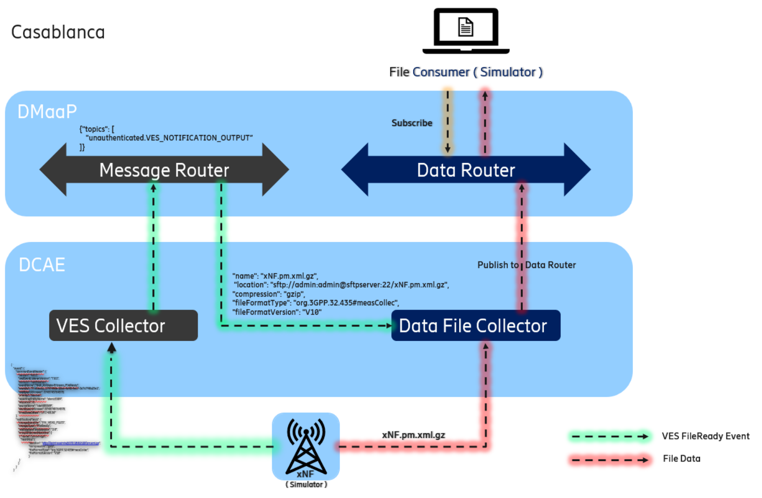
Data-File-Collector

Datafile Collector is responsible for collecting PM counter files from PNF (Physical Network Function) and then publish these files to Dmaap DataRouter.

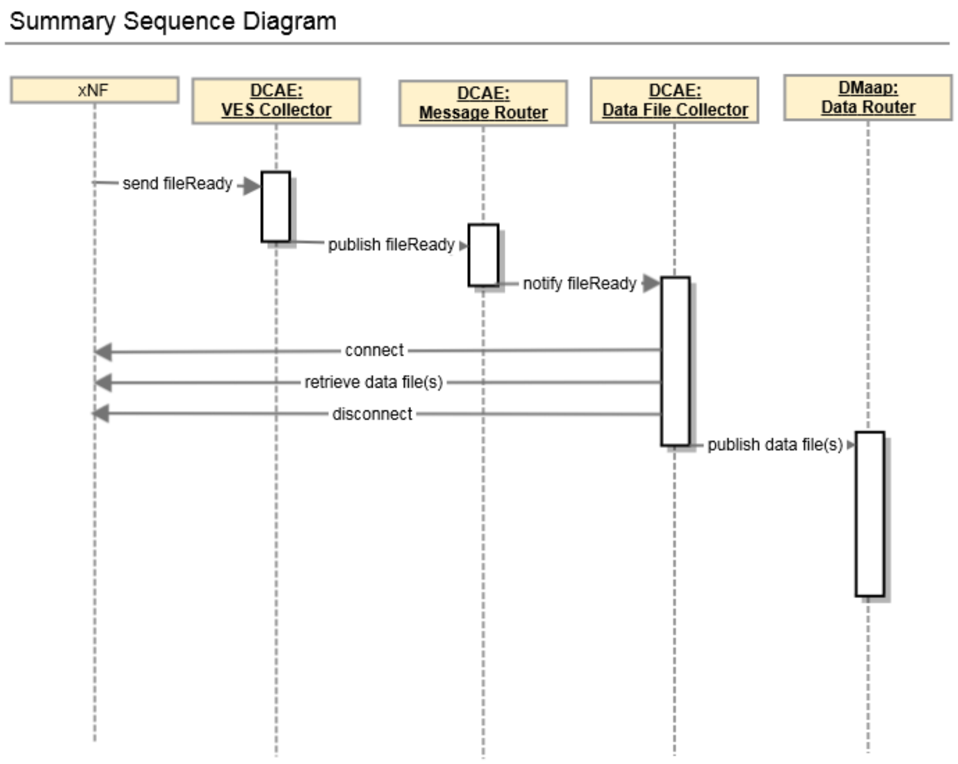


Process:

DataFile Collector (DFC) is a part of ONAP DCAEGEN2.

It handles the collection of data files which are notificated.

- Subscribes VES-Notification Event
    - Only processes pmFileReady Events ("changeType": "FileReady")
    - Skipps and logs other notification events
- Collects the files from the xNF which are linked in the notification FileReady event
- Sends the collected files to DataRouter by push event



Mangement Interfaces:

- Heartbeat: http://<container_address>:8100/heartbeat or https://<container_address>:8443/heartbeat
- Start DFC: http://<container_address>:8100/start or https://<container_address>:8433/start

- Stop DFC: http://<container_address>:8100/stopDatafile or https://<container_address>:8433/stopDatafile

Supported collection ways:

- http
- https
- sftp
- ftp

Retry:

- tries to download the files, if temporarily faults appear
- retries are limited to a configurable number of times
- increasing delay between each attempt
- Finally give up and log error
- Eeach not published file will be published with new files when new events are coming in???

API:

Regarding APIs the file collector can querry file-router, if a file has been published previously: https://docs.onap.org/projects/onap-dcaegen2/en/latest/sections/services/dfc/consumedapis.html

data-router:

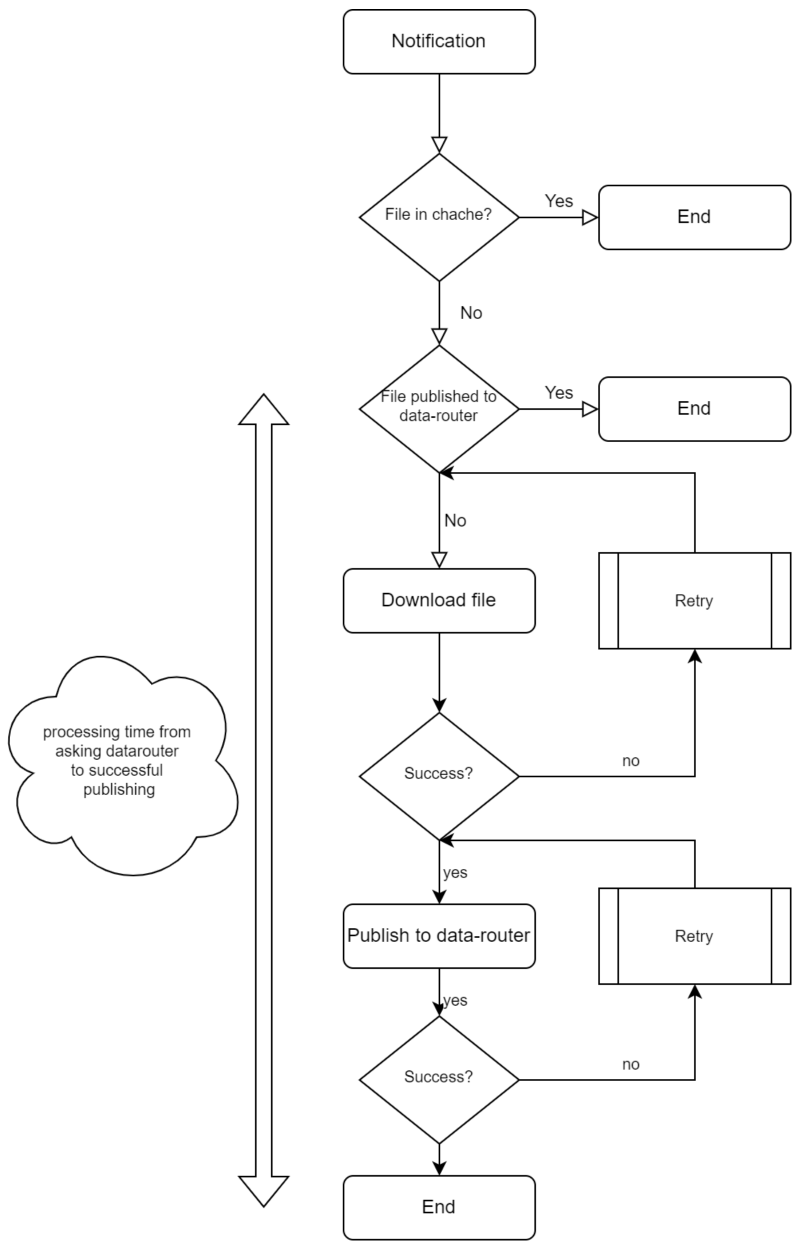
```
GET /feedlog/{feedId}?{queryParam}
```

- queryParam: type=pub&filename=FILENAME

HTTP Code	Body	Description
400	NA	error in query
200	[]	Not published yet
200	[$FILENAME]	Already published

Issues:

- In memory implementation of downloaded files
    - Check if files are published to data-router after in memory cache check
    - **This is not** thread save / **synchronizable over more instances**/pods
    - **Not scalable** because not synchronized cache of already downloaded files
- Restart will lead to building up cache again (asking data-router).
- **Maximum of 200 files are in a thread pool, unless they are not collected, the file collector is blocked.**
    - Happened, when files were older 24h (Nokia storing time) after restart plattform.



Problem with scalability

- Currently the file-collector is running as **single instance** (1 POD)
- The file collector **cannot** be configured to **run multible instances** (PODS)
- If the file-collector would be configurable for multible instances, the current problems will occur:
    - no synchronization about published files threadsafe and distributed (processing time of download and publishing)
    - could be, that more instances will process one file and publish it to datarouter several times.
    - result would be a multiplicated counters which are identical.

Analysis:

```
Flux<FilePublishInformation> createMainTask(Map<String, String> context) {
        return fetchMoreFileReadyMessages() //
            .doOnNext(fileReadyMessage -> threadPoolQueueSize.incrementAndGet()) //
            .doOnNext(fileReadyMessage -> counters.incNoOfReceivedEvents()) //
            .parallel(NUMBER_OF_WORKER_THREADS) // Each FileReadyMessage in a separate thread
            .runOn(scheduler) //
            .doOnNext(fileReadyMessage -> threadPoolQueueSize.decrementAndGet()) //
            .flatMap(fileReadyMessage -> Flux.fromIterable(fileReadyMessage.files())) //
            .flatMap(fileData -> createMdcContext(fileData, context)) //
            .filter(this::isFeedConfigured) //
            .filter(this::shouldBePublished) //
            .doOnNext(fileData -> currentNumberOfTasks.incrementAndGet()) //
            .flatMap(this::fetchFile, false, 1, 1) //
            .flatMap(this::publishToDataRouter, false, 1, 1) //
            .doOnNext(publishInfo -> deleteFile(publishInfo.getInternalLocation(), publishInfo.getContext())) //
            .doOnNext(publishInfo -> currentNumberOfTasks.decrementAndGet()) //
            .sequential();
}

private boolean shouldBePublished(FileDataWithContext fileData) {
        Path localFilePath = fileData.fileData.getLocalFilePath();
        boolean shouldBePublished = (publishedFilesCache.put(localFilePath) == null);
        if (shouldBePublished) {
            shouldBePublished = checkIfFileIsNotPublishedInDataRouter(fileData);
        }

        if (!shouldBePublished) {
            logger.debug("File: {} is being processed or was already published. Skipping.", fileData.fileData.
name());
        }
        return shouldBePublished;
}
```

The synchronization problem when having multible instances of file-collector can be found in this code block:

- *.filter(this::shouldBePublished)* <Line 11>: This method puts the file into local memory cache. This cache is only on one instance.
    - If not in local chache, the implementation asks the data-router, if file has been already published. This takes time
    - In this time, on another instance this could happen with another notification message, containing the same file to collect
    - Then both instances will put it to local cache and ask datarouter. Both will get the same answer
- The time period between asking data-router and *.doOnNext(publishInfo -> deleteFile(publishInfo.getInternalLocation(), publishInfo.getContext()))* <Line 15> is critical.
    - This has to be synchronized over the different instances

Has someting like this happened in the single instance even after restart?

Every file has only once been published to data-router

This is the log from the data-router:

```
MariaDB [datarouter]> select count(*) as count, FILENAME from log_records group by FILENAME order by count desc;
+-------+------------------------------------------------------------------------------+
| count | FILENAME                                                                     |
+-------+------------------------------------------------------------------------------+
|  1596 | NULL                                                                         |
|     1 | PM202209280145+020024C20220928.0130+0200-20220928.0145+0200_MRBTS=999965.xml.gz |
|     1 | PM202209270400+020024C20220927.0300+0200-20220927.0400+0200_MRBTS=515027.xml.gz |
|     1 | PM202209271545+020024C20220927.1530+0200-20220927.1545+0200_MRBTS=999965.xml.gz |
|     1 | PM202209262245+020024C20220926.2230+0200-20220926.2245+0200_MRBTS=999965.xml.gz |
|     1 | PM202209280330+020024C20220928.0315+0200-20220928.0330+0200_MRBTS=999965.xml.gz |
|     1 | PM202209261045+020024C20220926.1030+0200-20220926.1045+0200_MRBTS=515027.xml.gz |
|     1 | PM202209271745+020024C20220927.1730+0200-20220927.1745+0200_MRBTS=999965.xml.gz |
|     1 | PM202209261645+020024C20220926.1630+0200-20220926.1645+0200_MRBTS=515027.xml.gz |
|     1 | PM202209261400+020024C20220926.1300+0200-20220926.1400+0200_MRBTS=515027.xml.gz |
|     1 | PM202209280530+020024C20220928.0515+0200-20220928.0530+0200_MRBTS=999965.xml.gz |
|     1 | PM202209261200+020024C20220926.1100+0200-20220926.1200+0200_MRBTS=999965.xml.gz |
|     1 | PM202209271945+020024C20220927.1930+0200-20220927.1945+0200_MRBTS=999965.xml.gz |
|     1 | PM202209261000+020024C20220926.0900+0200-20220926.1000+0200_MRBTS=515027.xml.gz |
|     1 | PM202209270900+020024C20220927.0800+0200-20220927.0900+0200_MRBTS=515027.xml.gz |
```

Conclusion:

There seems to be no problem when running the file-collector as single instance.
Running on scaled **multi-instances will lead to unpredictable side effects**.