Keeping up a competitive Ceph RGW/S3 API
Who?

- Javier
- Computer Engineer and Software Developer
- Cloud and Virtualization team @ Igalia

@javimunhoz
linkedin.com/in/javiermunhoz
This talk

- Understanding upstream code investments in RGW/S3
- Going upstream with new features in RGW/S3
- Remarks
Understanding upstream code investments in RGW/S3
Press release

Amazon Web Services Launches

March 14, 2006 at 3:00 AM EST

S3 Provides Application Programming Interface for Highly Scalable Reliable, Low-Latency Storage at Very Low Costs

SEATTLE—(BUSINESS WIRE)—March 14, 2006—Amazon Web Services today announced "Amazon S3(TM)," a simple storage service that offers software developers a highly scalable, reliable, and low-latency data storage infrastructure at very low costs. Amazon S3 is available today at http://aws.amazon.com/s3.

Amazon S3 is storage for the Internet. It's designed to make web-scale computing easier for developers. Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. It gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites. The service aims to maximize benefits of scale and to pass those benefits on to developers.

Amazon S3 Functionality

Amazon S3 is intentionally built with a minimal feature set. The focus is on simplicity and robustness.
A first contact with AWS S3 (~2008)

- About two years after S3 was launched
- More than one year before Apple launched the iPad
- Litl webbook project
  - https://informationart.com/projects/litl
  - update system
  - global storage backend based on S3
AWS re:Invent 2017: Deep Dive on Amazon S3 & Amazon Glacier Storage Management with (STG311)
WEB LOG ANALYSIS

Amazon Web Services provides services and infrastructure to build modern, scalable, and highly available web applications. Amazon’s scalable computing and storage infrastructure allows these applications to generate huge amounts of log information.

This data can be an important source of knowledge for any company that is operating web applications. Analyzing logs can reveal information such as traffic patterns, user behavior, marketing profiles, etc.

However, as the web application grows and the number of visitors increases, storing and analyzing web logs becomes increasingly slow and unscalable.

This diagram shows how to use Amazon Web Services to build a scalable and flexible large-scale log analysis platform. The core component of this architecture is Amazon Elastic MapReduce, a web service that enables analysts to process large amounts of data easily and cost-effectively using a Hadoop-based framework.

The web front-end servers are running an Amazon Elastic Compute Cloud (Amazon EC2) instances.

Amazon CloudFront is a content delivery network that helps improve website performance and user experience on the web. This service also generates valuable log information.

Log files are ultimately stored in Amazon Simple Storage Service (Amazon S3), a highly available and scalable data store. Data is sent in parallel from multiple web servers or edge locations.

An Amazon Elastic MapReduce cluster processes the data set. Amazon Elastic MapReduce utilizes a hosted Hadoop framework, which processes the data in a parallel job.

When Amazon EC2 has unused capacity, it offers EC2 instances at a reduced cost, called the Spot Price. This price fluctuates based on availability and demand. If your workload is flexible in terms of time of completion or required capacity, you can dynamically expand the capacity and significantly reduce the cost of running your jobs.

Data processing results are pushed back to a relational database using tools like Apache Hive. The database can be an Amazon Relational Database Service (Amazon RDS) database. Amazon RDS is a fully managed database service that makes it easy for you to set up, operate, and scale a relational database in the cloud.

Like many services, Amazon RDS instances are provisioned on a pay-as-you-go model. After analysis, the data can be automatically deleted when no longer needed or manually deleted from your RDS instance snapshot, and then terminated. The database can then be recreated from the snapshot whenever needed.

Reasons to adopt RGW/S3

- Costs
- Flexibility (Open)
- Security and Privacy
- Geopolitics
- ...

...
Users and companies investing in RGW/S3

- Base storage consumers
- Cheap and massive storage consumers
- On-line storage providers (IaaS providers)
Base storage consumers

- S3 compatibility
- S3 feature coverage
Base storage consumers

- S3 compatibility
- S3 feature coverage

rgw s3 auth aws4 force boto2 compat = false

(2016)
Base storage consumers

- S3 compatibility
- S3 feature coverage

AWS Signature Version 4 core support (2016/03)
AWS Signature Version 4 chunked upload (2016/08)
AWS Signature Version 4 presigned url compatibility fix (2016/12)
...

Cheap and massive storage consumers

- S3 storage management UX
- Integration with product and services
Cheap and massive storage consumers

- S3 storage management UX
- Integration with product and services

Archive zone (2018-2019)
Cheap and massive storage consumers

- S3 storage management UX
- Integration with product and services

Ansible AWS S3 core module support (2016)
On-line storage providers (IaaS providers)

- Updated documentation and examples
- SDK, libs and client tooling support
- New S3 features and extensions
On-line storage providers (IaaS providers)

- Updated documentation and examples
- SDK, libs and client tooling support
- New S3 features and extensions

**Browser-Based Uploads Using POST (AWS Signature Version 4)** (2017)

The Amazon S3 feature documentation is available [here](#). It describes how users upload content to Amazon S3 by using their browsers via authenticated HTTP POST requests and HTML forms.

Those HTML forms consist of a form declaration and form fields. The form declaration contains high-level information about the request and the form fields contain detailed request information.

The technical details to craft a S3 HTML form are available [here](#). The HTML form also requires a proper POST policy (have a look [here](#) to create a POST policy).

The process for sending browser-based POST requests is as follows:

1. Create a security policy specifying conditions restricting what you want to allow in the request.
2. Create a signature that is based on the policy. For authenticated requests, the form must include a valid signature and the policy.
3. Create a HTML form that your users can access in order to upload objects to your Amazon S3 bucket directly.
On-line storage providers (IaaS providers)

- Updated documentation and examples
- SDK, libs and client tooling support
- New S3 features and extensions

(2016)
from libcloud.storage.types import Provider
from libcloud.storage.providers import get_driver
import libcloud

api_key = 'api_key'
secret_key = 'secret_key'

cls = get_driver(Provider.S3_RGW_OUTSCALE)
driver = cls(api_key, secret_key, region='eu-west-1')
container = driver.get_container(...
On-line storage providers (IaaS providers)

- Updated documentation and examples
- SDK, libs and client tooling support
- New S3 features and extensions

Requester Pays Bucket (2016)
Requester Pays Bucket

Virtual error buckets
Going upstream with new features
Going upstream with new features

- Complexity
- Risk
- Communication
Going upstream with new features

1) Understand the Problem
   1) Clarify the problem
   2) Research similar options
   3) Model the system and break the problem into pieces

2) Come up with a Plan
   1) Prioritize your work
   2) Map out your intended approach

3) Implement the Plan

4) Verify your Results
Going upstream with new features

- AWS Signature Version 4
  - S3 request authentication algorithm
  - Benefits over AWSv2
    - Verification of the requester via access key ID and secret access key
    - Request tampering prevention while the request is in transit
    - Replay attacks protection within 15 minutes of the timestamp in the request
  - Strategic feature
Going upstream with new features

- AWS Signature Version 4
  - Is it a real problem?
  - Anyone working on it?
  - Interest in going upstream?
  - What is the use case?

1) Understand the Problem
   1) Clarify the problem
   2) Research similar options
   3) Model the system and break the problem into pieces

2) Come up with a Plan
   1) Prioritize your work
   2) Map out your intended approach

3) Implement the Plan

4) Verify your Results
Going upstream with new features

- **AWS Signature Version 4**
  - Reference implementation available?
  - Other open implementations?
  - API spec, developer guide, etc.
  - Official client side examples

1) Understand the Problem
   1) Clarify the problem
   2) Research similar options
   3) Model the system and break the problem into pieces

2) Come up with a Plan
   1) Prioritize your work
   2) Map out your intended approach

3) Implement the Plan

4) Verify your Results
Going upstream with new features

- **AWS Signature Version 4**

1) **Understand the Problem**
   1) Clarify the problem
   2) Research similar options
   3) Model the system and break the problem into pieces

2) **Come up with a Plan**
   1) Prioritize your work
   2) Map out your intended approach

3) **Implement the Plan**

4) **Verify your Results**
Going upstream with new features

- AWS Signature Version 4
  - Break down for auth method
    - HTTP Authorization header
      - Transfer payload in a single chunk
      - Transfer payload in multiple chunks
    - Query string parameters
  - Test cases

1) Understand the Problem
   1) Clarify the problem
   2) Research similar options
   3) Model the system and break the problem into pieces

2) Come up with a Plan
   1) Prioritize your work
   2) Map out your intended approach

3) Implement the Plan
4) Verify your Results
Going upstream with new features

- AWS Signature Version 4
  - Break down for auth method
    - HTTP Authorization header
      - Transfer payload in a single chunk (1)
      - Transfer payload in multiple chunks (5)
    - Query string parameters (3)
  - Test cases (2) (4) (6)
Going upstream with new features

- AWS Signature Version 4
- Share the Plan
  1) HTTP Auth header (single chunk) + test cases
  2) Query string parameters + test cases
  3) HTTP Auth header (multiple chunks) + test cases

1) Understand the Problem
   1) Clarify the problem
   2) Research similar options
   3) Model the system and break the problem into pieces

2) Come up with a Plan
   1) Prioritize your work
   2) Map out your intended approach

3) Implement the Plan
4) Verify your Results
Going upstream with new features

- AWS Signature Version 4
  - Start with a simple skeleton
  - Implement specific and minimum functionality

1) Understand the Problem
   1) Clarify the problem
   2) Research similar options
   3) Model the system and break the problem into pieces

2) Come up with a Plan
   1) Prioritize your work
   2) Map out your intended approach

3) Implement the Plan
4) Verify your Results
Requests authenticated with the Query Parameters are treated as unsigned ... */
if (is_v4_payload_unsigned(exp_payload_hash) || is_v4_payload_empty(s) || is_non_s3_op) {
    return { ... };}
} else {
  /* We're going to handle a signed payload ... */
  if (!is_v4_payload_streamed(exp_payload_hash)) {
    ldpp_dout(s, 10) << "delaying v4 auth" << endl;
  /* payload in a single chunk */
  switch (s->op_type)
  {
    case RGW_OP_CREATE_BUCKET:
    case RGW_OP_PUT_OBJ:
      [...]
    case RGW_OP_PUBSUB_NOTIF_CREATE:
      break;
    default:
      dout(10) << "ERROR: AWS4 completion for this operation NOT IMPLEMENTED" << endl;
      throw -ERR_NOT_IMPLEMENTED;
  }
  } else {
    dout(10) << "body content detected in multiple chunks" << endl;
  /* payload in multiple chunks */
  switch(s->op_type)
  {
    case RGW_OP_PUT_OBJ:
      break;
    default:
      dout(10) << "ERROR: AWS4 completion for this operation NOT IMPLEMENTED (streaming mode)" << endl;
      throw -ERR_NOT_IMPLEMENTED;
    }
  dout(10) << "aws4 seed signature ok... delaying v4 auth" << endl;
  } [...]
  return { ... };}
}
Going upstream with new features

- AWS Signature Version 4
  - Things work as expected?
  - AWS S3 official clients work?
  - Implement test cases
  - Useful logging

1) Understand the Problem
   1) Clarify the problem
   2) Research similar options
   3) Model the system and break the problem into pieces

2) Come up with a Plan
   1) Prioritize your work
   2) Map out your intended approach

3) Implement the Plan
4) Verify your Results
Credential=HSMFile15LVRTPRBA=0/HC/20195903/us=cost-1/s3/aws4_request,
SignedHeaders=host;x-amz-content-sha256;x-amz-date
Signature=c55b41b0e13411e95026cd41397a13d45b9c95c3a01927319251b3856eab0f6

15:06:11.310 7f87047f27a0 1 ===== starting new request req=0x7f87047ed430 =====
15:06:11.310 7f87047f27a0 2 req 1550 0.000s initializing for trans_id = tx0000...

op=26R3WListBuckets_ObJStore_S3
15:06:11.310 7f87047f27a0 10 req 1550 0.000s s0:list_buckets verifying requester

15:06:11.310 7f87047f27a0 10 canonical request hash = elf753c3f...
15:06:11.310 7f87047f27a0 10 string to sign = Aws4-HMAC-SHA256

date_k = a6c02c594e61520b8a22054e1b755556eb4215d4a984270587b1005a8aaca9e83
region_k = 0a0e12c6e950cde56e956027ef0cea9a851819a332bfa13c358df30a9fc
service_k = 56a3064ca6500fe33cfbdc2b2a2e69c950feede0b2c1e55f30c5e9b17056
signing_k = 953c3917f353bca5a08782f8d4328ac222c573ac

15:06:11.326 7f87047f27a0 10 generated signature = c55b41b0e13411e95026cd41397a13d45b9c95c3a01927319251b3856eab0f6

req 1550 0.016s s0:list_buckets server signature=c55b41b0e...
req 1550 0.016s s0:list_buckets client signature=c55b41b0e...
req 1550 0.016s s0:list_buckets same=0
req 1550 0.016s s0:list_buckets rgw::auth::s3::LocalEngine granted access
req 1550 0.016s s0:list_buckets rgw::auth::s3::AWSAuthStrategy granted access
```python
# AWS4 specific tests
#

def check_aws4_support():
    if 'S3_USE_SIGV4' not in os.environ:
        raise SkipTest

[...]

@tag('auth_aws4')
nose.with_setup(teardown=_clear_custom_headers)
@attr(resource='object')
@attr(method='put')
@attr(operation='create w/invalid authorization')
@attr(assertion='fails 400')
def test_object_create_bad_authorization_invalid_aws4():
    check_aws4_support()
    key = _setup_bad_object({'Authorization': 'AWS4-HMAC-SHA256 Credential=HAHAHA'})
    e = assert_raises(boto.exception.S3ResponseError, key.set_contents_from_string, 'bar')
    eq(e.status, 400)
    eq(e.reason.lower(), 'bad request')  # some proxies vary the case
    assert e.error_code in ('AuthorizationHeaderMalformed', 'InvalidArgument')

[...]
```

https://github.com/ceph/s3-tests/blob/master/s3tests/functional/test_headers.py
Remarks

- A competitive S3 API responds to real market needs
- The value and impact of the S3 API is not well understood
- Users and companies are willing to invest in high value upstream contributions in RGW/S3
- A systematic and flexible process for investment in upstream contributions seems to work well for all parties
- New features and specific extensions are key to adopt RGW/S3 in new business contexts
Thanks!