Cloud storage with Apache jclouds

Andrew Gaul

19 November 2014

http://jclouds.apache.org/
http://gaul.org/

Overview

- What is Apache jclouds
- What is object storage
- Basic concepts
- How to put and get objects
- Advanced topics

What is Apache jclouds

- Interface with public and private cloud services via Java
- Provider-specific implementations, e.g., Amazon S3, OpenStack Swift
- Cross-provider, portable abstractions, e.g., BlobStore
- Supports all major providers

Why object storage

- Scale out to billions of objects and petabytes of data
- Lower cost, higher performance, better reliability
- Applications and tenants can share one object store
- Many competitive public and private cloud offerings

Why not object storage

- Must rewrite existing applications against new interfaces
- Distributed systems introduce complexity and additional failure modes
- Lacks features that databases and filesystems have

What is object storage

- Distributed system of storage nodes
- Supports narrow interface:
    - Create and delete containers
    - Put, get, and delete objects
    - List objects in a container
    - Modify and retrieve metadata

Example applications

- Store photos and videos for a social networking site
- Archive large data sets, e.g., scientific measurements

Basic concepts

- APIs and Providers
- Regions
- Containers
- Objects

APIs and Providers (1)

- API defines how to communicate with a cloud
- Provider specializes a given API, e.g., regions
- Support major APIs: Atmos, Azure, Google, S3, Swift
- Support many providers: Glacier, HP, Rackspace

APIs and Providers (2)

amazon web services™

Google Cloud Platform

hp Cloud Services

Windows Azure

RACKSPACE® CLOUD FILES™

DreamObjects

EMC²

SCALITY

Cleversafe

SwiftStack

Cloudian

openstack™

basho

ceph

AMPLIDATA

Regions

- Most public providers offer multiple geographic regions
- Usually associate a container with a given region
- Some providers can replicate between regions automatically, others offer a per-object cross-region copy

Containers

- Containers can contain many named objects
- Some providers have partial directory support
- Some providers allow public-read access containers

Objects

- Objects have a name, stream of bytes, and metadata map
- Must write entire stream of bytes but can read ranges
- Metadata map includes standard HTTP headers, e.g., Content-Type: text/plain and arbitrary user headers, e.g., x-amz-meta-foo: bar

Example put and get

```java
try (BlobStoreContext context = ContextBuilder
        .newBuilder("transient")
        .credentials("identity", "credential")
        .buildView(BlobStoreContext.class)) {
    BlobStore blobStore = context.getBlobStore();
    ByteSource payload = ByteSource.wrap(
        new byte[] { 1, 2, 3, 4 });
    Blob blob = blobStore.blobBuilder(blobName)
        .payload(payload)
        .contentLength(payload.size())
        .build();
    blobStore.putBlob(containerName, blob);

    blob = blobStore.getBlob(
        containerName, blobName);
    try (InputStream is =
            blob.getPayload().openStream()) {
        ByteStreams.copy(is, os);
    }
}
```

Consistency (1)

- Some blobstores have weak semantics and can return stale data for some time
- putBlob(name, data1), putBlob(name, data2), getBlob(name) can return data1 or data2

Consistency (2)

- Applications may need retry loops or other fallback logic
- S3 (standard region) and Swift have various forms of eventual consistency
- Atmos, Azure, and GCS have strong consistency, more like a file system

Data integrity (1)

- Provider guarantees object integrity at-rest via content hash
- Application can guarantee object integrity on-the-wire by providing and verifying Content-MD5 header
- Guava has useful helpers

Data integrity (2)

- Guarantee integrity during putBlob:

```
ByteSource payload = ByteSource.wrap(
        new byte[] { 1, 2, 3, 4 });
Blob blob = blobStore.blobBuilder(blobName)
        .payload(payload)
        .contentLength(payload.size())
        .contentMD5(payload.hash(Hashing.md5()))
        .build();
blobStore.putBlob(containerName, blob);
```

- This reads InputStream twice

Data integrity (3)

- Guarantee integrity during getBlob:

```
Blob blob = blobStore.getBlob(containerName, blob);
HashCode md5 = HashCode.fromBytes(blob.getMetadata()
    .getContentMetadata().getContentMD5());
try (HashingInputStream his = new HashingInputStream(
        blob.getPayload().openStream()),
        Hashing.md5()) {
    ByteStreams.copy(his, os);
    if (md5.equals(his.hash())) {
        throw new IOException();
    }
}
```

Large object sizes

- Some objects may not fit in memory or in a byte[]
- Non-repeatable payloads, e.g., InputStream
- Repeatable payloads, e.g., Guava ByteSource, can retry after network failure

Multi-part upload

- Some objects are too large for a single put operation, e.g., AWS S3 > 5 GB, Azure > 64 MB
- jclouds abstracts these details via:

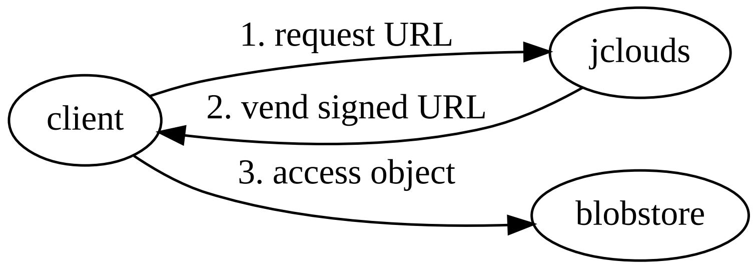
```
Blob blob = ...
blobStore.putBlob(containerName, blob,
    new PutOptions().multipart());
```

Many objects

- Amazon S3 supports an arbitrary number of objects per container
- Swift recommends max 500,000 objects per container
- Atmos recommends max 65,536 objects per directory
- Solution: shard over multiple containers or directories

URL signing (1)

- Most object stores allow creating a URL which your application can vend to external clients



URL signing (2)

- URL is cryptographically signed; allows time-limited access to a single object
- Clients interact directly with the object store, removing your application as the bottleneck

Amazon Glacier

- Optimized for storage, not retrieval
- Read requests can take several hours to complete
- Least expensive public cloud provider, often cheaper than private cloud

Local blobstores

- In-memory (transient) appropriate for unit testing
- Filesystem has production uses with limited number of objects or quantity of data
- Remote clients require running a HTTP server and implementing authorization

jclouds-cli

- Command-line tool useful for administrative and debugging tasks:

```
jclouds blobstore container-list \
    --provider aws-s3 \
    --identity $IDENTITY \
    --credential $CREDENTIAL
jclouds blobstore write \
    --provider aws-s3 \
    --identity $IDENTITY \
    --credential $CREDENTIAL \
    $CONTAINER_NAME $OBJECT_NAME $FILE_NAME
```

References

- https://github.com/jclouds/jclouds-examples
- https://github.com/andrewgaul/s3prox
- http://gaul.org/object-store-comparison/

Thank you!

http://jclouds.apache.org/
http://gaul.org/