An Assurance Process for Big Data Trustworthiness

Marco Anisetti[a], Claudio A. Ardagna[a], Filippo Berto[a]

[a] *Universita' degli Studi di Milano, Milan, Italy*

Abstract

Modern (industrial) domains are based on large digital ecosystems where huge amounts of data and information need to be collected, shared, and analyzed by multiple actors working within and across organizational boundaries. This data-driven ecosystem poses strong requirements on data management and data analysis, as well as on data protection and system trustworthiness. However, although Big Data has reached its functional maturity and represents a key enabler for enterprises to compete in the global market, the assurance and trustworthiness of Big Data computations (e.g., security, privacy) are still in their infancy. While functionally appealing, Big Data does not provide a transparent environment with clear non-functional properties, impairing the users' ability to evaluate its behavior and clashing with modern data-privacy regulations. In this paper, we present an enhanced assurance process for Big Data, which aims to increase transparency and trustworthiness of Big Data computations. The assurance process evaluates Big Data computations at all layers from the specific Big Data pipelines to the Big Data ecosystem underneath.

Keywords: Non-functional Assurance, Big Data Transparency, Trustworthiness, Security, Distributed Systems, Monitoring

1. Introduction

We live and operate in a data-driven ecosystem where huge amounts of data are collected, shared, and analyzed by multiple actors working within and across organizational boundaries. The benefits brought by this data-driven ecosystem in terms of value, performance, and quality, come at the price of increasing security and privacy risks. Data in fact can be sensitive and need to be protected and secured once stored and while processed, following strict regulations such as the General Data Privacy Regulation (GDPR) in Europe.

A number of different solutions protect the Big Data infrastructures and their data/processes by internal and external threats and attacks, resulting in the proliferation of ad-hoc solutions that prove a specific property or compliance to a specific regulation [1, 2, 3]. Each solution targets a very small part of the whole problem [4, 5, 6, 7, 8], missing the full picture. For example, Terzi et al. [4] presented a survey on a global perspective of Big Data security and privacy, while Yakoubov et al. [5] specifically focused on cryptographic approaches. Zhang et al. [6] proposed a scalable differential privacy approach for Big Data multidimensional anonymization based on MapReduce. In addition, the research community has started approaching the problem of protecting machine learning algorithms and corresponding modes [9, 10] at the core of Big Data systems, while neglecting the protection of the whole systems.

The need of trust and transparency of Big Data computations is clearly raising, representing a major barrier against the adoption of Big Data technologies especially in a multi-tenant environment. Service providers (data transformers/analyzers) are reluctant to take full responsibility over security and privacy breaches of their services. Customers (data owners/suppliers) do not have access to all security intelligence and log information, which impairs their ability to estimate risks. There is no evidence that their computations and information are correctly managed and protected, as well as on the status of service security and correct behavior of security and privacy controls.

This paper is in line with our previous work [11], where we first try to address the above gaps but in the framework of DevOps processes applied to Big Data pipelines. In this paper, we depart from the development process adopted, and from the best of our knowledge we present the first attempt to address the general problem of *Big Data pipeline assurance*. Assurance is the way to gain justifiable confidence that *i)* one or more security properties are consistently demonstrated by the target of an assurance evaluation and *ii)* the target operationally behaves as expected, despite failures and attacks [12]. Applying assurance to Big Data is a complex process that evaluates the trustworthiness of all layers of the Big Data ecosystem: *i)* the Big Data pipeline and all its tasks, *ii)* the Big Data engine and all services over which the pipeline is executed. The goal of our assurance solution is to increase the trustworthiness of Big Data applications, mitigating the typical user distrust in Big Data environments.[1] This distrust is typically based on the fact that service providers and customers lose, at least partly, control over the status of their data and applications, and Big Data technologies and analytics provide blazing fast inference on such data. Few approaches have already focused on Big Data assurance, tackling specific aspects like data integrity or authentication [13, 14, 15]. Gao et al. [16] focused on Big Data quality assurance, and just partially considered security and privacy quality metrics. Presenting a comparison of Big Data validation tools, the paper underlines a set of needs including lack of well-

[1]Ernst and Young – `https://www.ey.com/en_es/assurance/how-big-data-and-analytics-are-transforming-the-audit`

defined quality validation and assurance standards, as well as lack of available research results on quality models/metrics and certification programs. Some initial assurance solutions have also targeted the need of verifying non-functional properties of machine learning models to the aim of implementing trustworthy decision systems [17].

The contribution of this paper is threefold. First, we propose a novel assurance process and architecture that evaluates the trustworthiness of a Big Data environment at all layers. Second, we define an assurance methodology where: *i)* clients annotate a pipeline template with assurance requirements modeling their trust expectations in terms of non-functional properties, *ii)* a pipeline instance is generated from the template mapping all tasks, services, and requirements on real components, *iii)* an assurance confidence level is calculated modeling the trustworthiness level of the Big Data pipeline. Third, we evaluated our approach in the context of the H2020 EVOTION Policy-Making Big Data Platform [18], where policy makers design or select analytics templates to be instantiated and executed by the platform in a fully assisted and privacy-preserving way.

We note that our approach complements the 5V definition of Big Data [19] where *veracity* (i.e., messiness or trustworthiness of the data) is extended with Big Data assurance, increasing the Big Data transparency and the soundness of Big Data computations [20].

The paper is organized as follows: Section 2 defines the assurance process and the software architecture automating its evaluation; Section 3 defines the abstraction over the data pipeline and the Big Data environment needed to represent the process; Section 4 introduces a practical scenario where security assurance techniques are applied to a live Big Data pipeline, describing its tasks and its ecosystem; Section 5 describes our assurance methodology; Section 6 shows our experimental pipeline setup, describing the checks we introduced, the performance we obtained and a comparison in several execution scenarios. Finally, Section 7 presents the related works, while Section 8 summarizes our work and the obtained results.

2. Assurance Process and Architecture

We define our notion of assurance and the corresponding Big Data assurance architecture.

2.1. Assurance Process

Non-functional assurance is the degree of confidence to which a system supports non-functional (e.g., security and privacy) requirements. The assurance process is possibly supported by a number of recommendations and security benchmarks, which drive its activities. For instance, let us consider an assurance process evaluating a requirement on data confidentiality both in transit and at rest. It insists on two *security controls* as the targets of evaluation implementing *i)* a mechanism for channel and storage encryption and *ii)* an access control mechanism mediating data ingestion. The assurance process measures how much the security controls contribute to data confidentiality and the degree of confidence held by such verification. To this aim, the configuration of the encryption algorithm (e.g., encryption algorithm and key length), on one side, and the configuration of the access control system (e.g., type of access control and soundness of defined policies), on the other side, are verified.

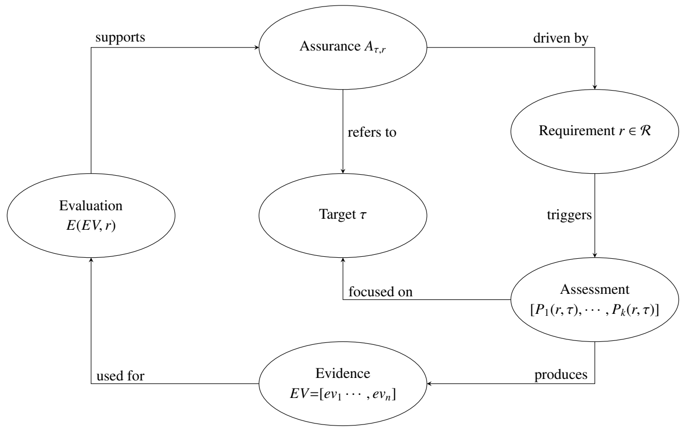

Figure 1: A view of the non-functional assurance process.

Figure 1 shows a view of our assurance process, which is driven by a set of requirements $\mathcal{R}$ and refers to a specific target τ. It is based on two sub-processes, namely, assessment and evaluation, described in the following.

Definition 2.1 (Assessment process). *Given a non-functional requirement $r \in \mathcal{R}$ and a target τ, an assessment process $ev = P(r, \tau)$ is a process P that collects evidence ev to prove r on τ.*

An assessment process is usually based on testing, monitoring, or formal methods [21, 22, 23], and produces an evidence in the form of test cases results, monitored events, and formal proofs, respectively. For instance, let us consider the encryption control for requirement confidentiality of data in transit. An assessment process can recurrently test whether the selected encryption algorithm follows specific standards/regulations. Service configurations can be then verified (e.g., by parsing the configuration files) to evaluate requirements on the key length. We note that multiple assessment processes P_i can be executed to prove a single requirement $r \in \mathcal{R}$ and, in turn, multiple evidence ev_i (possibly of different types) can be collected.

An evaluation process builds on the collected evidence and is executed as follows.

Definition 2.2 (Evaluation process). *Given a set of evidence $EV = [ev_1, \ldots, ev_n]$ retrieved by a set of assessment processes $[P_1(r, \tau), \ldots, P_k(r, \tau)]$ executed on target τ and requirement r, an evaluation process can be defined as a function $E(EV, r)$ returning a positive result if evidence $ev \in EV$ supports r, a negative result, otherwise.*

The Evaluation Process returns a Boolean value claiming whether evidence is (not) enough to support a given requirement. The strength of the evidence supporting the claim can differ depending on different evaluation parameters such as the type of probe, and the number and quality of evidence collectors, to name but a few. While the approach used to compute

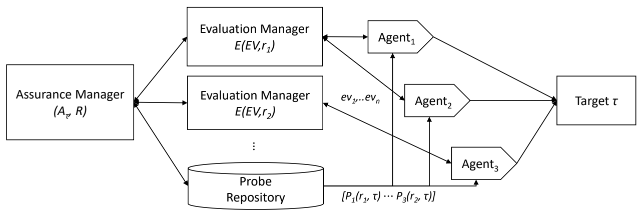

Figure 2: Assurance architecture.

whether a set of evidence ev supports or not a requirement r is out of the scope of this paper (a possible approach can be found in [24]), we consider the strength as a multiplier of the evaluation process results. For instance, if the evaluation is positive (i.e., = 1) but the strength is weak (e.g., = 0.3) the outcome of the evaluation process is weak itself (e.g., = 0.3).

Multiple evaluation processes E_i can be executed on specific subsets EV of evidence. Evaluation processes are the building blocks of our assurance process, which returns a single outcome as defined below.

Definition 2.3 (Assurance process). *An assurance process $A_{\tau,r}$ for a specific target τ is a process that takes as input a tuple $\langle EV, r\rangle$, where EV is the set of evidence and $r \in \mathcal{R}$ a requirement, and returns as output an assurance confidence level $l \in [0, 1]$.*

We note that Assurance Confidence Level l represents the level to which the collected evidence satisfies requirement r according to evaluation processes $E_i(EV_i \subseteq EV, r)$ (see Definition 2.2). Being the outcome of the evaluation processes $E_i \in [0, 1]$, the Assurance Confidence Level l is computed as follows.

Definition 2.4 (Assurance Confidence Level). *Given a set of evaluation results E_i, the assurance confidence level l is computed as follows: $l = \frac{|E_i>0|}{|E_i|} \cdot avg(E_i > 0)$ where $\frac{|E_i>0|}{|E_i|}$ is the frequency of positive evaluations and $avg(E_i > 0)$ is the average value of the positive evaluations.*

For instance, let us assume that *i)* a test-based assessment of the encryption service provides evidence satisfying the expectation on the algorithm and key length used for data encryption and *ii)* a monitoring-based assessment provides positive evidence on data ingestion (logs show that only authorized users can access ingested data). Collected evidence is evaluated following Definition 2.2 and two positive checks $E_1 = 1$ and $E_2 = 1$ retrieved. Confidence level is then computed according to Definition 2.4 as $l = 1 \cdot 1 = 1$ and returned by the assurance process $A_{\tau,r}$ in Definition 2.3.

The assurance process in Figure 1 is assumed to be trusted. In particular, we consider a chain of trust involving delegation [21], where the assessment is carried out by an assurance framework delegated by the service provider to execute the assurance evaluation.

2.2. Assurance Architecture

Figure 2 describes the architecture of our assurance framework implementing the assurance process in Figure 1. *Assurance Manager* is the owner of the assurance process. It is responsible to *i)* setup the evaluation processes on the given target τ according to a specific set $\mathcal{R}$ of requirements and *ii)* collect the evaluation process results used to compute the assurance confidence level. *Evaluation Manager* manages all activities for the requirement evaluation. It is responsible to *i)* setup the assessment process and *ii)* collect the evidence used to evaluate whether a requirement $r \in \mathcal{R}$ is supported. The assessment process is implemented instrumenting different *Assessment Probes* that executes the assessment process on τ and retrieves the corresponding evidence ev.

Probes Repository stores and maintains the probes, for instance, updating them in case new threats or guidelines are released. The probes are executed by *Assessment Agents* that are deployed and connected to the target of the assessment. We note that the probes in the probe repository can be configured by the assessment agents to inspect a specific target via specific target's hooks (e.g., APIs or the path of configuration files). They perform testing and monitoring activities, parse configuration files, execute code, and perform network traffic inspections (see Section 6).

When the assurance process verifies a set $\mathcal{R}$ of requirements, the following flow of actions is triggered. Assurance Manager instruments one or more Evaluation Managers with details on the evaluation to be executed, one for each requirement $r \in \mathcal{R}$. The Evaluation Managers communicate with the agents (deployed a priori) asking for the execution of probes addressing a specific assessment process. The agents download the requested probes from the repository and execute them against the target returning the collected evidence back to the corresponding Evaluation Manager. The Evaluation Managers evaluate the collected evidence and return the evaluation results to the Assurance Manager. Assurance Manager finally computes the confidence level.

3. Big Data Analytics Pipeline

We model a Big Data analytics pipeline as *i)* a set of tasks $t \in T$ implementing the *processing pipeline* p and *ii)* a set of services $s \in S$ implementing the ecosystem e and supporting the deployment and execution of the processing pipeline. We note that tasks t, services s, pipeline p, and ecosystem e are the targets τ of our assurance process. The pipeline and ecosystem are modeled at two levels of abstraction: *i) abstract level*, modeling the generic purposes of a task t and the abstract functionalities offered by a service s (template abstraction) and *ii) concrete level*, defining specific task implementations $\hat{t}$ in the pipeline and concrete services $\hat{s}$ used in the ecosystem (instance abstraction).

3.1. Processing Pipeline

A processing pipeline transforms data according to a specific goal. We assume our pipelines to be a sequential composition of tasks t.[2] Its abstract view can be defined as follows.

Definition 3.1 (p). *An Abstract Processing Pipeline p is defined using a BNF-like notation as*

$$\begin{aligned}
p &::= \langle T_I \oplus P \oplus A \oplus T_V \rangle \\
P &::= \epsilon \mid T_P \mid P \oplus T_P \\
A &::= \epsilon \mid T_A \mid A \oplus T_A \\
T_I &::= stream \mid fileSystem \mid DBMS \mid \ldots \\
T_P &::= cleaning \mid normalization \mid selection \mid \ldots \\
T_A &::= modeling \mid prediction \\
T_V &::= T_I \mid T_I \oplus visualization \mid
\end{aligned}$$

Operator $\oplus$ is the sequence operator connecting tasks' input and output in a pipeline fashion. A generic task t is classified according to its processing type: *i)* ingestion tasks $T_I \subset T$ (e.g., *stream*, *fileSystem*, *DBMS*) , *ii)* preparation tasks $T_P \subset T$ (e.g., *cleaning*, *normalization* and *selection*), *iii)* analytics tasks $T_A \subset T$ (e.g., *modeling*, *prediction*), *iv)* visualization tasks $T_V \subset T$ (e.g., *visualization*). We note that T_I and T_V are mandatory for p. We also note that ingestion tasks in T_I can be also used prior or replace the visualization in T_V.

An abstract processing pipeline p is instantiated in a concrete processing pipeline $\hat{p}$ as follows.

Definition 3.2 ($\hat{p}$). *Given an abstract processing pipeline p, a concrete processing pipeline $\hat{p}$ is produced by instantiating each generic task $t \in p$ in an executable task $\hat{t} \in \hat{p}$ in the form of a function call.*

We denote this instantiation process as $p \xrightarrow{p} \hat{p}$.

Example 3.1. *Let us consider an ingestion task $t_i \in T_I$, a preparation task $t_p \in T_P$ and a visualization task $t_v \in T_V$. An executable ingestion task $\hat{t}_i$ can ingest data from a queue system or files. An executable preparation task $\hat{t}_p$ can select data based on data columns/labels or apply a more sophisticated feature selection approach based on PCA or ICA. An executable visualization task $\hat{t}_v$ can save data to disk or send them to a visualization service.*

3.2. Big Data ecosystem

A Big Data ecosystem is composed of services $s \in S$ supporting the execution of the processing pipeline. Its abstract view is defined as follows.

Definition 3.3 (e). *An abstract Big Data ecosystem e is a 5-tuple $\langle S_I, S_C, S_S, S_V, S_E \rangle$, where $S_S \subset S$ is a set of storage services, $S_C \subset S$ is a set of computational services, $S_I \subset S$ is a set of ingestion services supporting data collection, $S_V \subset S$ is a set of visualization services supporting the visualization of the pipeline outcomes, and $S_E \subset S$ is a set of environmental services offering additional non-functional capabilities.*

[2]We note that more complex pipelines, including parallel or alternative tasks, can be generalized as a set of sequential pipelines.

We note that the ingestion services $s \in S_I$ (e.g., *streaming*, *load from batch*) are connected to tasks in T_I, while computational services $s \in S_C$ (e.g., *batch*, *stream*, *microbatch*) are connected to tasks in T_P and T_A to support preparation and analytics processing, and to tasks in T_I and T_V to support ingestion and visualization processing. Storage services $s \in S_S$ (e.g., *file storage*, *NoSQL storage*, *SQL storage*) are primarily connected to tasks in T_I and T_V and, if needed, to tasks in T_P and T_A to store/load temporary data during the preparation or analysis phases. Visualization services $s \in S_V$ (e.g., *dashboards*) are connected to tasks in T_V. Environmental services $s \in S_E$ (e.g., *access control*, *logging*, *annotation*, *authorization*) are connected to the entire pipeline.

Definition 3.4 ($\hat{e}$). *Given an abstract Big Data ecosystem e, a concrete Big Data ecosystem $\hat{e}$ is produced by instantiating each generic service $s \in e$ in a deployed service $\hat{s}$.*

We denote this instantiation process as $e \xrightarrow{e} \hat{e}$.

Example 3.2. *Let us consider Example 3.1 and a generic service $s_i \in e$ of type S_I, $s_c \in e$ of type S_C, and $s_v \in e$ of type S_V. s_i is instantiated into a service $\hat{s}_i$ deploying $HDFS$ and then used by the concrete ingestion task $\hat{t}_i$ in $\hat{p}$ to establish a connection and ingest data. s_c is instantiated into a service $\hat{s}_c$ based on $Spark$ computation engine used by the concrete preparation task $\hat{t}_p$ in $\hat{p}$ to elaborate ingested data. s_v is instantiated into a service $\hat{s}_v$ deploying $ApacheZeppelin(params)$ used by the concrete visualization task $\hat{t}_v$ in $\hat{p}$ to set up a Zeppelin visualization notebook where data can be visualized.*

3.3. Building a Big Data Analytics Pipeline

A Big Data Analytics Pipeline instantiates the abstract processing pipeline and Big Data ecosystem as follows.

Definition 3.5. *Given a pair $\langle p, e \rangle$, where p is the Abstract Processing Pipeline (Definition 3.1) and e is the corresponding Abstract Big Data Ecosystem e (Definition 3.3), a Big Data Analytics Pipeline is defined as a pair $\langle \hat{p}, \hat{e} \rangle$, where $\hat{p}$ is the concrete Processing Pipeline (Definition 3.2) and $\hat{e}$ is the corresponding concrete Big Data Ecosystem (Definition 3.4), such that $p \xrightarrow{p} \hat{p}$ and $e \xrightarrow{e} \hat{e}$.*

In the following, we refer to $\langle p, e \rangle$ as Big Data Analytics Pipeline template Π and to $\langle \hat{p}, \hat{e} \rangle$ as Big Data Analytics Pipeline instance I. Template Π defines technology-independent processing pipelines and environment services; instance I is the concrete technology-dependent instantiation of Π. We assume a correct instantiation function $\xrightarrow{I} = \xrightarrow{p} \cup \xrightarrow{e}$, for instance, by using the approach in [25]. In other words, for each template Π, we assume one or more consistent instances I, defining a one-to-many relation between templates and instances.

4. Reference Scenario

Our reference scenario is the H2020 EVOTION Policy Making Big Data Platform [18, 26], a collaborative solution offering analytics services or computing/data nodes based on different computation/storage frameworks. The platform, based on the Apache framework and Big Data Analytics-as-a-Service [25], offers an easy-to-use framework for policy makers to develop evidence-based policies following analytics results. Policy makers design or select specific analytics templates to be instantiated and executed by the platform in a fully assisted and privacy-preserving way. In our scenario, *i)* a policy maker c wants to execute a Big Data analytics pipeline continuously ensuring specific non-functional requirements $\mathcal{R}$ on the EVOTION Platform expressed in the template Π; *ii)* the EVOTION Platform as the service provider sp offers a Big Data analytics engine implementing a correct instance I for template Π.

In this context, assurance is key to increase the trust in the platform, especially in critical domains (e.g., health) where public policies use sensitive data. Policy maker c requests verifiable assurance from service provider sp that its non-functional requirements $\mathcal{R}$ on template Π are guaranteed on the running instance I. Such guarantees must be continuously checked by the assurance process to cope with emerging and evolving vulnerabilities and weaknesses. Non-functional requirements $\mathcal{R}$ have the same level of abstraction used for the template, being independent by the specific pipeline implementation and deployment ecosystem, and taken from a controlled vocabulary.

Example 4.1 (Running Example). *Let us consider a template $\Pi = \langle p, e\rangle$ in Definition 3.5 with $p = \langle$fileSystem $\oplus$ normalization $\oplus$ modeling(clustering) $\oplus$ fileSystem$\rangle$, and $e = \langle$LoadFilesystem, [BatchProcessing, Orchestration], StoreFilesystem, ϵ, AccessControl(AC)$\rangle$. We note that the visualization service is not needed in this example since it is focused on building a model. A concrete instantiation $\xrightarrow{I}$ of the above template Π into a* Big Data Analytics Pipeline Instance *I can be defined as $\langle \hat{p}, \hat{e}\rangle$ where $\hat{p} = \langle$loadFromHDFS() $\oplus$ normalization(all) $\oplus$ k-meansModeling(k) $\oplus$ saveToHDFS(model)$\rangle$ and $\hat{e} = \langle$Hadoop, [Spark, Airflow], Hadoop, [Knox, Ranger]$\rangle$.*

Table 1 summarizes the Example 4.1. The pipeline instance $\hat{p}$ defines an ingestion task (T_I) to ingest batches of data from HDFS ($\hat{t}_1$ = *loadFromHDFS()*) using the Hadoop service $\hat{s}_1$, performs a preparation task (T_P) normalizing all the data fields ($\hat{t}_2$ = *normalization(all)*) to fit into a modeling task (T_C) for k-means model creation ($\hat{t}_3$ = *k-meansModeling(k)*) based on Spark service $\hat{s}_2$. The model is then saved on HDFS ($\hat{t}_4$ = *saveToHDFS(model)*) using the Hadoop service $\hat{s}_1$ for later usage. The entire pipeline is orchestrated with Airflow service $\hat{s}_3$. To execute such pipeline an access control is requested using *Knox* (authentication) and *Ranger* (authorization), $\hat{s}_4$ and $\hat{s}_5$ respectively.

5. Assurance of Big Data Analytics Pipeline

Our assurance methodology is based on three sequential steps: *i) template annotation* that annotates the Big Data Analytics Pipeline Template (template in the following) with generic requirements, *ii) instance annotation* that annotates the Big Data Analytics Pipeline Instance (instance in the following) with specific requirements, and *iii) assurance evaluation* that evaluates the overall assurance.

Table 1: Running example: tasks, template Π and instance I.

Tasks in p and $\hat{p}$		
	t	$\hat{t}$
T_I	t_1 = *fileSystem*	$\hat{t}_1$ = *loadFromHDFS()*
T_P	t_2 = *normalization*	$\hat{t}_2$ = *normalization(all)*
T_A	t_3 = *modeling(clustering)*	$\hat{t}_3$ = *k-meansModeling(k)*
T_V	t_4 = *fileSystem*	$\hat{t}_4$ = *saveToHDFS(model)*

Services in e and $\hat{e}$		
	s	$\hat{s}$
S_I	s_1 = *LoadFilesystem*	$\hat{s}_1$ = *Hadoop*
S_C	s_2 = *BatchProcessing*	$\hat{s}_2$ = *Spark*
S_C	s_3 = *Orchestration*	$\hat{s}_3$ = *Airflow*
S_S	s_4 = *StoreFilesystem*	$\hat{s}_1$ = *Hadoop*
S_V	$s_5 = \epsilon$	
S_E	s_6 = *AC: Authentication*	$\hat{s}_4$ = *Knox*
S_E	s_6 = *AC: Authorization*	$\hat{s}_5$ = *Ranger*

$$\Pi = \langle p, e\rangle \quad \begin{array}{l} p = \langle t_1 \oplus t_2 \oplus t_3 \oplus t_4\rangle \\ e = \langle s_1, [s_2, s_3], s_4, s_5, s_6\rangle \end{array}$$

$$I = \langle \hat{p}, \hat{e}\rangle \quad \begin{array}{l} \hat{p} = \langle \hat{t}_1 \oplus \hat{t}_2 \oplus \hat{t}_3 \oplus \hat{t}_4\rangle \\ \hat{e} = \langle \hat{s}_1, [\hat{s}_2, \hat{s}_3], \hat{s}_1, [\hat{s}_4, \hat{s}_5]\rangle \end{array}$$

5.1. Template annotation

The template annotation is a process that annotates template $\Pi = \langle p, e\rangle$ with a set of non-functional requirements $r \in \mathcal{R}$ taken from two vocabularies $\mathcal{R}_p, \mathcal{R}_e \subset \mathcal{R}$, such that $\mathcal{R}_p \cup \mathcal{R}_e = \mathcal{R}$. We define two labeling functions i) $\lambda : T \rightarrow \mathcal{R}_p$ that associates each tasks $t \in T$ in pipeline p with a set of non-functional requirements $r \in \mathcal{R}_p$, *ii)* a labeling function $\gamma : S \rightarrow \mathcal{R}_e$ that associates each services $s \in S$ in the pipeline ecosystem e with a set of non-functional requirements $r \in \mathcal{R}_e$. We formally define an annotated Big Data analytics pipeline template as follows.

Definition 5.1 ($\Pi^{\lambda,\gamma}$). *An annotated Big Data analytics pipeline template is defined as $\Pi^{\lambda,\gamma}$ where λ and γ are two labeling functions such that:* i) *λ assigns labels $\lambda(t_i)$ corresponding to pipeline requirements in $\mathcal{R}_p$ to be satisfied by task t_i;* ii) *γ assigns a label $\gamma(s_i)$ corresponding to service requirements in $\mathcal{R}_e$ to be satisfied by service s_i.*

We note that labeling function λ (γ, resp.) can assign label $\lambda(p)$ ($\gamma(e)$, resp.) corresponding to pipeline (ecosystem, resp.) requirements in $\mathcal{R}_p(\mathcal{R}_e, resp.)$ to be satisfied by pipeline p (ecosystem e). We also note $\lambda(p)$ refers to requirements on the pipeline structure (e.g., the sequence of tasks), while $\gamma(e)$ refers to the environment where the services are deployed (e.g., the operating system or the service container).

Example 5.1 (Annotated template). *Following Example 4.1, a client specifies requirement Confidentiality in transit in $\mathcal{R}_p$ at* *template level. A complete annotation for requirement Confidentiality is discussed in Section 6. Template* Π *can be annotated as follows:*

- *All tasks* $(*)$ *in* $p \in \Pi$ *are annotated with* $\lambda(*)$ = *Confidentiality in transit, and the entire pipeline* $p \in \Pi$ *is annotated with* $\lambda(p)$ = *Pipeline integrity*
- *All services* $(*)$ *in* $e \in \Pi$ *are annotated with* $\gamma(*)$ = *Confidentiality in transit.*

We note that tasks, services, the pipeline and the service ecosystem can be annotated with zero or more requirements according to Definition 5.1. For instance, $\gamma(e)$ was empty in Example 5.1.

5.2. Instance annotation

The instance annotation is a process that annotates instance $I = \langle \hat{p}, \hat{e} \rangle$ with a set of concrete requirements $\hat{r} \in \hat{\mathcal{R}}$ taken from two vocabularies $\mathcal{R}_{\hat{p}}, \mathcal{R}_{\hat{e}} \subset \hat{\mathcal{R}}$, where $\hat{\mathcal{R}} = \mathcal{R}_{\hat{p}} \cup \mathcal{R}_{\hat{e}}$ is a specialization of $\mathcal{R} = \mathcal{R}_p \cup \mathcal{R}_e$. We define two labeling functions *i)* $\theta : \hat{T} \rightarrow \mathcal{R}_{\hat{p}}$ that associates each invocation of $\hat{t} \in \hat{T}$ in the pipeline $\hat{p}$ with a set of non-functional requirements $\hat{r} \in \mathcal{R}_{\hat{p}}$, *ii)* labeling function $\psi : \hat{S} \rightarrow \mathcal{R}_{\hat{e}}$ that associates each invocation of $\hat{s} \in \hat{S}$ in the ecosystem model $\hat{e}$ with a set of non-functional requirements $\hat{r} \in \mathcal{R}_{\hat{e}}$. Similarly to template annotation, we formally define an annotated Big Data analytics pipeline instance as follows.

Definition 5.2 ($I^{\theta,\psi}$). *An annotated Big Data analytics pipeline instance is defined as* $I^{\theta,\psi}$ *where* θ *and* ψ *are two labeling functions such that:* i) θ *assigns a label* $\theta(\hat{t}_i)$ *corresponding to pipeline requirements in* $\mathcal{R}_{\hat{p}}$ *to be satisfied by task* $\hat{t}_i$*;* ii) ψ *assigns a label* $\psi(\hat{s}_i)$ *corresponding to the service requirements in* $\mathcal{R}_{\hat{e}}$ *to be satisfied by service* $\hat{s}_i$*.*

We note that labeling function θ (ψ, resp.) can assign label $\lambda(\hat{p})$ ($\gamma(\hat{e})$, resp.) corresponding to pipeline (ecosystem, resp.) requirements in $\mathcal{R}_{\hat{p}}(\mathcal{R}_{\hat{e}}, resp.)$ to be satisfied by pipeline $\hat{p}$ (ecosystem $\hat{e}$).

An annotated instance $I^{\theta,\psi}$ is obtained by an annotated template $\Pi^{\lambda,\gamma}$ according to transformation function $\overset{R}{\rightarrow}$ as follows.

Definition 5.3 ($\overset{R}{\rightarrow}$). $\overset{R}{\rightarrow}$ *is a transformation function that receives as input the annotated template* $\Pi^{\lambda,\gamma}$ *and the pipeline instance I, and generates as output an annotated pipeline instance* $I^{\theta,\psi}$*, where:* i) $\Pi \overset{I}{\rightarrow} I$ *and* ii) *the generic pipeline requirements annotated with* $\lambda \in \Pi$ *are specialized into instance-specific requirements annotated with* $\theta \in I$ *and the generic ecosystem requirements annotated with* $\gamma \in \Pi$ *are specialized into instance-specific requirements annotated with* $\psi \in I$*.*

We note $\overset{R}{\rightarrow}$ can be either a manual or an automatic transformation. Any generic requirements $r \in \mathcal{R}$ can be specialized in one or more instance-specific requirement $\hat{r} \in \hat{\mathcal{R}}$, thus leading to one or more annotated pipeline instances $I^{\theta,\psi}$. For conciseness, we consider the instance annotation function $\overset{R}{\rightarrow}$ as given and carried out by the service provider on their instances a priori, according to the available templates. In case of multiple possible instantiations, the service provider selects the best one according to its deployment strategy.

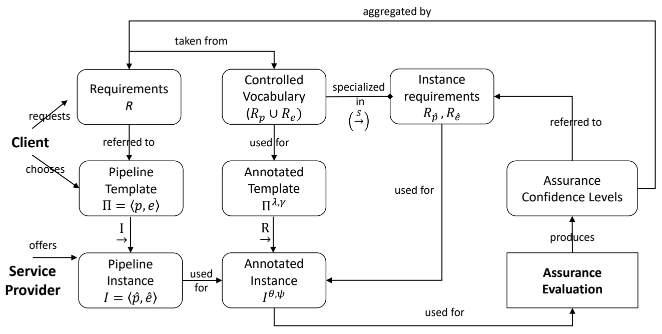

Figure 3: The Assurance methodology for Big Data Analytics Pipeline.

Example 5.2 (Annotated Instance). *Let us consider the annotated template* $\Pi^{\lambda,\gamma}$ *in Example 5.1. One possible annotated analytics pipeline instance* $I^{\theta,\psi}$ *can be as follows.*

- *The pipeline instance* $\hat{p} \in I$ *is annotated with* $\theta(*)$ = *Avoid connection to external services,* $\theta(*)$ = Avoid use of vulnerable libraries*, and* $\theta(\hat{p})$ = Pipeline Integrity
- *The ecosystem instance* $\hat{e} \in I$ *is annotated with* $\psi(\hat{s}_1)$ = *Encrypted HDFS and Inter-node communication security,* $\psi(\hat{s}_2)$ = *Inter-node communication security,* $\psi(\hat{s}_3)$ = *Orchestrator Confidentiality and* $\psi(\hat{s}_4)$ = *Communication channel security.*

We note that tasks $\hat{t}_1$ = *loadFromHDFS()* and $\hat{t}_4$ = *saveToHDFS(model)* are not affected by the request of confidentiality in transit expressed at template level in Example 5.1. This requirement is in fact considered by the ecosystem level only in the specific instance in Example 5.2 and addressed by $\hat{s}_1$ = *Hadoop*. In other words, annotation $\lambda(t_1)$ = *Confidentiality in transit* is transformed by the instance annotation function $\overset{R}{\rightarrow}$ to $\theta(\hat{t}_1) = \emptyset$ and $\theta(\hat{t}_4) = \emptyset$. We also note that confidentiality in transit is instantiated as $\theta(*)$ = *Avoid connection to external services* and $\theta(*)$ = *Avoid use of vulnerable libraries* for all tasks $(*)$ in the pipeline. We finally note that $\theta(\hat{p})$ = *Pipeline Integrity* refers to the verification of the pipeline structure and is associated with the corresponding requirement at service level $\psi(\hat{s}_3)$ = *Orchestrator Confidentiality*, while $\psi(\hat{s}_4)$ = *Communication channel security* refers to the authentication channel used by $\hat{s}_4$ (i.e., Knox).

5.3. Assurance evaluation

The assurance evaluation executes one or more assurance processes A in Definition 2.1 (using the architecture in Figure 2) on $I^{\theta,\psi}$ such that $\Pi^{\lambda,\gamma} \overset{R}{\rightarrow} I^{\theta,\psi}$. It can be formally defined as follows.

Definition 5.4 (Assurance Evaluation). *Let us consider template* $\Pi^{\lambda,\gamma}$ *and corresponding instance* $I^{\theta,\psi}$*, such that* $\Pi^{\lambda,\gamma} \overset{R}{\rightarrow}$

$I^{\theta,\psi}$. For each pair $\langle\tau,r\rangle \in \Pi^{\lambda,\gamma}$, where τ is a task t, a service s, a pipeline p, or an ecosystem e annotated with requirement $r \in \mathcal{R}$, the assurance evaluation first retrieves the corresponding set of pairs $\langle\hat{\tau}_i,\hat{r}_i\rangle \in I^{\theta,\psi}$, where $\hat{\tau}_i$ is an executable task $\hat{t}$, a deployed service $\hat{s}$, a concrete pipeline p, or a concrete ecosystem e annotated with concrete requirement $\hat{r} \in \mathcal{R}$. It then computes the overall assurance confidence level as $min(A_{\hat{\tau}_i,\hat{r}_i})$.

We note that the assurance evaluation calculates the assurance confidence level $l_{\tau,r}$ of each $\langle\tau,r\rangle$ expressed by the client on template $\Pi^{\lambda,\gamma}$. To this aim, it aggregates the results of multiple assurance processes A executed on the corresponding $\langle\hat{\tau}_i,\hat{r}_i\rangle \in I^{\theta,\psi}$, and computes $l_{\tau,r}$ as the minimum assurance level. More advanced approaches will be investigated in our future work. Figure 3 shows a complete view of our methodology applied to the scenario in Section 4.

Following the assurance process A in Section 2.1, the assessment is carried out via a specific set of assessment probes P_i suitable for the target τ and the specific requirements $\hat{r}$ annotated on the corresponding targets by θ or ψ. Probes are parametric and use the hooks offered by τ to carry out the inspections. To accomplish the heterogeneity of assurance verification, we define different assessment probes (see Table 2): *i) pipeline probes* focusing on the verification of pipeline tasks and orchestration, *ii) service probes* focusing on the verification of the Big Data services executing the pipelines, *iii) ecosystem probes* focused on the lower layers involving infrastructure behind the Big Data ecosystem.

Table 2: Types of assessment probes.

Probe Name	Description
Task probes	
Code Inspection	search instructions/pattern
Dependency Check	find vulnerable dependencies
Code Vulnerability Check	search vulnerable code
Lineage	verify sequence of actions using logs
Pipeline probes	
Parameters Check	check tasks' actual parameters
Orchestration Check	check the workflow structure
Service probes	
Vulnerability Check	search for vulnerability
Configuration Check	parse and verify configuration
Ecosystem probes	
Infrastructure	targets lower layers such as OS (see [21])
General purposes probes	
Testing	perform specific test cases on a target
Monitoring	monitor a target ore a time frame

Example 5.3 (Probes). *Let us consider a given task $\hat{t}_2$ = normalization(all) annotated by $\theta(\hat{t}_2)$ with a requirement $\hat{r}^{\theta}_1$ = Avoid vulnerable libraries and a given ecosystem service $\hat{s}_4$ = Knox annotated by $\psi(\hat{s}_4)$ with a requirement $\hat{r}^{\psi}_1$ = Authentication-enabled. A Code Vulnerability Check Probe P_i can be used to check whether task $\hat{t}_2$ relies on vulnerable libraries reducing the confidentiality ($\hat{r}^{\theta}_1$). A Configuration Check Probe P_j can be used to check whether authentication is requested by service $\hat{s}_4$ to trigger an analytics via a well-configured Apache Knox service ($\hat{r}^{\psi}_1$). A Testing Probe P_k can be used to check whether authentication is working, while trying to execute the pipeline ($\hat{r}^{\psi}_1$).*

The probes are structured according to our probe paradigm detailed in [27] where the probe receives parameters for connecting to the target (e.g., API hooks) and instruction on how to match the information retrieved by the target (e.g., test cases and expected outputs). It performs *i)* connection, *ii)* information retrieval and *iii)* matching against expectations. The information retrieved and the matching outcomes constitute the probe evidence.

Evidence EV retrieved by the probes is evaluated (see Definition 2.2) and used to compute the assurance level $l_{\tau,\hat{r}}$ (see Definition 2.3) of the target τ with respect to the given requirement $\hat{r}$.

Example 5.4 (Assurance Confidence Levels). *Following Example 5.3, evidence $EV_{\hat{t}_2,\hat{r}^{\theta}_1} = [CVE-2018-1334, 4.7]$ (obtained by P_1) points to a weakly-vulnerable (severity 4.7) version of spark library used in normalization code; evidence $EV_{\hat{s}_4,\hat{r}^{\psi}_1} = [[enabled = true], [(\langle valid-user\rangle, \langle valid-pwd\rangle, allow), (\langle valid-user\rangle, \langle wrong-pwd\rangle, deny) \cdots]]$ points to a check ($[enabled = true]$) on the Apache Knox configuration enabling authentication for pipeline execution (obtained by P_j) and a set of test cases ($[(\langle valid-user, valid-pwd\rangle, allow), (\langle valid-user, wrong-pwd\rangle, deny) \cdots]$) for authentication verification for pipeline triggering (obtained by P_k).*

Two evaluation processes analyse the collected evidence to check the support of the requirements. In this example, $E_1(EV_{\hat{t}_2,\hat{r}^{\theta}_1}, \hat{r}^{\theta}_1) = 0.5$, due to the presence of a vulnerable library but having a low severity score and $E_2(EV_{\hat{s}_4,\hat{r}^{\psi}_1}, \hat{r}^{\psi}_1) = 1$, due to the positive checks on the Knox authentication. The corresponding assurance confidence levels following Definition 2.4 are $l_{\hat{t}_2,\hat{r}^{\theta}_1} = 0.5$ and $l_{\hat{s}_4,\hat{r}^{\psi}_1} = 1$.

The outcome of the assurance process is a set of assurance confidence levels $l_{\tau,\hat{r}}$ on the annotated pipeline instance $I^{\theta,\psi}$. Following Definition 5.4, the assurance confidence levels are aggregated in a final assurance evaluation. Considering the assurance levels in Example 5.4, the overall assurance evaluation is equal to 0.5, the minimum between the retrieved assurance confidence levels.

We note that our assurance process immediately react against events impacting on the assurance evaluations, retrieving new evidence and computing the new assurance level. Events include new discovered vulnerabilities, new versions of ecosystem services, updates on the task or pipeline code, or new more effective probe deployed on the probe registry. In case the assurance level is no more satisfactory, the instance can be replaced with a new one if the requested changes are major. The instance can be adapted in terms of annotations if the requested changes are minor (e.g., new service versions, new probes or new vulnerabilities). While we consider adaptation for our future work, we provide a preliminary discussion in the experiments in Section 6.

6. Experimental Evaluation

We experimentally evaluate our solution first describing the execution of our assurance process on an extended version of our running example in Section 4. We then present a performance evaluation in terms of the computational effort requested by our assurance methodology.

6.1. Experimental setup

Our experimental setup is a portion of the H2020 EVOTION platform hosted on our premises including more than 100 tasks, 20 analytics templates, and 30 analytics instances. Pipelines and tasks are implemented in Spark 2.3 and Spark 2.2 using python or Java, and when needed using Spark mllib. Our H2020 EVOTION platform is grounded on the following ecosystem: Linux 5.16.19 (NixOS), Apache Hadoop 3.3.1, Apache Spark 3.2.1, Apache Airflow 2.2.4. The platform is deployed on a 'bare-metal' infrastructure using an AMD 5900x and 32GB of RAM. The service configurations and the pipeline implementations are available at `bit.ly/3uy1Op4`. Our Assurance architecture in Section 2.2 is deployed on top of the H2020 EVOTION Big Data engine, to exploit the parallelization capabilities and synchronize the execution of assurance evaluation activities with the instance triggering.

6.2. Assurance Evaluation Walkthrough

We present a detailed walkthrough of our methodology based on an extended version of our running example in Section 4, where the client asks for both confidentiality in transit and at rest. Table 3 shows the template $\Pi^{\lambda,\gamma}$ and instance $I^{\theta,\psi}$ annotations.

The walkthrough is organized as follows. First, we show the entire assurance process including details on the used probes. Second, we present the process to obtain the final assurance level.

6.2.1. Assurance Evaluation: Pipeline

Let us start considering the three annotations r_1^θ, r_2^θ and r_3^θ that are common to all the tasks $\hat{t}_i \in \hat{p}$. The assessment processes are implemented with the same set of probes P_1, P_2, and P_3 for r_1^θ, P_4 for r_2^θ, and P_5 and P_6 for r_3^θ (see Table 4). Table 5 shows the pseudocode of the different probes according to their types defined in Table 2.

$P_1(r_1^\theta, *)$ is a Lineage probe focused on verifying that the spark job implementing the tasks $\hat{t}_i$ is writing data on the HDFS offered by the service $\hat{s}_1 \in \hat{e}$ only exploring the spark DAG. It is focused on verifying the compliance to a given spark writing pattern (i.e., *expected_dag*) and explores the spark DAG using the spark log.

$P_2(r_1^\theta, *)$ is a Code Inspection probe focused on verifying connection to external data storage observable within the source code. It is focused on finding specific path patterns (i.e., *expected_paths*) within a task source code.

$P_3(r_1^\theta, *)$ is a Parameters Check probe focused on verifying if parameters for writing at rest (if any) are referring to the HDFS offered by the service $\hat{s}_1 \in \hat{e}$ or not. It is based on controlling the actual parameters of a given task and match them against expected parameters/patterns (in this case writing URLs)

Table 3: Requirements annotations for the running example in Section 4 considering confidentiality in transit and at rest.

$\mathcal{R}$	**Description**
Template requirements	
r_1^λ	*Confidentiality at rest and in transit* for all the t in p
r_2^λ	*Authorization* for the $fileSystem$ task t in p
r_1^γ	*Confidentiality at rest and in transit* for all the s in e
Pipeline Instance requirements derived from r_1^λ	
r_1^θ	*No temporarily unprotected data storage* for all the $\hat{t} \in \hat{p}$
r_2^θ	*Avoid connection to external services* for all the $\hat{t} \in \hat{p}$
r_3^θ	*Avoid use of vulnerable code/libraries* for all the $\hat{t} \in \hat{p}$
r_4^θ	*Pipeline integrity* checking the correct ordering of tasks $\hat{t} \in \hat{p}$
Pipeline Instance requirements derived from r_2^λ	
r_5^θ	*Check Authorization* for ingestion task $\hat{t}_1 \in \hat{p}$
r_6^θ	*Check ownerships at rest* for visualization tasks $\hat{t}_4 \in \hat{p}$
Ecosystem Instance requirements derived from r_1^γ	
r_1^ψ	*Encrypted HDFS* for the $\hat{s}_1 \in \hat{e}$
r_2^ψ	*Inter-node communication security* for the $\hat{s}_1$ and $\hat{s}_2 \in \hat{e}$
r_3^ψ	*Orchestrator Confidentiality* for the $\hat{s}_3 \in \hat{e}$
r_4^ψ	*Communication channel security* for the $\hat{s}_4 \in \hat{e}$
r_5^ψ	*Authentication-enabled* for the $\hat{s}_4 \in \hat{e}$
r_6^ψ	*Authorization policies-enabled* for the $\hat{s}_5 \in \hat{e}$
r_7^ψ	*Vulnerability check* for all the services $\hat{s} \in \hat{e}$

Table 4: Assurance evaluation process for pipeline and tasks for the scenario in Table 1. It includes assurance probes, evidence, evaluations and assurance levels.

Pipeline tasks $\hat{t} \in \hat{T}$				
$\hat{t}$	$\mathcal{R}$	$P(r,\tau)$	$E(EV,r)$	$A_{\tau,r}$
$\hat{t}_1$	r_1^θ	$P_1(r_1^\theta,\hat{t}_1)$	[1.0]	1.0
	r_2^θ	$P_2(r_2^\theta,\hat{t}_1), P_3(r_2^\theta,\hat{t}_1), P_4(r_2^\theta,\hat{t}_1)$	[1.0, 1.0, 1.0]	1.0
	r_3^θ	$P_5(r_3^\theta,\hat{t}_1), P_6(r_3^\theta,\hat{t}_1)$	[0.75, 1.0]	0.88
	r_5^θ	$P_8(r_5^\theta,\hat{t}_1)$	[1.0]	1.0
$\hat{t}_2$	r_1^θ	$P_1(r_1^\theta,\hat{t}_2)$	[1.0]	1.0
	r_2^θ	$P_2(r_2^\theta,\hat{t}_2), P_3(r_2^\theta,\hat{t}_2), P_4(r_2^\theta,\hat{t}_2)$	[1.0, 1.0, 1.0]	1.0
	r_3^θ	$P_5(r_3^\theta,\hat{t}_2), P_6(r_3^\theta,\hat{t}_1)$	[0.75, 1.0]	0.88
$\hat{t}_3$	r_1^θ	$P_1(r_1^\theta,\hat{t}_3)$	[1.0]	1.0
	r_2^θ	$P_2(r_2^\theta,\hat{t}_3), P_3(r_2^\theta,\hat{t}_3), P_4(r_2^\theta,\hat{t}_3)$	[1.0, 1.0, 1.0]	1.0
	r_3^θ	$P_5(r_3^\theta,\hat{t}_3), P_6(r_3^\theta,\hat{t}_1)$	[0.75, 1.0]	0.88
$\hat{t}_4$	r_1^θ	$P_1(r_1^\theta,\hat{t}_4)$	[1.0]	1.0
	r_2^θ	$P_2(r_2^\theta,\hat{t}_4), P_3(r_2^\theta,\hat{t}_4), P_4(r_2^\theta,\hat{t}_4)$	[1.0, 1.0, 0.0]	0.66
	r_3^θ	$P_5(r_3^\theta,\hat{t}_4), P_6(r_3^\theta,\hat{t}_1)$	[0.75, 1.0]	0.88
	r_6^θ	$P_9(r_6^\theta,\hat{t}_4)$	[1.0]	1.0
$\hat{p}$	r_4^θ	$P_7(r_4^\theta,\hat{p})$	[1.0]	1.0

$P_4(r_2^\theta, *)$ is a Code Inspection probe focused on verifying external network connections used for setting up data traffic. It is based on checks for connections to external sources (if any).

$P_5(r_3^\theta, *)$ is a Code Vulnerability Check probe focused on finding vulnerable code used in the spark task $\hat{t}_i$. It is based on pylint as tool for python vulnerability check.

$P_6(r_3^\theta, *)$ is a Dependency Check probe (see Table 5) focused in finding vulnerable libraries used in the spark task $\hat{t}_i$. It is based on analysis of the "requirements.txt" file.

Considering r^{θ}_{4}, it requests to verify absence of hidden tasks or a wrong sequence of task in the pipeline compared to the one declared. It is implemented with a Orchestration Check $P_7(r^{\theta}_{4}, \hat{p})$ targeting the python airflow description of pipeline DAG and verifying the ordering and presence of additional tasks compared to what is expected.

Considering r^{θ}_{5} it is annotated on the ingestion task $\hat{t}_1$ only. The assessment process was implemented with the probe $P_8(r^{\theta}_{5}, \hat{t}_1)$ realized via Testing Probe verifying that the users executing the pipeline against ownership of the source of data ingested in order to check that the pipeline has the right to carry out the ingestion.

Considering r^{θ}_{6} it is annotated on the visualization task $\hat{t}_4$ only. The assessment process is implemented with the probe $P_9(r^{\theta}_{6}, \hat{t}_4)$ realized via Testing Probe focused on the ownership at rest in order to verify that the pipeline write the data preserving the original ownership of the pipeline executor avoiding confidentiality infringement.[3]

Table 4 shows results of our assurance process applied to the scenario in Table 1. Probes P_1 to P_3 and P_6 to P_9 did not identify any significant issue in the pipeline, regardless the target resulting in an optimal result of 1.0. On the other hand, P_5 on every target and for every requirements and $P_4(r^{\theta}_{2}, \hat{t}_4)$ identified issues. More specifically P_5 identified several warnings and coding convention violations, including too lengthy lines and methods naming conventions, resulting in a result of 0.75. $P_4(r^{\theta}_{2}, \hat{t}_4)$ identified a networking connection within $\hat{t}_4$ that should not be there resulting in 0 as EV. The final Assurance Confidence Levels, represented in the last column of Table 4, are calculated according to our the assurance confidence levels aggregation in Definition 2.4.

6.2.2. Assurance Evaluation: Ecosystem

Table 7 shows the pseudocode of the different service ecosystem probes dividing them into the different probe types as defined in Table 2. Let us start considering $\hat{s}_1$ (i.e., hadoop HDFS) used for ingestion and visualization. It is associated with requirements r^{ψ}_{1} and r^{ψ}_{2}. In both cases, to implement the relative assessment processes, Configuration Check probes $P_{10}(r^{\psi}_{1}, \hat{s}_1)$ and $P_{11}(r^{\psi}_{2}, \hat{s}_1)$ are used. Probe P_{10} is focused on verifying whether the HDFS is configured to be encrypted, while probe P_{11} is focused on verifying that secure channels are active between the different storing nodes of the HDFS. Considering $\hat{s}_2$ (i.e., Spark) and $\hat{s}_3$ (i.e., Airflow), which are services used to execute the pipeline, they are both associated with requirements r^{ψ}_{2} and r^{ψ}_{3}, respectively. The relative assessment processes are implemented using specific Configuration Check probes $P_{12}(r^{\psi}_{2}, \hat{s}_2)$ and $P_{13}(r^{\psi}_{3}, \hat{s}_3)$ aimed to verify specific security features of Spark for internode communication and Airflow security, respectively.

$\hat{s}_4$ (i.e., Knox), which is used to provide authentication mechanisms, it is associated with requirements r^{ψ}_{4} and r^{ψ}_{5}. The corresponding assessment processes are implemented using Testing probe $P_{14}(r^{\psi}_{4}, \hat{s}_4)$ to test the security of the channel used to carry out authentication (Kerberos) and a Configuration Check probe $P_{15}(r^{\psi}_{5}, \hat{s}_4)$ aimed to verify the authentication configuration.

$\hat{s}_5$ (i.e., Ranger), which is used to authorize users to carry out analytics, is associated with requirements r^{ψ}_{6}. The relative assessment processes are implemented using Monitoring probe $P_{16}(r^{\psi}_{6}, \hat{s}_5)$ aimed to verify policies triggered by Ranger.

According to requirement r^{ψ}_{7}, for every service $\hat{s} \in \hat{e}$, a vulnerability check needs to be executed. The assessment process is implemented as a Vulnerability Check probe $P_{17}(r^{\psi}_{7}, *)$ aimed to find vulnerabilities that can be relevant for requirement r^{ψ}_{7} on all services $\hat{s} \in \hat{e}$.

Table 6 shows results of our assurance process applied to the scenario in Table 1. Each probe in the table produced a single value. More than half of them obtained a low confidence level (0.1) due to the sub-optimal configurations adopted by the relative services $\hat{s}$. $P_{10}(r^{\psi}_{1}, \hat{s}_1)$ warned about a misconfiguration over the in-transit data encryption, while $P_{11}(r^{\psi}_{2}, \hat{s}_1)$ signaled that the registry security configuration was not enabled. $P_{12}(r^{\psi}_{2}, \hat{s}_2)$ detected that network encryption was not enabled for Spark, while $P_{13}(r^{\psi}_{4}, \hat{s}_3)$ highlighted the encryption fernet key being unset. $P_{17}(r^{\psi}_{7}, \hat{s}_5)$ found several registered CVEs of various severity, the worst of which had a score of 4.3, resulting in a confidence level of 0.57. The remaining probes did not identify significant issues and returned an optimal value.

6.2.3. Overall Assurance Evaluation

The final assurance evaluation was carried out according to Definition 5.4 leading to the following assurance confidence levels $l_{*,r^{\lambda}_{1}} = 0.66$, $l_{*,r^{\lambda}_{2}} = 1$ and $l_{*,r^{\gamma}_{1}} = 0.1$ aggregated on the relevant targets $*$. We note that the assurance confidence level for r^{λ}_{1} *Confidentiality at rest and in transit* is impacted negatively by the assurance evaluation on r^{θ}_{2} related to *Avoid connection to external services* carried out by P_4 probe on $\hat{t}_4$ = *saveToHDFS(model)*. We also note that the ecosystem of services clearly shows the lowest assurance level due to severe configuration issues for $\hat{s}_1$, $\hat{s}_2$, and $\hat{s}_3$ retrieved by the relative probes.

6.3. Performance Evaluation

We evaluate the computational effort requested by our assurance process for the walkthrough in Section 6.2 considering the different probe types adopted and different changing scenarios as follows. We note that our assurance solution reuses evidence from related assessment processes in the framework of computing the final assurance level.

(a) Contextual Changes (CC): It includes changes due to the external factors like new version of the probe used in the repository or new discovered vulnerabilities. This scenario requires re-execution of the new probes as well as of the vulnerability-related probes.

(b) Pipeline Changes (PC): It includes changes due to different parameters and changes at orchestration level. This scenario requires the re-execution of the task or orchestration probes showing changes only.

[3]Current Big Data engines may lead to issues of data ownerships in a multi-tenant context.

Table 5: Pipeline probe scripts: Pseudocode

Code Inspection

Probe 2 and 4: $P_1(r_1^{\theta}, *)$ $P_4(r_2^{\theta}, *)$

```
def p2(expected_paths, task_id):
evidence.source_code = get_source(task_id)
evidence.detected_paths = get_hdfs_paths(
    evidence.source_code)
for path in evidence.detected_paths:
    if path not in expected_paths:
        evidence.warnings.append(
            f"Unexpected path {path}")
return evidence
```

```
def p4(task_id, expected_services):
evidence.activity = get_network_activity(task_id)
evidence.services = parse_services(evidence.activity)
for service in evidence.services
    if service not in expected_services:
        evidence.warnings.push(
            f"Unexpected service {service}")
return evidence
```

Dependency Check

Probe 6: $P_6(r_3^{\theta}, *)$

```
def p6(task_id)
evidence.requirements = get_requirements(task_id)
output = call_safety(evidence.requirements)
evidence.warnings = json.loads(
    output.decode().strip())
return evidence
```

Parameters Check

Probe 3: $P_3(r_1^{\theta}, *)$

```
def p3(expected_params, task_id)
params = get_params(task_id)
evidence.params = params
for param, exp_param
in zip(params, expected_params):
    if not param.matches(exp_params):
        evidence.warnings.push(
            f"Param {param} not matching")
return evidence
```

Orchestration Check

Probe 7: $P_7(r_4^{\theta}, \hat{p})$

```
def p7(expected_dag)
dag = get_current_dag()
evidence.dag = dag
for task in dag:
    exp_task = expected_dag.find(task)
    if exp_task is None:
        evidence.warnings.push(
            f"Unexpected task {task}")
    if task.deps != exp_task.deps:
        evidence.warnings.push(
            f"Unexpected {task} deps.")
return evidence
```

Lineage

Probes 1: $P_1(r_1^{\theta}, *)$

```
def p1(expected_dag, app_id):
evidence.logs = get_spark_logs(app_id)
for job in evidence.logs.jobs:
    if job not in expected_dag:
        evidence.warnings.push(f"Unexpected job {job}")
return evidence
```

Testing

Probe 8 and 9 : $P_8(r_5^{\theta}, \hat{t}_1)$, $P_9(r_6^{\theta}, \hat{t}_4)$

```
def p8(user, file_path):
try:
    file = access_as_user(user, file_path)
    file.head()
    evidence.output = "File can be ingested"
except PermissionError:
    evidence.output = "Permission Error"
except MissingFile:
    evidence.output = "File not found"
return evidence
```

```
def p9(expected_permissions, user, file_path):
try:
    file = access_as_user(user, file_path)
    permissions = file.get_permissions()
    evidence.permissions = permissions
    if permissions == expected_permissions:
        evidence.output = "Permissions match"
    else:
        evidence.output = "Invalid permissions"
except PermissionError:
    evidence.output = "Permission Error"
except MissingFile:
    evidence.output = "File not found"
return evidence
```

Code Vulnerability Check

Probe 5: $P_5(r_3^{\theta}, *)$

```
def p5(task_id)
evidence.source_code = get_source(task_id)
evidence.warnings = call_pylint(source_code)
return evidence
```

Table 6: Assurance evaluation process for Ecosystem and services for the scenario in Table 1. It includes assurance probes, evidence, evaluations and assurance levels.

Ecosystem services $\hat{s} \in \hat{S}$				
$\hat{s}$	$\mathcal{R}$	$P(r,\tau)$	$E(EV,r)$	$A_{\tau,r}$
$\hat{s}_1$	r_1^{ψ}	$P_{10}(r_1^{\psi}, \hat{s}_1)$	[0.1]	0.1
	r_2^{ψ}	$P_{11}(r_2^{\psi}, \hat{s}_1)$	[0.1]	0.1
$\hat{s}_2$	r_2^{ψ}	$P_{12}(r_2^{\psi}, \hat{s}_2)$	[0.1]	0.1
$\hat{s}_3$	r_4^{ψ}	$P_{13}(r_4^{\psi}, \hat{s}_3)$	[0.1]	0.1
$\hat{s}_4$	r_4^{ψ}	$P_{14}(r_4^{\psi}, \hat{s}_4)$	[1.0]	1.0
	r_5^{ψ}	$P_{15}(r_5^{\psi}, \hat{s}_4)$	[1.0]	1.0
$\hat{s}_5$	r_6^{ψ}	$P_{16}(r_6^{\psi}, \hat{s}_5)$	[1.0]	1.0
$\hat{s}_5$	r_7^{ψ}	$P_{17}(r_7^{\psi}, \hat{s}_5)$	[0.57]	0.57

(c) Ecosystem Changes (EC): It includes different versions/configurations of services at ecosystem level. This scenario requires the re-execution of the ecosystem level probes only.

(d) Instance Changes (IC): It includes changes requesting complete assurance re-evaluation.

More specifically, in our walkthrough, we consider *i)* CC assuming new vulnerabilities discovered for all the services used at ecosystem level, *ii)* PC assuming new version of k-means modeling task only, *iii)* EC assuming new version of HDFS service, *iv)* IC assuming a new task for ingestion based on Hive instead of HDFS. We also consider a scenario where the assurance is re-evaluated without any changes.

Figure 4 shows a stacked histograms of the time requested to execute the entire assurance process in the different scenarios considering the different adopted probe types.

The pipeline taken as example is training a k-means model based on the dataset 'digits' from the *scikit-learn*[4] library. The model implementation is provided by the Spark ML library[5]. Finally the trained model is serialized and saved in an Hadoop storage. The time necessary to execute the pipeline is 54 seconds.

The pipeline probes are executed in the CC, PC and IC scenarios requiring 0.13 seconds, while the service probe are executed in the EC and IC scenarios in 3.65 seconds. The general purpose probes have a total execution time of 5.79 seconds in scenarios EC and IC, while in CC their evaluation is completed

[4] `https://scikit-learn.org`

[5] `https://spark.apache.org/mllib`

Table 7: Service ecosystem probe scripts: Pseudocode

Configuration check

Probe 10 and 11: $P_{10}(r_1^{\psi}, \hat{s}_1)$ $P_{11}(r_2^{\psi}, \hat{s}_1)$

```
def p10(config_endpoint)
config = get_hadoop_configuration(config_endpoint)
evidence.config = config
warnings = []
if config["dfs.encrypt.data.transfer"] == "false":
    warnings.append("In−transit data encryption is disabled")
if config["yarn.intermediate−data−encryption.enable"] == "false":
    warnings.append("Intermediate data encryption is disabled")
...
evidence.warnings = warnings
return evidence

def p11(config_endpoint)
config = get_hadoop_configuration(config_endpoint)
evidence.config = config
warnings = []
if config["dfs.permissions.enabled"] == "false":
    warnings.append("FS access control is disabled")
if config["dfs.permissions.superusergroup"] == "supergroup":
    warnings.append("Task has unrestricted permissions")
if config["hadoop.registry.secure"] == "false":
    warnings.append("Registry security is not enabled")
if config["hadoop.security.authorization"] == "false":
    warnings.append("Authentication is disabled")
...
evidence.warnings = warnings
return evidence
```

Probe 12 and 13: $P_{12}(r_2^{\psi}, \hat{s}_2)$ $P_{13}(r_3^{\psi}, \hat{s}_3)$

```
def p12(app_id, api_endpoint):
config = get_spark_config(app_id, api_endpoint)
evidence.config = config
warnings = []
if config["spark.network.crypto.enabled"] != true:
    warnings.append("Network encryption disabled")
if config["spark.io.encryption.enabled"] != true:
    warnings.append("IO encryption disabled")
...
evidence.warnings = warnings
return evidence

def p13(api_endpoint):
config = get_airflow_config(api_endpoint)
evidence.config
warnings = []
if config.core.get("fernet_key") is None:
    warnings.append("Fernet key is not set")
if config.kubernetes.verify_ssl is False:
    warnings.append("SSL cert. check disabled")
...
evidence.warnings = warnings
return evidence
```

Probe 15: $P_{15}(r_5^{\psi}, \hat{s}_4)$

```
def p15(exp_user_perms, api_endpoint)
for user, exp_perms in exp_user_perms.items():
    perms = get_perms(user, api_endpoint)
    evidence.permissions[user] = perms
    if !exp_perms.matches(perms):
        evidence.warnings.append(
            f"unexpected perms. {perms} for {user}")
return evidence
```

Vulnerability Check

Probe 17: $P_17(r_7^{\psi}, *)$

```
def p17(service):
evidence.warnings = openvas_analysis(service)
return evidence
```

Monitoring

Probe 16: $P_{16}(r_6^{\psi}, \hat{s}_5)$

```
def p16(expected_policies, api_endpoint):
logs = get_ranger_logs()
evidence.logs = logs
warnings = []
for event in logs:
    if !event.matches(expected_policies):
        warnings.append(
        f"Unexpected event {event}")
evidence.warnings = warnings
return evidence
```

Testing

Probe 14: $P_{14}(r_4^{\psi}, \hat{s}_4)$

```
def p14(auth_config, api_endpoint):
result = authenticate(auth_config, api_endpoint)
if result.failed():
    evidence.output = "Authentication failed"
elif result.unauthorized():
    evidence.output = "Not authorized"
...
return evidence
```

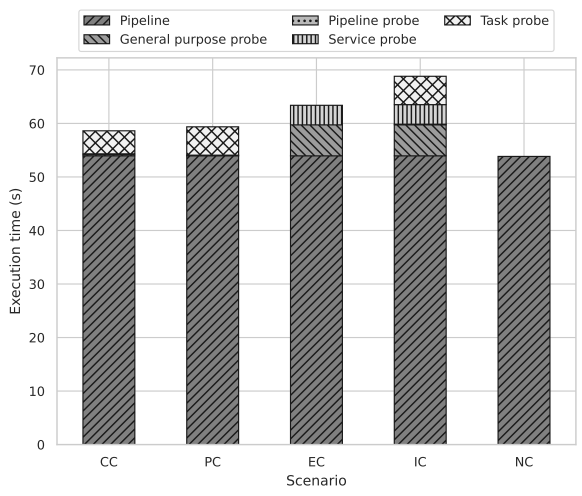

Figure 4: Performance on the walkthrough in different scenarios: contextual changes (CC), pipeline changes (PC), ecosystem changes (EC), instance changes (IC) and no changes (NC). Computational time requested by the different probes aggregated by probe types.

is 0.25 seconds as only part of them has to be re-calculated. Similarly the task probes total evaluation time is 5.29 seconds in scenarios PC and IC, dropping to 4.30 seconds in CC.

We note that, in this experiment, *i)* we are considering a sequential execution of the required probes, but parallelization is possible in most cases, saving a significant amount of time; *ii)* the time requested for the IC assessment is obviously greater than the others, since it is executing all the probes evaluations; *iii)* the pipeline probe evaluations are particularly efficient as they only require information already stored in the configuration of the pipeline instance; *iv)* the total probes evaluation execution time is negligible compared to time required by the pipe in a Big Data context. Concerning the CC and IC scenarios, we note that the vulnerability analysis probe $P_17(r_7^{\psi}, *)$ is not included in Figure 4 due to its long execution time, approximately 3 minutes, resulting in a strongly unbalanced graph and unreadable bars. We also note that assurance levels coming from other assessment processes can be reused in case the targets and the requirements are the same. This is typically the case of ecosystem services used by different pipelines. Figure 4 shows this effect clearly for NC scenario where all of the assessment processes are reused from the previous runs and the total requested time is almost the one needed for the pipeline only.

7. Related Work

The recent years have seen a rapid growth in the usage of Big Data for a large number of fields. This trend is followed by the awareness of the necessity of protecting the data and its by-products with transparent and comprehensive techniques of security assurance. The most straightforward approach taken into consideration to address these necessities was based on the integration of security policies in the Big Data processes in order to mitigate or prevent risks originated by the mishandling of data [28]. Some of the initial security assurance techniques focused on Big Data targeted primarily on securing the stored data [29, 30, 31, 32] and only later expanded to a wider definition that includes its handling in the form of data pipelines [33], securing them according to the CIA (*Confidentiality*, *Integrity*, *Availability*) triad [34, 35, 36]. This includes security measures like access control policies on the resources and network isolation of the services. However these security hardening solutions are not enough to cope with dynamicity of Big Data as a service scenario. The complex structure and variety of the stored data reflect an even more diverse usage of the information contained. Lack of a complete understanding of the security aspects of all the stored data may expose the users to significant risks [36]. Users personal data may be protected by a policy that prevents a malicious actor to acquire all their records, but it may still be possible to de-anonymize them from leaky statistical data if the k-anonymity property, defined in [37], is not properly enforced [38, 39]. The growth of high-sensitive data (i.e. financial, healthcare) and the introduction of privacy focused regulations like the General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA) pushed towards the inclusion of *Privacy* in the primary aspects of the security assurance [40, 34, 32]. Emphasizing how breaches may occur at any step in the pipeline [41]. Researchers highlighted the necessity of automated processes to ensure compliance with privacy regulations [33] and high performance [40]. Due to the size and complexity of the systems, monitoring has been the primary approach applicable to defend Big Data applications according to Elsayed et al. [42]. More advanced techniques include continuous monitoring of the behavior of the data handlers, in order to identify malicious or unexpected actions [9, 10]. Additional security mitigations solutions adopted anomaly detection techniques to detect model poisoning [43] or advanced access control and storage monitoring [44], although their focus is especially localized and small. Security assurance, on the other hand, enables its users to address privacy and security concerns by actively testing the whole Big Data pipeline environment, its configuration and its behavior, verifying a target set of non-functional properties, demonstrating if the applied countermeasures are effective [12]. The literature is lacking in this particular aspect of assurance, while more focus is given to the evaluation of quality of the systems, the models and the data, as in [45, 46, 47].

8. Conclusions

We proposed an assurance process for Big Data that aims to increase Big Data trustworthiness and transparency. Our Big Data process carries a higher level of trust, which makes it suitable for critical Big Data scenarios, as for instance the ones implementing a process for security governance. Concluding we showed a preliminary application of our approach in the framework of a security compliance verification for Apache Hadoop.

To conclude, the increasing trend towards Big Data process outsourcing and the lack of Big Data trustworthiness represents a major barrier against high-quality Big Data computations. Users, in fact, (fully) anonymize data and results to reduce their liability in case of data breach. The assurance verification in Section 2 permits to unleash the full power of Big Data, fostering its adoption in critical scenarios where sensitive data are used and promoting the "*Big Data-as-a-Service*" paradigm where trustworthiness is of paramount importance. It is important to remark that Big Data verification is mandatory also when anonymized data are used, for instance, to ensure that the level of anonymity reached with anonymized data is not violated during processing or while presenting the final results due to correlation with different data sources [48].

Acknowledgments

Some of the work reported here was partially funded by the program "piano sostegno alla ricerca" of Università degli Studi di Milano.

References

[1] G. D'Acquisto, J. Domingo-Ferrer, P. Kikiras, V. Torra, Y.-A. de Montjoye, A. Bourka, Privacy by design in big data: an overview of privacy enhancing technologies in the era of big data analytics, arXiv preprint arXiv:1512.06000 (2015).

[2] W. Chang, G. Fox, N.-P. NIST, Nist big data interoperability framework: Volume 3, use cases and general requirements (2015-10-22 2015).

[3] CSA, Big data security and privacy handbook: 100 best practices in big data security and privacy, Tech. rep., CSA, `https://cloudsecurityalliance.org/download/big-data-security-and-privacy-handbook/` (2016).

[4] D. S. Terzi, R. Terzi, S. Sagiroglu, A survey on security and privacy issues in big data, in: Proc. of 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST), IEEE, 2015, pp. 202–207.

[5] S. Yakoubov, V. Gadepally, N. Schear, E. Shen, A. Yerukhimovich, A survey of cryptographic approaches to securing big-data analytics in the cloud, in: Proc. of 2014 IEEE High Performance Extreme Computing Conference (HPEC), IEEE, 2014, pp. 1–6.

[6] X. Zhang, L. Qi, W. Dou, Q. He, C. Leckie, R. Kotagiri, Z. Salcic, Mrmondrian: scalable multidimensional anonymisation for big data privacy preservation, IEEE Transactions on Big Data 8 (1) (2017) 125–139.

[7] G. Manogaran, C. Thota, M. V. Kumar, Metaclouddatastorage architecture for big data security in cloud computing, Procedia Computer Science 87 (2016) 128–133.

[8] P. P. Sharma, C. P. Navdeti, Securing big data hadoop: a review of security issues, threats and solution, Int. J. Comput. Sci. Inf. Technol 5 (2) (2014) 2126–2131.

[9] D. B. Rawat, R. Doku, M. Garuba, Cybersecurity in big data era: From securing big data to data-driven security, IEEE Transactions on Services Computing 14 (6) (2021) 2055–2072.

[10] I. Hababeh, A. Gharaibeh, S. Nofal, I. Khalil, An integrated methodology for big data classification and security for improving cloud systems data mobility, IEEE Access 7 (2019) 9153–9163.

[11] M. Anisetti, N. Bena, F. Berto, G. Jeon, A devsecops-based assurance process for big data analytics, in: Proc. of 2022 IEEE International Conference on Web Services (ICWS), 2022 (to appear).

[12] C. A. Ardagna, R. Asal, E. Damiani, Q. H. Vu, From security to assurance in the cloud: A survey, ACM Computing Surveys (CSUR) 48 (1) (2015) 1–50.

[13] C. Liu, J. Chen, L. T. Yang, X. Zhang, C. Yang, R. Ranjan, R. Kotagiri, Authorized public auditing of dynamic big data storage on cloud with efficient verifiable fine-grained updates, IEEE Transactions on Parallel and Distributed Systems 25 (9) (2013) 2234–2244.

[14] A. Kiesow, N. Zarvic, O. Thomas, Continuous auditing in big data computing environments: Towards an integrated audit approach by using caatts., in: Proc. of GI-Jahrestagung, 2014, pp. 901–912.

[15] Y. Yang, W. Li, D. Yuan, Reliability Assurance of Big Data in the Cloud: Cost-effective Replication-based Storage, Morgan Kaufmann, 2014.

[16] J. Gao, C. Xie, C. Tao, Big data validation and quality assurance–issuses, challenges, and needs, in: Proc. of 2016 IEEE symposium on service-oriented system engineering (SOSE), IEEE, 2016, pp. 433–441.

[17] M. Anisetti, C. A. Ardagna, E. Damiani, P. G. Panero, A methodology for non-functional property evaluation of machine learning models, in: Proc. of the 12th International Conference on Management of Digital EcoSystems, MEDES '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 38–45.

[18] M. Prasinos, I. Basdekis, M. Anisetti, G. Spanoudakis, D. Koutsouris, E. Damiani, A modelling framework for evidence-based public health policy making, IEEE Journal of Biomedical and Health Informatics 26 (5) (2022) 2388–2399.

[19] Y. Demchenko, P. Grosso, C. De Laat, P. Membrey, Addressing big data issues in scientific data infrastructure, in: Proc. of 2013 International conference on collaboration technologies and systems (CTS), IEEE, 2013, pp. 48–55.

[20] A. Jakóbik, Big Data Security, Springer International Publishing, 2016, Ch. 12, pp. 241–261.

[21] M. Anisetti, C. A. Ardagna, E. Damiani, F. Gaudenzi, A semi-automatic and trustworthy scheme for continuous cloud service certification, IEEE Transactions on Services Computing 13 (1) (2017) 30–43.

[22] P. Stephanow, N. Fallenbeck, Towards continuous certification of infrastructure-as-a-service using low-level metrics, in: Proc. of 2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), 2015, pp. 1485–1492.

[23] G. Gigante, D. Pascarella, Formal methods in avionic software certification: the do-178c perspective, in: Proc. of International Symposium On Leveraging Applications of Formal Methods, Verification and Validation, Springer, 2012, pp. 205–215.

[24] M. Anisetti, C. Ardagna, E. Damiani, G. Polegri, Test-based security certification of composite services, ACM Trans. Web 13 (1) (dec 2018).

[25] C. A. Ardagna, V. Bellandi, M. Bezzi, P. Ceravolo, E. Damiani, C. Hebert, Model-based big data analytics-as-a-service: take big data to the next level, IEEE Transactions on Services Computing 14 (2) (2018) 516–529.

[26] G. Spanoudakis, P. Katrakazas, D. Koutsouris, D. Kikidis, A. Bibas, N. H. Pontopidan, Public health policy for management of hearing impairments based on big data analytics: Evotion at genesis, in: Proc. of 2017 IEEE 17th International Conference on Bioinformatics and Bioengineering (BIBE), 2017, pp. 525–530.

[27] M. Anisetti, C. A. Ardagna, E. Damiani, F. Gaudenzi, A security benchmark for openstack, in: Proc. of 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), 2017, pp. 294–301.

[28] M. Anisetti, C. A. Ardagna, C. Braghin, E. Damiani, A. Polimeno, A. Balestrucci, Dynamic and scalable enforcement of access control policies for big data, in: Proc. of the 13th International Conference on Management of Digital EcoSystems, 2021, pp. 71–78.

[29] C. Tankard, Big data security, Network security 2012 (7) (2012) 5–8.

[30] G. D. Samaraweera, J. M. Chang, Security and privacy implications on database systems in big data era: A survey, IEEE Transactions on Knowledge and Data Engineering 33 (1) (2021) 239–258.

[31] D. Yadav, D. H. Maheshwari, D. U. Chandra, Big Data Hadoop: Security and Privacy, SSRN Journal (2019).

[32] M. Khan, M. D. Ansari, Security and privacy issue of big data over the cloud computing: a comprehensive analysis, IJRTE-Scopus Indexed 7 (6s) (2019) 413–417.

[33] L. Wang, J. P. Near, N. Somani, P. Gao, A. Low, D. Dao, D. Song, Data capsule: A new paradigm for automatic compliance with data privacy regulations, in: Heterogeneous Data Management, Polystores, and Analytics for Healthcare, Springer, 2019, pp. 3–23.

[34] E. Bertino, E. Ferrari, Big data security and privacy, in: A comprehensive guide through the Italian database research over the last 25 years, Springer, 2018, pp. 425–439.

[35] L. Zhou, A. Fu, S. Yu, M. Su, B. Kuang, Data integrity verification of the outsourced big data in the cloud environment: A survey, Journal of Network and Computer Applications 122 (2018) 1–15.

[36] J. Koo, G. Kang, Y.-G. Kim, Security and privacy in big data life cycle: A survey and open challenges, Sustainability 12 (24) (2020).

[37] P. Samarati, Protecting respondents identities in microdata release, IEEE Transactions on Knowledge and Data Engineering 13 (6) (2001) 1010–1027.

[38] S. Kavitha, S. Yamini, P. Raja Vadhana, An evaluation on big data generalization using k-anonymity algorithm on cloud, in: 2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO), 2015, pp. 1–5.

[39] S.-B. Jang, A study of performance enhancement in big data anonymization, in: Proc. of 2017 4th International Conference on Computer Applications and Information Processing Technology (CAIPT), 2017, pp. 1–4.

[40] H. K. Patil, R. Seshadri, Big data security and privacy issues in healthcare, in: Proc. of 2014 IEEE international congress on big data, IEEE, 2014, pp. 762–765.

[41] A. Mehmood, I. Natgunanathan, Y. Xiang, G. Hua, S. Guo, Protection of big data privacy, IEEE access 4 (2016) 1821–1834.

[42] M. Elsayed, M. Zulkernine, Towards security monitoring for cloud analytic applications, in: Proc. of 2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS), 2018, pp. 69–78.

[43] M. Sameen, S. O. Hwang, Timpany—detection of model poisoning attacks using accuracy, IEEE Access 9 (2021) 139415–139425.

[44] G. Ra, D. Kim, D. Seo, I. Lee, A federated framework for fine-grained cloud access control for intelligent big data analytic by service providers, IEEE Access 9 (2021) 47084–47095.

[45] J. Gao, C. Xie, C. Tao, Big data validation and quality assurance – issuses, challenges, and needs, in: Proc. of 2016 IEEE Symposium on Service-Oriented System Engineering (SOSE), 2016, pp. 433–441.

[46] P. Zhang, X. Zhou, W. Li, J. Gao, A survey on quality assurance techniques for big data applications, in: Proc. of 2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService), 2017, pp. 313–319.

[47] D. Lee, Big data quality assurance through data traceability: A case study of the national standard reference data program of korea, IEEE Access 7 (2019) 36294–36299.

[48] B. A. Bouna, C. Clifton, Q. Malluhi, Efficient sanitization of unsafe data correlations, in: Proceedings of the workshops of the EDBT/ICDT 2015 joint conference (EDBT/ICDT), Brussels, Belgium, Citeseer, 2015, pp. 278–285.