ESGF and DREAM
Distributed Resources
for the Earth System Grid Federation (ESGF) Advanced Management

Luca Cinquini

JPL Task Plan No. 82-19888
June 2017 Quarterly Report

This report covers the period from the start of the project (approximately, October 2016) to the end of June 2017. During this period, JPL staff, in accordance with the Task Plan mutually agreed upon with LLNL, performed the following activities:

1. First version of Docker-based ESGF architecture

The DREAM team has agreed on using Docker as the core technology to modularize and generalize the ESGF architecture, as necessary pre-requisite to port the ESGF infrastructure to other scientific domains. Docker is the industry leading "containerization" technology, which allows applications to be built as "black boxes" that contain all the necessary software to run them, and can be deployed on any Docker-enabled host. The team has undertaken the monumental task of converting the current ESGF installation software into separate Docker images, which can be run together as a set of interactive containers. At this time, most of the major ESGF components have been converted to Docker (ESGF search, Identity Provider, Openid Relying Party, Thredds Data Server, Solr, CoG, Apache httpd, Postgres) and can be run together to stand up a full ESGF Node (see Figure 1). In this architecture, all of the specific site configuration is restricted to a location on the local host, outside of the Docker containers, so that it can survive image upgrades. Additionally, the data used by each application (the Postgres database, the Solr index, the TDS catalogs, the CoG site data) are stored on specific Docker volumes, which are managed by Docker independently of the containers, so that they can also persist through software updates.
One of the many advantages of Docker is the ability to mix-and-match images into more complex applications by writing "docker-compose" configuration files. This functionality was leveraged to create alternative versions of an ESGF Node, specifically an ESGF Data Node only, an ESGF Index Node only, and an ESGF Index Node with Solr Cloud support. This is extremely important as it showcases the ability of ESGF to create target architectures that are specifically suited to a data center or environment (for example, a Tier 2 ESGF node that only needs to serve data to a parent Index Node), and it enables scaling to multiple host clusters, and on commercial cloud environments.

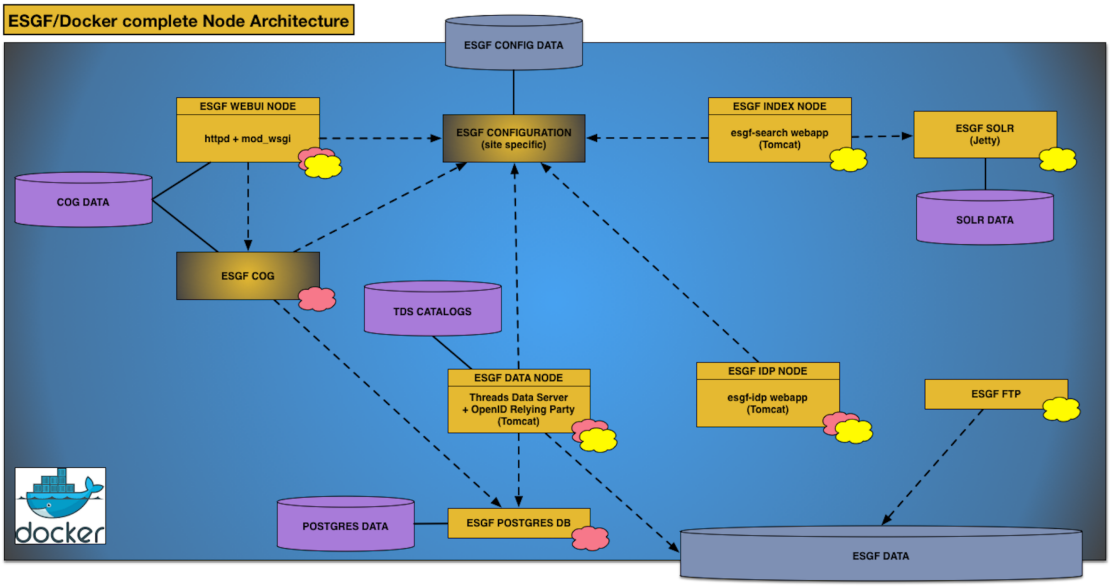

Figure 1: Docker-based architecture for a complete ESGF Node.

2. Analysis of RESTful API Description Languages

One of DREAM goals is to expose each service module through a well defined RESTful API that hides internal implementation details, and can be used by ubiquitous HTTP clients to interact with the service. To this goal, the team has conducted a survey of the currently available standards and languages for describing REST APIs. The result was that many such competing languages are currently in use, without a clear winner that could be identified. Ultimately, the team has decided to adopt RAML (REST API Markup Language) to describe all DREAM service APIs, because of its maturity, its expressive power, and its modular constructs. RAML seems to be the best choice to describe APIs that are independently designed and implemented. Another possibility was to adopt Swagger, which is instead more like a framework of tools to define APIs first, then build client and server side software to implement them and use them. Fortunately, there are tools that can be used to automatically convert RAML specifications to Swagger, and vice-versa.

3. Big Data Requirements for Hydrology

During 2016, the team has started involvement in the Hydrology domain by executing a careful analysis of the data challenges involved with understanding and predicting water availability across the U.S. The resolution of this problem necessitates the combined analysis on massive hydrological datasets from disparate sources, including model output data, in-situ observations, remote sensing, historical data, and others. Other key computing challenges described in the document include orchestrating different model execution environments,

optimizing data movement, working with distributed data archives, running analytics in real time, and development of machine learning algorithms.