REFINITIV DATA PLATFORM

CLIENT FILE STORE

USER GUIDE FOR PUBLISHER/SUBSCRIBER

REFINITIV®

DATA IS JUST
THE BEGINNING®

Contents

About this document

The Client File Store (CFS) is a capability of the Refinitiv Data Platform (RDP) that provides authorization and enables access to content files stored in publisher-supplied repositories. This guide provides details on the CFS REST API working environment and supporting resources. It is intended for software engineers who are familiar with the general principles of APIs and assumes they are familiar with their intended programming language.

Feedback

We invite your comments, corrections, and suggestions about this document: access the Feedback option under Help & Support at MyRefinitiv. Your feedback helps us continue to improve our user assistance.

Support

The Refinitiv Statement of Service is available on MyRefinitiv. MyRefinitiv is the Refinitiv portal that provides a single access point for timesaving support services, along with billing, user management, and information. For support, please raise a query by accessing Help & Support at MyRefinitiv.

You are encouraged to subscribe to the following support channels to keep informed of changes to products and data, and to be notified of any service issues or changes:

- **Change Notifications**
  - **Product** change notifications detail new, enhanced, or changed functionality, which may require your action, in products that you use.
  - **Content** change notifications alert you to upcoming changes to real-time and historical data across all asset classes that are relevant to you.
  - **RIC** change notifications inform you of planned changes to Reuters Instrument Codes.
- **Service Alerts**

You can subscribe to alerts about planned maintenance and unplanned service issues affecting your products and services and be notified via SMS or email.

Your Personal Information

Refinitiv is committed to the responsible handling and protection of personal information. We invite you to review our Privacy Statement, which describes how we collect, use, disclose, transfer, and store personal information when needed to provide our services and for our operational and business purposes.

1 Getting Started

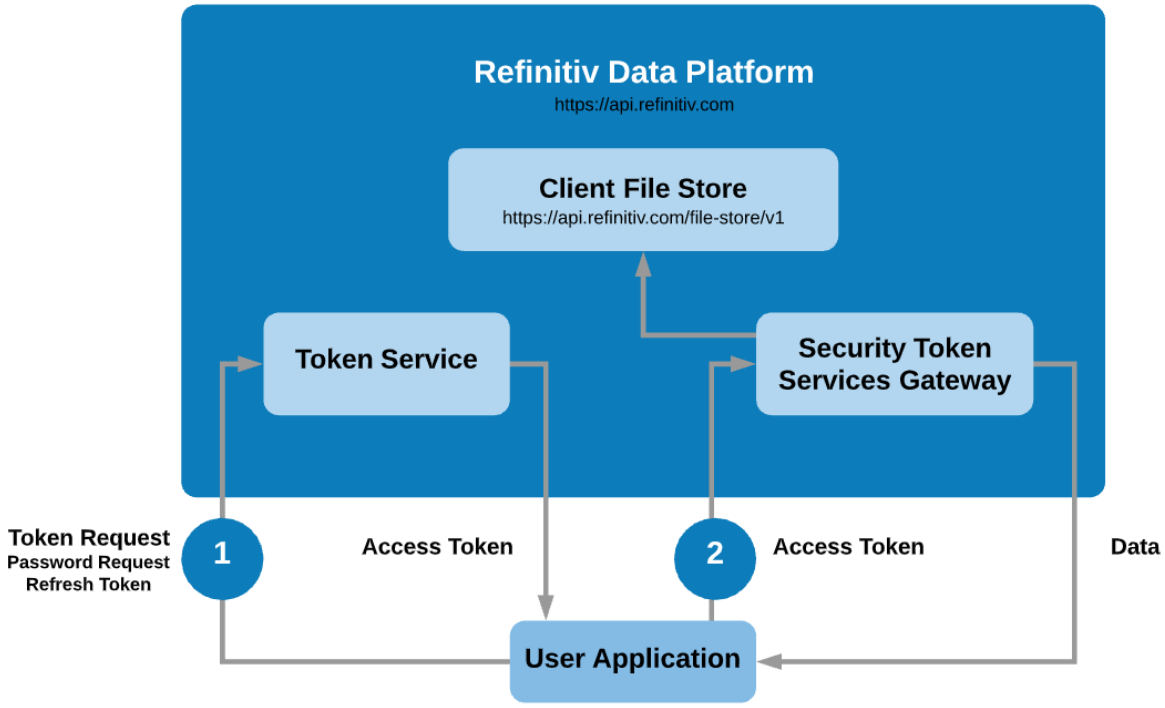
The Client File Store (CFS) is a component of the Refinitiv Data Platform (RDP). This chapter introduces RDP and describes how to interact with it. New users should review this chapter, as understanding RDP will enable you better utilize CFS. Users who are experienced with RDP can skip ahead to About the Client File Store.

1.1 About Refinitiv Data Platform

RDP integrates and distributes real-time, reference, and analytics data along with your proprietary and third-party customer data. Utilizing a range of cloud capabilities and web-based REST APIs, RDP provides a significantly enhanced user experience with real-time and non-real-time content accessible in an integrated way.

1.2 Application Workflow

The CFS APIs are considered protected resources and require that your application be authenticated before making a data call. This authentication and provisioning of access token is based on OAuth 2.0 specification.



An application starts by getting an access token for CFS through a third-party REST API application, such as Postman and Python. Access tokens are valid for five minutes. Once a valid token is received, this token is sent with every request to get data. The RDP component Security Token Service (STS) verifies that a token is included in the data request. If the token is still valid and has appropriate scope, the request is allowed to access the CFS. Once authenticated, you can interact with the CFS APIs.

1.3 Learning Resources

1.3.1 Refinitiv Developer Community

The Developer Community provides information about RDP APIs. The Refinitiv Data Platform APIs portal provides details on the available API services, including the CFS, and offers a quick start section, code samples, tutorials, and other documentation.

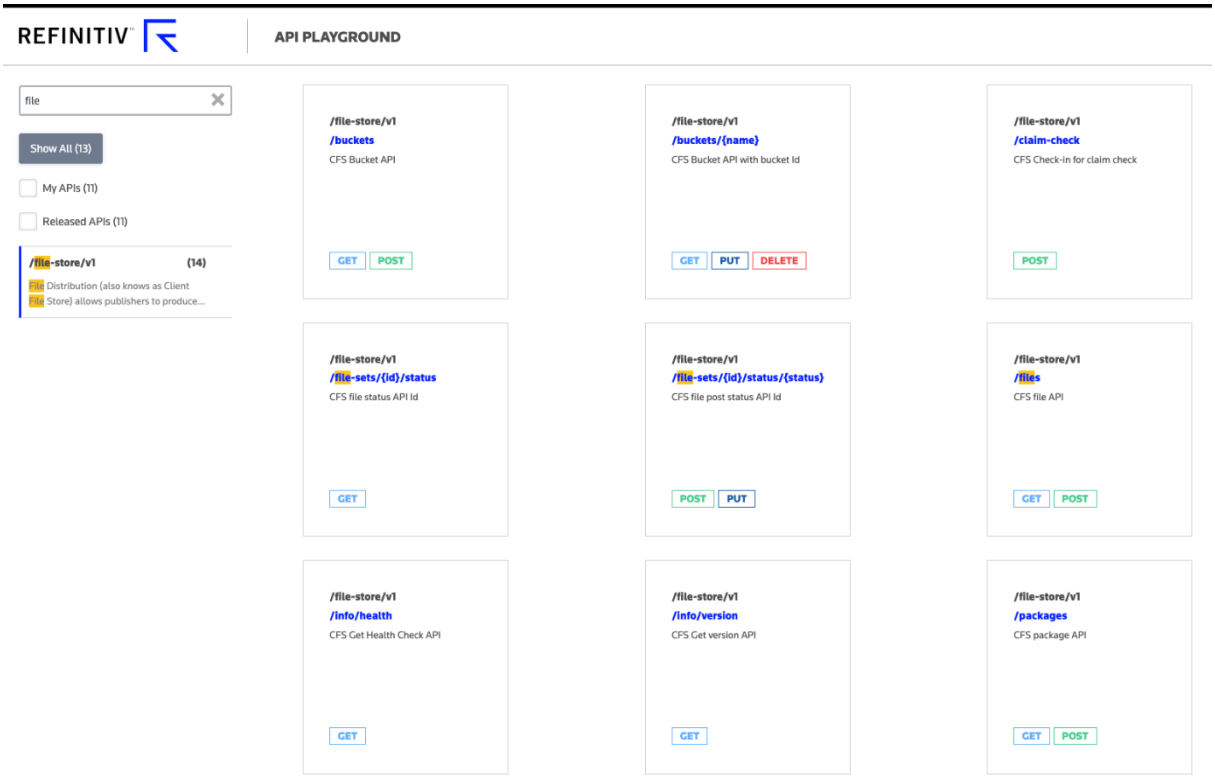
The portal requires free registration to access documentation and sample code.

1.3.2 API Playground

The API Playground is a front end to the CFS APIs via a Graphical User Interface (GUI). It acts as an interactive reference guide and tool. It allows you to experiment with the CFS APIs, send requests, view responses, and review Swagger documentation. The API Playground URI is:

https://apidocs.refinitiv.com/Apps/ApiDocs

Access to this site is available via a valid username and password, which you can obtain from your account manager. This site is best viewed using Google Chrome. To access the CFS APIs, enter "`File Distribution`" in the **Filter** box. The available APIs are returned in the body of the screen:



The screen defaults to the **Playground** tab, from where you can view sample requests and response data. The available methods and their corresponding descriptions are shown in the pane. The URI identifies the data request being made on the Refinitiv domain.

1.3.3 Setting Up Your Application

The web-based REST APIs available via the CFS use the HTTP Request Respond protocol. You can use any major programming language to build your APIs.

For instructions on setting up a development environment using Python (version 3.x is recommended over earlier versions) and Postman, see the Introductory Tutorial section in the Refinitiv Data Platform APIs portal on the Developer Community (requires login).

1.4 Requesting Authentication Tokens

When you submit a request to any of the CFS APIs, you must authenticate that you are authorized to do so by providing a temporary authentication token. For RDP, the token endpoint is:

```
<Environment URI>/auth/oauth2/v1/token
```

When requesting a token, you must specify your Client ID.

Note: For an overview on requesting authentication tokens, see the Introductory Tutorial section in the Refinitiv Data Platform APIs portal on the Developer Community (requires login).

2 About the Client File Store

2.1 CFS Definitions

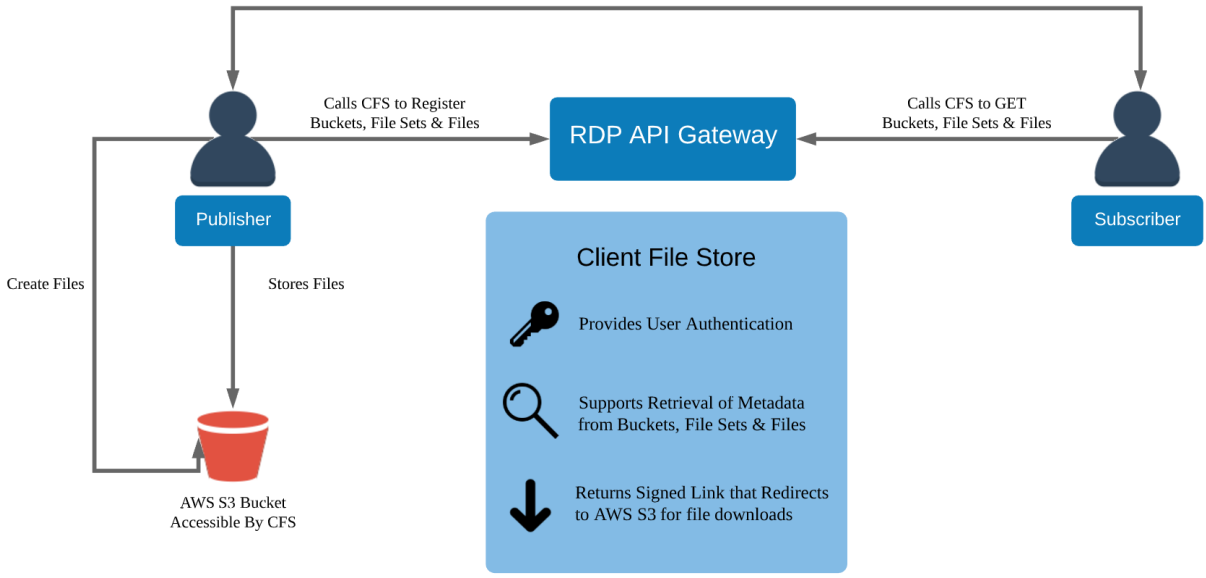
The **Client File Store (CFS)** is a capability of the Refinitiv Data Platform (RDP) that provides authorization and enables access to content files stored in publisher-supplied repositories. One such repository, used as an example throughout this document, is Amazon Web Services (AWS) S3. CFS is designed as a self-service metadata tool and is intended for both Publishers and Subscribers. Publishers call CFS APIs to post metadata about their files stored in these repositories, while Subscribers call CFS APIs to discover and download those files. CFS defines content ownership so that Publishers are isolated, and Subscribers can trust the source of the content.

CFS provides buckets and file-sets to organize files to simplify the interaction with Publishers and Subscribers. Buckets can contain any number of file-sets, and file-sets can contain any number of files. A bucket can be owned by one or more Publishers. Publishers control access to that bucket, as well as to the file-sets and files in it, using entitlements and claims. Publishers can assign attributes to aid in the discovery of files by Subscribers.

In addition to Publishers, CFS users with write Access have full access to buckets, file-sets, and files and are authorized to make POST, PUT, and DELETE calls.

Subscribers can query for files to which they are entitled by attributes, such file name or date range. To retrieve a file, Subscribers are presented with a signed URI that redirects to the file in the Publisher's repository for downloading**.** Subscribers can register to receive automatic notifications and alerts about file availability.

It is important to understand that the CFS does not store the files directly. Files are stored in publisher-supplied repositories (currently on AWS S3 while support for other locations is planned for future releases). Publishers are responsible for uploading their files to AWS S3 buckets and ensuring that CFS can access those buckets. Publishers control the content file structure and entitlements of their files. CFS self-service means that Publishers can add, update or delete files at any time without configuration or coding changes and without requiring involvement from Refinitiv.



2.2 URI Structure

The CFS API web service is accessed through the Uniform Resource Identifier (URI). All following references in this document will assume this base portion of the URI has been specified:

Base URI

Production: https://api.refinitiv.com/file-store/v1

Pre-Production: https://api.ppe.refinitiv.com/file-store/v1

API Docs

Production: https://apidocs.refinitiv.com/

Pre-Production: https://apidocs.ppe.refinitiv.com/

2.3 Key Capabilities

2.3.1 Self-Service Publishing

CFS provides the ability for Publishers to onboard and publish content files, determine subscriptions, and define the metadata for that content.

2.3.2 Query Publisher-Provided Metadata

CFS provides additional functionality using publisher-provided metadata. The CFS allows Subscribers to query files by bucket, date range, file-set, and Publisher, as well as by other attributes.

2.3.3 Restrict File Access through Claims

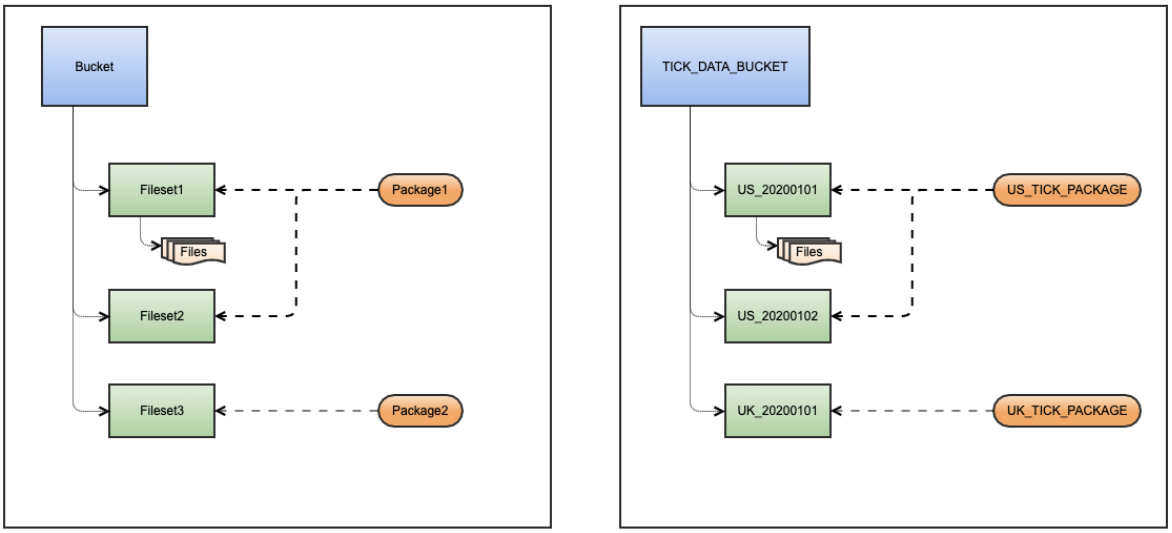
Claims control who can view and download the content. Publishers must provide claims for published files and file-sets. Only authorized Subscribers can view and download these files.

2.3.4 Secure File Retrieval

Subscribers use CFS to find and download content files provided by Publishers to which they are entitled. Authentication is required to access files. File discovery can be done by querying or registering for notifications and alerts. CFS provides a signed URL that redirects to AWS S3 that the Subscriber can use to download the file.

2.4 Key Components

The following diagrams show relationship among 4 key components – Bucket, Package, File-set, and File.



2.4.1 Buckets

CFS facilitates buckets for use by Publishers to organize file-sets and files. Buckets store metadata about the files stored in publisher-supplied repositories. Buckets align with subscriptions and can contain multiple file-sets and files.

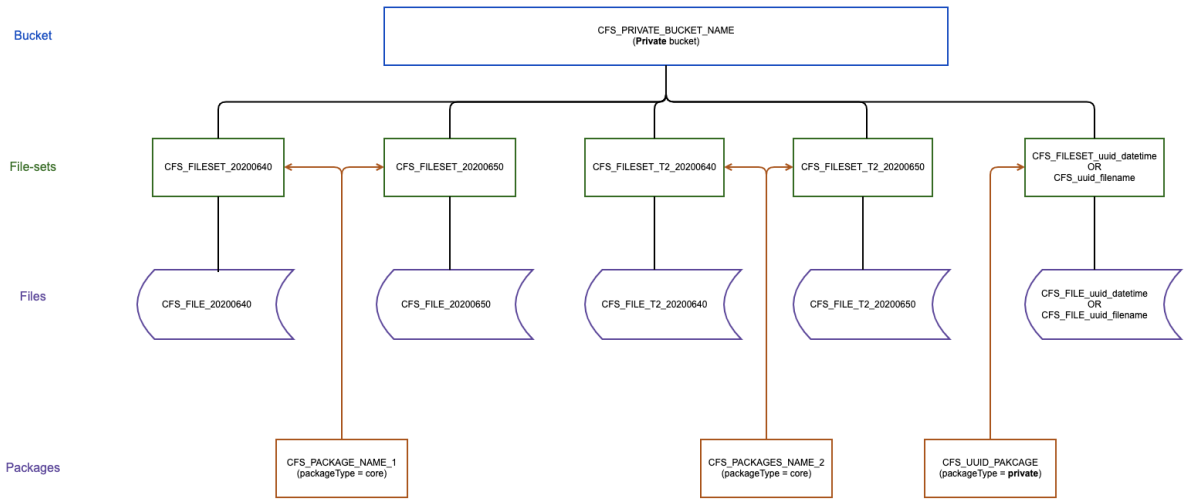
Publishers are responsible for creating buckets with the CFS API. This is a one-time process. The resulting bucket is owned by one or more Publishers and is assigned a unique name that cannot be assigned to another bucket. A Publisher can have multiple buckets if they provide more than one dataset.

Attributes are used to allow Subscribers to filter and search for content. Attributes are one method that a Subscriber can use to find files and/or file-sets.

19.1 Private Buckets

A private bucket is a special bucket type that provide a capability to validate the subscriber permission on file-sets and file level. If the publisher wants to publish a confidential file-sets or files which don't want other subscriber who doesn't have a permission to be able to search it. Private bucket will be a suitable option. There is no permission checking required to search file-sets and files for a publisher user.

The private bucket can support every package types.



2.4.2 Packages

A package is an indivisible set of file-sets that are all delivered together. Each package can contain single or multiple file-sets.

Publishers will define the metadata for each package of content available to Subscribers in CFS. Publishers are responsible for creating packages and assigning claims to them. Publishers need to create packages first before creating file-sets into packages.

Claims are used to control download of actual files. CFS does not manage or create claims but only enforces them. Claims must be created in AAA. Subscriber must have access to all of claims in that package to be able to download those files.

20.1 Private Package

As we have introduced the private bucket, we observe a lot of use case that publisher wants to create a package based on the subscriber UUID. To make a package reusable by adding the UUID into a package name and also hide a sensitive information, we have introduced the new package type called “private”.

When the subscriber trying to query the packages from GET /packages or GET /packages/{packageId} interface, if the package is created using the private package type and subscriber doesn’t have a permission on that package, it will not be returned into the result set.

Right now, the private package is only supporting the “UUID” claim type.

2.4.3 File-Sets

A file-set is an indivisible set of files that are all delivered together. They can consist of multiple files that make up one large file or a grouping of files that represent related content. A file-set can also contain a single file. The Publisher decides the appropriate organization of their file-sets.

Publishers are responsible for creating file-sets into packages. A Publisher can have multiple file-sets in a bucket. Once all files have been added to the file-set and are ready for download, the Publisher updates the file-set status to READY. This enables the Publisher to control the release of their files and sets expectations for when the files will be available to Subscribers. File-sets with a status of READY cannot be updated or modified. Updates must be published as a new file-set.

To access a file-set, Subscribers must have access to the bucket in which the file-set resides and have all of the claims associated with the file-set.

2.4.4 Files

Subscribers can only access the files to which they are entitled. On AWS S3, Subscribers can access files using a signed URI that redirects to the file on AWS S3 for downloading.

Files are available for a defined period of time that is determined by the Publisher.

2.5 CFS API

The base URI for the CFS APIs is `<Environment URI>/file-store/v1`.

The available APIs are listed below.

Path	Operation	Description	Publisher Admin	Publisher	Subscriber
/buckets	POST	Create a bucket	√		
/buckets	GET	Return a list of buckets based on given filter criteria		√	√
/buckets/{name}	GET	Get a bucket by name		√	√
/buckets/{name}	PUT	Update a named bucket	√		
/buckets/{name}	DELETE	Delete a named bucket	√		
/package	POST	Create a package		√	
/package	GET	Return a list of packages based on given filter criteria		√	√
/package/{id}	GET	Get a package metadata by identifier		√	√
/package/{id}	PUT	Update a package by identifier		√	
/package/{id}	DELETE	Delete a package by identifier		√	
/file-sets	POST	Create a file-set		√	
/file-sets	GET	Return a list of file-sets that you have rights to view		√	
/file-sets/{id}	GET	Get file-set information for a specified identifier		√	√
/file-sets/{id}	PUT	Update a specified file-set by identifier		√	
/file-sets/{id}	DELETE	Delete a specified file-set by identifier		√	
/file-sets/{id}/status/{status}	POST	Update the status of a file-set by identifier		√	
/files	POST	Publish a new file for Subscribers		√	
/files	GET	Return a list of files based on given filter criteria		√	√
/files/{id}	GET	Return a file (metadata) by identifier		√	√
/files/{id}	PUT	Update a file by identifier		√	
/files/{id}	DELETE	Delete a file by identifier		√	
/bulk-publish	POST	Publish a fileset and multiple files		√	

2.6 CFS Environment and Policy

2.6.1 CFS Archiving Policy

Client File Store service is a metadata store service. It does not maintain the files internally, but all files are owned and maintained by each publisher in their owned S3 bucket. Depends on each publisher / content set, they could configure how long FileSet and File will be available to subscriber to see (Controlling via “Available From” and “Available To” attribute when publish FileSet). However, if any FileSet that been deleted by the Publisher, the metadata still being maintain by Client Fire Store service for at least 60 days for auditing purpose.

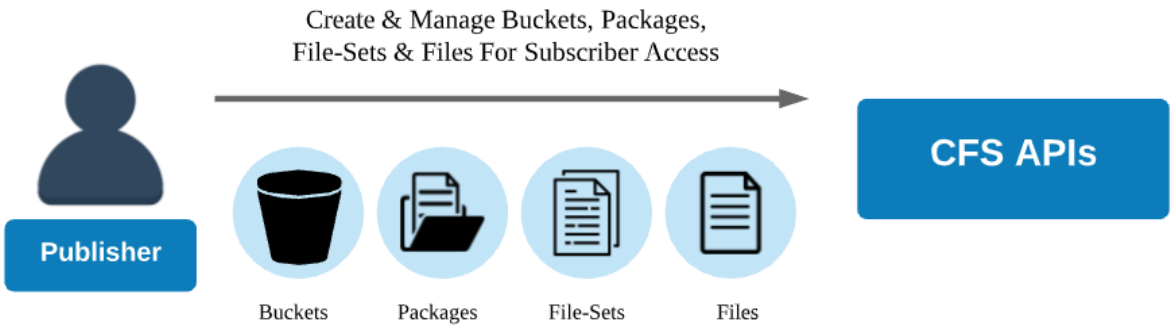
2.6.2 CFS Environment and Disaster Recovery Site

Client File Store service is running on AWS cloud environment with multi-regional support with Live/Live. The support regions are US East (N. Virginia) and Europe (Ireland). Normally clients will be redirect to the closet region based on geolocation. And if there is services disruption in any region, the clients request will be automatically redirect to alternative region.

However, as the S3 storage location are maintained by each Publisher, it could be that the file storage still subject to single S3 region unless the publisher applies MRAP (Multi-region Access Point) storage location when publish to CFS.

3 Publisher

3.1 Publisher Workflows



This chapter provides an overview the available calls that Publishers can use to create buckets, file-sets and files preceded by Best Practices. Each of these calls assumes that you have access to the Authorization Token that must be sent on each request.

1. **Creating a CFS Bucket**

Make a ***POST*** request to: ***<Environment URI>/file-store/v1/buckets***
Creating a bucket is a one-time process. The Publisher assigns a bucket name, defines the claims and subscriptions for it, and provide dates for how long the bucket will be available.

2. **Creating a CFS Package**

Make a ***POST*** request to: ***<Environment URI>/file-store/v1/package***
The Package is a collection of File-sets to be distributed to subscriber. The owner can assign who has permission to update file-sets in the package, and which group of subscribers can view content on each package through claims.

3. **Creating a File-Set for a Bucket**

Make a ***POST*** request to: ***<Environment URI>/file-store/v1/file-sets***
The Publisher must be permissioned for, or be the owner of, the bucket in which the file-set resides. Any attributes applied to the file-set must also be assigned to the bucket.

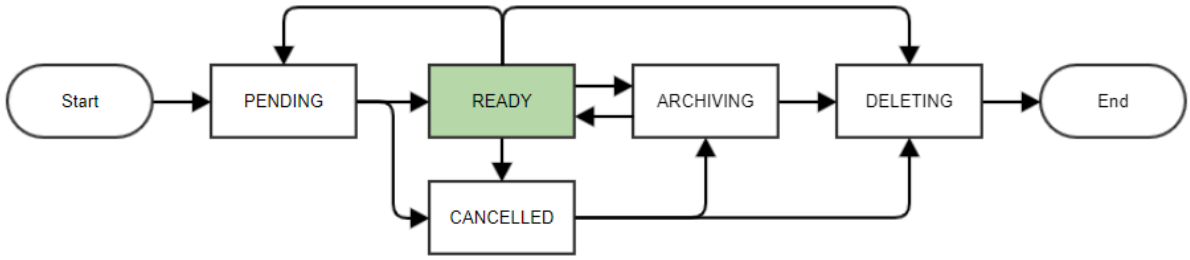
4. **Adding a File for a File-Set**

Make a ***POST*** request to: ***<Environment URI>/file-store/v1/file***
The Publisher must be permissioned for, or be the owner of, the bucket to which the file-set belongs. The attributes applied to the file-set must also be assigned to the bucket. Each file needs to be added with a single file call. This action adds a file from the Publisher's repository that can be published to Subscribers. A reference to the file is stored, not the actual file.

5. **Updating the File-Set Status**

Make a ***POST*** request to: ***<Environment URI>/file-store/v1/file-sets/{id}/status***

- New file-sets status is **PENDING** by default.
- When files in the files set have been added, the Publisher can set the file-set status to **READY**. This enables the Publisher to control the release of their files and sets expectations for when they will be available to subscribers.
- Publisher may amend list of files under the file-set by updating the status to **PENDING**. (That file-set must not be in available period yet)
- Publisher can obsolete file-set by updating file-set status to either **CANCELLED**, **ARCHIVING**, or **DELETING**. The file-set will no longer available to subscribers.



Note: Obsoleted file-set cannot be reverted to PENDING or READY status. If Publisher need to re-publish the file-set, they must create a new file-set instead.

3.2 Prerequisites

3.2.1 Prepare an AAA account

The AAA account will provide access to API Docs and allow retrieval of the authorization token that must be sent for each request to access the endpoints.

For any account to have permissions to publish files to CFS, the proper POs for publishing must be applied to the AAA account. Ensure that publisher has the permission for the desired role

Role	Permission	PO
Publisher - Administrator	Create/Modify/Remove CFS Buckets	API_ACCESS_CONTROL: API_CFS_PUBLISHER_SETUP_WRITE
Publisher	Publish/Modify Filesets/Files under existing CFS Buckets	API_ACCESS_CONTROL:API_CFS_PUBLISHER_WRITE

3.2.2 Create a storage

CFS only supports files that have been stored in AWS S3. Follow the Amazon Web Services' Getting Started with Amazon S3 guide to create your storage in the know as the bucket.

CFS does not store your actual files, and a storage location must exist before CFS can allow retrieval of the documents.

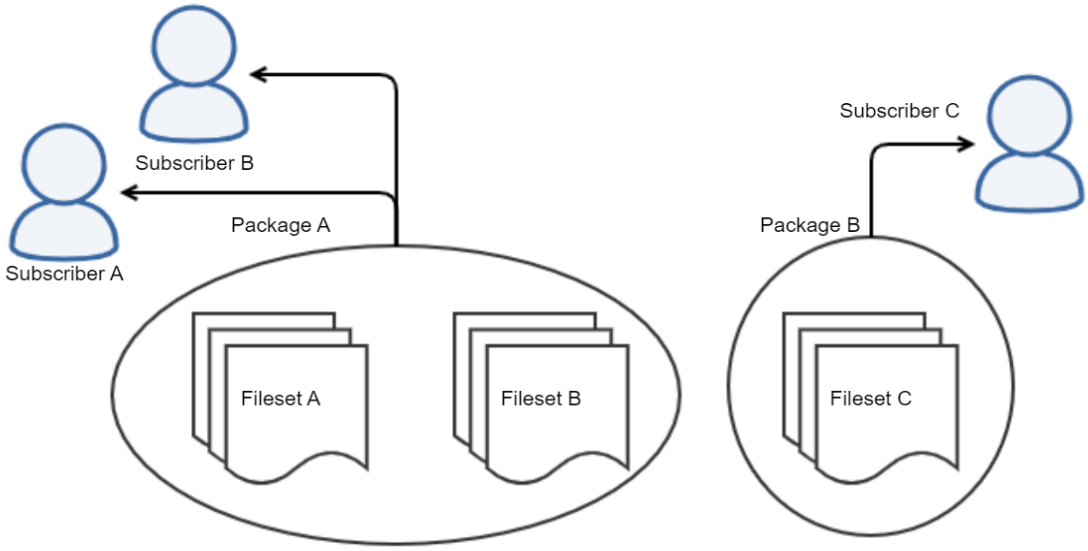
3.2.3 Grant CFS permission to your storage

Non-public S3 buckets need to define a user IAM role/policy to allow CFS to access the actual files. Please reference the Grant Access Permission for CFS V1 guide under the Appendix section.

3.3 Best Practices

3.3.1 Packages

Best practices for CFS Package, Package as an indivisible set of file-sets that are all delivered together or grouping of file-sets. Publisher may group multiple related file-sets which could belong to different buckets into the same package. The subscriber who has sufficient claim permission for the package will be able to access those file-sets and files. A single file-sets also could be shared across multiple packages. When create the package, package name cannot duplicate with existing packages in CFS. However, publisher can change the package name after created.



3.3.2 Buckets

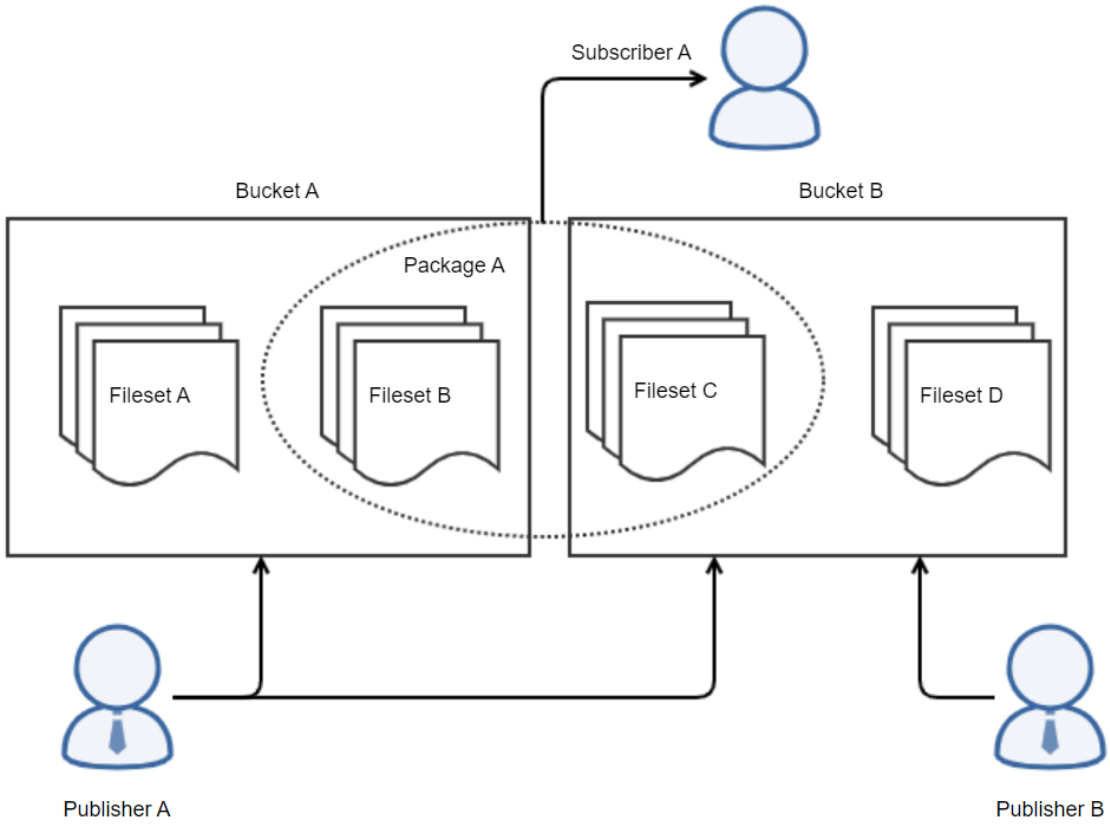
Best practices for CFS Bucket treat them as a collection of Filesets. Buckets are coherent to publisher perspective, whereas packages to subscribers. Each bucket has 'Bucket Writers' to define who has permission to add, modify or remove Filesets from that Bucket. When create the bucket, bucket name cannot duplicate with existing buckets in CFS, but Bucket writers can change the bucket name after it was created.

3.3.3 Files and File-sets

Best practices for CFS treats a file-set as an indivisible set of files that are all delivered together. One may subscribe to a file-set and when the data is ready, the subscriber will, without selecting individual files in the set, download all items in the file-set. File-sets will often be composed of a single file. Some cases where there may be multiple files in a file-set would be because of the file size or because two files are of different types that are co-dependent.
One may think that the *file-set* and *files* endpoint can be used as a traditional hierarchical structure, where arbitrary files can be put into a file-set as an organizational structure. However, while this does meet the requirements for a few use cases, in most circumstances, this is not best practice because of the interplay with other services, such as alerting.
Published file-sets requiring changes or corrections should not be amended. Instead, the file-set should be marked **CANCELLED** or **DELETING** or be made unavailable by specifying an expired AvailableTo date constraint, and then replaced with a new file-set.

3.3.4 Defining Attributes

Attributes allow a Publisher to define content-specific key/value pairs that can help define the specific data within a file-set. They can also be used by Subscribers to search CFS for specific entries.
The Publisher provides the list of attribute names (keys) in the bucket. These names are then used to validate the items allocated to a file-set for consistency. The key/value pairs are then applied to file-sets and can be used as query parameters when searching file-sets.



The values provided by the Publisher are not validated by CFS. They are free-form strings and it is the Publisher's responsibility to ensure consistency and accuracy.

Once the attribute has been defined on the bucket, it cannot be renamed or removed as they are dependent to file-sets attributes.

Bucket Attributes	Fileset Attributes
attributes: DayOfWeek	`{ attribute: DayOfWeek, value: Monday}`
attributes: CatalogSet, Issue	`{ attribute: CatalogSet, value: 12},` `{ attribute: Issue, value: 3}`

3.3.5 Handling Replica Lag

File Distribution stores data into AWS RDS service. File Distribution takes advantage of the cluster replication concept to enhance read latency and workload capacity.

This advantage came with a drawback. There are 20-50 milliseconds lags for a write (create/update) operation that will be replicated completely to reader instances.

When the publisher tries to create an item (POST) then attempt to retrieve it (GET) immediately, publishers may receive response with 404-Not Found error due to RDS replica lag.

To prevent this issue. We recommended the below topics.

- If you need to perform validation against newly created item, use the response from POST/PUT request instead of submitting another GET request to CFS again.
- We recommended slowing down the operation between writing and reading at least 100 milliseconds.

3.4 Publisher Quick Start

3.4.1 Create a CFS Bucket

Publisher can create Buckets to store collection of filesets. The Bucket metadata information can be browsed by all publishers and subscribers regardless of the user claims.

When creating bucket, publisher need to specify list of users (including himself) to grant write access into the writeAccessUser list. Only the user who have the write access to assigned bucket can update or delete the bucket.

The Bucket metadata information can be browsed by all publishers and subscribers regardless of the user's claims, but publisher can optionally set the bucket to be restricted and make it hidden from search result. Bucket's publisher information will also be hidden when the user try to fetch individual bucket information.

Make a ***POST*** request to: *<Environment URL>/file-store/v1/buckets* to create a bucket. The sample payload is similar to the following:

Bucket API: Request Example

```
{
  "name": "TestBucketName",
  "attributes": [
    "DayOfWeek"
  ],
  "availableFrom": "2019-09-28T00:25:04Z",
  "availableTo": "2022-08-28T00:25:04Z",
  "description": "test bucket",
  "publisherName": "Griffin Team",
  "writeAccessUsers": [
        "AAA-user-Id"
  ],
  "restricted": false
}
```

After request to create a bucket, you will get a response to the following:

Bucket API: Response Example


```
{
    "name": "TestBucketName",
    "attributes": [
        "DayOfWeek"
    ],
    "availableFrom": "2020-04-01T08:24:51Z",
    "availableTo": "2022-08-28T00:25:04Z",
    "publisherName": "Griffin Team",
    "description": "test bucket",
    "created": "2020-04-01T08:24:51Z",
    "modified": "2020-04-01T08:24:51Z",
    "writeAccessUsers": [
        "AAA-user-Id"
    ],
    "createdBy": "AAA-user-Id",
    "modifiedBy": "AAA-user-Id",
    "restricted": false,
    "private": false
}
```


3.4.2 Create a CFS Package

CFS uses **Packages** to group logically related file-sets together. When creating a Packages, Publisher can assign a package name, package type, contact email and defines the claims for it.

When creating packages, claims can be specified. A subscriber must have access to the bucket and needs to contain all of the claims of the package to be able to get access to the package.

Make a ***POST*** request to: *<Environment URL>/file-store/v1/packages* to create a package. The sample payload is similar to the following:

Package API: Request Example

```
{
    "packageName" : "CFSPackages",
    "packageType" : "core",
    "description" : "CFS Packages API"
    "contactEmail" : "cfs.admin@refinitiv.com",
    "claims": [
        {
            "type": "CLAIM_TYPE",
            "value": CLAIM_VALUE
        }
    ],
    "writeAccessUsers": [
        "AAA-user-Id"
    ]
}
```

After request to create a package, you will get a response to the following:

Package API: Response Example

```
{
    "packageId": "4014-463a-3a9e15d4-b2a4-58c562da2e5b",
    "packageName": "CFSPackages",
    "description" : "CFS Packages API"
    "contactEmail": "cfs.admin@refinitiv.com",
    "claims": [
        {
            "type": "CLAIM_TYPE",
            "value": CLAIM_VALUE
        }
    ],
    "writeAccessUsers": [
        "AAA-user-Id"
    ],
    "created": "2020-03-26T06:53:42Z",
    "createdBy": "AAA-user-Id",
    "modified": "2020-03-26T06:53:42Z",
    "modifiedBy": "AAA-user-Id",
    "packageType": "core"
}
```

3.4.3 Create CFS File-Set and File

A. Create File-Set and File Individually

A.1. Create a CFS File-Set

A Fileset in CFS contain a file or list of files that group together to be delivered to consumers. Each Fileset is associated with a CFS Bucket and CFS Package

- **Bucket**: describe where the Fileset is belong to. User with the respective Bucket write access will have permission to **publish** new file-set.
- **Package**: collection of Filesets to be deliver to **consumer**. Access permission is determined by the Package's claims.

Make a ***POST*** request to: *<Environment URL>/file-store/v1/file-sets* to create a fileset. The sample payload is similar to the following:

Fileset API: Request Example

```
{
    "name": "CFS-Fileset",
    "bucketName": "TestBucket",
    "attributes": [
        {
            "name": "DayOfWeek",
            "value": "Monday"
        }
    ],
    "packageId": "4787-98ae-87213d3e-a768-3295e065f480",
    "availableFrom": "2020-09-28T00:25:04.123Z",
    "availableTo": "2020-10-31T17:26:51.123Z",
    "contentFrom": "1970-01-01T00:00:00.123Z",
    "contentTo": "2020-04-16T08:41:43.123Z"
}
```

After request to create a package, you will get a response to the following:

Fileset API: Response Example


```
{
    "id": "4609-2a08-e542bfe7-a370-38c9bb26575e",
    "name": "CFS-Fileset",
    "bucketName": "TestBucket",
    "packageId": "4787-98ae-87213d3e-a768-3295e065f480",
    "attributes": [
        {
            "name": "DayOfWeek",
            "value": "Monday"
        }
    ],
    "files": [],
    "numFiles": 0,
    "contentFrom": "1970-01-01T00:00:00Z",
    "contentTo": "2020-04-16T08:41:43Z",
    "availableFrom": "2020-09-28T00:25:04Z",
    "availableTo": "2020-10-31T17:26:51Z",
    "status": "PENDING",
    "created": "2020-07-07T09:04:56Z",
    "modified": "2020-07-07T09:04:56Z",
    "createdBy": "AAA-user-Id",
    "modifiedBy": "AAA-user-Id"
}
```


A.2. Making a Fileset to be available to consumer

When a Fileset is created, it will only be visible to the publisher until its **status** is marked as **"READY"**

A Fileset status can be update to READY when following conditions is satisfied:

- At least one File is contained in the Fileset. (See Creating a CFS File for detail)
- The Fileset availability time frame is valid (**availableFrom** < current time < **availableTo**)

Make a ***POST*** request to: <*Environment URL>/file-store/v1/file-sets/{fileset_id}/status/ready* with *empty request body* to publish a Fileset to consumer.

If Fileset is published successfully, it should return empty response with http code 204 (an error will return if the publish request is not successful).

A.3. Create a CFS File

Files are attached to file-sets. In general, the only form of searching will be used on files would be by filesetId to find all Files for a File-Set. When creating a file, claims are not specified.

A.3.1. [POST] Create File

Make a ***POST*** request to: <*Environment URL>/file-store/v1/*files to create a file. The sample request/response as shown below

Create File API: Request Example

```
{
    "filename": "test cfs file",
    "filesetId": "452a-7a51-1dc3ee4d-94cb-3304413ee170",
    "description": "demo test cfs file",
    "storageLocation": {
        "@type": "s3",
        "url": "https://s3.amazonaws.com/bucket/key.json",
        "rolearn": "arn:aws:iam::xxxxxxxx:role/Role"
    },
    "fileSizeInBytes": 999,
    "fileType": "testfile"
}
```

After request, it'll return response as shown below

Create File: Response Example

```
{
    "id": "4446-fd91-400a7e26-9288-99442e1a6191",
    "filename": "test cfs file",
    "filesetId": "452a-7a51-1dc3ee4d-94cb-3304413ee170",
    "fileType": "testfile",
    "description": "demo test cfs file",
    "storageLocation": {
        "url": "https://s3.amazonaws.com/bucket/key.json",
        "rolearn": "arn:aws:iam::xxxxxxxx:role/Role"
        "@type": "s3"
    },
    "createDateTime": "2020-03-31T20:47:40Z",
    "updateDateTime": "2020-03-31T20:47:40Z",
    "href": "http://localhost:8080/file-store/v1/files/4446-fd91-400a7e26-9288-
99442e1a6191/stream",
    "fileSizeInBytes": 999,
    "createdBy": "AAA-user-Id",
    "modifiedBy": "AAA-user-Id"
}
```

A.3.2. [GET] Get File by Id

Make a ***GET*** request to : *<Environment URL>/file-store/v1/files/{id}* to get the file metadata. The sample request/response as shown below

Get File API: Request Example

```
<Environment URL>/file-store/v1/files/4446-fd91-400a7e26-9288-99442e1a6191
```

After request, it'll return response as shown below

Get File: Response Example

```
{
    "id": "4446-fd91-400a7e26-9288-99442e1a6191",
    "filename": "test cfs file",
    "filesetId": "452a-7a51-1dc3ee4d-94cb-3304413ee170",
    "fileType": "testfile",
    "description": "demo test cfs file",
    "storageLocation": {
        "url": "https://s3.amazonaws.com/bucket/key.json",
        "rolearn": "arn:aws:iam::xxxxxxxx:role/Role"
        "@type": "s3"
    },
    "createDateTime": "2020-03-31T20:47:40Z",
    "updateDateTime": "2020-03-31T20:47:40Z",
    "href": "http://localhost:8080/file-store/v1/files/4446-fd91-400a7e26-9288-
99442e1a6191/stream",
    "fileSizeInBytes": 999,
    "createdBy": "GE-123",
    "modifiedBy": "GE-123"
}
```

A.3.3. [GET] Get All File

Make a ***GET*** request to : *<Environment URL>/file-store/v1/files?filesetId={filesetId}* to create a file. The sample request/response as shown below

Get All File API by filesetId: Request Example

```
<Environment URL>/file-store/v1/files?filesetId=452a-7a51-1dc3ee4d-94cb-3304413ee170
```

Get All File API by createdSince: Request Example

```
<Environment URL>/file-store/v1/files?filesetId=452a-7a51-1dc3ee4d-94cb-
3304413ee170&createdSince=2020-03-29T20:47:00Z
```

Get All File API by modifiedSince: Request Example

```
<Environment URL>/file-store/v1/files?filesetId=452a-7a51-1dc3ee4d-94cb-
3304413ee170&modifiedSince=2020-03-29T20:47:20Z
```

After request, it'll return response as shown below

Get All File: Response Example


```
{
    "value": [
        {
            "id": "4446-fd91-400a7e26-9288-99442e1a6191",
            "filename": "test cfs file",
            "filesetId": "452a-7a51-1dc3ee4d-94cb-3304413ee170",
            "fileType": "testfile",
            "description": "demo test cfs file",
            "storageLocation": {
                "url": "https://s3.amazonaws.com/bucket/key.json",
                "rolearn": "arn:aws:iam::xxxxxxxx:role/Role"
                "@type": "s3"
            },
            "createDateTime": "2020-03-31T20:47:40Z",
            "updateDateTime": "2020-03-31T20:47:40Z",
            "href": "http://localhost:8080/file-store/v1/files/4446-fd91-400a7e26-9288-
99442e1a6191/stream",
            "fileSizeInBytes": 999,
            "createdBy": "AAA-user-Id",
            "modifiedBy": "AAA-user-Id"
        },
        {
            "id": "4a17-a85a-00de4ace-92d6-69a6706d5228",
            "filename": "filesets-1",
            "filesetId": "452a-7a51-1dc3ee4d-94cb-3304413ee170",
            "storageLocation": {
                "url": "https://s3.amazonaws.com/bucket/key.json",
                "@type": "s3"
            },
            "createDateTime": "2020-03-29T18:31:16Z",
            "updateDateTime": "2020-03-29T18:31:16Z",
            "href": "http://localhost:8080/file-store/v1/files/4a17-a85a-00de4ace-92d6-
69a6706d5228/stream",
            "fileSizeInBytes": 999,
            "createdBy": "AAA-user-Id",
            "modifiedBy": "AAA-user-Id"
        }
    ]
}
```

Get All by specify pageSize

Get All File API by pageSize: Request Example

```
<Environment URL>/file-store/v1/files?filesetId=452a-7a51-1dc3ee4d-94cb-
3304413ee170&pageSize=1
```

After request, it'll return response as shown below

Get All File by pageSize: Response return @nextLink

```
{
    "value": [
        {
            "id": "4446-fd91-400a7e26-9288-99442e1a6191",
            "filename": "test cfs file",
            "filesetId": "452a-7a51-1dc3ee4d-94cb-3304413ee170",
            "fileType": "testfile",
            "description": "demo test cfs file",
            "storageLocation": {
                "url": "https://s3.amazonaws.com/bucket/key.json",
                "rolearn": "arn:aws:iam::xxxxxxxx:role/Role",
                "@type": "s3"
            },
            "createDateTime": "2020-03-31T20:47:40Z",
            "updateDateTime": "2020-03-31T20:47:40Z",
            "href": "http://localhost:8080/file-store/v1/files/4446-fd91-400a7e26-9288-
99442e1a6191/stream",
            "fileSizeInBytes": 999,
            "createdBy": "AAA-user-Id",
            "modifiedBy": "AAA-user-Id"
        }
    ],
    "@nextLink": "/file-store/v1/files?pageSize=1&filesetId=452a-7a51-1dc3ee4d-94cb-
3304413ee170&skipToken=ZmlsZUlkPTQ0NDYtZmQ5MS00MDBhN2UyNi05Mjg4LTk5NDQyZTFhNjE5MQ"
}
```

Get All by specify skipToken (please use @nextLink above as urls and if you specify pageSize, it'll fetch one record per page

Get All File API by pageSize: Request Example

```
<Environment URL>/file-store/v1/files?pageSize=1&filesetId=452a-7a51-1dc3ee4d-94cb-
3304413ee170&skipToken=ZmlsZUlkPTQ0NDYtZmQ5MS00MDBhN2UyNi05Mjg4LTk5NDQyZTFhNjE5MQ
```

After request, it'll return response as shown below

Get All File by skipToken: Response return @nextLink for next page

```
{
    "value": [
        {
            "id": "4a17-a85a-00de4ace-92d6-69a6706d5228",
            "filename": "filesets-1",
            "filesetId": "452a-7a51-1dc3ee4d-94cb-3304413ee170",
            "storageLocation": {
                "url": "https://s3.amazonaws.com/bucket/key.json",
                "@type": "s3"
            },
            "createDateTime": "2020-03-29T18:31:16Z",
            "updateDateTime": "2020-03-29T18:31:16Z",
            "href": "http://localhost:8080/file-store/v1/files/4a17-a85a-00de4ace-92d6-
69a6706d5228/stream",
            "fileSizeInBytes": 999,
            "createdBy": "AAA-user-Id",
            "modifiedBy": "AAA-user-Id"
        }
    ],
    "@nextLink": "/file-
store/v1/files?skipToken=ZmlsZUlkPTRhMTctYTg1YS0wMGRlNGFjZS05MmQ2LTY5YTY3MDZkNTIyOA&pageSize
=1&filesetId=452a-7a51-1dc3ee4d-94cb-3304413ee170"
}
```

B. Create File-Set and File with Bulk-Publish request

Rather than Creating a CFS File-Set and Creating a CFS File individually, it is possible to setup multiple CFS file with a new file-set using single API call.

The Bulk-Publish API call support create up to 10 CFS files per request and posted files will be immediately available to subscribers which can significantly reduce number of API call onto CFS for publishing a file.

The CFS Bucket and Package still need to be created as prerequisite for the new file-set to be associated to.

Make a POST request to: <Environment URL>/file-store/v1/bulk-publish to create a file-set. The sample payload is similar to the following:

Bulk-Publish API: Request Example


```
{
    "bucketName": "TestBucket",
    "packageId": "4787-98ae-87213d3e-a768-3295e065f480",
    "filesetName": "CFS-Fileset",
    "attributes": [
        {"name": "DayOfWeek","value": "Monday"}
    ],
    "availableFrom": "2020-09-28T00:25:04.123Z",
    "availableTo": "2020-10-31T17:26:51.123Z",
    "contentFrom": "1970-01-01T00:00:00.123Z",
    "contentTo": "2020-04-16T08:41:43.123Z",
    "files": [
        {
            "filename": "test cfs file",
            "description": "demo test cfs file",
            "fileType": "file",
            "fileSizeInBytes": 999,
            "storageLocation": {
                "@type": "s3",
                "url": "https://s3.amazonaws.com/bucket/key.json",
                "rolearn": "arn:aws:iam::xxxxxxxx:role/Role"
            }
        }
    ]
}
```


After the bulk-publish request, you will get a response to the following:

Bulk-Publish API: Response Example


```
{
    "id": "4a83-d527-8213526b-97ad-cb6e4f7486fd",
    "name": "CFS-Fileset",
    "bucketName": "TestBucket",
    "packageId": "4787-98ae-87213d3e-a768-3295e065f480",
    "attributes": [
        {"name": "DayOfWeek","value": "Monday"}
    ],
    "files": [
        "409f-40c0-6399e672-88d3-e5fe0b3ad429"
    ],
    "numFiles": 1,
    "availableFrom": "2020-09-28T00:25:04.123Z",
    "availableTo": "2020-10-31T17:26:51.123Z",
    "contentFrom": "1970-01-01T00:00:00.123Z",
    "contentTo": "2020-04-16T08:41:43.123Z",
    "status": "READY",
    "created": "2022-03-08T09:27:41Z",
    "modified": "2022-03-08T09:27:41Z",
    "createdBy": "GEXXXX-12345",
    "modifiedBy": "GEXXXX-12345"
}
```

Make a GET request to: <Environment URL>/file-store/v1/files/{id} to get the file metadata. The sample request/response as shown below

Get FileAPI: Request Example

```
<Environment URL>/file-store/v1/files/409f-40c0-6399e672-88d3-e5fe0b3ad429
```

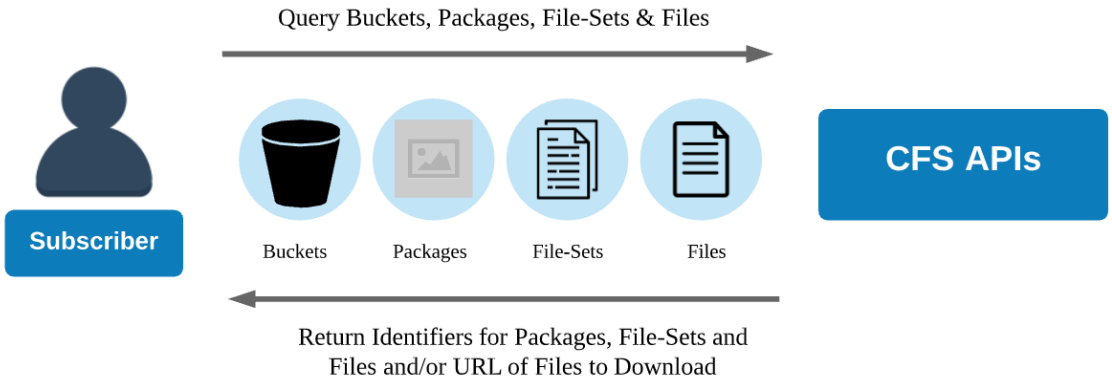
After request, it'll return response as shown below

Get File API: ResponseExample

```
{
    "id": "409f-40c0-6399e672-88d3-e5fe0b3ad429",
    "filename": "test cfs file",
    "filesetId": "4a83-d527-8213526b-97ad-cb6e4f7486fd",
    "fileType": "file",
    "description": "demo test cfs file",
    "storageLocation": {
        "url": "https://s3.amazonaws.com/bucket/key.json",
        "rolearn": "arn:aws:iam::xxxxxxxx:role/Role",
        "@type": "s3"
    },
    "created": "2022-03-08T09:27:41Z",
    "modified": "2022-03-08T09:27:41Z",
    "href": "http://localhost:8080/file-store/v1/files/409f-40c0-6399e672-88d3-
e5fe0b3ad429/stream",
    "fileSizeInBytes": 999,
    "createdBy": "test",
    "modifiedBy": "test"
}
```

4 Subscriber

4.1 Subscriber Workflows



Subscribers use CFS to find and download content provided by Publishers. A limited understanding of attributes and claims are necessary to successfully use CFS. For Subscribers, attributes are mechanisms to find content and claims are used to restrict the content that they can see.

This chapter provides an overview of the available calls for finding buckets, file-sets, and files, and for downloading files. Each of these calls assumes that the Subscriber has access to the Authorization Token (see page 6) that must be sent on each request.

Note that queries may result in more objects than can be returned in one response. By default, a query against file-sets and files can return up to 25 objects. If the request generates more than 25 objects, the results will include a ***nextLink*** to indicate that more data is available. If the results have not yielded the desired item, click the URI specified in the ***nextLink*** to retrieve the next page of data. The ***nextLink*** continues to appear until all results have been returned.

If you want to change the 25 object default, you can do so by adding at the end of the GET File-Sets or Files URL***?pageSize=N***, where ***N*** is the numbers of items to return on a page (each page will contain N results, the last one may contain less). Queries that do not return any data can indicate that no items matching the criteria exist, or that the Subscriber's account is not set up or permissioned for the requested items.

Finding Buckets

- Make a ***GET*** request to: ***<Environment URI>/file-store/v1/buckets?{****params****}***

  Subscriber can freely browse and search over buckets in CFS.

Finding Packages

- Make a ***GET*** request to: ***<Environment URL>/file-store/v1/packages?:****parameters*

  Subscriber can browse and filter list of packages under each bucket.

Finding File-sets

- Make a ***GET*** request to: ***<Environment URI>/file-store/v1/file-sets?{*params*}***

  Subscriber can browse and search over file-sets under each bucket that they have access to. However, Subscribers can see only file-sets that they have access claim permission and such file-sets must be marked as READY by the Publishers too.

Finding and downloading Files

- Make a ***GET*** request to: ***<Environment URI>/file-store/v1/files?{*params*}***

  Subscriber can browse and filter list of files under each authorized file-sets.

- Make a ***GET*** request to: ***<Environment URI>/file-store/v1/files/{*id*}/stream***

  CFS can redirect the Subscriber to download the actual file from the Publisher.

4.2 Subscriber Best Practices

4.2.1 Checking new available File-Set

Many of subscribers to CFS content set usually need to continuously check if there is a new File-Set available in CFS. This document aiming to provide the recommended solution on how to check for new File-Set efficiently which will benefit both CFS Subscriber and CFS system itself.

4.2.1.1 Interval Polling by using the modifiedSince

In order to see a list of File-Set, all subscribers need to call file-set API (/file-store/v1/file-sets) and specified the CFS Bucket Name that they interested.

The recommended way to continuously checking for new File-Set is to specify 2 parameters in the request

- **bucket**: This is a mandatory parameter to identify content test that subscriber interest
- **modifiedSince**: This is the parameter to limit the returned File-Set only for the File-Set that has been modified after a specified time.

Base Fileset API

```
<Environment URL>/file-store/v1/file-sets?bucket=<bucketname>&modifiedSince=<datetime>
```

For example, if subscriber query data from Bucket "STARMINE_PREDICTIVE_ANALYTICS_ARM_AMERS" last check time at "2022-01-26T00:00:00Z" GMT. Assuming check done in approximate 10 minutes interval. Subscribers should record the current time that they checking for new File-Set and using that time in modifiedSince parameters in the next call.

First Polling @ 00:10:30 GMT

```
https://api.refinitiv.com/file-store/v1/file-sets?bucket=STARMINE_PREDICTIVE_ANALYTICS_ARM_AMERS&modifiedSince=2022-01-26T00:00:00Z
```

Second polling

- The first polling is done at 00:10:30 GMT
- In the second polling 00:10:30 GMT should be used as modifiedSince parameters in the API call

Second Polling @ 00:21:02 GMT

```
https://api.refinitiv.com/file-store/v1/file-sets?bucket=STARMINE PREDICTIVE ANALYTICS ARM AMERS&modifiedSince=2022-01-26T00:10:30Z
```

Third polling

- The second polling is done at 00:21:02 GMT
- In the third polling 00:21:02 GMT should be used as modifiedSince parameters in the API call

Third Polling @ 10:31 GMT

```
https://api.refinitiv.com/file-store/v1/file-sets?bucket=STARMINE PREDICTIVE ANALYTICS ARM AMERS&modifiedSince=2022-01-26T00:21:02Z
```

Benefit for this solution

- Subscriber does not need the logic to determine if the FileSet returned from the response are new or not.
- Reduce the number of API call to just one for each polling interval check

4.2.1.2 Using File Notification via AWS SQS

CFS offer an alternative way to be notified when the new File-Set available through SQS. Instead of polling request to file-set API. Subscriber could continuously monitor the queue and received all required information (including file id) and able to download new file directly through file stream API.

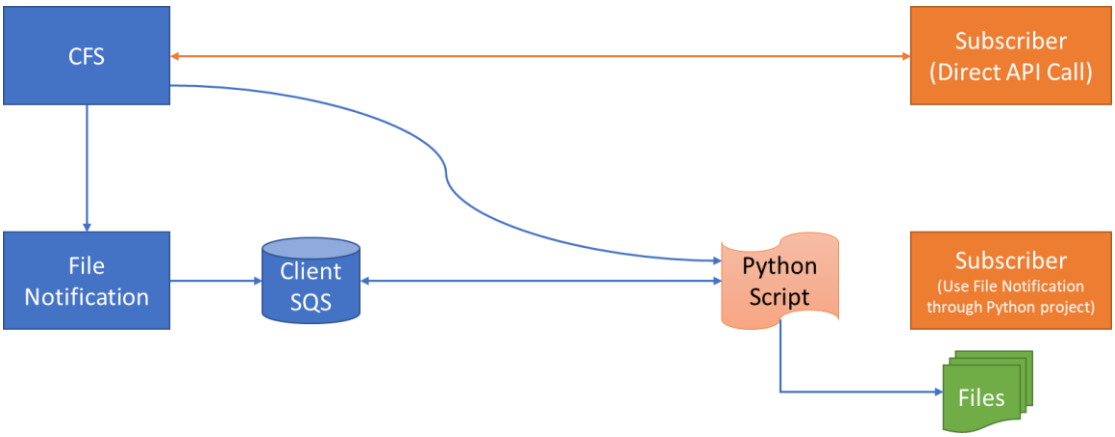
You can follow the instructions below to subscribes File Notification.

1. Subscribers make a subscription to API **/message-services/v1/file-store/subscriptions** by specified CFS content that they interest. E.g., Bucket Name, Package Id, Fileset Attribute.
2. Subscribers received
   a. Subscription ID
   b. SQS Endpoint
   c. Decryption Key
3. Subscribers request credential to access the provided SQS.
4. Subscribers continually checking the SQS in short interval and as soon as the new FileSet that matched with the criteria available message will be available in user specific SQS
5. Fetch message from SQS and decrypt the message to see FileSet / File detail
6. Call CFS API to generate the pre-signed URL to download the file.

CFS Developer also provide the full function python script to do all the above steps and help download the file to specified location automatically. The code is available on https://github.com/Refinitiv-API-Samples/Research.Message.Distribution.Tools

Subscriber could

- Use this python as is to use File Notification and download the new file automatically to the specified folder
- Or use the sample code in the project to incorporate with subscriber's workflow



Benefit for this solution

- Subscriber does not need the logic to determine if the File-Set returned from the response are new or not.

- Immediately when new File-Set available, subscriber will be notified to SQS which will much shorter the time between publication and download.

4.3 Subscriber Quick Started

4.3.1 Finding Buckets

As a Subscriber, you will not need to find buckets, as the names of the buckets that contain your content will be provided to you by the Publishers.

However, Subscribers can perform 2 actions for finding buckets information:

- Bucket Lookup
- Get Individual Bucket Information

4.3.1.1 Bucket Lookup

Subscribers can perform a search to get a list of buckets. Subscribers will not see the restricted buckets in the search result if they are not listed on the bucket's write user access. Also, information of bucket's Publisher will only be available to those defined under the bucket's write user access.

Make a ***GET*** request to: ***<Environment URL>/file-store/v1/buckets?:****parameters* to perform bucket lookup.

The following **optional parameters** can be provided to filter the search result:

- *name* - The name of the bucket. Partially matched results are returned.
- *createdSince* - Return all buckets that have a created date after the specified Datetime.
- *modifiedSince* - Return all buckets that have a modified date after the specified Datetime.
- *availableFrom* - Return all buckets that become visible to permissioned users after the specified Datetime.
- *availableTo* – Return all buckets that is no longer visible to permissioned user after the specified Datetime.
- *attributes* – Return a list of publisher-defined attributes of the buckets.
- *pageSize* - The number of buckets that will be shown on one page. Default value is 25.
- *skipToken* - A token to retrieve the next set of result that exceeds page size.

The bucket list result will be paginated and controlled by *pageSize* and *skipToken* parameter. If there are more matching bucket results, the ***@nextLink*** will be included to retrieve the proceeding result list.

Below is an example of the response of the corresponding bucket result.

Bucket API: Response Example


```
{
    "value": [
        {
            "name": "test-bucket-name-de6f3acd-2fd4-41cc-900a-7de149f09058",
            "attributes": [
                "AttributeName1"
            ],
            "availableFrom": "2020-04-01T04:45:24Z",
            "availableTo": "2020-04-26T04:45:24Z",
            "publisherName": "publisher name based on user",
            "description": "appval-generated test bucket",
            "restricted": true
        }
    ],
    "@nextLink": "/file-
store/v1/buckets?skipToken=YnVja2V0SWQ9NDcwNS03YWQ5LWM5OWE5NWI0LTgwNzQtMDM2Mzg3YzdjMDE0"
}
```


4.3.1.2 Get Individual Bucket Information

Subscribers can retrieve information of each individual bucket by specifying the bucket name. Restricted buckets can be accessed by any user with this method. Like bucket lookup, information of bucket's Publisher will only be available to those defined under the bucket's write user access.

Make a ***GET*** request to: ***<Environment URL>/file-store/v1/buckets/:****bucketName* to get the bucket information.

If the bucket is found, you will get a response like the following example.

Bucket API: Response Example


```
{
    "name": "test-cfs-griffin1",
    "attributes": [
        "DayOfWeek"
    ],
    "availableFrom": "2020-04-19T00:25:04Z",
    "availableTo": "2022-08-28T00:25:04Z",
    "publisherName": "Griffin Team",
    "description": "test bucket",
    "restricted": false
}
```


4.3.2 Finding Packages

Subscribers can perform 2 actions for finding packages information:

Packages Lookup

Get Individual Package Information

4.3.2.1 Packages Lookup

Subscribers can perform a search to get a list of packages.

Make a ***GET*** request to: ***<Environment URL>/file-store/v1/packages?:****parameters* to perform packages lookup.

The following **optional parameters** can be provided to filter the search result:

- *packageName* - The name of the package. Partially matched results are returned.
- *packageType* - Return all packages that match the specified package type.
- *bucketName* - Return all packages that are associated with the specified bucket name.
- *page* - Filter results by a specific pagination index (If client has already specified this query parameter, the skipToken logic will be excluded)
- *includedTotalResult* - The total search result will be counting and added to the first response message.
- *skipToken* - A token to retrieve the next set of result that exceeds page size.
- *pageSize* - The number of packages that will be shown on one page. Default value is 25.
- *includedEntitilementResult* - CFS will perform a permission check on each package against the client permission.

The packages list result will be paginated and controlled by *page*, *pageSize*, and *skipToken* parameter. If there are more matching package results and there is no "*page*" specified on the query parameter, the ***@nextLink*** will be included to retrieve the proceeding result list.

Below is an example of the response of the corresponding package result.

Package API: Response Example

```
{
    "value": [
        {
            "packageId": "4809-f767-51fa80c1-83c7-70a500692c4b",
            "packageName": "OATTestingPackage",
            "contactEmail": "example@refinitiv.com",
            "created": "2020-06-16T12:04:00Z",
            "modified": "2020-06-16T12:04:00Z",
            "packageType": "bulk",
            "bucketNames": []
        }
    ]
}
```

4.3.2.2 Get Individual Package Information

Subscribers can retrieve information of each individual package by specifying the package ID.

Restricted packages can be accessed by any user with this method. Like package lookup, information of package's Publisher will only be available to those defined under the package's write user access.

Make a ***GET*** request to: ***<Environment URL>/file-store/v1/packages/:****packageId* to get the package information.

If the package is found, you will get a response like the following example.

Package API: Response Example

```
{
"packageId": "4809-f767-51fa80c1-83c7-70a500692c4b",
"packageName": "OATTestingPackage",
"contactEmail": "example@refinitiv.com",
"created": "2020-06-16T12:04:00Z",
"modified": "2020-06-16T12:04:00Z",
"packageType": "bulk",
"bucketNames": []
}
```

4.3.3 Finding File-Sets

Subscribers can perform 2 actions for finding file-sets information:

Get File-set from Specific ID

Search File-set by Criteria

4.3.3.1 Get File-set from Specific ID

Make a ***GET*** request to: ***<Environment URL>/file-store/v1/filesets/{****filesetId****}*** to get metadata about specific file-set.

The response will also briefly list files under the file-set (limited to 10 file IDs)

Below is an example of the response of the corresponding file-set result.

Fileset API: Response Example


```
{
    "id": "4a0a-9aa6-3f314d21-985c-a1060479f3a9",
    "name": "test_mxrhjwojntlsohrspuux",
    "bucketName": "test_wgvoemcuhfpuvtdeqxqj",
    "packageId": "0001-0001-00000001-0001-000000000001",
    "attributes": [],
    "files": [
        "4000-1fc6-35996201-a07f-766acd9cb7a5",
        "4000-2880-e22a48a9-953e-ee48e0c46ef5",
        "4000-2ecc-bf251408-bc0e-c5369c0e253a",
        "4000-3c93-3f81a647-9ede-c665675b31a0",
        "4000-4024-11a5af31-9f17-140412ccbe39",
        "4000-438e-4e422755-aad7-509fa365ad7b",
        "4000-4ba2-98cf458d-972d-aed504b46a4e",
        "4000-504e-a65ffead-ba16-bbf30d9d68f5",
        "4000-53c6-462dd4ed-8cff-e463136e35d0",
        "4000-8e1d-350b9385-8fa0-811ae33e0ef2"
    ],
    "numFiles": 82,
    "contentFrom": "1970-01-01T00:00:00Z",
    "contentTo": "2020-04-16T08:41:43Z",
    "availableFrom": "2020-09-28T00:25:04Z",
    "availableTo": "2020-10-31T17:26:51Z",
    "status": "READY"
}
```


4.3.3.2 Search File-set by Criteria

Subscribers can perform a search to get all file-sets matching the specified query parameters.

Make a ***GET*** request to: ***<Environment URL>/file-store/v1/filesets?:****parameters* to perform file-set lookup.

The following **optional parameters** (besides '*bucket*' parameter) can be provided to filter the search result:

- ***bucket*** - **(required)** The name of the bucket for file-sets to be searched. Only exactly matched results are returned.
- *name* - The name of the file-set. Only exactly matched results are returned.
- *packageId* - Package ID
- *status* - Filter file-set by status (Ready/Pending)
- *availableFrom* - Return all file-sets that become visible to permissioned users after the specified Datetime.
- *availableTo* - Return all file-sets that is no longer visible to permissioned user after the specified Datetime.
- *contentFrom* - Filter results by the age of the content within the file-set.
- *contentTo* - Filter results by the age of the content within the file-set.
- *createdSince* - Return all file-sets that have a created date after the specified Datetime.

- *modifiedSince* - Return all file-sets that have a modified date after the specified Datetime.
- *attributes* - Return a list of publisher-defined attributes of the file-sets.
- *pageSize* – The number of file-sets that will be shown on one page. Default value is 25.
- *skipToken* - A token to retrieve the next set of file-set result that exceeds page size.

The file-set list result will be paginated and controlled by *pageSize* and *skipToken* parameter. If there are more matching file-set results, the ***@nextLink*** will be included to retrieve the proceeding result list.

Below is an example of the response of the corresponding file-set result.

Fileset API: Response Example


```
{
    "value": [
        {
            "id": "4000-3516-ffdcdb12-ac56-199b5d85edb4",
            "name": "test_idrjtixnrtesgfbwxbqt",
            "bucketName": "test_wcdsdecgqhvitqitkfhq",
            "packageId": "0001-0001-00000001-0001-000000000001",
            "attributes": [],
            "files": [],
            "numFiles": 0,
            "contentFrom": "1970-01-01T00:00:00Z",
            "contentTo": "2020-04-16T08:41:43Z",
            "availableFrom": "2020-09-28T00:25:04Z",
            "availableTo": "2020-10-31T17:26:51Z",
            "status": "READY"
        },
        {
            "id": "4000-6d1c-81e0d4f0-955f-b65268de8a26",
            "name": "test_sgvrfqsjppuwdkpuifyt",
            "bucketName": "test_wcdsdecgqhvitqitkfhq",
            "packageId": "0001-0001-00000001-0001-000000000001",
            "attributes": [],
            "files": [],
            "numFiles": 0,
            "contentFrom": "1970-01-01T00:00:00Z",
            "contentTo": "2020-04-16T08:41:43Z",
            "availableFrom": "2020-09-28T00:25:04Z",
            "availableTo": "2020-10-31T17:26:51Z",
            "status": "READY"
        }
    ],
    "@nextLink": "/file-store/v1/file-
sets?bucket=test_wcdsdecgqhvitqitkfhq&pageSize=2&skipToken=ZmlsZXNldElkPTQwMDAtNmQxYy04MWUwZ
DRmMC05NTVmLWI2NTI2OGRlOGEyNg"
}
```

4.3.4 Finding Files and Download Files

Subscribers can perform 3 actions for finding files information and downloading files:

Get File from Specific ID

Search File by Criteria

Download File through CFS API

4.3.4.1 Get File from Specific ID

Make a ***GET*** request to: ***<Environment URL>/file-store/v1/files/{*****id*****}*** to get metadata of a file in CFS.

Below is an example of the response of the corresponding result.

File API: Response Example

```
{
    "id": "4446-fd91-400a7e26-9288-99442e1a6191",
    "filename": "test cfs file",
    "filesetId": "452a-7a51-1dc3ee4d-94cb-3304413ee170",
    "fileType": "testfile",
    "description": "demo test cfs file",
    "storageLocation": {
        "url": "https://s3.amazonaws.com/bucket/key.json",
        "rolearn": "arn:aws:iam::xxxxxxxxxxx:role/Role",
        "@type": "s3"
    },
    "createDateTime": "2020-03-31T20:47:40Z",
    "updateDateTime": "2020-03-31T20:47:40Z",
    "href": "http://localhost:8080/file-store/v1/files/4446-fd91-400a7e26-9288-
99442e1a6191/stream",
    "fileSizeInBytes": 999
}
```

4.3.4.2 Search File by Criteria

Make a ***GET*** request to: ***<Environment URL>/file-store/v1/files?:****parameters* to perform file lookup.

The following **optional parameters** (besides '*filesetId*' parameter) can be provided to filter the search result:

- ***filesetId*** - **(required)** The name of the file-set for files to be searched. Only exactly matched results are returned.
- *createdSince* - Return all files that have a created date after the specified Datetime.
- *modifiedSince* - Return all files that have a modified date after the specified Datetime.
- *pageSize* - The number of files that will be shown on one page. Default value is 25.
- *skipToken* - A token to retrieve the next set of file result that exceeds page size.

The file list result will be paginated and controlled by *pageSize* and *skipToken* parameter. If there are more matching file results, the ***@nextLink*** will be included to retrieve the proceeding result list.

Below is an example of the response of the corresponding files result.

File API: Request Example

```
<Environment URL>/file-store/v1/files?filesetId=4a0a-9aa6-3f314d21-
985c-a1060479f3a9&modifiedSince=2020-05-03T00:00:00Z&pageSize=2
```

File API: Response Example


```
{
    "value": [
        {
            "id": "4000-1fc6-35996201-a07f-766acd9cb7a5",
            "filename": "test_ysvygdrpckipciihtpjz",
            "filesetId": "4a0a-9aa6-3f314d21-985c-a1060479f3a9",
            "fileType": "file",
            "description": "My File Description",
            "storageLocation": {
                "url": "https://s3.amazonaws.com/35be22bc-e4fb-300e-99b9-029c6b3b7809.json",
                "@type": "s3"
            },
            "created": "2020-07-03T05:34:43Z",
            "modified": "2020-07-03T05:34:43Z",
            "href": "http://localhost:8080/file-store/v1/files/4000-1fc6-35996201-a07f-
766acd9cb7a5/stream"
        },
        {
            "id": "4000-2880-e22a48a9-953e-ee48e0c46ef5",
            "filename": "test_tuwpgtpkyewwowdojjdo",
            "filesetId": "4a0a-9aa6-3f314d21-985c-a1060479f3a9",
            "fileType": "file",
            "description": "My File Description",
            "storageLocation": {
                "url": "https://s3.amazonaws.com/35be22bc-e4fb-300e-99b9-029c6b3b7809.json",
                "@type": "s3"
            },
            "created": "2020-07-03T05:47:44Z",
            "modified": "2020-07-03T05:47:44Z",
            "href": "http://localhost:8080/file-store/v1/files/4000-2880-e22a48a9-953e-
ee48e0c46ef5/stream"
        }
    ],
    "@nextLink": "/file-store/v1/files?pageSize=2&filesetId=4a0a-9aa6-3f314d21-985c-
a1060479f3a9&skipToken=U2Vla0lkPTQwMDAtMmVjYy1iZjI1MTQwOC1iYzBlLWM1MzY5YzBlMjUzYQ"
}
```

4.3.4.3 Download File through CFS API

Make a ***GET*** request to: ***<Environment URL>/file-store/v1/files/{*`id`*}/stream*** to retrieve Publisher's file through CFS API.

CFS will generate S3 pre-signed url, then it should automatically redirect and start downloading the file from S3.

If an error message is returned from Amazon, contact Publisher to double-check the following configuration:

- ***url*** under `storageLocation` is currently pointed to existing file on S3.
- S3 bucket is configured to Grant Access Permission for CFS V1.
    - If the S3 bucket and file are granted by Role-based permission, ***roleArn*** under `storageLocation` must be provided.

5 Appendix

5.1 Grant Access Permission for CFS V1

This section intended for new Publisher to grant permission for CFS to be able to access file on Publisher AWS S3 bucket.

5.1.1 CFS Identity

Identity	Environment	Value
Account Id	Pre-Production	854202341060
Account Id	Production	113932484821
Principal	Pre-Production	arn:aws:iam::854202341060:role/a204890-file-dist-rest-api-ecs-taskdef-ppe
Principal	Production	arn:aws:iam::113932484821:role/a204890-file-dist-rest-api-ecs-taskdef-prod

5.1.2 Verify S3 object encryption

There are 6 possible outcomes of object encryption on Amazon S3 when creating objects:

- No encryption
- Encryption using Amazon S3-managed keys using default bucket encryption
- Encryption using Amazon S3-managed keys and specified using the x-amz-server-side-encryption request header
- Encryption using AWS KMS with default bucket encryption
- Encryption using AWS KMS and specifying the customer master key (CMK) in the x-amz-server-side-encryption request header
- Encryption using customer-provided encryption keys

If your object is encrypted with AWS KMS. You will need to grant **"kms:Decrypt"** policy. Which allows CFS to generates S3 pre-signed URL with decrypt permission to your file.

5.1.2.1 Using AWS Console

You can use the AWS Management Console to describes object metadata by following these steps.

1. Sign-in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/ Search S3 object you want to store in CFS.
2. In the **Buckets** list, choose the name of the bucket that contains the object that you want to store in CFS.
3. In the **Objects** list, select the object that you want to store in CFS
4. In the **Properties** page, Check the **Server-Side Encryption**.
5. If **"KMS ARN"** appears, your object is encrypted with AWS KMS.

5.1.2.2 Using AWS Cli

Before you start these instructions, you must install AWS Cli version 2.

Use the following procedure to describes object metadata.

1. Open your local terminal or shell, enter following command to ensure your AWS Cli is version 2.

```
aws --version
```

2. Enter following command to describes S3 object metadata.

```
aws s3api head-object --bucket <your-bucket-name> --key <your-object-path>
```

3. If attribute **"SSEKMSKeyId"** appears, your object is encrypted with AWS KMS.

5.1.3 Grant CFS permission

CFS Support IAM Role-based policy permission granting on CFS API. This also allows the S3 Bucket Owner to control multiple linked resources permission at once.

- Due to current AWS Limitation on role chaining session duration limit, presigned S3 url will only have 1 hour expiration duration for file that access grant via this method.

1. Enter IAM Management Console. Select **Roles**, Then **Create role**.
2. Select **Another AWS account.** Then type CFS account id. And then click **'Next:Permissions'.**
3. AWS Console will navigate you to next screen, click **'Create policy'**.
4. Then you will navigate to new tab, Click **'JSON'.** Then checkout one of policy below.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::<bucket-name>/*"
        }
    ]
}
```

Ex. Once you conclude your policy for CFS, Click **'Preview Policy'.**

5. Naming your policy and description(optional). Then click **'Create Policy'.**
6. Go back to previous tab (before you perform step 3). Click refresh button. Select your created policy. And then click 'Next: Tags'.

Tips: You can use filter policies to find your custom policy under 'Create policy' button.

7. Add your tags (optional), Then click **'Next: Review'.**
8. **Naming your role contains with 'CFS' word.** Then click **'Create role'.**
   We allow the role name using the format below:
   "arn:aws:iam::*:role/*EdsCfsS3Access*" or **"arn:aws:iam::*:role/*CFS*"**
9. After you created a role. AWS will navigate you to roles page. **Find your role, then click on your role name.**
10. Navigate to Trust Relationships, then click **'Edit trust relationship'.**
11. Ensure your json contains STS policy statement below. Then click **'Update Trust Policy'.**

Example STS Policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "<CFS-ARN>"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Warning: Usage of role-based policies required to specific 'roleArn' in 'storageLocation' in the request body through POST /files

Example KMS Policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "kms:Decrypt"
            ]
            "Resource": [
                "arn:aws:s3:::<bucket-name>/*",
                "<KMS-ARN>"
            ]
        }
    ]
}
```

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "kms:Decrypt"
            ]
            "Resource": [
                "arn:aws:s3:::<bucket-name>/*",
                "<KMS-ARN>"
            ]
        }
    ]
}
```

6 Troubleshooting

Problem	Possible Causes	Action
Cannot access file distribution API and get a message "*403 - access denied. Scopes required to access the resource*"	User does not have a permission on Refinitiv API scopes.	• Make sure that the scope "`trapi.cfs.subscriber.read`" is added to the header when requesting an access token. • Contact Help & Support at MyRefinitiv to request AAA team for account permission.
Cannot download file and get a message "*403 - Access denied: User "[x]" cannot access file*"	User does not have a permission on that package claims.	• Check package claim information which is linked to the filesetId • Contact Help & Support at MyRefinitiv to request package owner or Product Manager to grant more permissions.
Cannot make a request to file distribution API and get a message "*429 - Too many requests*"	• User makes numerous requests to file distribution API in a single point of time • Too many requests are injected to file distribution system.	Throttle limit on file distribution API is set based on user request and the whole system. If user needs to make a lot of requests, there are 2 possible options to do: • Add a little delay for each request. • Contact Help & Support at MyRefinitiv for consultation on your use case scenario.
Cannot use a pre-signed URL and get 4xx HTTP response with XML format	AWS S3 service is responses in xml formatted. User might encounter these errors, when using incorrect S3 URL pattern or missing IAM policy for CFS access.	• Make sure you are using the Encoded URL in the “storageLocation.url”. • Validate your IAM policy is correct
Cannot create/update and got a response "409-Conflict" with a message like "duplicate with [x] id: [x]"	• The user tries to create the duplicated object • Data replication lag	• Create with a different name. • Delay the operation at least 100 milliseconds. • Contact Help & Support at MyRefinitiv for consultation on your use case scenario.