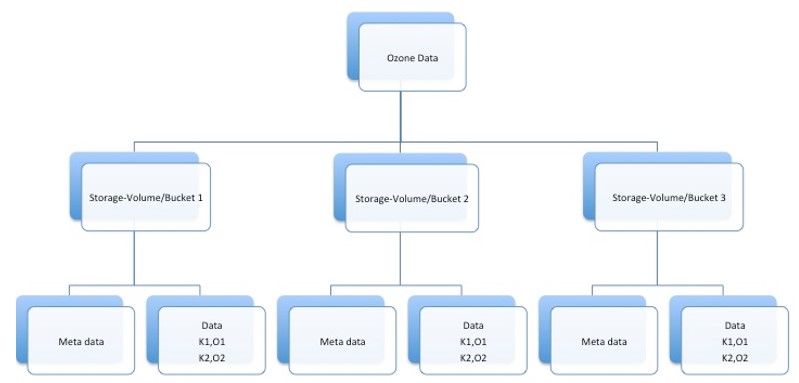
Ozone系统架构

# 1．Ozone的数据模型

Ozone提供了类似AWS S3的key-object键值对象存储，数据的组织如下图所示：



数据层次从下往上依次是：

1. Key/Object

对象存储系统需要存储的数据，Key和Object对象是随机的字节数组，单个key大小期望实现小于1K，但是value对象可以从小到几百字节达到几百兆。

1. Buckets

Key/Object对象被组织成Bucket，Bucket有唯一的Key集合。

1. Storage Volume

Bucket存储在Storage-Volume中，Storage Volume全局唯一，其对于Buckets和存储的数据量有配额上的限制。在私有云中，可以被用来创建给用户（类似于Linux中的Home目录）、项目及租户，管理员为分配的Storage-Volume分配配额；在公有云中，可以为云账户创建Storage Volume。

Bucket使用storage-volumeName/bucketName两部分组成唯一标识，Object使Storage-

VolumeName/bucketName/ObjectKey来组成唯一名字。

Ozone类似于Azure Blob Storage(WASB)，但是Ozone使用Storage Volume替代WASB的账户概念。要实现Ozone对象存储系统需要提供下功能（Basic Requirements）：

1. 管理员创建Storage Volume
2. 提供Bucket创建和删除操作，每个Buckt的URL唯一，且不能被重命名。只有Storage Volume的所属者或者租可以创建Volume的Bucket
3. 可以从Storage Volume中查出Buckets列表
4. 根据给定的key在bucket中创建或者删除Object对象。对象的数据以流式的形式传送到Ozone Service中。当Object对象写满后，只允许读不保证对象的局部写操作
5. 可以查询Buckets中的数据内容
6. 可以创建、更新及删除Bucket的ACL

对于存储数据及存储实体而言需要以下要求:

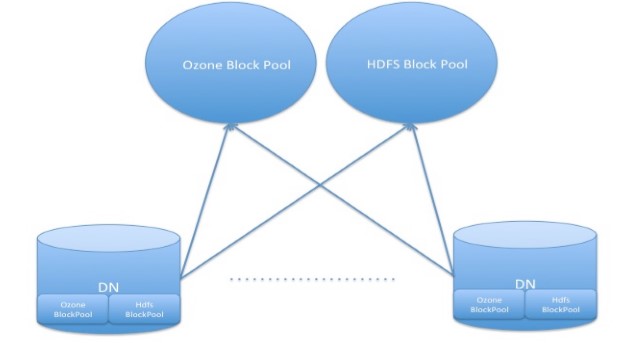
1. Storage Volume Name，长度大概在3-64bytes大小
2. Bucket Name，与Storage Volume要求相同
3. Key Sizes: 1KB
4. Object Size: 5G
5. 系统总Buckt数量：1000w
6. 单Bucket 对象数量：100w
7. 单Storage Volume中的Bucket数：1000

Ozone的metadata包括：

1. Storage Volume Level metadata: Storage Volume的名称和所属者
2. Bucket Level Metadata: Owner, ACL,全局唯一的Bucket名，Bucket ID

# 2.Hign Level Design

Ozone与HDFS共享Datanode数据存储，DataNodes同时在HDFS和Ozone中存储数据，Ozone的数据使用独立的BlockPool（具有Block Pool ID），在该BlockPool中可以设置多独立的Ozone namespace。在DN中可以存储多个HDFS和多Ozone Block pools。如下图所示：



## 2.1 Storage Container容器

Storage Container是用来存储Ozone数据（bucket）和元数据的存储单元。Storage Container与HDFS block一样，共同存在DataNode上。但是与HDFS不同的是，Ozone没有一个类似于NameNode的中心节点来维护元数据。Ozone的元数据以分布式的方式存在不同的Storage Containers中。Storage Container类似于HDFS Block以Whole的形式保持副本，这些副本的操作具有强一致性。Storage Container的存储大小决定于副本的复制能力和结点恢复能力，存储大小是可配置的，但是至少要比Object的允许的最大值大。

一个Bucket有百万数量级的Object，总数据量可以达到TB，远远大于一个Storage Container，因此可以将一个Bucket分成多partition分区，每个分区存储在一个Container中（Storage Container可以容纳最大数量的partition，但是对象只能来自一个Bucket，一个Object完全存在于单一的Container中）。

Storage Container由Datanode实现（用户如果只需要使用HDFS的Block功能，可以通过配置禁用该功能），Storage Container的语义和要求如下：

1. Bucket元数据

Bucket元数据数据单元KB级别，每个Storage Container可以保存上百万Bucket的元数据。

需要有update操作，因为buckets的ACL可以更新(要保证ACL更新的原子性)

满足基本的get/put API

在Container中可以执行列举操作来查看所有的bucket信息

1. Bucket数据

Object数据量可以从几百KB到几百兆

通过Object Key来访问数据（Container中存储的数据要包含存储对象的索引）

支持单Object的读写 Streaming API

不支持Object数据的append及in-place upate数据操作

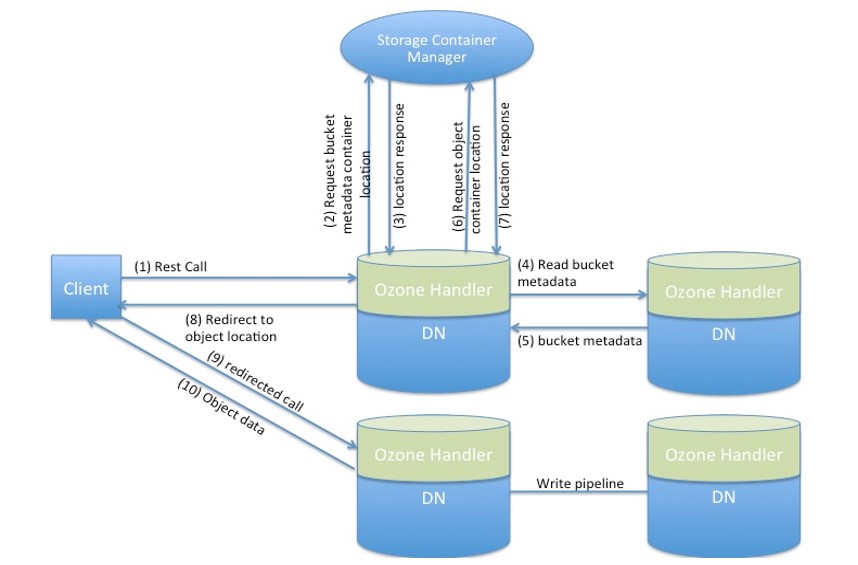
## 2.2 DataNode中Storage Container

### 2.2.1 Storage Container标识符

storage container使用唯一标识符，类似于HDFS的BlockID，并不包括实际的container的位置。Object的key会映射到storage container标识符，Storage Container Manager使用该标识符来定位包含该Storage Container的Datanode。类似，Bucket name也会映射到包含buckets元数据的该Storage Container标识符。Storage Container identifier和HDFS Block ID一样是64bit。

### 2.2.2 Storage Container Service的调用流程

下图为Storage Container和Ozone Handler的调用流程图，主要包含Storage Container Service服务和Ozone Handler处理器：

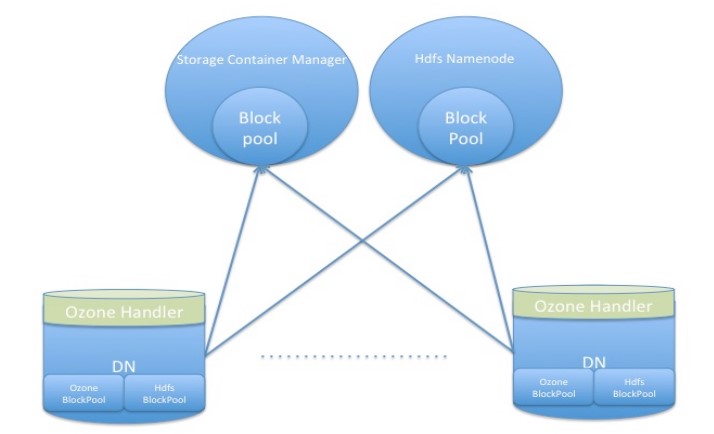


1. Ozone Handler in the Datanode

Ozone Handler是Datanode上所允许的Ozone模块组件，Handler上包含了一个Http Server并实现了Ozone REST方式API。Ozone Handler与Storage Container Manager的管理器交互来查询Container的位置，与Datanode中的Storage Container交互来实现多种操作。当不需要Ozone功能时，可以禁用该功能组件。

1. Storage Container Manager

SCM比较类似于HDFS上的block管理功能，其从各个DataNode中收集心跳，并跟踪每个Storage Container的位置。其维护Storage Container的映射关系，以前缀匹配的方式去查询Storage Container。在DataNode中，同时管理HDFS和Ozone数据，Ozone的数据使用单独一个blockPool-ID（由SCM提供）。由于在HDFS上运行，尽量复用NameNode的block 代码来实现container的管理，这也意味着使用Datanode上的block pool service。如下图所示：



# 3.Ozone的实现

**1）Mapping Object-Key to a Storage Container**

推荐使用hash Partitioning Scheme实现Object-key到Storage Container的映射。Object经过Hash计算后，使用bucketID作为前缀，构成的value以前缀匹配的方式映射到Storage Container。当Container的容量扩大一定程度后进行split，使用extendiable hash实现container的re-map。Storage container manager管理器会存储这些映射关系到前缀树中以此实现高效的前缀匹配.每个bucket都有属于自己的前缀树。

1. **Range Partitioning vs Hash Paritioning**

在对象存储中，Range分区是另外一项受欢迎的对key进行分区的技术，但是偏向于哈希分区的方法，原因如下：

Range分区需要Range索引，索引同样非常大而且需要分布式存储，这复杂化了分区的分割操作，需要额外的索引更新。

Range分区在搜索过程中引进一种额外的跳跃，第一次Range索引就需要去读。

Range分区同样有热点问题，相近name的key会造成想用范围的访问并发量太大。

然而，Range分区提供了有序访问以及有序列表展示的优势，但是对于一个对象存储而言，一个有序的访问并不是很重要的需求，列表有序的展示可以在后面的阶段通过二级索引来实现。

1. **Mapping a bucket to a storage Container**

Bucket的元数据同样存在storage container中，使用bucket name作为key计算哈希计算。多个bucket的元数据可以存储在storage container中，存储元数据的container中使用专门定义的bucketid(例如0)来作为container id的前缀。

1. **Storage Container Requirements**

Storage Container存储在各个DataNode上.对于Storage Container我们有以下的要求:

* Storage Container能够可靠地被复制.
* Storage Container保持严格的一致性.
* Storage Container对于内部存储的对象能提供高效的键值对的查询方式.
* Storage Container必须能够以流式的方式对object进行读写操作.
* Storage Container必须能够支持get/put接口,来存储和更新bucket元数据.
* Storage Container对于内部存储的bucket必须能提供一个原子的更新操作.
* Storage Container能够进行分裂当他们达到一定大小限制的时候.

1. **对于Storage Container的实现**

* 尽可能的复用HDFS现有的block pool管理的功能定，将storage Container作为Block类的扩展类，hdfs block由标识符，生成时间记录和大小组成。这3个属性同样应用在storage container中
* 为了保证一致性和持久性，storage container实现了少量的原子性和持久化操作，比如事务。Container对这些操作提供了可靠的保证，在第一阶段中，实现以下事务操作：

Commit: 这个操作促进了对象从被写到最后的确认结束，一旦操作成功，这个container就会保证对象可读

Put：这个操作使用小规模的写操作，如元数据的写操作

Delete:删除对象

* Container中每个事物都会有一个事务ID，并且必须被持久化
* 为storage container实现新的data pipeline，要求不同类型的更新和恢复语义操作。
* 在考虑使用leveldb jni作为storage container的原型设计，leveldbjni满足对storage键值对的存储需求

1. **Data Pipeline Consistency**

Data Pipeline管道链流式复制副本数据到container中，container的副本类似于HDFS的Block的副本标记，每个pipeline副本标记记录在pipeline创建的时候会被更新，所以任何旧的container会被撤销。HDFS在Block恢复的时候额外使用block length副本长度来判断副本是否已经更新到最新。类似storage container用事务ID来判断container副本是最新的。

# 4.Ozone API

**1) Cluster Level APIs**

* PUT StorageVolumes
  + API - PUT /admin/volume/{StorageVolume}
    - 创建一个storage volume
    - 只有管理员才允许调用这个操作
* HEAD StorageVolume
  + API - HEAD /admin/volume/{StorageVolume}
    - 检测Storage Volume是否存在
    - 只有管理员才能调用这个操作
* GET
  + 列出集群中所有的Storage Volume.
* DELETE Storage Volumes
  + API - DELETE /amin/volume/{StorageVolume}
    - 删除一个volume如果他是空的

**2)Storage Volume层级的API**

* GET Buckets
* API - GET /
  + 利用用户的认证信息登录
  + 返回请求发送者所拥有的buckets列表
* Get User Buckets
  + API - GET /admin/user/userid
    - 利用用户认证信息进行登录, 如果他/她有权限阅读其他用户的信息,返回那个用户所独有的buckets列表信息.

1. **Bucket Level API**

* LIST objects in a bucket
  + API - GET/{bucketName}
    - 返回buckets中最多1000个数量的对象key
* PUT bucket
  + API - PUT/{bucketName}
    - 为请求发送者创建bucket
  + GET/PUT Bucket ACL
    - API - /{Bucket}?acl
    - 允许用户获取/设置bucket的ACL
* HEAD bucket
  + API - HEAD/{bucketName}
  + 检查bucket是否存在,前提是请求发送者有权限访问此bucket
* DELETE bucket
  + API - DELETE/{bucketName}
    - 删除bucket如果此bucket为空的话

1. **Object Level APIs**

* GET object
  + API - GET/{bucketName}/{key}
    - 返回给定key所代表的对象值,如果这个值存在
    - 在第一阶段暂不支持ACL,因此只有所属用户才能读写自身的bucket.
* PUT object
  + API - PUT/{bucketName}/{key}
    - 在bucket中创建一个对象
    - 在第一阶段暂不支持ACL,因此只有所属用户才能读写自身的bucket.
    - 不支持局部的上传,只有对象全部上传成功了才被认为是一次成功的操作.
* HEAD object
  + API - HEAD/{bucketName}/{key}
    - 检测对象是否存在
    - 在第一阶段只有bucket的所属用户才能调用此操作
* DELETE object
  + API - DELETE/{bucketName}/{key}
    - 删除对象

参考链接：

http://blog.csdn.net/androidlushangderen/article/details/50767774

https://issues.apache.org/jira/browse/HDFS-7240