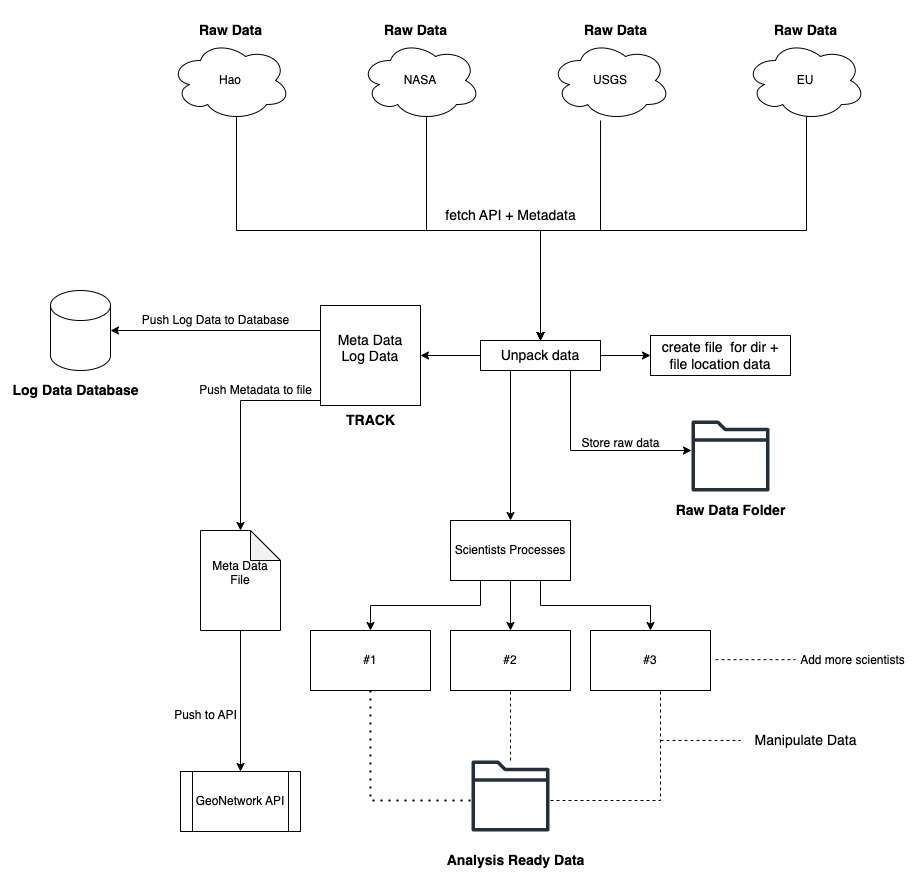
**Data Flow and High-level Architecture Diagram**

**Project Goal:**

Our goal while working with National Resources Canada was to create a system for them to remove unnecessary manual tasks from their work pipeline. To this end, our task was to create a modular automated ETL pipeline, one where we could add new scientists’ workflows as time progressed. This would all be done in order to bring more automation to the workplace processes. Having data ready before the work day begins on a regular schedule allows for research to begin at an earlier stage. This alleviates the current manual processing bottleneck that exists in the current pipeline.



**Explanation**:

Depending on the scientist, data will be grabbed from specific APIS, these include; NASA, USGS, and Copernicus (EU). The type of data each scientist will want is unique and will require different requests, sometimes just one call and others multiple along with specific scripts to have the data set up properly. This data will be formatted and unpacked, at this juncture we will know where this data will come from and go due to its metadata, that must be added. This metadata needs to be set up properly following the HNAP documentation (Brian has access) and then needs to be sent off to the GeoNetorkAPI for storage. We also need to add logging data that will keep track of everything that is happening, and this will be sent to its own MongoDB. The locations of everything and convenient file data will all be stored on a file in a directory housing all-important location data. The final step here will be storing the data in its entirety in a folder to be named and set up according to the proper conventions. This Data will be the unpacked data with its meta and log data to be stored in case something goes wrong with the data transformation section. We now get to the heart of the project, the scientist's processes, this part is very tricky as it needs to be modular, and since all scientists are working with different types and sets of data as well as the complexity of transformation, the process needs to work for all. The idea we were using was storing the entire scripting process in a database, which could be called by a Python program and read into a console to run all data through a list of scripts. So as long as the processes by scientists could be made into programmatic scripts then they would all work. This Transformed data would then be loaded into another folder and kept as Analysis Ready Data to be used by scientists and at a later time be transferred to a current NRC data cube.

**GIS Data**

Dr.Andre Dyk requires GIS data pulled from the Copernicus website; currently, a scientist is going to the Copernicus website, searching for the best-fit tile (closest to Aug 1st and with minimal if not no cloud cover ) and downloading said data. This file is then pulled into ArcGIS or QGIS and a tile is set up. They can then remove cut lines, fit and overlap tiles to get better metrics and stitch multiple tiles together into a mosaic or virtual raster. The current method is slow and time-consuming and so a faster-automated approach is required. This involves fetching data from the Copernicus API, which with research may be fetched with the help of this GitHub repo <https://github.com/nasa/HLS-Data-Resources>. The approach we took was to look through Copernicus’s API list until we found one that would give us simple tile data. This ended up being far more complicated than we assumed it would be. Hundreds of ill-documented APIs give partial data, and in many cases, since we are not GIS experts data we did not understand. We took information from the downloadable file from the CB and tried to match that. The problem here lay in the fact that there do not seem to be any API calls that return XML data based on search returns, meaning we would need to create them from scratch, but with no explanation of how the XML worked it was a near impossible task. The XML documents for reference were hundreds of thousands of lines long, holding data that correlated to tileset data and also pointed to different areas in this directory where bands matching data was held. This was a non-starter.

After this failure, we learned that the GitHub repo above may be of use and that we could get proper data but ran out of time before the project ended. But presuming you can get the correct data the next logical step would be to use the Python console native to QGIS and have it read in that tileset data, transform it into a useful format and then have it programmatically build a virtual raster (mosaic). <https://docs.qgis.org/2.18/en/docs/user_manual/plugins/python_console.html>.

**Dr.Haos Data and R/RStudio**

When it came to Dr.Hao, we were given his data and R scripts, which was a great start. We set up a VM hosting the wanted Rocky Linux 9 distro, as well as setting up Pentaho, R, and R studio. The real problem we had here was with the dependencies of Dr.Hao. In order to get a seamless handover we would require a few things, one would be complete uninterrupted access to Haos Computer and Environment, and the other would be a direct copy of all download directories and downloads. We would need this as if our environment was any different from his there would be a chance it wouldn't work as he has different dependencies or versions or setups. He ended up having legacy versions of many dependencies that no longer exist on the internet, as they are completely deprecated. Rstudio and R weren't able to install them. We had trouble even finding the locations of some of the R dependencies, as the installs would fail no matter what we did. We found new CRAN mirrors, added the install option to install all dependencies, and all required downloads, and had the library pointing to the correct and then alternate state, and yet almost all of the dependencies would fail. We went on to try and manually install all the files but even directly putting some of the known dependencies into the folder with the same format, not only failed to recognize the dependency but also failed stating dependencies don't exist. But if a dev could get his literal exact files exactly how they are now and could make sure the correct env setup was created to mimic his then you could proceed. We were advised by teachers to stop, as it was impossible to proceed, and so we did and moved on to the GIS project.

**Pentaho Data-Integration**

Pentaho was another process that ended up being far more finicky than it should have been. Installing it on Windows and Mac was possible, but did not allow for the use of plugins and the marketplace, both of which were required. We tried multiple versions, fresh installs and even using the exact version that our stakeholder was using, but it never worked. After countless hours of trying to get it working, we on advice from a stakeholder decided to attempt it on Linux to see if the Windows and Mac versions were bugged, and it turned out they were. The reason the marketplace was so necessary was that there was a Python executor plugin, that we needed to run R scripts and move data around for Dr. Hao's data. We would have Python fetch API data through Pentaho, and have it move this data and transform it by passing it through to the R scripts and then Pentaho/python would handle the data on the other side. Pentaho is hard to install on its own, as the version we were using 9.4 was released incomplete, so you need to follow a certain number of steps and different download files to make it all work. One requirement that is not stated, and can be really hard to find the right version is making sure you have libwebkitGTK installed on your linux server. This allows the user to run the spoon.sh file which launched Pentaho.

Small guide to getting it all to work on linux:

* Download pentaho base
* Download pentaho hadoop
* Unzip base
* Unzip hadoop, Should automatically add it to the base
* Then we need to install the libwebkit
* Yum whjatprovides libweb\*
* Sudo yum install webkit2gtk3-2.42.5-1.el9.i686
* Yes install
* Cd /usr/local/pentaho/data-integration
* ./spoon.sh

There is a chance that even if all goes well and the marketplace works, you can click the install button and it will never actually install the plugin, every single plugin will fail, no matter what. The alternative is to find the plugin from the marketplace directly download it and move it into the plugin file located inside the Pentaho directory. Usually located at /usr/local/Pentaho/data-integration/plugins unzipped.

Marketplace link: <https://marketplace.hitachivantara.com/pentaho/>

\*\* be warned Pentaho changes their site constantly, so this may become deprecated.

**Setting up the Env**

The Env and Vm all got set up with relative ease, but as we are not expert admins or linux users, we got caught on small problems along the way. Dependencies and installation methods just not working, or have been deprecated but not stated to come up more often than we would have liked. Another problem we faced was how some programs just don't work on our version of linux or were never created for our version of RHEL, this caused us to find many workarounds.

Useful sites and links for future:

APIS:

**NASA -** [**https://api.nasa.gov/**](https://api.nasa.gov/)

**USGS -** [**https://earthquake.usgs.gov/fdsnws/event/1/**](https://earthquake.usgs.gov/fdsnws/event/1/)

**Copernicus -** [**https://documentation.dataspace.copernicus.eu/APIs.html**](https://documentation.dataspace.copernicus.eu/APIs.html)

**Sentinel2-L2A -** [**https://documentation.dataspace.copernicus.eu/APIs/SentinelHub/Data/S2L2A.html**](https://documentation.dataspace.copernicus.eu/APIs/SentinelHub/Data/S2L2A.html)

**OGR, GDAL, PDAL**

- Brief intro: <https://stackoverflow.com/questions/4496074/what-is-relationship-between-gdal-fdo-and-ogr>

[GDAL](https://gdal.org/index.html) - translation library for geospatial data formats

[PDAL](https://pdal.io/en/2.6.0/python.html) - VERY USEFUL, read documentation from link

[ArcPy](https://pro.arcgis.com/en/pro-app/arcpy/get-started/what-is-arcpy-.htm) - not only for spatial analysis, it’s also for data conversion, management, and map production

[ReportLab](https://www.reportlab.com/) - may be used to create report templates for GIS development

[Graphana](https://grafana.com/)  - useful for making graphs and dashboards

[MetaData Tools](https://www.montecarlodata.com/blog-open-source-data-lineage-tools/) - 5 Different open source metadata tools

[CentOS Stream 9](https://www.centos.org/centos-stream/) - Current newest most stable + longest supported CentOS Linux distro

[Logging W/ Python](https://docs.python.org/3/howto/logging.html) - Tracking events that happen when software runs. Adding logging calls to their code to indicate that certain events have occurred

Looks like we can have handlers that can determine when to log based off of what's going on.

[Ansible](https://www.ansible.com/) - How to deal with dependencies when transferring project

**Things that would go better on a second round**

* Understanding the strengths of the group better at the start
* Better communication with stakeholders
* Better inner-group communication
* Better understanding of GIS concepts
* Speaking to scientists earlier
* Getting several rounds with scientists
* Better allocation of resources internally
* Understanding scope better
* This is a very large and complicated project, better thorough understanding of resources would have helped