Bulk Data Import Specification Proposal

First and foremost, a new proposal for Bulk Data Import specification has been written by Dan Gottlieb and reviewed by Josh Mandel, the architect of the SMART project. The import proposal is built on top of already existing Bulk Data export capabilities. It introduces the idea of the data providers “notifying” the data consumers when new data is available using the so called “ping” request (which is why this specification is also called PnP, or ping and pull). Upon receiving such notification, data consumers can use the already existing Bulk Data export capabilities to pull the new data. Consumers should also provide a way for the data providers to keep track of ongoing imports and even to cancel them.

Another new concept introduced in this specification is the idea of static exports. Those are exports that have already been completed and then the manifest file with pointers to the exported data files is sent to the data consumers so that they can just download those files. This is a way to reduce the load on the data providers by eliminating the need to run through the full export cycle every time.

Below is the diagram of the Bulk Data Import flow, as provided by the Bulk Data Import Proposal:
To implement this specification, we first need to determine which parts of this exist already, and what we need to add to the existing ecosystem to make this work. Bulk Data Export Servers and Clients already exist. Here is another version of the same diagram where existing functionality is white and the missing pieces are in orange:
On the **Data Provider** end (which might be an EHR with a bulk export-capable FHIR implementation), developers will have to add some functionality to make sure that:

- The data provider can kick-off new imports, which is a way to notify clients when new data is available
- The data provider can track the progress of current import operations

On the **Data Consumer** end, developers need to add some functionality to make sure that:

- An export is started (using a Bulk Data client) when an import kick-off request is received.
- The status of the currently running exports is available to those who initiated that import.
- There is of course a lot more to be done on the data consumer side, so that the data is ingested and/or stored somewhere, but that is beyond the scope of the Bulk Data Import flow.
Implementation Approach

Given that implementing a Bulk Data Import requires development efforts on both the server providing the data and the server receiving the data, we decided to try to extract the functionality on the data provider side into a component - the Bulk Import Mediator. The Mediator will hopefully minimize the amount of work that needs to be done by developers. Here is a diagram of our solution:

We are using some readily available components and services like HAPI and Amazon S3. Those are used as is and do not require any customizations (which is why they are rendered with white background). That proves that Bulk Data import capabilities can be added on top of existing servers without having to modify them.
Finally, here is a diagram of the different ways we have implemented the Bulk Data export, import and feeding the data into a post-processing pipeline. The red outlines represent what we are currently using for the production Cumulus implementation, and the blue outlines represent the bulk import prototype. To make this work, we had to create several new components highlighted in orange and described below.

**Primary Components**

**Bulk Data Import Server (Data Consumer)**

This server is an experimental prototype implementation of a Data Consumer application, as defined in the Bulk Data Ping and Pull Import Proposal. It is available as an online demo [here](#), but people will have more options if they run it locally.
The server behaves like a Bulk Data Client and consumes bulk-data NDJSON files. The imported files are immediately discarded in the online demo version. When running locally, it can also be configured to temporarily store the files in the filesystem, or to upload them to S3 bucket.

This server comes with an UI for client registration and a key generator. Bulk Data Providers or third party apps (like the Mediator described below) must be registered as clients of this server in order to use its capabilities.

**Links**

- **Online version:** [https://bulk-import-consumer.herokuapp.com/](https://bulk-import-consumer.herokuapp.com/)
- **Source Code:** [https://github.com/smart-on-fhir/bulk-import-consumer](https://github.com/smart-on-fhir/bulk-import-consumer)

**Bulk Data Import Client (Mediator)**

This is a command line app implementing all the necessary functionality for starting and monitoring Bulk Data Imports. Its purpose is to encapsulate all the new functionality (from the data provider side) in a reusable way. Once configured properly, it can start Bulk Data imports, monitor their status and progress, and even cancel them.

**Links**

- **Source Code:** [https://github.com/smart-on-fhir/bulk-import-client](https://github.com/smart-on-fhir/bulk-import-client)
Additional Components

There are some additional components that we had to develop in order to fully test relevant workflows. First, we needed an authentication proxy server to add SMART Backend services to our HAPI server. Second, since at the time of this writing there is no data provider that supports Bulk Import Ping requests, we also wanted to support the use of a standard Bulk Data Client in our pipeline for now, as that removes some of the complexity. These new components are described below.

Bulk Data Client v2

We created a brand new version of our Bulk Data client in order to address some of the integration requirements discovered while trying to use Bulk Data files in a post-processing pipeline. Even though the client is just a tool for downloading NDJSON files, there are a number of additional issues arising when using it against different servers or when the downloaded data needs to be processed further. Below is a list of new features added to the new client to address those issues.

- **Different reporters** - By default the client “reports” everything to the terminal in an user-friendly way. However, if the client is part of some data processing pipeline, its output ends up in various log files which may require different formatting, achieved by using different reporters.

- **Requests customization** - Some Bulk Data servers or EHRs might require special HTTP request headers or other customizations. That is why we made it possible to further customize every request that the Bulk Data client makes.

- **Data validation** - The new client does not blindly trust the incoming data, but runs some basic validations on it. These include:
  - Verifying that every single line in an NDJSON file is valid JSON.
  - Lines are not longer than a certain limit (which might crash the client machine in case of insufficient memory).
- The ResourceType of each resource line matches the expected resource type, as advertised in the export manifest provided by the server.
- The number of resources in every file matches the number specified in the export manifest (if any).

- **Enforce common naming convention** - Every Bulk Data server names the files in slightly different ways. This might be an issue for post processing tools expecting to handle data the same way, regardless of where it comes from. The new client can be configured to rename the downloaded files in a consistent way.

- **Add destination to manifest** - Downloaded files might end up in various locations on the file system or be uploaded to another destination. Since that location is not stored anywhere after the export is completed, any script that tries to process the downloaded data will have to be manually configured to look for it in a certain destination. To automate that we add the destination of each file to the (already downloaded) manifest. Clients use the manifest to find out what to download, but in this way we are also making it possible for other tools to further process the data by using the manifest file as a single source of truth.

- **Support different destinations** - in addition to being able to store downloaded files in the local file system, the new client can also be configured to upload them to AWS S3 bucket, or to arbitrary HTTP endpoint, or even to discard them and not store them anywhere.

- **Saving the manifest** - as mentioned above, it can be beneficial for the post-processing tools to use the manifest file as a starting point, so we make it possible for that file to be stored along with the actual data files. Additionally, the Bulk Data export is a long-running process. Tools may be watching the destination for any new files, but those would arrive in different order over time. We save the manifest file last, when everything else is fully downloaded and written/uploaded to the destination. This way, any tools can just watch for new manifest files as a signal that a new export has just been completed.

- **Attachments processing** - tools that will further process the downloaded Bulk Data may have different requirements and expectations about the data. For example, they may want to process attachments but prefer to find those inline in the DocumentReference resources, rather than loading them
from separate files. The client can be configured to modify the DocumentReference resources and put any attachments inline, given that they have a certain mime type and are below a certain size.

- **PDF to Text** - To make it easier to process clinical notes, the new client can also convert PDF notes to plain text.

**Links**

- **Documentation**: [https://docs.smarthealthit.org/bulk-data-client/](https://docs.smarthealthit.org/bulk-data-client/)

**Authentication Proxy**

Our auth-proxy app is designed to sit in front of an open bulk-data-capable FHIR server. It will intercept all Bulk Data requests, require authentication and then pass the requests up to the actual FHIR server. For simplicity, a pre-configured instance is currently available at [https://bulk-data-auth-proxy.herokuapp.com/](https://bulk-data-auth-proxy.herokuapp.com/). It is configured to work with the Bulk Data server at [http://3.139.7.109:8080/fhir](http://3.139.7.109:8080/fhir), which is a HAPI R4 FHIR server with sample data. This means we now have two versions of that server: