MOdel based coNtrol framework for Site-wide OptimizatiON of data-intensive processes

D4.5 – Final Big Data Storage and Analytics Platform

Deliverable ID	**D4.5**
Deliverable Title	**Final Big Data Storage and Analytics Platform**
Work Package	**WP4 – Cross-sectorial Data Lab**
Dissemination Level	**PUBLIC**
Version	**1.1**
Date	**2019/07/04**
Status	**Final**
Lead Editor	**CAP**
Main Contributors	

Published by the MONSOON Consortium

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 723650.

Document History

Version	Date	Author(s)	Description
0.1	2019/05/24	Jean Gaschler (CAP)	First Draft with TOC
0.2	2019/06/20	Peter Bednar (TUK)	Added architecture and description of the development deployment
0.3	2019/06/26	Peter Bednar (TUK)	Added description of components' Docker files
0.4	2019/07/03	Guillaume Charbonnier (CAP)	Add sections about deployment on premise and cloud
1.1	2019/07/11	Jean Gaschler (CAP)	Final version

Internal Review History

Version	Review Date	Reviewed by	Summary of comments
1.0	2019/07/03	Marco Dias (GLN)	Fully accepted with small typo corrections
1.0	2019/07/10	LCEN Team	Fully accepted

Table of Contents

1 Executive Summary

This document describes the distributed platform for Big Data Storage and Analytics that provides resources and functionalities for storage batch, and real-time processing of the big data. The platform combines and orchestrates existing technologies from Big Data and analytic landscape and sets a distributed and scalable run-time infrastructure for the data analytics methods developed in the project. The high-level architecture with its provided interfaces for cross-sectorial collaboration is presented. The solutions and technology options available for each logical component of the architecture are briefly explained.

The development approach in MONSOON is iterative and incremental, including three prototyping cycles (ramp-up phase, period 1, and period 2). The physical architecture of the Big Data Storage and Analytics Platform as realized from high-level architecture and the chosen technology stack for a ramp-up phase deployment is thoroughly explained. The platform and its components have been deployed in TUK environment to provide simplified storage infrastructure for the collected monitoring data from both aluminium and plastic domains. The deployment setup and configuration of the physical and virtual infrastructures are also presented.

1.1 Related documents

ID	Title	Reference	Version	Date
[RD.1]	Grant Agreement-723650-MONSOON	723650	final	24/06/2016
[RD.2]	D3.3 – Final Real Time Communication Framework	--	final	31/03/2019
[RD.3]	D3.6 – Final Virtual Process Industries Resources Adaptation	--	final	31/05/2019
[RD.4]	D3.8 – Final Runtime Container	--	final	31/04/2019
[RD.5]	D2.6 - Final Requirements and Architecture Specifications	--	final	31/03/2019
[RD.6]	D2.7 - MONSOON Initial Cross-sectorial Domain Model	--	final	10/08/2017
[RD.7]	D2.8 - MONSOON Final Cross-sectorial Domain Model	--	final	30/11/2018
[RD.8]	D4.3 - Initial Big Data Storage and Analytics Platform	--	final	31/03/2017
[RD.9]	D4.4 - Updated Big Data Storage and Analytics Platform	--	final	30/11/2018

2 Introduction

In the context of MONSOON work package structure, Task 4.2 (Big Data Storage and Analytics Platform) deals with the setup and deployment of distributed and scalable run-time infrastructure for the data analytics methods developed in project. It provides main integration interfaces for cross-sectorial collaboration between the site operational platform and cloud Data lab platform and programming interfaces for implementation of the data mining processes.

The document presents the final specifications of the Big Data Storage and Analytics platform.

The document is divided as follows. Section 3 covers the last architecture updates about the Big Data Storage and Analytics platform. It also describes the high-level architecture components, exposed interfaces and the candidate technologies which can be used to realize the platform. Section 4 presents the different platformsinfrastructures (development on TUKE premises, integration on Capgemini premises and the production platform on the Cloud).

3 Platform Architecture and Components

The platform architecture was specified in [RD.6]and further developed in [RD.8] and [RD.9] deliverables. Deliverable [RD.6]described how the architecture is divided into components and how components interact in order to implement functionalities specified for the Data Lab platform. Deliverable [RD.8] then specified which technology can be used for the implementation of the components and which components were deployed for ramp-up phase. The initial deployment was revised in [RD.9] which provides initial containerization of the platform improving the modularity and deployment. The conceptual architecture introduced in [RD.6]is depicted for reference on the following figure.

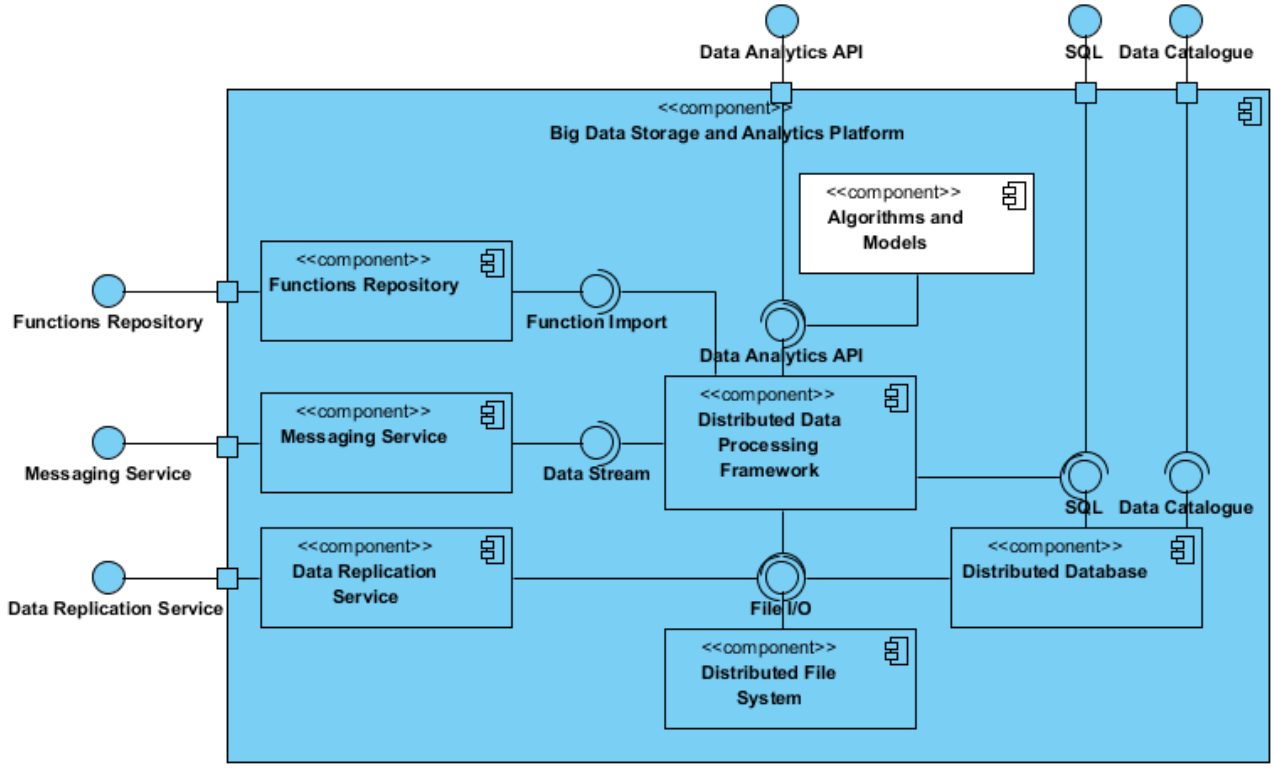

Figure 1- Big Data Storage and Analytics Platform conceptual architecture.

The final version of the reference implementation is summarized on the following schema.

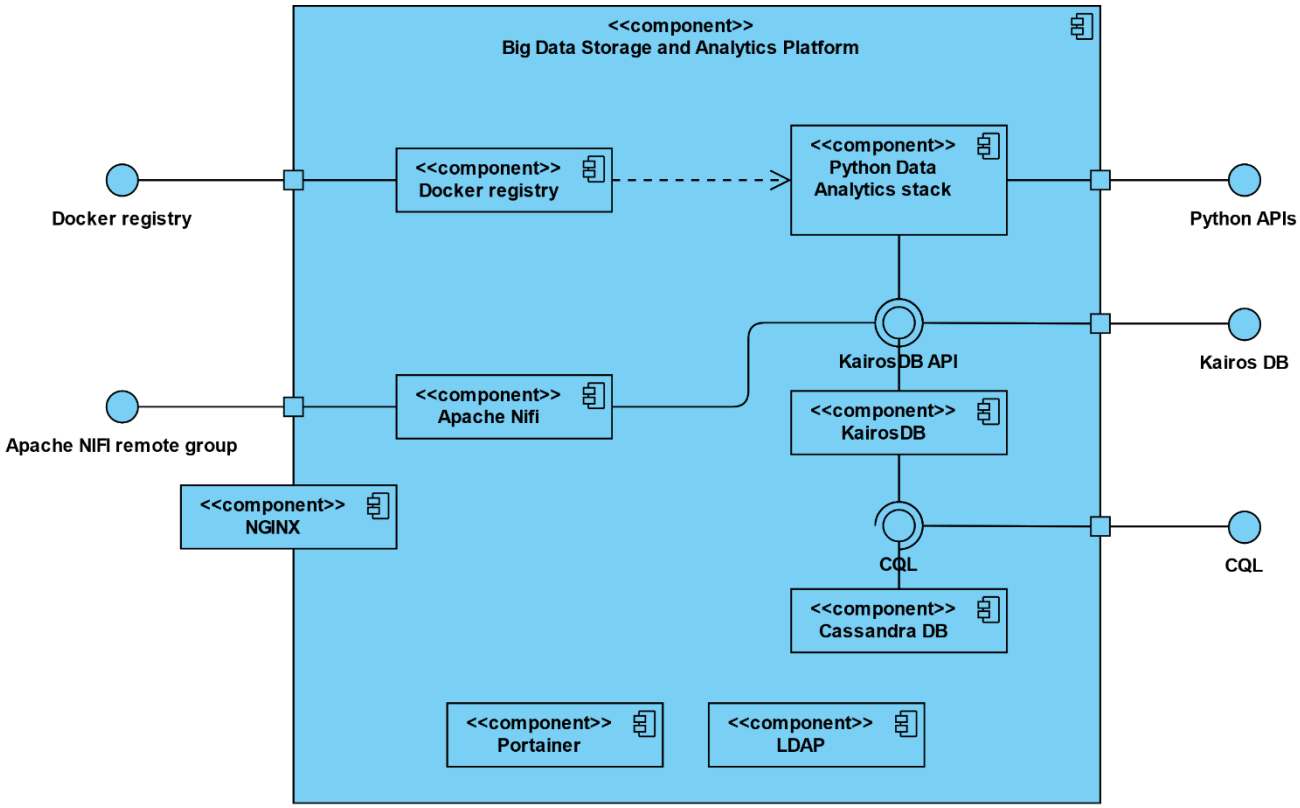

Figure 2- Big Data Storage and Analytics Platform final implementation.

The following subchapters describe main changes from the previous versions for particular component. The main goal of this final refinement was to simplify architecture by implementing multiple components using the same technology or by changing the overall dataflow in the platform.

3.1 Messaging and Data Replication service

Inthe final version of the platform, Messaging and Data Replication service is implemented using the Apache Nifi infrastructure. Data for both real-time and batch updates are sent directly between Apache Nifi instances by the native Apache Nifi communication protocol. Data Lab instance specifies input data port for each tenant (i.e. production site data provided such as plastic or aluminium for our pilot case). The communication is based on the push communication pattern, i.e. it is initialized by the site instance, which integrates Data Lab as the remote process group in the NiFi's dataflow. Data Lab instance is just listening for the incoming asynchronous updates. Data are directly harmonized to the common JSON file already in the site instance. Besides the real-time JSON updates, it is also possible to sendbatchcompressed archives of multiple JSON updates (batch updates can be sent to the same port as real-time data, or for better scalability, Data Lab interface can be configured with multiple input ports for one provider, i.e. one for real-time updates and one for batch updates of large historical data). One of the main advantages of this approach is that it is possible to track data provenance for each update across the deployment environments, i.e. it is possible to directly track whole processing history how the record was processed on the site and how it was processed in Data Lab before it was stored in the Distributed database. All data provenance logging data can be centrally processed and stored in the Data Lab instance of the Apache Nifi itself, so implementation of the data provenance doesn't require deployment of the any other technology.

3.2 Distributed data storage

In the initial version of the platform, distributed data storage was divided into the Distributed File System and Distributed Database following the data lake pattern where the heterogenous data are at first stored in the unstructured distributed file system and then incrementally harmonized, structured and transferred to the

distributed database. Since in the final version of the platform the data are harmonized already in the site environment and Data Lab receives data in unified JSON format, data are sent directly to the Distributed Database. The file system is used only to store backup archives of updates. Since the total volume of compressed archives is not so high and archive data doesn't require flexible data access, it is now possible to implement file system using more conventional non-distributed technologies such as federation of the local storage disks or Network Area Storage (NAS) shared drives. This substantially simplifies the deployment of the overall architecture, because it is not necessary to maintain distributed file system services on each data node and other components of the platform can access data directly on local file system without any client software or directly in the Distributed Database.

The implementation of the Distributed Database is based on the combination of Cassandra DB columnar distributed database and KairosDB time-series database. Cassandra DB is the wide-column distributed store, which provides persisted storage for all structured data through the SQL-like query interface.KairosDBis implemented as the frontend to the Cassandra DB and provides query interface for the time-series data, which is the most common data format in the process industry. Replication and consistency of data is enforced by Cassandra DB.

3.3 Data processing framework

The core of the Data processing framework is the Python environment which was introduced in deliverable [RD.9]. Besides the upgrade of the versions of libraries already, the environment was extended with the Kairos DB client library which provides the main interface to access data stored in the Distributed data storage. The interface allows to fetch specified metrics from the KairoDB database, filter data according to various constraints and additionally aggregate data in specified time windows.

3.4 Platform management tools

In the final version of the platform, all components are dockerized and can be deployed in the cluster of servers with the stand-alone installation of the Docker runtime environment or in the Swarm cluster. Configuration of the components is implemented in the Docker files and Docker compose files. As the main tool for configuration and monitoring of the Data Lab cluster, we have adopted Portainer application, which provides user interface for the system administrator for management of the running containers. Portainer application itself is running in the Docker container. Besides the monitoring and configuration of the software components, user management was integrated using the common LDAP server with the phpLDAPAdmin web interface for administrators and self-service web application. Self-service web application is provided for the end users to manage their own Data Lab accounts. It is secured by the security gateway and provided on the same web address as the rest of the Data Lab components which provides user interface.

3.5 Optional extensions

For the extension of the final platform to achieve better scalability or interoperability, we have proposed and tested the following optional components:

- The Messaging service can be extended with the MQTT Mosquitto broker to provide support for the MQTT protocol versions 5.0, 3.1.1 and 3.1.
- In the case of more heterogenous data and high-volume data, the Distributed storage can be reconfigured, and tools can additionally use HDFS distributed file system (as it was implemented in the previous versions of the platform).
- Data processing framework can be extended with the distributed computation frameworks such as Apache Flink and Apache Spark.

4 Platform Deployment

This chapter describes the different environments where the Data Lab is deployed:

- The development of all tools is done on the TUKE platform
- The integration of tools and preparation to the production deployment isdone on a specific Capgemini Platform
- The final production platform isin the Cloud.

4.1 Development platformon TUKE

Development deployment of the platform is running on the physical servers and virtualized environment based on Microsoft Hyper-V. The physical infrastructure consists of three servers with the following configuration:

- 24 logical CPU cores with Intel Xeon 3.5 GHz
- 96 GB operating memory
- 1120 GB local disk storage (6x500 GB drives configured in RAID5)
- 3x 1Gbps network interfaces (NIS)

Additionally, each physical server is connected to the dedicated network-attached storage server using the iSCSI protocol. Storage is configured with one RAID 6 volume, which combines 10x 3TB hard drives. Volume is further divided to three iSCSI logical units (NUMs) mapped to three targets dedicated for each server. Each target is hosted by dedicated network interface.

The physical infrastructure is mapped to the four virtual servers: one gateway server, application server, and two data servers. Each server has installed stand-alone Docker runtime environment. Gateway and application virtual servers are sharing the same physical server and both data servers are running on the dedicated physical server (data from one pilot is stored on one virtual and physical data server). The following diagram shows how the components are deployed in the development virtualized environment.

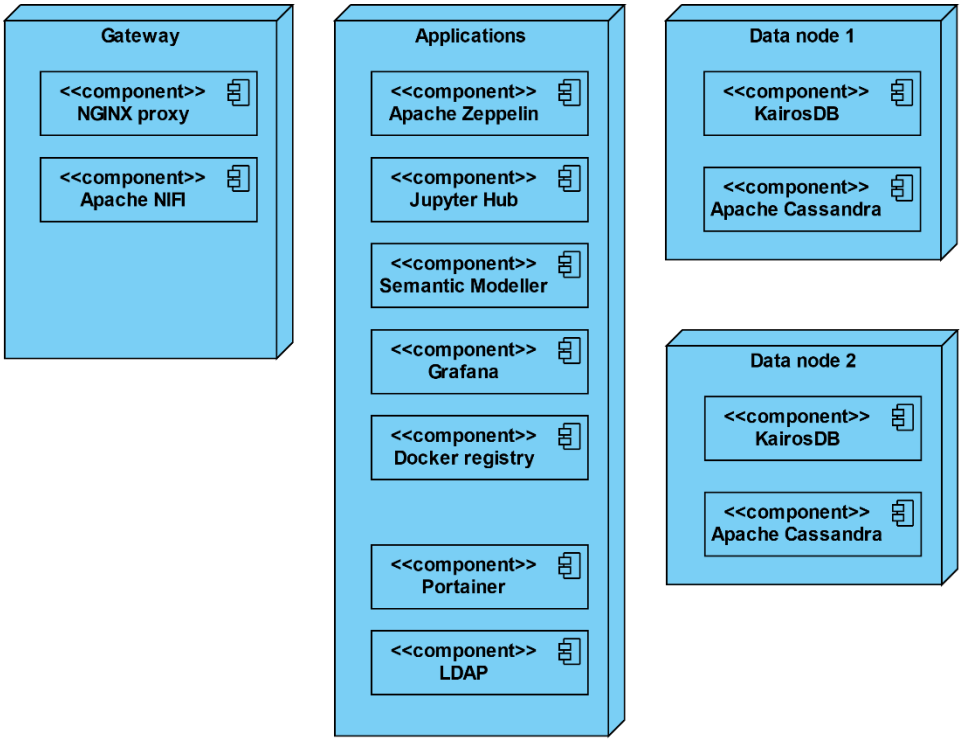

Figure 3- Deployment of the components in the development environment.

4.2 Integration platform on Capgemini

The integration platform is hosted in the Capgemini infrastructure. For security reason, only Capgemini users have access to it, so they performed all tasks of integration.

The integration platform is running on a physical server with CentOS base operating system and virtualized environment based on KVM, qemu and libvirt. The physical infrastructure consists of one server with the following configuration:

- 16 physical core / 32 virtual cores with Intel(R) Core(TM) i9-7960X CPU @ 2.80GHz
- 128GB operating memory
- 512GB SSD NVMe + 4TB HDD
- 3 x 10Gbps network interfaces (NIS)

The physical infrastructure is mapped to eight virtual servers: three Docker Swarm managers, four Docker Swarm workers as well as one NFS server. Both masters and workers have NFS folders mounted in their filesystems to achieve distributed storage. The infrastructure provisioning as well as the deployment of the platform has been automated and performedusing Ansible.

4.2.1 Ansible overview

Ansible is an IT automation engine that automates configuration management, application deployment, intra-service orchestration, and many other IT needs against multiple systems in infrastructures at the same time called inventory.

Playbooks are Ansible's configuration, deployment and orchestration language. They can describe policies you want part of your inventory to enforce, or declare state that your remote systems should be in.

Complex tasks can be performed with few lines of declarative *yaml*, using standard modules. Fully independent or interdependent collections of variable, tasks, files, templates and modules can be packaged as roles, and be imported in one or several playbooks.

4.2.2 Infrastructure provisioning

Several Ansible roles are dedicated to infrastructure provisioning:

- *cloud_images*: Download Centos 7.5 cloud images and prepare each virtual machine image individually using cloud-init. It includes network configuration, root user generation as well as ssh access.
- *libvirt_monsoon*: Define virtual networks, virtual storage pools, storage volumes and virtual machines on hypervisor host.
- *common-pkg*: Install common packages on virtual machines.
- *users*: Create administrator and system users and define ssh policies on virtual machines.
- *firewall*: Define firewall policies on virtual machines and hypervisor hosts
- *get-docker*: Install Docker Community Edition on virtual machines.
- *nfs*: Setup Network File System configuration and mounts on virtual machines.
- *init-swarm*: Create or join Docker Swarm cluster on each virtual machine.
- *haproxy*: Deploy a single instance of haproxy load balancer on hypervisor host to balance traffic to Swarm cluster services.

A master playbook is used to execute all roles sequentially. Once this playbook is running, the infrastructure is ready and platform can be deployed.

4.2.3 Platform deployment

Platform deployment is performed using a single playbook. This playbook is responsible for Docker secret generation, runtime configuration rendering from templates and application deployment. Templates are highly configurable for efficient reuse in several deployment scenarios.

This playbook leverages the *docker_stack* Ansible modules which can deploy applications directly to Docker Swarm cluster using Ansible standard library.

By default, the playbook deploys all applications sequentially, but a configuration file can be used to specify the list of applications to deploy. This is useful when updating a single component of MONSOON platform.

4.3 Production (Cloud platform)

The production platform is deployed in the Microsoft Azure cloud environment. It is running on four virtual private servers with the following configuration:

- 4 logical CPUs cores with Intel® Haswell 2.4 GHz E5-2673 v3
- 16 GB operating memory
- 32 GB optimized local SSD

Each virtual machine leverages SMB3.0 protocol to communicate with Microsoft File Storage to ensure redundancy and high availability access. Local storage is used solely by operating systems and applications running in Docker containers store their data in the shared Microsoft File Storage.
An Azure load balancer is exposed publicly and redirect traffic to the Azure instances Swarm cluster.

4.3.1 Infrastructure provisioning

Environment is provisioned using Ansible. A configuration file describes the desired inventory and a single playbook is used to create the Azure instances thanks to *azure_rm_virtualmachine*module in Ansible standard library. Once Azure instances are generated, a subset of Ansible playbooks used to deploy the Integration

platform hosted by Capgemini, can be used to deploy the platform (roles related to virtualization and load balancing are skipped).

5 Conclusion

This deliverable presented the updated version of the Big Data Storage and Analytics Platform of the MONSOON project. The platform combines and orchestrates existing technologies from Big Data and analytic landscape and sets a distributed and scalable run-time infrastructure for the data analytics methods developed in the projectThe physical architecture of the platform and the chosen technology stack have been precisely described; solutions and technology options available for each logical component have been presented.

The Cloud platform is now filled with production data until the end of the Monsoon project (September 2019) and used by the data scientists and experts of the MONSOON project.

Acronyms

Acronym	Explanation
HDD	Hard Drive Disk
MQTT	Message Queue Telemetry Transport
SMB	Server Message Block
SSD	Solid State Disk

List of Figures

List of Tables

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 723650.

Appendix A: Dockerfiles

The following tables describe Dockerfile configuration of the main components which are deployed in the Big Data Storage and Analytics Platform.

Table 1 - Dockerfile for the gateway

Component / Description	**gateway** Gateway provides common secured access point for all Data Lab components (including the Messaging and Data replication services or user interfaces of Development Tools and Semantic Modelling framework. Gateway is implemented using the Nginx reverse proxy. Nginx is an open source reverse proxy server for HTTP, HTTPS, SMTP, POP3, and IMAP protocols, as well as a load balancer, HTTP cache, and a web server (origin server). The nginx project started with a strong focus on high concurrency, high performance and low memory usage. It is licensed under the 2-clause BSD-like license
Environment variables	SERVER_NAME - fully qualified public name of the gateway server (corresponds to the public address of the MONSOON Data Lab instalation). JUPYTERHUB_URL - URL of the JupyterHub local installation mapped to /jupyter/ location. ZEPPELIN_URL - URL of the Apache Zeppelin local installation mapped to /zeppelin/ location. GRAFANA_URL - URL of the Grafana local installation mapped to /grafana/ location. MODELLER_URL - URL of the Semantic framework local installation mapped to /modeller/ location. PASSWORD_URL - URL of the Password self-service application mapped to /password/ location. REGISTRY_URL - URL of the local Docker image registry mapped to /v2/ location.
Volumes	/etc/nginx/secrets - repository with the private and public TLS keys and certificate logs are printed to the console
Networks/Exposed ports	public HTTPS 443 port connected to host network and frontend private network

Table 2 - Dockerfile for Nifi

Component / Description	**nifi** Apache Nifi provides implementation of the Messaging Service for real-time updates of data and Data replication Service for batch updates. Apache NiFi is a software designed to automate the flow of data between software systems. The NiFI design is based on the flow-based programming model and offers features which prominently include the ability to operate within clusters, security using TLS encryption, extensibility (users can write their own software to extend its abilities) and improved usability features like a portal which can be used to view and modify behaviour visually.
Environment variables	TRUSTSTORE_PATH, TRUSTSTORE_PASSWORD, TRUSTSTORE_TYPE - configuration of trust store file which contains certificates of the remote NiFi instances sending the data to the Data Lab. KEYSTORE_PATH, KEYSTORE_PASSWORD, KEYSTORE_TYPE - configuration of key store files where the public and private TLS key is stored

	NIFI_WEB_PROXY_HOST, NIFI_WEB_HTTP_HOST, NIFI_REMOTE_INPUT_HOST - configuration of proxy address which should corresponds to the gateway public server URL
Volumes	/opt/secrets - local directory with the truststore and keystore files used for the web-frontend and NiFi-to-NiFi communication. /opt/nifi/nifi-1.6.0/conf - directory where the configuration and dataflow files are stored /opt/nifi/nifi-1.6.0/logs - logs directory /opt/nifi/nifi-1.6.0/database_repository - database directory /opt/nifi/nifi-1.6.0/flowfile_repository - directory where flowfile repository is stored, flowfile repository stores data between processing steps /opt/nifi/nifi-1.6.0/content_repository - directory where content of the flowfile is stored between processing steps /opt/nifi/nifi-1.6.0/provenance_repository - directory where data provenance records are stored
Networks/Exposed ports	public 8443 port for web interface and NiFi-to-NiFi communication connected to host network and database backend networks for each tenant

Table 3 - Dockerfile for Cassandra

Component / Description	**cassandra** Apache Cassandra is the main persistent storage for the data stored in the Data Lab environment. It is responsible for reliable and scalable storage of data enforcing the security and consistency. Apache Cassandra is a free and open-source, distributed, wide column store, NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra offers robust support for clusters spanning multiple datacentres, with asynchronous masterless replication allowing low latency operations for all clients.
Environment variables	none
Volumes	/var/lib/cassandra - directory where database files are stored
Networks/Exposed ports	one instance connected to database backend network for specific tenant or shared by multiple tenants

Table 4 - Dockerfile for KairosDB

Component / Description	**kairosdb** KairosDB provides query interface for time-series data and it is the main interface for accessing data stored in the Data Lab environment. Development environment is connected to the KairosDB interface through the client software library communicating with the KairoDB instance with HTTP protocol.
Environment variables	CASSANDRA_HOSTS - list of Cassandra DB hosts
Volumes	/opt/kairosdb - directory where the configuration files and logs are stored
Networks/Exposed ports	one instance connected to database backend network for specific tenant or shared by multiple tenants