Data Services to Improve Access to Scientific Image Data

Leo Sciortino
School of the Arts
Rochester, New York

Advisor: Richard Kidder

University of Rochester Laboratory for Laser Energetics

December 2023

1. Abstract

An investigation of methods to access image data stored from experimental campaigns was conducted. The study found that there are several image data formats that have been historically used by LLE. Images are currently stored on LLE file servers, with files indexed through LLE's relational Oracle database. This investigation explored technologies to provide easy and secure access to image data, minimizing the need for direct database and file system interactions by the user. The project focused on image parsing, storage, and retrieval mechanisms for multiple formats including HDF4, HDF5, TIFF, and JSON (JavaScript Object Notation). Services to provide data processing (e.g., background subtraction) were also investigated. Project software was tested using Python, NodeJS, PL/SQL, and an Oracle database. Overall, the research found that using Python and its associated libraries in conjunction with web services is a viable option for presenting and processing image data.

2. Introduction

HDF **(**Hierarchical Data Format) file format is a type of file that is typically used to store scientific data. At the Laboratory for Laser Energetics (LLE) an HDF file will typically store an image and its corresponding attributes. Currently most HDF files at LLE are stored in HDF4[1] format, which is outdated for most applications compared to its successor HDF5.[2]

This investigation looked to improve the current pipeline that principal investigators (PIs) use to retrieve image data from experiments on the OMEGA and OMEGA EP laser systems. It was found that there are two methods that PIs use to access image data from OMEGA archives,

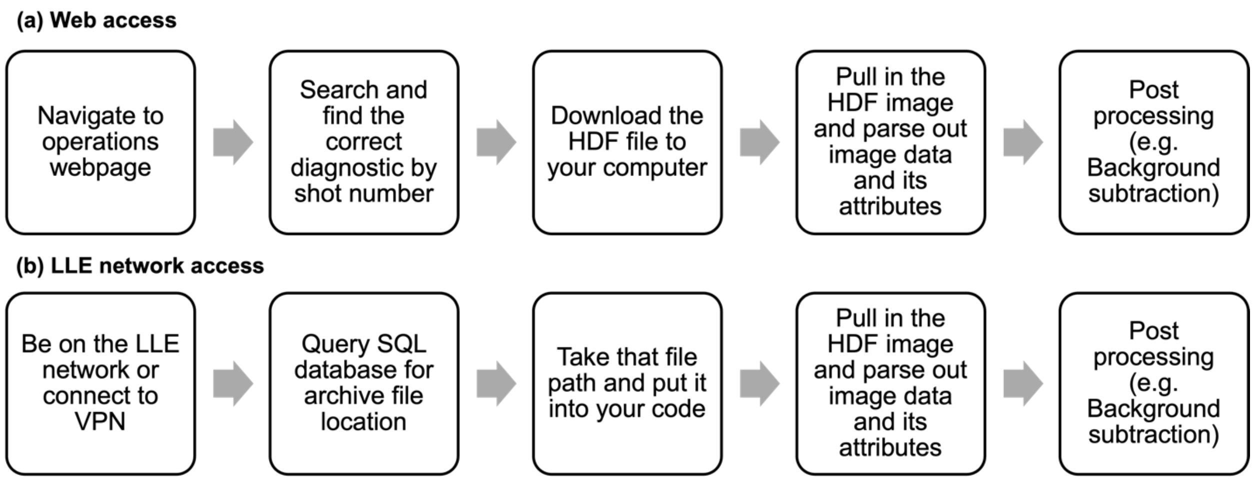
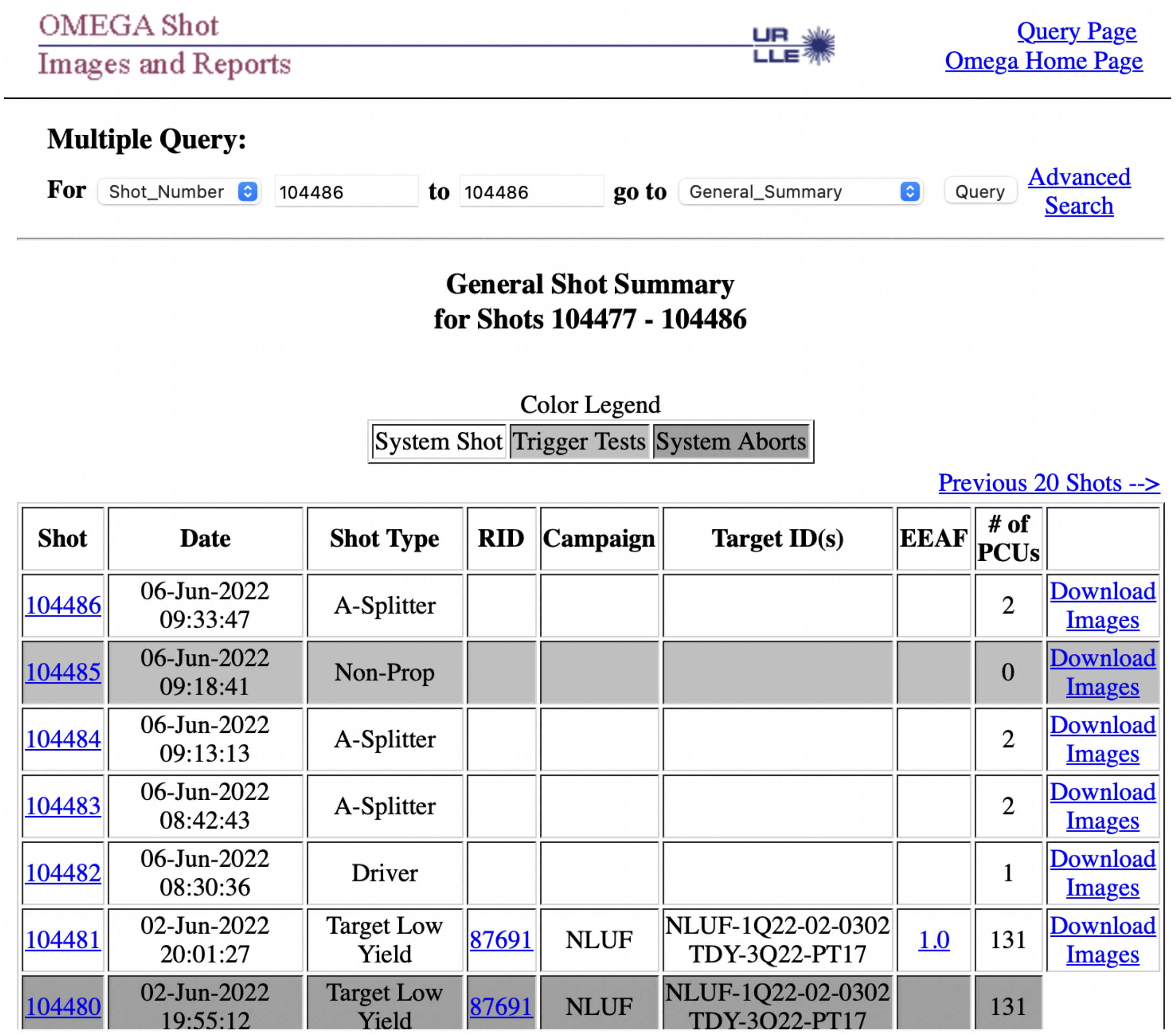

(a) Web access
Navigate to operations webpage
Search and find the correct diagnostic by shot number
Download the HDF file to your computer
Pull in the HDF image and parse out image data and its attributes
Post processing (e.g. Background subtraction)
(b) LLE network access
Be on the LLE network or connect to VPN
Query SQL database for archive file location
Take that file path and put it into your code
Pull in the HDF image and parse out image data and its attributes
Post processing (e.g. Background subtraction)

Fig. 1 Two ways PIs can retrieve images and their corresponding data. Both are slow and cumbersome.

both of which were reviewed for process improvement (Fig. 1). The first method (Fig. 1(a)) uses web access, which doesn't require PIs to be on the LLE network but uses traditional early 2000s web access technology through the OMEGA Shot Images and Reports page (Fig. 2). These pages require user interaction to locate and download data, making data retrieval slow and cumbersome.

These webpages are also difficult for LLE to maintain. They use outdated programming practices such as inline SQL (structured query language) database calls, and inline called shell scripts. This can easily introduce data security problems through SQL injection where a user can add to the end of a SQL query compromising the security of the database. Additionally, these webpages are written in PERL, which has become less popular for modern web development. Poor documentation of this code also hinders its further development. For a developer to modify these scripts one must spend hours searching through dozens of files.

The process to access data was found to be problematic as the user must navigate to the correct shot number and then find the diagnostic and download the image onto their computer. PIs also have an option to download the data package that contains all or part of the relevant image data for that shot. Since multiple users work on these campaigns it is likely that these images are copied an unnecessary number of times. After the PI obtains the image data they can open the data for processing.



Multiple Query:

For Shot_Number 104486 **to** 104486 **go to** General_Summary Query Advanced Search

General Shot Summary
for Shots 104477 - 104486

Color Legend

System Shot	Trigger Tests	System Aborts

Previous 20 Shots -->

Shot	Date	Shot Type	RID	Campaign	Target ID(s)	EEAF	# of PCUs	
104486	06-Jun-2022 09:33:47	A-Splitter					2	Download Images
104485	06-Jun-2022 09:18:41	Non-Prop					0	Download Images
104484	06-Jun-2022 09:13:13	A-Splitter					2	Download Images
104483	06-Jun-2022 08:42:43	A-Splitter					2	Download Images
104482	06-Jun-2022 08:30:36	Driver					1	Download Images
104481	02-Jun-2022 20:01:27	Target Low Yield	87691	NLUF	NLUF-1Q22-02-0302 TDY-3Q22-PT17	1.0	131	Download Images
104480	02-Jun-2022 19:55:12	Target Low Yield	87691	NLUF	NLUF-1Q22-02-0302 TDY-3Q22-PT17		131	

Fig. 2 OMEGA Shot Images and Reports Webpage. Data can be slow and hard to find

PIs with LLE network access can directly access the database and file systems (Fig. 1(b)) instead of going through the web interface. This isn't available to external PIs. This process requires the user to have an account in the database or use a common database account. Both processes introduce security concerns. For a common database account, the password is shared among users, so it isn't known who is retrieving what data. This process is also cumbersome for the user as the PI must learn to write SQL scripts for their analysis. A PI with network access must use the SQL queries to first locate the files, and then must directly access the file on the file server. If desired, the PI can directly implement post processing into their analysis code.

It is also clear that many LLE PIs do similar post processing to images to get them ready for analysis. It seems unnecessary to have this post processing done on local machines a redundant number of times. It seems that that some post processing could be done on LLE servers prior to PI access. Specifically, many PIs subtract the background from images before their scripts perform more post processing.

3. Image Storage and Retrieval

Development was first focused on the best way to store and retrieve images and image data. One idea that was explored was storing HDF Images in an Oracle[3] database. Three ways were considered. The first way was storing them as BLOBs (binary large object files). In short, a BLOB just stores the binary in the database as it would be in a file system, but the file address can essentially be indexed relationally. Unfortunately, this doesn't allow for easy post processing within the database.

B-Files are another common way to store files in a database. These are essentially just links to files that are stored on a file server (the data for the file isn't stored in the table space). Although these can be indexed very fast, it was concluded that this file type wasn't the right fit for the project since our goal was to do some post processing on images. A very similar ARCHIVE_FILES table space also already exists, which helps map shot numbers to file system locations. In the future these tables could be converted to use B-Files instead of just storing the file path as VARCHARs (variable-length character strings).

We also explored storing image data directly in the database. This involved storing image pixels in rows and columns in the database. We tested basic post processing of images stored in table space with SQL. Specifically, we tried background subtraction, which was easily achieved. This could also have been very easily added to a script through PL/SQL[3], which is built into the SQL developer software. The one downside of this method is that it wasn't fast compared to other methods. This is because data base tables are built on a key value relationship that is based on hash functions. A hash function[4] is a one-way and collision resistant function that makes accessing a value from a key fast. A hash function isn't needed when it is desired to access most of the table, which is what we want to do when accessing an image. So, overall, it is inefficient to use database tables to store and process images.

In all three of the above methods used to store and process images within a SQL database, a Python[5] data handler is still required. If fully implemented, a Python data handler script would be used to insert images into the database. When a user requests image data, the Python data handler then pulls data from the database. To originally populate this database, all the file paths of images would have to be pulled off the ARCHIVE_FILES table mentioned above, and all that data would have to be copied into the database using the specified method through the Python script. Similar scripts could also be written in JavaScript.

4. Image Postprocessing

Since the data handler script was written in Python it was an ideal place to also do post processing in Python. Whether a PI is interested in high level or low-level processing, Python is well supported in the scientific community. We started by converting images into Python arrays. It was found that this was slow since Python does a lot of type checking. We then discovered the NumPy[7] library. While Python is a high-level and interpreted language, NumPy bridges the gap between high-level programming and low-level performance, making it a powerful tool for numerical computations and array manipulations. NumPy is implemented in C, which is a fast low-level language. Images were converted to NumPy arrays for post processing. NumPy's fast array manipulation capability is ideal for image processing or transformation. For our basic background subtraction postprocessing we found NumPy to be four times faster than the SQL table processing that was mentioned in section 3.

We wanted to add the ability for PIs to pull image data in any format. After post processing was done, we implemented an on-the-fly image type converter in Python. The Python library Pillow[8] was used to allow data to be exported in various formats including PNG, JPEG, and TIFF. The h5py[9] Python library was used so that the image could also be returned in HDF5, the new version of the HDF file format.

5. Proposed Solution

Figure 3 provides an overview of the proposed process whereby users can access images from the data base and specify postprocessing that is to be performed before the image is returned to the user.

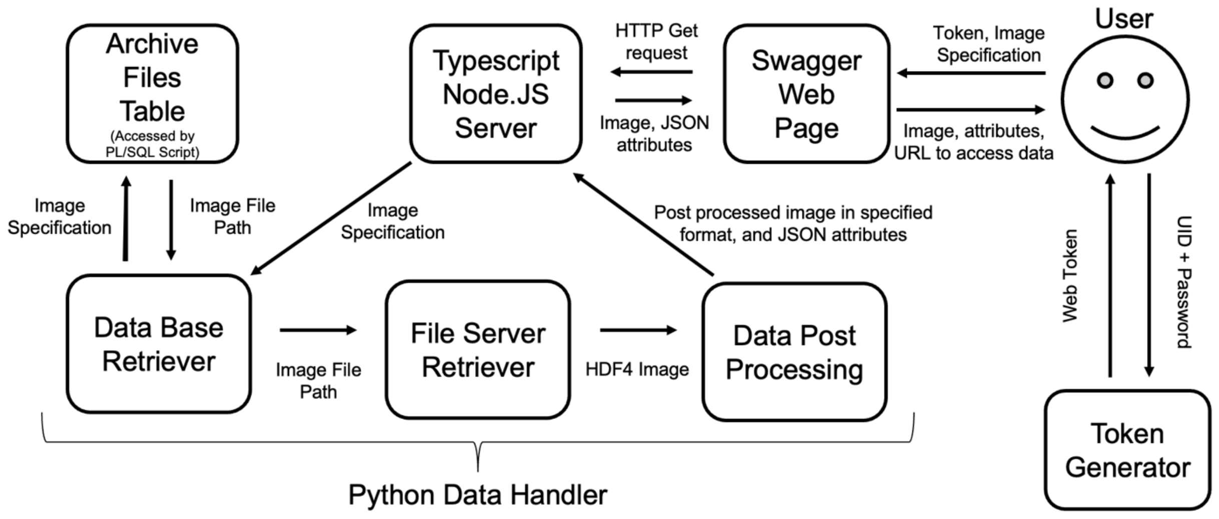


Fig. 3 The new proposed process for PIs to access and process image data. Note that Image Specification includes the diagnostic, shot number, post processing, and desired image format. Authentication is done with the web token within the Node.JS server.

.

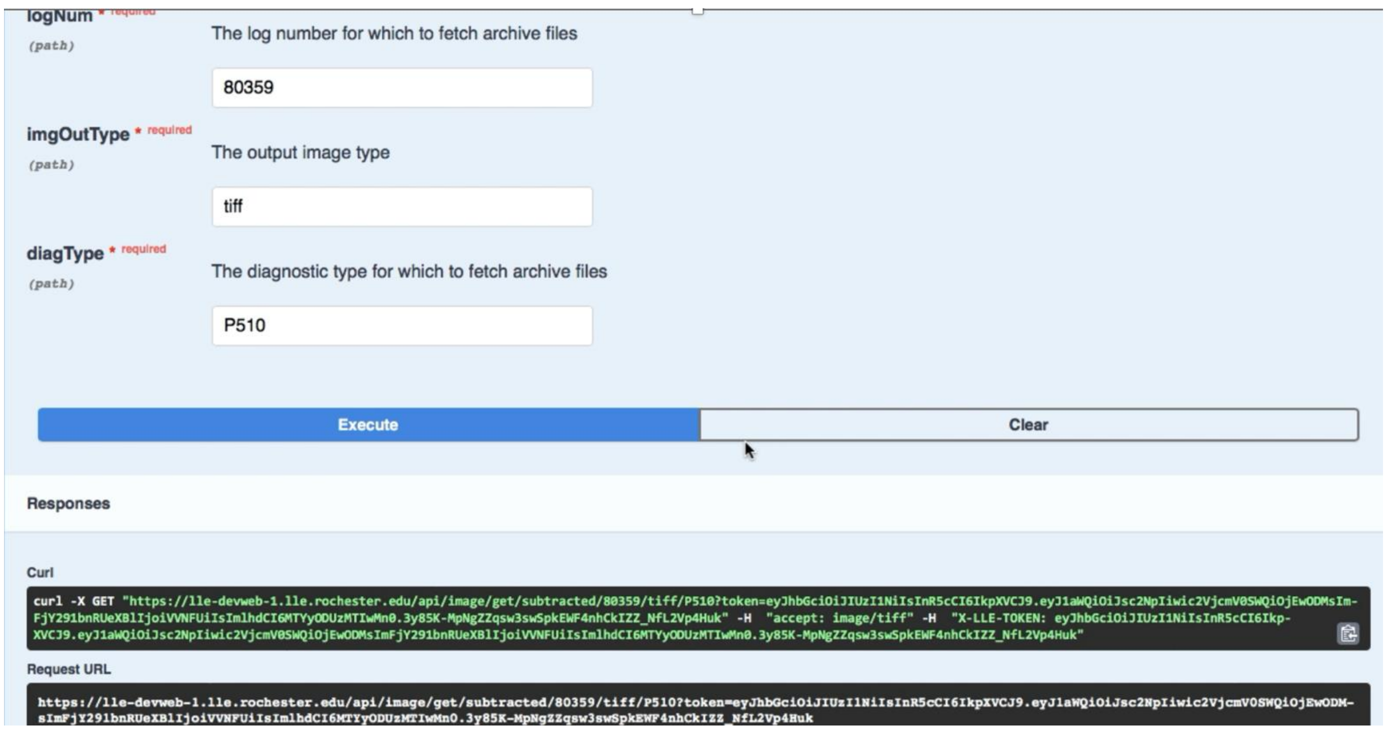
LLE's RESTful (Representational State Transfer) API (application programming interface), which pre-existed to return operations data to PIs, was leveraged, to provide new routes for processed data. A RESTful API is an API that has numerous properties, the most important being that it is stateless, and has a uniform interface[10]. Stateless means that each request to a server is independent of other requests. This means that any request sent must contain all information needed for the server to return the required data. The user cannot count on the server saving information from past requests. Uniform interface means that meta data is sent and requested in a common format. This means that a RESTful API should have a unique URI (uniform resource identifier) for each resource. Generally, the client tells the server which resource it would like to identify through a URL (Uniform Resource Locator). To identify what the server needs to do with this resource, RESTful APIs typically implement HTTP (Hypertext Transfer Protocol). Common HTTP requests are GET, PUT, POST, and DELETE.

Node.JS[11] is an open-source, server-side JavaScript runtime environment that allows developers to run JavaScript code outside of a web browser. It enables the execution of JavaScript on the server, providing a powerful platform for building scalable, high-performance web applications and network services. We used Node.js with the Express[12] web application framework to build these routes. To test and browse the API we used the Swagger[13] API client. This gives users an efficient way to test and graphically view API endpoints.

When implementing new API routes we break the code down into two parts: the route and the controller. The same controller can be used multiple times if the same type of data is being used for multiple routes. In this case the controller would call the Python script from Typescript (Typescript is just a type enforced version of JavaScript which can be used with Node.JS), which would then go and retrieve the data, do whatever post processing was specified by the user, and then return the image for the user to view through Swagger. Swagger also provides a URL that PIs can embed into their code. The corresponding HDF attributes can be returned in JSON[14] (JavaScript Object Notation). This is a very easy format to work with as many programming languages easily support its manipulation. All the PI must do first is get an authenticated web token. This doesn't require them to be behind the LLE firewall. Fig. 3 shows how the user interacts with the Swagger page and how Swagger works with each part of the system.

There are many other opportunities for built-in image processing. Instead of one PI having to go to another to get their data or code, this processing could be implemented directly in Python. Scientists could get the image and data already processed through an API call by specifying what type of processing they would want done directly as a query.

It was realized that a pre-existing REST API used to return operations data could be leveraged to return processed data. We created HTTP GET routes for the processed data. A GET route is an API call that can return stateless information. This could greatly reduce the amount of processing that PIs do when they receive an image. Figure 4 shows how a user could access a sample diagnostic from a specific shot number. It can be also seen in the Request URL that users can specify the type of post processing; in this example background subtraction is shown with the route /api/image/get/subtracted.



The log number for which to fetch archive files

80359

The output image type

tiff

The diagnostic type for which to fetch archive files

P510

Curl

```
curl -X GET "https://lle-devweb-1.lle.rochester.edu/api/image/get/subtracted/80359/tiff/P510?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1aWQiOiJsc2NpIiwic2VjcmV0SWQiOjEwODMsIm-
FjY291bnRUeXBlIjoiVVNFUiIsImlhdCI6MTYyODUzMTIwMn0.3y85K-MpNgZZqsw3swSpkEWF4nhCkIZZ_NfL2Vp4Huk" -H  "accept: image/tiff" -H  "X-LLE-TOKEN: eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp-
XVCJ9.eyJ1aWQiOiJsc2NpIiwic2VjcmV0SWQiOjEwODMsImFjY291bnRUeXBlIjoiVVNFUiIsImlhdCI6MTYyODUzMTIwMn0.3y85K-MpNgZZqsw3swSpkEWF4nhCkIZZ_NfL2Vp4Huk"
```

Request URL

```
https://lle-devweb-1.lle.rochester.edu/api/image/get/subtracted/80359/tiff/P510?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1aWQiOiJsc2NpIiwic2VjcmV0SWQiOjEwODM-
sImFjY291bnRUeXBlIjoiVVNFUiIsImlhdCI6MTYyODUzMTIwMn0.3y85K-MpNgZZqsw3swSpkEWF4nhCkIZZ_NfL2Vp4Huk
```

Fig 4. View of Swagger API client showing the interface for accessing P510 HDF images

6. Conclusion

This work focused on finding new ways to store, manipulate, and return image data for PIs. It was found that processing image data directly in a database is not an ideal option. The proposed solution of using a Python data handler that is called by a Typescript Node.js server seems to be the best option for user experience, ease of processing, and speed.

7. Acknowledgments

I would like to start off by thanking Dr. Craxton for organizing this amazing summer program. It has been an incredible opportunity for my academic growth. I would like to thank Ms. Trubeger for doing the behind-the-scenes administrative work and organizing all of our Zoom meetings. I would like to thank my advisor Mr. Kidder for the countless hours he spent on Zoom with me helping figure bugs out and helping guide me through the project. I also wouldn’t have been able to do this without the help from the tremendous web and database team: Andy Zeller, Russ Edwards, Riley Adams, Barb Tate, and Tyler Coppenbarger. It was also great to compare with the other students in the program about what we have done.

8. References

1. https://supprt.hdfgroup.org/products/hdf4
2. https://supprt.hdfgroup.org/products/hdf5
3. https://docs.oracle.com/cd/B19306_01/server.102/b14220/intro.htm
4. https://www.oracle.com/database/technologies/appdev/plsql.html
5. https://csrc.nist.gov/glossary/term/hash_function
6. https://python.org
7. https://numpy.org

8. https://pillow.readthedocs.io/en/stable/

9. https://www.h5py.org

10. https://aws.amazon.com/what-is/restful-api

11. https://nodejs.org

12. https://expressjs.com

13. https://swagger.io

14. https://json.org