for tasks - Draft detailed high level architecture for **main ApacheRocketMQ scaler** and discuss with mentor - Write code, refactor code, implementation - Write test
15 June - 29 June	2 week - Checkpoint	- Divide into subtasks, create backlog for tasks - Draft architecture to implement test methods(unit test, integration, E2E test) fo**r main RocketMQ scaler** - Refactor, checkstyle,code coverage, install Best Practice coding tools -Discuss with mentor -Continue writing documentation
29 June - 13 July	2 week - Checkpoint	- Divide into subtasks, create backlog for tasks **- Unit test, integration test, E2E test for RocketMQ with KEDA** **-** Deploy KEDA **- Code review & feedback** - Discuss with mentor & community

13 July - 27 July	2 week	**-Research, draft architecture and implementation for metrics** **- Code review & feedback** - Continue writing documentation - Discuss with mentor
27 July - 10 Aug	2 week	- Refactor code, code coverage, benchmark, profiler test, code review with mentor - Dashboard for KEDA deployment implementation https://github.com/kedacore/dashboard - Discuss with mentor & community
10 Aug- 31Aug		- Write a final documentation for review/ evaluation - Discuss with metor & community

Resources

https://github.com/kedacore/sample-go-rabbitmq

https://github.com/kedacore/charts

https://github.com/kedacore/dashboard

https://github.com/apache/rocketmq-exporter

https://github.com/kedacore/keda

https://github.com/kedacore/keda-olm-operator

https://github.com/kedacore/keda-scaler-durable-functions

https://dev.to/anirudhgarg_99/scale-up-and-down-a-http-triggered-function-app-in-kubernetes-using-keda-4m42

- View KEDA operator pod via kubectl
- View logs & telemetry of KEDA operator container

Scheduling

Schedule : Weekday: 5 hours/ day/ Weekend: 8-10 hour/day

Other commitments

Summer lectures, self studies, Work

Community engagement

Discuss, feedback with mentor & community, code review