INTERNATIONAL STANDARD

**ISO/IEC 20547-3**

First edition
2020-03

# Information technology — Big data reference architecture —

## Part 3: Reference architecture

*Technologies de l'information — Architecture de référence des mégadonnées —*

*Partie 3: Architecture de référence*



Reference number
ISO/IEC 20547-3:2020(E)



ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

# Contents

Page

# Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 42, *Artificial intelligence*.

A list of all parts in the ISO/IEC 20547 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

# Introduction

The ISO/IEC 20547 series is intended to provide users with a standardized approach to developing and implementing big data architectures and provide references for approaches. ISO/IEC TR 20547-1 provides users with an overview of the reference architecture framework described in this document and a process for applying that framework in developing an architecture. ISO/IEC TR 20547-2 provides a collection of big data use cases and decomposes those use cases into technical considerations that big data architects and system implementers can consider. This document describes the reference architecture in terms of User and Functional views. Those views can be used by the big data architect to describe their specific system. ISO/IEC 20547-4 describes the security and privacy aspects unique to big data. ISO/IEC TR 20547-5 provides a list of standards and their relationship to the reference architecture that architects and implementers can consider as part of the design and implementation of their system.

Each of these parts is built on the common vocabulary and concepts described in ISO/IEC 20546.

In general terms, reference architecture provides an authoritative source of information about a specific subject area that guides and constrains the instantiations of multiple architectures and solutions (see 3.2). Reference architectures generally serve as a reference foundation for solution architectures and can also be used for comparison and alignment purposes.

The key goal of this reference architecture is to facilitate a shared understanding across multiple products, organizations, and disciplines about current architectures and future direction.

The reference architecture presented in this document provides an architecture framework for describing the big data components, processes, and systems to establish a common language for the various stakeholders named as big data reference architecture (BDRA). It does not represent the system architecture of a specific big data system. Instead, it is a tool for describing, discussing, and developing system-specific architectures using an architecture framework of reference. It provides generic high-level architectural views that are an effective tool for discussing the requirements, structures, and operations inherent to big data. The model is not tied to any specific vendor products, services or reference implementation, nor does it define prescriptive solutions that inhibit innovation.

# Information technology — Big data reference architecture —

# Part 3: Reference architecture

## 1 Scope

This document specifies the big data reference architecture (BDRA). The reference architecture includes concepts and architectural views.

The reference architecture specified in this document defines two architectural viewpoints:

— a user view defining roles/sub-roles, their relationships, and types of activities within a big data ecosystem;

— a functional view defining the architectural layers and the classes of functional components within those layers that implement the activities of the roles/sub-roles within the user view.

The BDRA is intended to:

— provide a common language for the various stakeholders;

— encourage adherence to common standards, specifications, and patterns;

— provide consistency of implementation of technology to solve similar problem sets;

— facilitate the understanding of the operational intricacies in big data;

— illustrate and understand the various big data components, processes, and systems, in the context of an overall big data conceptual model;

— provide a technical reference for government departments, agencies and other consumers to understand, discuss, categorize and compare big data solutions; and

— facilitate the analysis of candidate standards for interoperability, portability, reusability, and extendibility.

## 2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 8000-2, *Data quality — Part 2: Vocabulary*

ISO/TS 8000-60, *Data quality — Part 60: Data quality management: Overview*

ISO 8000-61, *Data quality — Part 61: Data quality management: Process reference model*

ISO/IEC 38500, *Information technology — Governance of IT for the organization*

ISO/IEC 38505-1, *Information technology — Governance of IT — Governance of data — Part 1: Application of ISO/IEC 38500 to the governance of data*

ISO/IEC TR 38505-2, *Information technology — Governance of IT — Governance of data — Part 2: Implications of ISO/IEC 38505-1 for data management*

ISO 55000, *Asset management — Overview, principles and terminology*

ISO 55001, *Asset management — Management systems — Requirements*

ISO 55002, *Asset management — Management systems — Guidelines for the application of ISO 55001*

ISO/IEC/IEEE 42010, *Systems and software engineering — Architecture description*

ISO/IEC 20546, *Information technology — Big data — Overview and vocabulary*

ISO/IEC 17789, *Information technology — Cloud computing — Reference architecture*

## 3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 8000-2, ISO/TS 8000-60, ISO 8000-61, ISO/IEC 38500, ISO/IEC 38505-1, ISO/IEC TR 38505-2, ISO 55000, ISO 55001, ISO 55002, ISO/IEC/IEEE 42010, ISO/IEC 20546, ISO/IEC 17789 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

**3.1**
**data**
reinterpretable representation of *information* (3.3) in a formalized manner suitable for communication, interpretation, or processing

[SOURCE: ISO/IEC 2382:2015, 2121272]

**3.2**
**reference architecture**
authoritative source of information about a specific subject area that guides and constrains the instantiations of multiple architectures and solutions

Note 1 to entry: This document utilizes the definition of reference architecture from DoD "reference architecture description"[7].

Note 2 to entry: Reference architectures generally serve as a foundation for solution architectures and can also be used for comparison and alignment of instantiations of architectures and solutions.

**3.3**
**information**
*data* (3.1) that are processed, organized and correlated to produce meaning

Note 1 to entry: Information concerns facts, concepts, objects, events, ideas, processes, etc.

**3.4**
**activity**
specified pursuit or set of tasks

[SOURCE: ISO/IEC 17789:2014, 3.2.1]

**3.5**
**knowledge**
maintained, processed, and interpreted *information* (3.3)

[SOURCE: ISO 5127:2017, 3.1.1.17]



**3.6**
**functional component**
functional building block needed to engage in an *activity* (3.4), backed by an implementation

[SOURCE: ISO/IEC 17789:2014, 3.2.3]

**3.7**
**data governance**
property or ability that needs to be coordinated and implemented by a set of *activities* (3.4) aimed to design, implement and monitoring a *strategic plan for data asset management*

Note 1 to entry: Governance of data is described in ISO/IEC 38505-1.

Note 2 to entry: Data asset is understood as a set of data items, or data entities, that have a real or potential benefit for an organization. Data asset is a subset of asset defined in ISO 55000. A benefit is an advantage to the organization of the actionable knowledge derived from an analytic system. It is often ascribed to big data due to the understanding that data has potential benefit that was typically not considered previously.

Note 3 to entry: A strategic plan for data asset management is a document specifying how *data management* (3.15) is to be aligned to the organizational strategy. This term has the same meaning as strategic asset management plan (SAMP) defined in ISO 55000 with data point of view.

**3.8**
**data quality**
degree to which the characteristics of data satisfy stated and implied needs when used under specified conditions

[SOURCE: ISO/IEC 25024:2015, 4.11]

**3.9**
**data quality management**
coordinated activities to direct and control an organization with regard to data quality

[SOURCE: ISO 8000-2:2018, 3.4.9]

**3.10**
**party**
natural person or legal person, whether or not incorporated, or a group of either

[SOURCE: ISO/IEC 17789:2014, 7.2.3]

**3.11**
**policy**
intention and direction of an organization as formally expressed by its top management

[SOURCE: ISO 55000:2014, 3.1.18, modified — The term has been changed to the singular form and the final stop has been removed from the definition.]

**3.12**
**role**
set of *activities* (3.4) that serves a common purpose

[SOURCE: ISO/IEC 17789:2014, 3.2.7]

**3.13**
**stream**
list of flow objects attached to a port of a flow object

[SOURCE: ISO/IEC 10179:1996, 4.33, modified — by deleting leading article and trailing full stop.]

**3.14**
**sub-role**
subset of the *activities* (3.4) of a given *role* (3.12)

[SOURCE: ISO/IEC 17789:2014, 3.2.9]

**3.15**
**data management**
set of *activities* (3.4) aimed to implement the big data architecture that best meet business goals by following the strategic plan for data management assessment

**3.16**
**data lifecycle**
stages in the management of a data

Note 1 to entry: The target of lifecycle (defined in ISO 55000) is data in this document.

**3.17**
**application programming interface**
**API**
boundary across which application software uses facilities of programming languages to invoke services

[SOURCE: ISO/IEC 18012-2:2012, 3.1.4, modified — Note 1 to entry has been removed and the final stop has been deleted from the definition.]

## 4 Abbreviated terms

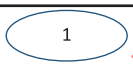
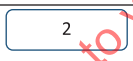
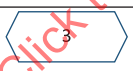
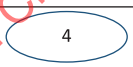
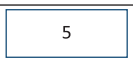
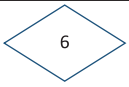
| | |
|---|---|
| ACID | atomicity, consistency, isolation, and durability |
| API | application programming interface |
| CEP | complex event processing |
| CPU | central processing unit |
| BDA | big data auditor |
| BDAP | big data application provider |
| BDAcP | big data access provider |
| BDAnP | big data analytics provider |
| BDC | big data consumer |
| BDCP | big data collection provider |
| BDFP | big data framework provider |
| BDIP | big data infrastructure provider |
| BDP | big data provider |
| BDPlaP | big data platform provider |
| BDPreP | big data preparation provider |
| BDProP | big data processing provider |

| | |
|---|---|
| BDRA | big data reference architecture |
| BDSD | big data service developer |
| BDSO | big data system orchestrator |
| BDSP | big data service partner |
| BDVP | big data visualization provider |
| DG | data governance |
| DM | data manager |
| DQM | data quality manager |
| PII | personally identifiable information |
| RA | reference architecture |

## 5 Conventions

The diagrams that appear in this document are presented using the conventions that are shown in Table 1. This notation is used as described in ISO/IEC 17789.

**Table 1 — Legend to the diagrams used throughout this document**

| Object | Meaning |
|---|---|
|  | Party |
|  | Role |
|  | Sub-Role |
|  | Activity |
|  | Functional component |
|  | Cross-cutting aspect |

## 6 Big data reference architecture concepts

### 6.1 General

This document defines a BDRA that serves as a fundamental reference point for big data standardization and which provides an overall architecture framework for the basic concepts and principles of a big data system.

This document describes the logical relationships between the roles/sub-roles, activities, and functional components, and cross-cutting aspects that comprise a big data system architecture.

Standards can be relevant to some of these relationships. Standards associated with a relationship can be used to:

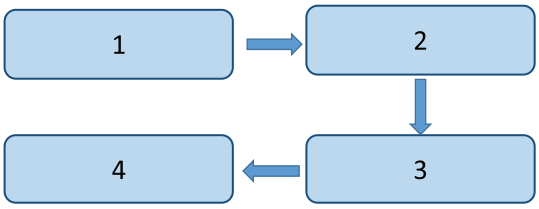
— specify degrees of information flow or other types of interoperability; and/or

— ensure specified degrees of quality (e.g. security or service level).

Logical relationships defined in this architecture are a significant part of specifying the BDRA and its behaviour. The relationship describes matters such as the categories of information flows between the functional components in the BDRA.

## 6.2 Views

Big data can be described using a viewpoint approach. Four distinct viewpoints are used in the BDRA (see Figure 1 and Table 2):




**Key**

1 user view
2 functional view
3 implementation view
4 deployment view

**Figure 1 — Transformations between architectural views**

**Table 2 — BDRA views**

| BDRA view | Description of the BDRA view | Scope |
|---|---|---|
| User view | The ecosystem of big data with the stakeholders (used in ISO/IEC/IEEE 42010), the roles, the sub-roles and the big data activities | Within scope |
| Functional view | The functions necessary for the support of big data activities | Within scope |
| Implementation view | The functions necessary for the implementation of big data within service parts and/or infrastructure parts | Out of scope |
| Deployment view | How the functions of big data are technically implemented within already existing infrastructure elements or within new elements to be introduced in this infrastructure | Out of scope |

NOTE While details of the user view and functional view are addressed within this document, the implementation and deployment views are related to technology and vendor-specific big data implementations and actual deployments, and are therefore out of the scope of this document.

## 6.3 Overview of user view

The user view addresses the ecosystem of big data with the following concepts:

— **parties**: a party is a natural person or legal person, whether or not incorporated, or a group of either or both parties in a big data ecosystem are its stakeholders;

— **roles and sub-roles**: a role is a set of big data activities that serves a common purpose. a sub-role is a subset of the big data activities for a given role, and different sub-roles can share the big data activities associated with a given role;

— **activities**: an activity is defined as a specified pursuit or set of tasks. big data activities need to have a purpose and deliver one or more outcomes and these are conducted using functional components;

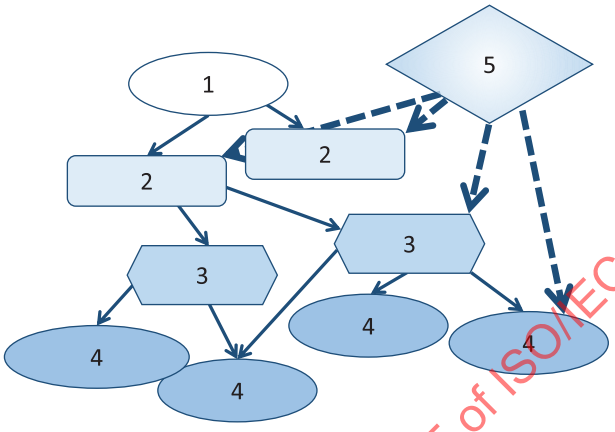


— **cross-cutting aspects**: cross-cutting aspects can be shared and can impact multiple roles, and big data activities. Cross-cutting aspects may map to multi-layer functions and their associated functional components which implement the activities within the cross-cutting aspect.

NOTE A party can assume more than one role at any given point in time and can engage in a specific subset of activities of that role. Examples of parties include, but are not limited to, large corporations, small- and medium-sized enterprises, government departments, academic institutions and private citizens.

Figure 2 illustrates the entities that are defined for the user view.




**Key**

1 party
2 role
3 sub-role
4 activity
5 cross-cutting aspect

**Figure 2 — User view entities**



## 6.4 Overview of functional view

The functional view is a technology-neutral view of the functions necessary to form a big data system. The functional view describes the distribution of functions necessary for the support of big data activities.

The functional architecture also defines the dependencies between functions.

The functional view addresses the following big data concepts:

— **functional components**: a functional component is a functional building block needed to engage in an activity, backed by an implementation;

— **functional layers:** a layer is a set of functional components that provide similar capabilities or serve a common purpose;

— **multi-layer functions**: the multi-layer functions include functional components that provide capabilities that are used across multiple functional layers, and they are grouped into subsets.

NOTE Not all layers or functional components are necessarily instantiated in a specific big data system.

Figure 3 illustrates the concepts of functional components, layers and multi-layer functions.

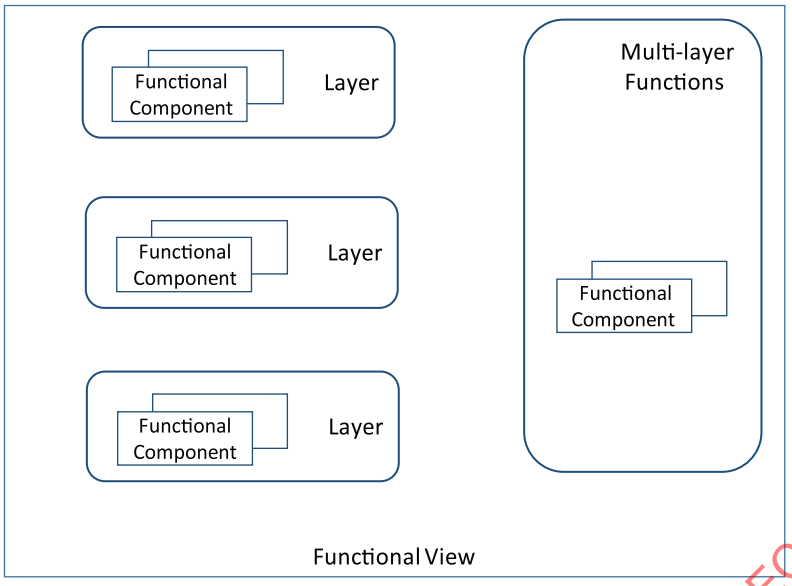



**Figure 3 — Functional layering**

## 6.5 Relationship between the user view and the functional view

Figure 4 illustrates how the user view provides the set of big data activities that are represented within the functional view.

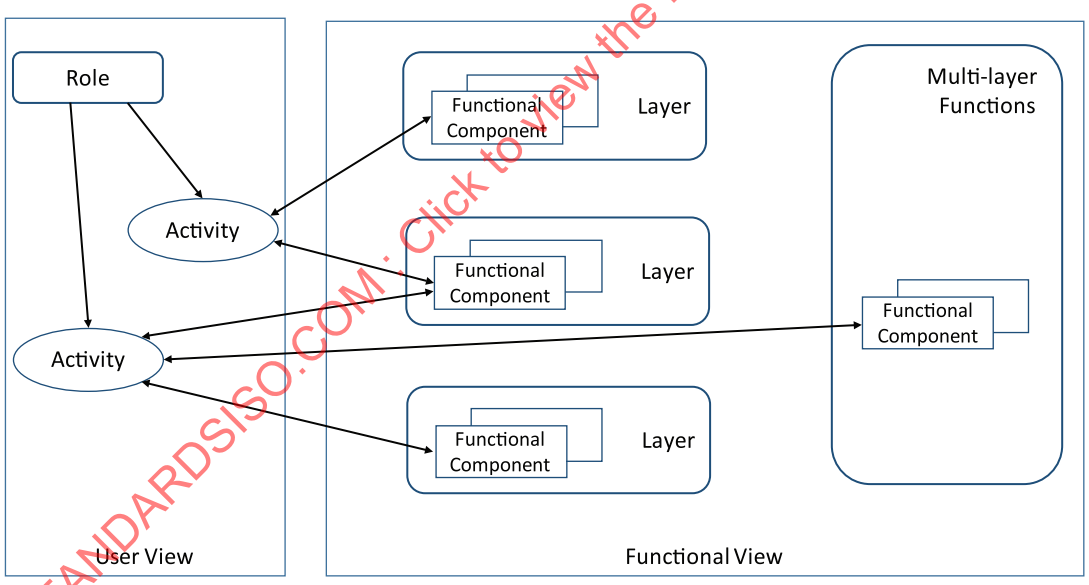



**Figure 4 — From user view to functional view**

## 6.6 Relationship of the user view and functional view to cross-cutting aspects

Cross-cutting aspects, as their name implies, apply both across the user view and across the functional view of big data.

Cross-cutting aspects apply to roles and sub-roles in the user view and they directly or indirectly affect the activities that those roles perform.

Cross-cutting aspects also apply to the functional components within the functional view which are used when performing the activities described in the user view (See Figure 4).

Cross-cutting aspects of big data described in Clause 9 include:

— security and privacy;

— management;

— data governance.

# 7 User view

## 7.1 Big data roles, sub-roles, and activities

Given that distributed services and their delivery are at the core of big data, all big data related activities can be categorized into three main groups: activities that use big data, activities that provide big data analytics services and activities that provide data.

This clause contains descriptions of some of the common roles and sub-roles associated with big data.

It is important to note that a party can play more than one role at any given point in time. When playing a role, the party can restrict itself to playing one or more sub-roles. Sub-roles are a subset of the big data activities of a given role.

As shown in Figure 5, the roles of big data are:

— big data application provider (BDAP) (see 7.2);

— big data framework provider (BDFP) (see 7.3);

— big data service partner (BDSP) (see 7.4);

— big data provider (BDP) (see 7.5);

— big data consumer (BDC) (see 7.6).

NOTE Big data provider is any data provider to the BDRA.

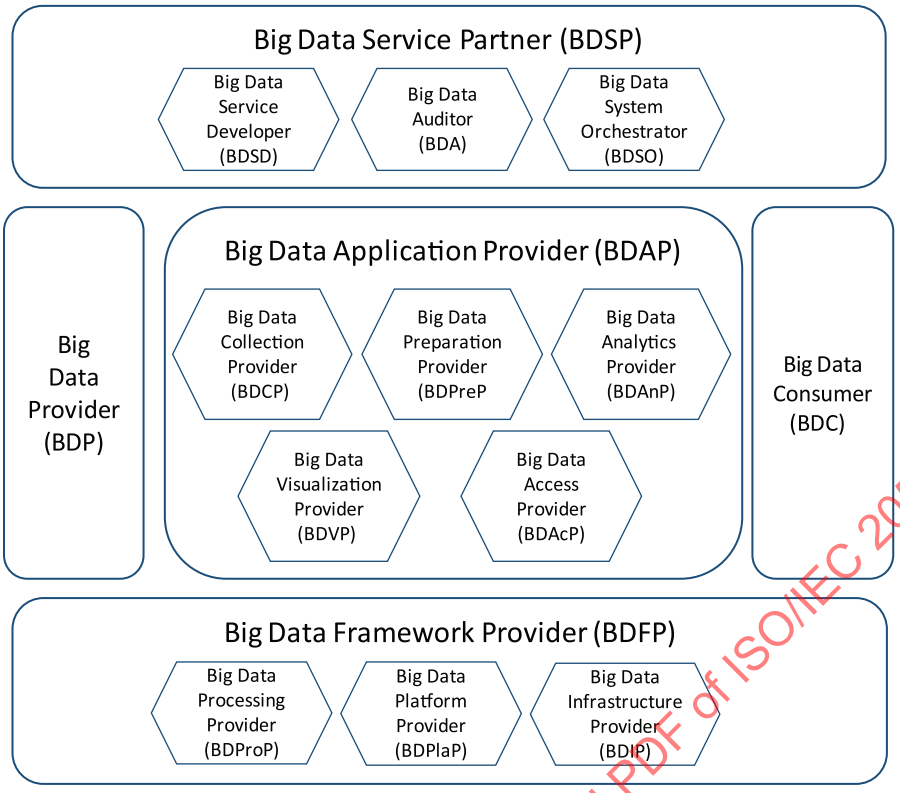



**Figure 5 — Big data roles**

Annex B provides examples of the relationship of roles in big data ecosystems.

Each of the sub-roles shown in Figure 5 is described in more detail in 7.2 to 7.6.

## 7.2 Role: Big data application provider (BDAP)

### 7.2.1 General

The BDAP executes the manipulations of the big data lifecycle. This is where the general capabilities within user view of the big data reference architecture as shown in Figure 5 are combined to produce the specific data system.

NOTE 1 While the activities of an application provider are the same whether the solution being built concerns big data or not, the methods and techniques have changed because the data and data processing is parallelized across resources.

NOTE 2 As data propagates through the ecosystem, it is being processed and transformed in different ways in order to extract the value from the information. Each activity of the big data application provider can be implemented by independent stakeholders and deployed as stand-alone services.

NOTE 3 The BDAP can be a single instance or a collection of more granular big data application providers, each implementing different steps in the big data lifecycle. Each of the activities of the big data application provider can be a general service invoked by the data provider or big data consumer, such as a web server, a file server, a collection of one or more application programs, or a combination.

NOTE 4 The BDAP is in charge of the implementation, testing and validation of the data quality business rules, requirements and metrics that assure the correct management of data in the big data system. Any big data application provider can apply the data quality requirements throughout the big data lifecycle.

The BDAP is composed of the following five sub-roles as shown in Figure 6:

— big data collection provider (BDCP) (see 7.2.2);



— big data preparation provider (BDPreP) (see 7.2.3);

— big data analytics provider (BDAnP) (see 7.2.4);

— big data visualization provider (BDVP) (see 7.2.5);

— big data access provider (BDAcP) (see 7.2.6).

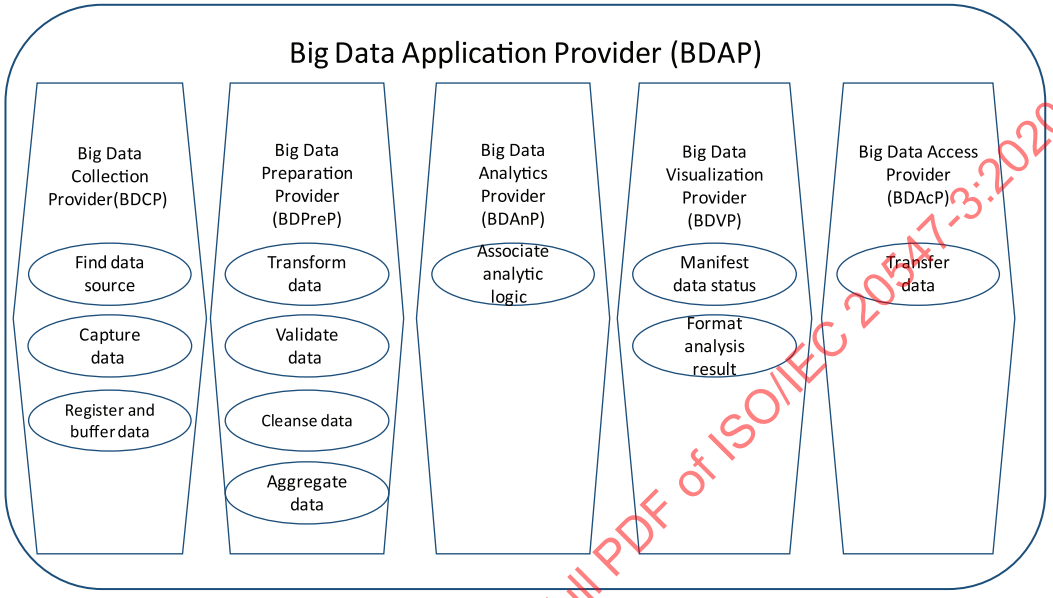



**Figure 6 — Big data activities relating to big data application provider sub-roles**



### 7.2.2 Sub-role: big data collection provider (BDCP)

The BDCP is a sub-role of BDAP which is responsible for the collection of big data from data provider. This can be a general service, such as a file server or web server to accept or perform specific collections of data, or it can be an application specific service designed to pull data or receive pushes of data from the data provider.

The BDCP activities are as follows:

— the **find data source** activity is focused on searching and storing data source information as a form of metadata which can be used for capturing and/or storing data;

— the **capture data** activity is focused on converting available data (e.g. web document, blog data, etc.) into a form that can be handled by system;

— the **register and buffer data** activity is focused on storing data into data registry or holding data before transferring it to other tasks or processes.

### 7.2.3 Sub-role: big data preparation provider (BDPreP)

The BDPreP is a sub-role of the BDAP which is responsible for preparing data from raw data to ready for analyzing data.

The BDPreP activities are as follows:

— the **transform data** activity is focused on converting data or information from one format to another;

— the **validate data** activity is focused on ensuring that the data is correct based on the validation constraints such as correctness, meaningfulness, security and privacy, etc.;

— the **cleanse data** activity is focused on detecting inaccurate part of data and correcting them by replacing, modifying or deleting;

— the **aggregate data** activity is focused on combining two or more data into one dataset in summary form.

Data validation and data cleansing should be guided by the application of the data quality management.

#### 7.2.4 Sub-role: big data analytics provider (BDAnP)

The BDAnP is a sub-role of BDAP which is responsible for analysing big data in order to meet the requirements of the data processing algorithms for processing the data to produce insights that address the technical goal.

The BDAnP activity includes an associate analytic logic activity which involves modelling data processes with associated logic for extracting information from the data based on the requirements of the application.

#### 7.2.5 Sub-role: big data visualization provider (BDVP)

The BDVP is a sub-role of BDAP which is responsible for presenting data source information or analysis result to big data consumer. The objective of these activities is to format and present data in such a way as to optimally communicate meaning and knowledge.

The BDVP activities are as follows:

— the **manifest data status** activity involves describing data status in data storage. this can include various visualization, classification criteria, etc.;

— the **format analysis result** activity involves formatting processed data for clear and efficient communication. This can include visual representation, overlaying, etc.

#### 7.2.6 Sub-role: big data access provider (BDAcP)

The BDAcP is a sub-role of BDAP which is responsible for exchanging big data between big data application and data provider or big data consumer.

The BDAcP activity includes a **transfer data** activity focused on passing or moving big data from one system to another system with data transmission integrity, continuity, security and privacy.

### 7.3 Role: big data framework provider (BDFP)

#### 7.3.1 General

The BDFP consists of one or more hierarchically organized instances of the components. There is no requirement that all instances at a given level in the hierarchy be of the same technology.

NOTE In fact, most big data implementations are hybrids that combine multiple technology approaches in order to provide flexibility or meet the complete range of requirements, which are driven from the big data application provider.

The BDFP is comprised of the following three sub-roles as shown in Figure 7:

— big data infrastructure provider (BDIP) (see 7.3.2);

— big data platform provider (BDPlaP) (see 7.3.3);

— big data processing provider (BDProP) (see 7.3.4).



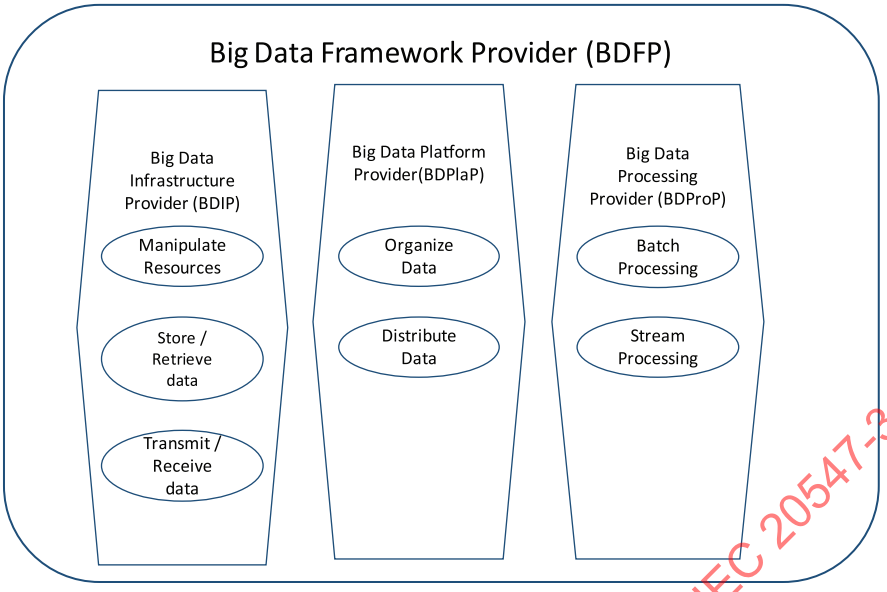



**Figure 7 — Big data activities relating to big data framework provider sub-roles**

### 7.3.2 Sub-role: big data infrastructure provider (BDIP)

The BDIP is a sub-role of BDFP which is responsible for providing system resources including system facilities (e.g. networking, computing, storage, etc.) and physical environment (e.g. computer rooms, electric powers, air conditioners, etc.).

The BDIP activities are as follows:

— the **manipulate resources** activity is focused on handling or controlling physical or virtual resources;

— the **store/retrieve data** activity involves persisting and recalling data from storage (manipulating data at rest.);

— the **transmit/receive data** activity is focused on transferring data via network (putting data in motion.).

### 7.3.3 Sub-role: big data platform provider (BDPlaP)

The BDPlaP is a sub-role of BDFP which is responsible for providing platforms to organize and distribute big data on big data infrastructure.

The BDPlaP activities are as follows:

— the **organize data** activity involves arranging, indexing and linking the data in ways that are suitable for the specific applications and analytics;

— the **distribute data** activity involves allocating data across infrastructure resources to maximize data locality for distributed computation performance.

### 7.3.4 Sub-role: big data processing provider (BDProP)

The BDProP is a sub-role of BDFP which is responsible for supporting computing and analytic processes for BDAP activities.

The BDProP activities are as follows:

— **process data in batches**: process data in large increments and on a non-continuous basis. Batch process is used when response time is not critical. Batch processing is most often associated with the volume of the data or complexity of the analysis;

— **process data in streams**: process data continuously in small increments (typically individual records or data elements). Stream processing is used when response time is critical and is most often associated with the velocity of the data.

## 7.4 Role: big data service partner (BDSP)

### 7.4.1 General

The BDSP is a role which is engaged in support of, or auxiliary to, activities of among the big data application provider, the big data framework provider, the big data provider or the big data consumer, or all.

A BDSP's big data activities vary depending on the type of partner and their relationship with other roles in big data ecosystem.

The BDSP is comprised of the following three sub-roles as shown in Figure 8:

— big data service developer (BDSD) (see 7.4.2);

— big data auditor (BDA) (see 7.4.3);

— big data system orchestrator (BDSO) (see 7.4.4).

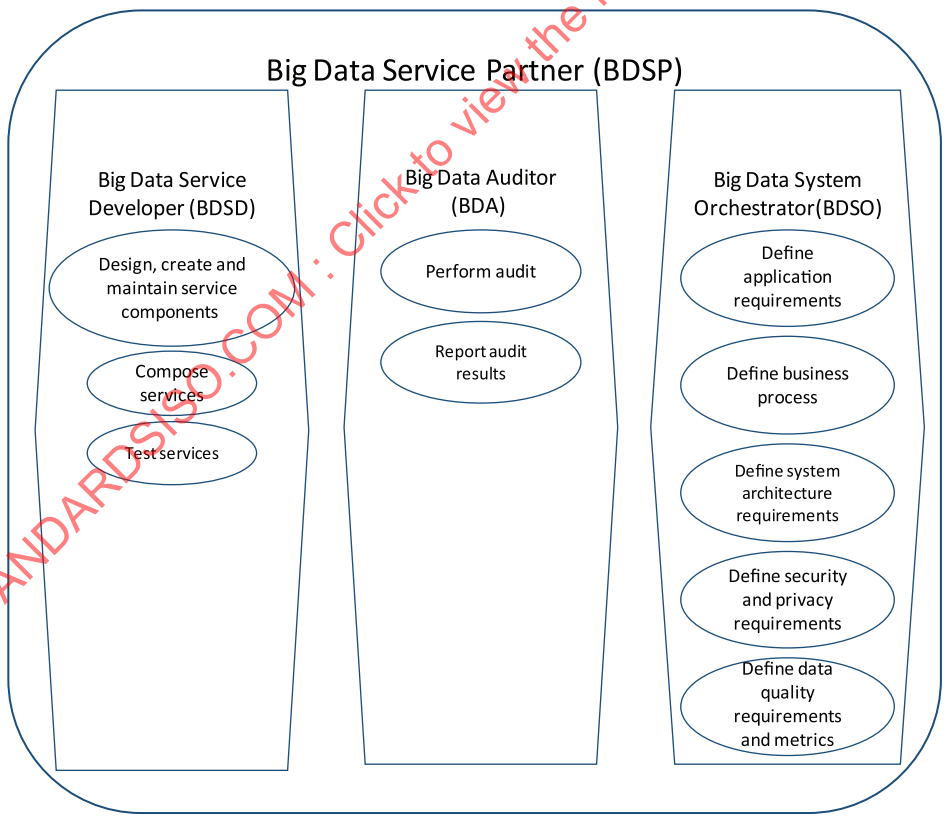



**Figure 8 — Big data activities relating to big data service partner sub-roles**

### 7.4.2 Sub-role: big data service developer (BDSD)

The BDSD is a sub-role of BDSP which is responsible for designing, developing, testing and maintaining the implementation of a big data service. This can involve composing the service implementation from existing service implementations.

The BDSD activities are as follows:

— the **design, create and maintain service components** activity involves designing and creating software components that are part of the implementation of a big data service, and providing fixes or enhancements to service implementations;

— the **compose services** activity is focused on composing services using existing services by intermediation, aggregation of them;

— the **test services** activity focuses on testing the components and services created by the big data service developer.

### 7.4.3 Sub-role: big data auditor (BDA)

The BDA is a sub-role of BDSP with the responsibility of conducting an audit of the provision and use of big data services. A big data audit covers veracity of data sources, operations, performance, security and privacy, and examines whether a specified set of audit criteria are met.

NOTE 1 There is a variety of specifications for the audit criteria, for example, addresses security considerations[8].

The BDA activities are as follows:

— the **perform audit** activity involves requesting or obtaining audit evidence, conducting any required tests on the system or data being audited and obtaining evidence programmatically;

— the **report audit results** activity involves providing a documented report of the results of performing an audit.

NOTE 2 The BDA is responsible for the assessment of data quality, the definition and evaluation of data quality service levels, the continuous measurement and surveillance of data quality.

### 7.4.4 Sub-role: big data system orchestrator (BDSO)

The BDSO is a sub-role of BDSP which provides the overarching requirements that the system should fulfil, including policy, governance, architecture, resources, and business requirements, as well as monitoring activities to ensure the system complies with those requirements.

The BDSO activities are as follows:

— the **define application requirements** activity deals with the overarching requirements that big data application should fulfil;

— the **define business process** activity deals with a partially ordered set of enterprise activities that can be executed to realise a given objective of an enterprise or a part of an enterprise to achieve some desired end-result;

— the **define system architecture requirements** activity deals with conceptual requirements for defining the structure, behaviour, and view of a big data system;

— the **define security and privacy requirements** activity is focused on defining security and privacy requirement from a governance point of view;

— the **define data quality requirements and metrics** activity is focused on development and the promotion of data quality awareness, and the definition of the data quality business rules, requirements, metrics.

## 7.5 Role: big data provider (BDP)

A big data provider (BDP) makes data available to itself or to others. In fulfilling its role, the BDP creates an abstraction of various types of data sources such as raw data or data previously transformed by another system and makes them available through different functional interfaces.

NOTE 1 The concept of a data provider is not new, the greater data collection and analytics capabilities have opened up new possibilities for providing valuable data.

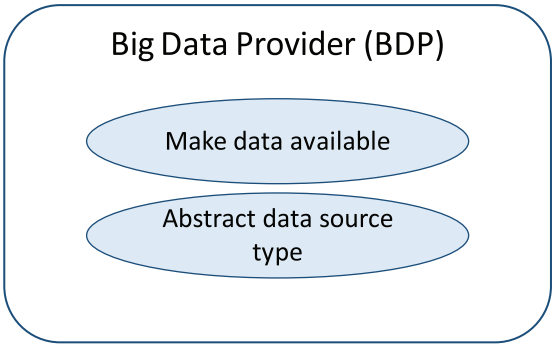



**Figure 9 — Big data activities relating to big data provider**

The BDP activities are as follows (see Figure 9):

— the **make data available** activity is focused on opening or distributing data source to the outside of the originally purposed system;

— the **abstract data source type** activity involves publishing metadata or data catalogue for the purpose of distributing data through a registry.

NOTE 2 When making data available to others, the big data provider can monitor the data and may manage the identified data quality issues according to the data quality management.

## 7.6 Role: big data consumer (BDC)

A big data consumer (BDC) receives the output of the big data system. In many respects, it is the recipient of the same type of functional interfaces that the big data provider (BDP) exposes to the big data application provider (BDAP). After the system adds value to the original data sources, the BDAP then exposes that same type of functional interfaces to the big data consumer (BDC).

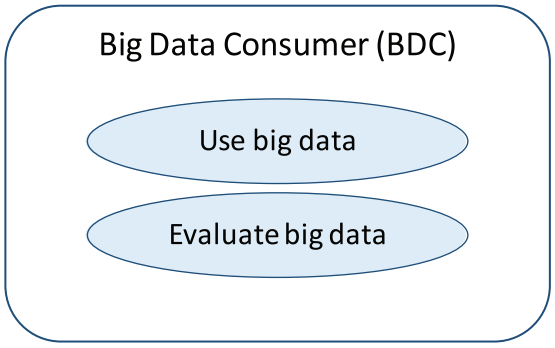



**Figure 10 — Big data activities relating to big data consumer**

The BDC activities are as follows (see Figure 10):

— the **use big data** activity focuses on using the results of big data analysis or utilizing application interfaces provided by big data application provider for big data consumer's business purposes;



— the **evaluate big data** activity involves rating the quality of big data or big data application as feedback.

# 8 Cross-cutting aspects

## 8.1 General

Cross-cutting aspects include:

— **security and privacy:** this aspect relates to how systems and data are secured by preserving their confidentiality, integrity and availability from risk and how personally identifiable information (PII) are protected from unauthorized use;

— **management**: this aspect relates to how system components and resources are provisioned, configured, utilized, and monitored;

— **data governance:** this aspect relates to how data is controlled and managed within the system over its lifecycle.

## 8.2 Security and privacy

Security and privacy issues affect all other roles and sub-roles in big data ecosystem and functional components of the BDRA. The security and privacy interacts with the Big Data System Orchestrator for policy, requirements, and auditing and also with both the big data application provider and the big data framework provider for development, deployment, and operation.

Security related considerations in big data include:

— **confidentiality** which ensures that systems and data are not made available or disclosed to unauthorized individuals, entities, or processes;

— **integrity** which ensures that systems and data are accurate and complete;

— **availability** which ensures that systems and data are accessible and usable on demand by an authorized entity.

Privacy related considerations in big data include:

— **unlinkability** which ensures that a PII principal may make multiple uses of resources or services without others being able to link these uses together;

— **transparency** which ensures that an adequate level of clarity of the processes in privacy-relevant data processing is reached so that the collection, processing and use of the information can be understood and reconstructed at any time;

— **intervenability** which ensures that PII principals, PII controller, PII processors and supervisory authorities can intervene in all privacy-relevant data processing. (See ISO/IEC 20547-4[28], ISO/IEC 27000[29].)

## 8.3 Management

The big data characteristics of volume, velocity, variety, and variability demand a versatile system and software management platform for provisioning, software and package configuration and management, along with resource and performance monitoring and management. Big data management involves system, data, security, and privacy considerations at scale, while maintaining a high level of data quality and secure accessibility.

Management related considerations in big data include the following.

— **Provisioning**: provisioning is the act of configuring system resources to support a specific task. Provisioning can take place at multiple levels across the system architecture from allocation of resources for virtual machines to allocation of resources for a specific job on one or more nodes. These considerations involve efficient use and configuration of resources to support one or more tasks.

— **Configuration**: these considerations involve the proper setting of parameters within system elements for optimal execution and use of system resources.

— **Package management**: these considerations involve management of the package baselines for system components to maintain security and operational reliability of the system.

— **Resource management**: these considerations involve how resources within the system are being utilized to support the various workloads supported by the system based on their priority.

### 8.4 Data governance

Data governance is a property or an ability that needs to be coordinated and implemented by set of activities of roles and sub-roles in the user view charge of guaranteeing that data used in the business processes creates value and effectively responds to the business needs.

Data governance plans and defines:

— an **organizational strategy** related to the management of data in order to ensure that data is aligned with the business;

— a **data quality management strategy** which is a set of constraints and actions aimed to ensure that data meets the quality needs defined by the business (see Annex C for more detail).

## 9 Functional view

### 9.1 Functional architecture

#### 9.1.1 General

The functional architecture for big data describes big data in terms of a high-level set of functional components layers. The functional layers represent sets of functional components with similar capabilities that are required to perform the big data activities described in Clause 8 for the various roles and sub-roles involved in big data the specification and implementation of a big data architecture.

The functional architecture describes functional components in terms of a layering architecture where specific types of functions are grouped into each layer as illustrated in Figure 12.

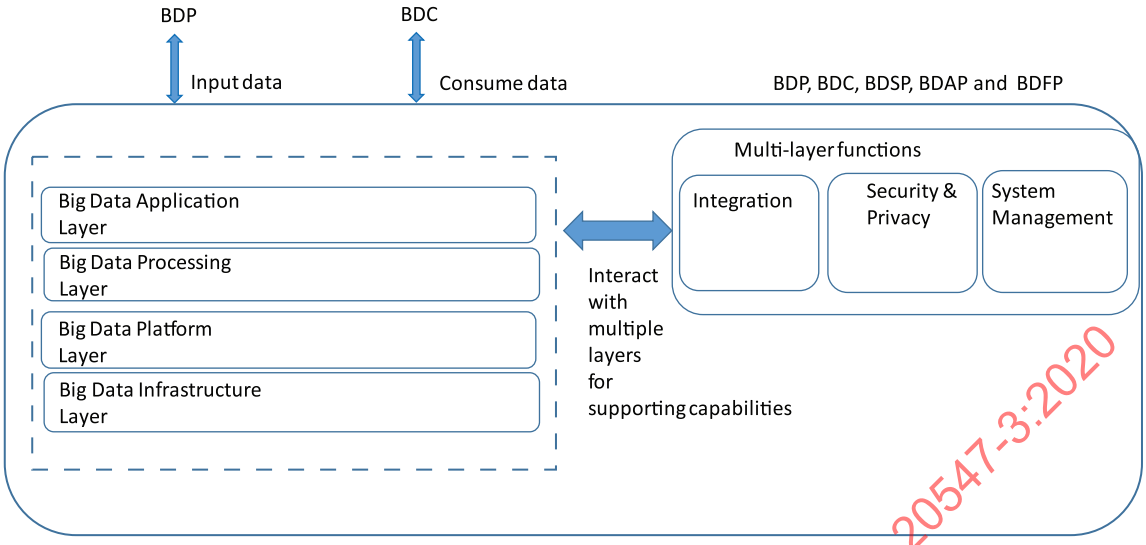



**Figure 11 — BDRA layer-based architecture**

The BDP and BDC shown above can be external to the big data system under architecture development or internal components (since one application provider in the big data architecture can provide input to or consume input from another application provider within the architecture). Annex A provides additional information on mapping the big data reference architecture functional view to other system integration reference architecture.

Big data user view roles and activities including BCP, BDC, BDSP, BDAP and BDFP are implemented by four-layered functions and/or multi-layer functions as shown in Figure 11. For the purposes of defining a specific architecture, recommended best practice is that the architect document the specific functional components that provide interfaces from those layers to the big data architecture.

### 9.1.2 Layering architecture

#### 9.1.2.1 General

The layering architecture used in the BDRA has four layers, plus a set of functions that span across the layers. The four layers are:

— big data application layer (see 9.1.2.2);

— big data processing layer (see 9.1.2.3);

— big data platform layer (see 9.1.2.4);

— big data infrastructure layer (see 9.1.2.5).

The functions that span the layers are called the multi-layer functions.

The layering architecture is shown in Figure 11 and each of the internal layers in the layering architecture is described in 9.1.2.2 to 9.1.2.5.

#### 9.1.2.2 Big data application layer

The big data application layer provides the application supporting functionalities, including data collection, preparation, analytics, visualization and access functions. These functions are achieved through interfaces with the BDP, the big data processing layer, the big data platform layer, and the BDC.

#### 9.1.2.3 Big data processing layer

The big data processing layer provides framework and library components to execute the analytics specified by the application provider layer. Within this layer, the components manage execution of the analytic tasks across the system. The components often interact with the platform layer to determine where data is stored on the system and direct the analytics for that data to the corresponding node in order to provide data locality to the computations. They also interact with resource management components within the multi-layer functions to balance computations across the system.

#### 9.1.2.4 Big data platform layer

The big data platform layer provides storage and organization components for the data processed by the system. These components draw on resources from the resource layer and, in the case on in-memory storage, coordinate with the resource management components in the multi-layer functions for the resources required. The platform layer components focus primarily on providing efficient organization of the data for access from the application provider and processing layers within the system.

#### 9.1.2.5 Big data infrastructure layer

The big data infrastructure layer is where the resources reside. This includes equipment typically used in a data centre such as servers, networking switches and routers, storage devices, and also the corresponding non-big data-specific software that runs on the servers and other equipment such as host operating systems, hypervisors, device drivers and generic systems management software.

The big data infrastructure layer also represents and houses the big data transport network functionality which is required to provide underlying network connectivity between the big data application provider and the BDP/BDC, as well as within the big data application provider and between peer big data application providers.

### 9.1.3 Multi-layer functions

The multi-layer functions include a series of functional components that interact with functional components of the above four other layers to provide supporting capabilities including and not limited to:

— security systems capabilities (authentication, authorization, auditing, validation, encryption);

— integration capabilities (linkage of different components to achieve the required functionality);

— management capabilities (deployment, configuration, monitoring, multi-tenancy resource, high-availability, and big data lifecycle).

The multi-layer functions described above may support cross cutting aspects or activities from roles that have system architecture wide applicability.

## 9.2 Functional components

### 9.2.1 General

This subclause describes the big data architecture in terms of the common set of big data functional components. A functional component is a functional element of the BDRA which is used to perform an activity or some part of an activity and which has an implementation artefact in a concrete realization of the architecture, e.g. a software component, a subsystem or an application.

Figure 12 presents a high-level overview of the BDRA functional components organized by means of the layering architecture.



The term framework as used for functional component names within Figure 12 and associated text clauses is defined in ISO/IEEE 11073-10201 as "a structure of processes and specifications designed to support the accomplishment of a specific task".

NOTE Given the range of applications/domains involved in big data and the rapid evolution of big data technologies, describing an exhaustive list of possible functional components within these layers is both voluminous and can never be complete. Therefore, only general categories of components are presented here.

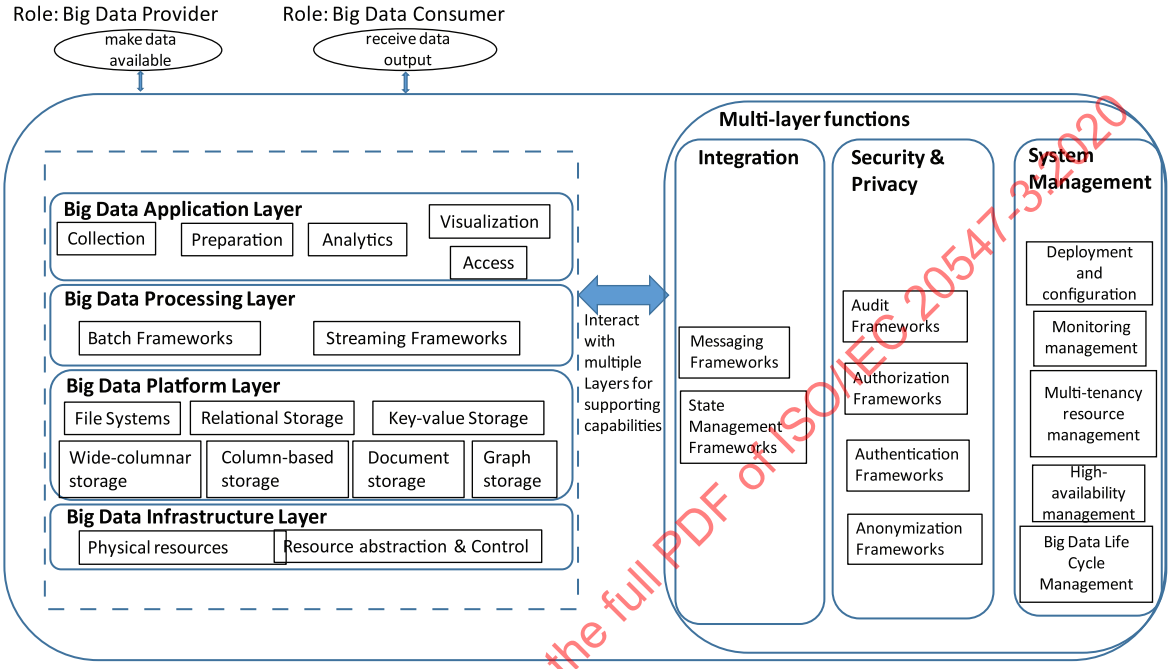



**Figure 12 — Functional components of BDRA**



### 9.2.2 Big data application layer functional components

#### 9.2.2.1 General

The big data application layer with the functional components supports the activities of the big data application provider. It provides the primary interface to external components including big data providers and big data consumers. The components here invoke components in the big data processing layer and big data platform layer to implement the big data application layer activities. The following are the primary functional components within this layer.

#### 9.2.2.2 Collection functional component

The collection functional component is used to establish the mechanisms to import data from big data provider and store data for further processes:

— establish connection;

— import data;

— store data.

This category of component is concerned with getting data into the system. These components can efficiently implement their functions given the volume and velocity of the incoming data.

#### 9.2.2.3 Preparation functional component

The preparation functional component is used to prepare data fit for a specific analysis process. The detailed functionalities consist of data aggregation, data cleansing, data conversion/transformation, data calculation field creation, data optimization, data partition, data summarization, data alignment, data validation, data virtualization and store prepared data. Data virtualization is an approach to data management where an application can access and change data without knowledge of the physical formatting and storage of the data. Data transformation changes data from one format to another, which includes encryption/decryption, compression/decompression, decimation, pivots, and normalization on data.

#### 9.2.2.4 Analytics functional component

The analytics functional component is used to encapsulate the specialized computations that need to take place on the data for information finding and/or knowledge extraction to meet the applications requirements by using specified algorithms.

NOTE 1 Classes of algorithms for machine learning include but are not limited to correlation, classification, data fusion, data integration, data mining, artificial intelligence, pattern recognition, predictive modelling, regression, cluster analysis, spatial analysis, audio analysis, visual analysis, textual analysis, etc. Text analysis algorithms include sentiment analysis, named entity recognition, and theme detection. Machine learning algorithms include correlation, classification, pattern recognition, predictive modelling, regression, cluster analysis, and spatial analysis. In many cases, big data systems combine several of these types of algorithms into a work flow on the data. For example, a system can use named entity recognition to extract specific entities (people, places, orgs, etc.) from unstructured chunks of text then feed that information as features into a K-nearest neighbour or K-means clustering algorithm to categorize the text chunks.

NOTE 2 A class of analytics function is operational data analysis, i.e. the analysis of log files, system status data, alert information, etc., for system operation and maintenance. Typical query and analysis include log text file search, multi-dimensional aggregation analysis, etc. Numerical analysis algorithms include fast Fourier transforms, linear algebra, and N-Body methods. graph algorithms include community detection, subgraph/motif finding, finding diameter, clustering coefficient, page rank, maximal cliques, connected component, betweenness centrality, shortest path.

NOTE 3 The critical characteristics of these algorithms for big data are that they need to be able to operate in parallel within the processing layer and deal with the distributed nature of the data within the platform layer.

#### 9.2.2.5 Visualization functional component

The visualization functional component is used to present analyzed data to the big data consumer in a meaningful manner. The detailed functionalities consist of:

— exploratory data visualization (multi-dimension, multi-resolution, interaction, animation, simulation, statistical graphics, surface rendering, volume rendering);

— knowledge/explanatory visualization (reports and customer summarization presentation).

NOTE The critical aspects of visualizing big data is presenting large datasets in a manner that can be easily navigated and is comprehensible. In addition, it can need to operate on the data in a distributed parallel fashion.

#### 9.2.2.6 Access functional component

The access functional component is used to provide big data consumer access to the results of big data application layer. The detailed functionalities consist of:

— access rights management;

— data export (e.g. via application programming interface, protocol or query language); and

— secure data access.

NOTE Big data consumers connect through this functional component by web services, user interfaces, and/or APIs, protocols, etc. which are used to access/retrieve data. The unique issue for big data here involves how to present the data to the big data consumer considering the big challenge of volume and velocity aspects.

### 9.2.3 Big data processing layer functional components

#### 9.2.3.1 General

The big data processing layer components are primarily focused on performance (e.g. producing results of computations within the requisite period of time). The big data processing layer provides functional components to mainly support big data characteristics of volume, velocity and variety. The big data processing layer adopts different processing engines on different data storage and schedule computation on near or local storage. This layer provides abstraction functionalities for the operations of big data application layer. User operation is abstracted as data source, filter, map, window, aggregation, etc. The big data processing layer completes the execution process with data flowing from one operator to another, and from input to output. Parallel data processing is implemented in this layer.

NOTE 1 In traditional database systems, the big data processing layer components are called the execution engine. The big data processing layer is more about runtime. The keyword "big" means not only the big data from the source; the intermediate data can be bigger than raw data.

NOTE 2 In parallelizing operations, the processing layer components typically allocate work to nodes in the cluster first based on data locality (e.g. the data in the platform layer needed for the computation is on the node) and then based on memory and CPU resources.

NOTE 3 An example of this is the map/reduce programming pattern, where computation on individual records is distributed to nodes based on data locality during the map phase and then the results from each node are merged and sorted during the reduce phase.

The big data processing layer adopts different processing engines on different data storage and schedule computation on near or local storage.

Typically, frameworks within the big data processing layer are categorized based on how many elements and how fast they process. The common forms of evaluation are one block (batch) or one element (streaming).

#### 9.2.3.2 Batch frameworks functional component

The batch frameworks functional component mainly aims to solve the problem of volume. It takes a batch of elements as the basic unit to process. Elements that were received are blocked to form a batch based on their distribution within the platform layer for processing to maximize data locality. After each node has processed its batch, the results are, synchronously or asynchronously, forwarded to the next step which can be another loop of processing (as done in the bulk synchronous parallel pattern) or summarization of the results (as done in the map/reduce pattern). The time required for a batch analytic to complete can vary from hours to sub-seconds depending on analytic and data. Ad-hoc query and daily operational analysis report applications can need different response times. When the response time is within minutes, hours or longer-range level, it is often referred to as offline processing. When it is within seconds or sub-second range, it is referred to as interactive processing. However, just because a system is designed to be interactive does not mean all response times are in the seconds or sub-seconds range. A poorly written analytic/query, one that has complex joins between data or one that simply must process a large volume of records can take minutes or hours to complete.

#### 9.2.3.3 Streaming frameworks functional component

##### 9.2.3.3.1 General

The streaming frameworks functional component mainly aims to solve the problem of velocity. The process model is pipelined and every element is forwarded to next operator with minimal latency. The instant response is the main concern and every element is valuable in the moment. while some

operations require elements to be blocked or buffered, for example to perform sliding window aggregations. However, in an ideal situation, the data flows continuously through the processing pipeline. The messaging frameworks functional component (see 9.2.6.2.2) is used to communicate between operators across nodes. When the data is too big and/or fast for the system to keep up, the system may employ temporary storage, choose to drop excess data or be forced to employ a rate limit mechanism with the producer to avoid a system crash.

The basic characteristic of streaming framework is data flow. The data flow is internally a directed acyclic graph that contains operator as vertex and event stream as edge. The operator can be parallelized and the event stream can be partitioned. Complex event processing (CEP) is more advanced than pure streaming and can be queryable, which adds more practical characteristics: event ordering, event processing guarantee, state store, and stream partitioning/operator parallelism.

The four characteristics are described in 9.2.3.3.2 to 9.2.3.3.5.

#### 9.2.3.3.2 Event ordering

Event ordering is guaranteed by custom global timestamp or sequence id, both of which are marked by feeder. Event ordering can be handled by time or count. Event ordering is relative to windowed stream. When event time is used, the event ordering means the event should be evaluated in window operator in timestamp order. Out-of-order and delayed events should be reordered, discarded or immediately evaluated. When event count is used, the event ordering means the event should be evaluated in window operator in sequence id. Event time or sequence id should be monotonically increasing.

#### 9.2.3.3.3 Event processing guarantee

Events should be processed with fault-tolerance mechanism in the presence of failures. Especially when streaming is partitioned, operator is paralleled, and data is distributed. Data stored in memory and data stored as persisted storage in file system should be guaranteed in the window duration. Two important phases that should be given special attention are receiving before processing (Receiver) and committing after processing (Processor).

Event processing guarantees are typically divided into the following three classes:

— **at-most-once:** this class means the receiver phase should receive once from data source and does not need to maintain the offset received, and the processor phase is not guaranteed. The received event can be received but no results returned. This is simple and with low latency, but the correctness is not guaranteed.

— **at-least-once:** this class means the receiver phase can replay and receive an event multiple times and the processor phase can process the events repeatedly. All events can be received and processed, but the result may not be accurate. Additional manual offset maintenance mechanism should be supported to meet event replay, and a duplication mechanism can be supported to reduce repeated treatment. This adds extra overhead but can achieve low latency and a certain degree of guarantee.

— **exactly-once**: the event is received once and processed once, no lost and no replay. The receiver and processor phases are both guaranteed. The two phases both need independent fault tolerance and failure recovery mechanism to make atomic and durable storage. This adds high overhead due to frequent IO operation, but best guarantees correctness.

#### 9.2.3.3.4 State store

Typical streaming frameworks have a pipeline process model, while CEP over streaming frameworks need additional state to support the window operation which is for continuous query where the event is stored for a period of time to generate the window. In traditional CEP, the window is small and the event is stored in a buffer. While events in window can be massive in modern CEP for big data, state store can provide support for high volume streams. Extra storage is needed for fault tolerance and failure recovery, replication, write ahead log (WAL) and checkpoint are classic methods to solve the



problem, so state store can support distributed and ACID semantics in limited ways and the trade-off is in the performance and correction.

#### 9.2.3.3.5 Stream partitioning/operator parallelism

This characteristic relates to scalability. Stream and operator are executed in directed acyclic graph. The goal of streaming frameworks is paralleling the execution to the greatest extent. Streaming partitioning serves event distribution and operator parallel serves parallel computation. Scheduler makes parallel computation execute with local events. Stream partition by key (e.g. sensor id, user ID, account ID), and aggregation function are separately evaluated on partitioned stream. Stream metadata, communication coordination, dynamic resource allocation and push/pull fetch strategy are needed to help the stream partitioning in distributed environment. Operator parallelism is the need to meet fast computation, but more mechanism needs to help to coordinate global state (barrier, event order). Operators are often one by one, some operators are applicable to chained to reduce network communication overhead.

### 9.2.4 Big data platform layer functional components

#### 9.2.4.1 General

The components of the big data platform layer provide the service to store, organize and retrieve the data in support of the higher layers. Accordingly, this layer provides for the logical data organization and distribution combined with the associated access application programming interfaces (APIs) or methods. This may also include data registry and metadata services along with semantic data descriptions such as formal ontologies or taxonomies.

NOTE One aspect in architecting this layer is to select or improve the data organization and storage methods for high data utilization and better query or retrieval performance. Especially with the rapid increase of volumes of big data (in e.g. finance, banking, media, manufacturing industry) and service scenarios, the users call for enhanced performance for different queries and analyses with less duplication and redundancy in data storage.

Subclauses 9.2.4.2 to 9.2.4.8 describe the general categories of these components.

#### 9.2.4.2 File systems functional component

File systems organize chunks of data (typically defined as records) accessed as a named entity within a defined namespace. While local filesystems are often used within big data systems for storing intermediate data local to a processing node, distributed file systems are far more prevalent for persistent data storage. The difference being that distributed file systems manage the distributions and replication of data blocks across nodes and the namespace rather than being stored with the data is managed through a central name service often running in a master/slave or multi-master manner to provide fault tolerance.

Distributed file systems (also known as cluster file systems) seek to overcome the throughput issues presented by the volume and velocity characteristics of big data, combine I/O throughput across multiple devices (spindles) on each node, with redundancy and failover mirroring or replicating data at the block level across multiple nodes. The data replication of a distributed file system is specifically designed allow the use of heterogeneous commodity hardware across the big data cluster. Thus, if a single drive or an entire node should fail, no data is lost because it is replicated on other nodes and throughput is only minimally affected because that processing can be moved to the other nodes. In addition, replication allows for high levels of concurrency for reading data and for initial writes.

Distributed object stores (DOSs) (also known as global object stores) are a unique example of distributed file system organization. Unlike the approaches described above, which implement traditional file system hierarchy namespace approaches, DOSs present a flat namespace with a globally unique identifier (GUID) for any given chunk of data. Generally, data in the store is located through a query against a metadata catalogue that returns the associated GUIDs. The GUID generally provides the underlying software implementation with the storage location of the data of interest. These object

stores are developed and marketed for storage of very large data objects, from complete datasets to large individual objects (e.g. high-resolution images in the tens of gigabytes [GBs] size range).

#### 9.2.4.3 Relational storage functional component

In the relational storage model, data is stored as rows with each field representing a column organized into a table based on the logical data organization.

NOTE Big data implementations of relational storage models are relatively mature and have been adopted by a number of organizations. They are also maturing very rapidly with new implementations focusing on improved response time. Many big data implementations take a brute force approach to scaling relational queries. Essentially, queries are broken into stages but more importantly processing of the input tables is distributed across multiple nodes (often as a map/reduce job).

The actual storage of the data can be flat files (delimited or fixed length) where each record/line in the file represents a row in a table. Increasingly however these implementations are adopting binary storage formats optimized for distributed file systems. These formats often use block level indexes and column-oriented organization of the data to allow individual fields to be accessed in records without needing to read the entire record. Despite this, most big data relational storage models are still batch-oriented systems designed for very complex queries which generate very large intermediate cross-product matrices from joins so even the simplest query can require tens of seconds to complete.

#### 9.2.4.4 Key-value storage functional component

The principles of key-value stores underpin all the other storage and indexing models. From a big data perspective, these stores effectively represent random access memory models. While the data stored in the values can be arbitrarily complex in structure all the handling of that complexity should be provided by the application with the storage implementation often providing back just a pointer to a block of data. Key-value stores also tend to work best for 1-1 relationships (e.g. each key relates to a single value) but can also be effective for keys mapping to lists of homogeneous values. When keys map multiple values of heterogeneous types/structures or when values from one key need to be joined against values for a different or the same key then custom application logic is required. It is the requirement for this custom logic that often prevents key-value stores from scaling effectively for certain problems.

Key-value stores generally deal well with updates when the mapping is one to one and the size/length of the value data does not change. The ability of key-value stores to handle inserts is generally dependent on the underlying implementation. Key-value stores also generally require significant effort (either manual or computational) to deal with changes to the underlying data structure of the values. Distributed key-value stores are the most frequent implementation utilized in big data applications. One problem that shall always be addressed (but is not unique to key-value implementations) is the distribution of keys across over the space of possible key-values.

Specifically, keys shall be chosen carefully to avoid skew in the distribution of the data across the cluster. When data is heavily skewed to a small range it can result in computation hot spots across the cluster if the implementation is attempting to optimize data locality. If the data is dynamic (new keys being added) for such an implementation, then it is likely that at some point the data can require rebalancing across the cluster. Non-locality optimizing implementations employ various sorts of hashing, random, or round-robin approaches to data distribution and does not tend to suffer from skew and hot spots. However, they perform especially poorly on problems requiring aggregation across the dataset.

#### 9.2.4.5 Wide columnar storage functional component

Unlike traditional relational data that store data by rows of related values, columnar stores organize data in groups of like values. The difference here is subtle but in relational databases an entire group of columns are tied to some primary-key (frequently one or more of the columns) to create a record. In columnar stores, the value of every column is a key and like column values point to the associated rows. The simplest instance of a columnar store is little more than a key-value store with the key and value roles reversed. In many ways, columnar data stores look very similar to indexes in relational databases. In addition, implementations of wide columnar stores that follow the sparse, distributed



multi-dimensional sorted map model (where arbitrary byte arrays are indexed/accessed based on row and column keys) introduce an additional level of segmentation beyond the table, row and column model of the relational model, that is called the column family. Wide columnar stores add an additional dimension known as the column family.

#### 9.2.4.6 Column-based storage functional component

By organizing and storing the data by the columns (instead of by the rows in row-based stores), columnar databases are well-suited for big data applications which require a wide spectrum of analysis, such as multi-dimensional OLAP (online analytic processing) query, big and small scan query. Various column-based sorting, indexing and compression techniques, e.g. multi-dimensional indexing, dictionary coding, etc., can be applied to increase the query performance.

#### 9.2.4.7 Document storage functional component

Modern document stores have evolved to include extensive search and indexing capabilities for structured data and metadata and that is why they are often referred to as semi-structured data stores. Within a document-oriented data store each document encapsulates and encodes the metadata, fields, and any other representations of that record. While somewhat analogous to a row in a relational table, one reason document stores have evolved and gained in popularity is that most implementations do not enforce a fixed or constant schema. While best practices hold that groups of documents should be logically related and contain similar data, there is no requirement that they be alike or that any two documents even contain the same fields. That is one reason that document stores are frequently popular for datasets which have sparsely populated fields, since there is far less overhead normally than traditional RDBMS systems where null value columns in records are actually stored. Groups of documents within these types of stores are generally referred to as collections and like key-value stores some sort of unique key references each document.

#### 9.2.4.8 Graph storage functional component

While social networking sites have certainly driven the visibility of and evolution of graph stores (and processing as discussed below), graph stores have been a critical part of many problem domains from military intelligence and counter terrorism to route planning/navigation and the semantic web for years. Graph stores represent data as a series of nodes, edges, and properties on those. Analytics against graph stores include very basic shortest path and page ranking to entity disambiguation and graph matching.

Graph storage approaches can actually be viewed as a specialized implementation of a document storage scheme with two types of documents (nodes and relationships). In addition, one of the most critical elements in analyzing graph data is locating the node or edge in the graph where the analysis is to begin. To accomplish this, most graph databases implement indexes on the node or edge properties. Unlike, relational and other data storage approaches, most graph databases tend to use artificial/pseudo keys or guides to uniquely identify nodes and edges. This allows attributes/properties to be easily changed due to both actual changes in the data (someone changed their name) or as more information is found out (e.g. a better location for some item or event) without needing to change the pointers to/from relationships.

Typically, distributed architectures for processing graphs assign chunks of the graph to system nodes then the system nodes use messaging approaches to communicate changes in the graph or the value of certain calculations along a path. Even small graphs quickly elevate into the realm of big data when one is looking for patterns or distances across more than one or two degrees of separation between graph nodes.

Depending on the density of the graph, this can quickly cause a combinatorial explosion in the number of conditions/patterns that need to be tested. A specialized implementation of a graph store known as the resource description framework (RDF) is part of a family of specifications from the World Wide Web Consortium (W3C) that is often directly associated with semantic web and associated concepts. RDF triples, as they are known, consist of a subject (Mr X), a predicate (lives at), and an object (Mockingbird Lane). Thus, a collection of RDF triples represents a directed labelled graph. The contents of RDF stores are frequently described using formal ontology languages like OWL or the RDF Schema (RDFS)

language, which establishes the semantic meanings and models of the underlying data. To support better horizontal integration (Smith, et al., 2012)[16] of heterogeneous datasets extensions to the RDF concept such as the data description framework (DDF) (Yoakum-Stover & Malyuta, 2008)[17] have been proposed which add additional types to better support semantic interoperability and analysis. Graph data stores currently lack any form of standardized APIs or query languages. However, the W3C has developed the SPARQL query language for RDF which is currently in a recommendation status and there are several systems such as Sesame which are gaining popularity for working with RDF and other graph-oriented data stores.

### 9.2.5 Resource layer functional components

#### 9.2.5.1 General

The resource layer functional components include:

— resource abstraction and control;

— physical resources.

#### 9.2.5.2 Resource abstraction and control functional component

The resource abstraction and control functional component is used by big data application providers (BDAPs) to provide access to the physical computing resources through software abstraction. Resource abstraction needs to ensure efficient, secure and reliable usage of the underlying infrastructure. The control feature of the functional component enables the management of the resource abstraction features.

NOTE 1 When the big data system is deployed within a cloud computing environment the resource abstraction functions are be provided by the cloud computing environment as defined in ISO/IEC 17789[6].

The resource abstraction and control functional component enables big data application providers (BDAPs) to offer qualities such as rapid elasticity, resource pooling and on-demand self-service. The resource abstraction and control functional component can include software elements such as hypervisors, virtual machines, virtual data storage, and time-sharing.

For the network, these are the resources that transfer data from one component to another within the infrastructure layer. Besides, the network infrastructure can also include automated deployment, provisioning capabilities, or agents and infrastructure-wide monitoring agents that are leveraged by the management/communication elements to implement a specific model.

For computing, the logical distribution of cluster/computing infrastructure can vary from a dense grid of physical commodity machines in a rack to a set of virtual machines running on a cloud service provider or to a loosely coupled set of machines distributed around the globe providing access to un-used computing resources.

NOTE 2 A hypervisor is a piece of computer software, firmware or hardware that creates and runs virtual machines. In this form, a hypervisor runs natively on the bare metal and manages multiple virtual machines consisting of operating systems (OS) and applications.

#### 9.2.5.3 Physical resources functional component

The physical resources functional component represents the elements needed by the big data application providers to run and manage the big data systems that they offer.

Physical resources include hardware resources, such as computers (CPU and memory), networks (routers, firewalls, switches, network links and network connectors, storage components (hard disks) and other physical computing infrastructure elements. These resources can include those that reside inside cloud data centres (e.g. computing servers, storage servers, and intra-data centre networks), and those that reside outside of data centres, typically networking resources, such as inter-data centre networks and core transport networks.



For the network, the volume and velocity characteristics of big data often are driving factors in the implementation of the internal and external connectivity of network infrastructure.

For computing, these are the physical servers that execute and hold the software of the other big data system components. Computing infrastructure also frequently includes the underlying operating systems and associated services used to interconnect the cluster resources via the networking elements.

For storage, these are resources that provide persistence of the data in a big data system. The storage infrastructure can include any resource from isolated local disks to storage area networks (SANs) or network attached storage.

These are the physical plant resources (power, cooling) that should be accounted for when establishing an instance of a big data system. While the resource components may be deployed directly on physical resources or on virtual resources, at some level all resources have a physical representation. Physical resources are frequently used to deploy multiple components that are duplicated across a large number of physical nodes to provide what is known as horizontal scalability. Virtualization is frequently used to achieve elasticity and flexibility in the allocation of physical resources and is often referred to as infrastructure as a service (IaaS) within the cloud computing community.

In this form, the accelerating units are resources that improve the efficiency for big data systems' computing, storage or transferring speed. The volume, variety and velocity of big data ask a higher and more flexible processing speed than the traditional way.

NOTE For example, the accelerating units for computing includes but not limited to graphics processing unit, customised gate array for acceleration by field-programmable gate array.

### 9.2.6 Multi-layer functional components

#### 9.2.6.1 General

The multi-layer functions include a series of functional components that provide services to the functional components in other layers.

#### 9.2.6.2 Integration layer functional components

##### 9.2.6.2.1 General

Integration layer functional components provide services to connect the functionality of the components in the same layer or across different layers.

The integration functional components may include but are not limited to:

— messaging frameworks (see 9.2.6.2.2);

— state management frameworks (see 9.2.6.2.3).

##### 9.2.6.2.2 Messaging frameworks functional component

Messaging frameworks functional component provides services, for example in the form of APIs, for message routing and exchange, including but not limited to the reliable queuing, transmission, and receipt of data between nodes in a horizontally scaled cluster, or components in the same or across different vertical layers defined in Figure 12. For example, a network resource in the resource layer can send information regarding its health status to the system management components via APIs offered by the messaging frameworks.

#### 9.2.6.2.3 State management frameworks functional component

The state management frameworks functional component is used by functional components to persist or maintain state across nodes in a distributed environment, to ensure state consistency and persistency lest resource or system failures occur. The persisted state information can be input to the system management components for resource monitoring or management.

### 9.2.6.3 Security and privacy layer functional components

#### 9.2.6.3.1 General

Security and privacy components are used to facilitate interoperability in BDRA without compromising privacy, confidentiality, or integrity. Security and privacy components are tightly coupled to all functional components via APIs.

NOTE Security and privacy components form a fundamental aspect of the reference architecture. This is geometrically spanning or cross-cutting main components, indicating that all components are affected by security and privacy considerations. Thus, the role of security and privacy is correctly depicted in relation to the components but does not expand into finer details, which can be more accurate but are best relegated to a more detailed security and privacy reference architecture. The following are the general categories of components implemented to support security and privacy aspects.

Security and privacy components interface and leverage a number of system management components to perform data capture and tracking.

#### 9.2.6.3.2 Audit framework functional component

The audit framework functional component is used by other components to record events within the system. Events can involve users, components, jobs and their actions run, stop, access data, update data, etc. These components often leverage platform layer components to record and persist their data but can, for security purposes, persist the data outside of the big data architecture. The audit trails or logs maintained by these components can be used to help track provenance of data, for recovery of data/state in the event of a system component failure, or to forensically analyse a system crash or incursion.

#### 9.2.6.3.3 Authentication framework functional component

The authentication frameworks functional component provides access control to underlying data and services within other components and also access to the system as a whole from external elements. Authentication involves the presentation of an identifier (e.g. user name) and an access key or keys (e.g. password or certificate) that is verified against a reference store. Typically, the component being authenticated communicates with the component they wish to access providing the identifier and key. The accessed component then invokes the authentication services and receives an answer as to whether to allow or reject the access. While, ideally, authentication services should be centralized within a single component, the prevalence of multiple components across all the layers can involve different layers or components requiring different authentication components.

#### 9.2.6.3.4 Authorization framework functional component

The authorization frameworks functional component supports mapping a user or component identifier to the privileges they have in accessing resources (both data and processing) within the cluster.

NOTE Examples of privileges that can apply to any given resource or element within the cluster are read or access, write, delete, execute, traverse and terminate.

The privileges can apply at different granularities within the resource. For example, many big data platforms are now implementing field/element level access control as opposed to record or file/dataset level control.