Importing NRD2 Data Feed to GCP

Posted on October 18, 2023

The intention of this document is to show you the basics of how to download the WhoisXML API's NRD2 data feed provided by WhoisXML API to a GCP Cloud Storage bucket by leveraging a serverless Cloud Functions. GCP Cloud Functions acts as a serverless compute service that allows you to write and execute code without provisioning or managing servers. GCP Cloud Storage is an object storage service for storing and retrieving files. This document will guide you through the process of configuring both GCP Cloud Functions and a GCP Cloud Storage bucket.

Out of scope:

- Scheduling a function for Cloud Functions
- ETL pipelining
- Importing the Python requests module
- Advanced Security
- Clean-up, life cycle file management

Prerequisites

Please ensure you have the following setup:

- GCP Account

- Basic to Intermediate knowledge of GCP services, specifically GCP Cloud Functions and Cloud Storage
- Some familiarity with Python which will be used in Cloud Functions
- Access to the WHOIS NRD2 data feed. In this example, we will be using the NRD2 Ultimate: Simple files. You will need an API key with access to the data feed. Please contact us for more information. For more information on the NRD2 specifications, please visit here.

Ultimate

- **Data included:** discovered, registered, updated and dropped domains, generic and country TLDs, WHOIS records.
- **Filename format:** *nrd.%DATE%.**ultimate**.[daily].[data|stats].[csv|json]*
- **Filename example:** nrd.2021-12-20.ultimate.daily.data.csv.zip, nrd.2021-12-20.ultimate.daily.data.json.zip
- **Average file sizes:**

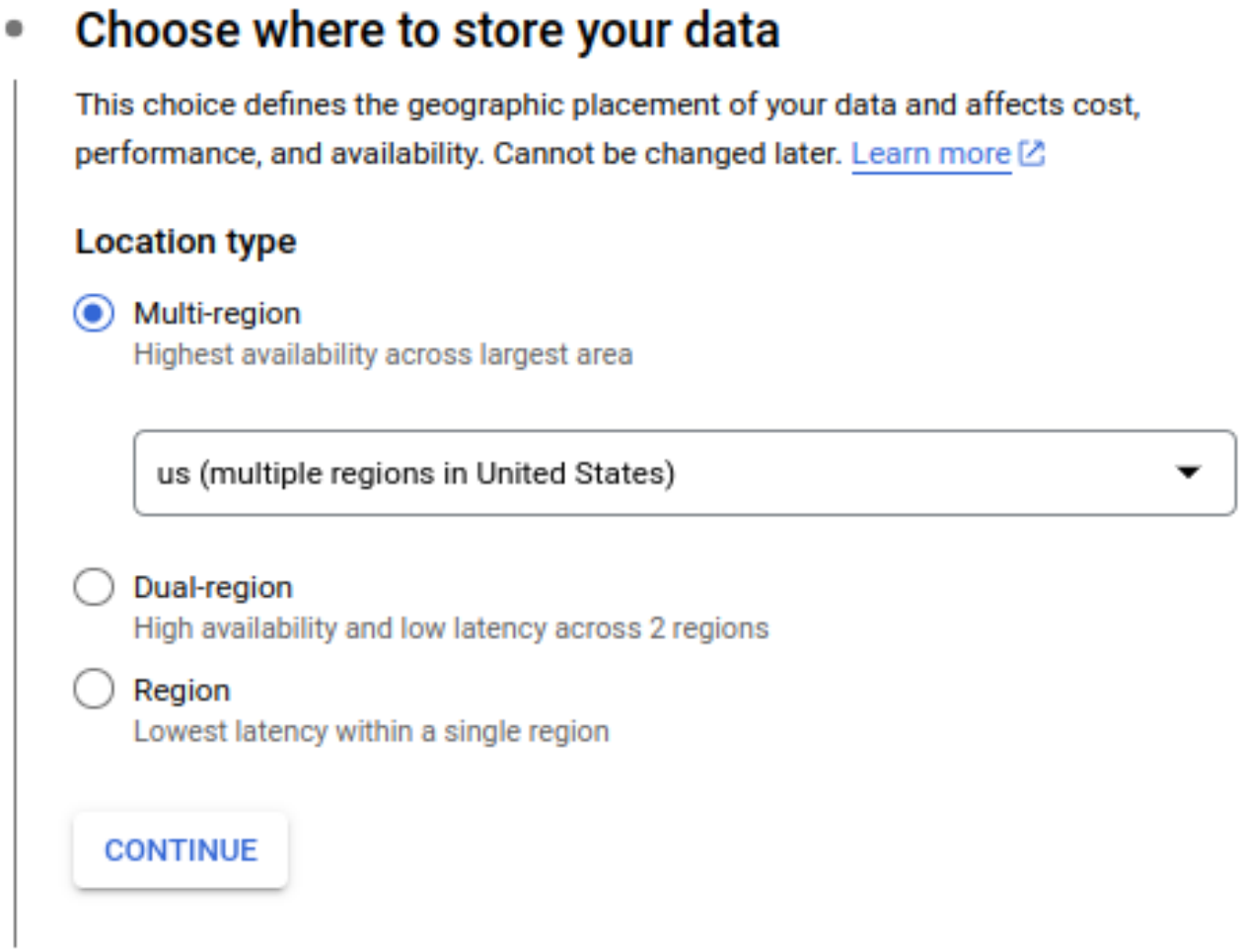
File	Gzip size	Unpacked size	Rows
ultimate.daily.data.csv.gz	423.9MiB	3.7GiB	709.3K
ultimate.daily.data.json.gz	526.0MiB	6.2GiB	709.3K

Step 1: Create a GCP Cloud Storage Bucket

The first step is to create a Cloud Storage bucket to write the NRD2 file.

- In the GCP Console, navigate to the Cloud Storage service.
- Click on “Create”.
- Give the bucket a unique name and select the appropriate region and a storage class for

your data.


Choose where to store your data
This choice defines the geographic placement of your data and affects cost, performance, and availability. Cannot be changed later. Learn more
Location type
Multi-region
Highest availability across largest area
us (multiple regions in United States)
Dual-region
High availability and low latency across 2 regions
Region
Lowest latency within a single region
CONTINUE

- ## Choose a storage class for your data

A storage class sets costs for storage, retrieval, and operations, with minimal differences in uptime. Choose if you want objects to be managed automatically or specify a default storage class based on how long you plan to store your data and your workload or use case. Learn more

○ Autoclass
Automatically transitions each object to hotter or colder storage based on object-level activity, to optimize for cost and latency. Recommended if usage frequency may be unpredictable. Can be changed to a default class at any time. Pricing details

◉ Set a default class
Applies to all objects in your bucket unless you manually modify the class per object or set object lifecycle rules. Best when your usage is highly predictable. Can't be changed to Autoclass once the bucket is created.

◉ Standard
Best for short-term storage and frequently accessed data

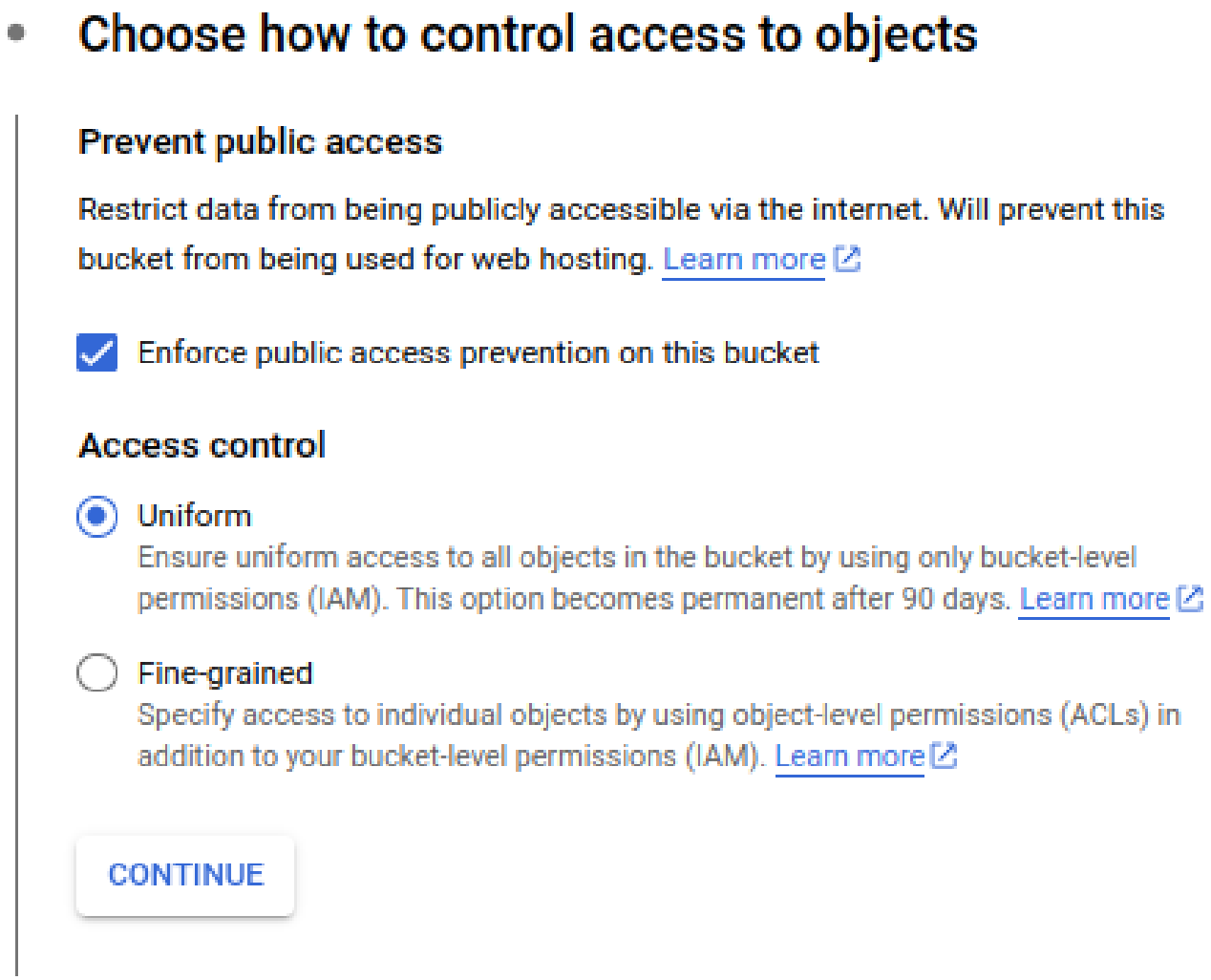
○ Nearline
Best for backups and data accessed less than once a month

○ Coldline
Best for disaster recovery and data accessed less than once a quarter

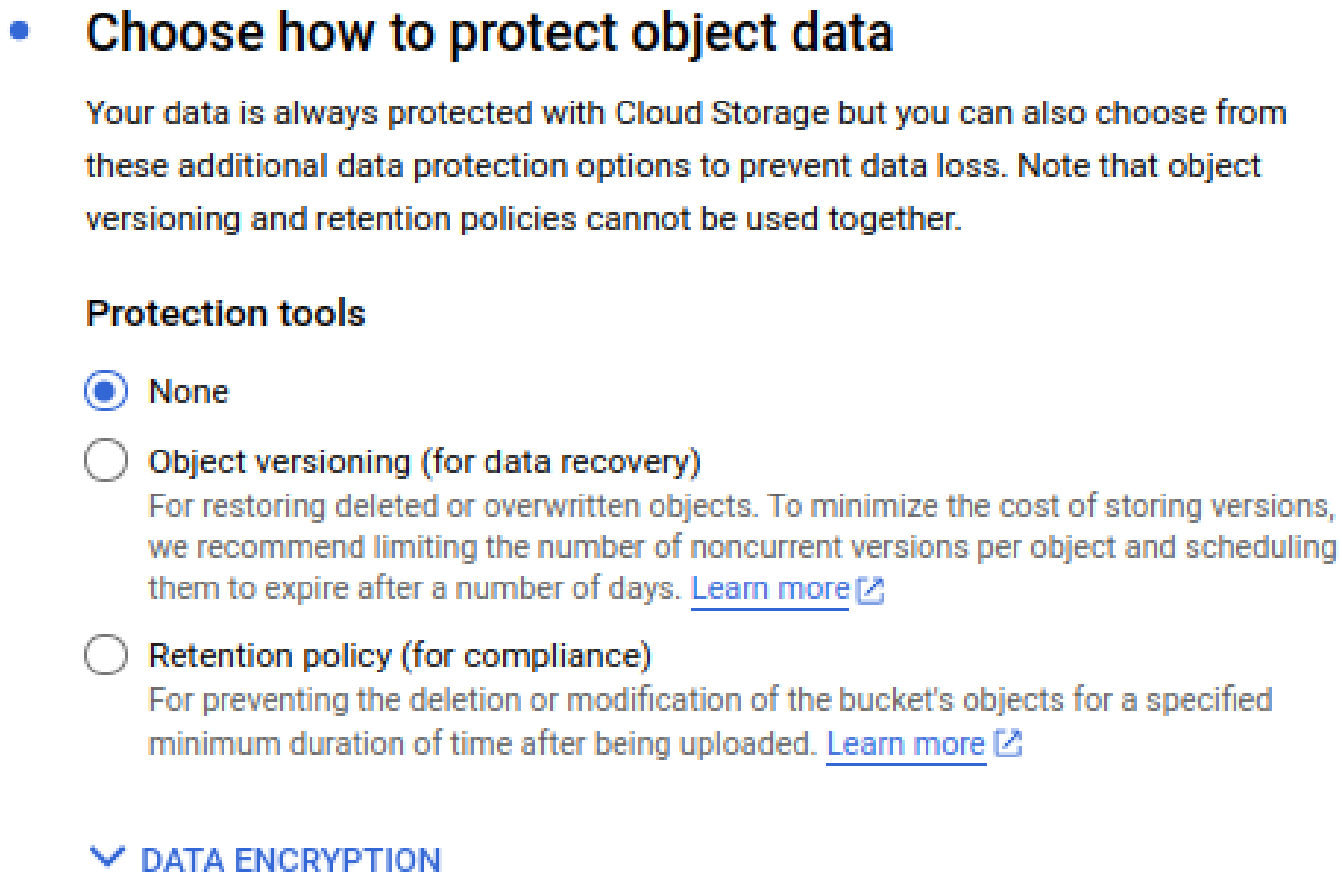
○ Archive
Best for long-term digital preservation of data accessed less than once a year

CONTINUE

- Then, choose how you would like to manage access to objects. By default, it prevents public access and has a uniform policy at the bucket level.



- Finally, choose how to protect object data. You have the option to do object versioning or use a retention policy.



Step 2: Creating a Function in Cloud Functions

Now the magic begins.

- Navigate to the Cloud Functions service in the GCP console.
- Click on “Create Function”. Provide your function with a descriptive name. On the configuration page, specify “Require authentication” in the HTTPS trigger section.

Basics

Environment
2nd gen

Function name *
import_nrd2

Region *
us-central1 (Iowa)

Trigger

HTTPS

Authentication

Allow unauthenticated invocations
Check this if you are creating a public API or website.

Require authentication
Manage authorized users with Cloud IAM.

URL

https://us-central1-wxa-data-migration.cloudfunctions.net/import_nrd2

ADD TRIGGER

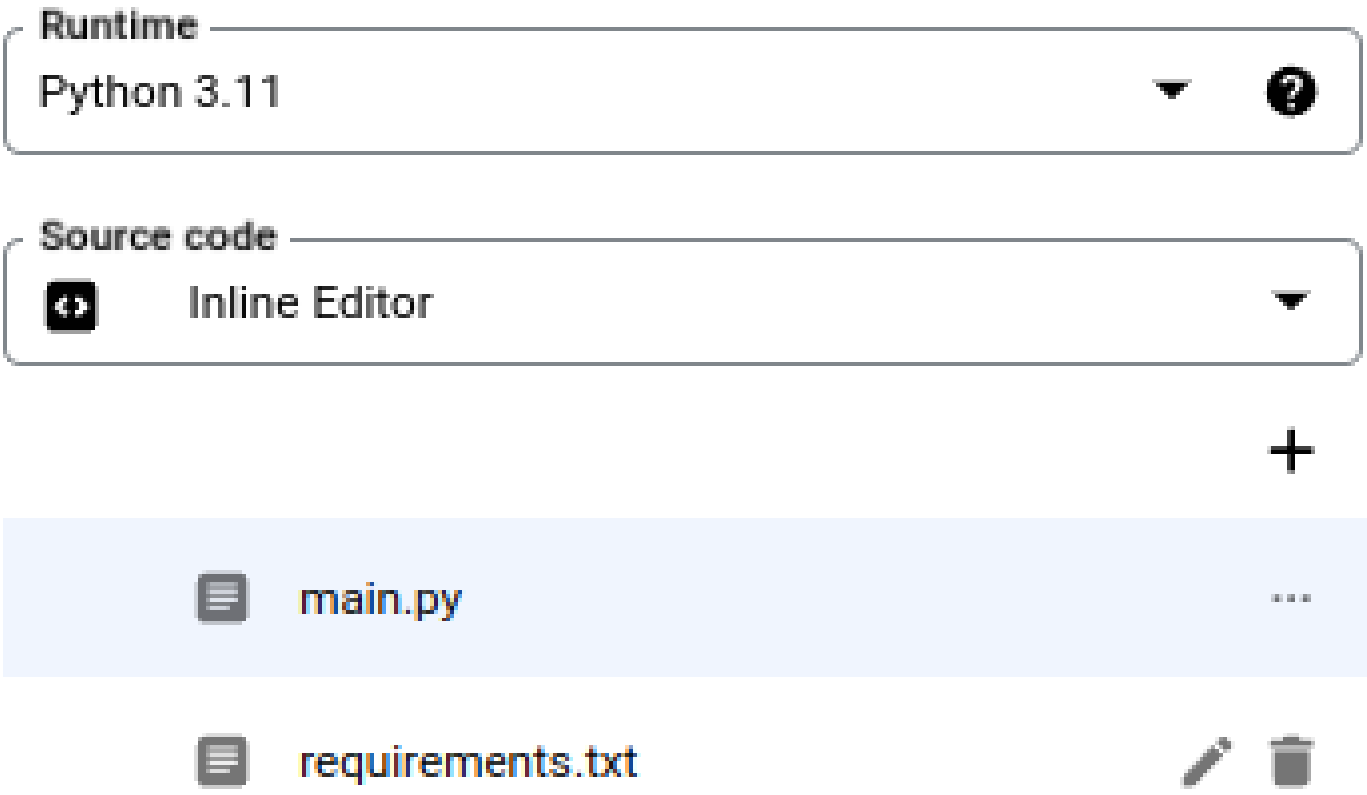
Runtime, build, connections and security settings

- Click on “Next”.

Step 3: Write the function to import the NRD2 .csv file to Cloud Storage

The example code snippet uses the Python requests module, and you may need to import it.

- First, select Python 3.* as your Runtime.



- Then, copy over requirements.txt.

```
blinker==1.6.3
cachetools==5.3.1
certifi==2023.7.22
charset-normalizer==3.3.0
click==8.1.7
cloudevents==1.10.0
deprecation==2.1.0
Flask==2.3.3
functions-framework==3.4.0
google-api-core==2.12.0
google-auth==2.23.3
google-cloud-core==2.3.3
google-cloud-storage==2.12.0
google-crc32c==1.5.0
google-resumable-media==2.6.0
googleapis-common-protos==1.61.0
gunicorn==20.1.0
idna==3.4
itsdangerous==2.1.2
Jinja2==3.1.2
MarkupSafe==2.1.3
packaging==23.2
protobuf==4.24.4
pyasn1==0.5.0
pyasn1-modules==0.3.0
requests==2.31.0
rsa==4.9
urllib3==2.0.6
watchdog==3.0.0
Werkzeug==3.0.0
```

- The below Python code provides the entry point for the lambda_handler function:

Example code:

```
import os
from datetime import datetime, timedelta
import requests
from requests.auth import HTTPBasicAuth

import functions_framework
from google.cloud import storage


def download_nrd_file(url, bucket_name, blob_name, authUserPass):
    project_id = "wxa-data-migration"
    storage_client = storage.Client(project=project_id)
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(blob_name)

    CHUNK_SIZE = 1024 * 1024
    try:
        # Download the binary file in chunks
        response = requests.get(
            url, stream=True, auth=HTTPBasicAuth(authUserPass, authUserPass)
        )
        response.raise_for_status()

        # Create a temporary file to store chunks
        temp_file = "/tmp/temp_file"

        with open(temp_file, "wb") as f:
            for chunk in response.iter_content(chunk_size=CHUNK_SIZE):
                f.write(chunk)
```

```
        # Upload the binary file to Cloud Storage from the temporary file
        blob.upload_from_filename(temp_file)

        # Clean up the temporary file
        os.remove(temp_file)

        return True
    except Exception as e:
        print(f"Error: {str(e)}")
        return False


@functions_framework.http
def lambda_handler(request):
    # Calculate yesterday's date in YYYY-MM-DD format
    yesterday = (datetime.now() - timedelta(days=1)).strftime("%Y-%m-%d")

    # Define the URL of the CSV file you want to download
    nrd_url = f"https://newly-registered-domains.whoisxmlapi.com/datafeeds/Newly_Registered_Domai

    # Define your API Key here
    apiKey = "<YOUR_API_KEY>"

    # Define the Cloud Storage bucket and object/key where you want to store the file
    bucket_name = "nrd2"
    blob_name = f"nrd2-simple-{yesterday}.csv.gz"

    try:
        # Download the NRD2 file with basic authentication
        success = download_nrd_file(nrd_url, bucket_name, blob_name, apiKey)

        print("Status code returned is ", str(success))
        if success:
            # Upload the NRD file to Cloud Storage
```

```
        print(f"Uploading file to ", bucket_name, blob_name)
        return {
            "statusCode": 200,
            "body": "NRD2 file successfully downloaded and stored in GCP",
        }
    else:
        bodyStr = f"Failed to download {nrd_url}"
        return {"statusCode": 500, "body": bodyStr}
except Exception as e:
    return {"statusCode": 500, "body": str(e)}
```

- Specify “Entry point” to be the entry function of your code. In our case, it’s “lambda_handler”.

Step 4: Testing your new function

The last step is to test the function to ensure it can a) successfully retrieve the NRD2 file, and b) write it to the Cloud Storage bucket:

- Click on “TEST FUNCTION” at the top of the page, and you should see something similar. GCP will fire up a Cloud Shell and set up the testing environment.

- Click on “RUN TEST” in the bottom right corner.

If your function is set up correctly, the function will retrieve the file, and write it to the Cloud Storage bucket. You can navigate to the Cloud Storage bucket to verify it’s there.

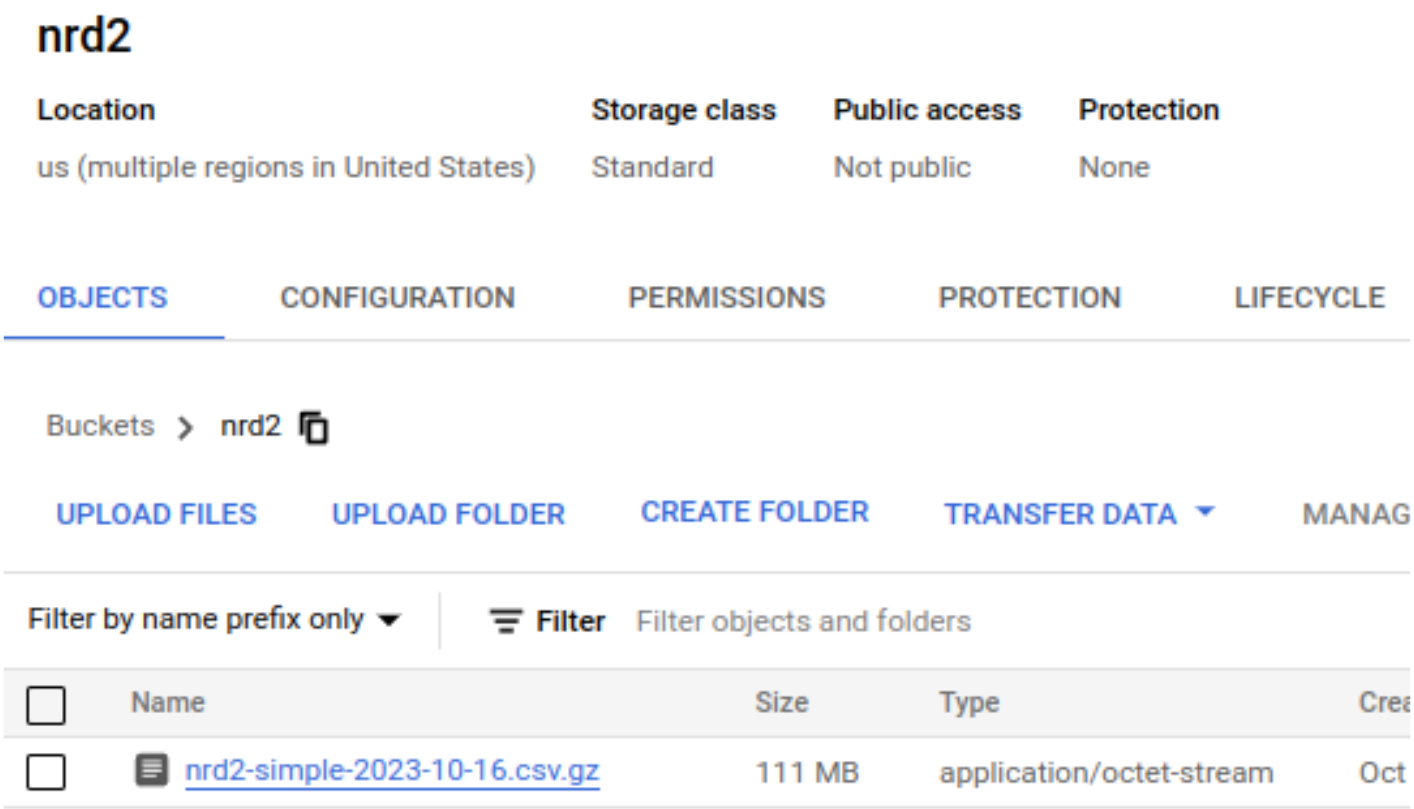
The output of the console should resemble this.

```
############################################################
[11:10:17 AM] - Function is ready to test
############################################################


[11:10:25 AM] - Status code returned is  True
Uploading file to  nrd2 nrd2-simple-2023-10-16.csv.gz

[11:10:26 AM] - Execution response: {"body":"NRD2 file successfully downloaded and stored in GCP","statusCode":200}
```

Then, in Cloud Storage, you'll have a new object added to the bucket.



Conclusion

The steps we took in GCP are similar to the AWS setup. After walking you through the process, the next step is to determine what you want to do with this data, such as import it into BigQuery, or MySQL database.