



SOFTWARE TOOL ARTICLE

REVISED Sherlock: an open-source data platform to store, analyze and integrate Big Data for computational biologists

[version 3; peer review: 2 approved]

Balazs Bohar^{1,2}, David Fazekas^{1,2}, Matthew Madgwick^{1,3}, Luca Csabai^{1,2},
Marton Olbei^{1,3}, Tamás Korcsmáros¹⁻⁴, Mate Szalay-Beko¹

¹Earlham Institute, Norwich Research Park, Norwich, UK²Department of Genetics, Eotvos Lorand University, Budapest, Hungary³Gut Microbes and Health Programme, Quadram Institute Bioscience, Norwich Research Park, Norwich, UK⁴Department of Metabolism, Digestion and Reproduction, Imperial College London, London, UK

V3 First published: 21 May 2021, 10:409
<https://doi.org/10.12688/f1000research.52791.1>
 Second version: 10 Aug 2022, 10:409
<https://doi.org/10.12688/f1000research.52791.2>
 Latest published: 12 Jan 2023, 10:409
<https://doi.org/10.12688/f1000research.52791.3>

Abstract

In the era of Big Data, data collection underpins biological research more than ever before. In many cases, this can be as time-consuming as the analysis itself. It requires downloading multiple public databases with various data structures, and in general, spending days preparing the data before answering any biological questions. Here, we introduce Sherlock, an open-source, cloud-based big data platform (<https://earlham-sherlock.github.io/>) to solve this problem. Sherlock provides a gap-filling way for computational biologists to store, convert, query, share and generate biology data while ultimately streamlining bioinformatics data management. The Sherlock platform offers a simple interface to leverage big data technologies, such as Docker and PrestoDB. Sherlock is designed to enable users to analyze, process, query and extract information from extremely complex and large data sets. Furthermore, Sherlock can handle different structured data (interaction, localization, or genomic sequence) from several sources and convert them to a common optimized storage format, for example, the Optimized Row Columnar (ORC). This format facilitates Sherlock's ability to quickly and efficiently execute distributed analytical queries on extremely large data files and share datasets between teams. The Sherlock platform is freely available on GitHub, and contains specific loader scripts for structured data sources of genomics, interaction and expression databases. With these loader scripts, users can easily and quickly create and work with specific file formats, such as JavaScript Object Notation (JSON) or ORC. For computational biology and large-scale bioinformatics projects, Sherlock provides an open-source platform empowering data management, analytics, integration and collaboration through modern big data technologies.

Open Peer Review**Approval Status**

	1	2
version 3 (revision) 12 Jan 2023		
version 2 (revision) 10 Aug 2022	 view 	 view
version 1 21 May 2021	 view	 view

1. **John H. Morris** , University of California, San Francisco, San Francisco, USA

2. **Nadezhda Doncheva** , University of Copenhagen, Copenhagen, Denmark

Any reports and responses or comments on the article can be found at the end of the article.

Keywords

Software, Computational biology, Network biology, Systems biology, Data lake, big data, data management, data integration



This article is included in the **Artificial Intelligence and Machine Learning** gateway.



This article is included in the **Bioinformatics** gateway.

Corresponding author: Tamás Korcsmáros (t.korcsmaros@imperial.ac.uk)

Author roles: **Bohar B:** Conceptualization, Data Curation, Methodology, Software, Writing – Original Draft Preparation, Writing – Review & Editing; **Fazekas D:** Conceptualization, Data Curation, Methodology, Resources, Software, Writing – Review & Editing; **Madgwick M:** Methodology, Software, Writing – Review & Editing; **Csabai L:** Software, Writing – Review & Editing; **Olbei M:** Software, Writing – Original Draft Preparation, Writing – Review & Editing; **Korcsmáros T:** Conceptualization, Data Curation, Project Administration, Resources, Supervision, Writing – Review & Editing; **Szalay-Beko M:** Conceptualization, Data Curation, Methodology, Resources, Software, Supervision, Writing – Original Draft Preparation, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

Grant information: MM is supported by a Biotechnological and Biosciences Research Council (BBSRC) funded Norwich Research Park Biosciences Doctoral Training Partnership (grant number BB/S50743X/1), as an NPIF Award. MO was supported by the BBSRC Norwich Research Park Biosciences Doctoral Training Partnership (grant BB/M011216/1). The work of T.K. and M.Sz.B was supported by the Earlham Institute (Norwich, UK) in partnership with the Quadram Institute (Norwich, UK) and strategically supported by the UKRI BBSRC UK grants (BB/J004529/1, BB/P016774/1, and BB/CSP17270/1). T.K. was also supported by a BBSRC ISP grant for Gut Microbes and Health BB/R012490/1 and its constituent projects, BBS/E/F/000PR10353 and BBS/E/F/000PR10355.

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Copyright: © 2023 Bohar B *et al.* This is an open access article distributed under the terms of the **Creative Commons Attribution License**, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

How to cite this article: Bohar B, Fazekas D, Madgwick M *et al.* **Sherlock: an open-source data platform to store, analyze and integrate Big Data for computational biologists [version 3; peer review: 2 approved]** F1000Research 2023, 10:409 <https://doi.org/10.12688/f1000research.52791.3>

First published: 21 May 2021, 10:409 <https://doi.org/10.12688/f1000research.52791.1>

REVISED Amendments from Version 1

In this new version we fixed some spelling and grammar mistakes. We changed the figure of the first use case (ID Mapping) and the text of it accordingly.

Any further responses from the reviewers can be found at the end of the article

Introduction

Most bioinformatics projects start with gathering a lot of data. As bioinformaticians work on bespoke or public datasets (for example, gene expression or mutation datasets), they require external reference data in almost all cases. Most projects may call for genome annotations, gene ontologies, tissue-specific expression datasets, drug-related databases, and many other reference datasets. One of the reasons why we use the term 'bioinformatics' in the first place is because we cannot correlate and process all these datasets manually. We need the help and the power of computers and databases (Greene *et al.*, 2014). Thanks to the current technical advancement, many solutions exist worldwide to utilize the available computational possibilities (Marx, 2013).

Cloud storage solutions such as Amazon's S3 (<https://aws.amazon.com/s3/>) or Google Cloud Storage (<https://cloud.google.com/storage>) offer scalability and flexibility to the matching compute solutions. More importantly, they allow the storage of large datasets on the same infrastructure as the large-scale analyses, with the additional benefit of allowing data sharing across teams and collaborators. Some companies have utilized cloud resources to offer storage, access to shared datasets, and transparent data sharing. Cloud storage also improves reliability, as the data is backed up in several geographical locations (Kasson, 2013). However, in most of the cases, these cloud storage companies only offer data storage solutions, but this does not include platforms to execute or work with data.

To address these issues, we repurposed concepts and open source tools from the top players of the software industry, like Facebook (Meta), Netflix and Amazon. To replace the tedious process of manual data collection before the first steps of any bioinformatics project, we developed a new platform, Sherlock.

The Sherlock platform has two main parts:

- a query engine, which is responsible for executing the given Structured Query Language (SQL) queries
- a Data Lake, required for data storage, which consists of multiple different databases or datasets, where the Query engine can execute queries against the data. The combination of these two parts in Sherlock streamlines efficient data collection, data integration, and data preprocessing (Khine & Wang, 2018).

Implementation and operation

Overview of the Sherlock platform

Sherlock is an open source and freely accessible platform that combines several different software technologies (<https://earlham-sherlock.github.io/>). One core software that Sherlock uses is Docker and the Docker swarm. It is essential to clarify that Sherlock is only a platform, which means it does not include data; instead, it provides a dockerized platform where one can work on and manipulate the data (Figure 1). Then by leveraging robust database architectures (Presto query engine and Hive metastore) and providing them with a channel to connect to the Data Lake (where the data is stored), the user can run different analytical queries on top of the data files.

As described in the introduction, Sherlock is flexible and can handle bespoke or public data. On the one hand, it can convert any bespoke structured data in a table into a standard file format used by a platform. On the other hand, with the scripts in the GitHub repository (<https://github.com/earlham-sherlock/earlham-sherlock.github.io/tree/master/loaders>), Sherlock can work with different publicly available databases as well.

Architecture

Docker & Docker Swarm. Docker is an open and widely used technology to isolate Linux processes and provide them with a reproducible, well-defined runtime environment independent of the operating system (Matthias & Kane, 2015). Each element in the Sherlock platform is running inside a separate Docker container. A Docker container is a standard unit of software that packages up all of the necessary code and all the dependencies, so the application runs quickly and reliably in an isolated environment on any machine. Container orchestration tools are used when a complex solution (like Sherlock) requires the interconnection of many docker containers distributed among multiple separate Linux machines. In Sherlock, we are using the standard Docker Swarm orchestration platform (Smith, 2017), as it is easier to use and

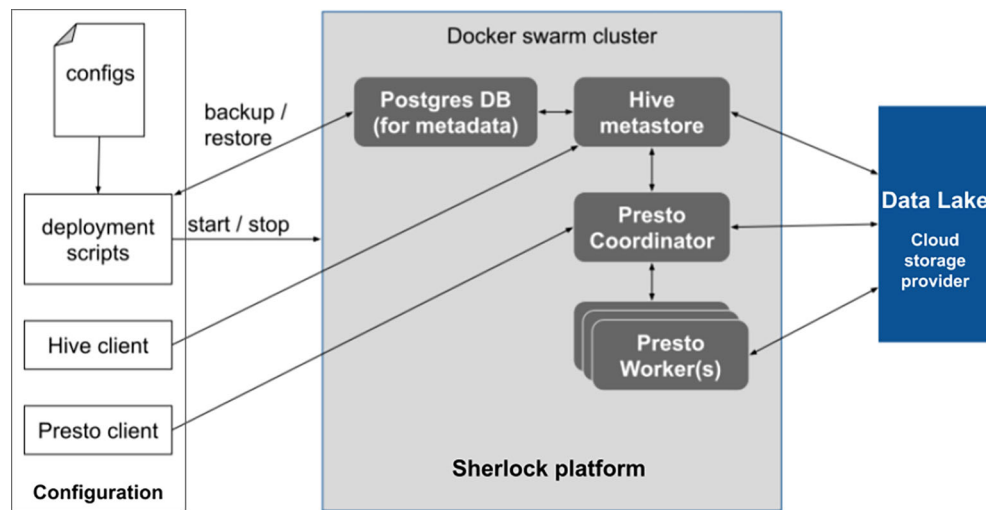


Figure 1. Overview of the Sherlock platform and its relationship with the Data Lake. The schematic shows how the different tools and the deployment modules interact inside the core of Sherlock. Only the Hive Metastore, the Presto Query Engine and the possible worker nodes have a connection to the Data Lake. The blue box represents the Data Lake, where the data is stored.

operate than other orchestration frameworks (like Kubernetes (<https://kubernetes.io>)). These industry-standard technologies make Sherlock's management (starting, stopping and monitoring) straightforward while enabling us to implement more advanced features, for example, the live on-demand scaling of the Sherlock's analytical cluster. This scalability is the most significant advantage of Docker Swarm. The entire cluster can be shut down or reduced to the minimum requirements in the cloud when it is not used. Then, it can be scaled up to hundreds of nodes if necessary for executing very complex analytical queries. Sherlock uses a hierarchical structure to achieve this, where several worker nodes and at least one manager node are responsible for handling the worker nodes' resources and ensuring that the cluster operates efficiently.

Query Engine. The first core part of Sherlock is the Query Engine (Figure 2), which is responsible for running the given question through SQL commands on top of the data files and retrieving the 'answer' for the user. In general, query engines (Hive, Impala, Presto) "sit" on top of a database or server and execute queries against data. More specifically, many of them use SQL query engines for C-R-U-D (create, read, update, delete) operations. Query engines facilitate access to the data in relational systems and enforce data policies that relational data models and database management systems require. In Sherlock, we are using the Presto Query Engine (<https://prestodb.io>), developed by Facebook. Presto provides a high-performance Query Engine wrapped in an easily deployable and maintainable package. Furthermore, the Presto Query Engine can handle many different types of data storage solutions.

With Presto, we can formalize analytical questions using SQL. SQL can be used for stream processing in a relational data stream management system (RDSMS). Moreover, one of SQL's most beneficial features is handling structured data, such as data that resides in a fixed field within a table, allowing us to easily search and manage the data. The data can be arranged and analyzed in various ways, such as sorting alphabetically or totalling a set of values (Silva *et al.*, 2016). When combined with a Query Engine, SQL can be used to connect to the Data Lake, read and integrate the data stored within it and execute different analytical queries, either sequentially or in parallel, depending on the power of the given machine.

Hive Metastore. The second core part of Sherlock is the Hive metastore (<https://docs.cloudera.com/runtime/7.0.1/hive-metastore/topics/hive-hms-introduction.html>), which is a service that stores metadata related to Apache Hive and other services, in a backend RDBMS, such as MySQL or PostgreSQL. In Sherlock, the Hive metastore contains only the meta-information about the different tables (their schema or location) in the Data Lake for the Presto Query Engine to query. The Hive metastore is the intermediate layer between the Presto query engine and the data lake, enabling Presto to access the meta-information and query the desired data. Sherlock provides simple scripts to save this metadata in the Data Lake, either when the user wants to make a backup or before the termination of the analytical cluster.

Deployment

For Sherlock, we developed a dockerized version of Presto that also incorporates the Hive metastore. This addition of the query engine and metastore makes Sherlock cloud-agnostic, so it can be installed at any cloud provider, providing it in a

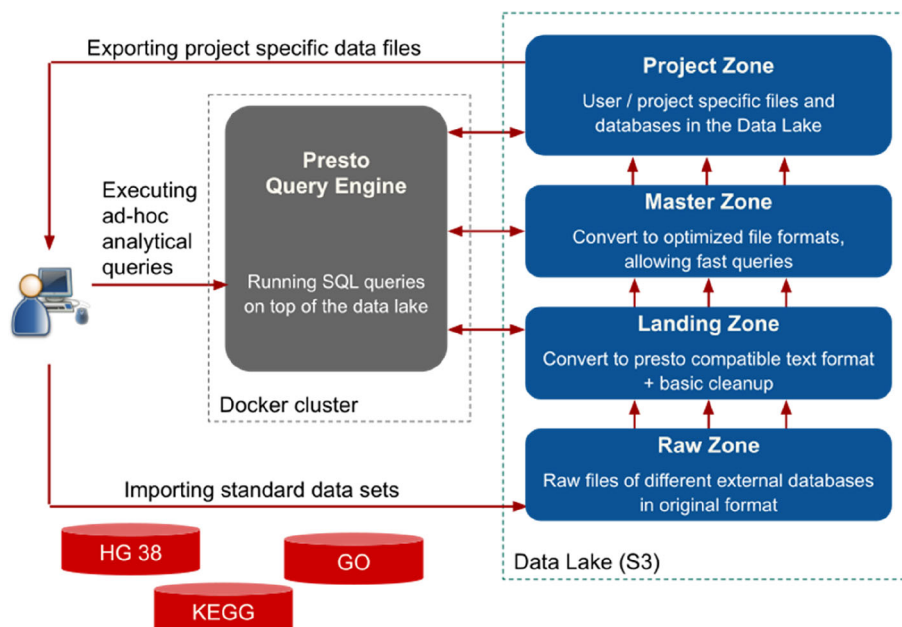


Figure 2. A schematic representation of the relationship between Sherlock's two main components: the Query Engine (dockerized) and the Data Lake. Left side: The core component of the platform: the Presto Query Engine. It allows the user to execute different analytical SQL queries on top of the data files in the Data Lake (right side). Right side: The structure of our Data Lake. They are separated into four different zones. Working from bottom to top; 1) The Raw Zone contains the raw data from the different external or bespoke databases in their original formats. 2) The landing Zone, where the data is in JSON Lines format, is compatible with the Presto Query Engine. 3) The master zone, where the data is in a common optimized and compressed format, is called ORC. This format enables faster query execution than the landing zone. 4) The Project Zone, where we store exclusively the data needed for specific/active projects.

Linux machine with Docker installed. The advantage of running Presto in a dockerized environment is that it enables the automatic installation and configuration of the whole Presto Query Engine. Furthermore, the whole platform can be fired up on a local machine, multiple machines or any cloud service. We show how to make a set of Linux virtual machines with Docker installed, then start a distributed Presto cluster by using the Sherlock platform under the deployment section of the GitHub page of Sherlock (https://earlham-sherlock.github.io/docs/deployment_guide.html).

The Data Lake – data storage

A Data Lake is a simple network storage repository that can be accessed by external machines (<https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/>). All the data imported into, transformed in, or exported from Sherlock will be stored here as simple data files. The most significant advantage of using a Data Lake is that its operation will be identical on a local machine as on the cloud. Therefore, the data stored in the Data Lake is well-structured and placed in reliable storage. Furthermore, the Data Lake can be scaled independently from the query engine. The technologies in Sherlock are compatible with both most common Data Lake solutions, Hadoop Distributed File System (HDFS), a distributed file system designed to run on commodity hardware, and the Simple Storage Service (S3) storage formats. However, we only described S3 in all of our examples, as S3 is more widely-used, modern, and accessible. Although HDFS is an older standard solution, it has some compelling features which can result in better performance. Still, it is much more challenging to set up, maintain and in most cases, its extra features are not necessary for the given project. In contrast, S3 is a standard remote storage API (Application Programming Interface) format, introduced first by Amazon (<https://aws.amazon.com/s3/>). As of writing, users can purchase S3 storage as a service from all major cloud providers like Digital Ocean, Amazon AWS, Google Cloud or Microsoft Azure. Each of these can be compatible with the Sherlock platform.

Having somewhere to store the data is only half of having an operational Data Lake. One can not stress enough how important it is to organize the data in a well-defined 'folder' structure. Many Data Lake deployments become unusable after a few years due to poor maintenance, resulting in a large amount of data 'lost' in the thousands of folders or inconsistent file formats and no ownership over the data. Our solution is to separate all of the data in the Data Lake into four different main folders, representing different stages of the data and different access patterns in the Data Lake. Inside

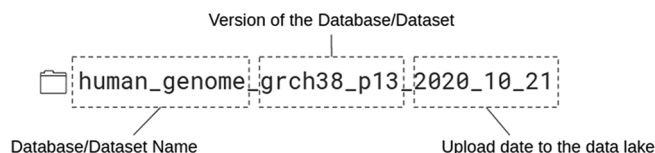


Figure 3. An example naming convention for data stored inside the Data Lake. The first part of the folder name is the name of the given database/dataset. In this example case, the human genome. The next part is the version of the data, and the last is the uploaded date to the Data Lake. <https://jsonlines.org/> <https://orc.apache.org/docs/>.

each of these main folders, we can create subfolders, and it is a good practice to incorporate the name of the dataset, the data owner name, the creation date (or other version info), and the file formats somehow into their paths.

We separated our Data Lake into four main zones, which are built on top of each other (Figure 2). The first zone is the raw zone. We archived all the database files into the raw zone in their original formats. For example: if we downloaded the human genome, then we put the fasta files here, under a separate subfolder. The subfolder's name should contain the exact version (for example: hg38_p12) (Figure 3), and also, we put a small readme file in the folder, where we listed some metadata, like the date and the URL of the download, etc. Usually, these files cannot be opened with Presto, as the file format is incompatible in most cases.

The next zone is the landing zone. We needed to develop specific loader scripts, which convert and extract the raw datasets into the landing zone. We converted the data to a text file in JSON Lines format (<https://jsonlines.org/>), which Presto can open. JSON Lines is a specific JSON format, where each text file line represents a single JSON record. Each JSON file for a given dataset is placed into a separate sub-folder in the landing zone. It is then registered by Presto which sees the dataset now as a table. Then, Presto will be able to load the data and perform other operations, for example it can execute simple data transformations. However, we do not recommend using the tables in this zone for querying because processing the large JSON files is very slow.

Using Presto, we converted the data from the landing zone into an optimized (ordered, indexed, binary) format for the next zone, which is the master zone. The main idea is that we use the tables in the master zone later for analytical queries. Here the data is in a more detailed and exact format, called Optimized Row Columnar (ORC), which is a free and open-source column-oriented data storage format (<https://orc.apache.org/docs/>). With ORC, Sherlock can perform SQL queries much faster than the JSON text file format from the previous zone (landing zone). If necessary, advanced bucketing or partitioning on these tables can be used to optimize the given queries. Furthermore, the master zone contains the 'single source of truth'. This means that the data here cannot be changed, only extended upon, for example, adding a new version of the datasets.

For illustrating the query time speed difference between the landing and the master zone, we ran the following SQL query in both zones 20 times (Figure 4):

```
SELECT
bgee.molecule_id, uni.to_id, bgee.molecule_id_type, bgee.tissue_uberon_id, bgee.
tissue_uberon_name, bgee.score

FROM
bgee_2021_04_05 bgee

LEFT JOIN
uniprot_id_mapping_2021_04 uni

ON
uni.from_id_type = 'ensembl_gene_id'

AND
uni.to_id_type = 'uniprotac'

AND
uni.from_id = bgee.molecule_id
```



```

WHERE
bgee.tax_id = 9606

AND
bgee.tissue_uberoid = 2113

AND
uni.to_id IS NOT NULL

ORDER BY
score

DESC

LIMIT
100;

```

This query is selecting the top 100 most highly expressed genes in a given tissue (the human liver) and their UniProt identifiers. In both zones, for this query, we used the UniProt (UniProt Consortium, 2021) mapping table (version 2021_04 from the UniProt website) and the Bgee (Bastian *et al.*, 2021) table (downloaded from the Bgee database on 5th April 2021). Both tables have a large file size and number of rows: the UniProt mapping table is 1.1 GB and it has 25,458,069 and the Bgee table is 550 MB and it has 2,876,219 rows. Despite this huge amount of data in each table, the queries running in the master zone are 20–30 times faster than the queries in the landing zone, because of the ORC format.

Additionally, we benchmarked the master zone and Pandas, an industry-standard python package used for data wrangling (<https://pandas.pydata.org>), to compare their performance. For this comparison, we converted the landing zone tables into tab-separated value (TSV) format and generated the same query as seen above in Pandas. We repeated this experiment 20 times to select the human liver's top 100 most highly expressed genes as outlined in the query above. We found that Sherlock is still 3–4 times quicker than pandas at performing the example query (Figure 4).

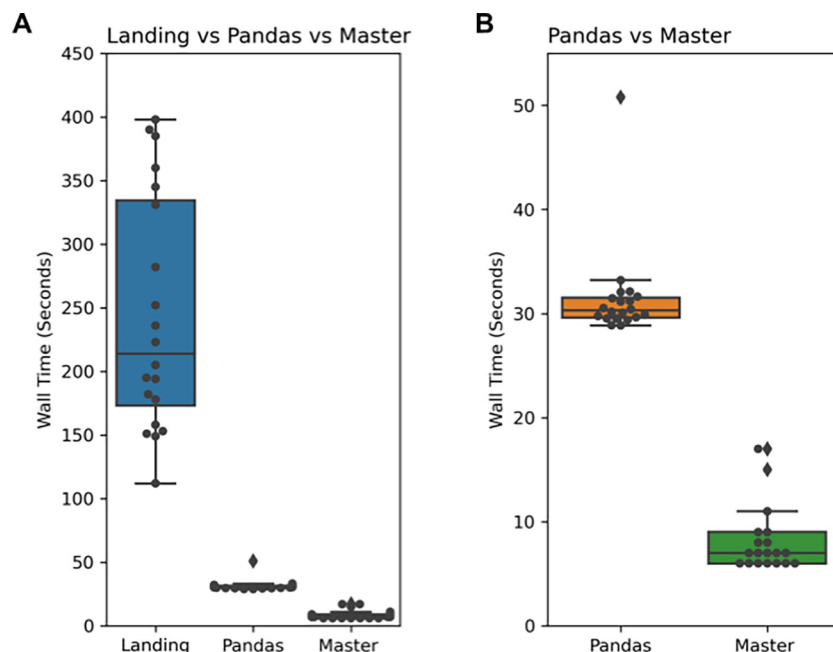


Figure 4. Results of the average run time (in seconds) of executing 20 queries on the landing zone vs Pandas vs the master zone. A: The wall time comparison between the three different approaches for 20 repeated queries. After evaluating the run time performance, queries running in the master zone are ~20–30 times faster than those in the landing zone and ~3–4 times faster than Pandas. The significant deviation in the landing zone is due to the connection with S3 on Digital Ocean. During longer queries, it can result in a bottleneck hence the need for master zone's optimized data format. B: The speed difference between Pandas and the master zone's ORC format. The master zone queries ~3–4 times quicker than Pandas. Therefore, it demonstrates the performance increase of adopting the ORC format in the master zone, which enables this increase in performance.

The last zone is the Project zone. This zone saves and holds the tables needed only for specific projects. We can even create multiple project zones, for each group, project or user. However, it is crucial to have a rule implemented to indicate the owner of the tables; as mentioned before, this dramatically increases the effectiveness of Sherlock and ease of maintenance.

The main reason for having this multi-level folder structure is because of reusability. With the raw and processed data in the different zones, the users can easily access all the data and there is no need to spend unnecessary time acquiring them again. The other part is that the data is more specific and compressed in the upper zones, allowing for running fast and efficient SQL queries. The other advantage is that storage space is cheaper compared to computing capacity. In light of this, if we store our data in several copies, we do not have to pay significantly more. This allows us to keep the raw data and increase the chance of reusability.

Functionality of Sherlock

Here, we describe the key steps of how the query engine and the Data Lake can be used together (Figure 5). The first step is to set up the platform itself by starting a Docker Swarm with the deployment modules on a cluster with several worker nodes. These modules start different services, but each in a separate container. These deployment modules are configurable (Figure 1). One can specify exactly how many resources a service can use simultaneously (e.g. CPU and memory allocation). Inside a running Docker Swarm, Sherlock has two main parts: the Hive Metastore and the Presto query engine. Sherlock follows the industry best practices in its architecture to enable scalability. Usually, the main idea behind scalable batch processing architectures is to separate data storage and data analytics (Dean & Ghemawat, 2008). This architecture allows us to scale the analytical power and the storage size independently and even dynamically. This is the case with Sherlock when deployed in the cloud. The basic order of operations is as follows; an SQL query is submitted to the Presto query engine, then it will connect to the Data Lake and fetch the necessary data files. Presto executes the in-memory distributed query and then it returns the results to the user (Figure 5).

Sherlock has been designed for computational biologists, especially for network and systems biologists. It can contain specific interaction, expression and genome data-related databases with the help of the loader scripts that can create the specific file formats from the source databases and upload them into the Data Lake. With these individual loader scripts, users can make and work with the necessary file formats from the different data sources without having to develop their own loading scripts. The whole source code of the platform and the loader scripts (Table 1) are freely available on GitHub: <https://github.com/earlham-sherlock/earlham-sherlock.github.io>.

It cannot be emphasized enough that maintaining the data and a well-defined folder structure in the Data Lake is essential. Furthermore, keeping the folders and the tables in the Data Lake clean and very documented is vital. Often a user may not have been the one who uploaded or generated the data, which sometimes makes it difficult to gain an understanding of what a table contains. Without this knowledge, a user would struggle to formulate a query. There are two straightforward solutions to this. We provide a Web User Interface for Sherlock where a user can check the Schemas of tables held in the Data Lake. Alternatively, a more technical one, where the user can check the columns' names using a simple SQL query. From our experience, however, the best practice to check the content of the different tables is to create a wiki or a well-documented description of the master (query) zone, where everybody can find the needed information and the column structure of the different tables.

In the next section, we will outline three different use cases of Sherlock, describing how to set up the Sherlock platform and how to load data into its Data Lake.

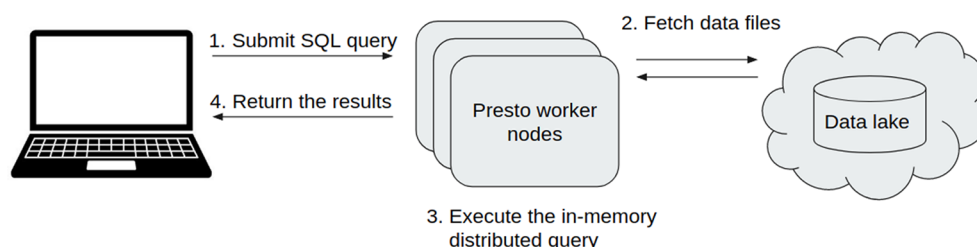


Figure 5. Overview of how the query engine and the Data Lake work together. This schematic represents the different steps, how a user can submit an analytical SQL query and a summary of the workflow of this query until the results are returned to the user.

Table 1. The specific loader scripts, which are already included in Sherlock's GitHub repository.

Datatype	Loader script	Source	Reference
General datasets	DBSnp database	https://www.ncbi.nlm.nih.gov/snp/	(Smigielski <i>et al.</i> , 2000)
	Gene Ontology	http://geneontology.org	(Ashburner <i>et al.</i> , 2000)
	Gene Ontology Annotations		
	Human Genome	https://www.gencodegenes.org	-
	Uberon Gene Ontology	https://www.ebi.ac.uk/ols/ontologies/uberont	(Mungall <i>et al.</i> , 2012)
	Uniprot ID Mapping data	https://www.uniprot.org	(UniProt Consortium, 2021)
Interaction databases	BioPlex database	https://bioplex.hms.harvard.edu	(Huttlin <i>et al.</i> , 2020)
	Dorothea database	https://dorothea.opentargets.io/#/	(Garcia-Alonso <i>et al.</i> , 2018)
	HINT database	http://hint.yulab.org	(Das & Yu, 2012)
	HuRI database	http://www.interactome-atlas.org	(Luck <i>et al.</i> , 2020)
	InBioMap database	https://inbio-discover.com	(Li <i>et al.</i> , 2017)
	IntAct database	https://www.ebi.ac.uk/intact/	(Orchard <i>et al.</i> , 2014)
	IRefIndex database	http://irefindex.uio.no	(Razick <i>et al.</i> , 2008)
	Mentha database	https://mentha.uniroma2.it	(Calderone <i>et al.</i> , 2013)
	Omnipath database	https://omnipathdb.org	(Türei <i>et al.</i> , 2021)
	STRING database	https://string-db.org	(Szklarczyk <i>et al.</i> , 2019)
Expression data	Bgee database	https://bgee.org	(Bastian <i>et al.</i> , 2021)

Use cases

Platform and Data Lake set up

Before we introduce some relatively simple use cases of how the Sherlock platform can be used with a Data Lake, we will first explain the fundamentals of the platform. This consists of downloading the GitHub repository, loading data to the Data Lake with the help of the different loader scripts and using the SQL language to execute analytical queries (Figure 6).

As described in the previous section, the first step of using the platform is to set it up by downloading the GitHub repository (<https://github.com/earlham-sherlock/earlham-sherlock.github.io>). Within the repository, extensive documentation and a readme file describe how to commission the platform on various solutions. This includes on a single laptop/computer or a cluster hosted by a cloud provider. After the successful operational set-up step, data can now be uploaded to the Data Lake. First, we have to download the whole database of interest and upload them to the raw zone of the Data Lake. With the help of the loader scripts, we can convert the raw database files into JSON file formats, which can be uploaded and registered in the landing zone with PrestoDB using a simple SQL query. An example query can be found on the GitHub repository (https://github.com/earlham-sherlock/earlham-sherlock.github.io/tree/master/table_definitions/landing_zone). This step allows the Hive Metastore to see the structure of the tables in the landing zone (columns of the tables). Because of this, we can query this zone as well, but as we highlighted before, it will be much slower than the master zone. With simple SQL queries, we can convert the JSON tables from the landing zone into the ORC format and upload them to the master zone. An example of this can be found on the GitHub repository (https://github.com/earlham-sherlock/earlham-sherlock.github.io/tree/master/table_definitions/master_zone).

After this, with the help of the configuration and setup modules, we have an operational platform connected to our data-filled Data Lake using the raw databases/files of interest. From this stage, there are a couple of opportunities to use the platform:

1) download the command line interface of PrestoDB (<https://prestodb.io/docs/current/installation/cli.html>) and in terminal connect to the query engine with a simple command:

```
./presto --server localhost:8089 --catalog sherlock
```

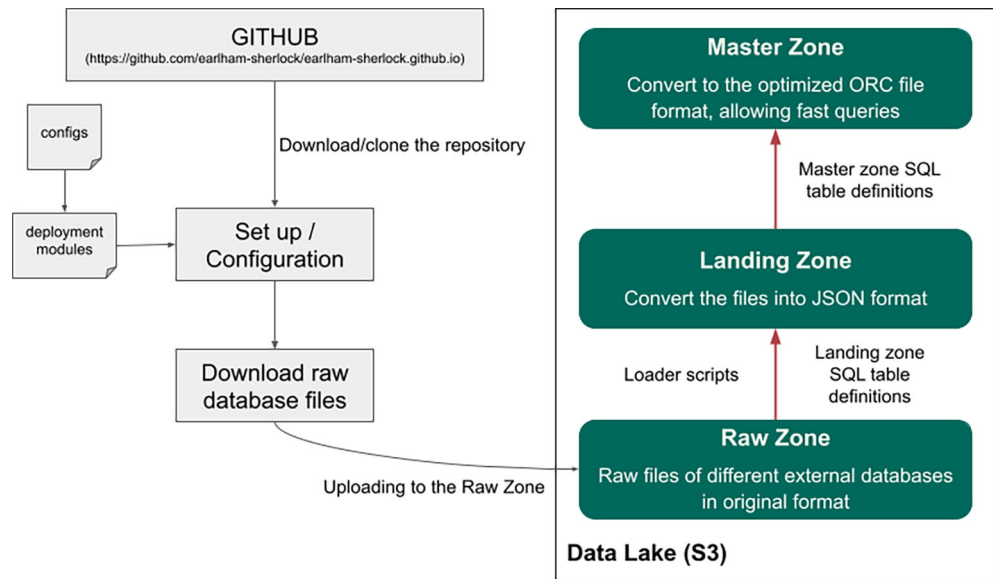


Figure 6. This figure represents the different steps how the whole platform can be configured and how data can be uploaded to the Data Lake.

2) Sherlock provides a simple Web User Interface (Web UI) for the coordinator virtual machine. This interface is not part of the general Presto Query Engine, it is an open-source and freely available tool, which is running in a separate docker container alongside the Sherlock platform (<https://yanagishima.github.io/yanagishima/>). This can be linked together by adding the bind address of the ssh command:

```
ssh {coordinator virtual node IP/name} -L 8090:localhost:8090
```

Then opening a web browser and going to the localhost:8090 URL, the Web User Interface will come up (Figure 7).

This Web UI has a simple user interface and some useful features that can help the users handle and manipulate the data inside the different zones without using a terminal. It has a built-in command line interface (CLI), where users can write their SQL queries. As we described earlier, it can give information about the contents of different tables (what the columns

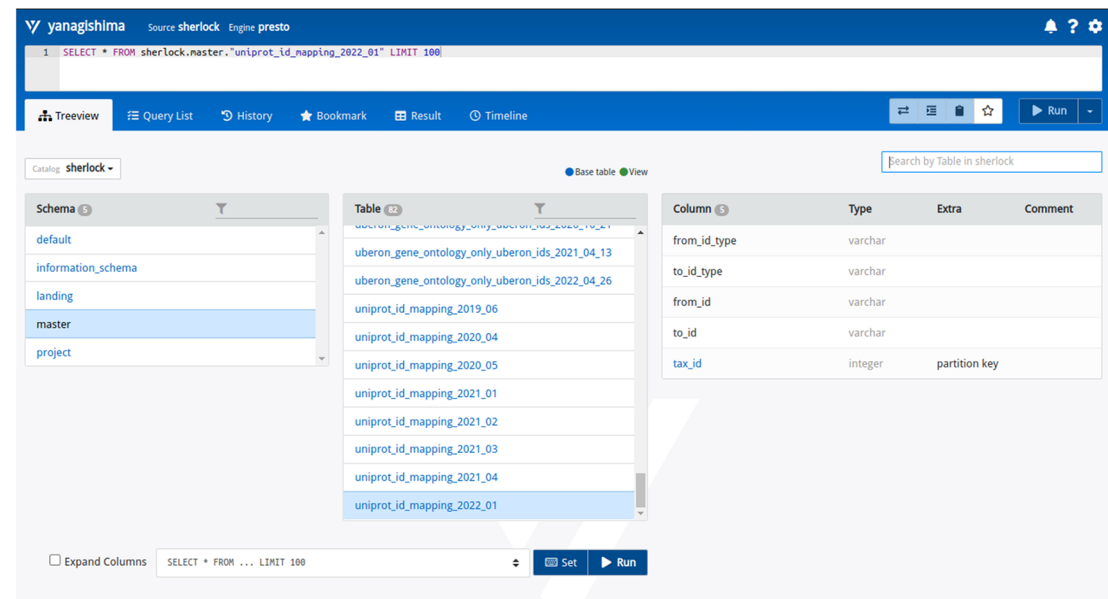


Figure 7. The Web User Interface (Web UI) for Sherlock. At the top, there is a command line interface where the user can write SQL queries manually. From left to right are the zones (schemas), the tables inside the zones, and the column structure of the table of interest.

are) with the ‘Treeview’ menu point. The Web UI has a history feature that can help to search back the latest SQL queries and save the time to rewrite them. Furthermore, frequently used and essential queries can be bookmarked again to save the users time. Last but not least, after a user runs the given analytical SQL query or queries, they will get back the results and can download them in a table format.

In the following subsections, we will highlight three use cases of using the platform with our Data Lake, which contains the databases collected in [Table 1](#).

Use case 1: Identifier (ID) mapping

In this use case the goal is to show how Sherlock can run simple SQL queries with merging tables to map different identifiers. For this first use case, use the STRING table (downloaded on the 9th of September 2021), the UniProt mapping table (with version 2021_03 from the UniProt website) and the Bgee table (downloaded on the 5th of April 2021). All of the three tables were uploaded to the landing zone and then converted to the master zone using the steps described in the previous section. One of the most repetitive and crucial tasks for those working in bioinformatics is identifier (ID) mapping. Working with many different datasets from multiple sources, all carrying diverse identifiers, is challenging. In addition, these identifiers can have clashing structures, making it complicated and time-consuming to work with them - resulting in the development of different scripts or the use of various tools to work with them simultaneously. Users have to make different scripts or use various tools to work with these identifiers at once. Nowadays, the principal idea behind these ID mapping steps is to have a separate table or tables, called mapping tables, which contain the different identifiers. The best practice is to have only one mapping table, which includes all the necessary identifiers for a given project. The limitation of this approach is that when a team has to work with so many different identifiers at once, this mapping table can be large, which increases the computational time to extract the data from the mapping table. In Sherlock, users can work with just one mapping table, which can be made quickly with the help of the provided loader scripts from the GitHub repository, which can significantly shorten the time needed for ID mapping. Sherlock can execute these queries very quickly despite the large size of the mapping tables, owing to the implemented ORC format. To demonstrate this capability in Sherlock, we search for genes from the STRING table ([Szklarczyk et al., 2019](#)), and we would like to get the UniProt identifiers of those genes, which expressed in the brain (using the Bgee table as well).

Three different datasets are needed to execute this example SQL query. Table one contains the information from the STRING database (not showing all of the columns from the actual table), the second table is a mapping table containing the mapping between the different types of protein identifiers from different sources (Ensembl (https://www.ensembl.org/info/genome/stable_ids/index.html) and UniProt (https://www.uniprot.org/help/accession_numbers)). The third table (Bgee_2021_04_05) contains the tissue specific informations of the proteins with their UniProt identifiers ([Figure 8](#)). With the following example query, Sherlock can map between different identifiers, using these three tables from the master zone.

```
SELECT
to_id

FROM
master.string_2021_09_09_human string

LEFT JOIN
master.uniprot_id_mapping_2021_03 uniprot

ON
string.interactor_a_id = uniprot.from_id

LEFT JOIN
master.bgee_2021_04_05 bgee

ON
uniprot-to_id = bgee.uniprot_id
```

The SQL query will connect the two tables through matching protein identifiers, selecting only those expressed in the brain according to the matching identifiers (light green and purple in [Figure 8](#)). The query results in the first two proteins from the master.string_2021_09_09_human table (light green color). If the query does not find any matches between the Ensembl and UniProt IDs, it skips them. This is demonstrated in [Figure 8](#), where the third protein in master-string_2021_09_09_human table does not have a match in the uniprot_id_mapping_2021_03 table.

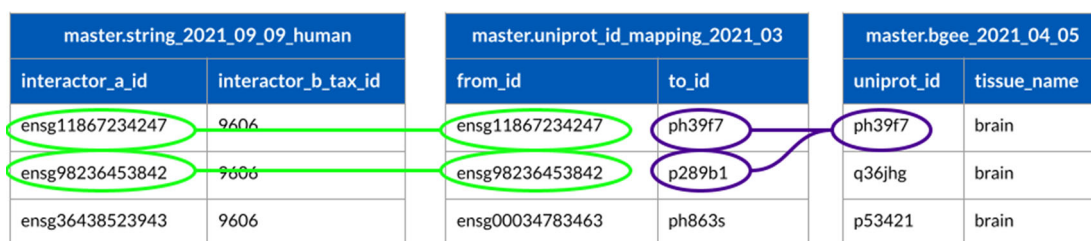


Figure 8. This schematics describing the mapping between identifiers of two tables in the master zone ('string_2021_09_09_human', which contains the human interactions from the STRING database, 'uniprot_id_mapping_2021_03', which contains the mapping information between different types of identifiers and 'bgee_2021_04_05', which contains the tissue information of the proteins).

Use case 2: Tissue specificity

In this case, we would like to query the top 100 most highly expressed genes in a given tissue (the human liver) and their protein interactors. The limitation of this use case is similar to the previous one: the user has to download the differently structured interaction databases from web resources and then write new scripts, or use online tools, to get the data into a workable format. These are also prerequisite steps with the Sherlock platform, e.g. having to download the different sources and run preprocessing steps before working with them. However, with the provided loader scripts, the user only has to do it once, which is less time-consuming. With Sherlock, the user can get this data from multiple interaction tables at once very quickly, thanks to the ORC format. In this example, we are using the BGee table and the OmniPath table from the master zone of the Data Lake. The BGee table was downloaded on the 5th of April 2021 and the OmniPath table was downloaded on the 13th of September 2021. Both tables were created and formatted with the steps from the beginning of the User Cases section.

With the following SQL query Sherlock can give the answer for what are the top 100 most highly expressed genes in the human liver:

```

SELECT
molecule_id, molecule_id_type, tissue_uberoid, tissue_uberoid_name, score,
interactor_a_id, interactor_b_id

FROM
master.bgee_2021_04_05 bgee

LEFT JOIN
master.omnipath_2021_09_13

ON
bgee.molecule_id = omnipath_2021_09_13.interactor_a_id

LEFT JOIN
master.omnipath_2021_09_13

ON
bgee.molecule_id = omnipath_2021_09_13.interactor_b_id

WHERE
tax_id = 9606

AND
bgee.tissue_uberoid = 2113

ORDER BY
score DESC

LIMIT
100;

```

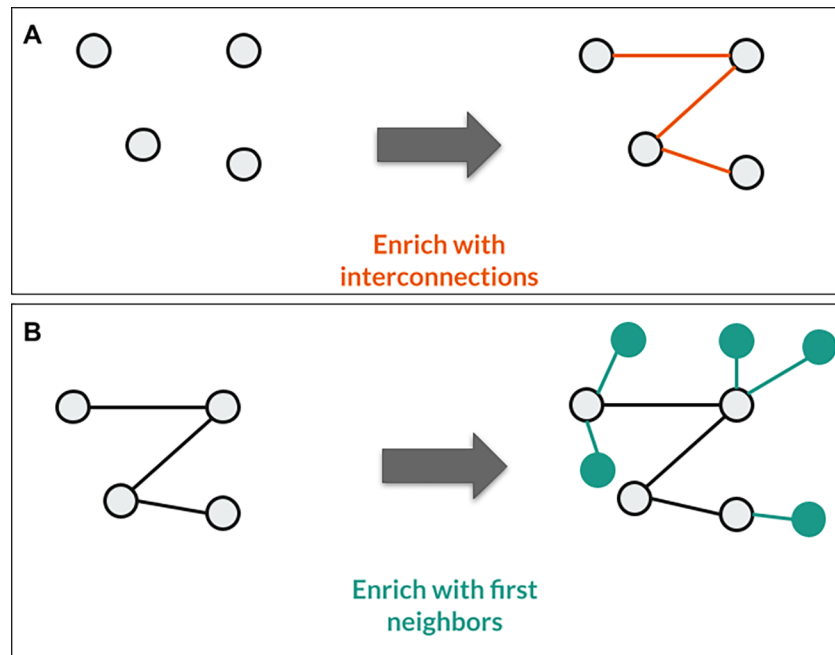


Figure 9. Representation of the third use case. A: With the given SQL query, Sherlock can find the interconnections between the selected proteins in this step. B: After a single logical operator change between the two queries (AND operator to the OR operator), Sherlock can also identify the selected proteins' first neighbors.

This query will select the top 100 highly expressed genes from a table imported from the BGee resource (Bastian *et al.*, 2021) held in the master zone. The result will be filtered to return only those genes expressed in the colon, and the query will select their protein interactors from the Omnipath interaction database resource. The results are ordered by the scores, so only the colon genes with the top 100 scores and their first neighbors will be returned.

Use case 3: Network enrichment

In the third example, the goal is to query the interconnections between specific genes of interest and their first interaction partners as well, using the OmniPath database from the Data Lake with the help of Sherlock (Figure 9). It often happens in various bioinformatics projects that one needs the direct interaction partners of a given protein, or has to find the interconnections between different ones. With the help of the Sherlock platform one can query quickly on several different databases at the same time, which are in the Data Lake. This can definitely shorten the time required to answer the biological questions.

In this example, we only use a single interaction database, the OmniPath table from the master zone as in the second use case.

The SQL query for finding the connections between the proteins (Figure 9A) is the following:

```
SELECT
interactor_a_id, interactor_b_id

FROM
master.omnipath_2020_10_04

WHERE
interactor_a_id

IN
('o95786', 'q96eq8', 'q6zsz5', 'q01113')
```

```

AND
interactor_b_id

IN
('o95786' , 'q96eq8' , 'q6zsz5' , 'q01113');

```

To find the first neighbors of the given proteins we have to use a different SQL query (Figure 9B), as below:

```

SELECT
interactor_a_id, interactor_b_id

FROM
master.omnipath_2020_10_04

WHERE
interactor_a_id

IN
('o95786' , 'q96eq8' , 'q6zsz5' , 'q01113')

OR
interactor_b_id

IN
('o95786' , 'q96eq8' , 'q6zsz5' , 'q01113');

```

Only a single logical operator changed between the two queries (AND in Figure 9A, OR in Figure 9B). The reasoning behind this is that if we want to find the interactions between the proteins, the query has to select only those interactions from the OmniPath table where the source and the target protein are uniformly included among the proteins we examined. However, in the other case, it is enough to select all interactions where the source or the target proteins are among the proteins of interest when we want to enrich the network.

Discussion

Sherlock provides a new, gap-filling method for computational biologists to store, convert, query, generate or even share biological data efficiently and quickly. This novel platform provides a simple, user-friendly interface to work with every day and widely used big data technologies, such as Docker or PrestoDB. Sherlock leverages the ORC format so users can work with extremely large datasets in a relatively short computational time. Thus, Sherlock provides a platform to facilitate data management and analytics for biological research projects.

Some other existing platforms support Big Data in computational biology, for example, Galaxy (<https://galaxyproject.org>) or AnVIL (<https://anvilproject.org>). Galaxy and AnVIL are general workflow management platforms that provide workspaces. In these workspaces, a user can run workflows of analytical programs (tools), which were developed by someone else already. Public databases are often available through import tools in Galaxy, AnVIL or even through R. Sherlock on the other hand focuses more on storing and combining data sets and allowing flexible, ad hoc queries to manipulate this data or build new data sets based on the existing ones. Sherlock does not provide any workflow system; therefore, Sherlock cannot run custom code to perform complex analytical workflows (like running simulations on molecules or executing machine learning modules). Instead, it provides some basic data manipulation (filtering, joining, sorting) which enables answering simple analytical questions. In brief, Sherlock focuses more on the data side, while Galaxy and AnVIL focus more on a research project's workflow and analytics. Ideally, one can combine the two to receive the best from both worlds.

With Sherlock, if more advanced analytics is needed, one can easily generate custom data sets in standard formats, consumable by Galaxy or AnVIL tools (e.g. JSON, CSV or ORC). The results of the analytical workflows can then be imported to Sherlock again, enabling users to easily access this data and run ad hoc queries on them. In summary, when someone needs to run the same workflow/pipeline many times, then Galaxy and AnVIL are the better options, because they provide workspaces where workflows can be executed. On the other hand, when someone needs to work with more extensive databases with the same data format and wants to run multiple analytical queries to answer biological questions, then Sherlock would be the better option because this platform focuses on storing, manipulating, building and extending upon existing datasets.

We envision a more general data access mechanism with Sherlock as it uses industry-standard formats (such as JSON and ORC). It is important to note that with Sherlock, one can work with more project-specific and more customizable data. In the Sherlock platform, users can process and integrate data with their own needs. Sherlock is not database specific and it is flexible to research group-specific needs. For example, Sherlock is perfect when a group works with only a certain number of distinct or bespoke data types and data formats.

Since we made Sherlock, plenty of similar platforms have appeared worldwide, but none of these solutions is explicitly designed for computational biologists. One of them, for example, is the Qubole platform (<https://www.qubole.com>). The Qubole platform also offers data storage and analytical query solutions, but operating the Sherlock platform is cheaper. Usually, research groups have their own servers or at least have access to one or more, and they have the opportunity to have data storage solutions. Having a server enables one to set up and use the platform itself because all the source code is freely available on GitHub and can be customized to the user's requirements. In this case, one has to pay only for the data storage space. Although Sherlock is compatible with any cloud provider, like Amazon AWS or Google Cloud, it was mainly developed and configured to the storage solutions of Digital Ocean.

To specifically support computational biologists, Sherlock contains specific database loader scripts that the users can use to create and upload the specific file formats to the Data Lake. We are constantly upgrading the already included datasets in our Data Lake. We also provide new database loader scripts in the Github repository (to extend interaction and expression data) and different mapping scripts. We will also develop different converting scripts to easily handle Sherlock compatibility with other file formats, such as TSV (tab-separated value) or CSV (comma-separated value).

Sherlock has a lot of valuable features, which are the following:

- Store all datasets in redundant and organized cloud storage
- Convert all datasets to standard, optimized file formats
- Execute analytical queries on top of data files
- Share datasets among different teams/projects
- Generate operational datasets for particular services or collaborators
- Provides a storage solution for big data computational biology projects

We intend to regularly update Sherlock's loader scripts to ensure compatibility with commonly used biological databases in the future and cover as many interactions and expression databases as possible. Furthermore, we plan to improve our source code to make more detailed documentation. Right now, updating the included databases in the Data Lake is a manual process, but we also would like to automate this with a script which can handle all of the updates at once. We would also like to include more common and general computational biology examples in the repository. To aid this, we are developing tutorials and extending the use cases of how Sherlock can be deployed and utilized for research projects. Our main goal is to disseminate the Sherlock platform widely, and we aim to enable computational and systems biologists to manage their large-scale datasets more effectively.

Conclusion

Sherlock provides an open-source platform empowering data management, analytics, and collaboration through modern big data technologies. Utilizing the dockerization of Presto and the Hive Metastore, Sherlock is not only powerful but also a flexible and fast solution to store and analyze large biological datasets effectively and efficiently. Sherlock can be used to execute queries on top of a Data Lake, where all the data is stored in a 'folder-like' structure, providing the added benefit of well-defined folder structures, which helps and encourages correct data management of biological data. With a scalable query engine and ORC format, Sherlock can run SQL queries faster than other solutions, which can significantly reduce the lead time of the research project. In conclusion, Sherlock provides a 'plug and play' state-of-the-art data storage platform for large biological datasets via repurposing concepts and open source tools created by large software companies.

Data availability

All data underlying the results are available as part of the article and no additional source data are required.

Software availability

Software available from: <https://earlham-sherlock.github.io/>

Source code available from: <https://github.com/earlham-sherlock/earlham-sherlock.github.io>

Archived source code available from: <http://doi.org/10.5281/zenodo.4738516> (Bohár *et al.*, 2021)

License: MIT

Acknowledgements

We would like to thank all of the members of the Korcsmaros Group for their advice and feedback. MM is supported by a Biotechnological and Biosciences Research Council (BBSRC) funded Norwich Research Park Biosciences Doctoral Training Partnership (grant number BB/S50743X/1), as an NPIF Award. MO was supported by the BBSRC Norwich Research Park Biosciences Doctoral Training Partnership (grant BB/M011216/1). The work of T.K. and M.Sz.B was supported by the Earlham Institute (Norwich, UK) in partnership with the Quadram Institute (Norwich, UK) and strategically supported by the UKRI BBSRC UK grants (BB/J004529/1, BB/P016774/1, and BB/CSP17270/1). T.K. was also supported by a BBSRC ISP grant for Gut Microbes and Health BB/R012490/1 and its constituent projects, BBS/E/F/000PR10353, and BBS/E/F/000PR10355. Work of M.Sz.B was supported by a UKRI-BBSRC Flexible Talent Mobility Account Fellowship at the Earlham Institute (BB/R50659X/1).

References

- Ashburner M, Ball CA, Blake JA, *et al.*: **Gene Ontology: tool for the unification of biology.** *Nat. Genet.* 2000; **25**(1): 25–29.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Bastian FB, Roux J, Niknejad A, *et al.*: **The Bgee suite: integrated curated expression atlas and comparative transcriptomics in animals.** *Nucleic Acids Res.* 2021; **49**(D1): D831–47.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Bohár B, Szalay-Beko M, Fazekas D, *et al.*: **earlham-sherlock/earlham-sherlock.github.io: First release of the official Sherlock platform (Version v1.0.0).** *Zenodo.* 2021, May 5.
[Publisher Full Text](#)
- Calderone A, Castagnoli L, Cesareni G: **mentha: a resource for browsing integrated protein-interaction networks.** *Nat. Methods.* 2013; **10**(8): 690–91.
[PubMed Abstract](#) | [Publisher Full Text](#)
- Das J, Yu H: **HINT: High-quality protein interactomes and their applications in understanding human disease.** *BMC Syst. Biol.* 2012; **6**: 92.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Dean J, Ghemawat S: **MapReduce: simplified data processing on large clusters.** *Commun. ACM.* 2008; **51**(1): 107.
- García-Alonso L, Iorio F, Matchan A, *et al.*: **Transcription factor activities enhance markers of drug sensitivity in cancer.** *Cancer Res.* 2018; **78**(3): 769–780.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Greene CS, Tan J, Ung M, *et al.*: **Big data bioinformatics.** *J. Cell. Physiol.* 2014; **229**(12): 1896–1900.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Huttlin EL, Bruckner RJ, Navarrete-Perea J, *et al.*: **Dual Proteome-scale Networks Reveal Cell-specific Remodeling of the Human Interactome.** *BioRxiv.* 2020.
- Kasson PM: **Computational biology in the cloud: methods and new insights from computing at scale.** *Pac. Symp. Biocomput. WORLD SCIENTIFIC;* 2013, pp. 451–53.
[PubMed Abstract](#) | [Publisher Full Text](#)
- Khine PP, Wang ZS: **Data lake: a new ideology in big data era.** *ITM Web of Conferences.* 2018; **17**: 03025.
[Publisher Full Text](#)
- Li T, Wernersson R, Hansen RB, *et al.*: **A scored human protein-protein interaction network to catalyze genomic interpretation.** *Nat. Methods.* 2017; **14**(1): 61–64.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Luck K, Kim D-K, Lambourne L, *et al.*: **A reference map of the human binary protein interactome.** *Nature.* 2020; **580**(7803): 402–8.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Marx V: **The Big Challenges of Big Data.** *Nat. Methods.* 2013.
[Publisher Full Text](#)
- Matthias K, Kane SP: **Docker: Up & Running: Shipping Reliable Containers in Production.** 1st ed. Sebastopol, CA: O'Reilly Media; 2015.
- Mungall CJ, Torniai C, Gkoutos GV, *et al.*: **Uberon, an integrative multi-species anatomy ontology.** *Genome Biol.* 2012; **13**(1): R5.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Orchard S, Ammari M, Aranda B, *et al.*: **The MIntAct project - IntAct as a common curation platform for 11 molecular interaction databases.** *Nucleic Acids Res.* 2014; **42**(Database issue): D358–63.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Razick S, Magklaras G, Donaldson IM: **iRefIndex: a consolidated protein interaction database with provenance.** *BMC Bioinformatics.* 2008; **9**: 405.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Silva YN, Almeida I, Queiroz M: **SQL: from traditional databases to big data.** *Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE '16.* New York, New York, USA: ACM Press; 2016; pp. 413–18.
- Smigielski EM, Sirotkin K, Ward M, *et al.*: **dbSNP: a database of single nucleotide polymorphisms.** *Nucleic Acids Res.* 2000; **28**(1): 352–55.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Smith R: **Docker Orchestration.** Packt Publishing; 2017.
- Szklarczyk D, Gable AL, Lyon D, *et al.*: **STRING v11: protein-protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets.** *Nucleic Acids Res.* 2019; **47**(D1): D607–13.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- Türei D, Valdeolivas A, Gul L, *et al.*: **Integrated intra- and intercellular signaling knowledge for multicellular omics analysis.** *Mol. Syst. Biol.* 2021; **17**(3): e9923.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)
- UniProt Consortium: **UniProt: the universal protein knowledgebase in 2021.** *Nucleic Acids Res.* 2021; **49**(D1): D480–89.
[PubMed Abstract](#) | [Publisher Full Text](#) | [Free Full Text](#)

Open Peer Review

Current Peer Review Status:  

Version 2

Reviewer Report 06 December 2022

<https://doi.org/10.5256/f1000research.136593.r147162>

© 2022 Doncheva N. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.



Nadezhda Doncheva 

¹ Novo Nordisk Foundation Center for Protein Research, University of Copenhagen, Copenhagen, Denmark

² Novo Nordisk Foundation Center for Protein Research, University of Copenhagen, Copenhagen, Denmark

In the revised version of the article, the authors have very thoroughly addressed the comments of both reviewers. The flow of the article has benefited from restructuring the Implementation section, the added instructions related to setting up and using the Sherlock platform will be very helpful for its future users, and the Discussion section nicely puts Sherlock in the context of other tools used in the field. With this, the Sherlock platform can become an important addition to the set of tools used by computational biologists, especially those who work a lot with public datasets and datasets from collaborators.

I have a minor remark related to use case 1, which has been changed in the revised version of the article. In the current version, Figure 8 only contains two tables and the last paragraph still refers to the protein identifiers expressed in the brain although this part is not covered by the example and query anymore. I personally found the previous version of use case 1 and the corresponding Figure 5 more interesting and relevant. Thus, I would recommend using the example with the three tables (the last one with the tissue information) if possible and, if that is considered too complex, then it would make sense to fix the last paragraph in this section to reflect the new query and figure.

There are also a few "typos" that could be easily fixed:

- Page 11, first column, third paragraph: "*which version 2021_03*" should probably be "with", but also can just be "version 2021_03".
- Page 12, first column, second paragraph: the sentence that starts with "*With Sherlock, the user can quickly...*" has "quickly" repeated twice.
- Page 12, second column, second paragraph: there is a "*ref3*" in the middle of the white

space that probably does not belong there.

- Page 13, second column, first paragraph: the sentence "*which enables to answer simple...*" seems to be missing a word in the middle, I assume it should be "*which enables users/computational biologists/...*".

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Computational and network biology, development of software tools, analysis and integration of large datasets

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 21 Dec 2022

Tamás Korcsmáros

We thank the reviewers for their feedback on our manuscript and the constructive comments. The comments and questions led to an appropriate revision of the manuscript.

We thank the Reviewer for the general comments and the approval of the second version of the manuscript. We also really appreciate the suggestions about Use Case 1 and minor typos. We changed the Use Case 1 accordingly and also fixed the typos in the manuscript.

Competing Interests: No competing interests were disclosed.

Reviewer Report 06 September 2022

<https://doi.org/10.5256/f1000research.136593.r147161>

© 2022 Morris J. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.



John H. Morris 

¹ Resource for Biocomputing, Visualization and Informatics, Department of Pharmaceutical Chemistry, University of California, San Francisco, San Francisco, CA, USA

² Resource for Biocomputing, Visualization and Informatics, Department of Pharmaceutical Chemistry, University of California, San Francisco, San Francisco, CA, USA

I appreciate the authors' modifications in response to the reviewers' input and believe the resulting article to now be appropriately targeted to computational biologists with knowledge of SQL and basic database theory. I believe the tool will be useful as an engine for integrating local data sets into public data for further computational analysis.

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: computational biology

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Author Response 21 Dec 2022

Tamás Korcsmáros

We thank the reviewers for their feedback on our manuscript and the constructive comments. The comments and questions led to an appropriate revision of the manuscript.

We thank the Reviewer for approving the second version of the paper. We also thank you very much for all of the constructive comments that helped us to improve the manuscript.

Competing Interests: No competing interests were disclosed.

Version 1

Reviewer Report 29 June 2021

<https://doi.org/10.5256/f1000research.56112.r85970>

© 2021 Doncheva N. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.



Nadezhda Doncheva

¹ Novo Nordisk Foundation Center for Protein Research, University of Copenhagen, Copenhagen, Denmark

² Novo Nordisk Foundation Center for Protein Research, University of Copenhagen, Copenhagen, Denmark

³ Novo Nordisk Foundation Center for Protein Research, University of Copenhagen, Copenhagen, Denmark

In this article, the authors present an open-source platform called Sherlock, which can be very useful for computational biologists working with a lot of different datasets within one or several projects. The platform is designed to make use of well-known resources and concepts such as Docker, and PrestoDB. Sherlock is easily extendable and can be used both on a local computer or in connection with a cloud storage solution. Once Sherlock is properly set up and the needed data sets imported and formatted, the data can be integrated and analyzed using SQL queries as shown in the three simple use cases.

Sherlock can become a great addition to the suite of tools used by computational biologists, especially when it comes to bigger projects involving a lot of public datasets as well as own data generated by collaborators. The article is nicely written and gives a good overview of the Sherlock platform as well as possible use cases. However, it can be further improved in three main aspects as detailed below: making clear who is the target audience, improving the documentation about setting up Sherlock, and improving the flow of the Implementation and Operation section.

Both the title and abstract mention that Sherlock is designed for biologists and only on page 6, the text mentions computational biologists specifically. However, both from the description of the setup and the specific use cases, it seems to me that the actual target audience should be clearly stated as “computational biologists”. It is worth considering changing the “for biology” part in the title since it is not very clear or specific as it is now.

One major concern is that neither the manuscript nor the online documentation seem to describe in enough detail how to set up the data needed for the three presented use cases. There is a nice deployment guide and the use case examples seem reasonable, but the step in-between is missing. The use cases assume that the user has an already configured Sherlock system with the necessary files stored, properly formatted and the right tables created, etc. I specifically checked the GitHub documentation, among others the “Loading ... data to Sherlock” sections and I don’t think enough detail is given for a user like me to set it up (and I consider myself a computational biologist). I would need to know which raw data files from STRING were retrieved and stored in the raw zone, how were the JSON files generated that need to be copied to the landing zone, etc. I was able to find the scripts, which should help for this step (<https://github.com/earlham-sherlock/earlham-sherlock.github.io/tree/master/loaders>) but they are also not described in enough detail, especially in connection with the use cases. So, what I think is needed is the following:

1. add a short description to each use case in the manuscript about which databases are needed to execute the example queries;
2. on GitHub, add a step-by-step tutorial for how to set up the specific databases/tables needed for the use cases. For use case 1, the tutorial should include information about how and from what data to create the tables “master.mapping”, “master.tissues”, “master.string_proteins”, while for use case 2 and 3, the needed tables are “master.bgee_2020_11_16” and “master.omnipath_2020_10_04”.

Another major comment is that the structure of the whole section “Implementation and Operation” can benefit from a bit of reorganization and section renaming. In the following, I will give some specific details and examples of how this can be done.

- The Overview section begins with a lengthy description of Docker and Docker Swarm and only gives an overview of the different components of Sherlock at the end of the section. The section will read much better if it begins with a high-level overview of the different components as it is partly done in the third paragraph (last one on page 3). This part can also refer to Figure 1. Then, each of the components can be explained in more or less detail (depending on the order and what follows in the next sections). For example, the end of paragraph one, which describes the scalability of Docker Swarm, also fits well with the last paragraph of the Overview section. This section could also be renamed to “Overview of the Sherlock platform”.

- In the Overview section, there seems to be a bit of confusion about the architecture & setup versus the usage workflow. The second paragraph starts with a sentence stating that “the first step of using Sherlock is to start a Docker Swarm”, while the third paragraph states that “the starting step for using Sherlock is to submit an SQL query to the Presto query engine”. I can see how both of these statements are true, but it will be easier to read if it is explained that one thing is part of the general, one-time setup of the system (which also fits more with the whole architecture), while the other one is the actual user workflow in a day-to-day usage (more relevant to the “Functionality of Sherlock” section).
- As a follow-up on that, it would be possible to move the whole description of Docker together with the “Query engine” and “Hive metastore” subsections into a section entitled “Architecture” or optionally “Architecture and setup” (instead of “Sherlock as a platform”).
- The last section about the functionality of Sherlock is currently very short and actually describes this day-to-day workflow. It could easily become part of the first overview section or else be extended by a few more details that do not fit so well into the Overview/Architecture sections.
- So, for the whole big section, the order would be “Overview of the Sherlock platform”, “Architecture (and setup)”, “Deployment”, “The Data lake – data storage” and optionally “Functionality of Sherlock”.

In the following list, I have pointed out several minor things about the text or figures that can also be improved.

- The abstract says that “Sherlock is designed to analyse, process, query and extract the information from extremely complex and large datasets.” I think to be precise, it should say that it is designed “to facilitate/enable/allow users in analysing, processing, ... ”
- The description of Docker in the first paragraph of the Overview section could be shortened a bit (or moved to a more fitting place a bit later in the description).
- The legend of Figure 1 can be improved by describing the elements from left to right instead of right to left. It is not clear what a “Minio S3 server” is since it is not mentioned anywhere else in the manuscript. The first column of boxes in the Figure could also be surrounded by a bigger box and referred to as the user input (configuration) or something similar. Then it would be easier to refer to it.
- Page 4, second paragraph: “A Query Engines... are distributed” should be either singular or plural but not both.
- Page 4, third paragraph: the phrase “designed for working data” should probably be “designed for working with data” and “which held in” should be “which is held in”.
- Page 4, third paragraph: the sentence “Moreover it is particularly really useful to handle with structured data” can be rephrased for more clarity.
- Page 4, last paragraph: the sentence “With regards to Sherlock, Presto stores the metadata about the folders in the Data Lake and Hive metastore, which contains only the meta

information about the different folders, which is in the Data Lake” also needs to be rephrased. There are too many “which” and it is not clear what refers to what and what information is actually contained/known to Presto and Hive metastore.

- Figure 2 does not look very nice in the PDF as opposed to the online version on GitHub, which has nicer formatting and quality. I would recommend replacing the figure in the PDF with the online version and making sure it still looks fine.
- Page 5, last paragraph: “on a local machine, on the cloud and the data...” should probably be “on a local machine and on the cloud and that the data...”.
- Page 5, last paragraph: the semicolon in the sentence “the most common Data Lake solutions;” could be replaced by a colon and “and with the Simple...” by “and Simple ...”.
- Page 6, first paragraph of “Functionality of Sherlock”: how should the last sentence be understood? Does the user execute a “in-memory distributed query” or is this done also by the Presto engine?
- Page 6, last paragraph: the comma in “contain specific, interaction, ...” is unnecessary.
- Page 7, there is an issue with the Omnipath reference in the table, it looks different and leads to a log-in page in the online version of the article.
- Page 7, first sentence “...we will outline three different use cases on how Sherlock can be used and what we can use it for” could be rephrased to sound better. Maybe just “... we will outline three different use cases of Sherlock”.
- Page 9, second paragraph: should this sentence “to enrich a network with interaction data” rather say “to retrieve interaction data for a gene list”?
- Figure 6: The two actions “Enrich with interconnections” and “Enrich with first neighbors” are repeated. I think it is enough to state this once below the arrow since it nicely describes the actual process. The picture before and after the arrow could have some other labels, but it is also fine if they do not have their own label.
- Page 10, second paragraph: the sentence “it contains specific database loader scripts which they are able to create and upload specific file formats to the Data Lake” needs some revising to make clear who is “they”. Maybe it should say “which the users are able to use to upload or can create themselves...”.
- Page 10, third paragraph: “followings” should be “following”.
- Page 10, last paragraph: there is probably a stop (sentence end) missing between “structure” and “providing” in “a ‘folder-like’ structure providing the added...”.
- I am not able to see the images in the html version of the article in Firefox on Mac, I can only see a name (f1db3fb0-55ef-43b8-9ea8-7f6486a79c9a_figure1.gif). If I right-click and try to see it in another tab, I see some XML code that says Access denied. I checked and I don’t

have the same issue with other articles.

Is the rationale for developing the new software tool clearly explained?

Partly

Is the description of the software tool technically sound?

Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

Partly

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?

Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?

Yes

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: Computational and network biology, development of software tools, analysis and integration of large datasets

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Author Response 21 Jul 2022

Tamás Korcsmáros

Response to Reviewers for Manuscript ID: f1000research.52791.1

We thank the reviewers for their feedback on our manuscript and the constructive comments. The comments and questions led to a significant and appropriate revision of the manuscript.

Answers for reviewer Nadezhda T. Doncheva

In this article, the authors present an open-source platform called Sherlock, which can be very useful for computational biologists working with a lot of different datasets within one or several projects. The platform is designed to make use of well-known resources and concepts such as Docker, and PrestoDB. Sherlock is easily extendable and can be used both on a local computer or in connection with a cloud storage solution. Once Sherlock is properly set up and the needed data sets imported and formatted, the data can be integrated and analyzed using SQL queries as

shown in the three simple use cases.

Sherlock can become a great addition to the suite of tools used by computational biologists, especially when it comes to bigger projects involving a lot of public datasets as well as own data generated by collaborators. The article is nicely written and gives a good overview of the Sherlock platform as well as possible use cases. However, it can be further improved in three main aspects as detailed below: making clear who is the target audience, improving the documentation about setting up Sherlock, and improving the flow of the Implementation and Operation section.

We thank the Reviewer for these general comments and the summary about the manuscript. We also really appreciate the suggestions about the three main aspects.

Both the title and abstract mention that Sherlock is designed for biologists and only on page 6, the text mentions computational biologists specifically. However, both from the description of the setup and the specific use cases, it seems to me that the actual target audience should be clearly stated as "computational biologists". It is worth considering changing the "for biology" part in the title since it is not very clear or specific as it is now.

Thank you very much for raising it, it is a good point. We agree with the Reviewer's view, and modified the manuscript throughout to reflect that Sherlock is primarily for computational biologists.

One major concern is that neither the manuscript nor the online documentation seem to describe in enough detail how to set up the data needed for the three presented use cases. There is a nice deployment guide and the use case examples seem reasonable, but the step in-between is missing. The use cases assume that the user has an already configured Sherlock system with the necessary files stored, properly formatted and the right tables created, etc. I specifically checked the GitHub documentation, among others the "Loading ... data to Sherlock" sections and I don't think enough detail is given for a user like me to set it up (and I consider myself a computational biologist). I would need to know which raw data files from STRING were retrieved and stored in the raw zone, how were the JSON files generated that need to be copied to the landing zone, etc. I was able to find the scripts, which should help for this step (<https://github.com/earlham-sherlock/earlham-sherlock.github.io/tree/master/loaders>) but they are also not described in enough detail, especially in connection with the use cases. So, what I think is needed is the following:

add a short description to each use case in the manuscript about which databases are needed to execute the example queries.

On GitHub, add a step-by-step tutorial for how to set up the specific databases/tables needed for the use cases. For use case 1, the tutorial should include information about how and from what data to create the tables "master.mapping", "master.tissues", "master.string_proteins", while for use case 2 and 3, the needed tables are "master.bgee_2020_11_16" and "master.omnipath_2020_10_04".

Thank you very much for your advice, these are really good and useful points. We have added the suggested clarifications for the github repository and for the Use Cases section in the revised manuscript as well.

Another major comment is that the structure of the whole section "Implementation and

Operation” can benefit from a bit of reorganization and section renaming. In the following, I will give some specific details and examples of how this can be done.

The Overview section begins with a lengthy description of Docker and Docker Swarm and only gives an overview of the different components of Sherlock at the end of the section. The section will read much better if it begins with a high-level overview of the different components as it is partly done in the third paragraph (last one on page 3). This part can also refer to Figure 1. Then, each of the components can be explained in more or less detail (depending on the order and what follows in the next sections). For example, the end of paragraph one, which describes the scalability of Docker Swarm, also fits well with the last paragraph of the Overview section. This section could also be renamed to “Overview of the Sherlock platform”.

Thank you for bringing this to our attention, we have now improved this section and made the suggested restructuring changes.

In the Overview section, there seems to be a bit of confusion about the architecture & setup versus the usage workflow. The second paragraph starts with a sentence stating that “the first step of using Sherlock is to start a Docker Swarm”, while the third paragraph states that “the starting step for using Sherlock is to submit an SQL query to the Presto query engine”. I can see how both of these statements are true, but it will be easier to read if it is explained that one thing is part of the general, one-time setup of the system (which also fits more with the whole architecture), while the other one is the actual user workflow in a day-to-day usage (more relevant to the “Functionality of Sherlock” section).

Thank you for raising it as well, we have fixed these inaccuracies. In the Use Cases section, we added a general set up guide of the platform and the Data Lake. After that, we introduce how to run basic analytical SQL queries in different use cases.

As a follow-up on that, it would be possible to move the whole description of Docker together with the “Query engine” and “Hive metastore” subsections into a section entitled “Architecture” or optionally “Architecture and setup” (instead of “Sherlock as a platform”).

Thank you for your suggestion. We changed the section names and structure accordingly.

The last section about the functionality of Sherlock is currently very short and actually describes this day-to-day workflow. It could easily become part of the first overview section or else be extended by a few more details that do not fit so well into the Overview/Architecture sections.

Thank you, this is a good idea. We created a new subsection for the whole docker part and also extended it with more details.

So, for the whole big section, the order would be “Overview of the Sherlock platform”, “Architecture (and setup)”, “Deployment”, “The Data lake – data storage” and optionally “Functionality of Sherlock”.

Thank you for your advice. We have restructured the whole Implementation and Operation section based on your suggestions, and also we fixed the minor points that

you raised.

Thank you very much for all of the constructive comments that helped us to improve the manuscript.

Competing Interests: No competing interests

Reviewer Report 17 June 2021

<https://doi.org/10.5256/f1000research.56112.r85971>

© 2021 Morris J. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.



John H. Morris

¹ Resource for Biocomputing, Visualization and Informatics, Department of Pharmaceutical Chemistry, University of California, San Francisco, San Francisco, CA, USA

² Resource for Biocomputing, Visualization and Informatics, Department of Pharmaceutical Chemistry, University of California, San Francisco, San Francisco, CA, USA

³ Resource for Biocomputing, Visualization and Informatics, Department of Pharmaceutical Chemistry, University of California, San Francisco, San Francisco, CA, USA

In this article, the authors describe a system, Sherlock, for storing and accessing data using SQL. The system described is flexible, and uses modern approaches to the standard ETL (extract, transform, load) processes common in a data warehouse solution. The system seems like it would be extremely useful for a narrow set of use cases, somewhat useful for a larger set of use cases, and of questionable utility for most computational biology use cases.

Sherlock's main utility, it seems to me, is for facilities that have large repositories of self-generated data. In that circumstance, the overhead of learning the column names, appropriate foreign keys, and the appropriate SQL syntax to manipulate the data would be well-warranted, and I would imagine in those circumstances, the data could be stored locally on some fast media that would provide significant performance advantages, particularly given the distributed nature of presto. I can also see Sherlock being useful in circumstances where there is a relatively static pipeline that operates on data sources that are more commonly updated. In this circumstance, the ability to have scripts to stage the data and use a common SQL syntax that doesn't change would be useful. In other circumstances, I'm less convinced. First, while I appreciate the value of the multiple-level folder structure (Raw Zone --> Landing Zone --> Master Zone --> Project Zone) that results in a number of copies of databases that could be extremely large. The examples given are relatively small, but consider downloading TrEMBL, which is 132 Gb compressed, and reprocessing that to get at least 3 copies (assuming one would subset it for the Project Zone). The scary part is that TrEMBL is small by modern standards -- image datasets are significantly larger. Second, while I agree that for software developers, SQL is a well-known and convenient query language, it is certainly non-trivial to construct efficient queries and not the best interactive tool for exploring

data sets. Further, there is no attempt to fit this tool into the more broad set of tools commonly used by computational biologists. For example, why would I use Sherlock for scRNA-seq analysis if I have access to Galaxy or AnVIL?

So, not to be totally negative, I think Sherlock has a place in the tool suite for computational biology, but I don't think that place is well articulated in the article. My recommendations for improving the article are:

1. The second sentence in the introduction talks about working on bespoke datasets, which I suspect is the crux of Sherlock. The rest of the article seems to ignore that, and even the use cases focus purely on public data. This weakens the case for Sherlock. The use cases are trivially (OK, so relatively) easy using a combination of python, perhaps with pandas, data files and web services. The power of Sherlock is the ability to *integrate* public databases with bespoke databases, but that really isn't highlighted sufficiently. That needs to be the focus on the user cases, I think.
2. Discuss Sherlock in the context of other tools (not technologies) that support computational biology, such as Galaxy and AnVIL as two examples. When would I use those tools? When would I use Sherlock? Why is Sherlock better in that case?
3. Be clearer as to the target user. In several cases, the article states that this is "designed specifically for biologists". I actually don't think that's correct. It's designed for computational biologists or bioinformaticians with a very strong computational background. I don't know many bench biologists who would be able to formulate some of the SQL queries, even if it was easy to figure out all of the column names. I don't *think* the authors believe that SQL is the user interface for the system as much as an API for accessing the data, but that also wasn't clear in the text.
4. The role of Hive is somewhat obscure to me. I got that it stores the metadata, but *what* metadata? Is that where I can look up the schema to get the column names for joins and projections? In the use cases, it seems like there were several jumps in logic. Where does the user figure out that the ensemble ID in the STRING database is stored as "ens_id"? How well exposed is the schema for each of the tables in the data lake and how does a user (who didn't happen to be the one who imported the data) figure that out?
5. One of the main points in the article is the potential for high performance due to the parallelism of presto, but there were no performance numbers offered, either for the ETL phase or for the queries themselves.
6. Finally, as a more minor point, there are several places where the manuscript could use a bit of an edit pass. For example, on page 2 it states "A Query Engines, for example " shouldn't be plural.

Sherlock clearly represents a *lot* of work, and I do believe that it could be valuable to the appropriate audience and that it should be published. My take on the manuscript is that it's not clear who it is written for. If it's written for a computer scientist, then information about performance metrics and local vs. cloud storage trade-offs and scalability, etc., are missing. If it's written for the computational biologist, then it's not really necessary to address the trade-offs between docker swarm and kubernetes, or why SQL can be considered composed of three main

sub-languages, but it is important to explain why Sherlock is better than Galaxy or AnVIL and a little more detail on how the user figures out things like column names, etc.

Sorry if this comes across as overly negative. I honestly do see how much work is involved and I can imagine where it might be extremely useful in some environments, but that just doesn't come across clearly in this submission.

Is the rationale for developing the new software tool clearly explained?

Partly

Is the description of the software tool technically sound?

Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?

Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?

Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?

Partly

Competing Interests: No competing interests were disclosed.

Reviewer Expertise: computational biology

I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Author Response 21 Jul 2022

Tamás Korcsmáros

Response to Reviewers for Manuscript ID: f1000research.52791.1

We thank the reviewers for their feedback on our manuscript and the constructive comments. The comments and questions led to a significant and appropriate revision of the manuscript.

Answers for reviewer John H. Morris

In this article, the authors describe a system, Sherlock, for storing and accessing data using SQL. The system described is flexible, and uses modern approaches to the standard ETL (extract,

transform, load) processes common in a data warehouse solution. The system seems like it would be extremely useful for a narrow set of use cases, somewhat useful for a larger set of use cases, and of questionable utility for most computational biology use cases.

Sherlock's main utility, it seems to me, is for facilities that have large repositories of self-generated data. In that circumstance, the overhead of learning the column names, appropriate foreign keys, and the appropriate SQL syntax to manipulate the data would be well-warranted, and I would imagine in those circumstances, the data could be stored locally on some fast media that would provide significant performance advantages, particularly given the distributed nature of presto. I can also see Sherlock being useful in circumstances where there is a relatively static pipeline that operates on data sources that are more commonly updated. In this circumstance, the ability to have scripts to stage the data and use a common SQL syntax that doesn't change would be useful.

We thank the Reviewer for these general comments and we agree with the specific use cases described by the Reviewer on where Sherlock could be useful.

In other circumstances, I'm less convinced. First, while I appreciate the value of the multiple-level folder structure (Raw Zone --> Landing Zone --> Master Zone --> Project Zone) that results in a number of copies of databases that could be extremely large. The examples given are relatively small, but consider downloading TrEMBL, which is 132 Gb compressed, and reprocessing that to get at least 3 copies (assuming one would subset it for the Project Zone). The scary part is that TrEMBL is small by modern standards -- image datasets are significantly larger. Second, while I agree that for software developers, SQL is a well-known and convenient query language, it is certainly non-trivial to construct efficient queries and not the best interactive tool for exploring data sets. Further, there is no attempt to fit this tool into the more broad set of tools commonly used by computational biologists. For example, why would I use Sherlock for scRNA-seq analysis if I have access to Galaxy or AnVIL?

We thank the Reviewer for this comment, which is really helpful as these were not properly covered in the original submission. Regarding the multi-level folder structure, we added a paragraph into the end of "The Data Lake - data storage" section to explain its usefulness further. Regarding the last part of this comment, we added paragraphs to the "Discussion" section of the revised manuscript.

So, not to be totally negative, I think Sherlock has a place in the tool suite for computational biology, but I don't think that place is well articulated in the article. My recommendations for improving the article are:

The second sentence in the introduction talks about working on bespoke datasets, which I suspect is the crux of Sherlock. The rest of the article seems to ignore that, and even the use cases focus purely on public data. This weakens the case for Sherlock. The use cases are trivially (OK, so relatively) easy using a combination of python, perhaps with pandas, data files and web services. The power of Sherlock is the ability to integrate public databases with bespoke databases, but that really isn't highlighted sufficiently. That needs to be the focus on the user cases, I think.

We would like to thank you for this comment and also thanks for the suggestions, we modified the manuscript accordingly. We added that part into the "Overview of the platform" section.

Discuss Sherlock in the context of other tools (not technologies) that support computational biology, such as Galaxy and AnViL as two examples. When would I use those tools? When would I use Sherlock? Why is Sherlock better in that case?

Thank you for these important questions not yet addressed in the manuscript. We added new paragraphs about AnViL and Galaxy into the "Discussion" section to address these.

*Be clearer as to the target user. In several cases, the article states that this is "designed specifically for biologists". I actually don't think that's correct. It's designed for computational biologists or bioinformaticians with a very strong computational background. I don't know many bench biologists who would be able to formulate some of the SQL queries, even if it was easy to figure out all of the column names. I don't *think* the authors believe that SQL is the user interface for the system as much as an API for accessing the data, but that also wasn't clear in the text.*

Thank you for this comment as well. This is a valid point as indeed Sherlock was designed for computational biologists or data scientists. Therefore, we clarified it in the revised manuscript and updated the title as well.

*The role of Hive is somewhat obscure to me. I got that it stores the metadata, but *what* metadata? Is that where I can look up the schema to get the column names for joins and projections? In the use cases, it seems like there were several jumps in logic. Where does the user figure out that the ensemble ID in the STRING database is stored as "ens_id"? How well exposed is the schema for each of the tables in the data lake and how does a user (who didn't happen to be the one who imported the data) figure that out?*

Thank you for raising this. Hive is actually more of a technical part, which is used by PrestoDB. Sherlock stores metadata (columns structure) about the tables for Presto to query the files. For best practice, each user/user group needs to add a schema to describe what is inside the tables (wiki page, loader scripts, SQL editors). We have extended the manuscript to clarify these points in the "Functionality of Sherlock" section.

One of the main points in the article is the potential for high performance due to the parallelism of presto, but there were no performance numbers offered, either for the ETL phase or for the queries themselves.

Thank you for pointing out that this comparison was missing. We added test measurement to the manuscript into the "The Data Lake - data storage" section.

Finally, as a more minor point, there are several places where the manuscript could use a bit of an edit pass. For example, on page 2 it states "A Query Engines, for example " shouldn't be plural.

Thank you for raising this, we reviewed and corrected the text now.

*Sherlock clearly represents a *lot* of work, and I do believe that it could be valuable to the appropriate audience and that it should be published. My take on the manuscript is that it's not clear who it is written for. If it's written for a computer scientist, then information about performance metrics and local vs. cloud storage trade-offs and scalability, etc., are missing. If it's written for the computational biologist, then it's not really necessary to address the trade-offs between docker swarm and kubernetes, or why SQL can be considered composed of three main sub-languages, but it is important to explain why Sherlock is better than Galaxy or AnVIL and a little more detail on how the user figures out things like column names, etc.*

Thank you very much for this summary and the suggestions. We agree with the Reviewer's view. Accordingly, we modified the manuscript throughout to meet with the interest and needs of a computational biologist. In some cases, we kept some background information (such as trade-offs between docker swarm and kubernetes) but rephrased them in a restructured arrangement.

Sorry if this comes across as overly negative. I honestly do see how much work is involved and I can imagine where it might be extremely useful in some environments, but that just doesn't come across clearly in this submission.

Thank you very much for all of the constructive criticism and helping us to improve the manuscript.

Competing Interests: No competing interests

The benefits of publishing with F1000Research:

- Your article is published within days, with no editorial bias
- You can publish traditional articles, null/negative results, case reports, data notes and more
- The peer review process is transparent and collaborative
- Your article is indexed in PubMed after passing peer review
- Dedicated customer support at every stage

For pre-submission enquiries, contact research@f1000.com

F1000Research