BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

MONITORING THE OPENSTACK SWIFT OBJECT STORE USING BEANSTALK EVENTS

SLEDOVÁNÍ OBJEKTOVÉHO ÚLOŽIŠTĚ OPENSTACK SWIFT POMOCÍ BEANSTALK UDÁLOSTÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. NEMANJA VASILJEVIĆ**
AUTOR PRÁCE

SUPERVISOR **RNDr. MAREK RYCHLÝ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2021

Brno University of Technology
Faculty of Information Technology

Department of Information Systems (DIFS) Academic year 2021/2022

Master's Thesis Specification

Student: **Vasiljević Nemanja, Bc.**
Programme: Information Technology
Field of study: Information Systems and Databases
Title: **Monitoring the OpenStack Swift Object Store Using Beanstalk Events**
Category: Databases
Assignment:

1. Explore OpenStack Swift object storage, especially its architecture and activities. Study also MinIO object storage. Learn about the object storage OpenIO Software Defined Storage and in which way it uses Beanstalk to monitor and distribute events over the storage.
2. Design a service that will monitor activities in OpenStack Swift and, following the pattern of OpenIO, publish Swift events using the Beanstalk protocol. Consider also the ability to monitor and publish events from MinIO.
3. After consulting with the supervisor, implement the proposed service over OpenStack Swift/MinIO so that compatibility with OpenIO is guaranteed. For verification, also implement a sample client that will be able to subscribe to events using Beanstalk from both OpenIO and OpenStack Swift/MinIO.
4. Test the solution, evaluate and discuss the results. Publish the resulting software as open-source.

Recommended literature:

- Raúl GRACIA-TINEDO, Josep SAMPÉ, Gerard PARÍS, Marc SÁNCHEZ-ARTIGAS, Pedro GARCÍA-LÓPEZ and Yosef MOATTI: Software-defined object storage in multi-tenant environments. *Future Generation Computer Systems*. 99, 54-72, 2019. ISSN 0167-739X. Available at [https://doi.org/10.1016/j.future.2019.03.020]
- OpenStack Docs: Object Storage monitoring. The OpenStack project [online]. 2021 [seen 2021-09-29]. Available at [https://docs.openstack.org/swift/ussuri/admin/objectstorage-monitoring.html]
- Send notifications on PUT/POST/DELETE requests - swift-specs 0.0.1.dev82 documentation. OpenStack Foundation [online]. 2016 [seen 2021-09-29]. Available at [https://specs.openstack.org/openstack/swift-specs/specs/in_progress/notifications.html]

Requirements for the semestral defence:

- Items 1 and 2 finished and item 3 in progress.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Rychlý Marek, RNDr., Ph.D.**
Head of Department: Kolář Dušan, doc. Dr. Ing.
Beginning of work: November 1, 2021
Submission deadline: May 18, 2022
Approval date: October 21, 2021

Abstract

The goal of this thesis is to create software that is able to monitor and publish event notification from Openstack Swift as well as from OpenIO Software Defined Storage (SDS) to Beanstalk queue. This thesis also proposes solution for publishing event notifications from MinIO to Beanstalk queue.

To accomplish this goal, new middleware is proposed that can be run inside pipeline of Proxy Server in OpenStack Swift and as (filter) part of asynchronous service Event-Agent inside OpenIO SDS.

Proposed middleware allows users to specify if they are interested in publishing event notifications for specific objects/containers using metadata. User can specify set of rules involing object properties, such as name (prefix, sufix, substring) and size, and only events satisfying those rules will be published.

The contribution of this thesis is unique software capable of event monitoring from both OpenIO SDS and Openstack Swift.

Abstrakt

Cílem této práce je vytvořit software, který je schopen monitorovat a publikovat notifikace o události z Openstack Swift i z OpenIO Software Defined Storage (SDS) do fronty Beanstalk. Tato práce také navrhuje řešení pro publikování notifikaci o událostech z MinIO do fronty Beanstalk.

K dosažení tohoto cíle je navržen nový middleware, který lze spouštět uvnitř pipeline proxy serveru v OpenStack Swift a jako (filtr) součást asynchronní služby Event-Agent uvnitř OpenIO SDS.

Navržený middleware umožňuje uživatelům určit, zda mají zájem o publikování notifikaci o události pro konkrétní objekty/kontejnery pomocí metadat. Uživatel může specifikovat sadu pravidel zahrnující vlastnosti objektu, jako je název (prefix, přípona, podřetězec) a velikost, a budou publikovány pouze události splňující tato pravidla.

Přínosem této práce je unikátní software schopný monitorování událostí z OpenIO SDS i Openstack Swift.

Keywords

OpenIO Software Defined Storage, Openstack Swift, MinIO, Beanstalk queue, Event monitoring, Event notification, Amazon S3 event notification, Object storage

Klíčová slova

OpenIO Softwarově definované úložiště, Openstack Swift, MinIO, Beanstalk fronta, Monitorování událostí, Oznámení o událostech, Amazon S3 oznámení o události, Objektové úložiště

Reference

VASILJEVIĆ, Nemanja. *Monitoring the OpenStack Swift Object Store Using Beanstalk Events.* Brno, 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor RNDr. Marek Rychlý, Ph.D.

Monitoring the OpenStack Swift Object Store Using Beanstalk Events

Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Nemanja Vasiljević
January 1, 2022

Acknowledgements

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant apod.).

Contents

Chapter 1

Introduction

In the current world, cloud computing became the most popular way of delivering different types of services throught Internet. One of the most popular cloud service is cloud storage, which allows users to store data in remote locations mentained by third party. Based on how cloud storage manages data, cloud storage can be divided into 3 types: Block storage, File storage and Object storage. Object storage manages data as objects, each object typically includes data itself and some additional informations stored in objects metadata. Since data are stored in remote locations, to which users don't have direct and complete access, some users or external services might want to receive informations about certain events (for example change of content) in storages where their data are located.

Importance of this thesis is to provide event informations to users in OpenIO SDS and OpenStack Swift, which will allow user to react on those events and possibly prevent/detect unwanted actions. Providing event notifications will allow users to have better picture on what is going on in their storage and impove monitoring in these object storages.

There was two attempts[14][16] to solve this issue within OpenStack Swift which were not officially accepted and their solution is outdated. Currently there is no official solution for publishing event notification in OpenStack Swift nor OpenIO Software-Defined Storage (hereinafter SDS).

My interest in this topic steems from its possible impact to extensive amount of users that OpenStack Swift and OpenIO have. Contibuting to open source projects is something that I always wanted to be part of. Possiblity to improve user experience in OpenStack Swift and OpenIO SDS and allow those storages to be even more competative against comercial storages (Amazon, Google, ...) is another reason why I choose this topic.

The goal of this thesis is to create program/middleware which wil publish event notification to user specified destination. One of supported destination will be Beanstalk queue, but program will allow to easily add other types of destinations (for example Kafka) using predefined interface. Proposed program will allow user to specify, using objects metadata (such as name prefix/sufix and object size) and type of event, which event notification should be published. Program will be able to run within OpenStack Swift as well as OpenIO SDS. This thesis will strive to find such solution that could be officially accpeted as part of OpenStack Swift and OpenIO SDS.

Structure of thesis - TODO - there will be probably change in chpaters structure

Chapter 2

Background

This chapter introduces Object storage, its core concepts, and the underlying technologies. After introducing the Object storage, for sufficient understanding of this master thesis topic, it is essential to explain how Software-defined storage manages data, and what types of events can occur inside. The last part of this chapter describes the concept of event notifications, why they are essential, and current interfaces for publishing them to users.

2.1 Object storage

Object storage, also known as *object-based storage (OBS)*, handles data as objects instead of hierarchical methods used in file systems[28]. The object storage is designed to handle data as whole objects, making it an ideal solution for any unchanging data. Data in object stores are changed by replacing objects or files, and therefore object stores are the preferred mechanism for storing such files[29].

2.1.1 Key concepts

Key concepts of object storage are[22]:

- Objects - An object typically consist of user data and metadata uploaded to object storage.
- Containers/Buckets - represents logical abstraction used to provide a data container in object storage. An object with the same name in two different containers represents two different objects. This concept segregates data using bucket ownership and a combination of public and secret keys bound to object storage accounts, allowing users and applications to manipulate with authorized data for specific types of manipulation (read/write/update).
- Metadata - Additional information about data, such as date of creation and last modification, size, hash.
- Access Control Lists(ACLs) - used as primary security construct in object storage, stored in account or bucket level and allows owners to grant permissions for certain operations based on UUID, email, ...
- Object Data protection - two primary data protection schemes in object storage are Replication and Erasure Coding.

Replication is a method used to ensure data resilience. Data are copied into multiple locations/disks/partitions. In case of failure, data are used from a secondary copy to recreate the original copy or as a primary copy.

Erasure coding is a process through which the data is separated into fragments. Then fragments are expanded and encoded with redundant pieces and stored across different storage devices. Erasure coding adds redundancy and allows object storage to tolerate failures.

2.1.2 Object data

With object storage techniques, each object contains[22]:

- **Data** - user-specified data that needs to be stored in persistent storage. Such data can be binary data, text file, image, etc.
- **Metadata** - additional data that describe objects data. Metadata can be divided into two types: Device-managed metadata is additional information maintained by a storage device and used as part of object management in physical storage[28]. The second type is Custom metadata, to which users can store any additional information in key and value pairs. In object storage, metadata is stored together with the object.
- **A universally unique identifier (UUID)** - This ID, created using a hashing process based on object name and other additional information, is assigned to each object in object storage. Using ID object storage systems can tell apart objects from one another. ID is also used to extract data in a system without knowing their physical location/drive and offset.

2.1.3 Access to object storage

Object storage services provide a RESTful interface [31] over HTTP protocol to store and access objects. This approach allows users to create, read, delete, update, or even query objects anytime and anywhere simply by referencing UUID (or using specific attributes for querying), usually with a proper authentication process. The most popular interfaces for communicating with object storages are Amazon S3 (Simple Storage Service) API and OpenStack Swift API.

2.1.4 Pros and cons of object storage

Pros:

- Capable of handling a large amount of unstructured data
- Reduced TCO and cheap COTS - Object storage is designed to utilize cheap COTS(Commercial off-the-shelf) components. As a result Total Cost of Ownership(TCO) is lower than owning homemade Network-Attached Storage(NAS)[29].
- Unlimited scalability - Since object storages are built on distributed systems, they scale very well compared to traditional storages, where they often have an upper limit[21].
- Wide-open metadata - allows users to store custom metadata and the possibility of creating metadata-driven policies, such as compression and tiering.

Cons:

- No in-place update - object must be manipulated as a whole unit.
- No locking mechanism - object storage does not manage object-level locking, and it is up to applications to solve concurrent PUT/GET.
- Slower - this makes object storages a poor choice for applications that need rapid and frequent access to data.

2.2 Software-Defined storage

Software-Defined Storage(SDS) is a storage architecture that separates software storage from hardware allowing greater scalability, flexibility, and control over the data storage infrastructure. With the growth of Software-Defined Networks(SDN) and the need for Software-Defined Infrastructure(SDI), which aims to virtualize network resources and separate control plane from the data plane, this principle was needed to be applied on Object storage as well[24].

To overcome limitations of traditional storage infrastructures, the Software-Defined Storage (SDS) is imposed as a proper solution to simplify data and configuration management while improving the end-to-end control functionality of conventional storage systems[27]. Furthermore, while traditional storages like storage area networks (SAN) and network-attached storage (NAS) provides scalability and reliability, SDS provides it with much lower cost by utilizing industry-standard or x86 system and therefore removing dependency on expensive hardware[19].

2.2.1 Principles

There is no clear definition on criteria for defining software-defined storage, although several fundamental principles can be deducted[23]:

- **Scale-out** - SDS should enable low-cost horizontal scaling (by adding new commodity hardware to existing infrastructure) compared to vertical scaling with more powerful (and expensive) hardware.
- **Customizable** - SDS should offer system storage customization to meet specific storage QoS requirements. This will allow users to choose storage solution based on their requirements/performance and avoid unnecessary overpaying.
- **Automation** - once QoS is defined process of deployment and monitoring on object storage should be automated and done without the need for human resources.
- **Masking** - SDS can mask underlying storage system and distributed system as long as they provide common storage API and meet required QoS. SDS can offer block or File API even though data are saved in object storage (like Ceph does).
- **Police Management** - SDS Software must manage storage according to specified policies and QoS requirements despite being in multi-tenant space. SDS must be capable of handling failures and autoscale in case of change in workloads.

2.2.2 Architecture

As previously described, the main characteristic of SDS is to separate storage functions into control plane and data plane.

Control plane - the control plane represents an abstraction layer with main goal to virtualize storage resources. It offers high-level functions that are needed by the customer to run the business workload and enable optimized, flexible, scalable, and rapid provisioning storage infrastructure capacity. These capabilities span functions like policy automation, analytics and optimization, backup and copy management, security, and integration with the API services, including other cloud provider services[26].

Data plane - the data plane encompasses the infrastructure where data is processed. It consists of all basic storage management functions, such as virtualization, RAID protection, tiering, copy services (remote, local, synchronous, asynchronous, and point-in-time), encryption, compression, and data deduplication that can be requested by the control plane. The data plane is the interface to the hardware infrastructure where the data is stored. It provides a complete range of data access possibilities and spans traditional access methods, such as block I/O (for example, iSCSI), File I/O (NFS, SMB, or Hadoop Distributed File System (HFDS)), and object-storage[26].

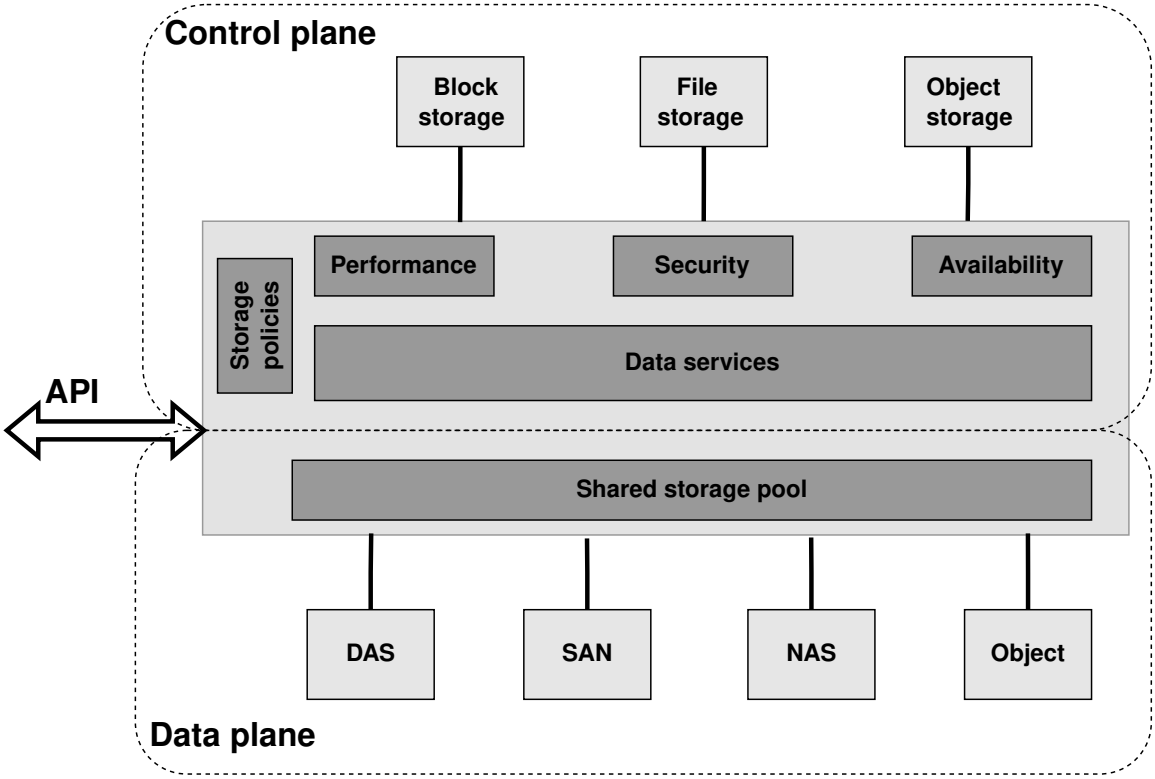

Figure 2.1: SDS data and control plane.[17], remade

2.3 Beanstalk queue

Beanstalk queue or shorter beanstalkd is a fast, simple and lightweight working queue[3]. The primary use case is to manage workflow between different parts of workers of application through working queues and messages. Beanstalkd was developed for the need of Facebook application in order to reduce average response time[3]. Provided by simple protocol design, heavily inspired by Memcached, implemented in programming language C, Beanstalkd offers lean architecture, which allows it to be installed and used very simply, making it perfect for many use cases[13].

2.3.1 Beanstalkd elements

Beanstalkd is a priority queue with server-client architecture. The server represents queues where jobs are saved based on priority. Beanstalkd architecture is composed of several components:

- Jobs - tasks stored by the client
- Tubes - used for storing tasks, each tube contains a ready queue and a delay queue.
- Producer - creates and sends jobs to beanstalkd using command „put“.
- Consumer - process „listening“ on an assigned tube, reserves and consumes jobs from the tube.

2.3.2 Job Lifecycle

Each job is uniquely assigned to one worker at a time. The client creates a job and inserts it to beanstalkd tube using „put“ command. While being in the tube, the job can be in next states[4]:

- **Ready** - the task is free and can be executed immediately by the consumer.
- **Delayed** - the task has assigned delay time that needs to expire before execution. After delay time expires, beanstalkd will automatically change its state to READY.
- **Reserved** - the task is reserved and is being executed by the consumer. Beanstalkd is responsible for checking whether the task is completed in time (TTR - Time to run).
- **Buried** - reserved task, the task will not be removed nor executed until the client decides. This state is often used for further inspection in debugging process when failure or undefined behavior occurs during task execution.
- **Deleted** - the task is deleted from the tube, beanstalkd no longer maintains these jobs.

Figure 2.2 describes the life cycle of a job in beanstalkd tube. Job is created by Producer using put command. Beanstalkd allows the Producer to add delay time before the task is ready for execution, setting the job state to DELAYED. After delay time expires, beanstalkd will automatically change job state to READY. The Producer can specify job priority and jobs with the READY state are stored in the priority queue. A job with the biggest

priority is reserved and executed by a Consumer. After successfully executing the task, the Consumer will delete the job from beanstalkd. If some error occurs, the Consumer can bury the task. The Consumer can decide that he is not interested in completing the reserved task. Using the release command (with optional delay) job state will be changed back to READY (or DELAY). Jobs with the BURIED state will not be touched by beanstalkd server until the client „kicks“ them to READY state.

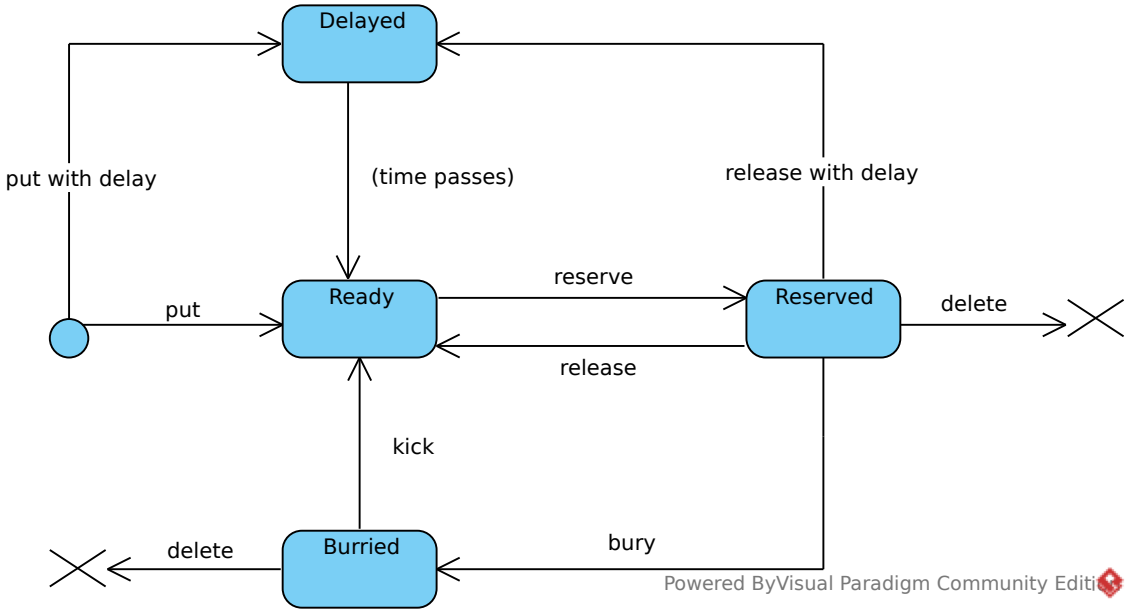

Figure 2.2: State machine diagram of job in Beanstalkd tube

2.3.3 Key characteristics

Key beanstalkd characteristics are:

Asynchronous - beanstalkd allows producers to put jobs in the queue, and workers can process them later.

Distributed - in the same way as Memcached, beanstalkd can be distributed, although this distribution is handled by clients. The beanstalkd server does not know anything about other beanstalkd running instances.

Persistent - beanstalkd offers support for persistent jobs during which all jobs are written to binlog. In case of a power outage, after restarting beanstalkd instance, it will recover jobs content from the logs.

Not secured - beanstalkd is designed to be run in a private/secure network. Therefore it does not support authentication or authorization.

Scalability - beanstalkd can be scaled horizontally, although it must be done on the client side, where each client would connect to multiple servers and then use specific algorithms(e.g., Round-robin) to switch between the different servers.

2.4 Event notifications

An *event* is a runtime operation executed by a software element, representing a significant change or occurrence in a system. Event is created in order to make some information available to other software elements not specified by the operation[30].

Event notification is a message created by a system in orderto notify other parts of the system that an event has taken place[20]. Event notifications are usually used for monitoring and asynchronous job processing.

In object storage, event notifications are used to notify users or tenants about specific changes and occurrences in their bucket or account. Typical event notifications include creating new (or updating existing) objects in the bucket. In addition, most object vendors offer Publish/subscribe notifications, allowing users to subscribe to certain types of event notifications using predefined rules. Information about rules specifying event notifications is usually stored in the upper-level metadata (bucket or account).

2.4.1 CloudEvents

Publishers tend to describe event data differently due to non-existing standards or formats. The lack of a common way to describe events means developers have to learn how to handle events for each event source. To solve this problem, CloudEvents was created.

CloudEvents is a specification for describing event data in common way[6] hosted by Cloud Native Computing Foundation (CNCF)[5]. CloudEvents goal is to dramatically simplify event specification and delivery across services, platforms and beyond. CloudEvents has been integrated by many popular object storage vendors, such as Oracle Cloud, IBM Cloud Code Engine, Azure, Google Cloud etc.

Attributes in CloudEvents specification can be divided into 3 categories:

Required attributes - set of attibures that are required to be included in all events[7]:

- id (string) - event identifier, must not be empty.
- source (URI-reference) - identifies context in which event occured, must not be empty.
- specversion (string) - the version of CloudEvents specification, must not be empty.
- type (string) - value describing the type of occurred event. Often this attribute is used for policy enforcement, routing and monitoring.

Event data attirbutes - attibures containing and describing event data:

- datacontenttype (string) - content type of data value (allows data to carry any type of content).
- dataschema (URI) - identifies the schema that data adheres to.
- data - data payload

Optional attributes :

- time - timestamp
- subject (string) - the subject of the event in the context of the event producer.

- extension attributes - custom attibutes allowing external systems to attach metadata to an event.

```
{
    "specversion" : "1.0",
    "type" : "com.github.pull_request.opened",
    "source" : "https://github.com/cloudevents/spec/pull",
    "subject" : "123",
    "id" : "A234-1234-1234",
    "time" : "2018-04-05T17:31:00Z",
    "comexampleextension1" : "value",
    "comexampleothervalue" : 5,
    "datacontenttype" : "text/xml",
    "data" : "<much wow=\"xml\"/>"
}
```

Listing 2.1: Example of event described using CloudEvents specification and JSON

2.4.2 Amazon S3 event notifications

Amazon Simple Storage Service (S3) is one of the most popular cloud object storages providing a REST web service interface. Amazon S3 is reliable, scalable, commercial and one of the most popular object storage that manages Web-Scale computing by itself[25]. As a result, Amazon S3 has a big impact on object storage and most other object storage vendors crated compatible S3 API for their services.

One of the monitoring features that Amazon S3 provides is Event Notification, which offers users to receive notifications when certain events happen in their S3 bucket. To enable such notifications, users need to create a notification configuration that identifies which events Amazon S3 should publish[2]. Notifications are configured at the bucket level and then applied to each object in the bucket.

Amazon S3 provides limited event destinations to which event notification messages can be send[1]:

- Amazon Simple Notification Service (Amazon SNS) - flexible, fully managed push messaging service, can be used to send messages to mobile phones or distributed services.
- Amazon Simple Queue Service (Amazon SQS) queues - reliable and scalable hosted queues for storing messages as they travel between computers.
- AWS Lambda - serverless, event-driven compute service. Lambda can run custom code in response to the Amazon S3 bucket event (if the lambda function writes to the same bucket that triggers the notification, it can create an execution loop).
- Amazon EventBridge - serverless event bus service used to receive events from AWS. It allows users to define rules to match events and deliver them to defined targets.

By this date, Amazon S3 does not support CloudEvents specification and describes event data in its own way. Some of event types that Amazon S3 can publish are[1]:

Event type	**Desription**
s3:TestEvent	after enabling the event notifications, Amazon S3 publishes a test notification to ensure that topic exist and bucket owner has permissions to publish specified topic.
s3:ObjectCreated:*	An object was created (regardless on operation).
s3:ObjectCreated:Put	An object was created by an HTTP PUT operation.
s3:ObjectCreated:Post	An object was created by HTTP POST operation.
s3:ObjectCreated:Copy	An object was created an S3 copy operation.
s3:ObjectCreated: CompleteMultipartUpload	An object was created by the completion of a S3 multi-part upload.
s3:ObjectRemoved:*	An object was removed (regardless on operation).
s3:ObjectRemoved:Delete	An object was deleted by HTTP DELETE operation.
s3:ObjectRemoved: DeleteMarkerCreated	An versioned object was marked for deletion.

Chapter 3

OpenIO SDS

OpenIO Software-defined storage is open source object storage that is perfectly capable of traditional use cases (such as archiving, big data, cloud). However, at the same time, combined with Grid for Apps (3.3.1), it opens the door for users to create an application that needs much more sophisticated back-end operations. These applications include industrial IoT, machine learning and artificial intelligence, as well as any other applications whose workflow can benefit from automated jobs or tasks[18]. In addition, OpenIO SDS is event-driven storage with the ability to intercept events seamlessly and transparently to the rest of the stack.

3.1 Key characteristics

Hardware agnostic

OpenIO SDS is fully software-defined storage capable of running on x86 or ARM hardware with minimal requirements. Cluster nodes can be different from each other, allowing different generations, types and capacities to be combined without affecting a performance or efficiency[9]. OpenIO has built-in support for heterogeneous hardware allowing every node to be used at its maximum performance.

No SPOF architecture

Every single service used to serve data is redundant from object chunks stored in a disc to the directory level, every information is duplicated. As a result, there is no single point of failure (SPOF) in the cluster and a node can be shutdown without affecting overall availability or integrity[15].

Cluster organization

Instead of a traditional cluster ring-like layout, OpenIO SDS is based on a grid of nodes 3.1. It is flexible and resource-conscious. Compared to other object storage solutions, cluster organization is not based on static data allocation that usually use Chord peer-to-peer distributed hash table algorithm. Instead, OpenIO SDS uses distributed directory for organizing data and metadata hash tables, which allows the software to attain the same level of scalability but with better and more consistent performance[9].

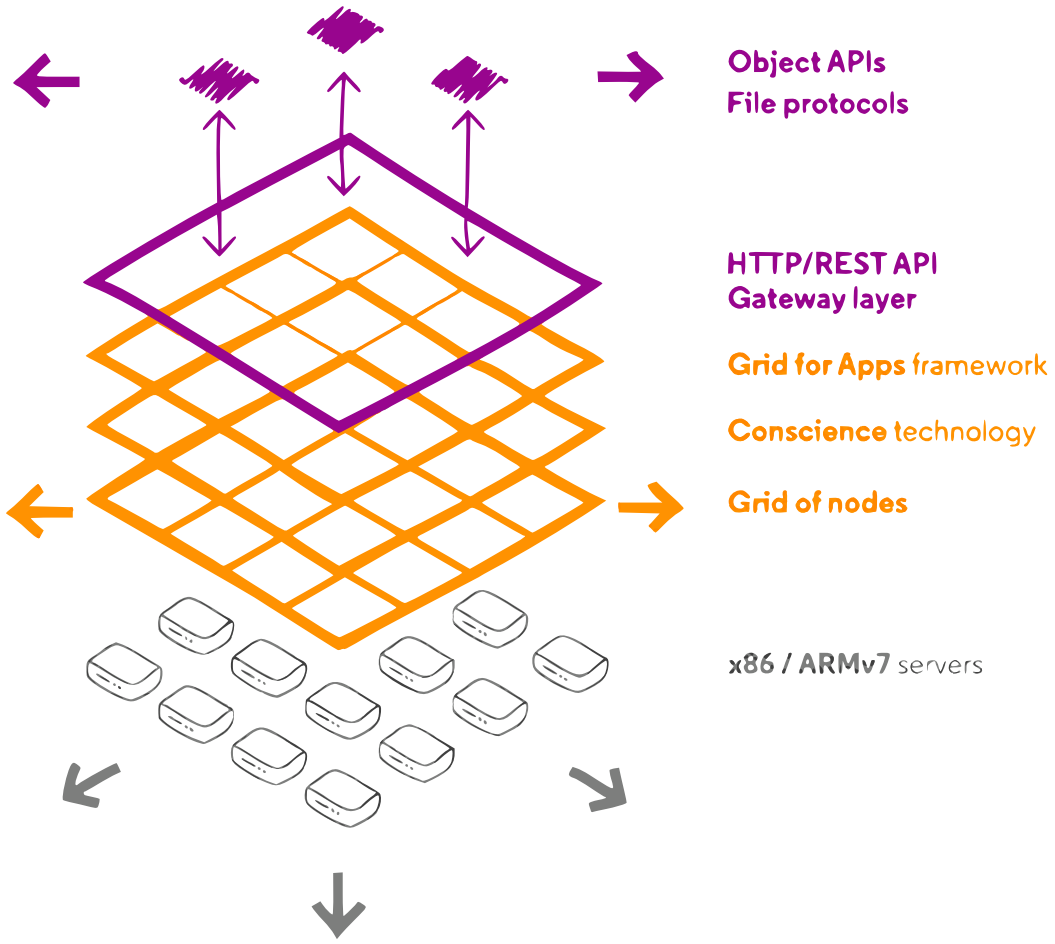

Figure 3.1: Layered view on OpenIO SDS architecture.[12]

Tiering

With tiering, OpenIO SDS offers users to configure a pool containing a group of hardware that can then be used to store specific types of objects. For example, users can create a pool of high-performance hard disks (e.g. SSDs) and use the pool to store objects that require low latency. This feature is realized by a mechanism called storage policies. Multiple storage policies can be defined in one particular namespace. Storage policies can also be used for specifying how many replicas should be created for a specific dataset[15].

3.2 Data organization

Multi-tenancy is one of the core concepts in OpenIO SDS. Data objects are stored within following hierarchy: Namespace/Account/Container/Object 3.2. Multiple namespaces can be configured in each cluster, providing multi-region/zone logical layouts for applications and segregated workloads depending on a tenant or geo-distribution need[10]. There is no classic subdirectory tree. Instead, objects are stored in a flat structure in the container level. However, like many other object storages, there is a way to emulate a filesystem.

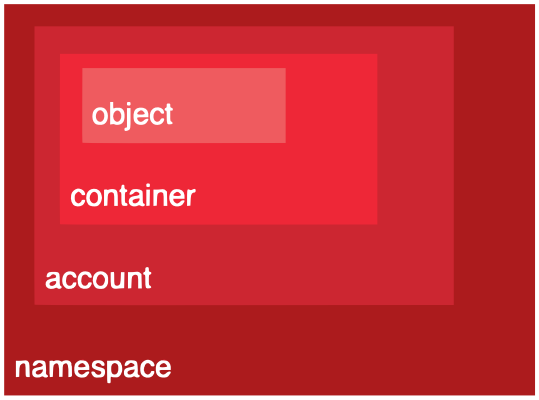

Figure 3.2: Object data organization in OpenIO SDS.[15]

3.2.1 Namespace

A coherent set of network services working together to run OpenIO's solutions. It hosts services and provides operations such as service configuration and monitoring.

3.2.2 Account

An account usually represents a tenant and is the top level of data organization. Each account owns and manages a collection of containers. In addition, the account keeps track of namespace usage for each customer (i.e. bytes occupied by all of a customer's objects)[15].

3.2.3 Container

Container represents an object bucket. Each container belongs to one (and only one) account and is identified by a unique name within the account. The container carries additional information specifying how to manage its objects (e.g. how to secure them)[15].

3.2.4 Object

Object is the smallest data unit visible by a customer and represents a named BLOB with metadata. OpenIO SDS allows several objects to be stored in a container and are considered as versions of the same object. Classic API operations (PUT/GET/DELETE) will be directed towards an object with the latest version. If the size of an object is larger than the specified limit at the namespace level, the object will be divided into chunks of data. This behavior allows capacity optimization as well as distributed reads that could be particularly useful for high-speed video streaming of large media[15].

3.3 Serverless computing

3.3.1 Grid For Apps

Like Amazon AWS Lambda, OpenIO offers an event-driven compute service called Grid for Apps that works on top of OpenIO.

Grif for Apps intercepts all the events that happen in the storage layer, and based on user configuration, triggers specific applications or scripts to act on data (metadata) stored in object storage[18]. The application is executed in cluster nodes and utilizes free unused resources available in the cluster. This improves efficiency (fewer data moving since object data are already available) and saves money (no need for external resources)[18].

Grid for Apps allows customers to perform operations such as metadata enrichment, data indexing and search (e.g. indexing metadata to Elasticsearch), pattern recognition, machine learning, data filtering, monitoring, etc[18].

Grid for Apps in OpenIO is realized using service event-agent and beanstalkd queue.

3.3.2 Event-agent

Event-agent is an OpenIO service responsible for handling asynchronous jobs. It relies on beanstalkd backend to manage jobs. Event-agent key characteristics are[11]:

- Stateless
- CPU intensive
- Must be deployed on every server of the cluster

Every event that occurs in OpenIO is inserted in beanstalkd tube. Event-agent is listening to the beanstalkd tube and consumes jobs from it. Consumers are produced using Eventlet Network Library [8]. The number of workers can be configured.

In event-agent, users can specify handlers for each type of event in event-handler.conf. Some of the event types in OpenIO are storage.content.new, storage.container.deleted, etc.

Events handler is defined as a pipeline containing applications that will react to the event. For example, deleting an object will invoke storage.content.deleted event. Event-agent will handle the event using content_cleaner application, which deletes objects chunks from object storage.

```
[handler:storage.content.deleted]
pipeline = content_cleaner

[handler:storage.content.new]
pipeline = notify

[filter:content_cleaner]
use = egg:oio#content_cleaner

[filter:notify]
use = egg:oio#notify
tube = oio-rebuild
queue_url = ${QUEUE_URL}
```

Listing 3.1: Example of event-agent handler configuration

OpenIO offers users to process events outside of the event-agent. In order to do that, users can use the application „notify“ which will send an event to a specified beanstalkd tube. Then a user can create a custom consumer process that will execute the job from beanstalkd tube. Example of such configuration is displayed in listing 3.1

Chapter 4

OpenStack Swift

Chapter 5

MinIO

Chapter 6

Solution draft

6.1 Current state

6.2 Middleware for OpenStack Swift and OpenIO SDS

6.3 Adapter for MinIO

Chapter 7

Implementation, experiments and assessment

Chapter 8

Conclusion

Bibliography

[1] *Amazon S3 Event notification types and destinations* [online]. [cit. 2021-12-27]. Available at: https://docs.aws.amazon.com/AmazonS3/latest/userguide/notification-how-to-event-types-and-destinations.html.

[2] *Amazon S3 Event Notifications* [online]. [cit. 2021-12-27]. Available at: https://docs.aws.amazon.com/AmazonS3/latest/userguide/NotificationHowTo.html.

[3] *Beanstalkd* [online]. [cit. 2021-12-27]. Available at: https://beanstalkd.github.io/.

[4] *Beanstalkd protocol* [online]. [cit. 2021-12-27]. Available at: https://raw.githubusercontent.com/beanstalkd/beanstalkd/master/doc/protocol.txt.

[5] *Cloud Native Computing Foundation* [online]. [cit. 2021-12-27]. Available at: https://www.cncf.io/.

[6] *CloudEvents* [online]. [cit. 2021-12-27]. Available at: https://cloudevents.io/.

[7] *CloudEvents Specification* [online]. [cit. 2021-12-27]. Available at: https://github.com/cloudevents/spec/blob/v1.0.1/spec.md.

[8] *Eventlet* [online]. [cit. 2021-12-27]. Available at: https://eventlet.net/.

[9] *OpenIO - Key Characteristics* [online]. [cit. 2021-12-27]. Available at: https://docs.openio.io/latest/source/arch-design/overview.html.

[10] *OpenIO SDS: Core Concepts* [online]. [cit. 2021-12-27]. Available at: https://docs.openio.io/latest/source/arch-design/sds_concepts.html.

[11] *OpenIO Services* [online]. [cit. 2021-12-27]. Available at: https://docs.openio.io/latest/source/arch-design/sds_services.html.

[12] *Teratec OpenIO* [online]. [cit. 2021-12-27]. Available at: https://teratec.eu/gb/qui/membres_Openio.html.

[13] *How To Install and Use Beanstalkd Work Queue on a VPS* [online]. 2013 [cit. 2021-12-27]. Available at: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-beanstalkd-work-queue-on-a-vps.

[14] *OpenStack Swift proposed solution 1* [online]. 2015 [cit. 2021-12-27]. Available at: https://review.opendev.org/c/openstack/swift/+/196755.

[15] *OpenIO Core Solution Description*. OpenIO, 2016. Available at: https://www.openio.io/resources/.

[16] *OpenStack Swift proposed solution 2* [online]. 2016 [cit. 2021-12-27]. Available at: https://review.opendev.org/c/openstack/swift/+/388393.

[17] *Software Defined Storage (SDS)* [online]. 2016 [cit. 2021-12-27]. Available at: http://www.sjaaklaan.com/?e=167.

[18] *OpenIO Next-Generation Object Storage and Serverless Computing Explained*. OpenIO, 2018. Available at: https://www.openio.io/wp-content/uploads/2018/03/OpenIO-NextGenObjectStorageAndServerlessComputingExplained.pdf.

[19] *What is software-defined storage?* [online]. 2018 [cit. 2021-12-27]. Available at: https://www.redhat.com/en/topics/data-storage/software-defined-storage.

[20] *What is event-driven architecture?* [online]. 2019 [cit. 2021-12-27]. Available at: https://www.redhat.com/en/topics/integration/what-is-event-driven-architecture.

[21] Amar Kapadia, S. V. *OpenStack Object Storage (Swift) Essentials*. Packt Publishing, 2015. ISBN 978-1-78528-359-8.

[22] Anil Patil, H. S. M. L. R. M. d. S. . *Cloud Object Storage as a Service: IBM Cloud Object Storage from Theory to Practice*. IBM, 2017. ISBN 0738442453.

[23] Chen, Y.-F. R. The Growing Pains of Cloud Storage. *IEEE Internet Computing*. 2015, vol. 19, no. 1, p. 4–7. DOI: 10.1109/MIC.2015.14.

[24] Gracia Tinedo, R., Sampé, J., París, G., Sánchez Artigas, M., García López, P. et al. Software-defined object storage in multi-tenant environments. *Future Generation Computer Systems*. 2019, vol. 99, p. 54–72. DOI: https://doi.org/10.1016/j.future.2019.03.020. ISSN 0167-739X. Available at: https://www.sciencedirect.com/science/article/pii/S0167739X18322167.

[25] Gulabani, S. *Amazon S3 Essentials*. Packt Publishing, 2015. ISBN 9781783554898.

[26] Larry Coyne, E. F. P. G. R. H. C. D. M. A. M. T. P. B. S. C. V. *IBM Software-Defined Storage Guide*. IBM, 2018. ISBN 0738457051. Available at: https://www.redbooks.ibm.com/redpapers/pdfs/redp5121.pdf.

[27] Macedo, R., Paulo, J. a., Pereira, J. and Bessani, A. A Survey and Classification of Software-Defined Storage Systems. New York, NY, USA: Association for Computing Machinery. 2020, vol. 53, no. 3. DOI: 10.1145/3385896. ISSN 0360-0300. Available at: https://doi.org/10.1145/3385896.

[28] Mesnier, M., Ganger, G. and Riedel, E. Object-based storage. *IEEE Communications Magazine*. 2003, vol. 41, no. 8, p. 84–90. DOI: 10.1109/MCOM.2003.1222722.

[29] O'Reilly, J. *Network Storage: Tools and Technologies for Storing Your Company's Data*. Elsevier, 2017. ISBN 9780128038635; 0128038632.

[30] Pethuru Raj, H. S. *Architectural Patterns*. Packt Publishing, 2017. ISBN 9781787287495.

[31] Zheng, Q., Chen, H., Wang, Y., Duan, J. and Huang, Z. *COSBench: A Benchmark Tool for Cloud Object Storage Services*. 2012. 998-999 p. ISBN 978-1-4673-2892-0.