



Ceph RGW Message Queue API for Serverless Computing

Yuval Lifshitz

 yuvalif

Huamin Chen

 rootfs
 root_fs

Red Hat

Agenda

- Problem Statement
- Review of Ceph RGW Bucket Notification
- PubSub, Push, and Pull
- Overview of Ceph RGW Message Queue API and KEDA Integration
- Message Queue Architecture and Deep Dive

Why MQ APIs in Ceph RGW?

- Existing **PubSub** (pull mode) is going to be deprecated as it has several deficiencies:
 - require special zone
 - non standard APIs
 - functional limitations
 - consumers don't scale
- Existing **bucket notification** (push mode) APIs in Ceph RGW enable a host of use cases:
 - Automated Data pipeline
 - Automated ETL

Why MQ APIs in Ceph RGW?

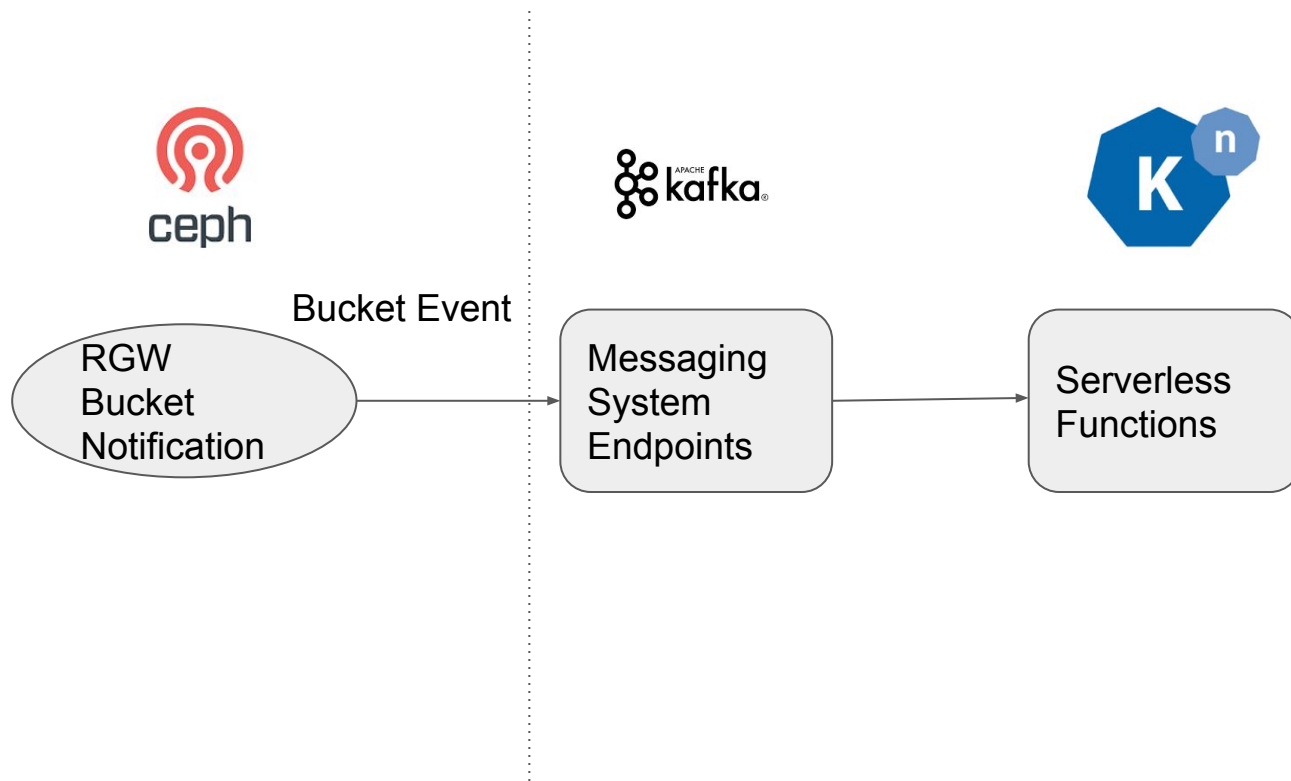
- Pushing bucket notifications **directly** into the serverless functions (e.g. to knative over kafka or http) works well for simple event handling
- Pushing bucket notifications to a **message broker** can handle more complex cases (e.g. long running executions that may fail midway). But may introduce new complexity in the form of a message broker...

So... Native MQ APIs are the answer!

Ceph RGW Bucket Notification Push Mode

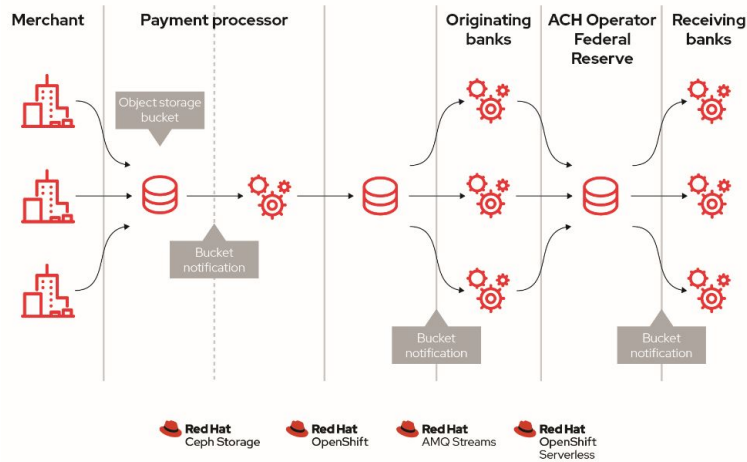
- **Functionality**
 - Tracking object changes in a bucket
 - Provides AWS compatible REST API
- **Topic**
 - Aggregates different published events
- **Notification**
 - Changes on bucket are published to a topic
 - Delivered to Kafka, AMQP, HTTP endpoints
 - Filtering based on object name, attributes and tags
- **Event**
 - S3 compatible event schema

Ceph RGW Bucket Notification Push Mode

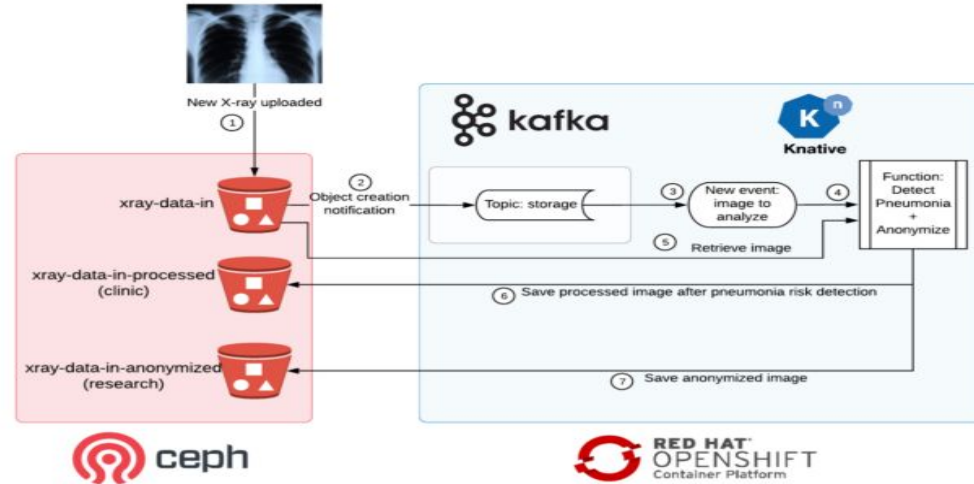


Applications: Automated Event Driven Data Pipeline

Real Time. Event Driven. Scalable. Automated.



Data pipeline with Ceph notifications and KNative Serving



Source:

<https://www.redhat.com/fr/resources/automating-data-pipelines-overview?source=searchresultlisting&page=16>

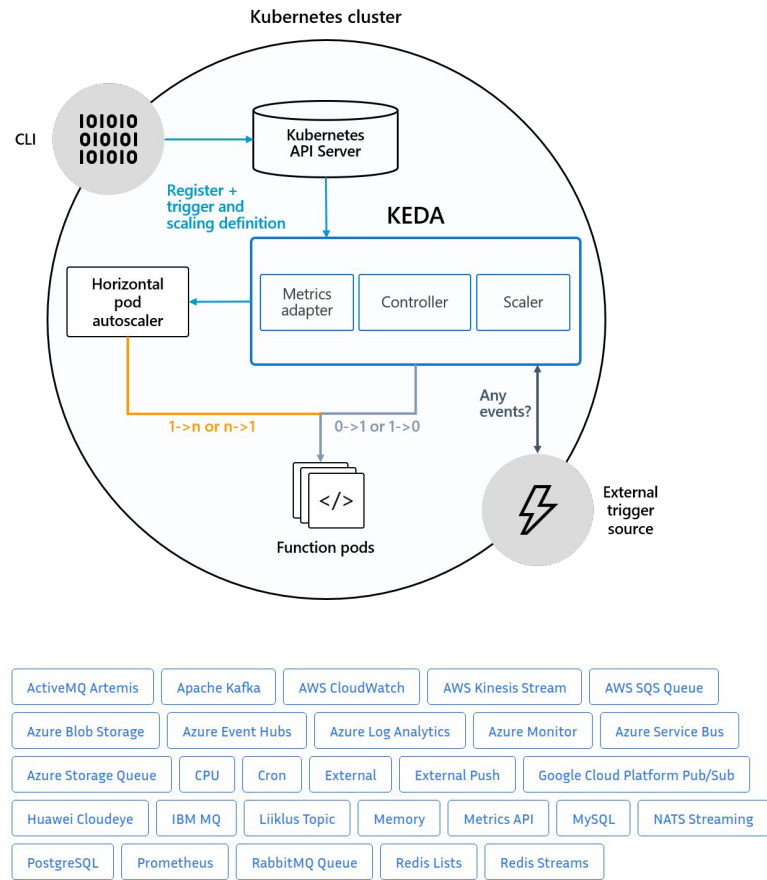
<https://medium.com/analytics-vidhya/automated-data-pipeline-using-ceph-notifications-and-kserving-5e1e9b996661>

PubSub, Push, and Pull

	PubSub (deprecated)	Message Push	Message Pull
Message Delivery Mechanism	Notifications stored in bucket in a special pubsub zone	Notifications sent to an external message broker	Notifications stored in RADOS backed FIFO
Serverless Function Programming Model	Function constantly polling for new events in the notification bucket	Based on the external message queue	Function reading from the FIFO based on autoscaling trigger
Autoscaling Trigger	none	Based on Serverless function utilization	Based on the approximated queue size
Producer Reliability	RADOS	RADOS until acked by external message queue	RADOS
Consumer Reliability	Notifications deleted after consumer acks. Stateful consumer	Based on the external message queue	Notifications deleted after consumer acks or timeout expires. Stateless consumer

KEDA Overview

- KEDA is a Kubernetes event driven lightweight Serverless framework.
- Key concepts:
 - Scaler
 - Metrics
- Can be a Knative event source



Message FIFO - Introduction

- Based on AWS SQS API
 - Implements a subset of it (the parts needed for bucket notifications) with minor modifications
 - Allows for standard tools (e.g. boto3) integration
- Using Ceph for durability and scalability
- Most of the implementations is inside the OSD as an “Object Class” for performance and scalability
- Similar to other Object Classes there will be C++ client libraries for the message FIFO
- At RGW level (REST APIs) the intent is to expose only the APIs needed for bucket notifications
 - in the future we may expose a fully functional REST based message FIFO

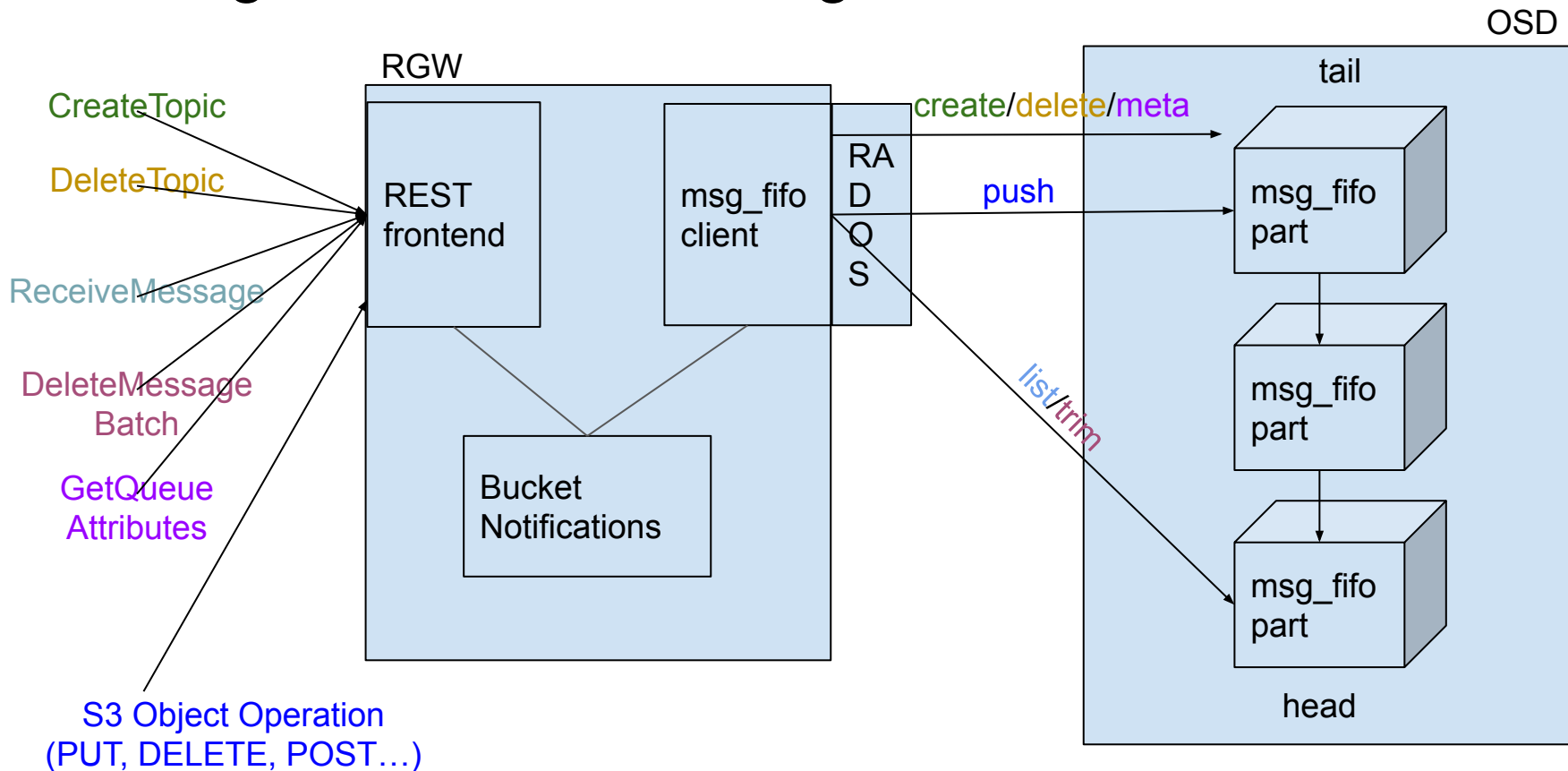
Message FIFO - C++ API

- `create (max_part_size, max_entry_size, visibility_timeout, retention_period)`
- `delete()`
- `meta () -> max_part_size, max_entry_size, visibility_timeout, retention_period, appx_queue_size`
- `push (entry)`
- `list (max_entries, start_marker)`
 - `start_marker` is optional. If not provided will list from the end of the queue
- `trim (start_marker, end_marker)`
 - `start_marker` is optional. If not provided will trim from the end of the queue until `end_marker`

Message FIFO - REST API

- `ReceiveMessage (QueueUrl, MaxNumberOfMessages)`
- `DeleteMessageBatch (QueueUrl, DeleteMessageBatchRequestEntry)`
 - There must always be 2 elements in the batch (start id and end id)
- `GetQueueAttributes (QueueUrl)`
 - May return: `VisibilityTimeout`, `MaximumMessageSize`, `MessageRetentionPeriod`, `ApproximateNumberOfMessages`
- Message FIFO is created and deleted based on the bucket notification topic creation (with endpoint of type message FIFO). No explicit `Create/Delete Queue` API added
- Messages are queued from the bucket notification mechanism inside the RGW. No explicit `SendMessage/Batch` API added

Message FIFO - Block Diagram



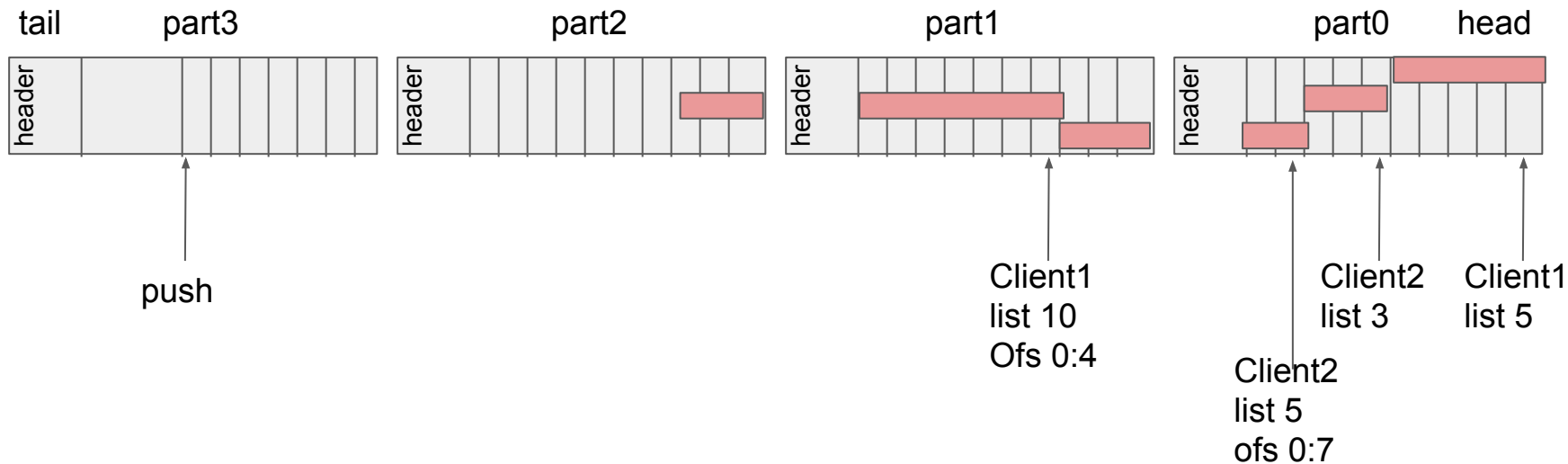
Message FIFO - Server Durability and Scalability

- FIFO is based on RADOS objects
 - can grow with available storage for its pool
- Based on Ceph's durability
- Operations inside the RGW are synchronous - writing the notification to the queue happens during the processing of the object request that triggered the notification
- RGWs availability and scale achieved as all state is saved in the RADOS objects
- One OSD writes to the tail and one read from head. Multiple FIFOs can be used to allow for OSD scalability
- Main object accessed only when part is added or removed

Message FIFO - Client Durability and Scalability

- “At-least-once” guarantee
 - Duplicate notifications may exist upon failures
- Multiple clients can read from the same FIFO since the FIFO is marking red segments and make them invisible to consequent reads
- Clients should trim entries once they are done processing them (or storing them in some other persistent storage)
- Segments which are invisible are becoming visible again after “visibility timeout” (if not trimmed). This allows for other clients to pick up on unfinished work
- Segments that were red, but not trimmed, are eventually deleted after the “retention period”

RGW Message FIFO - Visibility Timeout



RGW Message FIFO - Visibility Timeout Expiry

