AWS S3

Ravindra Kudache

Outline

- **Concepts**
- **How to Access S3**
- **Scalability**
- **Availability**
- **Versioning**
- **Consistency**

http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl

Concepts of S3 – Bucket

- Container for objects stored in S3
- Unlimited size
- Organize the namespace at the highest level
- Internet accessible storage via HTTP/HTTPS
- Global unique name

From: http://techblogsearch.com/a/amazon-announced-new-s3-usability-enhancements.html

http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl

Concepts of S3 – Object

- Similar to files.
- No hierarchy.
- Objects are immutable.
- Size up to 5 TB
- Uniquely identified within a bucket by a key(name) and a version ID

http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl

Concepts of S3 – Key

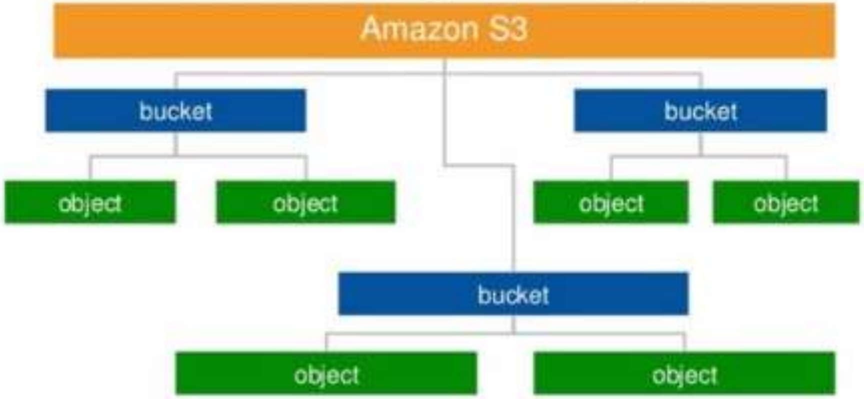
- **Name of an object**
- **Unique identifier for an object within a bucket**
- **Use the object key to retrieve the object**

Concepts of S3 – Namespace

Globally Unique

↓

Bucket Name + Object Name (key)

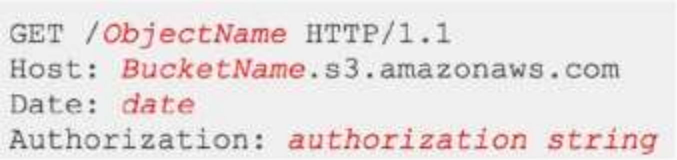


How to Access S3 – CLI & REST

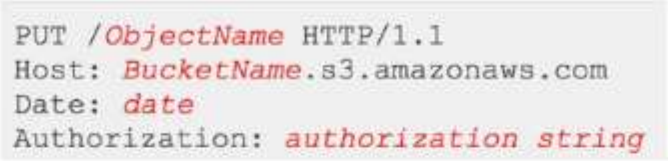
- **AWS Command**
  - ✦ **Put Object**

    aws s3api put-object bucket key
  - ✦ **Get Object**

    aws s3api get-object bucket key outfile
- **REST API**
  - ✦ **Put Object**
  - ✦ **Get Object**



```
GET /ObjectName HTTP/1.1
Host: BucketName.s3.amazonaws.com
Date: date
Authorization: authorization string
```



```
PUT /ObjectName HTTP/1.1
Host: BucketName.s3.amazonaws.com
Date: date
Authorization: authorization string
```

How to Access S3 – AWS SDK

SKD for Java

- **Get Object**

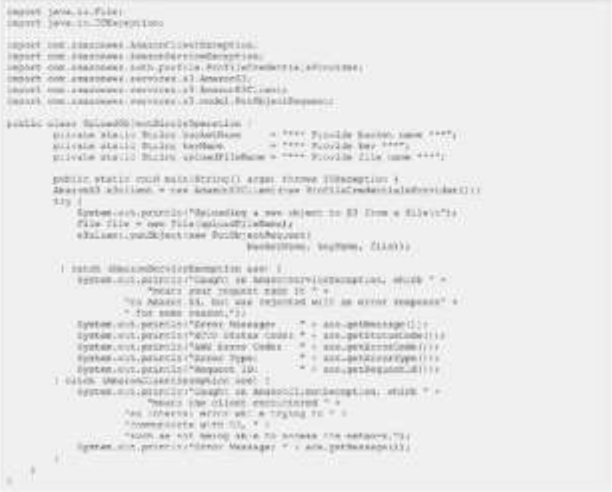
AmazonS3 s3Client = new AmazonS3Client(new ProfileCredentialsProvider());

S3Object object = s3Client.getObject(new GetObjectRequest(bucketName, key));

- **Put Object**

AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());

s3client.putObject(new PutObjectRequest(bucketName, keyName, file));



```
import java.io.File;
import java.io.IOException;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.PutObjectRequest;

public class UploadObjectSingleOperation {
	private static String bucketName     = "*** Provide bucket name ***";
	private static String keyName        = "*** Provide key ***";
	private static String uploadFileName = "*** Provide file name ***";

	public static void main(String[] args) throws IOException {
	AmazonS3 s3client = new AmazonS3Client(new ProfileCredentialsProvider());
	try {
	    System.out.println("Uploading a new object to S3 from a file\n");
	    File file = new File(uploadFileName);
	    s3client.putObject(new PutObjectRequest(
	                             bucketName, keyName, file));

	 } catch (AmazonServiceException ase) {
	    System.out.println("Caught an AmazonServiceException, which " +
	              "means your request made it " +
	          "to Amazon S3, but was rejected with an error response" +
	          " for some reason.");
	    System.out.println("Error Message:    " + ase.getMessage());
	    System.out.println("HTTP Status Code: " + ase.getStatusCode());
	    System.out.println("AWS Error Code:   " + ase.getErrorCode());
	    System.out.println("Error Type:       " + ase.getErrorType());
	    System.out.println("Request ID:       " + ase.getRequestId());
	} catch (AmazonClientException ace) {
	    System.out.println("Caught an AmazonClientException, which " +
	              "means the client encountered " +
	          "an internal error while trying to " +
	          "communicate with S3, " +
	          "such as not being able to access the network.");
	    System.out.println("Error Message: " + ace.getMessage());
	}
    }
}
```

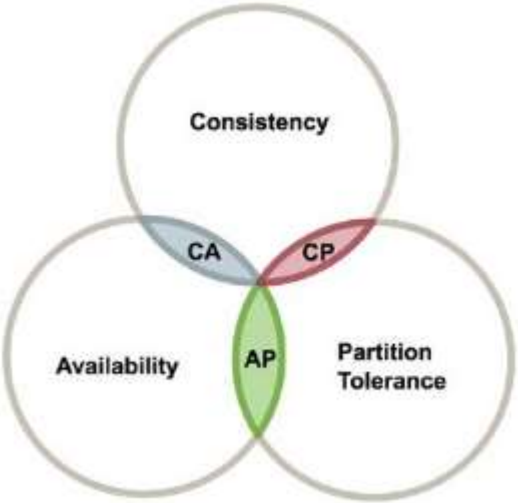
From: http://docs.aws.amazon.com/AmazonS3/latest/dev/RetrievingObjectUsingJava.html

CAP Theorem

In the presence of a network partition, strong consistency and high availability can not be achieved at the same time

Amazon S3 (AP)

- **High availability**
- **Eventual consistency**
- **Read-after-write consistency**

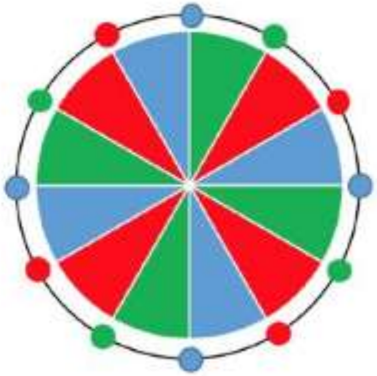


From: http://berb.github.io/diploma-thesis/original/061_challenge.html

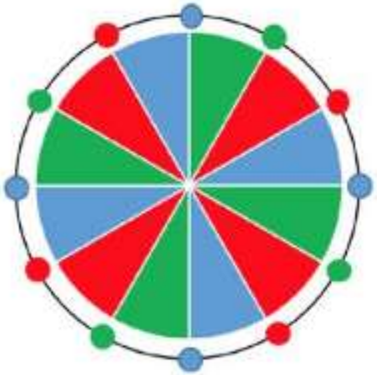
Scalability – Consistent Hashing

Avoid re-hashing everything

- **Object key mapped to a point on the edge of ring**
- **Map machine to many random points on the edge of ring**
- **Machine responsible for arc before its nodes**
- **If a machine joins, it adds new nodes**
- **If a machine leaves, its range moves to after machine**



Availability – Replication



Pick n next nodes (different machines!)
n -- Replication factor

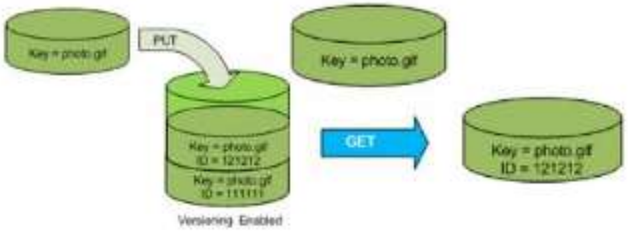
If one node fails, the other nodes still have the key information

Versioning – Basics

- **keep multiple versions of an object in one bucket**
- **protect objects from unintended overwrites and deletions**
- **retrieve previous versions of them**
- **Write faster**

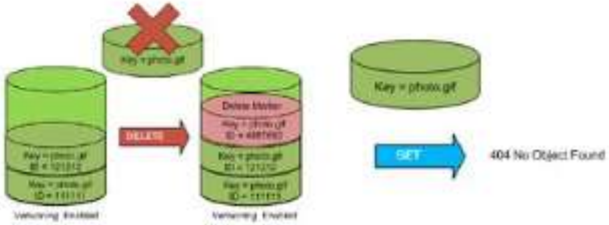
- PUT doesn't overwrite: pushes version

- GET returns most recent version



– DELETE doesn't wipe

– GET will return not found

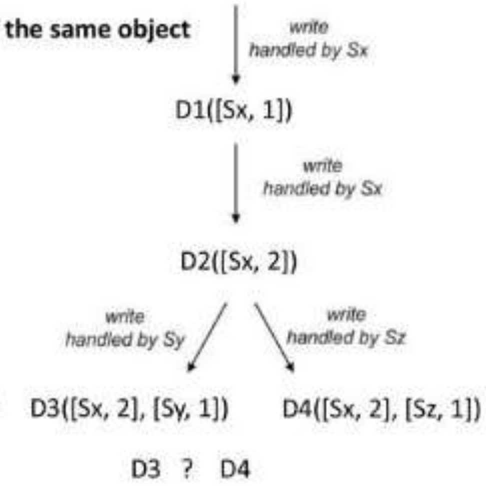


Versioning

Using vector clock to capture causality between different versions of the same object

--D: object
--Sx,Sy,Sz: nodes
--[Sx, 1]: vector clock [node, count]

- **One vector clock is associate with every version of every object**
- **Determine the versions of an object by examining their vector clocks**
- **If the counters on D1's clock is less than or equal to all of the nodes in the second clock, D1 is ancestor of the second, can be forgotten**
- **Otherwise, the two changes require reconciliation**



write handled by Sx

D1([Sx, 1])

write handled by Sx

D2([Sx, 2])

write handled by Sy

write handled by Sz

D3([Sx, 2], [Sy, 1])

D4([Sx, 2], [Sz, 1])

D3 ? D4

Version evolution of an object over time

Consistency

Read-after-write consistency

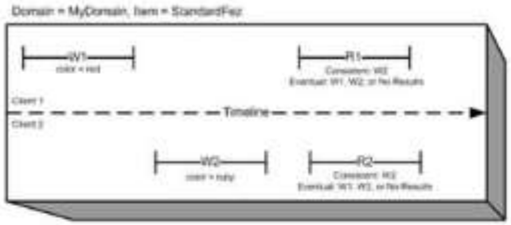
- PUTS of new objects in S3 bucket
- Guarantee visibility of new data to all applications
- Allow you to retrieve objects immediately after creation

Eventual consistency

- Overwrite PUTS and DELETES of objects in S3 bucket
- Weak consistency, to achieve high availability
- If no additional updates are made to a given data item, all reads to that item will eventually return the same value.

Updates to a single key are atomic

Eventual Consistency


Domain = MyDomain, Item = StandardFez
W1
color = red
R1
Consistent: W2
Eventual: W1, W2, or No Results
Client 1
Timeline
Client 2
W2
color = ruby
R2
Consistent: W2
Eventual: W1, W2, or No Results

From: http://docs.aws.amazon.com/AmazonS3/latest/dev/Introduction.html

Both W1 (write 1) and W2 (write 2) complete before the start of R1 (read 1) and R2 (read 2)

- **Consistent read**

  R1 and R2 both return color = ruby.

- **Eventually consistent read**

  R1 and R2 might return color = red, color = ruby, or no results, depending on the amount of time that has elapsed.

Consistency Protocol

A consistency protocol similar to those used in quorum system is used to maintain consistency among its replicas.

- **This protocol has two key configurable values: R and W.**
    - **R is the minimum number of nodes that must participate in a successful read operation.**
    - **W is the minimum number of nodes that must participate in a successful write operation.**
- **N is the total number of nodes. Setting R and W.**
- **R+W ? N**

- END