Simple Storage Service (S3)

@ Journée AuDACES, juin 2024

Antoine Mahul antoine.mahul@uca.fr

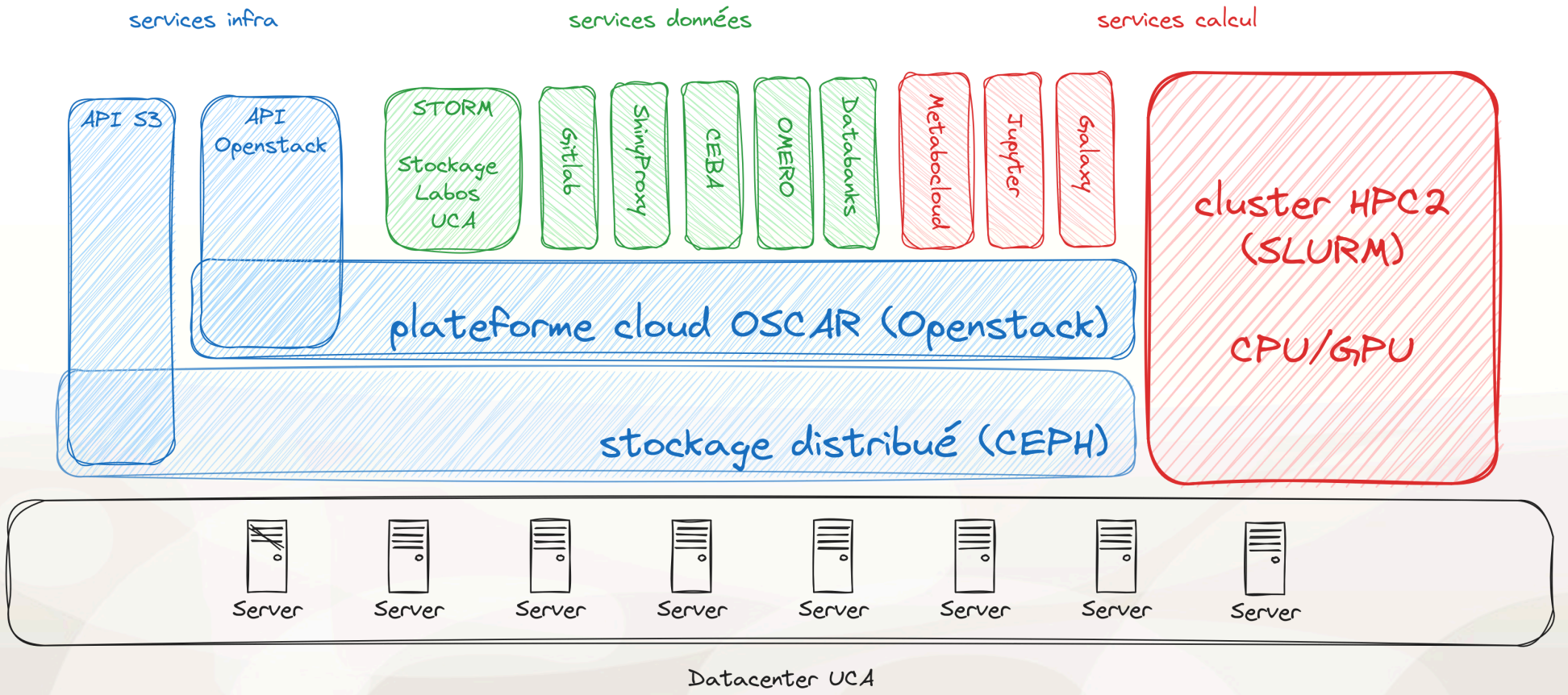
MÉSOCENTRE

UNIVERSITÉ
Clermont Auvergne

Le Mésocentre Clermont-Auvergne

- structuration en 2014
- intégré à la DOSI en 2017: Pôle d'Appui à la Recherche & Mésocentre
- appui au calcul scientifique, au stockage et traitement des données de recherche
- ressources informatiques mutualisées dans le datacenter de l'UCA
  - HPC2: cluster + partition GPU
  - stockage CEPH (CephFS / S3)
  - OSCAR: cloud IaaS (VM à la demande, K8s)
- ressources humaines mutualisées:
  - 3 x Mésocentre
  - 3 x plateforme IFB Auvergne Bioinformatique (AuBI)
  - 1 x projet transverse Cloud Environnemental (CEBA)
  - 1 x Cellule Science Ouverte UCA
  - 1 poste ASR **vacant** CDD 2 ans pour iRODS

Panorama des services



CEPH

- logiciel Open Source
- initié par Sage Weil lors de sa thèse (Université of Califormia, Santa Cruz)
- developpé par ~~InkTank~~, ~~Redhat~~, IBM
- Ceph : middleware de stockage distribué
  - croissance horizontale (*scale out*)
  - réplication / erasure coding selon un *domain failure*
  - algorithme de placement des données déterministe
  - aucun point de défaillance unique
  - accès en mode bloc: RBD (disques des VM dans Openstack)
  - accès en mode fichier : CephFS (utilisé pour STORM)
  - accès en mode objet API S3: RadosGW

Infra CEPH mésocentre

- 3 racks dans le DC UCA
- 6 contrôleurs
- 23 noeuds de stockage (+6 en cours d'installation)
- 374 disques HDD & SSD (OSD)
- SSD (mode bloc): 219 TB bruts (+278 TB)
- HDD (modes fichier & objet): 3.5 PB (+2.3 PiB)
- politique de réplication : 1 réplicat par rack.
- en production depuis 2015

API S3 (Simple Storage Service)

> *Object storage, often referred to as object-based storage, is a data storage architecture ideal for storing, archiving, backing up and managing high volumes of static unstructured data—reliably, efficiently and affordably.* (IBM)

- API REST (RESTful web services)
  - une URL = une ressource
  - un verbe HTTP (POST, GET, PUT, DELETE) == une action (create, read, update, delete)
- définie par Amazon en 2006 pour le service de stockage AWS S3
- devenue de fait l'API de référence du stockage objet (~~standard~~)
- 2 types de ressources
  - *bucket* : un conteneur d'objets (espace de nommage, unité de gestion et d'indexation)
  - *object*: id + donnée + métadonnée (~ *blob*)

API S3 pour quoi faire ?

- backend de stockage primaire pour certains logiciels (Gitlab, Galaxy, iRods...)
- stockage secondaire pour certains softs de backup (restic, commvault)
- stockage de grands volumes de données
- support natif dans les bibliothèques/outils de traitement de données

Clients S3

- Clients en ligne de commande:
  - **aws cli**
  - s3cmd / s5cmd / rclone
  - goofys
- Clients graphiques: WinSCP / S3 GUI / Cyberduck...
- Bibliothèques
  - Python: **boto3**, libcloud, s3fs
  - C/C++: ceph-libs3, aws-sdk-cpp, aws-c-s3
  - PHP: aws/aws-sdk-php
  - Go: aws-sdk-go/service/s3
  - Rust: rust-s3, aws-sdk-s3

    ...

Config clients AWS

Remarque: les docs ne sont pas toujours très claires pour changer les points d'accès S3 (par défaut: AWS)...

Variables d'environnements pour `aws cli` et `boto3` :

```
$ export AWS_SECRET_ACCESS_KEY=XXX
$ export AWS_ACCESS_KEY_ID=YYY
$ export AWS_ENDPOINT_URL=https://s3.mesocentre.uca.fr
$ export AWS_S3_ENDPOINT=s3.mesocentre.uca.fr
$ export AWS_REGION=fr-clermont-mesocentre
```

```
# List buckets
$ aws s3 ls
2020-02-10 21:26:46 datascience
2023-02-26 15:03:07 povrayapi
2024-03-31 16:27:07 uca-eoscfe-catalog
2024-04-01 18:54:32 uca-eoscfe-data
```

```python
import boto3

# Create AWS S3 client
s3 = boto3.client('s3')

# List buckets
response = s3.list_buckets()
for bucket in response['Buckets']:
    print(f'  {bucket["Name"]}')
```

Opérations de base: buckets

- Le nom du bucket doit être compatible avec la syntaxe DNS
- Un bucket est par défaut privé. Il peut être public: tous le monde peut y accéder sans authentification (avec un client http ou s3)
- Un bucket doit être vide pour être supprimé

Avec AWS CLI:

```
# Create a bucket
$ aws s3 mb audaces2024

# Create a public bucket
$ aws s3api create-bucket --acl public-read \
      --bucket audaces2024-public

# Delete a bucket (must be empty)
$ aws s3 rb audaces2024-public
```

Avec Boto3:

```
import boto3
s3 = boto3.client('s3')

# Create a bucket
s3.create_bucket(Bucket="audaces2024")

# Create a public bucket
s3.create_bucket(Bucket="audaces2024-public", ACL='public-read')

# Delete a bucket (must be empty)
s3.delete_bucket(Bucket="audaces2024-public")
```

Opérations de base: objects

- L'id (*key*) d'un objet peut contenir des `/` pour organiser les objets dans des pseudo-répertoires

Avec AWS CLI:

```
# Create an object from local file
$ aws s3 cp E0033.tif s3://audaces2024/toto/E0033.tif

# Download an object to a local file
$ aws s3 cp s3://audaces2024/toto/E0033.tif ./myfile

# Create an object from stdin
$ echo "hello world" | aws s3 cp - s3://audaces2024/hello

# Download an object to stdout
$ aws s3 cp s3://audaces2024/hello - | less

# Delete an object
$ aws s3 rm s3://audaces2024/toto/E0033.tif
```

Avec Boto3:

```
import boto3
s3 = boto3.client('s3')

# Create an object from local file
s3.upload_file(Filename="./E0033.tif",
               Bucket="audaces2024",
               Key="toto/E0033.tif")

# Download an object to a local file
s3.download_file('audaces2024', 'toto/E0033.tif', './myfile')

# Create an object from python
data = "Hello World"
s3.put_object(Body=data, Bucket='audaces2024', Key='hello')

# Download an object to stdout
response = s3.get_object(Bucket='audaces2024', Key='hello')
data = response['Body'].read()
```

Accès S3 via bibliothèques python de plus haut niveau

Depuis pandas (avec `boto3` et `s3fs`):

```
import pandas as pd
df = pd.read_csv("s3://audaces2024/98755099999_2020.csv",
                 parse_dates=['DATE'], index_col='DATE')
print(df.info)
...
df2.to_csv("s3://audaces2024/results.csv")
```

Analyse distribuée avec Dask:

```
import dask
import dask.dataframe as dd

dask.config.set(scheduler='processus')
dask.config.set(num_workers=4)
df = dd.read_csv("s3://datascience/data/01262099999/*.csv")
df.DATE = data.DATE.astype('datetime64')
df.persist()
...
```

Avec GeoPandas (Pandas pour les données géospatiales)

```
# pip install geopandas pyarrow pyogrio fsspec 'gdal<=3.7.3'
import geopandas as gpd
gpd.options.io_engine = "pyogrio"

gdf = gpd.GeoDataFrame(df,
      geometry=gpd.points_from_xy(df.LONGITUDE, df.LATITUDE))
gdf.to_parquet("s3://audaces2024/gdf.parquet")
```

Avec DuckDB (SQL pour données orientées colonnes)

```
import duckdb
r = duckdb.sql("""CREATE SECRET secret (TYPE S3,
PROVIDER CREDENTIAL_CHAIN,CHAIN 'env',
ENDPOINT 's3.mesocentre.uca.fr');""")

r = duckdb.sql("""SELECT COUNT(*)
FROM read_parquet('s3://audaces2024/gdf.parquet');""")
print(r)
```

Métadonnées

- En plus des métadonnées classiques sur les objets (propriétaires, dates, checksum), l'usager peut ajouter ses propres métadonnées aux objets
- On retrouve les métadonnées dans les headers HTTP avec le préfixe `x-amz-meta-`

Avec AWS CLI:

```
# Object creation with metadata
$ aws s3 cp --content-type image/tiff \
            --metadata "fairease.catalog.mediatype=COG"
            E0033.tif s3://audaces2024/toto/E0033.tif

$ aws s3api head-object --bucket audaces2024
                        --key toto/E0033.tif
```

Avec Boto3:

```
import boto3
s3 = boto3.client('s3')

# Download an object to a local file
s3.upload_file(Filename="./E0033.tif", Bucket="audaces2024",
               Key="toto/E0033.tif",
               ExtraArgs={"Metadata": {
                   "fairease.catalog.mediatype": "COG",
              }}
)

meta = s3.upload_file(Bucket="audaces2024",
                      Key="toto/E0033.tif")
```

Presigned URL

- permet de générer une URL de partage vers un objet d'un bucket privé
- nécessite de spécifier une durée de validité de l'URL en secondes

Avec AWS CLI:

```
# Presigned URL for sharing object
$ aws s3 presign s3://audaces2024/toto/E0033.tif --expires-in 86400
https://s3.mesocentre.uca.fr/audaces2024/toto/E0033.tif?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-...
```

Avec Boto3:

```
import boto3
s3 = boto3.client('s3')

url = s3.generate_presigned_url('get_object',
    Params={"Bucket": 'audaces2024', "Key": 'toto/E0033.tif'},
    ExpiresIn=86400)

print(url)
```

Versioning

- Le versioning permet de conserver différentes versions des objets d'un bucket
- Le versioning s'applique au niveau d'un bucket (et concerne tous les objets du bucket)

Avec AWS CLI:

```
# Activate versioning
$ aws s3api put-bucket-versioning \
--bucket audaces2024 \
--versioning-configuration Status=Enabled

# Create versions
$ echo "hello1" | aws s3 cp - s3://audaces2024/hello
$ echo "hello2" | aws s3 cp - s3://audaces2024/hello

# Show version
$ aws s3api list-object-versions \
--prefix hello \
--bucket audaces2024 | jq
```

Avec Boto3:

```
import boto3

bucket = boto3.resource('s3').Bucket('audaces2024')
bucket.Versioning().enable()

bucket.put_object(Body="hello1", Key='hello')
bucket.put_object(Body="hello2",  Key='hello')

versions = bucket.object_versions.filter(Prefix='hello')
for version in versions:
    obj = version.get()
    print(obj.get('VersionId'), obj.get('LastModified'))
    print(obj.get('Body').read().decode('utf-8'))
```

Versioning: Lifecycle

- Lorsque le versioning est activé, on peut définir un cycle de vie des versions et notamment des durées de rétention

Avec AWS CLI:

```
$ cat ./lifecycle.json
{"Rules": [{
  "ID": "Delete old versions after 10 days",
  "Status": "Enabled",
  "NoncurrentVersionExpiration": {
    "NoncurrentDays": 10
  }
}]}

$ aws s3api put-bucket-lifecycle-configuration \
--bucket audace2024 \
--lifecycle-configuration  file://lifecycle.json
```

Avec Boto3:

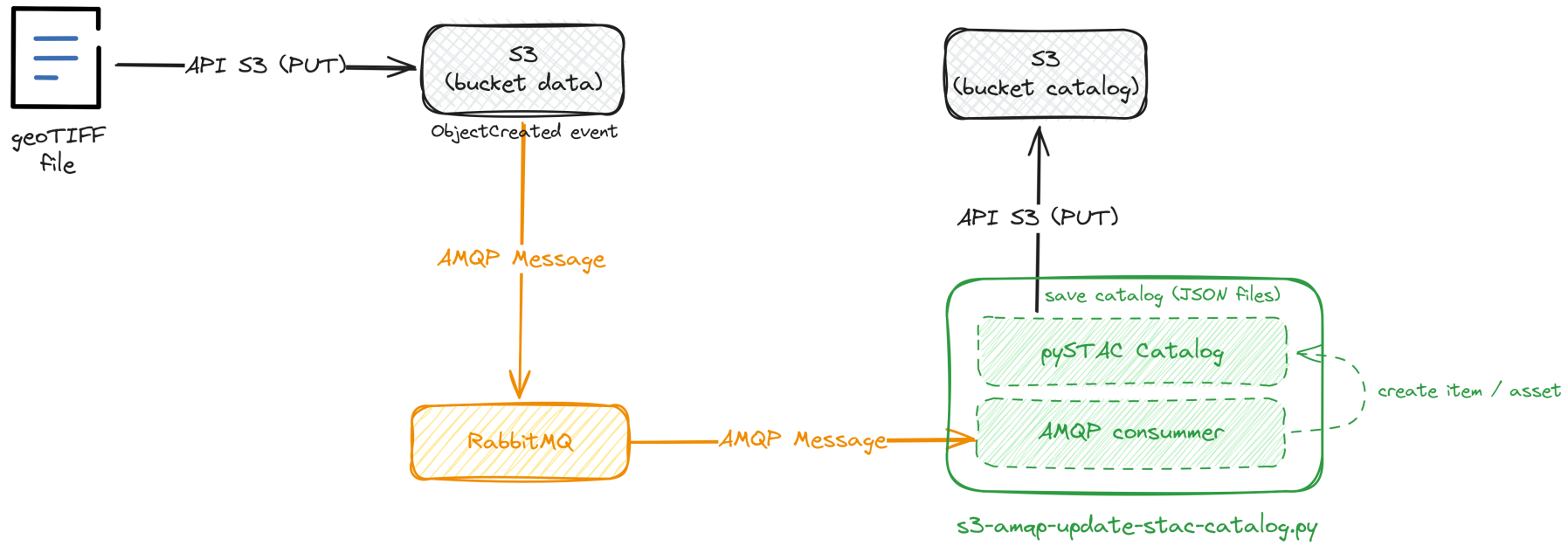
```
import boto3

life10days = {"Rules": [{
    "ID": "Delete old versions after 10 days",
    "Status": "Enabled",
    "NoncurrentVersionExpiration": {
        "NoncurrentDays": 10
    }
}]}

bucket = boto3.resource('s3').Bucket('audaces2024')
bucket.LifecycleConfiguration().put(
  LifecycleConfiguration=life10days
)
```

Notifications S3 (1)

- Notification par message AMQP des événements d'un bucket S3
- Expérimentations dans le cadre du projet EOSC FAIR-EASE
- Objectif: maintenir automatiquement un catalogue spatio temporel (STAC) d'un jeu de données depuis un bucket



Notifications S3 (2)

```
## Create SNS Topic
sns = boto3.client('sns', **client_options, config=Config(signature_version='s3'))
topic_attributes = {
    "push-endpoint": "amqp://192.168.81.196:5672",
    "amqp-exchange": "fairease-s3-events",
    "amqp-ack-level": "none",
    "persistent": "false"
}

topic = sns.create_topic(Name="fairease", Attributes=topic_attributes)
```

```
# Config of notifications
s3 = boto3.client('s3', **client_options)
notifconfig = {'TopicConfigurations': [{
  'Id': 'data',
  'TopicArn': topic['TopicArn'],
  'Events': [
    's3:ObjectCreated:*',
  ],
}]}

s3.put_bucket_notification_configuration(Bucket="uca-eoscfe-data", NotificationConfiguration=notifconfig)
```

Notifications S3 (3)

```python
# AMQP Consummer Example
import pika

connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.exchange_declare(exchange='fairease-s3-events', exchange_type='fanout')

result = channel.queue_declare(queue='', exclusive=True)
queue_name = result.method.queue

channel.queue_bind(exchange='fairease-s3-events', queue=queue_name)

print(' [*] Waiting for logs. To exit press CTRL+C')

def callback(ch, method, properties, body):
    print(f" [x] {body}")

channel.basic_consume(
    queue=queue_name, on_message_callback=callback, auto_ack=True)

channel.start_consuming()
```

Accès au S3 du Mésocentre Clermont-Auvergne

- sur projet via support.dsi@uca.fr
  - pour une durée limitée dans le temps (max 3 ans)
- sur le modèle de STORM
  - avec contribution financière annuelle du labo

Merci pour votre attention !