Rucio & Cloud Storage

Mario Lassnig

with content from Fernando, Rob, Johannes, James, Mihai, Cedric, Alba, Tobi

5th Rucio Community Workshop
10 - 11 November 2022, Lancaster UK
https://indico.cern.ch/event/1185600/

Cloud storage?

In recent years there has been **significant work** done **integrating Rucio with cloud storage**

Two major angles to consider when discussing clouds

Technical	Access tools, transfer protocols, monitoring, authn/z, accounting, billing, storage, …
Organisational	Deployed on-site or off-site Centralised or distributed Open or closed source software Public (institute, laboratory, …) or commercial In-kind contribution or paid service

It can get **complicated quickly,** e.g. …

Self-hosted MinIO S3 server on a CERN data centre VM using a centrally managed CephFS volume
WebDAV portal to self-hosted Nextcloud on a commercial hoster which points to free-tier AWS S3 storage
Experiment collaborates with commercial cloud provider and gets free storage with S3v4 protocol support

From a Rucio point of view, cloud storage is **storage that requires URL-based signatures**

Putting CephFS on top of RADOS requires some sort of storage system on top	-> *grid-style* storage
Putting Ceph Object Gateway S3 API on top of RADOS	-> cloud storage

Rucio credential mechanism

For namespace (*listing replicas*) and storage operations (*rucio upload/download*)

- Generate URL signatures **at the time of execution** of the command
- URL signatures are **generated server-side** by the Rucio server
  - No deployment of secrets necessary to clients
- The account must have **schema permission** (`perm_get_signed_url`) and **account attribute** (`sign_url`)
- The RSE must have several configurations applied

```
scheme      https
impl        rucio.rse.protocols.gfal.NoRename
attributes  sign_url: s3 | gcs | swift    verify_checksum: False        s3_url_style: path
            skip_upload_stat: True        strict_copy: True
```

Credential secrets configuration

- For S3 and SWIFT compatible interfaces (e.g. MinIO, Amazon, Ceph S3 Gateway), requires an entry in `rse-account.cfg`
- For Google Cloud Storage requires the JSON credential file from Google Cloud Console

```
"d87c29b7e3294df5eacc154effd99bae": {
    "access_key": "...",
    "secret_key": "...",
    "signature_version": "s3v4",
    "region": "us-west-2"
},
```

```
{
  "type": "service_account",
  "project_id": "rucio-test",
  "private_key_id": "be5e4aa2a0fc07e672d4051d8582c45fc630bc77",
  "private_key": "-----BEGIN PRIVATE KEY------...",
  "client_email": "rucio-test@rucio-test.iam.gserviceaccount.com",
  "client_id": "123456",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://accounts.google.com/o/oauth2/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/rucio-test%40rucio-test.iam.gserviceaccount.com"
}
```

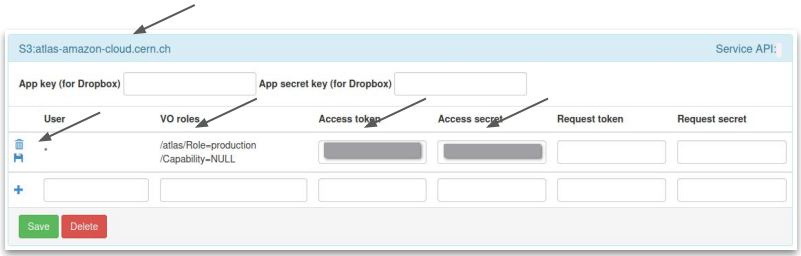
FTS credential mechanism

When adding rules for third-party-copy, the URL signatures are generated by FTS when needed

We don't know how long transfer jobs will be in the **queue of FTS**

URL signatures are **time-limited**

You cannot TPC from cloud storage to cloud storage



Credentials need to be inserted in FTS configuration

Secrets *:8446/config/cloud_storage*

Insert entry in specific format

GFAL Configuration *:8449/fts3/ftsmon/#/config/gfal2*

Cannot be edited directly, has to be set by FTS admin

```
[S3:ATLAS-SEAL-CLOUD.CERN.CH]
ALTERNATE=true
REGION=dummy
```

HTTP Configuration *:8449/fts3/ftsmon/#/config/http_plugin.so*

Cannot be edited directly, has to be set by FTS admin

```
# GCloud related options
[GCLOUD]
JSON_AUTH_FILE=/etc/fts3/gcloud_atlas.json
```

Integration with commercial clouds 1/2

Google Cloud Storage

- First integration R&D project incredibly painful
  - Shoehorn X.509 certificates into commercial cloud
  - Friendly administrators at sites
  - CERN-provided certificate injected into Google loadbalancer
- Custom proxy rules to accommodate our typical Tier-1 storage setup

Frontend

Protocol	IP:Port	Certificate	SSL Policy	Network Tier
HTTPS	[redacted]	atlas-google-europe-west1-cern-provided	GCP default	Premium

Routing rules

Hosts	Paths	Backend
All unmatched (default)	All unmatched (default)	atlas-europe-west1-litter
*	/*	atlas-europe-west1-litter
*	/atlasscratchdisk/*	atlas-europe-west1-scratchdisk
*	/atlasdatadisk/*	atlas-europe-west1-datadisk

Name	Created	Location type	Location	Default storage class	Last modified	Public access
atlas-europe-west1-datadisk	Sep 21, 2022, 8:26:54 AM	Region	europe-west1	Standard	Nov 1, 2022, 3:35:01 PM	Not public
atlas-europe-west1-litter	Sep 21, 2022, 10:47:42 AM	Region	europe-west1	Archive	Sep 21, 2022, 10:47:42 AM	Not public
atlas-europe-west1-scratchdisk	Sep 21, 2022, 8:27:21 AM	Region	europe-west1	Standard	Sep 21, 2022, 8:27:21 AM	Not public

SEAL Storage Technology

- Distributed cloud storage, offered 10PB of storage to ATLAS for a long-term R&D project
- Integration went relatively smooth with standard URL signature mechanism
- Same trick used for integration: SEAL administrators injected CERN-provided certificate in their loadbalancer

Integration with commercial clouds 2/2

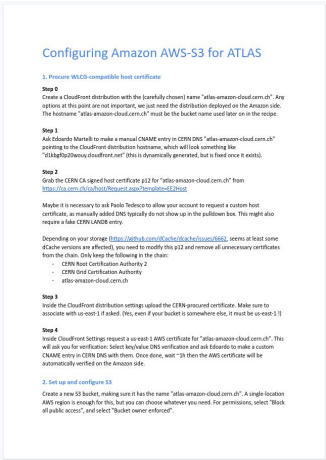
Now… Amazon

This is where it gets complicated

It worked out of the box (thanks to DigiCert) until they changed to their own custom CA

In ATLAS there's a US Tier-3 (FRESNO) with a considerably sized investment

Setting this up was… challenging: 6+ months of trial & error lead to this short document


Configuring Amazon AWS-S3 for ATLAS

1. Procure WLCG-compatible host certificate

Step 0
Create a CloudFront distribution with the (carefully chosen) name "atlas-amazon-cloud.cern.ch". Any options at this point are not important, we just need the distribution deployed on the Amazon side. The hostname "atlas-amazon-cloud.cern.ch" must be the bucket name used later on in the recipe.

Step 1
Ask Edoardo Martelli to make a manual CNAME entry in CERN DNS "atlas-amazon-cloud.cern.ch" pointing to the CloudFront distribution hostname, which will look something like "d1kbqf8p20wouy.cloudfront.net" (this is dynamically generated, but is fixed once it exists).

Step 2
Grab the CERN CA signed host certificate p12 for "atlas-amazon-cloud.cern.ch" from https://cs.cern.ch/cs/host/Request.aspx?template=EE2Host

Maybe it is necessary to ask Paolo Tedesco to allow your account to request a custom host certificate, as manually added DNS typically do not show up in the pulldown box. This might also require a fake CERN LANDB entry.

Depending on your storage (https://github.com/dCache/dcache/issues/6602, seems at least some dCache versions are affected), you need to modify this p12 and remove all unnecessary certificates from the chain. Only keep the following in the chain:

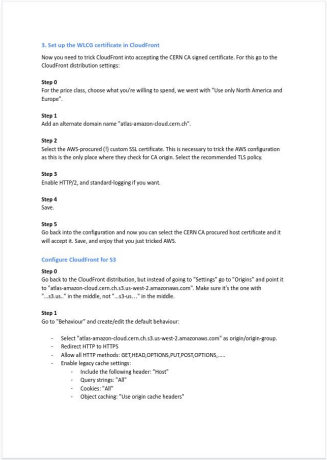
- CERN Root Certification Authority 2
- CERN Grid Certification Authority
- atlas-amazon-cloud.cern.ch

Step 3
Inside the CloudFront distribution settings upload the CERN-procured certificate. Make sure to associate with us-east-1 if asked. (Yes, even if your bucket is somewhere else, it must be us-east-1.)

Step 4
Inside CloudFront Settings request a us-east-1 AWS certificate for "atlas-amazon-cloud.cern.ch". This will ask you for verification: Select key/value DNS verification and ask Edoardo to make a custom CNAME entry in CERN DNS with them. Once done, wait ~1h then the AWS certificate will be automatically verified on the Amazon side.

2. Set up and configure S3

Create a new S3 bucket, making sure it has the name "atlas-amazon-cloud.cern.ch". A single-location AWS region is enough for this, but you can choose whatever you need. For permissions, select "Block all public access", and select "Bucket owner enforced".


3. Set up the WLCG certificate in CloudFront

Now you need to trick CloudFront into accepting the CERN CA signed certificate. For this go to the CloudFront distribution settings:

Step 0
For the price class, choose what you're willing to spend, we went with "Use only North America and Europe".

Step 1
Add an alternate domain name "atlas-amazon-cloud.cern.ch".

Step 2
Select the AWS-procured (!) custom SSL certificate. This is necessary to trick the AWS configuration as this is the only place where they check for CA origin. Select the recommended TLS policy.

Step 3
Enable HTTP/2, and standard-logging if you want.

Step 4
Save.

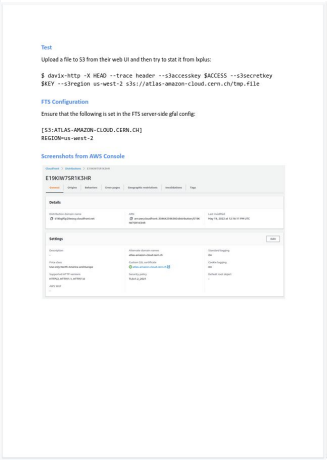
Step 5
Go back into the configuration and now you can select the CERN CA procured host certificate and it will accept it. Save, and enjoy that you just tricked AWS.

Configure CloudFront for S3

Step 0
Go back to the CloudFront distribution, but instead of going to "Settings" go to "Origins" and point it to "atlas-amazon-cloud.cern.ch.s3.us-west-2.amazonaws.com". Make sure it's the one with "...s3.us..." in the middle, not "...s3-us..." in the middle.

Step 1
Go to "Behaviour" and create/edit the default behaviour:

- Select "atlas-amazon-cloud.cern.ch.s3.us-west-2.amazonaws.com" as origin/origin-group.
- Redirect HTTP to HTTPS
- Allow all HTTP methods: GET,HEAD,OPTIONS,PUT,POST,OPTIONS,....
- Enable legacy cache settings:
  - Include the following header: "Host"
  - Query strings: "All"
  - Cookies: "All"
  - Object caching: "Use origin cache headers"


Test

Upload a file to S3 from their web UI and then try to stat it from lxplus:

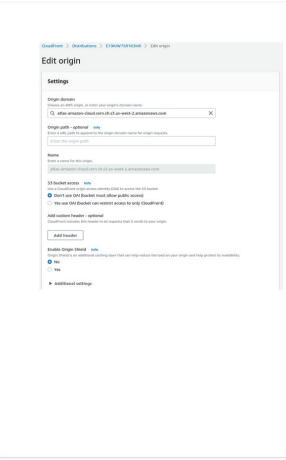
```
$ davix-http -X HEAD --trace header --s3accesskey $ACCESS --s3secretkey
$KEY --s3region us-west-2 s3://atlas-amazon-cloud.cern.ch/tmp.file
```

FTS Configuration

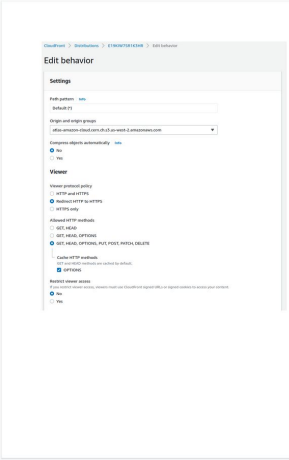
Ensure that the following is set in the FTS server-side gfal config:

```
[S3:ATLAS-AMAZON-CLOUD.CERN.CH]
REGION=us-west-2
```

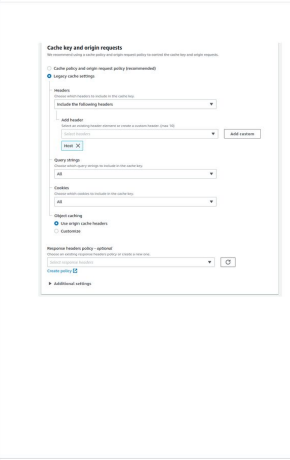
Screenshots from AWS Console


Edit origin

Settings


Edit behavior

Settings


Cache key and origin requests

Rucio ROOT Direct-IO mechanism

For interactive analysis and other processing cases, do not download file but do remote reads

The path returned from list-replicas usually can be fed straight into `TFile::Open()`

```
TFile::Open("https://mycloud:443/file.root?url_signature=1234");
```

S3 protocol **does not provide multi-range** byte requests
Amazon required CloudFront CDN anyway, which does **multi-range translation**
Others, e.g., Google Cloud Storage or MinIO, do not have this translation layer

Workaround is simply to disable multi-range requests through Davix

Have to append URL options to emulate: `#multirange=false&nconnections=30`

This is highly client dependent, *one size fits all* maybe not really applicable
We will have to investigate if we should simply make Rucio reply with these options
Would require a potential hint to list-replicas (`--use-for-direct-io=30`) or similar solution

Future work

Configuration / Setup

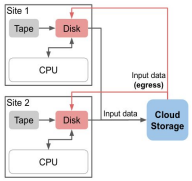
- Complicated, but grew organically from the ongoing Cloud R&D projects
- Needs a complete overhaul: esp. naming of attributes

Already identified features that we will need for production-level integration

- Access control right now is all-or-nothing, needs to be more fine grained
- Smarter peering mechanism
  - Static multihop distance config vs. dynamic cloud regions
  - The concept of cloud regions in Rucio is missing completely
- Security considerations
  - Right now completely dependent on X.509 with DNS-injection trick
  - Clouds typically support OpenID/OAuth2 flows, should be helpful for token migration work
- Throughput and cost control not yet implemented, if you have the access rights you get the "full cloud power"
- Bucket-copy transfertool, no need to go through FTS for this
- Cloud boosting option: Dynamically spend currency for extra throughput/storage
- Data lifetime considerations / different cloud QoS costs

Theoretical R&D studies: Simulation and evaluation of cloud storage caching (Tobi's PhD)

- Temporary cloud bursting to improve workflows needing tape recalls
- Demonstrates 15% improvement in job times



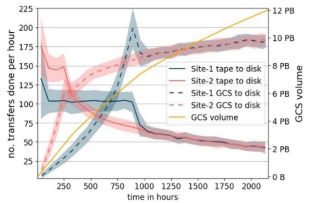


Fig. 8 The solid blue and red line show the number of transfers from tape to disk per hour for each site. The dashed lines show the number of transfers from GCS to disk per hour for each site. The orange line shows the GCS volume used.