LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-769890

Publication of LLNL Genomics Data to ESGF: a Technical Report for DREAM Project

S. Ames

March 18, 2019

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Publication of LLNL Genomics Data to ESGF: a Technical Report for DREAM Project

Sasha Ames

March 2019

Abstract

We present the process used to set up several example genomic data sets within an ESGF framework. The process begins with designing a project, staging the data, and handling the necessary configuration. We describe the publication process used to ingest the datasets, in essence, all required metadata. Finally, we describe the data web service that we introduce in the DREAM project which makes these datasets available for download or exploration.

1 Introduction

The overarching goals for the DREAM project [DREAM] have been to enhance the ESGF software stack [ESGF] in ways that may ultimately support the extensibility of the software components notably in their use for numerous scientific domains. Such an activity would bring the software beyond the original narrow scope of data formats with climate / Earth system modeling and observations. To some extent, many of the components are domain agnostic, notably the Solr backend. Based on prior experience with a number of genomics formats, we have chosen several for a proof of concept data management project using an existing ESGF node installation.

The ESGF data node stack and publication software are sufficiently extensible to support any file format although having use almost exclusively for netCDF data as used by the climate / earth system modeling and observations communities. The two software pieces requiring implementation for this effort are the specific data handler for files not in the netCDF format and a data service module (web service) that can serve files of various types. The service also provides several exploratory capabilities with server side file parsing for data inspection within the data file, namely .json and .fasta data types. In addition we use a somewhat non-standard ESGF publication preparation step, given the nature of not using netCDF files as expected by the standard tools.

This document gives an overview to the steps needed to achieve a working genomics-data publication. The process starts with designing a project. from that one can arrange and stage data, then perform the software-specific project configuration. We then describe both the specific data handler used for working with our non ESGF-standard datasets and our data service specifically implemented for this effort, with details on the service setup. Finally, we give several steps used to properly test data access to the published data sets using the dream-data service.

While little to no familiarity with ESGF is hopefully sufficient, it might be helpful to familiarize one with the system, the examples here might be useful: https://portal.enes.org/data/data-metadata-service/search-and-download/web-portal-based-data-search

The genomics datasets used as examples in this document have been taken from two research projects at LLNL: metagenomics classification [LMAT] and analysis of antimicrobial resistance genes using genome sequence graphs (publication pending).

2 Configuration and Publication

2.1 Design the Project

Data to be published must be well organized. The organization will ultimately map into file system directory hierarchy, comprise ESGF *dataset ids* and become searchable fields, which we refer to as facets. The combination and order of these facets is referred to as the Dataset Record Syntax (DRS). It is crucial early on to choose the facets that allow for distinguishing among various datasets.

Note that even within a project, we may encounter subsets of datasets where one particular facet may determine which other remaining facets should be used to characterize those datasets. In our example the 'application' facet makes this determination: each particular application several distinct facets to categorize datasets within that subset. On the other hand, a straightforward approach employed by many ESGF projects is to be completely consistent across the project, namely to have every facet for every dataset regardless of having a valid value. Motivation for using the first approach above is that that project itself is small enough in scope, such that it would be preferable for using 'application' to stick with our example, instead.

In our example, the project, activity, application, and dataset_category facets are consistently used across all datasets. However, the db_size, experiment, k_length, sra_id are specific to particular application and/or dataset category.

2.2 Arranging / Staging your Data

Once you have chosen the facets and values for the data, its time to build a data structure to hold the data in preparation for publication. Here, you will think hierarchically, those some facets can be grouped arbitrarily. For instance the descendant directories of the `query_output` could be grouped either way: `k_length` or `experiment` first. In the case below we place the `CARD_400` experiment before the particular k-mer lengths, which is reasonable given a single experiment and multiple k-mer length values in this example.

In our example we have the following:

```
DREAM/bioinfo/LMAT/db/full
             /metagenomes/hmp/SRR059552/v1
                              /SRR

               /Seqgraph_fixed/query_output/CARD_400/7
                                                   /11
```

Additionally, datasets are typically versioned. If changes to the data are not expected, eg. legacy datasets, the creation of such *version* directories is optional. In our example (above), we have create a version with the hmp sequence example datasets, and include a single version, which we have denoted with a `v1` directory.

2.3 Configuring the Project

In order to publish, the project must be configured using the format employed by the ESGF publisher module `esg-publisher`. This process is done through the `esg.<project>.ini` files. In our example, we have created the `esg.dream.ini` configuration file. Project configuration has the following crucial steps, where the latter two comprise the *DRS* definition and are conventionally congruent, meaning having the identical facets in the same order:

- Definition of Facets

- Directory Format
- Dataset ID

To define the facets, for an example of one of the configured facets, and following the additional DRS-item configuration properties, we have:

```
    sra_id | string | false | true | 4

directory_format = /p/user_pub/work/DREAM/bioino/%(application)s/ \
                    %(dataset_category)s/%(sra_id)s/%(version)s
dataset_id = DREAM.bioinfo.%(application)s.%(dataset_category)s.%(sra_id)s
```

The standard convention for the latter two properties in the file is to preserve the same order within both directory and dataset ID arrangements.

Not required but helpful are the following properties for facet definition:

- Options
- Defaults

In our example, we publish under a `bioinfo` activity and establish `activity_options` and `category_default` settings for that facet, which is universal across all dream demo datasets. Options are used exclusively with the *enum* facets, and list all options. The publisher will not allow a dataset to be published if the option detected in the dataset definition does not match one of the options listed in the project's configuration file.

the facets "model" and "experiment" are considered "special" facets. The original publication design created special database tables to manage these particular properties, as there was a need to attach additional metadata for each. These two special facets are not required. We have chosen not to use the model facet in our example, but we use experiment in the cases where appropriate. Additionally, the model facet requires a different means of configuration, and we consider the explanation beyond the scope of this demo, but explained in the context of CMIP models [CMIP] in the `esg-publish` documentation [ESG-PUB]. In our example, we consider an experiment called `CARD_400`: 400 synthetic gene variants from the CARD reference data. We add to our configuration:

```
experiment_options =
   dream | CARD_400 | 400 synthetic gene variants from CARD gene sequences
```

The project configuration additionally specifies the proper data handler discussed below.

```
format_handler_name = multiple_builtin
project_handler_name = basic_builtin
```

We also include the following configuration to configure our data handler (see below) to expect data files without variables, as such are not appropriate for all data project, for instance, sequence data files:

```
variable_per_file = false
variables_none = attr
```

The following are added to the top-level `esg-publisher` configuration - `esg.ini`:

2.3.1 Add an entry for the dream project:

```
project_options =
    ...
    dream | DREAM | N
```

In the above exmaple, N is the next integer in the table following the other projects.

2.3.2 Ensure that a dataset root is added where the sample data resides:

```
thredds_dataset_roots =
      logical_path | /path/to/data
```

Common logical path names are derived from the `/path/to/data`, eg. `path_to_data`.

2.3.3 Add the dream-data service to the list of file services:

```
        HTTPServer | /dream-data/ | HTTPServer | fileservice
```

2.4 Custom Handler

The `esg-publisher` uses a Python class know as a handler to process data files of a particular format. There are also project metadata specific handler classes, but such a handler was not needed to be defined solely for this effort.

The default file handler solely handles netCDF data, as customary for the present use of ESGF. We make use of the multi-format handler class, which can extend to allow for "generic" data not in netCDF. For netCDF data, the handler will pass the operation onto an underlying netCDF handler. For other data, the handler will return empty list or string objects, as it is assumed that for these generic format(s), the information will be provided externally.

One of the key methods within the handler class is the extraction of named variables within the datafile. In the specific genomic dataset examples, there is no notion of named variables within the data file we consider within our context in publishing. Our handler class is appropriately suited for this, as for some non-netCDF data like our example formats, there is no attempt to extract a variable. The handler return none such that the variable field is populated nonetheless. This behavior is comparable to the issue of facets that have no valid value for some subset of datasets, but in this case "variable" is another "special" facet, and must be populated. The handler also has functionality to inquire about the dimensionality of variables, which are beyond the scope of this demonstration effort but of potential use for future data types in scientific domains beyond Earth sciences.

2.5 Preparing for Publication

The required process needed to prepare for ESGF publication is the preparation of *mapfiles*. The `esgmapfile` utility included in the `esgf-prepare` (`esgprep`) package is the standard mechanism used for netCDF. Consult that package documentation or the ESGF publisher documentation for information on the mapfile format.

Mapfiles contain essential metadata needed by the publisher to ingest it all into the ESGF-managed modules (to be discussed in the next section). Given that our data is not in netCDF format, we elect to use a simplified collection of several scripts to complete metadata. These scripts are available for a proof of concept rather than comprehensive. The `get_meta.sh` script will determine three metadata fields: the file size, timestamp, and a checksum, presently using `SHA256`. The second

script `create_dset.py` parses the data path to generated a dot-delimited ESGF dataset id. For our purposes we can generate dataset IDs straight from paths. Our example:
`/path/to/data/DREAM/bioinfo/metagenomes/hmp/SRR059962/v1`
becomes
`DREAM.bioinfo.metagenomes.hmp.SRR059962` for the master dataset id. Specific version will add a `"vN"` where N is an integer version number. A convention used in ESGF for versions is to represent a data as a single 8 digit integer `YYYYMMDD`.

A mapfile entry then would take the following form:

```
DREAM.bioinfo.SeqGraph_fixed.query_results.CARD_400.15 |                    \
/p/user_pub/work/DREAM/bioinfo/SeqGraph_fixed/query_results/CARD_400/15/    \
CARD-400-rel-res.half.15.json | 2764679 | mod_time=1511802165 |              \
checksum=8be0a1f1f2fe5cfb474ff16a5969aa1200d3112ef31443f51bc1905f00825232 | \
checksum_type=SHA256
```

In our example, we use different facet key tuples in our dataset depending on which application or dataset category is present.

2.6 Run the Publisher

To get our sample data into an ESGF system, we run the esg-publisher software with the previously generated mapfile. Our custom handler, as described above, does not open the data files during the process as the metadata is present. We follow the recommended procedure of running the publisher in three phases to ingest into our PostgreSQL database, THREDDS catalog and Solr indexing. For the THREDDS phase, we alter the command line arguments to not reread the datafiles (`--noscan`) and specify the services (the publisher can be configured to support alternative data services to be represented and `fileservice` is the convention for a regular combination of services.

See the esg-publisher documentation for more information.

In our example we name the mapfile `DREAM.bioinfo.SeqGraph_fixed.map` which contains all the `SeqGRaph` project .json output files. We will invoke the publisher with the following three commands:

```
(esgf-pub)$  esgpublish --project dream --map REAM.bioinfo.SeqGraph_fixed.map
(esgf-pub)$  esgpublish --project dream --map REAM.bioinfo.SeqGraph_fixed.map \
--noscan --thredds --service fileservice
(esgf-pub)$  esgpublish --project dream --map REAM.bioinfo.SeqGraph_fixed.map \
--noscan --publish
```

3 Service Overview and Testing

3.1 Data Service

To effectively support access to genomic datasets within ESGF, we have created a prototype implementation of a lightweight data service. For the purpose of making individual files accessible over http(s), this process was straightforward rather than focusing on customizing TDS (THREDDS Data Service), which is the standard component for netCDF and other gridded file type used in ESGF for relevant domains. Our data service, which we named *DREAM Data Service*, makes publishing data of unknown data types easy. The only configuration for new file extensions, those that might be adopted by emerging tools in the domain, is whether the data file is a proprietary binary format or ASCII, in which the latter case, it might be advantageous to allow a browser to display the file as plain text. Other common mime types are supported. For instance, .json files, as part of one of

the demonstration projects has sub-contents that can be expanded or collapsed in recent browser file handling.

The data service resides on the ESGF "data node" where the publication database and publishing client generally reside (but neither a strict requirement for a data deployment). URLs that refer to the service will contain the server domain name, but these reside on a frequently (but not exclusively) remote index server.

The service supports interaction with two file types for server side data query. We provide this functionality for users at a convenience for large data where low bandwidth makes downloading an entire file impractical if not impossible.

1. .json files: As .json is widely adopted for semi-structured data and conveniently maps to list and dictionary types within Python, we support the server-side introspection of .json data. Specifically, we have the ability to get the following:

   - Length of array or dictionary
   - List of dictionary keys
   - Value for particular key
   - Array value from index

   These operations are performed with a simple query string syntax, but with additionally code in Python, it would be straightforward to construct a wrapper class which abstract away the query string and provide a convenient standard python programming interface, namely list and dictionary type operations, as python uses the `[ ]` and { } meta-characters to represent each as does .json.

2. .fasta: similar to .json, we provide introspection utilities to get information about a .fasta file without requiring a download. We have the following operations embedded:

   - Length (count of all sequences)
   - Get headers as a array
   - Get sequences
   - Get header / sequence at index
   - Get sequence from header (creates a map)

On the first operation, the file will be loaded into memory. Subsequent operations benefit from server-side caching.

3.1.1 Installation and configuration

The service written in Python is hosted in GitHub. A `requirements` text file lists the specific modules needed. The flask module is used to enable the web service capability. We have installed the service as a mod-wsgi application. To enable the module within Apache Httpd, as we have done in our test example, we add the following to the (esgf-)`httpd.conf` (the ESGF software stack customization of `httpd.conf`):

Provided the module has been installed at `/path/to/esgf-dream-data-service`, the directives install the service at the dream-data endpoint. In a test example, world-read permissions are needed so the Apache Httpd process can load the python module using `wsgi.py`.

```
WSGIDaemonProcess dreamdata python-path=/opt/esgf/virtual/python/lib/python\
2.7/site-packages:/path/to/esgf-dream-data-service user=apache group=apache\
threads=5
WSGIScriptAlias /dream-data /path/to/esgf-dream-data-service/dream-data.wsgi
<Directory /path-to/esgf-dream-data-service>
```

```
            Order allow,deny
            Allow from all
            AllowOverride None
</Directory>
<Location /dream-data>
   WSGIProcessGroup dreamdata
   WSGIApplicationGroup %{GLOBAL}
</Location>
```

3.2 ESGF data roots

It is essential for the data service to be able to map back the logical URL to the physical path as such is preferred to hide from an end user. The file `convert_path.py` contains the mappings of the logical THREDDS roots to physical paths. Additional roots can be added as string to string mappings. Note that these are the same roots as found in the top-level `esg.ini` file referenced above.

3.3 Testing

Manual browser based testing makes use of the ESGF CoG user interface. In a typical ESGF deployment the index site that is the same site where data is indexed. We can browse the datasets using the search facet selection bar on the left side of the page. Search typically begins using the "Project" facet. In our example, we have published these datasets under the `DREAM` project. To facilitate our search, we create a customized search demonstration page that contains all the facets relevant to our project (the typical facets for ESGF CoG sites are properties relevant to climate / Earth systems data). Checkboxes are selected to narrow down search results. In the center column of the browser window, results appear with several links. If we are interested in downloading one or more datasets, the `wget` script option or Globus download. For a single file, or to simply test download, the most convenient option is to open the "List Files" link (figure 1, which expands the list of files for the dataset. On the right of the each displayed file, there is a "HTTP Download link".

A simple click on the link performs the download. However, if we wish to use the interactive capabilities of the service, we should copy the link first. Next, on a command line, use `curl` or `wget`. Alternatively, we can browse for published data sets using the REST API directly: `https://www.earthsystemcog.org/projects/cog/esgf_search_restful_api`.

3.4 Service Metrics

We measured timings of several server-side query request. The main purpose of this exercise is to compare performance before and after the application caching with various sized datasets. Our expectation is for the larger datasets to exhibit the greater speedup due to our application caching. In Table 1, we present the measurements from six files from the Sequence-Graph data in .json format. Timings of requests to count the entries of each file (requiring parsing the .json) are performed cold and the performed again on the same file, after the cache has been populated, to measure the speedup due to the cache. We perform three requests of each (cold/warm) and average these, using the averages to compute the speedups. The first three files are within 10 percent in size of each other, while the second three (larger) are more than twice as large as the first, and those vary in size a bit more. Looking at Table 1, although there is some variability in speedups among the three smaller and three larger of the six datasets, we observe that our measurements show greater speedups as the dataset sizes increase, as expected.

Data Size (MB)	Response time speedup
2.7	1.2
2.8	1.15
3.0	1.7
8.5	2.6
9.1	2.4
10.1	2.6

Table 1: Data sizes (MB) and speedup times due to server-side caching

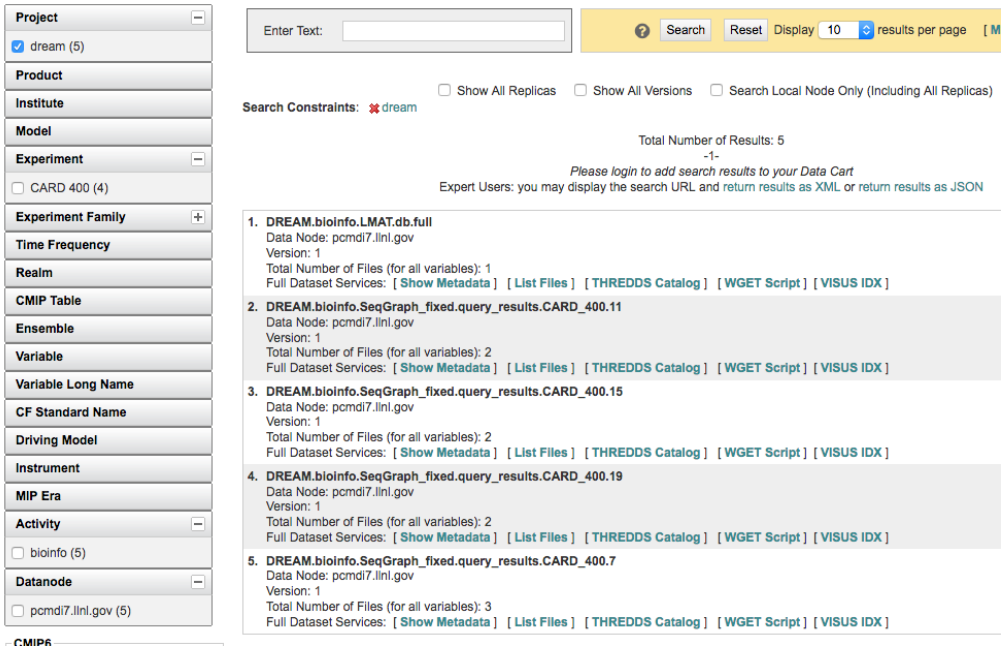

Figure 1: Example search results of the sample DREAM data on ESGF CoG.

3.5 Going Further

This report is limited in scope to the activities completed for the DREAM project. It does not cover the extent of additional datasets from the biological sciences domain. We can envision that bio-medical data that is single to multidimensional in nature would resemble the sorts of structured gridded products. For instance, tomographic imagery naturally fits a three-dimensional grid may have multiple channels, those represented naturally as variables for each. There has been previous work in publishing epidemiological datasets. of which have a geographical component [cite Dehaessler, Ames 2014 slides], and every reason to believe that future data of this type can be managed within ESGF A future production deployment would include more automation, as is done for ESGF.

Acknowledgement

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344

References

[CMIP] https://pcmdi.llnl.gov/mips/cmip/about-cmip.html

[COG] https://www.earthsystemcog.org/projects/cog/

[DREAM] https://dream.llnl.gov/

[ESGF] The Earth System Grid Federation: An Open Infrastructure for Access to Distributed Geo-Spatial Data, by Luca Cinquini, Dan Crichton, Chris Mattmann, Gavin Bell, Bob Drach, Dean Williams, John Harney, Galen Shipman, Feiyi Wang, Philip Kershaw, Stephen Pascoe, Rachana Ananthakrishnan, Neill Miller, Estanislao Gonzalez, Sebastian Denvil, Mark Morgan, Sandro Fiore, Zed Pobre and Roland Schweitzer, in proceedings of the IEEE 2012 Conference on eScience (DOI: 10.1109/eScience.2012.6404471).

[ESG-PUB] https://esgf.github.io/esg-publisher

[LMAT] Sasha K. Ames, David A. Hysom, Shea N. Gardner, G. Scott Lloyd, Maya B. Gokhale, Jonathan E. Allen. Scalable metagenomic taxonomy classification usng a reference genome database. Bioinformatics, July 2013.