D4.4 Medolution Platform APIs and Specification V2

Medolution

Medical Care Evolution

ITEA3 – Project 14003

Document Properties

Edited by :	François Exertier, Bull
Authors	François Exertier (Bull), Mathis Gavillon (Bull), David Kuik (Norima), James Eichele (Norima), Mihai Mitrea (IMT), Anil Sinaci (SRDC), Mert Başkaya (SRDC), Béchir Taleb Ali (Prologue), Wolfgang Thronicke (Atos DE)
Date	25/10/2017
Visiblity	Public
Status	Final

History of Changes

Release	Date	Author, Organization	Changes
0.1	09/2017	F. Exertier, Bull	Initial version, based on D4.1, ready for partners contribution
0.2	09/2017	F. Exertier, Bull	Bull contribution
0.3	10/2017	F. Exertier, Bull	Integration of contributions from SRDC, IMT, Atos DE
0.4	10/2017	F. Exertier, Bull	Integration of contribution from NORIMA
0.5	10/2017	F. Exertier, Bull	Add a glossary
0.6	10/2017	F. Exertier, Bull	Final version of NORIMA contribution integrated. Ready for review.
0.7	10/2017	F. Exertier, Bull	Integration of fixes from official PMT Review by Gokce Banu Laleci Erturkmen
1.0	10/2017	F. Exertier, Bull	Taking into account feedback from official PMT Review by Céline Badr. Final version.

Table of Contents

1. Executive Summary

This document is an update of the deliverable D4.1: Medolution Platform APIs and Specification V1. It includes new components and architecture patterns that partners have developed in the timeframe after the delivery of D4.1, as well as updates to the ones that were described in D4.1. The choice has been made to provide an update of the V1 specification in order to get a standalone reference document. The changes brought to D4.1 are highlighted and pointed in the Changes Log section 2.

This document describes the Medolution Core platform. The main role of the Medolution Core platform is to provide the execution support for Medolution applications in terms of Big Data related services, software engineering support (design, deployment), and connection to devices and data warehouses. It makes the link between data sources (data warehouses, devices defined in WP3…), and the applications developed in WP5, that will be built using big data related services provided by the platform, and that will potentially run on the platform.

The document describes the platform principles, as initiated in the FPP document and based on the Bull Big Data Capabilities Framework (BDCF). BDCF provides capabilities to build Big Data Applications by composing software components delivered in an associated catalogue, and then to deploy it on any Cloud but, in particular, on the WP4 hosting platform, which provides hardware features particularly suitable to Big Data processing. As the Medolution platform described in this document is based on BDCF, the term BDCF is sometime used as synonym of Medolution Platform.

The document includes the description of most of the components and blueprints (architecture patterns also called topologies) delivered with BDCF that are suitable for building Medolution applications dealing with Big Data processing. These are typically generic Big Data processing components and blueprints, including those related to Big Data Real Time stream processing. In addition, Atos DE has provided a blueprint representative of the LVAD (Left Ventricular Assist Device) Use Case, which should be implemented as a BDCF topology.

Among platform components, the device connector components have the role to collect data generated by devices and to route it to big data processing components.

Finally, security is addressed. In particular, Norima has provided the specification of components implementing anonymization.

The APIs of the WP4 platform are provided in two categories:

- the APIs of the catalogue components which will be used by applications are described in section 5,
- the platform API used to operate the platform is described in section 8.

2. Changes Log

This section highlights the main changes between the first version of Medolution Platform APIs and Specification V1 (D4.1 [1]) and the second version presented in this document (D4.4).

Some updates have been contributed on device and data connectors:

- The TOSCA definition of the Device Connector API has been added in section 5.3.3.2.
- Progress relative to designing a Medolution device connector when device data are not exposed has been added to section 5.3.4.
- An EHR Data Connector has been added in section 5.4.2.

Four components related to Data Analytics processing have been added in the Medolution Big Data Components Catalogue in section 5:

- Jupyter is described in section 5.5.3
- Python is described in section 5.5.2
- Apache NIFI is described in section 5.5.4
- Flink is described in section 5.5.5

Two Big Data topologies have been added in section 6, following up studies conducted during the project on streaming processing technics, in order to satisfy the low latency required by Healthcare applications dealing with devices data flows:

- The Kappa architecture is described in section 6.2
- Event Sourcing and CQRS architectures are described in section 6.3

Few considerations about scaling and security have been added to the section 6.4 describing the LVAD use case topology.

About security,

- Section 7.1 has been updated including an example of a secured data lake set up for the purpose of a partner use case.
- Section 7.2 has been updated according to some design changes in the anonymization components.
- Section 7.3 has been updated, adding results of work conducted on securing the Elastic components stack.

3. Introduction

This document describes the WP4 platform and its APIs which satisfies the following objectives defined in the FPP:

- Software Platform
  - to connect Heterogeneous Healthcare Data Sources (Sensors, Data Warehouses, Internet sources…)
  - to provide Big Data related technical services (mediation, storage, analysis…)
  - to connect, compose and host applicative services (WP5) based on Big Data services
- Reactive Big Data Framework
  - to deliver insights on-the-fly to healthcare staff, with low latency, based on experimenting in memory processing, stream processing, Lambda architectures
- Software Engineering
  - to build and deploy easily “à la carte” Big Data Software Stacks
- Security
  - to enable encryption, anonymization, tracking

As mentioned in Medolution deliverable D1.1 [2] about the State of the Art Analysis, regarding the IoT Big Data platforms, the innovative aspects of this Core Platform will be to reduce the complexity of building Big Data software applicative stacks, however still allowing components choice and facilitating cloud deployment; the focus will also be set on integrating advanced real-time Big Data processing technologies. Taking into account access to Healthcare devices and Data Privacy aspects in this platform will also be part of the solution. Medolution partners have provided innovative solutions to connect any kind of devices to the Medolution platform, enabling to route data to the appropriate data analysis and processing components (cf. section 5.3). Experimenting innovative anonymization technics will be possible through the integration of the anonymization system developed in this work package (cf. section 7.2).

Section 4 describes the principle of the platform, i.e. a composition PaaS based on a catalogue of Big Data and Medolution related software components, providing scalability, reliability and security.

Section 5 describes the platform catalogue components, mainly generic big data processing related components (storage, collect, analysis, and visualization) delivered with the BDCF software, as well as the specification of the Medolution Device Connector, which enables connection of the IoT devices of WP3 with the WP4 platform.

Section 6 is dedicated to blueprints that will be delivered with the platform to satisfy Medolution applications architecture requirements. It provides real time stream processing blueprints that are delivered as topologies in the BDCF catalogue, and the blueprint which supports the LVAD Medolution use case.

In section 7 the specification of anonymization components and general security features as supported in BDCF are briefly described.

Finally, section 8 provides a short description of the API to operate the platform.

4. Medolution Platform Description

This section describes the principles of the Medolution Platform.

4.1. Data and services hub platform

The Medolution platform aims to support whole Big Data capacity and IoT connectors as a service. It is composed by three main layers:

- **Virtualization**: hardware resources are shared between all applications. An Infrastructure as a Service (IaaS) provides the ability to allocate these resources on demand.
- **Orchestration**: each application requires some resources (compute, network, disk space) which are allocated on any IaaS platform.
- **Provisioning**: manage software requirements, installation and configuration.

The Big Data Capability Framework (BDCF) aims to provide those three layers. It brings a packaged solution to create Big Data application clusters on demand, through an intuitive administrative user interface. Big Data applications initially targeted by BDCF are mainly distributed storage and processing (Hadoop) applications based on MapR or Hortonworks and Log Analysis applications based on Elastic Stack components. In addition, BDCF provides other useful components like a message broker (Kafka), a consensus system (Consul), a studio for data scientists (RStudio) and other technical components like Java and Python, allowing a detailed technical architecture design.

All components described above are part of a catalogue and can be connected all together to create distributed applications. Catalogue and application design is managed through a software named Alien4Cloud which provides an intuitive graphical user interface (GUI). It is a tool, part of BDCF, that aims to provide management for enterprise cloud and help enterprise to move their applications to a cloud, and based on project constraints, reach continuous delivery.

As moving to the cloud for an enterprise is a structural change, Alien4Cloud leverages the TOSCA[1] standard that is the most advanced and supported standard for the cloud. TOSCA stands for Topology and Orchestration Specification for Cloud Applications, it is an OASIS open standard that defines the interoperable description of services and applications hosted on the cloud and elsewhere; including their components, relationships, dependencies, requirements, and capabilities, thereby enabling portability and automated management across cloud providers regardless of underlying platform or infrastructure. These characteristics also facilitate the portable, continuous delivery of applications (DevOps) across their entire lifecycle.

The usage principle of BDCF is illustrated in the figure below, where the application designer defines the blueprints of the components (also known as topology) which compose her/his application software suite through the graphical TOSCA tool (Alien4Cloud), selecting her/his components in the BDCF catalogue, assembling and configuring them, and then benefiting from the automated orchestration of this topology.

[1] https://www.oasis-open.org/committees/tosca

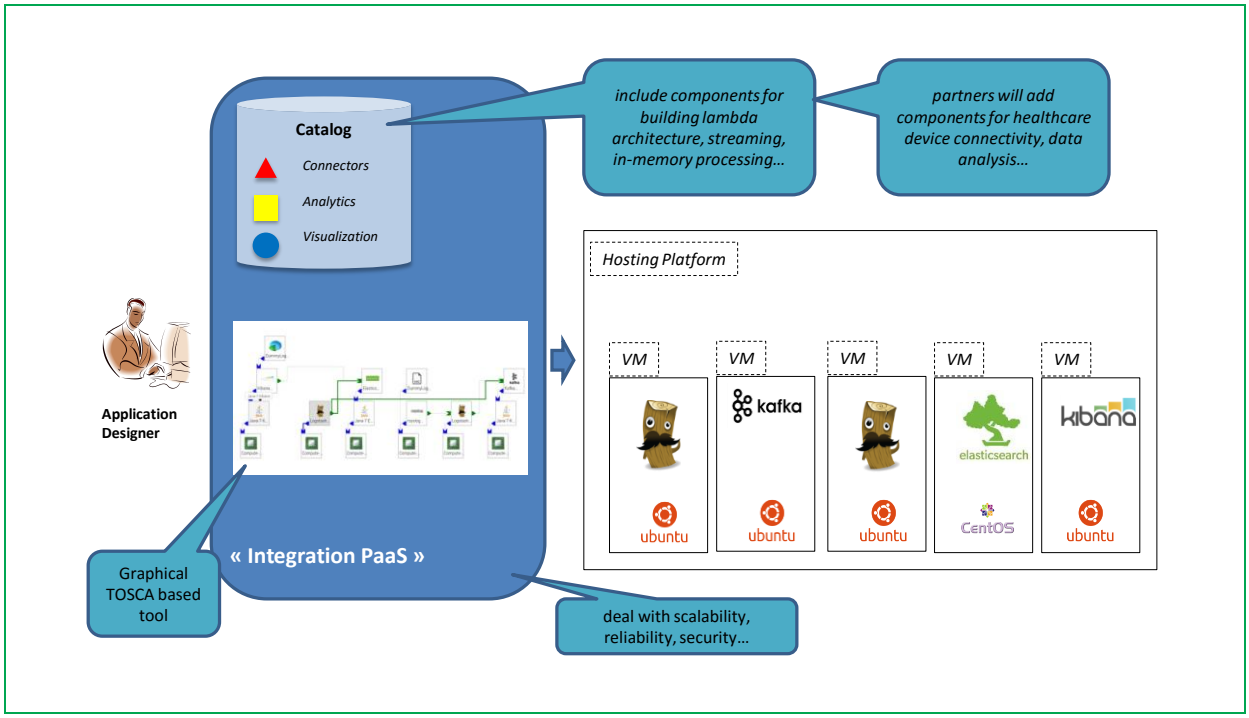

Figure 1: BDCF Platform

The Orchestration layer also brings reliability and scalability, which provides the elasticity to use the right amount of resources.

- **Reliability**: algorithms monitor the health of software components and virtual machine state. If one of those is falling down, the orchestration tool is able to instantiate new resources, to reconfigure and/or restart the service.
- **Scalability**: application designer can scale out some resources at deployment step or dynamically allocate more or fewer resources post deployment according to its needs.

This framework fits in the Medolution architecture as it enables the setting up and deployment of Big Data Applications that will:

- rely on big data analysis components to implement the smart Medolution applications of WP5 and the Medolution Use Cases. As the Medolution requirements are elaborated during the project, additional components and topologies (blueprints representing components assemblies) are added to the platform catalogue.
- rely on connector components to interconnect Medolution identified healthcare devices (WP3) and Medolution healthcare data warehouses to the data processing services. Such connector components are also introduced during the project.

4.2. Components catalogue and topologies

Big Data Capability Framework (BDCF) provides a catalogue of tools which allow Big Data actors to store, analyse, explore, and visualize data coming from some different kinds of sources. Those can be IoT devices, applications logs, etc...

In Medolution platform, these tools are described following the TOSCA standard. Each of them is called a component. Alien4Cloud allows connecting these components to each other to create a distributed application. This arrangement is called a topology.

Once it has been designed, the application can be deployed, which will install, setup and start software components.

Each component which follows the **TOSCA** standard must define at least some elements:

- **Properties**: fields which allow end-users to customize this component and could be used at installation, configuration or start step.
- **Attributes**: properties which will be generated at runtime.
- **Requirements**: define a dependency from this component to other ones.
- **Capabilities**: simple description of what kind of operation the component will do.
- **Artefacts**: attached items which will be uploaded and used during component instantiation.

Creating a BDCF component from existing software is a matter of defining the associated TOSCA elements and developing the scripts that will allow configuring, installing, starting, and stopping the component. A BDCF component developer guide [3] has been written and provided to Medolution partners (available on the project SharePoint).

Components of BDCF to be used in Medolution, as well as those to be developed and included in the catalogue for Medolution, are described in section 5.

4.3. Orchestrator and target Infrastructures

Big Data Capability Framework relies on an orchestrator to create applications over a Cloud infrastructure. The aim is to coordinate Cloud resources to be able to deploy a defined distributed application, based on its TOSCA description. The orchestrator will check available resources, will hold it and then use it. Some post deployment step can be defined to manage resiliency, scaling and monitoring. The current orchestrator supports some Cloud infrastructure like Amazon Web Services, VMWare or Azure. Medolution infrastructure is currently based on Openstack, an open source Infrastructure as a Service (IaaS) solution.

4.4. Medolution security and privacy

The main security goal in WP4 is to ensure data privacy in the context of Healthcare Big Data processing. This involves dealing with data privacy at the level the Big Data storage capabilities, and at the level of Data collection capabilities, i.e. HealthCare Data Warehouse connectors (like EHR connectors), Device Connectors.

Two aspects are addressed in the project:

- The Big Data Platform security by itself, i.e. securing data access and data communication with the platform itself
- Data Anonymization capabilities, where the specification regarding Data Anonymization components has been provided.

Both are described in section 7.

5. The Medolution Big Data components catalogue

This section describes the components that will be made available in the Platform catalogue to enable Medolution application providers to compose their application.

This includes:

- the TOSCA description of each component and corresponding relationships (how they connect to other components),
- their APIs,
- their high availability and scalability capabilities,
- their security features

These components are building blocks to support any kind of data analysis applications, especially for handling streaming real time big data processing, and for dealing with medical devices and information system connection, as it was required by the Medolution project (in FPP and D1.1 [2]). Not all these components are currently used in Medolution use cases; sometime it is a matter of timeline, since they will be used later in the project; also the use cases requirements do not cover all possibilities in term of data processing. The French and German uses cases are already using some of these components.

5.1. Core Foundational Components

Some of the components described in this section may use one or both of the technical components below:

- The Java component is a technical component allowing other software components to choose the required Java version.
- Consul is a technical component allowing other software components to discover each other in a flexible, highly available and fault tolerant way. The same Consul component is used for two different purposes: it can run as a Consul Server or as a Consul Agent. Consul Servers are responsible for storing and replicating data between themselves, while Consul Agents are responsible for registering functional services, store services' distributed configuration, performing health checks on nodes hosting those services and serving service discovery requests through their DNS interface.
- The Compute Node component represents a Virtual Machine to be allocated and on which the components will run.

5.2. Data collection and storage components

This section describes a set of BDCF components which are concerned with collecting data from various sources and storing them for future processing. Subsequent sections will describe components for processing data.

5.2.1. Elasticsearch

Elasticsearch is a search server based on Apache Lucene. It provides a distributed, multitenant-capable full-text search engine with a RESTful Web interface and schema-free JSON documents.

In the Elastic Stack architecture, Elasticsearch is the database that stores the logs sent by Logstash, and provides to Kibana the needed data for visualization.

The infrastructure administrator needs to understand the following Elasticsearch internal concepts:

- For Elasticsearch underlying concept, management and configuration refer to the official documentation: https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html
- For more information on Apache Lucene, refer to http://lucene.apache.org/.

5.2.1.1. Elasticsearch API

The Elasticsearch tool is accessible via a REST API which offers a way to read and write data on the datastore.

5.2.1.2. Elasticsearch Relationships

The only prerequisite of the Elasticsearch component is to be attached to a Java component that is configured to use Java 7 or greater. This Java component must be hosted on a Compute component.

The Elasticsearch component must be connected to a Consul Agent hosted on the same Compute node, to perform cluster discovery. This connection to a Consul Agent is prerequisite when:

- the Elasticsearch cluster has several nodes, or
- the Elasticsearch search_endpoint capability is used.

Elasticsearch has the capability to be used as a search_endpoint by the Logstash and Kibana components. This means that you can connect the components that use Elasticsearch by setting their search_endpoint prerequisite.

In the Elastic Stack chain, Logstash and Kibana use Elasticsearch as target database. The data is stored in indexes dedicated to those components.

Elasticsearch includes indexes containing the logs provided by Logstash. By default Logstash creates one different index per day, named Logstash-YYYY-MM-dd. It is possible to override this name in the Logstash configuration file by modifying the index option.

Kibana stores its configuration into Elasticsearch under the kibana index. Do not modify it manually or remove it if you want Kibana to work properly.

5.2.2. Kafka

Kafka is a distributed, partitioned and replicated publish-subscribe messaging system.

In the Elastic Stack architecture, Kafka offers the following features:

- Store into a Highly Available, fault tolerant system, the logs ingested by the Elastic Stack before their indexing by Logstash.
- Load-Balance data across the Kafka cluster and between its subscribers (Logstash Indexer in our case) by natively partitioning data across the cluster.
- Decouple the processing part of the Elastic Stack architecture from the ingestion part by implementing a retention queue.

The Kafka community uses the following terminology:

- Kafka maintains feeds of messages in categories named topics. Topics are split into partitions, which brings data reliability between all nodes of the Kafka cluster;
- Processes that publish messages to a Kafka topic are named producers
- Processes that subscribe to topics and process the feed of published messages are named consumers.
- Kafka runs as a cluster of one or more servers. Each of these servers is named a broker.

5.2.2.1. Kafka API

Kafka provides some Java API which allows a user to consume, publish or stream the content of a Kafka broker.

5.2.2.2. TOSCA component description

Kafka

In the Elastic Stack architecture, Kafka topics are connected with a Logstash instance named the shipper, which is responsible to ingest logs from different inputs (for instance Rsyslog or lumberjack) and which acts as the producer by publishing those logs into Kafka topics. On the other hand, Kafka topics are connected with another Logstash instance named the indexer, which consumes published logs in order to process them.

A Kafka node is hosted on a Java node, which is hosted itself on a Compute node. A Kafka node requires to be related to a Consul Agent hosted on its Compute node. On top of this stack you can deploy as many different topics as you need. Each topic has its own configuration.

The minimum and recommended version of Java is JRE7.

Properties

kf_heap_size*: This property allows setting the heap memory size that is allocated to Kafka java process, It allocates the same value to both initial and maximum values (ie -Xms and -Xmx java options).*

default: "1G"

zk_heap_size*: This property allows setting the heap memory size that is allocated to Zookeeper java process (ZooKeeper is a coordination service for distributed applications, the corresponding BDCF component, built on the MAPR Zookeeper element, is not detailed in this documentation). It allocates the same value to both initial and maximum values (ie -Xms and -Xmx java options).*

default : "500M"

repository*: This property gives the opportunity to specify an alternative download repository for this component artefacts. It is your responsibility to provide an accessible download url and to store required artefacts on it. You should specify only the base repository url. Artifacts names will be appended to it, so this property could be shared among several components using the inputs feature.*

default : ""

Requirements

***java**: Kafka should be hosted on a Java component. Java 7 or greater is required.*

***host**: Kafka component has to be hosted on a Compute.*

***consul**: Kafka component has to be connected to a local (hosted on the same Compute) Consul Agent. This is required to perform cluster discovery.*

***filesystem_endpoint**: Kafka may be connected to a filesystem in order to store its runtime data on it. A typical use case would be to link this filesystem to a block storage in order to achieve data resilience and recovery.*

Kafka Topic

A Kafka Topic should be hosted on a Kafka component and may be configured through the following properties.

Properties

***topic_name**: The topic name (value should match the following pattern: [-_A-Za-z0-9]+)*

default: “”

***partitions**: Number of partitions for this topic*

default : 1

***replicas**: Number of replicas for this topic. Should be at most the number of hosting Kafka Component instances.*

default : 1

***min_in_sync_replicas**: When a producer sets request_required_acks to in_syncs, min_insync_replicas specifies the minimum number of replicas that must acknowledge a write for the write to be considered successful. If this minimum cannot be met, then the producer will raise an exception (either NotEnoughReplicas or NotEnoughReplicasAfterAppend). When used together, min_insync_replicas and request_required_acks allow you to enforce greater durability guarantees. A typical scenario would be to create a topic with a replication factor of 3, set min_insync_replicas to 2, and produce with request_required_acks of in_syncs. This will ensure that the producer raises an exception if a majority of replicas do not receive a write.*

default : 1

***retention_minutes**: The number of minutes to keep a log file before deleting it.*

default: 10080 (7 days)

***segment_minutes**: The number of minutes after which Kafka will force the log to roll even if the segment file isn’t full to ensure that retention can delete or compact old data.*

default: 10080 (7 days)

***segment_bytes**: Segment file size for the log.*

default: 1073741824 (1GB)

Requirements

***kafka_host**: Kafka topics are hosted on Kafka components.*

5.2.2.3. Kafka Relationships

Any Kafka node is related to a Consul Agent hosted on the same Compute node. This relationship is obtained by binding the consul prerequisite of the Kafka node to the agent capability of the Consul node.

As explained above, in the Elastic Stack architecture Kafka topics are connected with a Logstash that publishes messages and another Logstash that consumes those messages.

5.2.3. Logstash

Logstash is a tool for receiving, processing and outputting logs. All kinds of logs are concerned: system logs, webserver logs, error logs, application logs, etc. Logstash provides a powerful pipeline for storing, querying, and analysing logs. It includes an arsenal of built-in inputs, filters, codecs, and outputs.

The Logstash event processing pipeline includes three stages:

Inputs → Filters → Outputs

Inputs generate events, Filters modify them, and Outputs ship them elsewhere.

- Inputs are used to get data into Logstash (see https://www.elastic.co/guide/en/logstash/2.3/input-plugins.html). Some of the most commonly-used inputs are:
  - **File:** Reads from a file on the filesystem.
  - **Syslog**: Listens on the port 514 for syslog messages and parses according to the RFC3164 format.
  - **Lumberjack:** Processes events sent in the lumberjack protocol.
  - **Elasticsearch:** Reads query results from an Elasticsearch cluster.
- Filters are intermediary processing devices in the Logstash pipeline (see https://www.elastic.co/guide/en/logstash/2.3/filter-plugins.html). You can combine filters with conditionals to perform an action on an event if it meets certain criteria. Some useful filters include:
  - **Grok:** Parse and structure arbitrary text. Grok is currently the best way in Logstash to parse unstructured log data into something structured and queryable.
  - **Mutate:** Perform general transformations on event fields. You can rename, remove, replace, and modify fields in your events.
  - **Drop:** Drop an event completely, for example, debug events.
  - **Clone:** Make a copy of an event, possibly adding or removing fields.
  - **Geoip:** Add information about geographical location of IP addresses.
- Outputs are the final phase of the Logstash pipeline (see https://www.elastic.co/guide/en/logstash/2.3/output-plugins.html). An event can pass through multiple outputs, but once all output processing is complete, the event has finished its execution. Some commonly used outputs include:

- **Elasticsearch:** Send event data to Elasticsearch. If you plan to save your data in an efficient, convenient, and easily queryable format Elasticsearch is the way to go.
- **File:** Write event data to a file on disk.

A Logstash node should be hosted on a Java node, which is hosted itself on a compute node.

The TOSCA component description exposes the following properties and artefacts:

Properties

version: *Version number*

repository: *Allows you to specify an alternative download repository for a Logstash binary.*

auto_reload: *Boolean property set to false by default. If set to true, Logstash monitors configuration changes and reloads configuration whenever it is changed.*

reload_interval: *If auto-reload is true, this property specifies how frequently to poll the configuration location for changes, in seconds.*

stdout: *Boolean property set to false by default. When set to true, the simple stdout output is added to the output configuration of Logstash.*

heap_size: *Allows setting the heap memory size for Logstash java process. Default value is 500M. It allocates the same value to both initial and maximum values (ie -Xms and -Xmx java options).*

log_level: *Define Logstash log level. By default it is quiet and very few logs are generated by Logstash. All logs are redirected to a file except if you set 'stdout' to 'true'.*

Artefacts

input_conf: *input configuration*

output_conf: *output configuration*

filter_conf: *filter configuration*

5.2.4. HBase

HBase is a non-relational database based on Hadoop distributed filesystem which allows storing billions of rows of millions of columns. This is a distributed and scalable database system.

HBase is made up of two distinct components: HBase Master and HBase Region server.

5.2.4.1. HBase Master

HBase Master is the database system endpoint. It allows user to ask for accessing or pushing data inside its database and manages HBase Region Server instances.

HBase API

HBase provides some Java classes which allow to interact with HBase master and to create table, upload, read data… A Web interface is also available for end user.

TOSCA component description

HBase Master implementation in BDCF is part of MapR distribution. The component name in the BDCF catalogue is MapRHBaseMaster.

Properties

MapRHBaseMaster does not have any properties.

Requirements

MapRHBaseMaster requires to be hosted on MaprWarden[2].

Capabilities

hbase_master_endpoint*: allows connecting any Alien4Cloud component to this one. It can be used for MapRHUE[3] requirement.*

Artefacts

tools*: MapR utility scripts.*

utils_scripts*: Common utility scripts for whole BDCF components.*

5.2.4.2. HBase Region Server

HBase Region server is the low level database management component. It stores data and answer user requests. In general, several instances are instantiated.

TOSCA component description

HBase Region Server implementation in BDCF is part of MapR distribution. The component name in A4C catalogue is MapRHBaseRegionServer.

Properties

MapRHBaseRegionServer does not have any properties.

Requirements

MapRHBaseRegionServer requires to be hosted on MaprWarden.

Capabilities

[2] Warden is a light Java application that runs on all the nodes in a MAPR Hadoop cluster and coordinates cluster services. The corresponding BDCF component is not detailed in this document.
[3] HUE is the User Interface to interact with Apache Hadoop and its ecosystem components, such as Hive, Pig, and Oozie. The corresponding BDCF component is not detailed in this document.

***hbase_region_server_endpoint**: allows connecting any Alien4Cloud component to this one.*

Artefacts

***tools**: MapR utility scripts.*

***utils_scripts**: Common utility scripts for whole BDCF components.*

5.2.5. Hive

Hive is a data warehouse system for Hadoop that facilitates easy data summarization, ad-hoc queries, and the analysis of large datasets stored in Hadoop-compatible file systems, such as the MapR Data Platform (MDP). Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. At the same time this language also allows traditional map/reduce programmers to plug in their custom mappers and reducers when it is inconvenient or inefficient to express this logic in HiveQL.

5.2.5.1. Hive API

Hive provides Java API to interact with HiveServer2 and Hive Metastore.

TOSCA component description

Properties

MapRHive does not have any properties.

Requirements

MapRHive requires to be hosted on MapRWarden. If you have multiple MapRHive components, you need to resolve these needs:

***hive_registerTo_haproxy**: can be connected to a HAProxyTCPHive component to manage load-balancing.*

***hive_connectsTo_mysql**: can be connected to a MySQLDatabase component to externalize the Hive Metastore.*

Capabilities

***hive_endpoint**: allows connecting any Alien4Cloud component to this one. It can be used for MapRHUE requirement.*

Artefacts

***tools**: MapR utility scripts.*

***utils_scripts**: Common utility scripts for whole BDCF components.*

***hive_files**: Required MapRHive template configuration files.*

5.2.6. Drill

Apache Drill is an open source, low-latency query engine for big data that delivers secure and interactive SQL analytics at petabyte scale. With the ability to discover schemas on-the-fly, Drill is a pioneer in delivering self-service data exploration capabilities on data stored in multiple formats in files or NoSQL databases. Drill is fully ANSI SQL compliant and integrates seamlessly with visualization tools.

5.2.6.1. Drill API

Drill provides a REST API to connect and interact with the service running queries, performing storage plugin tasks, such as creating a storage plugin, obtaining profiles of queries, and getting current memory metrics. A Web UI is also available.

TOSCA component description

Properties

MapRDrill does not have any properties.

Requirements

MapRDrill requires to be hosted on MapRWarden.

Capabilities

drill_endpoint*: allows connecting any Alien4Cloud component to this one.*

Artefacts

tools*: MapR utility scripts.*

utils_scripts*: Common utility scripts for whole BDCF components.*

5.2.7. Flume

Apache Flume provides a way to easily and efficiently move data from a source to a sink. It is based on a simple and flexible architecture which uses streaming data flows.

5.2.7.1. Flume API

Flume offers a way to interact with its service using a Java API.

TOSCA component description

Flume implementation in BDCF is part of MapR distribution. The component name in A4C catalogue is MapRFlume.

The output configuration is bound to MapR filesystem, although no relationship is necessary for that purpose.

When MapRFlume is not connected to Kafka, it is configured to open a port with Netcat on port 44444 and store data in MapR filesystem.

Properties

***component_version**: Version of the Flume component integrated*

default: 1.6.0

***hdfs_path**: Flume is configured to use HDFS as sink. An alternative path in MaprFS can be provided.*

default: maprfs:///user/mapr/flume/kafka_input/%y-%m-%d-%H

Requirements

MapRFlume requires to be hosted on MapRWarden

***kafka_input**: Input must be set to a Kafka Topic*

Capabilities

***flume_endpoint**: allows connecting any Alien4Cloud component to this one.*

Artefacts

***tools**: MapR utility scripts.*

***utils_scripts**: Common utility scripts for whole BDCF components.*

***flume_files**: Required Flume template configuration files. An example of this file is given in BDCF user documentation.*

5.3. Device connector component

Another way to collect data is to get it directly from devices. A device connector component is a Medolution component available in the platform catalogue, that can be easily incorporated in an application topology in order specify the input data flow for such an application. Its purpose is to collect a data flow from outside the platform, and to route it toward any Big Data component to be used for the application needs, i.e. an analytics processing component, a data storage component, etc.

In this section, different kinds of devices to be connected in the Medolution project are described. Section 5.3.1 describes the communication with devices that are conforming international data exchange standards. The issue with the devices whose data can only be accessed from proprietary interfaces is introduced in section 5.3.2. This results in two categories of devices to be considered. For the first category, devices whose data is directly accessible via the APIs provided

by the devices is described in section 5.3.3., The data to be collected with their respective standard and formatting information is described in section 5.3.3.1, and the Device Connector Component external API, i.e. the API to be used from outside the Medolution platform to inject data, is described in section 5.3.3.2. In section 5.3.4, the integration with devices where device manufacturer does not expose to the user the data generated by the device is presented.

5.3.1. Devices to be connected

There are several devices that can be integrated into the Medolution platform to collect information about the patients and deduce meaningful results for the physicians through health data analytics. Although Medolution prefers to use international and national standards for integration, most of the devices used in the healthcare sector are not conformant to the standards. It can be said that each manufacturer creates its own data models, serialization formats and/or protocols and this makes the integration impossible without their inclusion in the development process. For example, considering the LVAD use-cases, there are different LVAD manufacturers which are currently implanted into several patients. Most of them are even not providing any remote monitoring capability, though the one providing such a capability is not doing this through an open standard and this makes the integration impossible.

Apart from the proprietary devices, Medolution enables to collect information from many other devices which conform to international standards. This section covers the standards based connector components because the proprietary ones needs to be integrated through custom implementations for each device used for the respective use-cases.

The standards based interaction with the Medolution devices is one-way; data is collected from the devices into the Medolution platform. LVAD use-cases include devices (i.e. LVAD pumps) which require two-way communication; some parameters of the pumps are needed to be adjusted. These kinds of interactions are solved through custom implementations during the pilot implementations.

5.3.1.1. Bluetooth enabled medical devices

Personal health devices that use Bluetooth as a communication medium, based on ISO/IEEE 11073[4] standards, can be used as data sources to Medolution Big Data platform. These devices can be blood pressure monitors, glucometers and the like Continua Health Alliance compliant medical devices covering the Medolution use-cases defined in *D1.2 – As-Is Landscape of Clinical Care at Pilot Sites and To-Be Pilot Application Scenarios*.

ISO/IEEE 11073

ISO/IEEE 11073 Health Informatics - Medical / health device communication standards are a family of ISO, IEEE, and CEN joint standards addressing the interoperability of medical devices. Continua Health Alliance(CHA)[5] conformant products make use of the ISO/IEEE 11073 Personal Health Data (PHD) standards that specifically address the interoperability of personal health devices. The standard has the concept of agents and managers that defines communicating parties. Agents are typically small, battery-powered medical devices whereas managers are typically computers or smart phones. Agents are restricted to communicate with a single manager at a time yet managers can communicate with multiple agents. An example would be a smart phone as a manager, connected to multiple medical devices of a patient, e.g. weighing scale,

[4] http://www.11073.org/
[5] http://www.continuaalliance.org/

thermometer, and insulin pump. Communication is bi-directional; weighing scale can send measurements to phone as well as phone can control insulin pump. The standard only defines messages that travel between agents and managers but not the transport protocol. Bluetooth, Zigbee or USB can be used to transfer messages.

5.3.1.2. Activity tracker wristband

We enable activity tracking wristband as an additional data source to be able to collect more useful information from patients. Most of the commercial activity tracker wristbands has their own restricted APIs and does not allow direct data access with Bluetooth or other communication protocols. As a result of investigating other open options, Angel Sensor[6] stands out as it offers fully open access to its data with Bluetooth Low Energy communication.

Angel Sensor provides Heart Rate, Health Thermometer measurements as standard Bluetooth Low Energy (BLE) services as well as step count, blood oxygen saturation, fall detection, accelerometer, gyroscope measurements as custom BLE services.

5.3.1.3. Smart phone

In consequence of having Bluetooth enabled health and activity data producing devices to collect data from, a smart phone is used as a gateway between these data sources and the Medolution big data platform (BDCF). Both Android and iOS mobile devices can be used as a gateway since the requirements are only Bluetooth and an Internet connection. The devices send their measurements to the smart phone and an application (possible a dedicated Medolution app to collect information and show some statistics to the user) is responsible to pass this information conveniently to the Medolution BDCF servers.

Google Fit and Apple HealthKit

Using smart phones as a gateway enables us to gather more information from patients. Based on the gateway device in use, Google Fit[7] or Apple HealthKit[8] is utilized in order to collect more data from patients. Both Google Fit and Apple HealthKit is used as a health and activity data repository by other health and fitness applications, furthermore Google Fit collects and derives its own data using phone sensors and GPS location.

Google Fit provides various data types including heart rate, step count, nutrition and allows developers to create custom data types[9]. Apple HealthKit also provides similar fitness related data types along with health related types including blood glucose, oxygen saturation, body temperature and the like[10].

Mobile sensing library

Smart phones have various sensors and components that contain valuable information about patients. To reach out this data, we use a mobile sensing library which can be configured to collect desired data from mobile phone components periodically and store them in a preferred

[6] http://www.angelsensor.com
[7] https://www.google.com/fit/
[8] https://developer.apple.com/healthkit/
[9] https://developers.google.com/fit/android/data-types
[10] https://developer.apple.com/reference/healthkit/1627060-healthkit_constants

format so that the mobile application can use this stored data to submit to the Medolution Big Data platform.

5.3.2. Devices particularities with respect to data connection

The section above brings to light that from the functional point of view, the MEDOLUTION DEVICE CONNECTOR shall output a unitary data format, able to accommodate the large variety of heterogeneous data formats inner to the devices required by the MEDOLUTION use cases. Thinking beyond the strict MEDOLUTION framework, such a unitary data format should also be interoperable.

Actually, from the practical point of view, two different situations are encountered in practice. The first situation corresponds to the case in which the device manufacturer exposes the data generated by the application as well as the proprietary data format as direct output of the device. In such a situation, the MEDOLUTION DEVICE CONNECTOR ensures a syntax management function. Cf. Figure 2. From a conceptual point of view, the MEDOLUTION DEVICE CONNECTOR includes data formats and API.

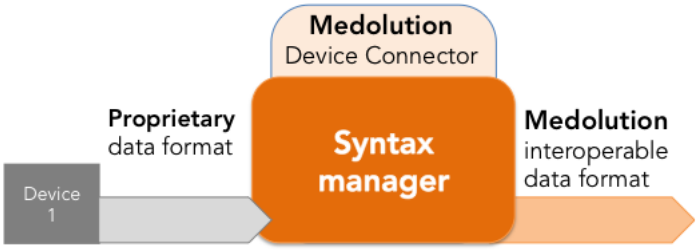

Figure 2: MEDOLUTION DEVICE CONNECTOR when the device proprietary data format is exposed.

The second situation corresponds to the case in which the device manufacturer does not expose to the user the data generated by the device; instead, it encapsulates them into a proprietary wrapper and subsequently forward them to a proprietary application which is supposed to process and to present them to the user. In such a situation, the MEDOLUTION DEVICE CONNECTOR becomes more complex, requiring an additional DEVICE PROXY whose functionality is to virtually de-capsulate the data, cf. Figure 3 .

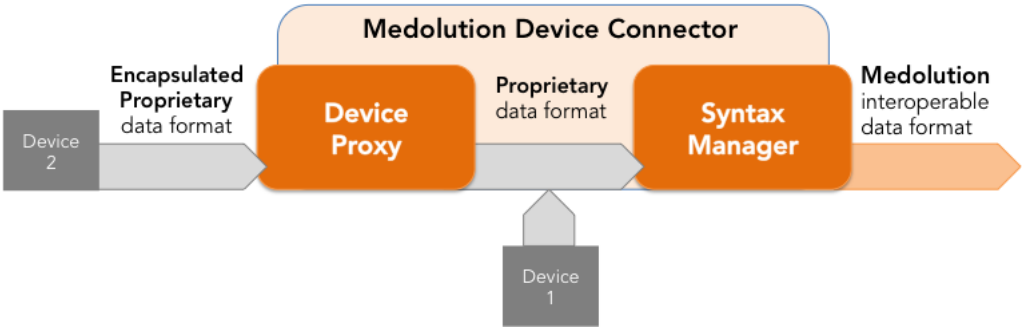

Figure 3: MEDOLUTION DEVICE CONNECTOR when the device proprietary data format is encapsulated (Device 2) vs. exposed (Device 1).

Of course, from the data format and interfaces point of view, the Device Proxy and the Syntax Manager should follow the same philosophy.

5.3.3. Medolution device connector when device data are exposed

5.3.3.1. Data to be collected

With the use of the devices and sensors, medical and personal data will be transferred to the big data processing environment of Medolution. Types of such data form an abstraction layer can be listed as follows:

- Personal health data from medical devices (e.g. pulse, blood pressure, blood glucose)
- Sensor data from implanted devices (e.g. LVAD)
- Fitness and health data from activity tracker wristband (e.g. heart rate, skin temperature, step count)
- Health and Fitness records from Google Fit or Apple Healthkit.
- Various information from mobile sensing library (e.g. GPS location, Wifi scan results)

Custom devices (which do not conform to the international standards, i.e. LVAD) serve data in proprietary formats, hence they will be processed through custom implementations during the project lifetime based on the specific data handling requirements.

For the ISO/IEEE 11073 based device integration and fitness/health sensor (wristlet) integration, smart phones play the intermediary role to collect data from these personal devices and send data to the Medolution Big Data platform through the specific APIs. This deliverable focuses on the second part: transferring data to the platform through the APIs. There might be other sources of data which can directly transfer data to the platform APIs without any Bluetooth intermediary such as a smart phone.

The API structure follows the REST principles and the data model of the payloads is designed to be based on Open mHealth.

Open mHealth

Open mHealth is a non-profit organization aiming integration of digital health data. They provide common data schemas for mobile health data types.

Data point schema is used as the common format, as illustrated in Figure 4.

```
{
    "body": {
        "heart_rate": {
            "value": 111,
            "effective_time_frame": {
                "date_time": "2016-03-15T07:30:11.099Z+02:00"
            },
            "unit": "beats/min"
        }
    },
    "header": {
        "id": "FBB8640D-8A2F-4584-9AB5-B857044E272C",
        "creation_date_time": "2016-03-15T13:09:08.802ZZ",
        "schema_id": {
            "version": "1.0",
            "namespace": "omh",
            "name": "heart-rate"
        },
        "patient_id": "A8BBACEE-CFFC-42A9-A147-92E90474720A"
    }
}
```

Figure 4: An example of Open mHealth Data Point Schema

Basically, “header” contains metadata about the resource whereas “body” contains actual measurement or value. Another “body” example is blood glucose schema as shown in Figure 5.

```
{
    "blood_glucose": {
        "unit": "mg/dL",
        "value": 120
    },
    "effective_time_frame": {
        "time_interval": {
            "start_date_time": "2013-02-05T07:25:00Z",
            "end_date_time": "2013-06-05T07:25:00Z"
        }
    },
    "descriptive_statistic": "minimum"
}
```

Figure 5: "Body" if a blood glucose measurement through Open mHealth

5.3.3.2. Device Connector Application Program Interface (API)

The associated API for device data collection acts as an “Upload Service” which is a Restful API and used by the Bluetooth intermediaries (smart phones) to transfer the collected obtained data from the devices to the Medolution Big Data platform. The API accepts binary and JSON payloads for POST operation.

5.3.3.2.1. Binary payload

Binary payload endpoint accepts SQLite database files containing two tables: Data and FileInfo. The table structure is presented in the ER diagram in Figure 6. The tables and the field can be described as follows:

- Data
  - id: Holds database entry id.
  - name: Holds the source of the value.
  - timestamp: Holds the timestamp of the value.
  - value: Holds the Open mHealth Data Point schema formatted value of the resource.
- FileInfo:
  - id: Holds database entry id.

- dbname: Holds database name.
- device: Holds unique device id which will be used to identify patients.
- installation: Holds unique installation id of the mobile application.
- uuid: Holds unique database file id.
- created: Holds the creation time of the database file.

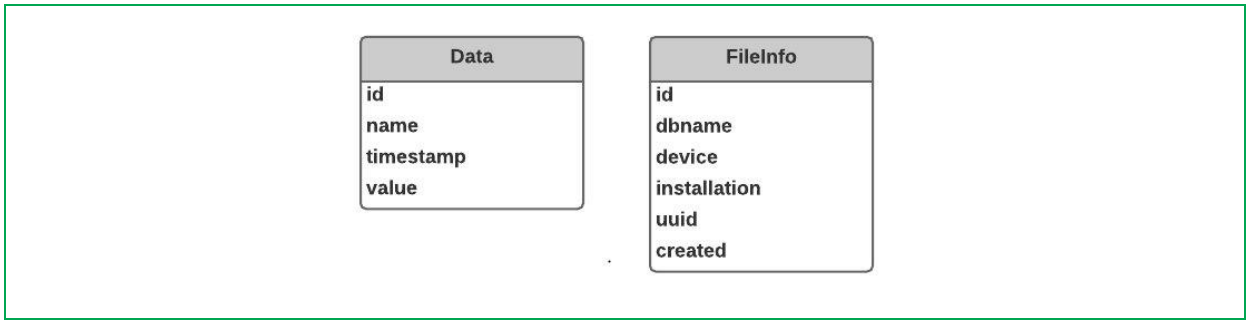

Figure 6: Simple ER diagram of the binary payload of the API

The HTTP Post command should be issued as follows:

- POST [base]/data
  - The [base] indicates base URL for REST service e.g. www.medolution.org/service/rest/upload

The content is provided using multipart form data format, with the parameter name “uploadedfile”. The service parses the given database file, and forwards its entries to relevant Big Data Platform entry points.

Table 1: Binary payload of the API

Binary Payload	
Allowed HTTP methods	POST
Path including path parameters	data
Query parameters	Not applicable
POST data	uploadedfile: {Database file content in binary}
Success POST response	200 ok empty
Error responses	400 Bad Request - if file is not in desired structure. 415 Unsupported Media Type - if content-type is not multipart form data 500 Internal Server Error - in case of internal exception

5.3.3.2.2. JSON payload

JSON payload endpoints accepts Open mHealth formatted JSON lists. HTTP POST command will be used as follows:

- POST [base]/[source]/[type]
  - The [base] indicates base URL for REST service e.g. www.medolution.org/service/rest/upload
  - The [source] indicates source of the data e.g. healthkit, ios.
  - The [type] indicates type of the data e.g. location, wifi.

The content is provided with application/json format. The service forwards each element in the given JSON List to relevant Big Data Platform entry points.

Table 2: JSON payload of the API

JSON Payload	
Allowed HTTP methods	POST
Path including path parameters	/{source}/{type}
Query parameters	Not applicable
POST data	JSON list string
Success POST response	200 ok empty
Error responses	400 Bad Request - if string is not in desired structure. 404 Not Found - if {source} or {type} is invalid 500 Internal Server Error - in case of internal exception

TOSCA component description

***Component Name*: DeviceConnector**

Properties

port: *The port of the API*

Type: Integer

Constraints: [1, 65535]

Default: 8080

base-uri: *Base uri of the API. (Binary payload complete Uri is -> [ip] : [port] / [base-uri] / data)*

Type: String

Default: "inbound"

Requirements

DeviceConnector component requires to be hosted on Java.

DeviceConnector component requires connecting to a Kafka Topic.

DeviceConnector component requires connecting to a Consul agent.

Capabilities

***kafka_output**: allows connecting to a Kafka topic.*

***consul:** allows connecting to a Consul agent*

Artifacts

***scripts:** Lifecycle scripts of the DeviceConnector component*

***utils_scripts:** Common utility scripts for whole BDCF components*

***consul_utils:** Common utility scripts for Consul operations*

***runnable:** Executable jar file of the RESTful web service*

5.3.4. Medolution device connector when device data are not exposed

In such a situation (i.e. case 2 in Figure 3), the data generated by the device are encapsulated into a proprietary wrapper and subsequently forward them to a proprietary application which is supposed to process and to present them to the user.

As explained above, in such a situation, a Device Proxy should be considered in conjunction with the Syntax Manager.

The DEVICE PROXY has three sub-modules: Virtual Access to the Application, the Virtual Control of the Application and the Semantic Data interpreter, cf. Figure 7.

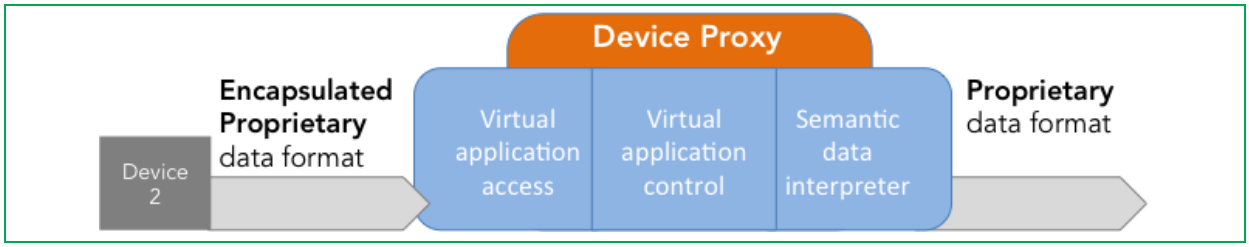

Figure 7: The Device Proxy overview

The interfaces towards and from the device proxy will follow the same philosophy as the Semantic Manager: they are based on REST API and JSON formats.

At the time of writing of this document, two incremental medical devices have been dealt with in Medolution.

First, the Device Proxy has been instantiated for a medical card reader (the hardware device ensuring the authentication of the medical professional and patients in France), as demonstrated during the first Medolution Review (Dec. 2016). Rather than giving access to the data themselves, this first step proved the possibility of virtualizing the devices together with their managing applications.

A second step is represented by the work carried-on between December 2016 and September 2017 which was devoted to the Proof of Concept for connecting FitBit[11] devices. The choice on the FitBit series is justified by both their outstanding market penetration (hence, devices relevant for the Medolution usage) and their strong constraints imposed by the manufacturer (hence, devices relevant from the technical point of view): according to the state-of-the-art, in order to build, specify, design and develop new applications for FitBit devices, the FitBit APIs must be used.

The Medolution solution, disruptive with respect to this state-of-the-art, is synoptically presented in Figure 8 below: while the device itself communicates with the FitBit server in a conventional way, data are intercepted at the server level and packed/converted into xml/JSON formats.

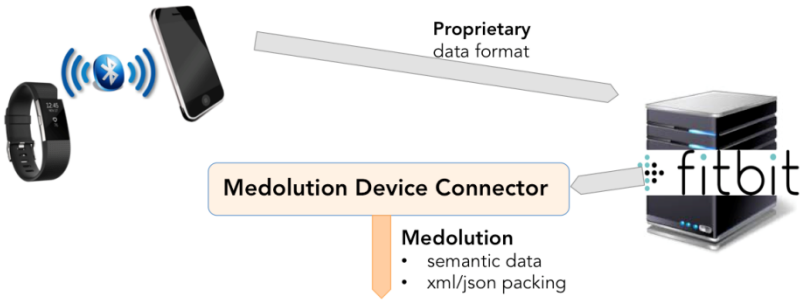

Figure 8: Synopsys for the Medolution Device Connector instantiation for FitBit devices

The principles presented in Figure 8 above are enabled by a client-server software solution working at two levels. First (see Figure 9 below), the session is initiated thanks to the credentials the FitBit owner disposes of. Secondly (see Figure 10 below), the semantic data are parsed and converted according to the Medolution specifications.

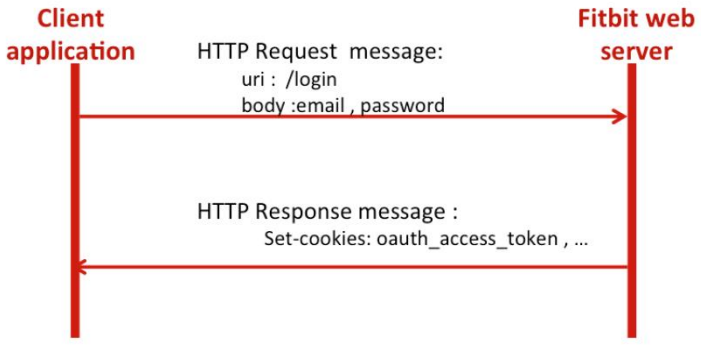

Figure 9: Medolution Device Connector instantiation for FitBit devices: session initiation

[11] www.fitbit.com

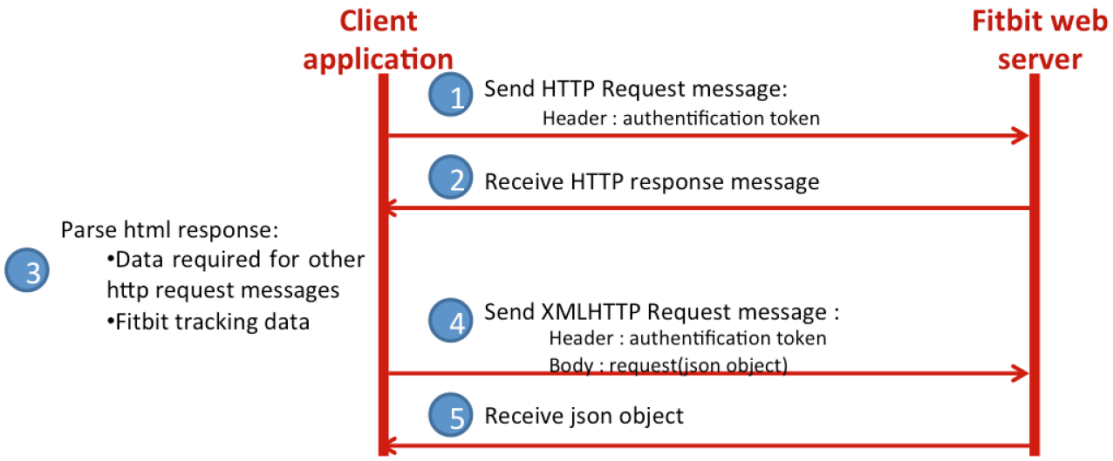

Figure 10: Medolution Device Connector instantiation for FitBit devices: data interception

On-going work is devoted on reconsidering and extending these principles for other types of devices, like the ones belonging to iHealth[12], for instance.

5.3.5. Conclusion

The MEDOLUTION Device Connector can be illustrated as in Figure 11 below.

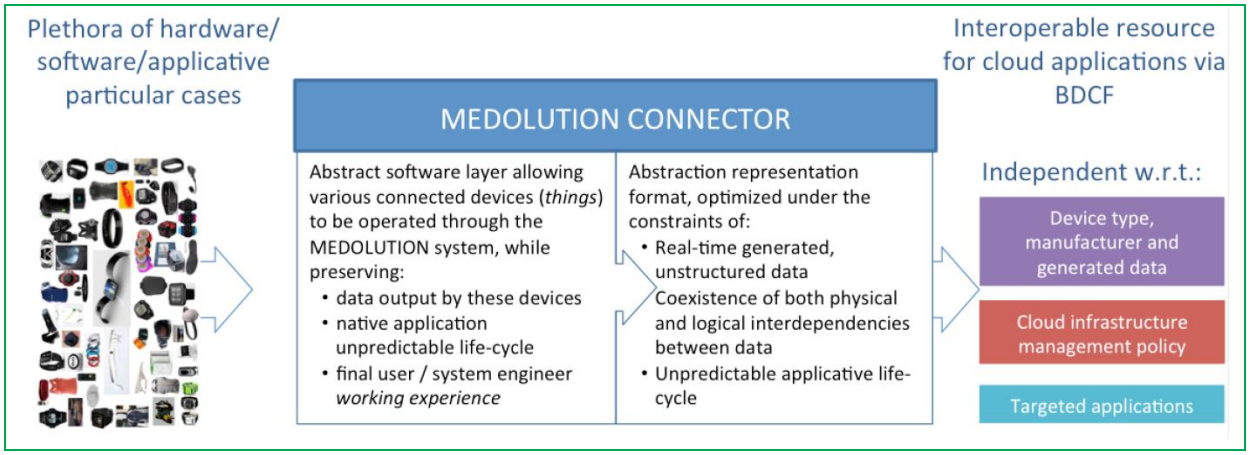

Figure 11: Global view on the device Connector component and on its possible solution.

The final objective is to provide the means for connecting any kind of device to the Medolution platform, through a BDCF component that will make that device data available for data analysis and processing components. This handles any kind of data format, standard or proprietary, and the cases where device data is not directly exposed but provided through proprietary device dedicated applications. While the PoC is already granted, further work will be carried out in order to identify the applicative perimeter of such a Medolution component.

[12] ihealthlabs.com

5.4. Data connector components

Data to be analysed could also be extracted from data warehouses like legacy databases, and also directly from the existing Health Information Systems (HIS) that are currently running within healthcare institutes, like hospitals, GP offices. In this case it will be useful to benefit from components which encapsulate the reusable data extraction code, and that will be easy to connect to Big Data storage or analytics processing components. This section presents a data connector component to extract data from relational databases, and a connector to access data from health information systems via HL7 standard interfaces.

5.4.1. Sqoop

Sqoop is a tool for efficiently transferring data from relational database management system (RDBMS) to Hadoop. Sqoop is designed as client-server. Sqoop implementation in BDCF is part of MapR distribution. Both client and server Alien4Cloud components are respectively named MapRSqoop2Client and MapRSqoop2Server.

5.4.1.1. MapRSqoop2Server API

Server side is reachable via a REST API. Full documentation is available here: https://sqoop.apache.org/docs/1.99.3/RESTAPI.html.

TOSCA component description

Properties

MapRSqoop2Server does not have any properties.

Requirements

MapRSqoop2Server requires to be hosted on MaprWarden.

Capabilities

sqoop2_server_endpoint*: allows connecting any Alien4Cloud component to this one. It can be used for MapRHUE requirement.*

Artefacts

tools*: MapR utility scripts.*

utils_scripts*: Common utility scripts for whole BDCF components.*

5.4.1.2. MapRSqoop2Client API

Client side provide a Java API which facilitates Sqoop integration in Java applications.

TOSCA component description

Properties

MapRSqoop2Client does not have any properties.

Requirements

MapRSqoop2Client requires to be hosted on MaprWarden.

Capabilities

sqoop2_client_endpoint*: allows connecting any Alien4Cloud component to this one.*

Artefacts

tools*: MapR utility scripts.*

utils_scripts*: Common utility scripts for whole BDCF components.*

5.4.2. EHR Data Connector

A vast majority of health data created within healthcare institutes like hospitals, GP offices, laboratories are locked up in the proprietary databases of the health information systems being used in these institutes like Hospital Information systems, EHR Repositories, Laboratory Information Systems, Radiology Information systems etc. Although these are stored mostly in proprietary formats, standard based interfaces have started to be largely adopted by the HIS, to enable interoperability among these systems, to exchange and re-use health data between disparate systems and organisations. HL7 is one of the de-facto standards in healthcare industry to enable interoperability. Hence, in Medolution, HL7 standard will be utilized as the basis of HER Data connector API.

EHR Data connector component is designed as a RESTful API providing an interface for EHR data integration on BDCF. The API is acting as an upload service and accepts HL7 CDA and HL7 FHIR payloads for POST operation. The service's main purpose is to transport created or given HL7 FHIR Bundles to the Big Data Platform entry point. Currently, Kafka is used for that purpose; the bundles are sent to a Kafka topic. A specialized FHIR Repository based on the MongoDB component of BDCF can be used for future integration.

5.4.2.1. HL7 Clinical Document Architecture (CDA)

Clinical Document Architecture (CDA), previously called Patient Record Architecture (PRA), is a document markup standard that specifies the structure and semantics of a clinical document (such as a discharge summary or progress note) for the purpose of exchange[13]. It is developed by Health Level Seven (HL7)[14], which is a not-for-profit ANSI accredited Standards Developing Organization. Primary goal of HL7 is to provide standards for the exchange of clinical and administrative data among healthcare systems.

A clinical document includes clinical observations and services about care events. A valid CDA document is encoded in XML and conforms to the CDA XML Schema Definition (XSD), once any possible user-specific extensions are removed. HL7 has released two versions of CDA so far. Release 1 (R1) was approved by ANSI in 2000, and Release 2 (R2) in 2005.

[13] http://www.hl7.org/implement/standards/cda.cfm
[14] http://www.hl7.org

A CDA R2 document has two main parts, the header and the body. The CDA header defines the context of the document by providing information on authentication, the encounter, the patient, and the involved providers whereas the CDA body includes the clinical report. The body part can be either an unstructured blob or a structured hierarchy that involves one or more section components. Within a section, narrative blocks and CDA entries are defined. Machine-processable clinical statements are represented by these CDA entries whereas the narrative blocks are human readable forms of these clinical statements.

CDA has nine entry classes derived from the HL7 v3 Reference Information Model (RIM[15]): Act, Observation, Observation-Media, SubstanceAdministration, Supply, Procedure, RegionOfInterest, Encounter and Organizer. These entry classes provide a consistent representation of clinical statements across various HL7 v3 specifications. For example, Observation is used for representing clinical observations, SubstanceAdministration is used for representing medication related events, Organizer is used for grouping clinical statements having a common context and Act, as a generic purpose class, is used when the remaining specific entry classes are not appropriate for defining the clinical information. According to their purpose of use, these entry classes enable all the fields to provide information about a clinical statement in a structured and/or coded manner; e.g. SubstanceAdministration has attributes for providing the medicine product, the route of administration, dose, rate and timing of quantity in a structured way.

HL7 Consolidated CDA (C-CDA)[16]

The Consolidated Templated implementation guide contains a library of CDA templates, incorporating and harmonizing previous efforts from Health Level Seven (HL7), integrating the Healthcare Enterprise (IHE), and Health Information Technology Standards Panel (HITSP). It represents harmonization of the HL7 Health Story guides, HITSP C32, related components of IHE Patient Care Coordination (IHE PCC), and Continuity of Care (CCD).

C-CDA implementation guide, in conjunction with HL7 CDA R2 standard, is used for implementing a number of different CDA documents including Continuity of Care Document (CCD), which is used in EHR Data Connector of Medolution.

5.4.2.2. HL7 Fast Healthcare Interoperability Resources (FHIR) [17]

FHIR is a next generation standards framework created by HL7. FHIR combines the best features of HL7's Version 2, Version 3 and CDA product lines while leveraging the latest web standards and applying a tight focus on implementability.

The basic building block in FHIR standard is a **"**Resource**"** which is smallest unit of exchangeable content between healthcare systems. FHIR standard defines several common resources so that different healthcare system implementers can use them to represent their data in their use cases. For example, Observation[18], Medication[19], CarePlan[20] are some clinical resource definitions, while Patient[21], Practitioner[22], Organization[23] are defined for identification of entities, and Encounter[24],

[15] http://www.hl7.org/implement/standards/rim.cfm
[16] http://www.hl7.org/implement/standards/product_brief.cfm?product_id=258
[17] https://www.hl7.org/fhir/
[18] https://www.hl7.org/fhir/observation.html
[19] https://www.hl7.org/fhir/medication.html
[20] https://www.hl7.org/fhir/careplan.html
[21] https://www.hl7.org/fhir/patient.html
[22] https://www.hl7.org/fhir/practitioner.html
[23] https://www.hl7.org/fhir/organization.html

Appointment[25], Order[26] are resource definitions for workflow management. Each resource can be used in different use cases. Below is the set of Resources in FHIR.

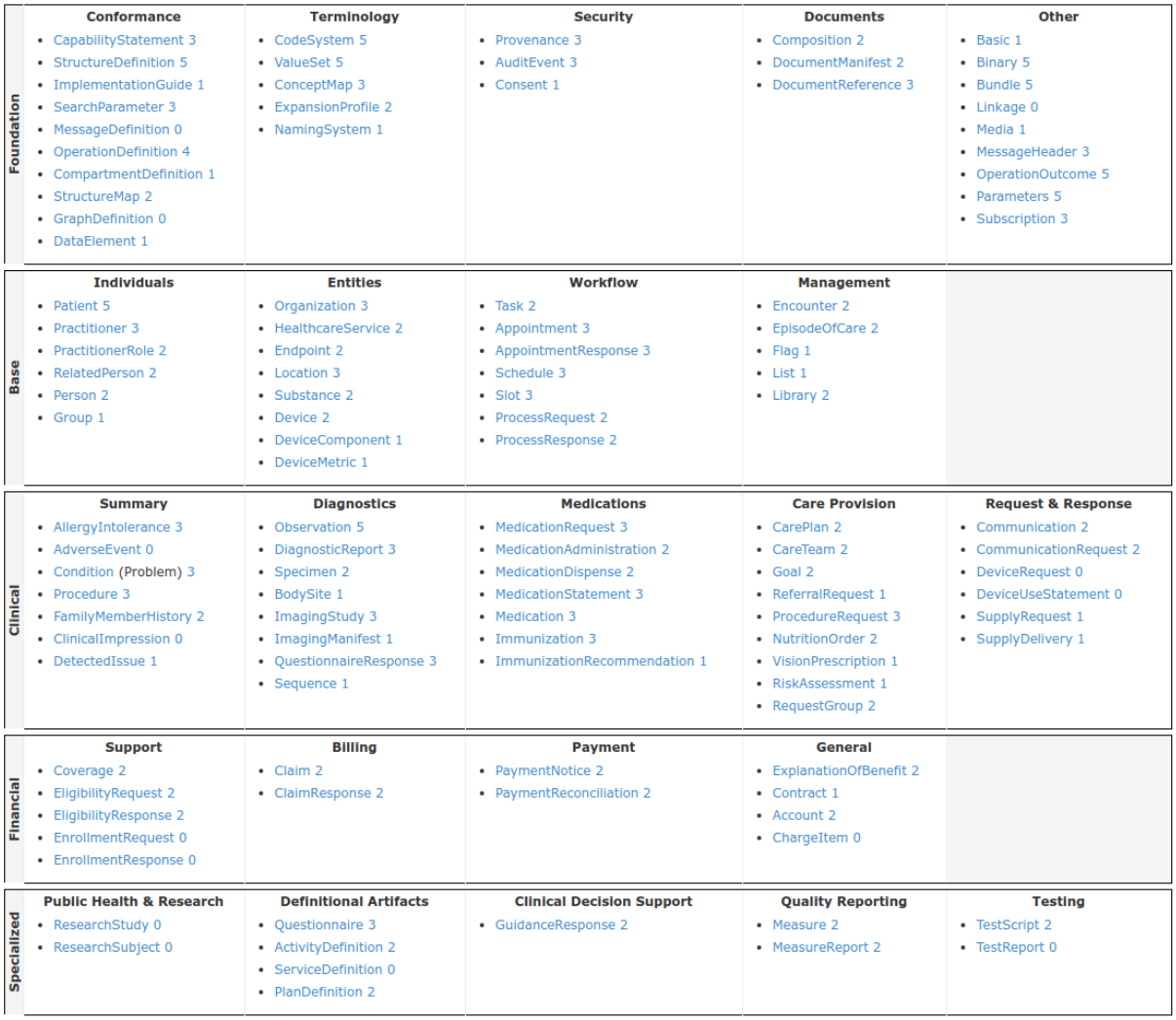

Figure 12: FHIR Resources

Bundle

Typically one CCD document corresponds to multiple FHIR resources. To achieve the mapping, FHIR Bundle resource, which is a container for a collection of resources, is used in EHR Data Connector of Medolution.

[24] https://www.hl7.org/fhir/encounter.html
[25] https://www.hl7.org/fhir/appointment.html
[26] https://www.hl7.org/fhir/order.html

5.4.2.3. CDA Payload

The service accepts HL7 Consolidated CDA (C-CDA) 2.1 Continuity of Care Document (CCD) instances encoded in XML, creates the corresponding HL7 Fast Healthcare Interoperability Resources (FHIR) Transaction Bundle JSON, and then sends the Bundle to the configured Kafka topic to be consumed by Big Data components on BDCF.

CDA Payload	
Allowed HTTP methods	POST
Path including path parameters	/cda
Query parameters	Not applicable
POST data	C-CDA CCD XML document
Success POST response	200 ok empty
Error responses	400 Bad Request - if xml document is not in CDA structure. 500 Internal Server Error - in case of internal exception

5.4.2.4. FHIR Payload

The service also accepts HL7 FHIR Bundle JSON Strings and forwards them to the configured Kafka Topic to be consumed by Big Data components on BDCF.

FHIR Payload	
Allowed HTTP methods	POST
Path including path parameters	/fhir
Query parameters	Not applicable
POST data	FHIR Bundle resource JSON
Success POST response	200 ok empty
Error responses	400 Bad Request – if faulty JSON. 500 Internal Server Error - in case of internal exception

5.4.2.5. TOSCA Component Description

***Component Name*: EHRDataConnector**

Properties

port: *The port of the API*

Type: Integer

Constraints: [1, 65535]

Default: 8081

base-uri: *Base uri of the API. (CDA payload complete Uri is -> [ip] : [port] / [base-uri] / cda)*

Type: String

Default: "ehr"

Requirements

EHRDataConnector component requires to be hosted on Java.

EHRDataConnector component requires connecting to a Kafka Topic.

EHRDataConnector component requires connecting to a Consul agent.

Capabilities

kafka_output*: allows connecting to a Kafka topic.*

consul: *allows connecting to a Consul agent*

Artifacts

scripts: *Lifecycle scripts of the EHRDataConnector component*

utils_scripts: *Common utility scripts for whole BDCF components*

consul_utils: *Common utility scripts for Consul operations*

runnable: *Executable jar file of the RESTful web service*

5.5. Data analytics components

The following subsections describe the BDCF components which allow application developers to analyse the data which has been gathered and stored using the components from the previous sections.

5.5.1. Rstudio

RStudio Server is a web based integrated development environment (IDE) for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging, and also workspace management.

5.5.1.1. Rstudio API

Rstudio has been developed to provide an integrated development environment as Web UI to allow end user to easily design an R application.

TOSCA component description

Properties

proxy_to_use*: Setup a proxy configuration in Renviron.site in order to allow downloading remote packages. If a value is set for this property then it will be used as http and https proxy (it should honour the unix http_proxy env var format). If not set or set to an empty string then the default environment proxy settings on the compute will be used (http_proxy, https_proxy and no_proxy). This is the default. If set to 'None' then proxies are not configured at all.*

default : ""

cran_mirror_to_use*: Mirror for R packages downloads to use. This allows setup and using your own mirror.*

default : "http://cran.r-project.org"

user_to_create*: Name of the unix account to be created and used to connect to RStudio*

default : "rstudio"

password_to_create*: Password of the unix account to be created and used to connect to RStudio*

default : "rstudio"

repository*: This property give the opportunity to specify an alternative download repository for this component artifacts. It is your responsibility to provide an accessible download url and to store required artifacts on it. You should specify only the base repository url. Artifacts names will be appended to it, so this property could be shared among several components using the inputs feature.*

default : ""

Requirements

host*: RStudio server component has to be hosted on a Compute.*

Attributes

url*: This attribute contains the URL used to access the RStudio UI. This attribute is only valid if the Compute on which RStudio is hosted is connected to a public network.*

5.5.2. Python

Python is a programming language commonly used by Data Scientists to develop data analysis algorithms. The Python component installs Anaconda and optionally some Python libraries. Anaconda is both a package manager, an environment manager and a Python distribution (Cf. https://docs.continuum.io).

A Python node is hosted on a Compute node. The components that need Python are hosted on the Python node.

TOSCA component description

Properties

repository*: Allows to specify an alternative download repository for Anaconda and the additionnal packages. The repository must contain the Anaconda install script (.sh) and a tar file containing librairies packaged as tar.bz2 (anaconda, conda, conda-env, configparser, csvkit, dbf, oauthlib, plotly, pybrain, requests-oauthlib, seaborn, twython).*

nlp_twitter (checkbox)*: If checked, installs additional packages for Natural Language Processing and twitter API (nltk, twython).*

dataviz (checkbox)*: If checked, installs additional packages for data visualization (seaborn, plotly).*

dataformat (checkbox)*: If checked, installs additional packages for data formatting (csvkit, configparser).*

pybrain (checkbox)*: If checked, installs Python-based reinforcement learning, artificial intelligence and neural network library (pybrain).*

Capabilities

Python_host*: Python is the component hosting Jupyter component*

Artefacts

scripts*: Python required scripts.*

utils_scripts*: Common util scripts for whole BDCF components.*

5.5.3. Jupyter

The Jupyter component installs the Jupyter notebook, a Web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text. Uses include data cleaning and transformation, numerical simulation, statistical modeling, machine learning and much more (Cf. http://jupyter.org/).

A Jupyter node is hosted on a Python node.

TOSCA component description

Properties

repository*: Alternative download repository for Anaconda and the additionnal packages.*

irkernel (checkbox)*: If checked, installs the iRKernel, a Jupyter kernel for the R language (https://github.com/IRkernel/IRkernel, https://irkernel.github.io).*

py35kernel (checkbox)*: If checked, installs the Jupyter kernel for Python v3.5.*

spark-kernel (checkbox)*: If checked, installs the Jupyter kernel that can interact with Apache Spark and allows you to develop in Scala language.*

Requirements

python*: Jupyter requires to be hosted on python component.*

filesystem_endpoint*: Jupyter may be connected to a filesystem in order to store its runtime data on it.*

Artefacts

scripts*: Jupyter required scripts.*

utils_scripts*: Common util scripts for whole BDCF components.*

data_file*: Jupyter additional scripts.*

5.5.4. NIFI

Nifi is a technical component allowing data acquisition, simple event processing, transport and delivery mechanism designed to accommodate the diverse data flows generated by connected systems (Databases, Sensors, Data Lakes, Data Platforms).

Nifi is used for data ingestion to pull data into NiFi, from numerous different data sources and create FlowFiles using a process group. A Process Group is a specific set of processes and their connections, which can receive data via input ports and send data out via output ports. In this manner, process groups allow creation of entirely new components simply by composition of other components. Processes are implemented by processors; Nifi includes almost 180 predefined processors for different kinds of treatments the user may customize.

Nifi executes within a JVM on a host operating system. The primary components of Nifi on the JVM are as follows:

- Web Server: The purpose of the web server is to host NiFi's HTTP-based command and control API.
- Flow Controller: The flow controller is the brain of the operation. It provides threads for extensions to run on, and manages the schedule of when extensions receive resources to execute.
- Extensions: There are various types of Nifi extensions that can be added.

- FlowFile Repository: The FlowFile Repository is where NiFi keeps track of the state of what it knows about a given FlowFile that is presently active in the flow.
- Content Repository: The Content Repository is where the actual content bytes of a given FlowFile live.
- Provenance Repository: The Provenance Repository is where all provenance event data is stored.

Figure 13 shows a Nifi template example, where processors are linked together.

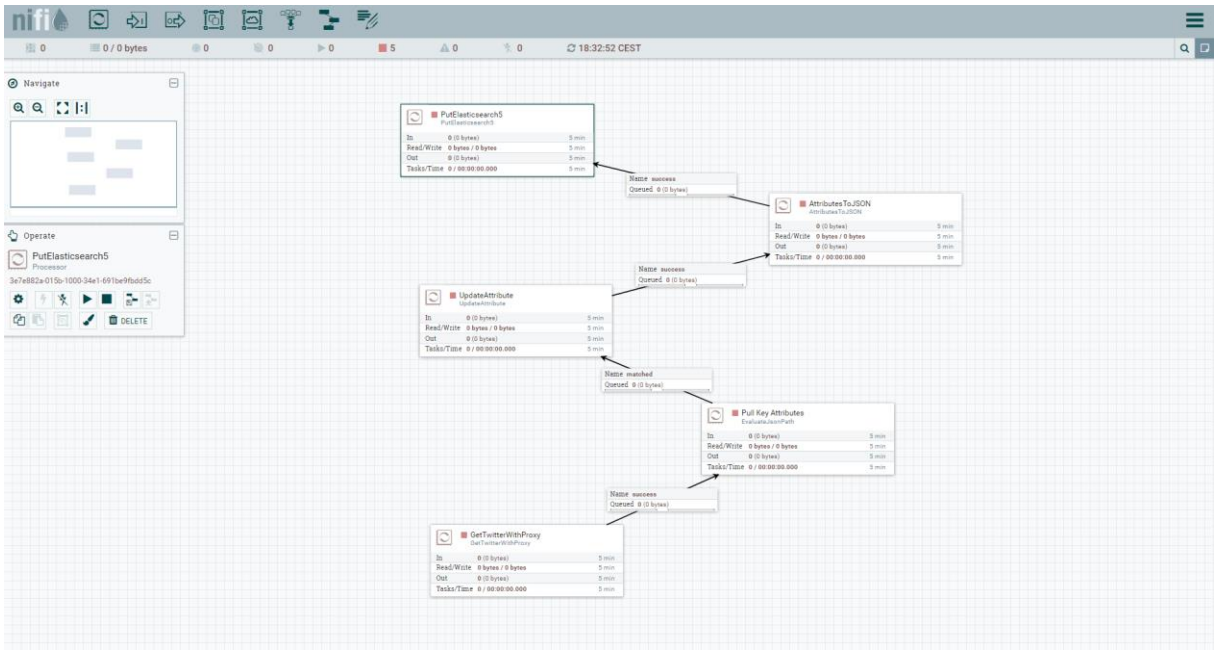

Figure 13: Nifi Template Example

The above template collects tweets from twitter API, extracts some of their fields, formats the date, parses the results into a JSON format and ingests them into Elasticsearch.

TOSCA component description

Properties

repository*: Specifies an alternative download repository for Nifi binary*

installation_directory*: Allows you to choose where to install Nifi binaries*

– Default: ~/nifi.

Requirements

java: *Nifi node must be hosted on a Java node, which is hosted itself on a compute node. The minimum version of Java is JRE8*

Artefacts

utils_scripts*: Common util scripts for whole BDCF components.*

Figure 14 shows Nifi node configuration within BDCF GUI.

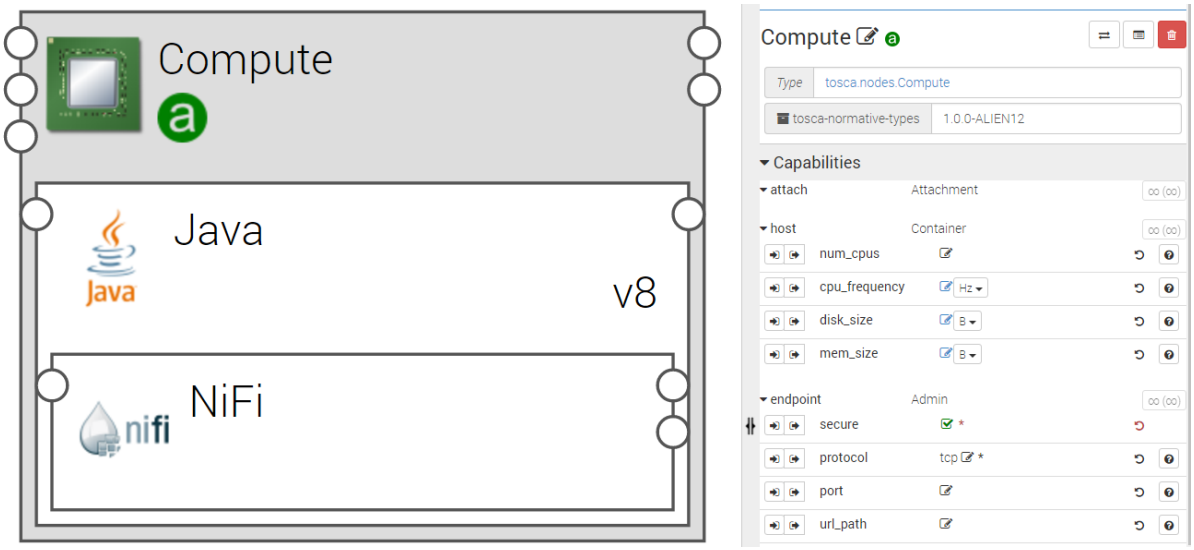

Figure 14: Nifi Node Configuration

Figure 15 below shows a typical BDCF Topology used to deploy the use case of tweets analysis of Figure 13. It deploys Nifi on one node and ElasticSearch and Kibana on another node. Tweets are filtered and formatted in Nifi and then stored in ElsaticSearch and displayed in Kibana.

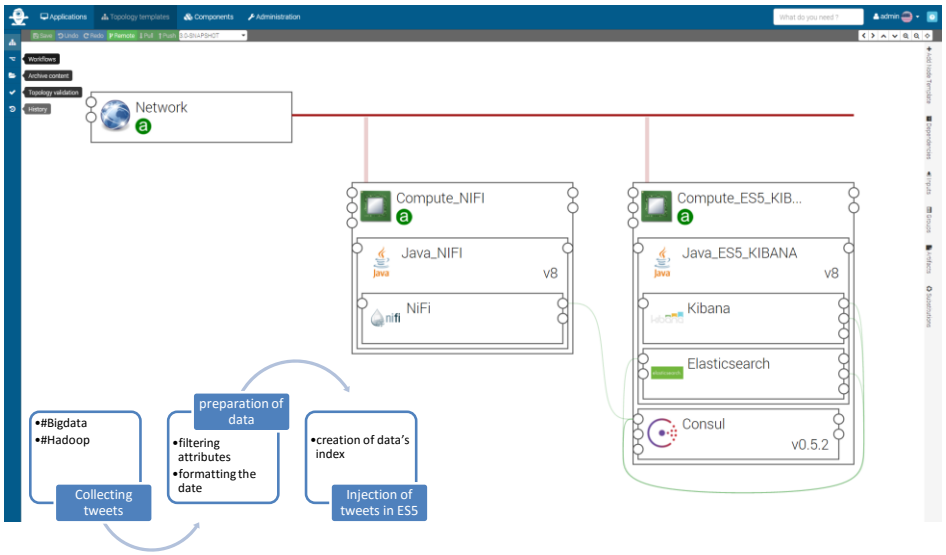

Figure 15: Nifi BDCF Topology Example

5.5.5. Flink

Flink_JobManager and Flink_TaskManager are the BDCF components implementing the Apache Flink software (https://flink.apache.org), an open source platform for distributed stream and batch data processing. Flink is a streaming dataflow engine that provides data distribution, communication, and fault tolerance for distributed computations over data streams. It is typically

used to implement a Kappa architecture as described in section 6.2, to support mixed batch and real time streaming processing.

5.5.5.1. Flink API

Flink includes several APIs for creating applications that use the Flink engine:

- **DataStream API** for unbounded streams embedded in Java and Scala
- **DataSet API** for static data embedded in Java, Scala, and Python
- **Table API** with a SQL-like expression language embedded in Java and Scala.

Flink also bundles libraries for domain-specific use cases:

- **CEP**, a complex event processing library
- **Machine Learning** library
- **Gelly**, a graph processing API and library.

For more details on Flink concepts, see https://ci.apache.org/projects/flink/flink-docs-release-1.1/concepts/concepts.html

Figure 16 shows a Flink application with one Jobmanager and two TaskManagers linked to it. One of the TaskManager is hosted on a scalable compute so the number of instances of TaskManager can be increased or decreased at runtime.

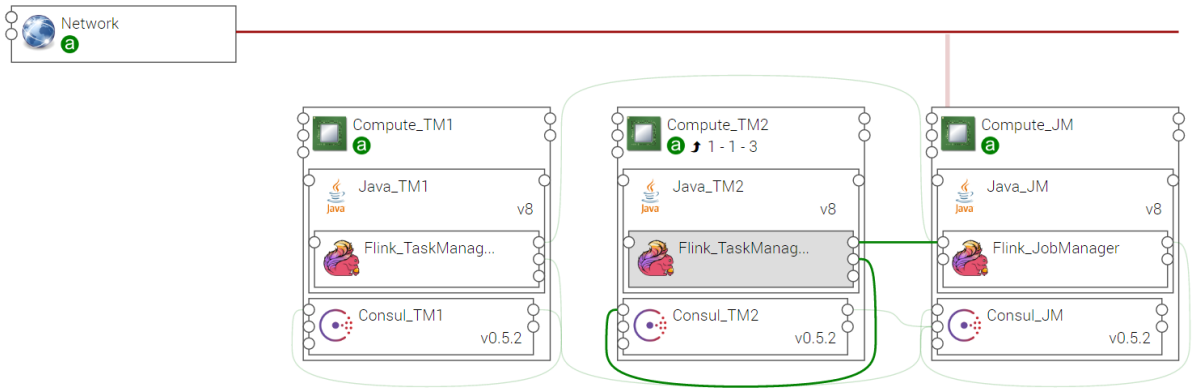

Figure 16: Flink Topology Example

5.5.5.2. Flink JobManager

Flink_JobManager is the master process, which coordinates the distributed execution. The Job Managers schedule tasks, coordinate checkpoints, coordinate recovery on failures, etc. At least one Job Manager is required.

TOSCA component description

Properties

***jobmanager.heap.mb**: JVM heap size (in megabytes) for the JobManager.*

– Default: 512

***jobmanager.rpc.port**: Port used by JobManager to communicate with TaskManager (not editable).*

***repository:** Alternative download repository for the Apache Flink binary.*

***component_version**: Version of the component (not editable).*

– Default: 1.1.3

Attributes

***url**: Python is the component hosting Jupyter component*

Capabilities

***jobmanager**: Capabilities offered to TaskManager(s) to connect to this JobManager*

Requirements

***java**: A Flink_JobManager node must be hosted on a Java node, which is hosted itself on a compute node*

***consul**: A Flink_JobManager node must be related to a Consul agent hosted on its Compute node*

5.5.5.3. Flink TaskManager

Flink_TaskManager is the worker process which executes the tasks (or more specifically, the subtasks) of a dataflow, and buffers and exchanges the data streams. At least one Task Manager is required.

TOSCA component description

Properties

***taskmanager.heap.mb**: JVM heap size (in megabytes) for the TaskManager.*

– Default: 512

***taskmanager.numberOfTaskSlots**: Number of parallel operator or user function instances that a single TaskManager can run. This value is typically proportional to the number of physical CPU cores that the TaskManager machine has (for example, equal to the number of cores, or half the number of cores).*

– Default: 1

***parallelism.default**: Default parallelism to use for programs that have no parallelism specified. For setups that have no concurrent jobs running, setting this value to NumTaskManagers * NumSlotsPerTaskManager will cause the system to use all available execution resources for the program execution.*

– Default: 1

***taskmanager.tmp.dirs**: List of directories into which Flink writes temporary files when data do not fit into main memory. The directory paths must be separated by ':' (colon character).*

– Default: /tmp

***repository**: Alternative download repository for the Apache Flink binary.*

***component_version**: Version of the component (not editable).*

– Default: 1.1.3

Requirements

***jobmanager_endpoint**: A Flink_TaskManager must be connected to a Flink_JobManager*

***java**: A Flink_TaskManager node must be hosted on a Java node, which is hosted itself on a compute node. The minimum version of Java is JRE6.*

***consul**: A Flink_TaskManager node must be related to a Consul agent hosted on its Compute node*

5.5.6. Spark

Spark is a fast and general-purpose cluster computing system which aims to analyse a large amount of data using different development languages like Java, Scala, Python and R. It provides some frameworks for machine learning, streaming and graph processing. Command line interface is the only way to execute Spark Jobs.

5.5.6.1. Spark API

BDCF offers three ways to launch Spark jobs:

- Standalone: jobs are started on compute Spark is installed. It is an easy way to test scripts at development step. The command is: spark-submit [spark-options] <path/to/spark/job>
- YARN client mode: jobs placements are managed by YARN resource manager (see section 5.7.2). Unlike YARN cluster mode, Spark driver is executed on the compute Spark is installed which gives an easy access to outputs. It uses all cluster resources, which allows launching more consumer jobs. The command is: spark-submit –master yarn-client [spark-options] <path/to/job/script>
- YARN cluster mode: all parts of Spark jobs are managed by YARN. Even Spark driver is launched on a YARN container. It is the recommended way to execute Spark jobs in production environment. The command is: spark-submit –master yarn-cluster [spark-options] <path/to/job/script>

TOSCA component description

Spark implementation in BDCF is part of MapR distribution. The component name in BDCF catalogue is MapRSpark.

Properties

MapRSpark does not have any properties.

Requirements

MapRSpark requires to be hosted on MapRWarden.

Capabilities

spark_endpoint*: allows connecting any Alien4Cloud component to this one.*

Artefacts

tools*: MapR utility scripts.*

utils_scripts*: Common utility scripts for whole BDCF components.*

5.5.7. Pig

Apache Pig is a high-level development framework which allows creating MapReduce jobs to analyse large amounts of data. It provides parallelization mechanisms to improve performances. Jobs are written using Pig Latin, a dedicated development language to facilitate high level comprehension.

5.5.7.1. Pig API

The only way to launch pig jobs is to use its command line interface: pig <path/to/pig/job>

TOSCA component description

Pig implementation in BDCF is part of MapR distribution. The component name in BDCF catalogue is MapRPig.

Properties

MapRPig does not have any properties.

Requirements

MapRPig requires to be hosted on MapRWarden.

Capabilities

pig_endpoint*: allows connecting any Alien4Cloud component to this one. It can be used for MapRHUE requirement.*

Artefacts

tools*: MapR utility scripts.*

***utils_scripts**: Common utility scripts for whole BDCF components.*

5.5.8. Mahout

The aim of Apache Mahout project is to provide implementations of machine learning algorithms.

5.5.8.1. Mahout API

Mahout algorithms are designed to be implemented in Java. Please refer to the official documentation (https://mahout.apache.org/) for more information.

TOSCA component description

Mahout implementation in BDCF is part of MapR distribution. The component name in BDCF catalogue is MapRMahout.

Properties

MapRMahout does not have any properties.

Requirements

MapRMahout requires to be hosted on MaprWarden.

Capabilities

***mahout_endpoint**: allows connecting any Alien4Cloud component to this one.*

Artefacts

***tools**: MapR utility scripts.*

***utils_scripts**: Common utility scripts for whole BDCF components.*

5.6. Visualization components

This section describes the components used to visualize the results of data analysis performed by components previously described.

5.6.1. Kibana

Kibana is an analytics and visualization platform that uses Elasticsearch to give you a better understanding of your data. It is a part of the Elastic Stack, which includes Elasticsearch, Logstash, Kibana.

A complete documentation is available at https://www.elastic.co/guide/en/kibana/current/index.html.

5.6.1.1. Kibana Interface

The Kibana interface includes four main sections:

- Discover
- Visualize
- Dashboard
- Settings

These sections are reached by tabs on the top of the main interface.

- **Kibana Discover:** The Discover page displays all of the Elastic Stack most recently received logs. Here, you can filter through and find specific log messages based on search queries then narrow the search results to a specific time range with the time filter.
- **Kibana Visualize:** The Visualize page is where you will create, modify, and view your own custom visualizations.
- **Kibana Dashboard:** The Dashboard page is where you can create, modify, and view your own custom dashboards. A dashboard allows you to combine multiple visualizations onto a single page. Dashboards are useful to get an overview of your logs, and to make correlations among various visualizations and logs. Dashboards can be refreshed automatically with a configurable period. They can be filtered by entering a search query, changing the time filter, or by clicking on the elements on the visualization.
- **Kibana Settings:** The Settings page lets you change various parameters like default values or index patterns.

5.6.1.2. Kibana Configuration

Basic configuration

A Kibana node must be hosted on a Java node, which is hosted on a Compute node. Kibana Dashboard nodes can be attached to the Kibana node using dashboard_host prerequisite.

A Kibana node requires to be related to an Elasticsearch node. Use the search_endpoint prerequisite to establish the relation with an already created Elasticsearch node.

A Kibana node is related to a Consul agent hosted on its Compute node. This is required to perform Elasticsearch cluster discovery. Use consul prerequisite to connect Kibana to its Consul Agent.

Kibana Properties

- repository: This property allows you to specify an alternative download repository for the Kibana binary.
    - Default : “”
- es_heap_size: This property allows setting the heap memory that will be allocated to the java process of the Elasticsearch client node associated to Kibana. It allocates the same value to both initial and maximum values (ie -Xms and -Xmx java options).
    - Default : “1G”

Kibana Capabilities

The Kibana component can be used as a dashboard_host by Dashboard components. The role of a Dashboard component is to carry a dashboard configuration (it has an artifact named

dashboard_file). This configuration is described in a .JSON file. Several Dashboard components can be connected to a Kibana component by using their dashboard_host prerequisite.

Kibana URL

Kibana URL is an address generated at the end of the topology deployment.

5.7. Data distribution & Resource management components

Performing Big Data analysis requires complex distributed computing configuration, managing data distribution and processing distribution. The components described in this section allow managing this distribution.

5.7.1. MapR Filesystem

MapR distribution has designed its own distributed filesystem dedicated to Big Data applications. It is an improved Hadoop filesystem with better performances, reliability, efficiency, maintainability and ease of use.

MapR filesystem management in BDCF is split into two different components: MapRCldb and MapRFileserver.

5.7.1.1. MapRCldb

The Container Location Database (CLDB) service is the equivalent of Hadoop Namenode. It tracks information about every container in MapR filesystem (location, size, volume, policies, quotas, usages, etc...). It also tracks file servers and node activities in the cluster.

Running the CLDB service on multiple nodes distributes lookup operations across those nodes for load balancing and also provides high availability.

MapR distribution is based on a licensing model. When a cluster is deployed, a MapR Base Edition license is registered which doesn't provide all functionalities (e.g. the NFS gateway can't be instantiated). The MapR Community Edition provides all functionalities with unlimited access for free. The MapR Enterprise Edition is dedicated to production environments, with high availability capabilities. For a full description of the MapR software editions, follow this link: https://www.mapr.com/products/mapr-distribution-editions.

TOSCA component description

Properties

replication*: You can set the replication level of your cluster. Replication is the number of times your data are copied in your cluster to provide data safety.*

Default value: 2.

Requirements

MapRCLDB requires to be hosted on MaprWarden.

MapRCLDB requires MapRFileServer to be installed on the same node.

Capabilities

***cldb_endpoint**: allows connecting any Alien4Cloud component to this one.*

Custom command

apply_license (license_url)

***license_url**: The URL of the new MapR license file to apply.*

Artefacts

***licenseFile**: The MapR license file to apply on the cluster at deployment step.*

***tools**: MapR utility scripts.*

***utils_scripts**: Common utility scripts for whole BDCF components.*

5.7.1.2. MapRFileserver

The MapRFileServer component is the access point to MapR filesystem. When an application wants to access to a file into the filesystem, it first communicates with CLDB and then with MapR fileserver, which is the equivalent of Hadoop DataNode.

TOSCA component description

Properties

***storage_pool_size**: Allow to customize the number of disks in a storage pool (see Mapr doc for more details)*

Default: 3 (Mapr default value)

Requirements

MapRFileServer requires to be hosted on MaprWarden.

***volume**: it requires to be connected to a block storage (30GB minimum) in order to format it as a MapR filesystem storage.*

***host**: it requires to be hosted on a MapR Warden component.*

Capabilities

***fs_endpoint**: allows connecting any Alien4Cloud component to this one.*

Artefacts

***tools**: MapR utility scripts.*

***utils_scripts**: Common utility scripts for whole BDCF components.*

5.7.2. YARN

YARN is the main Big Data application resource manager. It embeds the most recent MapReduce algorithms version (MR v2.0) and allows launching other types of applications. YARN is made up of two different components: YARN Resource manager and YARN Node manager.

YARN API

Both Resource and node managers are available through a Web UI and an RPC API. This allows managing and monitoring job executions.

5.7.2.1. YARN Resource manager

The MapRYarnRM component is the YARN Resource Manager of Hadoop. Using YARN, Big data applications run inside containers which are instantiated on YARN Node Manager. The Resource Manager is big data YARN application endpoint. It manages all resources available on the cluster and schedules deployment of applications on containers.

TOSCA component description

Properties

***yarn.scheduler.minimum-allocation-vcores**: Defines the minimum number of vcores allocated to a container.*

default: 1

***yarn.scheduler.maximum-allocation-vcores**: Defines the maximum number of vcores allocated to a container.*

default: 32

***yarn.scheduler.minimum-allocation-mb**: Defines the minimum memory allocation available for a container in Mb.*

default: 1024

***yarn.scheduler.maximum-allocation-mb**: Defines the maximum memory allocation available for a container in Mb*

default: 8192

Requirements

MapRYarnRM requires to be hosted on a MapRWarden.

MapRYarnRM component is required on MapR cluster to use ecosystem tools.

Capabilities

***rm_endpoint**: allows connecting any Alien4Cloud component to this one.*

Artefacts

***tools**: MapR utility scripts.*

***utils_scripts**: Common utility scripts for whole BDCF components.*

***yarn_files**: Required YARN template files.*

5.7.2.2. YARN Node manager

The MapRYarnNM component is the YARN Node Manager of Hadoop. It is managed by YARN Resource Manager and allows deploying containers which will execute big data applications.

TOSCA component description

Properties

***yarn.nodemanager.resource.cpu-vcores**: Defines the number of CPUs available to process YARN containers on this node.*

Default: Variable. This value is calculated by Warden.

***yarn.nodemanager.resource.memory-mb**: Defines the memory available to process Yarn containers on the node in Mb.*

Default: Variable. This value is calculated by Warden.

***yarn.nodemanager.resource.io-spindles**: Defines the number of disks available to process YARN containers.*

Default: Variable. This value is calculated by Warden.

***mapreduce.map.cpu.vcores**: Defines the number of CPUs available to process map operation.*

Default: 1

***mapreduce.map.memory.mb**: Defines the container size for map tasks in MB.*

Default: 1024

***mapreduce.map.java.opts**: Java options for map tasks.*

Default: -Xmx900m

***mapreduce.map.disk**: Defines the number of disks a map task requires.*

Default: 0.5

***mapreduce.reduce.cpu.vcores**: Defines the number of CPUs available to process reduce operation.*

Default: 1

***mapreduce.reduce.memory.mb**: Defines the container size for reduce tasks in Mb.*

Default: 3072

***mapreduce.reduce.java.opts**: Java options for reduce tasks.*

Default: -Xmx2560m

***mapreduce.reduce.disk**: Defines the number of disks that a reduce task requires.*

Default: 1.33

Requirements

MapRYarnNM requires to be hosted on MapRWarden.

A MapRYarnRM must be set on the cluster.

Capabilities

***nm_endpoint**: allows connecting any Alien4Cloud component to this one.*

Artefacts

***tools**: MapR utility scripts.*

***utils_scripts**: Common utility scripts for whole BDCF components.*

***yarn_files**: Required YARN template files.*

6. Main Medolution Big Data Topologies

This section describes typical Medolution components assemblies to satisfy Medolution application requirements. Big Data Capability Framework (BDCF) provides many Big Data components assemblies that cover some general use cases and can be re-used and/or enhanced inside the Medolution project. The figure below illustrates an assembly, also called topology, for an ElasticSearch, Logstash, and Kibana assembly, as it can be viewed in the BDCF GUI.

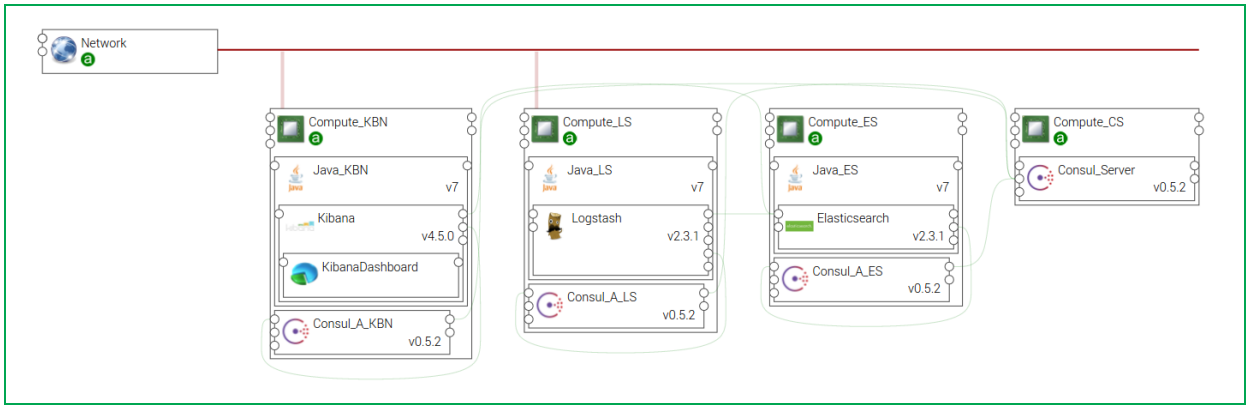

Figure 17: Smart Log Analytics assembly example

The complete list of available topologies is part of the BDCF user guide [4]; it contains topologies for Hadoop clusters configuration with MAPR or HortonWorks, topologies for MongDb, etc...

In the following subsections are described the Lambda Architecture topology, the Kappa Architecture topology, Event Sourcing and CQRS Architectures and the LVAD Use Case topology, which typically address identified Medolution requirements at the time being.

6.1. Lambda Architecture

The Lambda Architecture is generic, low latency, scalable and fault-tolerant data processing architecture. It aims to analyze large set of data both in a batch mode and in a real-time mode. It is composed of the following layers:

- The ingestion part is where data is coming usually in a continuous flow, and whose goal is to transfer it to both batch and speed layers. Scalability and retention capabilities of this layer are very important to cope with large amount of incoming data and to ensure that no data will be lost in case of failure.
- The batch layer is processing a huge set of data and performs very precise calculation on the global data set. This layer ensures persistency of the global data set and is usually set up on a distributed infrastructure.
- The speed layer is processing data which is directly coming from sources and performs live analysis. The idea is to minimize the latency to provide real time views on most recent data. Results are stored in the serving layer allowing data scientist to visualize and query these results.
- The serving layer is usually the place where results of the batch analysis and indexed to provide a fast and easy access. In the example below, it is also used by the speed layer.

- Visualization layer is the place where results are visualized.

BDCF provides a ready to use topology built from Alien4Cloud catalog components.

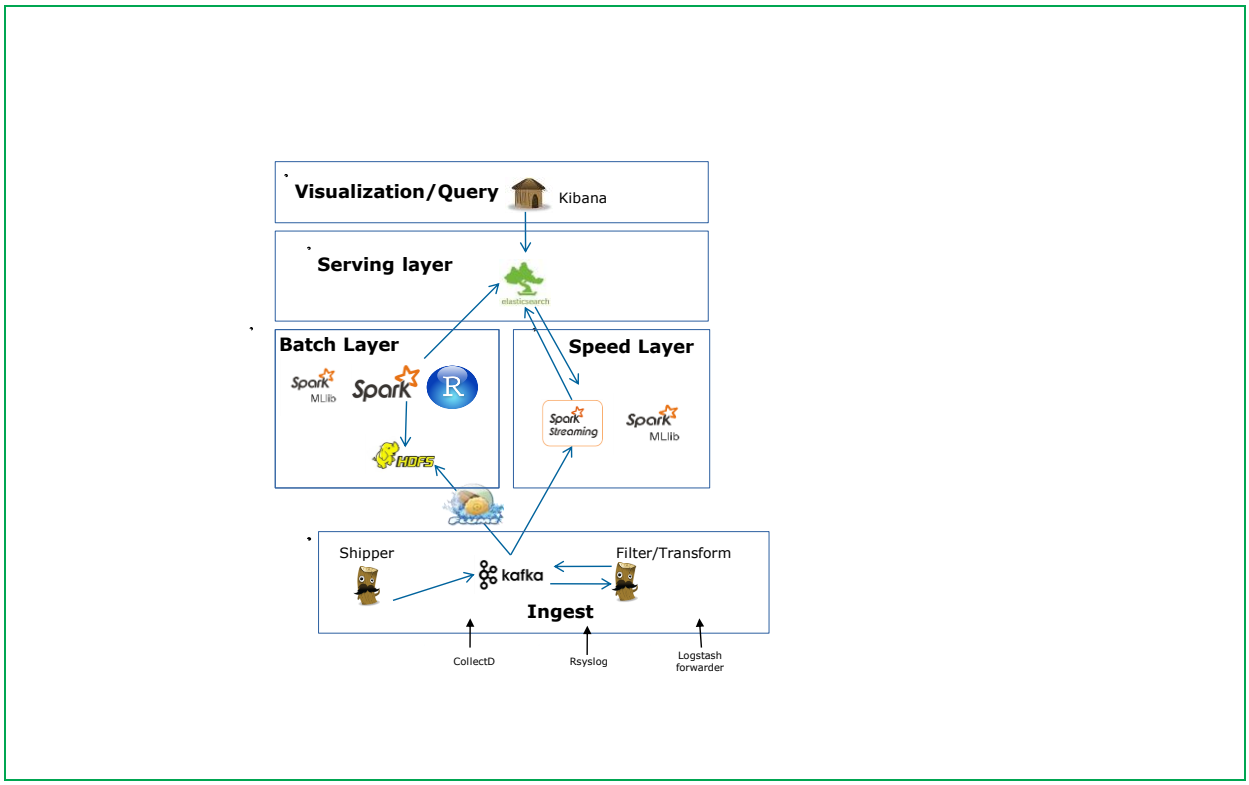

Figure 18: Lambda architecture

Figure 18 above illustrates this topology, the TOSCA topology diagram as visible in the BDCF GUI is shown below in Figure 19 (although not very readable this document, it can be zoomed in the GUI).

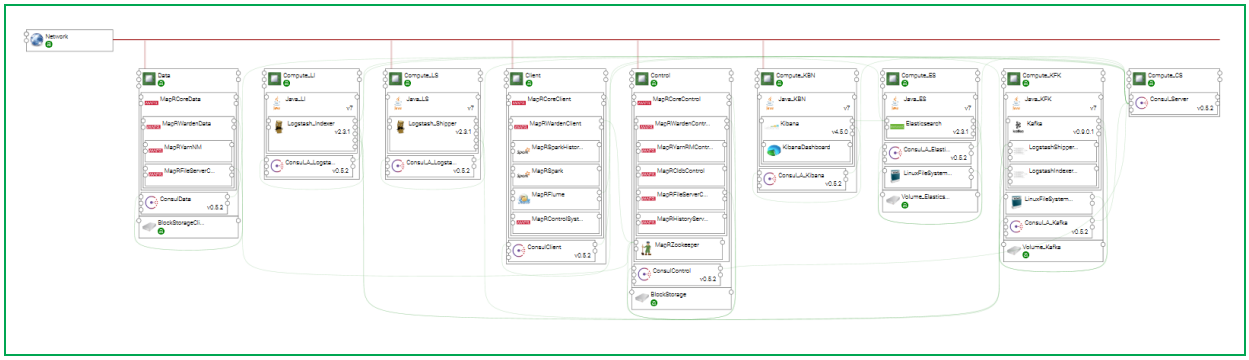

Figure 19: BDCF Lambda architecture representation

These components have been selected to match the requirements of each lambda architecture layer:

- **Batch layer**: The master data management & batch computation would be done on a MapR cluster. This cluster includes MapR Spark as computation framework.
- **Speed layer**: The streaming processing would be done on the same MapR cluster than batch layer.
- **Serving layer**: Elasticsearch is used to index results from batch and speed layer. These indexed results can be queried by Kibana.

The data ingestion in the processing architecture is made using a model similar to ELK-broker. After the data is consumed by Kafka coming from LogstashShipper, it can follow 3 streams:

- The data could follow the classic ELK-broker flow: LogstashIndexer, ElasticSearch and Kibana.
- The data could be acquired by a Spark streaming program running on MapR cluster.
- The data could be stored on MapRFS using MapRFlume for batch processing.

6.2. Kappa Architecture

The Kappa architecture simplifies the Lambda architecture by removing the batch layer and replacing it with a streaming layer. This simplification reduces the architecture to a single streaming engine capable of ingesting the needed volumes of data to handle both batch and real-time processing.

There are four main principles in the Kappa architecture:

1. Everything is a stream: Batch operations become a subset of streaming operations. Hence, everything can be treated as a stream.
2. Immutable data sources: Raw data (data source) is persisted and views are derived, but a state can always be recomputed as the initial record is never changed.
3. Single analytics framework: Keep it short and simple (KISS) principle. A single analytics engine is required. Code, maintenance, and upgrades are considerably reduced.
4. Replay functionality: Computations and results can evolve by replaying the historical data from a stream.

In order to respect principle four, the data pipeline must guarantee that events stay in order from generation to ingestion. This is critical to guarantee consistency of results, as this guarantees deterministic computation results. Running the same data twice through a computation must produce the same result. These four principles do, however, put constraints on building the analytics pipeline.

BDCF provides a ready to use topology built from its catalog components. These components were selected to match the requirements of Kappa Architecture:

1. The first component is a scalable, distributed messaging system with events ordering and at-least-once delivery guarantees. Kafka (Cf. 5.2.2) can connect the output of one process to the input of another via a publish subscribe mechanism. Using it, we can build something similar to the Unix pipe systems where the output produced by one command is the input to the next.
2. The second component is a scalable stream analytics engine. Flink (Cf. 5.5.5) is a streaming dataflow engine that provides data distribution, communication, and fault tolerance for distributed computations over data streams. One of its most interesting API features allows usage of the event timestamp to build time windows for computations.
3. The third and fourth components are a real time analytics store, Elasticsearch (Cf. 5.2.1), and a visualisation tool, Kibana (Cf. 5.6.1). Those two components are not critical, but they are useful to store and display raw data and results.

Figure 20 shows the resulting Kappa Topology Template provided by BDCF.

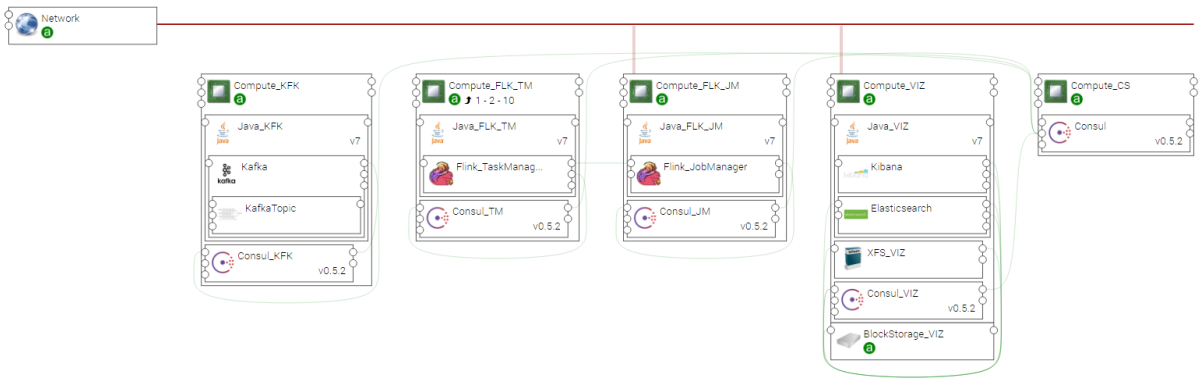

Figure 20: Kappa Topology

6.3. Event Sourcing and CQRS Architectures

Previously presented architectures deal both with Real Time Data Processing and Batch Data Processing. In addition to these types of data processing, Big Data applications often require a new constraint, which consists in keeping the complete history of data and a complete log of every state change. This is well answered by the he Event Sourcing pattern which defines an approach to handling operations on data that's driven by a sequence of events, each of which is recorded in an append-only store.

The CQRS, standing for **C**ommand **Q**uery **R**esponsibility **S**egregation, is an application architecture pattern most commonly used with event sourcing, which involves splitting an application into two parts:

- The **command** or **write** side, does not care about the queries
- On the other hand, the **query** or **read** side is all about the read access; its main purpose is making queries fast and efficient

Read and Write are logically and physically separated; can be scaled independently.

Such architecture is illustrated in Figure 21, where the “Write” part stores all the raw data, which may be used by Machine Learning or batch type Analytics, while the “Read” part is used to build views of data that may be used by real time applications providing real time visualisation, monitoring, alerting or analytics.

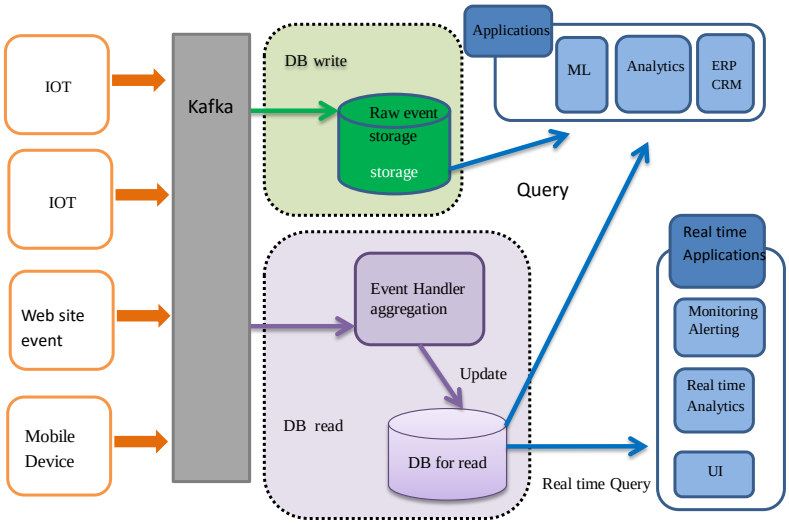

Figure 21: CQRS Architecture Example

The “Write” part is about raw event storage, history of data; NoSQL databases like MongoDB, Cassandra or Hbase may be used to implement it.

The “Read” part is about Data Preparation, where an Event Handler consumes events coming from Kafka, processes the aggregations and applies the resulting updates to a read store which reflects the kinds of queries run against it and contains the current state of data. Kafka Streams, Redis and MongoDB are candidates to implement this part.

Most of the building blocks are available in the BDCF catalogue to build a CQRS topology, although it has not yet been experimented.

6.4. Concrete architecture of the LVAD medical use case

The architecture below is a first draft that has been built to support the Medolution LVAD medical use case. This is typically a topology that will be supported in the Medolution platform, and corresponds to the kind of architecture that is straightforward to build with the BDCF integration PaaS. When the technical choices will be definitive, the corresponding components will be integrated in the Medolution platform catalogue (note that some of them are already there, like Kafka and R).

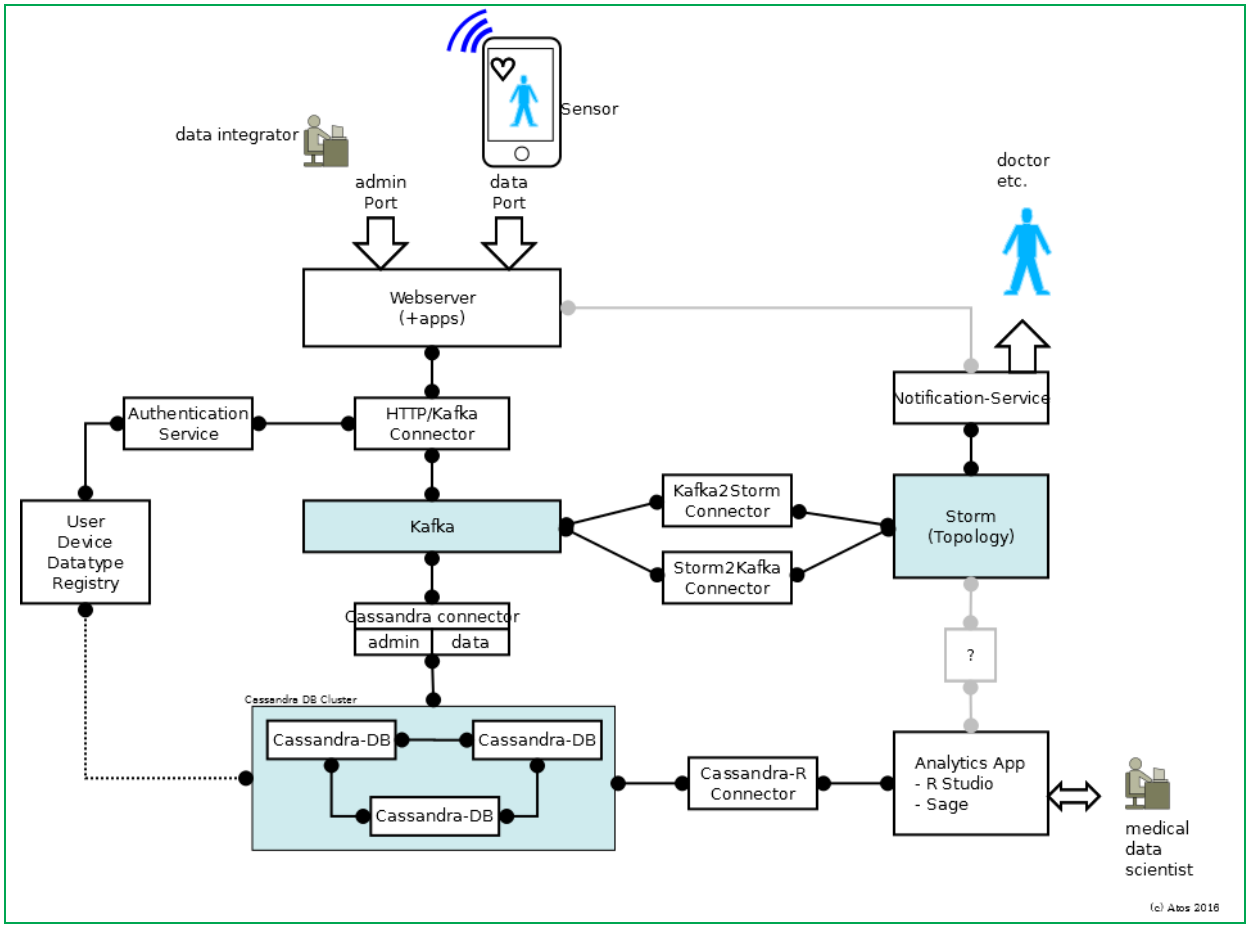

Figure 22: LVAD Medolution Cloud Scenario

In the LVAD use-case data from the patients LVAD devices and potential other sources is send to a data-analytics and store component which will be realized as virtualized services. The use-case supports two kinds of processing: Stream processing of the actual received data packages and batch-oriented analysis of historical data. The concrete architecture is a variant of the lambda architecture for data analytics but it has been extended to support the requirements of the Medolution platform. As depicted in Figure 22 the interface to the non-cloud components of Medolution (which is typically a local IoT platform or gateway) is a standard REST-based web interface where input data is received.

The processing in the cloud is designed as routing of messages to the respective processing elements. Therefore the data is validated and checked for the authorization. If it is identified as valid data from a valid source it is handed over to a Kafka component which serves as high-performance message queue. The architecture has been layered out so that arbitrary processing components and storage can be attached to Kafka by registering for target types of data. All data will be stored in a permanent database. In this scenario Cassandra is the database of choice, because it can be configured to create dependable and secured storage. For redundancy a minimum of three Cassandra instances is deployed to build a database cluster.

For near real-time analysis of data Storm is used to set up a processing network (topology). In this particular use-case there will be recognition nodes in Storms hub and spoke sub-architecture to detect patterns in the data-stream from LVAD devices which allow to detect or even to forecast critical conditions of the device. Such events will be sent to the notification service which will relay the message to a doctor or clinical personal.

In the more batch-oriented part standard tools like R studio, Sage or Jupyter can be used for processing. All three of them support a number of algorithms suitable for a data scientist to extract new findings from the set of received data.

The architecture is augmented by the required connector components which create the links between the elements. In these components also the semantic of the processing in captured because they contain the definitions which types of data are to be send where in the architecture. Additionally components are necessary to provide security and authorization mechanism for the connected devices. Therefore a registry of allowed communication endpoints (devices or gateways) and the associated data-types has to be maintained. With this a certificate or token based identification can be implemented.

The described architecture can be deployed in the cloud. In order to isolate the component and to be able to tune the performance and allow scalability the following deployment configuration is envisaged:

- One instance for Kafka and Kafka adapters
- For each Cassandra instance a separate instance (for dependability reasons the instances should be allocated on different hardware components in the cloud)
- One instance for the storm topology (This is the start configuration. If the Storm topology grows more instances are necessary to ensure the performance).
- One instance for the batch-oriented tools.

This basic configuration will be refined during the upcoming phases of the project. It is useful to create a blueprint from this architecture so for every hospital an individual cloud environment can be created.

Moreover, with such a blueprint integrated as a BDCF topology, the scalability can also be taken in charge automatically (i.e. assigning a “scalable instance” to a component for its deployment, and then it is possible increase at runtime the number of instances running this component). It has also to be noted that with the Medolution platform, deployment target may be any of public cloud, private cloud, on premises… and that most of the time for Medolution healthcare apps, private clouds, on premises (like the hosting platform of WP4) will be used. In order for it to work, scalability means that BDCF is able to automatically replicate all semantic information of the current network by auto configuring new nodes.

Scaling in the architecture can potentially be organized vertically or horizontally: The Cassandra database can easily be scaled horizontally by adding addition database instances. The restriction here can be the cloud network throughput when handling high loads of database operations which hit one node. In this case the applications must be able to select the most appropriate database node with enough capacity.

For security reasons the architecture can be partitioned in the cloud to isolate data-analytics and data-storage for a tenant like blueprint. An approach like that would eliminate additional access control on the database level thus creating a simpler and more efficient vertical design. In use-cases where special clients may want to go through the different partitions for instance to compare all medical cases and data, this may be more complicated.

7. Security

7.1. Security Principles

This section addresses how security aspects are covered within an application composed of Medolution components. Security mechanisms of the Medolution components involved in an applicative topology are activated to cover the following aspects:

- **Authentication**: to authenticate users to ensure unauthorized people cannot access to the platform;
- **Authorization**: to define which resources can be accessed by an authenticated user (it requires the authentication process to be setup);
- **Encryption**: to apply algorithms at least on data which is sent on the network, to ensure it cannot be decoded by unauthorized user;
- **Auditing**: log on the system all activities related to the three previous items;
- **Anonymization**: mechanisms to anonymize data before it is made available to Medolution users.

For example a secured datalake has been instantiated on the WP4 hosting platform for a partner whose data and algorithms require to be kept confidential. This datalake is composed of a MAPR Hadoop node (a topology has been built from BDCF MAPR related components), and may be accessed through three user interfaces

- HUE to visualize data stored in Hadoop file system
- NFS to access Hadoop file system
- YARN (cf. 5.7.2) application can be launched on this datalake cluster

Security mechanisms have been activated on the elements of the topology so that only authenticated users can access the datalake regardless of the interface used.

Most of the achievements regarding access control is integration work relying on components intrinsic mechanisms, and is described in sections 7.3 and 7.4. The design and implementation of a complete anonymization solution is achieved within the Medolution project and is described in the following section 7.2.

7.2. Data Anonymization

The data anonymization part below is a quite long section because it reflects an important achievement in the work package, bringing the capability to experiment innovative anonymization technics. Relationship with BDCF appears through the data sharing and data access connectors (sections 7.2.1 and 7.2.2), it is a very recent design choice; Norima is just starting the implementation, and we are aware that deeper integration and APIs will come later in the project, so only a high level view of the system is available in the timeline of this deliverable. Sections 7.2.1 to 7.2.8 give the high level view of the system by itself, while section 7.2.9 provides more insights about anonymization algorithms.

The Data Anonymization System is a Medolution Core Platform Service component that provides a common mechanism for Data Providers to safely publish data to Medolution with the knowledge that any access to this data will be governed by their Data Custody rules around re-identification risk and limited to the Data Researchers who have been explicitly approved. Conceptually, it is illustrated in Figure 23.

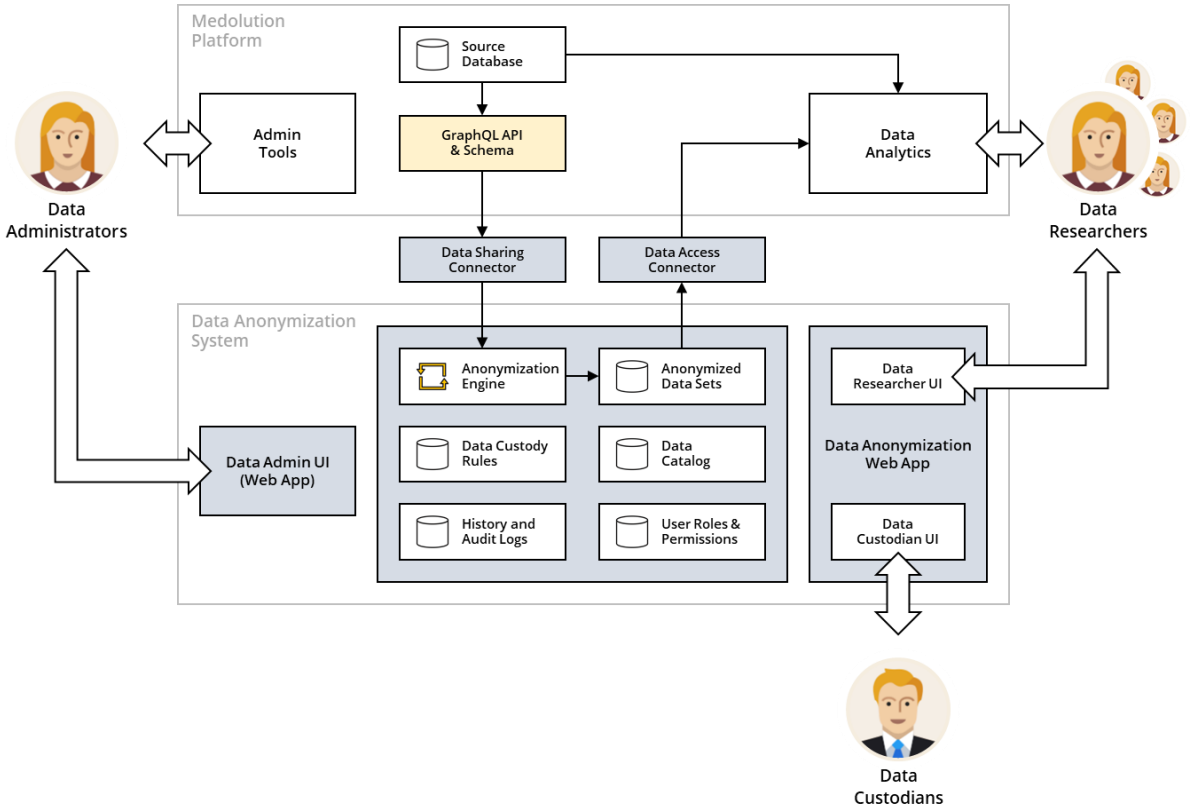

Figure 23: Data Anonymization System

The Medolution data anonymization solution will include two data connectors and a shared online system. The system will support three distinct user roles and will consist of seven primary sub-components:

User Roles

- **Data Administrator** – a technical expert, associated with a Data Provider, who is responsible for the connection between the source database and the Data Anonymization System
- **Data Custodian** – an agent who upholds the custody rules on behalf of a Data Provider. Data Custodians use the Data Anonymization System to generate anonymized data sets and to control who has access to them
- **Data Researcher** – a person who wants to access data for research purposes

Sub-Components

- **Data Sharing Connector** – a mechanism for Data Administrators to connect a source database from the Medolution platform to the shared online system
- **Data Access Connector** – a mechanism for Data Researchers in the Medolution platform to access anonymized data sets that have been approved for research purposes, and to provide them as input to Data Analysis tools running on the Medolution platform
- **Anonymization Engine** – a software tool that can processes large amounts of source data to generate anonymized data sets according to specific input parameters

- **Data Catalog** – a searchable directory that describes all of the data that is available for anonymization and research. The catalog will include descriptors such as variable names, data sources, number of records, and custody rules, but does not include the data records themselves

- **Data Administrator UI** – a web app that provides the mechanism for Data Administrators to configure the connection to a source database in the Medolution Data Lake and publish to the Data Catalog. Data Administrators will specify custody terms around usage as well as configuration settings such as re-identification risk thresholds, suppression limits, etc.
- **Data Custodian UI** – a component of the Data Anonymization web app that provides the mechanism for Data Custodians (assigned by Data Providers) to evaluate data access requests from Data Researchers, and supervise the distribution of anonymized data sets in accordance with the custody terms specified by each Data Provider
- **Data Researcher UI** – a component of the Data Anonymization web app that provides the mechanism for Data Researchers to request access to anonymized data in accordance with the Data Custody terms

The following sections describe the seven sub-components in more detail.

7.2.1. Data Sharing Connector

The Data Sharing connector links a data source on the Medolution platform to the shared online Data Anonymization System. Data administrators will enable the connection by implementing a generic data connector. At the time of writing this document GraphQL is used for a generic data connector, though other options might be considered during the project life. The GraphQL schema will describe the data in the source database and will follow a consistent format with other data sources in the Data Catalog. The Data Sharing connector will relay the GraphQL schema to the shared online system.

7.2.2. Data Access Connector

The Data Access connector links the shared online Data Anonymization System to the Medolution platform. Data Researchers who have been granted access to anonymized data sets will be able to connect those data sets to their preferred analytic tools.

7.2.3. Anonymization Engine

The Anonymization Engine employs multiple algorithms and optimization strategies to generate anonymized output data sets from un-anonymized source data. The purpose of the Anonymization Engine is to reduce the risk of re-identification for any individual whose information was part of the input data. The software is designed to find optimal data sets which reduce risk and maintain research value. Risk is reduced through generalization and suppression of records, and research value is evaluated based on the amount of data lost. See section 7.2.9 for details.

7.2.4. Data Catalog

The Data Catalog is a database of metadata that describes all data that is available for anonymization. Data Providers add to the Data Catalog when a Data Administrator connects the Data Sharing component and publishes their GraphQL schema to the online system. Data Researchers use the Data Catalog to determine what kind of data is available for research.

7.2.5. Data Administrator UI

The Data Administrator web app will provide a mechanism for Data Administrators to publish source data under their control to the Data Catalog so that Data Researchers can request anonymized data sets generated from that data. Data Administrators will use the UI to identify the attributes which must be anonymized, and the custody rules that must be followed for the data. Specifically, the system will connect to a GraphQL API, hosted by the Data Provider, which describes the data attributes and the quasi-identifiers (also called pseudo-identifiers) within the source data set so that the Anonymization Engine can apply the appropriate algorithms to reduce the re-identification risk. A quasi-identifier is an attribute which can be combined with other data elements to enable an antagonist to identify the subject of the data record (e.g. visit data, gender, age, diagnosis, etc.). Each quasi-identifier by itself doesn't identify the subject but when used in combination with other data, especially external data, can be used to re-identify a subject.

The GraphQL schema will optionally include configuration attributes for re-identification risk thresholds as well as other constraints such as suppression maximums, etc.

7.2.6. Data Custodian UI

The Data Custodian web app will provide a mechanism for Data Custodians to evaluate data access requests from Data Researchers, generate anonymized data sets, and control access to those data sets. More specifically, this element of the online system will allow Data Custodians to:

- Publish the terms of their data custody agreement, including restrictions (possibly legal terms) about combining data with external data sets for the purposes of re-identifying subjects.
- Grant or Deny access to the Data attributes based on these requests.
- Control the parameters of the Data Anonymization Engine.
- Record the acceptance of data custody terms.

7.2.7. Data Researcher UI

The Data Researcher web app provides a simple mechanism for Researchers to submit a query which includes the attributes which they want to retrieve. The attributes requested in the query will be looked up in the Data Catalog to identify the quasi-identifiers and build up the anonymization profile of the query. Once approved by Data Custodians, an automated data access sub-component will execute this query against the data sources in the Medolution data lake (via Data Sharing components and GraphQL APIs) and pass the results through the Anonymization Engine sub-component to ensure the pseudo-identifiers have been generalized or suppressed according to the anonymization profile. The query interface will also allow the specification of data loss precision and the maximum suppression level. These two constraints, when combined with re-identification risk (prescribed by the data provider above), guide the algorithms as described below.

Note: All requests, queries, and anonymized data sets will be stored for auditing purposes.

Data Researchers will not be exposed to the details of data anonymization. Instead, they will be able to:

- Browse and search the Data Catalog to discover the Data Providers and attributes which are available and the terms which they must adhere to.
- Request access to anonymized data sets by combining elements from the Data Catalog.
- Acknowledge the terms of use and agree to data custody terms.
- Query anonymized data sets when access has been granted by Data Custodians

7.2.8. Concrete application example:

The diagram in Figure 24 is a first draft that aims to illustrate how Medolution Data Anonymization system can support the Medolution use cases, using the LVAD medical use case (as described in section 6.4) as reference.

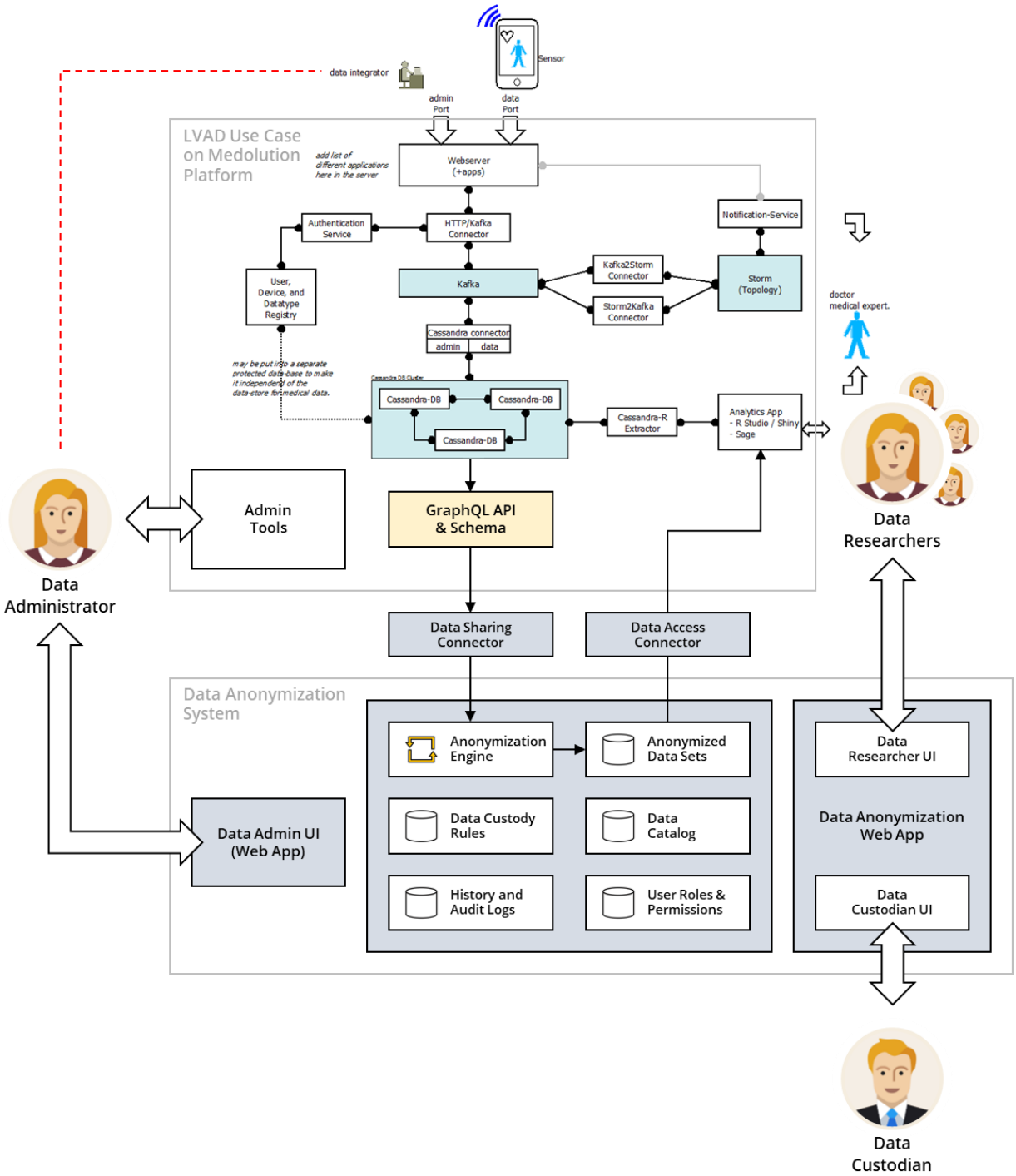

Figure 24: Support of LVAD Medolution Cloud Scenario by the Anonymization system

Since the LVAD Medolution Cloud Scenario represents a typical topology that will be supported by the Medolution platform, and corresponds to the kind of architecture that is straightforward to build

with the BDCF PaaS, the Data Sharing connector and the Data Access connector could be part of the Medolution platform Component Catalogue developed within WP4 to support Medolution applications. The on-going work will further define possible implementation options.

This example also illustrates several benefits of using the Anonymization system to support Medolution use cases and future Medolution applications. Besides providing a possibility to use health-related data of a given medical application for research in a privacy and security respectful way, the Medolution Anonymization system also aims to extend potential for health research when related custody terms and Medolution applications and services allow. Thus, in the example illustrated at Figure 24, other Data Researchers that use Medolution platform applications and services would be able to request anonymized data derived from the Medolution LVAD medical use case, provided related access conditions are met. At the same time, researchers from the Medolution LVAD medical use case would be able to request anonymized data from other sources in the Data Catalog of the Medolution Anonymization system.

7.2.9. Data Anonymization Concepts

This section explains the concepts used in the APIs described above. Re-identification risk, data loss precision and maximum suppression level are parameters of these APIs and are tightly related to the de-identification algorithms, which are built on generalization and suppression techniques.

7.2.9.1. Generalization

The de-identification algorithms perform varying levels of generalization on a particular quasi-identifier. The set of algorithms assigned to a quasi-identifier is an ordered set; where the order is based on the level of generalization performed by the algorithm.

When discussing generalization, we also need to understand the concept of an equivalency class. All records within a dataset belong to the same equivalency class if they have the same values for the set of quasi-identifiers. When quasi-identifiers are generalized, the number of equivalency classes will decrease and the size of the equivalency classes will increase as the level of generalization increases. The converse is also true, as the level of generalization decreases, then number of equivalency classes will increase and the size of the equivalency classes will decrease. Therefore, as the data becomes more generalized, the probability of re-identification is lessened due to larger equivalency class sizes (at the expense of data loss).

Assume the following:

- Visit date is a quasi-identifier
- For the 2015 calendar year there were 118 visits on each date except for the following:
    - 20150724 – 3 visits
    - 20150725 – 18 visits
    - 20150726 – 333 visits
- This gives us a total of (362 * 118) + 3 + 18 + 333 = 43 070 visits in total
- The set of de-identification algorithms for visit date are
    - *a0* – output date in yyyymmdd format
    - *a1* – output date in yyyymm format

- *a2* – output date in yyyy format

- The set of algorithms {*a0, a1, a2*} are ordered based on the level of generalization they perform (from most specific to least specific).

Applying the algorithms to de-identify a dataset we can see that the number of equivalency classes generated by a0 will be greater than the number of equivalency classes generate by applying a1. Similarly, the number of equivalency classes generated by a1 will be more than the number generated by a2.

7.2.9.2. Re-Identification Risk

The stakeholder will also determine an acceptable level of risk that an individual could be re-identified from the de-identified dataset. For the purpose of this document we will discuss the maximum risk of re-identification. A commonly used de-identification criterion is k-anonymity; which stipulates that each record in a dataset is similar to a least k-1 records. Therefore, we can see that maximum risk is equivalent to 1/k. When testing for maximum risk we need to look at the equivalency classes whose size is less than k as these represent the classes with the highest probability of re-identification.

Building on the example in the previous section, assume the following:

- The stake holder has specified the maximum risk of re-identification threshold is .01 (k=100)

If we were to apply de-identification algorithm *a0* (yyyymmdd), the size of the equivalency class for 20150724 is 3 and 20150725 is 18. This results in 1/3 and 1/18 chance of re-identification respectively, which are both higher than our risk threshold of .01; therefore, we could not release the data using a0. However, if we were to apply *a1* (yyyymm) we would end up with 12 equivalency classes, and based on our stated distribution the smallest equivalency class would be for 201502. The size of 201502 would be 28 days * 118 visits / day = 3304. The 1/3304 chance of re-identification is well within our stated threshold of .01. Based on this we could safely release the data using *a1* as our algorithm.

7.2.9.3. Suppression

The third aspect to the approach is the concept of suppression. This deals with what needs to be done to de-identified data to ensure the risk threshold is met. If we have an equivalency class with a re-identification probability greater than our risk threshold we will need to perform suppression on the data so that our threshold can be met.

Suppression, like generalization, introduces data loss. When we are evaluating de-identification algorithms to find the optimal set, the maximum amount of suppression that is acceptable must be specified. For example, if a set of de-identification algorithms require a suppression 80 percent of our records to achieve a .01 maximum risk to identification, it would clearly not be acceptable.

Introducing the concept of maximum amount of suppression to our running example, assume the following:

- The stakeholder has specified the maximum amount of suppression is .10 (10 percent).

If we were to apply algorithm *a0* (yyyymmdd) we would have to suppress the records for the 20150724 (3 records) and the 20150725 (18 records) equivalency classes as they exceed the maximum risk of re-identification threshold of .01. This gives us a suppression ratio of (3+18) / 43 070 = .00048. This is well within the specified maximum amount of suppression (.10). Therefore,

a0 would indeed be an acceptable de-identification algorithm has it results in an acceptable amount of suppression to achieve our risk threshold.

Without the concept of suppression, we would have rejected *a0* and we would have selected *a1*.

Now that we have identified the records to suppress, we need to specify the techniques to use when performing the suppression. We could do some of the following:

- Case-wise deletion.
  Remove all 21 records from the release data.

- Quasi-identifier omission.
  Omit the quasi-identifier (visit date) for the 21 records, but release all other data associated with the visits. (Ignore the fact that we can infer the visit date for the 21 records as they are the only 2 dates not represented in our released file; this is a contrived example to add context to our abstract discussion).

- Quasi-identifier generalization.
  Generalize the quasi-identifier for the 21 records. This would result in the visit date being generalized to yyyymm or 201507.

- Local cell suppression.
  We could suppress some, but not all, of the quasi-identifiers. When we remove a quasi-identifier the resultant equivalency class is less detailed than the original and therefore would be a larger set (if the entire data set was taken into account). We would keep omitting quasi-identifiers until the size of resultant equivalency class satisfies our maximum risk threshold. If had to remove all of the quasi-identifiers to satisfy our maximum risk threshold we effectively have “Quasi-identifier omission”

As we can see, not only do we need to know the risk tolerance when determining which de-identification algorithms should be used, we also need to know the maximum amount suppression that will be allowed.

7.2.9.4. Quantifying Data Loss

Generalization performed by applying a de-identification algorithm will cause data loss due to the amount of precision lost during the operation. A de-identification algorithm has a precision loss factor that indicates the degree of generalization performed. For example, generalizing a date to yyyymmddd will result in 365 discrete values for a given year, generalizing a date to yyyymmd results in 12 discrete values for a given year, and generalizing to yyyy will result in a single value for a given year. As we can see the different algorithms have different precision loss factors. Also note that the precision lost between different algorithms is not constant, the precision lost between yyyymmdd and yyyymm is different than the precision lost between yyyymm and yyyy.

Suppression also introduces data loss. To aid in quantifying data loss encountered during suppression we need to know the clinical significance of the quasi-identifiers.

During suppression, when we are performing local cell suppression, we would need to know which quasi-identifier is the least clinical significant. This would allow us to omit the quasi-identifiers in clinical significance order, omitted the least significant first.

The clinical significance of a quasi-identifier, along with the precision loss factor of the de-identification algorithms, will allow us to quantify data loss created by generalization and suppression for a given set of de-identification algorithms. This gives us the capability to compare the amount of data loss generated by different sets of de-identification algorithms so that we can choose the one with the least of amount of data loss.

As precision loss increases, the clinical significance will decrease. For example, a visit date in yyyymmdd format has more clinical significance than a visit date in yyyymm format. We have gone from 365 discrete values to 12, a factor of 30.4. However, this does not imply that the clinical significance has decreased by a factor of 30.4.

The magnitude of the change in clinical significance is independent of the precision loss factor.

7.3. Managing Access Rights

There is no global management of access rights in Medolution, each application will make use of the mechanisms provided by the building blocks it uses. It has been agreed that there was no added value neither innovation brought on these aspects in the Medolution challenges.

Big Data distributions like MapR, Hortonworks or Cloudera come with their own security process or give a way to connect to dedicated components, which helps end-user to securely interact with the entire ecosystem. The Big Data Capability Framework integrates such security features on MapR distribution only, allowing enabling the MapR wire-level secure mode at application deployment. This is documented inside the BDCF user guide [4].

MapR core system proposes an access rights management based on token principle. A user authenticates giving its credentials (user/password, Kerberos ticket…) and then the system returns a ticket which gives him rights to access to MapR services. Authorization is one part of MapR wire-level secure mode, as encryption is another.

During this second period of the project, in a similar way the setup of secured Hadoop MAPR clusters had been specified, the setup of secured ELK (Elastic stack) topologies (as the one illustrated in Figure 17) has been addressed. To secure the Elastic stack, the X-Pack software is used, and allows to configure Authentication and Access Control on Kibana (cf. 5.6.1) and ElasticSearch (cf. 5.2.1), as well as the encryption of communications within the Elastic stack (between Logstash, ElasticSearch and Kibana); the integration of an LDAP directory is also possible. In addition, X-Pack also enables the security audit, to monitor security events. This secured Elastic stack setup is not yet integrated in the BDCF V3.0 version.

7.4. Secured Data Transmission

As part of BDCF, such a feature is also already available on the Big Data platform, through the MapR wire-level secure mode, together with the access rights management.

When this mode is enabled, all MapR core services encrypt their communications, MapR file system in particular. That will prevent potential attacks on decoding for example healthcare related data.

As said in the previous section, secured data transmission within an Elastic topology is also available.

8. Medolution Platform APIs

Medolution Platform API is a RESTful interface that gives the ability to plug with existing enterprise systems (Development and Operations) and other stakeholders. This layer implements the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) standard. In fact, it makes possible to build and manage a full application lifecycle on hybrid infrastructure in the same way as using the GUI (Alien4Cloud). According to the Medolution project, Medolution Platform API could be used for the management of some preconfigured templates customized for Medolution use cases. It also allows the management and configuration of the interface between the deployed applications and other components such as IoT platforms, IoT gateways, portal platforms.

Using this RESTful interface, management features can be summarised as follows:

- Listing applications catalogue: allow Medolution user to list existing templates for on demand usage.
- Building a new application: Medolution API implements the TOSCA standard, which is used to build and orchestrate PaaS applications in the cloud.
- Deployment of applications: creation of the application that will be hosted on specific cloud platforms.
- Versioning and application lifecycle management
- Configuration of application APIs and access management.

9. Hosting Platforms

At this stage of the project, several deliveries of the BDCF framework have been made available on the hosting platform delivered by Bull in task 4.4 of the work package (V2.0, V2.1, V3.0…).

This hosting platform is described in the document describing the merged D4.2/D4.3 work package deliverable [5], it is composed of

- HW delivering high memory, compute and storage capabilities: a cluster architecture hosted in Bull French facilities providing a small datacentre for Cloud/Big Data, including a mix of standard x86 servers, large In-Memory Servers (Bullion 4/8TB up to 16 sockets), as well as different types of storage systems.
- An OpenStack IaaS, Liberty version, to virtualize this HW
- The BDCF framework to provide a PaaS access to this platform, i.e. creating VMs, deploying applications built from software components of the BDCF catalogue.

Within this platform, partners can bring their data and then install, test and validate their developments throughout the project whether they are technology providers or applications developers. The platform is located in a DMZ zone so as to allow all projects partners to have access to the platform resources and to their own private dedicated network zone and space.

Each partner may ask an access to this platform, the procedure is described in the Medolution SharePoint wiki, he will receive VPN and login information to access the BDCF portal (Alien4Cloud).

At this date, IMT, UPEC, SRDC, Maidis, Prologue, Norima and Sopheon have received access to the platform.

That way, any partner has the ability to test the platform by designing and deploying applications built from components already available in the catalogue. Of course partners have also the ability to integrate new components in the catalogue, a document “Components Developer Guide” has been provided for this purpose. Currently version 3.0 of the BDCF framework is available on the platform. “BDCF User Guide” [4] and “Component Developer Guide” [3] are available on the Medolution SharePoint.

10. List of Figures

11. List of Tables

12. Glossary

API	Application Programming Interface
BDCF	Bull Big Data Capabilities Framework
BLE	Bluetooth Low Energy
CCD	Continuity of Care Document
CDA	Clinical Document Architecture (HL7)
CEN	European Committee for Standardization
CEP	Complex Event Processing
CHA	Continua Health Alliance
CLDB	Container Location Database
CQRS	Command Query Responsibility Segregation
DMZ	DeMilitarized Zone
EHR	Electronic Health Record
ELK	ElasticSearch Logstash Kibana
ER	Entity Relationship
FHIR	Fast Healtcare Interoperability Resources (HL7)
FPP	Full Project Proposal
GUI	Graphical User Interface
HITSP	Health Information Technology Standards Panel
HL7	Health Level Seven
HTTP	HyperText Transfer Protocol
HW	Hardware
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IHE	Integrating the Healthcare Enterprise
IOT	Internet of Things
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LVAD	Left Ventricular Assist Device
NFS	Network File System
OASIS	Open Advanced Standard for the information society
PaaS	Platform as a Service
PCC	Patient Care Coordination
PHD	Personal Health Data
REST	Representational State Transfer
TOSCA	Topology and Orchestration Specification for Cloud Applications
UI	User Interface
VM	Virtual Machine
VPN	Virtual Private Network
XML	Extensible Markup Language

13. References

[1] ITEA Deliverable, *D4.1 Medolution Platform APIs and Specification V1,* Nov. 2016.

[2] Norima, *Medolution D1.1, State of the Art Analysis,* Nov. 2016.

[3] Atos Bull, *BDCF Component Developer Guide,* 2016.

[4] Atos Bull, *BDCF 2.0 User Guide,* 2016.

[5] Atos Bull, *Medolution D4.2 / D4.3, Medolution Core Platform and Hosting Platform,* 2016.