***Data and tools integration (TDI)***

One of the challenges encountered every day on forensic research is the large and heterogeneous collection of libraries, tools, and data available to us. Our project designs and prototypes solutions for the integration of many existing components into a single system which is flexible, robust and effective for forensic data exploration.

Over four sprints the TDI team will develop technology in response to the following questions: How do we efficiently access data stored in different formats (possibly in a distributed setting)? How do we combine different tools into a concise workflow? How do we combine the results of each tool into a single result? How do we execute this workflow efficiently? How do we integrate such concise workflow into Hansken?

The TDI’s technology is described by the diagram depicted in on Figure 1. It is composed by the Sherlock layer which interacts with Hadoop 2.0 cluster and Hansken instance both running on SURFsara cloud, but also with an elastic search cluster. The components composing the Sherlock layer are organized into two categories, operations and visualization. The operations group is where all tools for data classification, clustering, topic modelling are. They process raw files and traces created by Hansken to derive new information which is then consumed by the visualization tools.

The dashed boxes identify components which are not yet totally developed. The dashed lines identify the cases where the bridge between two different components, i.e., systems, is not yet established. All the other components are up and running, fully implemented and inter-linked.

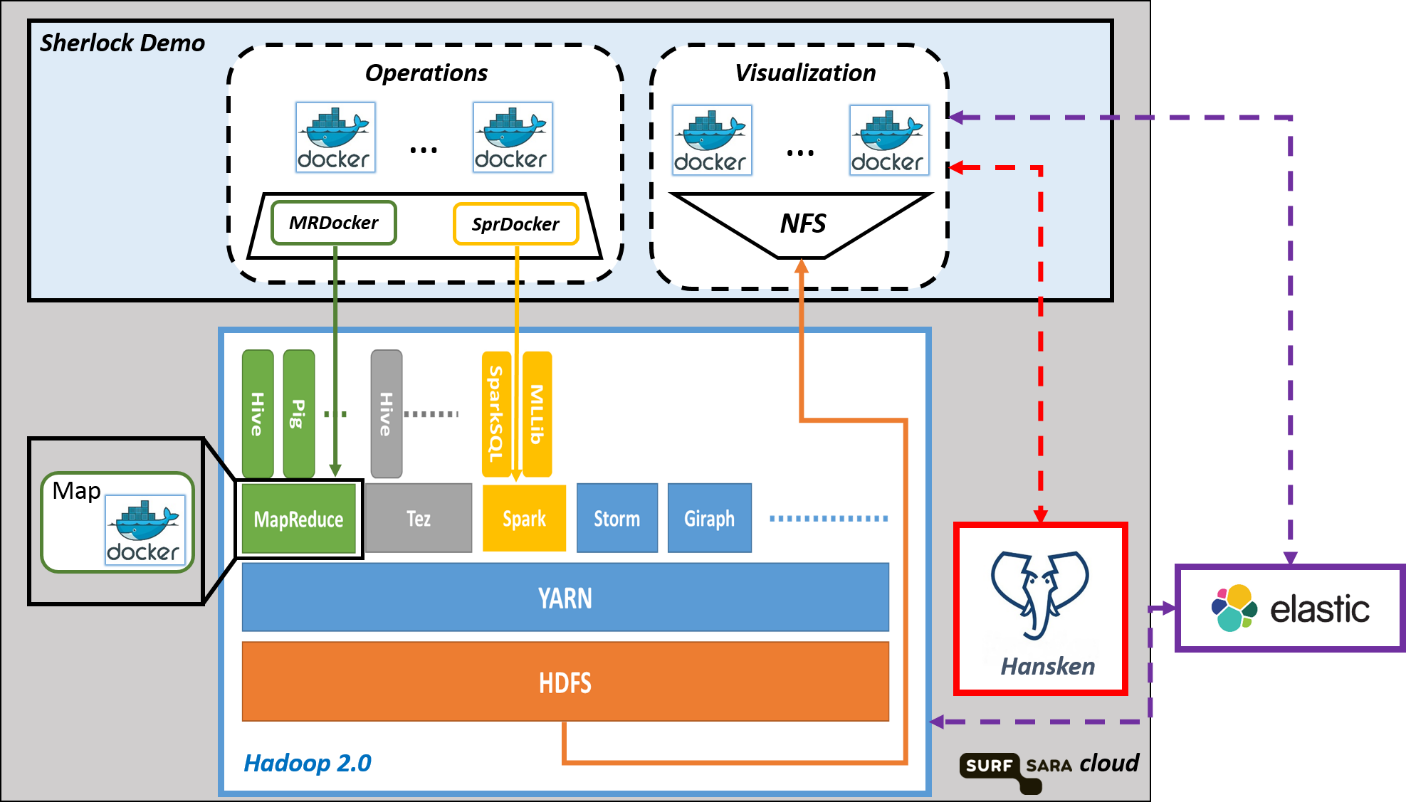


Figure 1.

The TDI team uses different state-of-the-art technologies to encapsulate and interlink each component in an efficient manner. Through Docker containers each library and all the context on which it depends is wrapped into a light virtual image, i.e., a Docker container. Through this technology the library is easily deployed on different platforms.

In the context of Hansken, HDFS is used as the data hub. On each step of the processing pipe-line each tool consumes input data stored in HDFS and the outputs the result back to HDFS to be visualized or simply re-used by other tools. Hence, to exploit data locality, have balanced resource utilization and fault-tolerance, the Hadoop 2.0 infra-structure is used to schedule and execute Docker instances over data stored in in HDFS. Furthermore, to make all the files stored in HDFS accessible as POSIX two options were considered: FUSE and NFS. With both solutions HDFS will be mounted as a directory to be provided as input to the Docker instance. After trying both options, NFS seems to be easier to setup and used by a larger user community.

As first step towards a Map-Docker function was implemented deployed using MapReduce processing paradigm. Such function runs a simple Docker instance using a Java "System call" and through Hadoop streaming API it reads from *stdin* and writes to the *stdout*. The second option is to explore the Docker Java API to implement more advanced options to load input data and output the results, but also to start and stop the Docker instance. A set of templates on how to do it can be found at [hadoop-streaming-docker](https://github.com/nlesc-sherlock/hadoop-streaming-docker).

In search for efficiency, the second step was to deploy Docker instances using Spark on the same YARN cluster. After setting up a SPARK cluster, [using the following instructions](https://github.com/nlesc-sherlock/data_tools_integration/blob/master/docs/setups/spark/On-Yarn.md), the deployment of Spark jobs was studied in the context of Latent Dirichlet Allocation (LDA) which is part of the machine learn library of Spark. LDA is used by the topic modeling team to classify emails by topic. To use LDA efficiently the input data was re-structured into a SequenceFile archives using [Forqlift](http://qethanm.cc/projects/forqlift/). With the data stores as SequenceFile archives the data preparation phase of LDA takes advantage of data locality and uses all the available resources.

To run a Docker instance a spark app called spark\_docker was developed. All the instructions on how to run Docker from a Spark app can be found at [spark-docker](https://github.com/nlesc-sherlock/spark-docker). The spark\_docker was tested in the context of image classification. The library uses neural networks to classify a set of images and it was developed by the deep learning group. Using a Docker image stored at [DockerHub](https://hub.docker.com/), spark\_docker instantiated one per node which used as input the local images and outputted into a CSV file the name of the file and the classification result. Such final result is then readable by the visualization layer using the NFS mount.

Currently TDI team is working on the integration of other team’s tools using spark\_docker into a concise workflow. For efficient resource utilization, efficient scheduling while co-existing with other applications running on the same cluster, TDI team is looking at Mesos and Kubernete. Furthermore, for flexible heterogeneous data integration and the use of external tables as input data is other direction under consideration where the data injection stands on the same principles as our internal Data Vaults project.