Big Data Access Control for Cloud-Native Hadoop Environments

Fidèle Tsognong, Benoit Martin Azanguezet Quimatio, and Marcellin Julius Nkenlifack

***Abstract*—The rise of Big Data necessitates robust access control for platforms like Hadoop. While traditionally deployed on physical servers within trusted networks, Hadoop is increasingly migrating to cloud-native, containerized environments. This transition introduces significant security challenges, as the compromise of a single container can potentially expose other resources. Existing Big Data access control models, designed for traditional configurations, often lack the necessary flexibility for dynamic cloud-native environments. This research proposes a usage control-based model to secure privileged access to Big Data and its processing within containerized environments. The paper analyzes existing access control solutions and explores Hadoop architectures in cloud-native deployments. It then presents a model leveraging usage control and multi-step authorization to address these evolving security needs. The proposed approach enhances traditional access control by incorporating organizational context and approval workflows for sensitive operations. It mitigates the risks associated with unbounded privileges and rogue container deployment by enabling real-time, reactive policy enforcement. Unlike existing models, this solution offers dynamic adaptability, fine-grained control, and improved resilience against insider threats, making it particularly well-suited for securing Big Data in modern, distributed environments.**

***Index Terms*—Big Data, Usage control, Authorization, Microservice, Cloud-native, Kubernetes, Open policy agent, Multi-step authorization.**

I. Introduction

As the volume, velocity and variety of data continues to grow exponentially, organizations are increasingly turning to Big Data technologies such as Hadoop to harness this valuable information. However, the massive scale and distributed nature of Big Data systems presents unique security challenges that require robust access control mechanisms.

Apache Hadoop is a comprehensive open source ecosystem for storing, processing and analyzing large data sets [1]. This ecosystem includes core components such as the Hadoop Distributed File System (HDFS) for fault-tolerant distributed storage, Yet Another Resource Negotiator (YARN) for resource management, and the MapReduce framework for parallel processing of large datasets. Traditionally, Hadoop clusters have been deployed directly on physical or virtual machines. Today, organizations are migrating to cloud-native technologies based on the use of containers.

A container [2] is a standardized software unit that encapsulates code and all its dependencies, enabling the application to run quickly and reliably, regardless of the target computing environment. Containerization provides portability, isolation, and efficient use of resources for applications, making it easier to deploy and manage complex systems such as Hadoop across different environments. However, setting up a cluster of secure containers presents a new challenge as the scope of attack expands: if one container is compromised and acts maliciously, other resources in the environment may be at risk [3]. This risk is further amplified by the presence of unbounded privileges, where privileged users can freely create and configure containers without sufficient oversight. Such conditions open the door to the creation of rogue containers, potentially bypassing security controls and escalating threats within the environment. These challenges necessitate robust, fine-grained access control mechanisms that can govern privileged actions, enforce accountability, and limit the impact of compromised components in containerized Big Data infrastructures.

Various access control models have been proposed for Hadoop, including HeAC, OT-RBAC [4] based on roles and tags, HeABAC [5] and H-RABAC [6] which determine access to Hadoop resources based on user, resource, and operational attributes. While effective in traditional architectures, these models fall short in addressing the dynamic nature and increased attack surface of cloud-native environments. Primarily designed for static deployments on dedicated servers, they lack the flexibility and agility required for dynamic and containerized environments.

While research has explored access control in containerized environments [7] and applied RBAC to Kubernetes [8], as well as investigated identity-based access control for multi-cloud settings [9], the specific challenge of securing Hadoop resources within cloud-native environments remains largely unaddressed.

This research addresses the implementation of an organizational access control model, based on the usage control (UCON) model, to secure Big Data processing in multi-tiered, multi-tenant and elastic cloud-native environments by including a multi-step authorization method based on approval of critical accesses.

The main contributions of this research are as follows:

- Conceptualization of a stateful access control model for cloud-native Hadoop, integrating Usage Control and

Manuscript received March 8, 2025; revised April 9, 2025. Date of publication July 15, 2025. Date of current version July 15, 2025. The associate editor prof. Toni Perković has been coordinating the review of this manuscript and approved it for publication.

All authors are with the Department of Mathematics and Computer Science, University of Dschang, Cameroon (e-mails: tsognong.fidele@gmail.com, {benoit.azanguezet, marcellin.nkenlifack}@univ-dschang.org).

Digital Object Identifier (DOI): 10.24138/jcomss-2025-0023

Organization-Based Access Control with multi-step authorization via approvals.

- Formalization of the model using temporal logic to define stateful authorization rules and enforcement mechanisms.
- Proposition of a reactive, real-time policy update architecture for effective access control across cloud-native Hadoop.
- Implementation of the model using Open Policy Agent (OPA) on a modern Hadoop environment, demonstrating feasibility and effectiveness.

The rest of the paper is organized as follows. Section II reviews related work on access control models for Big Data. Section III discusses existing cloud-native Hadoop architectures. Section IV presents our proposed access control model. Section V outlines a logical architecture for enforcing the proposed access control model across the cloud-native Hadoop ecosystem. Section VI details the implementation. Section VII evaluates and compares our work with similar models. Section VIII discusses the proposed model and implementation results. Finally, Section IX summarizes the study's key findings and suggests potential avenues for future research.

II. Related Work

Securing the processing of large amounts of data necessitates a multi-layered approach to access control, encompassing both service and data object permissions. Several access control models have been proposed for Hadoop.

A. Big Data Access Control Models

This section outlines existing access control models relevant to Big Data environments. Gupta et al.'s HeAC framework [4] formalizes Apache Ranger and Sentry's authorization mechanisms, incorporating object attribute-based permissions. Building upon this, Maanak et al.'s OT-RBAC [4] extends HeAC with role-based and tagged object-based controls. The HABAC model [5] introduces attribute-based access control and cross-service trust specifically within Hadoop. Bahloul et al.'s H-RCBAC [10] and Ait et al.'s H-RABAC [6] offer alternative approaches by combining the strengths of role-based and attribute-based access control. In a different vein, Idar Hafsa et al.'s D2SAC [11] focuses on automating the calculation of data sensitivity within Hadoop to inform access control decisions. Usage control mechanisms are explored by BigUCON and RQ-UCON [12], [13] to provide enhanced data protection in Hadoop. However, while these traditional security models in Hadoop-based Big Data ecosystems have evolved to incorporate attributes and even elements of usage control, they still fail to provide critical enforcement mechanisms necessary for securing modern cloud-native architectures. These models typically rely on static authorization policies that lack real-time adaptability, making them inadequate against dynamic, distributed workloads and containerized environments.

In the realm of Attribute-Based Access Control, BIG-ABAC [14] presents a significant advancement by enabling real-time policy evaluation and dynamic adaptation to contextual changes, moving beyond static policies for precise and efficient access management in dynamic and high-stakes environments like healthcare.

Addressing the complexities of cloud computing environments, DR-TBAC [15] proposes a novel dynamic access control system leveraging the Zero-Trust Architecture (ZTA). This framework introduces a Trust-Based Access Control (TBAC) model with dynamic trust assessment and integrates dynamic rules optimized using the Deep Q-Network (DQN) algorithm to achieve continuous authentication and adaptive authorization, aiming to enhance cloud security and outperform baseline models.

Focusing on the challenge of maintaining consistency across heterogeneous Big Data systems, Poltavtseva et al. [16] apply temporal logic. Their research analyzes various types and verification methods before proposing TLA+ as a suitable tool for formally verifying access control processes, emphasizing the role of time analysis in enhancing reliability for consistency in such complex environments.

A major missing component in these approaches is approval mechanisms for critical actions. Privileged operations, such as executing high-impact jobs, accessing sensitive datasets, or modifying security configurations, are often performed without multi-step validation. Without enforced approvals, attackers or compromised processes can escalate privileges, execute unauthorized analytics jobs, or manipulate data pipelines without immediate detection.

Another significant gap is the lack of granular control over service accounts. Service accounts, often used to automate workflows, tend to be unbounded privileges and lack continuous verification. Traditional security frameworks do not enforce strict identity validation for workloads using these accounts, allowing rogue containers to impersonate legitimate processes and push unauthorized data to Flink jobs, Kafka topics, or distributed storage layers.

Moreover, even when attribute-based and usage control models are applied, there is no real enforcement mechanism that integrates secured policy governance. Security policies themselves must be subject to structured validation, peer review, and continuous monitoring to prevent policy drift and misconfigurations.

To truly secure cloud-native Big Data infrastructures, a policy framework must go beyond traditional models by integrating dynamic enforcement, multi-step authorization for high-risk operations, and strict governance over both security policies and privileged identities.

B. Usage Control Model (UCON)

Access control systems determine whether a user is authorized to access a resource at the moment of a request. Usage control (UCON) extends this by introducing continuous evaluation, where access decisions are enforced throughout the usage period, not just at the time of request. UCON [17] is designed for open environments like the Internet, extending Attribute-Based Access Control (ABAC) by incorporating mutable attributes and ongoing enforcement. Attributes can change due to resource usage, potentially leading to access revocation if policy conditions are no longer met.

UCON shares key elements with traditional access models: subjects (active entities), objects (protected resources), and rights (usage functions allowing access). Subjects and objects have attributes represented as key-value pairs, which can be single values or sets. Access decisions in UCON rely on three factors: Authorizations (A), which check attribute-based permissions; Obligations (B), which enforce mandatory requirements for access; and Conditions (C), which evaluate environmental factors independent of the subject or object. The UCON model operates across three phases: pre-access, where initial policies are applied; ongoing, where access is continuously evaluated; and post-access, where updates may occur. Policies can be enforced before and during use, ensuring dynamic access control.

III. Cloud-native Hadoop Architectures

The traditional Hadoop architecture, as described by Rao et al. [18], reflects a comprehensive view of Big Data systems deployed on distributed infrastructures using tools such as Hadoop 3.0 and Spark 2.3. Their work highlights key stages of Big Data processing — from data ingestion to storage, analysis, and visualization — emphasizing distributed file systems, NoSQL databases, and machine learning libraries like Mahout, Spark MLlib, and FlinkML. However, while this traditional architecture is effective for large-scale data processing in on-premise or static cloud environments, it lacks the agility, elasticity, and microservice orientation required in modern cloud-native infrastructures.

The Cloud Native Computing Foundation (CNCF) defines cloud-native applications as loosely coupled microservices packaged in containers, dynamically orchestrated, and managed via declarative APIs across multiple servers [19]. This model enables scalability, portability, and resilience — characteristics increasingly demanded by Big Data processing pipelines.

Uber [20] underscored the scalability and efficiency gains achievable through this approach. Comcast's integration of YARN on Kubernetes [21] expanded the containerization landscape for Hadoop. Additionally, running Apache Spark and Flink on Kubernetes provides dynamic resource management for these frameworks. Beyond these core components, other Hadoop ecosystem tools like Hive and HBase are also being containerized.

Cloud-native technologies such as Docker and Kubernetes play a pivotal role in this evolution. Docker facilitates lightweight, consistent packaging of Hadoop components into isolated containers, ensuring reproducibility and portability across environments. Kubernetes enhances this setup by providing robust orchestration capabilities such as automatic scaling, load balancing, service discovery, and self-healing. These features align with the microservices paradigm, promoting modular, loosely coupled services that can be deployed and managed independently.

One of the most significant advantages brought by this transition is high availability. Kubernetes ensures that services remain continuously available by automatically rescheduling failed containers, distributing workloads across nodes, and supporting rolling updates without downtime. This contrasts sharply with traditional Hadoop deployments, where failures could lead to more significant service disruptions. As the Hadoop ecosystem evolves toward this cloud-native model, it introduces not only benefits like elasticity and availability but also new challenges in securing dynamic, distributed deployments — particularly in managing privileged access and maintaining consistent policies across ephemeral container instances.

IV. Proposed Model

This section introduces a novel access control model that integrates role-based, attribute-based, and usage control mechanisms to enhance security in Hadoop environments. To address the challenges posed by cloud-native architectures, the model incorporates Kubernetes RBAC [8] for managing access to cluster resources.

A. Concepts

Our model extends the usage control model by incorporating critical action approval. Building upon Park and Sandhu's foundation [17], we propose a comprehensive access control framework. This model encompasses roles, attributes, and organizational structures, with a focus on granular control over Hadoop and ecosystem services. By introducing mutable attributes and continuous evaluation, we address the dynamic nature of access requirements. The model's core components, including organizations, employees, services, objects, operations, authorizations, obligations and approvals are defined in Fig. 1 and described in Table I.

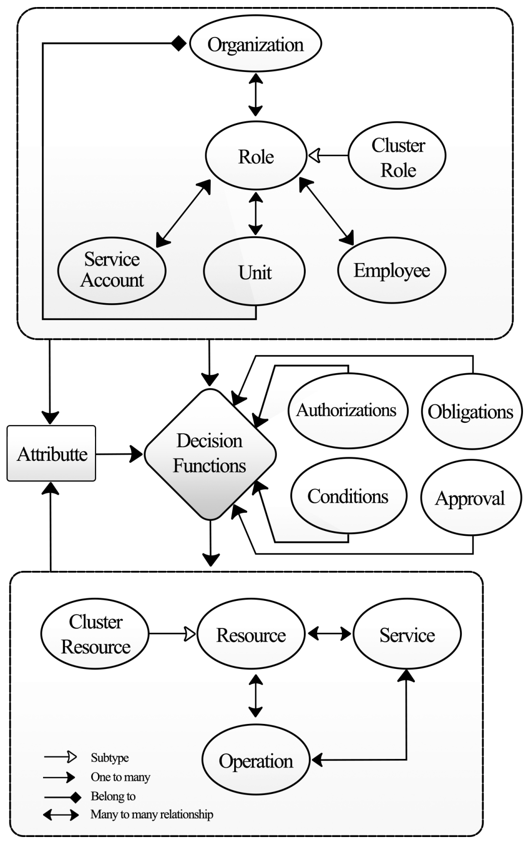

Fig. 1. Concepts of the proposed model.

TABLE I
FORMAL CONCEPTS

Basic sets and relations
ORG: finite set of organizations
$UNIT$: finite set of units
$ROLE$: finite set of roles
$SERVICE\_ACCOUNT$: finite set of service accounts
$EMPLOYEE$: finite set of employees
$ATTRIBUTE$: finite set of attributes
$OPERATION$: finite set of actions or operations on services and resources
$SERVICE$: finite set of Hadoop services and microservices
$RESOURCE$: finite set of resources and data
$SUBJECT = EMPLOYEE \cup SERVICE\_ACCOUNT$: set of subjects
$CLUSTER\_ROLE \in \mathbb{P}(ROLE)$: finite set of cluster roles
$CLUSTER\_RESOURCE \in \mathbb{P}(RESOURCE)$: finite set of cluster resources
$UH \in UNIT \rightarrow UNIT$: unit hierarchy
$RH \in ROLE \rightarrow ROLE$: role hierarchy
$UR \in UNIT \leftrightarrow ROLE$: assigning roles to units
$OU \in ORG \leftrightarrow UNIT$: assigning units to organizations
$EU \in EMPLOYEE \leftrightarrow UNIT$: assigning employees to units
$SaU \in SERVICE\_ACCOUNT \leftrightarrow UNIT$: assigning service accounts to units
$RA \in ROLE \leftrightarrow ATTRIBUTE$: assigning attributes to roles
$EA \in EMPLOYEE \leftrightarrow ATTRIBUTE$: assigning attributes to employees
$OA \in ORG \leftrightarrow ATTRIBUTE$: assigning attributes to organizations
$UA \in UNIT \leftrightarrow ATTRIBUTE$: assigning attributes to units
$SaA \in SERVICE\_ACCOUNT \leftrightarrow ATTRIBUTE$: assigning attributes to service accounts
$RSA \in RESOURCE \leftrightarrow ATTRIBUTE$: assigning attributes to resources
$SA \in SERVICE \leftrightarrow ATTRIBUTE$: assigning attributes to services
$OpA \in OPERATION \leftrightarrow ATTRIBUTE$: assigning attributes to operations
$RO \in RESOURCE \leftrightarrow OPERATION$: assigning operations to resources
$RS \in RESOURCE \leftrightarrow SERVICE$: assigning resources to services
$SO \in SERVICE \leftrightarrow OPERATION$: assigning operations to services
Access decision logic

We consider an object to be a service or a resource. For an access request (s, o, r), where s is a subject, o is an object, and r is an operation: Let $pa_1, \ldots, pa_m$ be authorization predicates; $ob_1, \ldots, ob_p$ be obligation actions; $pc_1, \ldots, pc_n$ be condition predicates; $app_1, \ldots, app_q$ be approval actions. An access policy is defined by two types of logical rules: access control decision rules and update rules (see Section IV-B3). Using the logic language in Section IV-B, we specify:

CR1 (Pre-decision):

$$permitaccess(s,o,r) \Rightarrow \blacklozenge \left(tryaccess(s,o,r) \wedge \bigwedge_{i=1}^{m} pa_i \wedge \bigwedge_{k=1}^{n} pc_k \wedge \bigwedge_{j=1}^{p} \blacklozenge ob_j \wedge \bigwedge_{f=1}^{q} \blacklozenge app_f \right)$$

CR2 (Continuous decision):

$$\Box \left(\neg \left(\bigwedge_{i=1}^{m} pa_i \wedge \bigwedge_{j=1}^{p} \left(\bigwedge_{a=1}^{k_j} pb_{j,a} \Rightarrow ob_j \right) \wedge \bigwedge_{f=1}^{q} \left(\bigwedge_{b=1}^{k_f} pb_{f,b} \Rightarrow app_f \right) \wedge \bigwedge_{k=1}^{n} pc_k \right) \wedge state(s,o,r) = accessing \Rightarrow revokeaccess(s,o,r) \right)$$

Where: $pb_{j,a}$ are predicates indicating when an ongoing obligation ob_j is required; $pb_{f,b}$ are predicates indicating when an ongoing approval app_f is required. An access request is permitted if all pre-decision components are satisfied; an ongoing access is maintained if all continuous decision components remain satisfied. Pre-decision and ongoing decision components may differ for the same access.

B. Formalism

Definition: Our logic model is a 6-tuple:

$$\mathscr{M} = (S, P_A, P_C, A_A, A_B, V_A),$$

where S is a sequence of system states, where each state represents a set of attribute-value assignments. Attributes are defined for various entities, including subjects (e.g., employees), organizations, resources, operations, and units. P_A is a finite set of authorization predicates derived from these entity attributes, while P_C is a finite set of condition predicates derived from system attributes. A_A is a finite set of usage control actions, A_B is a finite set of obligation actions, and V_A is a finite set of critical approval actions. A state is formally defined as a function that assigns values to attributes across the entities' attributes, including subjects, objects, and the system. The set A_A includes update actions and actions that modify the state of an access tuple (s, o, r).

A logical formula is constructed from predicates and actions using logical connectors and temporal operators, a fundamental aspect of temporal access control [22]. The language and its semantics are described in the formal usage control model [23].

1) A logical formula in our model is defined by the following BNF grammar:

$$\begin{aligned} \phi ::= \mathtt{a} \mid \mathtt{p}(t_1, \ldots, t_n) \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \\ \mid \phi \Rightarrow \phi \mid \Box\phi \mid \Diamond\phi \mid \bigcirc\phi \\ \mid \phi\, \mathcal{U}\, \phi \mid \blacksquare\phi \mid \blacklozenge\phi \mid \odot\phi \mid \phi\, \mathcal{S}\, \phi \end{aligned}$$

where a denotes an action, p a predicate of arity n, $t_1, ..., t_n$ are terms, and ϕ is a formula. If, in a sequence of states sq of a model M, a state s satisfies a formula ϕ, we write $M, sq, s \vDash \phi$. The satisfaction relation $\vDash$ is defined inductively on the

structure of ϕ and only for $s_0 \in sq$. Formally:
$M, sq, s_0 \vDash p \iff s_0[[p]], \quad p \in P_A \cup P_C$.
$M, sq, s_0 \vDash a \iff s_0[[a]]s_1, \quad a \in A_A \cup A_B \cup V_A$, and s_1 is the next state of s_0.
$M, sq, s_0 \vDash \neg\phi \iff M, sq, s_0 \nvDash \phi$.
$M, sq, s_0 \vDash \phi_1 \wedge \phi_2 \iff M, sq, s_0 \vDash \phi_1 \wedge M, sq, s_0 \vDash \phi_2$.
$M, sq, s_0 \vDash \phi_1 \implies \phi_2 \iff M, sq, s_0 \nvDash \phi_1 \vee M, sq, s_0 \vDash \phi_2$.
$M, sq, s_0 \vDash \Box\phi \iff \forall n \geq 0, M, sq, s_n \vDash \phi$.
$M, sq, s_0 \vDash \Diamond\phi \iff \exists n \geq 0, M, sq, s_n \vDash \phi$.
$M, sq, s_0 \vDash \bigcirc\phi \iff M, sq, s_1 \vDash \phi$.
$M, sq, s_0 \vDash \phi_1 U \phi_2 \iff \exists i \geq 0, M, sq, s_i \vDash \phi_2, \forall 0 \leq j < i, M, sq, s_j \vDash \phi_1$.
$M, sq, s_0 \vDash \blacksquare\phi \iff \forall n < 0, M, sq, s_n \vDash \phi$.
$M, sq, s_0 \vDash \blacklozenge\phi \iff \exists n < 0, M, sq, s_n \vDash \phi$.
$M, sq, s_0 \vDash \odot\phi \iff M, sq, s_{-1} \vDash \phi$.
$M, sq, s_0 \vDash \phi_1 S \phi_2 \iff \exists i < 0, M, sq, s_i \vDash \phi_2, \forall i < j \leq 0, M, sq, s_j \vDash \phi_1$.

For the sake of presentation, we will focus on the approval models, which form the core of our extension. The models of authorization, obligations and conditions retain their basic definitions assigned in the work of Park and Sandhu.

2) Approval as a Security Control: Approval is a control action performed by a supervisor, resulting in either an approve or reject outcome. This decision directly influences the ability of a user or service account to execute a job or an operation, ensuring fraud prevention, privilege enforcement, and compliance with security policies. Approval mechanisms are crucial for enforcing policies that require human judgment or multi-level authorization, especially when handling high-risk actions such as job execution on sensitive datasets. To integrate approval mechanisms into Hadoop's cloud-native access control, we adopt UCON-based approval models, which define when and how approval actions occur.

Pre-Approval – Required before a subject (user or service account) gains access to an object (Hadoop job submission). This ensures that only authorized workloads originating from verified Kubernetes namespaces, trusted CI/CD pipelines, or approved execution environments can submit jobs. Example: A System Administrator must approve any job that exceeds predefined memory or CPU thresholds.

Ongoing Approval – Required during the job execution process. This ensures continuous validation of runtime behavior, preventing unauthorized privilege escalation or resource misuse. Example: If a job unexpectedly exceeds its allocated resources, an automatic approval request is sent to a supervisor to allow or terminate execution dynamically.

As with the authorization models defined in Usage Control, we distinguish between different approval models depending on when actions are performed, as shown below. The specification for each approval model is given below:
$preV0$: An access decision is determined by a pre-approval and there is no attribute update before, during or after: $\blacklozenge app_1 \wedge ... \wedge \blacklozenge app_i \implies permit(s, o, r)$
$tryaccess(s, o, r) \wedge permit(s, o, r) \implies \bigcirc(permitaccess(s, o, r))$,

where $app_1, ..., app_i$ represent approval actions for access (s, o, r). This rule requires that an access can only be granted if all approval actions have been performed.

Example (preV0): A "Data Scientist" service account requests access to the sensitive dataset "customer-pii-data" in Hadoop. Access is granted only after explicit approvals from both the "Security Analyst" and "Compliance Officer," with no attribute updates before, during, or after the access request. Additionally, access is restricted based on organizational membership, time constraints, project involvement, and security conditions: $\blacklozenge approve\_security(s, o, r) \wedge \blacklozenge approve\_compliance(s, o, r) \wedge org\_unit(s, "DataScienceUnit") \wedge time\_window(08{:}00{-}18{:}00) \wedge project\_assigned(s, \text{"PII-required"}) \wedge \neg policy\_violation(s) \wedge \neg security\_alert("customer-pii-data") \implies permit(s, o, r)$;
$tryaccess(s, o, r) \wedge permit(s, o, r) \implies \bigcirc(permitaccess(s, o, r))$, where: $\blacklozenge approve\_security(s, o, r)$ represents explicit approval from the "Security Analyst", $\blacklozenge approve\_compliance(s, o, r)$ represents explicit approval from the "Compliance Officer", $org\_unit(s, "DataScienceUnit")$ ensures the subject belongs to the correct organizational unit, $time\_window(08{:}00{-}18{:}00)$ restricts access to business hours, $project\_assigned(s, "PII-required")$ verifies that access is linked to an approved project, $\neg policy\_violation(s)$ prevents access if the subject has previous policy violations, $\neg security\_alert("customer-pii-data")$ denies access if a security alert exists on the dataset.
$preV1$: An access control decision is made by pre-approval, and one or more attributes of the subject or object are updated prior to use: $\blacklozenge app_1 \wedge ... \wedge \blacklozenge app_i \implies permit(s, o, r)$;
$permitaccess(s, o, r) \implies permit(s, o, r) \wedge \blacklozenge(tryaccess(s, o, r) \wedge \Diamond(preupdate(attribute)))$.

Example: A "Data Scientist" service account requests access to the sensitive dataset "customer-pii-data" in Hadoop. Access is granted only after explicit approvals from both the "Security Analyst" and "Compliance Officer". However, before the dataset can be used, the approval validity must be updated to ensure time-limited access: $\blacklozenge approve\_security(s, o, r) \wedge \blacklozenge approve\_compliance(s, o, r) \implies permit(s, o, r)$;
$permitaccess(s, o, r) \implies permit(s, o, r) \wedge \blacklozenge(tryaccess(s, o, r) \wedge \Diamond(preupdate(approval\_valid(s, o, now() + duration))))$,

where: $preupdate(approval\_valid(s, o, now() + duration))$ updates the approval validity before usage.
$preV2$: An access control decision is determined by pre-approval, and one or more attributes of the subject or object are updated during use: $permitaccess(s, o, r) \implies \blacklozenge tryaccess(s, o, r) \wedge (\blacklozenge app1 \wedge \blacklozenge app2 \wedge ... \wedge \blacklozenge appi)$
$permitaccess(s, o, r) \implies \Diamond(onupdate(attribute) \wedge \Diamond endaccess(s, o, r))$.

Example: Let's complete the previous example saying that access is allowed, however, the number of times access is allowed must be limited to prevent excessive or unauthorized repeated access. *Pre-Approval:* $\blacklozenge approve\_security(s, o, r) \wedge$

$\blacklozenge approve\_compliance(s,o,r) \implies permit(s,o,r)$; *Usage Tracking:* $permitaccess(s,o,r) \implies \blacklozenge tryaccess(s,o,r) \wedge \Diamond(onupdate(usage\_count(s,o,r)+1))$; *Access Restriction:* $usage\_count(s,o,r) \leq max\_usage$;

where: $onupdate(usage\_count(s,o,r)+1)$ increments the usage count on each access, $usage\_count(s,o,r) \leq max\_usage$ restricts access if the count exceeds the threshold.

$preV3$: An access control decision is determined by a pre-approval, and one or more attributes of the subject or object are updated after use: $\blacklozenge app_1 \wedge \cdots \wedge \blacklozenge app_i \implies permit(s,o,r)$; $permitaccess(s,r,o) \implies permit(s,o,r) \wedge \blacklozenge(tryaccess(s,o,r)))$; $endaccess(s,o,r) \implies \Diamond(postupdate(attribute))$.

Example: In addition to the previous constraints, we introduce Usage History Tracking: The total number of times the dataset has been accessed should be recorded. Pre-Approval: $\blacklozenge approve\_security(s,o,r) \wedge \blacklozenge approve\_compliance(s,o,r) \implies permit(s,o,r)$, usage tracking: $permitaccess(s,o,r) \implies \blacklozenge tryaccess(s,o,r) \wedge \Diamond(onupdate(usage\_count(s,o,r)+1))$, usage History Tracking: $endaccess(s,o,r) \implies \Diamond(postupdate(total\_accesses(o)+1))$.

$onV0$: Access control is checked and the decision is determined by an approval during access, and there is no attribute update before, during or after use: $permitaccess(s,o,r) \implies \Box(\neg(app_1 \wedge ... \wedge app_i) \wedge (state(s,o,r) = accessing) \implies revokeaccess(s,o,r))$.

$onV1$: Access control is checked and the decision is determined by an approval during access, and one or more attributes of the subject or object are updated before use: $permitaccess(s,o,r) \implies \blacklozenge(tryaccess(s,o,r) \wedge \Diamond(preupdate(attribute)))$; $permitaccess(s,o,r) \implies \Box(\neg(app_1 \wedge \cdots \wedge app_i) \wedge (state(s,o,r) = accessing) \implies revokeaccess(s,o,r))$.

$onV2$: Access control is checked and the decision is determined by an approval during access, and one or more attributes of the subject or object are updated during use: $permitaccess(s,o,r) \implies \Box(\neg(app_1 \wedge \cdots \wedge app_i) \wedge (state(s,o,r) = accessing) \implies revokeaccess(s,o,r))$; $endaccess(s,o,r) \vee revokeaccess(s,o,r) \implies \blacklozenge(permitaccess(s,o,r) \wedge \Diamond(onupdate(attribute)))$.

$onV3$: Access control is checked and the decision is determined by an approval during access, and one or more attributes of the subject or object are updated after use: $permitaccess(s,o,r) \implies \Box(\neg(app_1 \wedge \cdots \wedge app_i) \wedge (state(s,o,r) = accessing) \implies revokeaccess(s,o,r))$. If the subject terminates access : $endaccess(s,o,r) \implies \Diamond(postupdate(attribute))$. When an access is revoked by the system: $revokeaccess(s,o,r) \implies \Diamond(postupdate(attribute))$.

Example: Ongoing approval for fraud prevention in Hadoop. A "Data Scientist" service account accesses the sensitive "customer-pii-data" dataset in Hadoop. To prevent unauthorized usage during access, the system enforces ongoing approval validation. During access, the system continuously checks whether the required approvals remain valid. If any of the approving entities withdraw their approval or if an explicit re-approval is required at set intervals, access is immediately revoked to prevent abuse. This mechanism is crucial in preventing a compromised Docker container from impersonating the service account and maintaining unauthorized access indefinitely. Ongoing Approval Requirement: $permitaccess(s,o,r) \implies \Box\big((state(s,o,r) = accessing) \wedge \neg(approve\_security(s,o,r) \wedge approve\_compliance(s,o,r)) \implies revokeaccess(s,o,r)\big)$. This ensures that access is actively monitored and revoked if the necessary approvals are not continuously maintained, making it impossible for an attacker to persist in an unauthorized session.

3) Access Policies: Our access control model employs a policy-based approach, defined by logical formulae derived from a fixed rule set. Access decisions are determined holistically, considering authorizations, obligations, conditions, and approvals. These elements are specified by predicates on entities and their attributes. An access request, represented as a tuple (subject, object, request), is evaluated against a set of authorization, obligation, condition, and approval predicates. Access control policies are defined by logical rules governing pre and continuous decision-making.

For an access (s,o,r), let $pa_1,\ldots,pa_m$ be a set of authorization predicates, $ob_1,\ldots,ob_p$ a set of obligation actions, $pc_1,\ldots,pc_n$ be a set of condition predicates, and $app_1,\ldots,app_q$ be a set of approval actions. An access policy can be specified by two types of logical rules: control rules and attribute update rules. The following control rules (CR) are specified for the *pre-decision* and *ongoing-decision* of a single-use process, respectively: $CR_1 : permitaccess(s,o,r) \Rightarrow \blacklozenge\big(tryaccess(s,o,r) \wedge \bigwedge_{i=1}^{m} pa_i \wedge \bigwedge_{k=1}^{n} pc_k \wedge \bigwedge_{j=1}^{p} \blacklozenge ob_j \wedge \bigwedge_{f=1}^{q} \blacklozenge app_f\big)$

$CR_2 : \Box\Big(\neg\big(\bigwedge_{i=1}^{m} pa_i \wedge \bigwedge_{j=1}^{p}(pb_{j1} \wedge \cdots \wedge pb_{jK_j} \Rightarrow ob_j) \wedge \bigwedge_{f=1}^{q}(pb_{f1} \wedge \cdots \wedge pb_{fL_f} \Rightarrow app_f) \wedge \bigwedge_{k=1}^{n} pc_k\big) \wedge (state(s,o,r) = accessing) \Rightarrow revokeaccess(s,o,r)\Big)$

where $1 \leq i \leq m$, $1 \leq j \leq p$, $1 \leq k \leq n$, $1 \leq f \leq q$, $pb_{j1}, ..., pb_{jK_j}$ are predicates for determining when an ongoing obligation ob_j is required, and pb_{f1}, ... , pb_{fL_f} are predicates for determining when an ongoing approval app_f is required. An access request can be granted if its pre-decision components are true, while an ongoing access can be continued if all ongoing decision components are true. For an access, its pre-decision and ongoing decision components may or may not be identical. The types of attributes update actions can be specified by update rules (URs) as defined in the UCON formal specification. The completeness and soundness properties of this policy specification language have been proved in the formal UCON specification [23].

V. Logical Architecture

The usage control model promotes the concept of continuity in the enforcement of policies. Indeed, the mutability of attributes introduces the need to carry out the usage control policy evaluation process continuously while an access is in progress. This is because the values of the attributes that previously authorized access can change so that the access right is no longer valid. In this case, the access is revoked.

A. Logical Architecture based on Open Policy Agents and the Approval of Critical Requests

We propose a logical architecture, with Hadoop components and ecosystem services packaged as microservices, that uses Open Policy Agent (OPA) [24] as the policy decision point and requires approval for critical requests. OPA's declarative Rego language facilitates policy management, enabling dynamic policy updates. To ensure real-time policy enforcement, we incorporate OPAL for policy change detection and propagation. Fig .2. illustrates the system components. The Open policy agents Administrative Layer (OPAL) is the central policy management component. It maintains policy and data repositories, accessible through REST APIs and a Pub/Sub channel for real-time updates. Edge-based OPAL clients subscribe to these feeds, aggregating data and synchronizing with policy changes. To facilitate dynamic policy updates, a Git-based repository and webhook integration are employed.

The Critical Request Approval Service is a crucial extension, handling access request validation. In this case the microservice (Hadoop component or ecosystem service) interacts with the OPA engine to determine authorization based on real-time policy and data and trigger the approval (dual control) in case of critical access.

This architecture potentially uses Pub/Sub channels for communication between Client <> Server and Server <> Server. The Pub/Sub system can use websockets, Redis, Kafka or any other message broker to synchronize all OPAL servers and clients. The control flow for critical actions is as follows:

$$\text{Users} \xrightarrow{\text{mapsto}} \text{App} \xrightarrow{\text{mapsto}} \text{PolicyAgent}$$
$$\text{then} \quad \text{App} \xrightarrow{\text{mapsto}} \text{Approval-Service}$$

Users interact with the microservice by triggering authorization requests which the microservice resolves with the OPA agent. The OPA agent also returns the severity of the request in the response, indicating whether or not it is critical according to the configured security policy. In the event of a critical request, the microservice in question triggers the hierarchical approval process for the request by calling the approval service via the dedicated REST API.

State Management. OPA's state management capabilities are facilitated through enriched input requests and data change notifications. Policies can dynamically adapt by accessing historical data and leveraging session-based state representations. Attribute modifications trigger policy reevaluations, streamlining the decision-making process compared to U-XACML. Moreover, OPA's sidecar deployment architecture enhances performance by minimizing latency.

B. System Flowchart Overview

The flow diagram in Fig. 3 illustrates the process of a user interacting with the Hadoop/Ecosystem Service, focusing on the authorization and approval mechanisms. Initially, the user sends a usage request to the service (1). The service then forwards an authorization request to the OPA (Open Policy Agent) (2), which evaluates the request against policies stored in the Policy Store (3). These policies are provisioned by OPAL (Policy Sync). The OPA returns a decision (ALLOW/DENY) along with any relevant metadata to the service (4). If the OPA decision is to DENY or ALLOW for non-critical requests (5a), the service responds directly to the user. However, for critical usage requests, the service forwards the request to an Approval Service (5b). The Approval Service then requests approval from designated Approvers (6), who can either Approve or Reject the request (7). The Approval Service sends the approval result back to the Hadoop/Ecosystem Service (8), which finally relays the overall decision (ALLOW/DENY) to the user (9).

C. Architecture of our Model on Kubernetes

Our simplified architecture proposes a modular solution for integrating the model on Kubernetes. The master node, which centralizes the management of the cluster, hosts the gatekeeper. This admission controller, based on OPA, intercepts and validates requests to create, modify or delete Kubernetes resources (pods, services, configurations, etc.). Administrators can define specific policies using OPA constraints, allowing or disallowing certain deployment behaviors. Hadoop services and their components are deployed on separate nodes. Each Hadoop pod co-exists with an OPA container as a sidecar. This configuration allows OPA to evaluate policies locally, improving performance by avoiding network latency. OPA policies can include a requireApproval flag to indicate whether a request should be subject to an approval process.

The Approval microservice manages the validation workflow for critical requests. It interacts with the hierarchical superiors to obtain their approval. The decision, which is binary (accept or reject), is taken by the last validating instance. The service also allows parallel validations with configurable decision rules (for example, priority to the most frequent decision or to the decision of the highest authority).

The administration service centralizes the configuration of the organizational structure and validation policies.

VI. Implementation

This demonstration shows dynamic, rule-based anomaly detection implemented using Big Data tools and technologies from the Hadoop ecosystem. The solution infrastructure is based on Docker containers, orchestrated by Docker Compose to ensure robust and scalable execution. Data is processed using Big Data frameworks such as Apache Flink and Apache Kafka.

A. Components Diagram

Our system consists of several key components: a backend, a frontend, a Flink processing engine, and an OPA authorization agent. The backend exposes a REST API for managing rules, controlling execution, and handling user interactions. It relays frontend actions to Flink via a Kafka topic (”control”) and

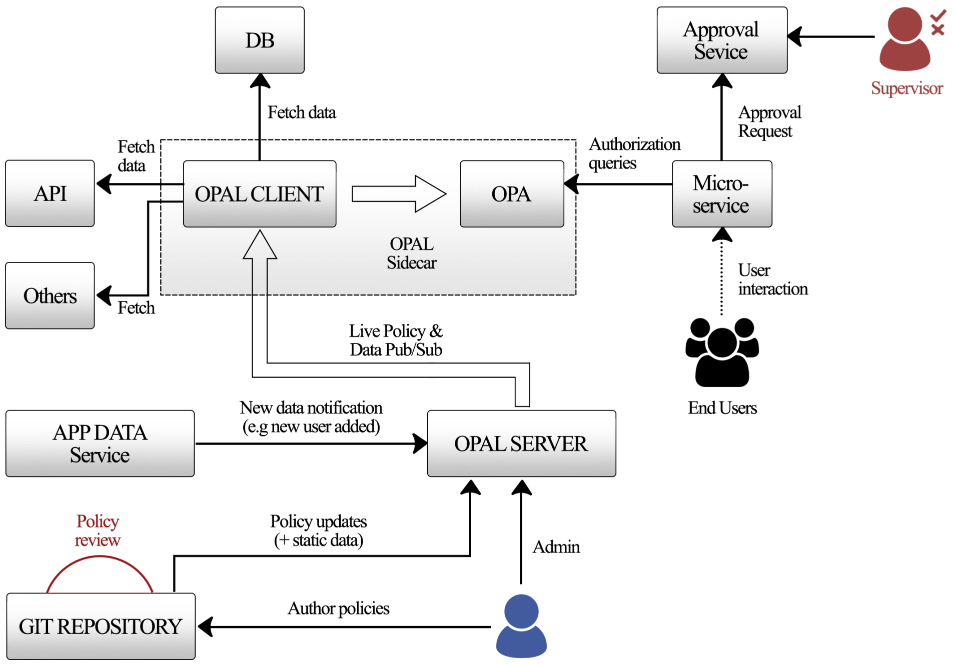

Fig. 2. Proposed logical architecture.

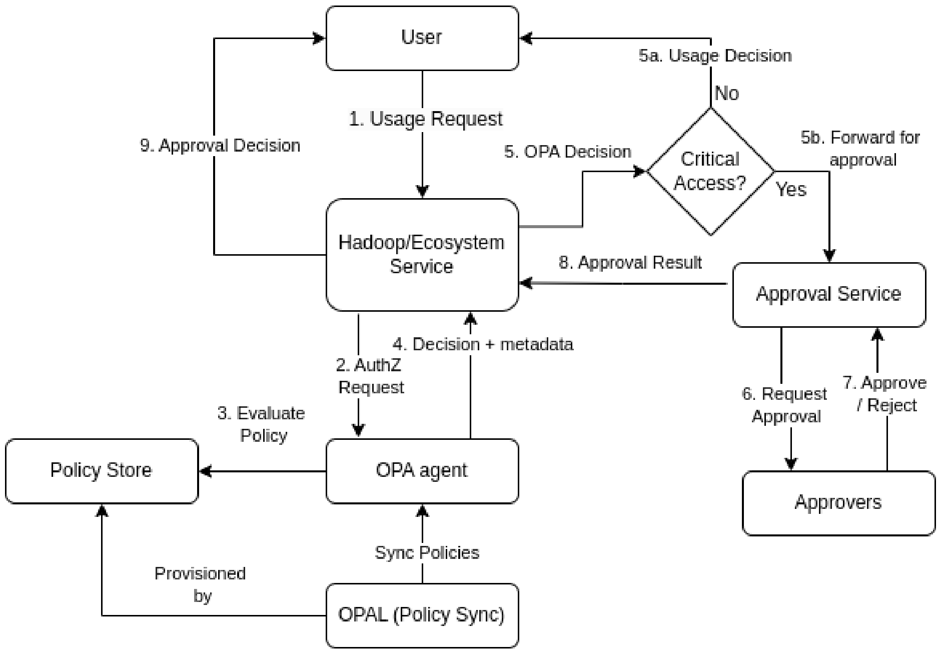

Fig. 3. Flow diagram of the security system.

generates simulated security events (e.g., access and login/logout events), which are streamed to Flink through an ”Events” topic. Flink processes these events and produces alerts, which the backend retrieves from the ”Alerts” topic and forwards to the frontend via WebSockets. Users can submit and monitor Flink jobs through the interface, while access decisions are enforced by querying the OPA agent. Controlled actions include job submission, cancellation, status viewing, rule management, and access to the detection dashboard. Additionally, usage control prevents excessive resource consumption by limiting a user’s memory and CPU usage. All components are deployed using Docker containers.

B. Proof of Concepts: Preventing Unauthorized Job Submission and Impersonation

In a distributed data processing environment like Hadoop and Flink, unauthorized job submissions and service account impersonation pose significant security risks. Attackers or malicious insiders may attempt to exploit misconfigured permissions to submit unauthorized jobs, impersonate privileged service accounts, or overconsume system resources. To mitigate these risks, our access control policy enforces ongoing authorization checks and usage control constraints during job execution. Firstly, job submission requests are only permitted if the user has explicit submission rights, possesses a valid resource quota, and their cluster role includes the required permissions. Additionally, ongoing validation ensures that a job continues to meet authorization criteria throughout its lifecycle. This prevents scenarios where an initially authorized job could later become unauthorized due to changes in role assignments, quota exhaustion, or compliance violations. Secondly, impersonation protection mechanisms prevent unauthorized services or users from running jobs under privileged service accounts. Each job submission is dynamically validated against contextual attributes such as active user sessions, source IP validation, and container identity verification. This ensures that jobs are not executed by unauthorized entities attempting to exploit service account credentials. Furthermore, to prevent abuse, we enforce usage control policies such as limiting simultaneous active jobs per user, tracking historical job execution patterns, and restricting dynamic segregation of duties (SoD)—ensuring that a user who submits a job cannot approve or modify their own execution pipeline. These policies, combined with real-time monitoring and revocation mechanisms, ensure continuous enforcement of security constraints, reducing the risk of unauthorized access, excessive resource consumption, and privilege escalation.

1) Anatomy of the OPA Policy Engine: The core components of the OPA policy engine are:

- Input: The JSON-enriched request containing attributes of users, actions, resources, and access contexts.
- Policy: The set of rules written in Rego that define the authorization logic.
- Data: External context stored in JSON, such as user organizations, units, roles and permissions.
- Decision: The outcome (e.g., `allow` or `deny`) is based on the evaluation of the input against the policy. In our case, the OPA outcome includes additional information related to approvals and obligations. The policy enforcer will then trigger the necessary approvals and fulfill those obligations before rendering the final response to the requester. To avoid tight coupling and overly complex policies while keeping the OPA policy standalone, the OPA engine does not interact directly with the approval system. Instead, the policy enforcer, which controls access to the Hadoop components and ecosystem services, manages these interactions.

2) Rego Policy Implementation: The Rego policy in Listing 1 enforces fine-grained access control in Hadoop based on user roles, permissions, and context. It grants access to data owners, analysts, and engineers when role-based conditions are met, while job submissions require valid permissions, quotas, and active sessions. Deny rules enhance security by limiting concurrent jobs and preventing impersonation. Service accounts have restricted access for Kafka ingestion and Flink execution in Kubernetes. For high-risk actions, OPA signals approval requirements instead of enforcing them. The enforcer triggers approval workflows, while an external service handles delegation, escalation, and workflow execution. Decoupling approval logic from OPA ensures scalability and flexibility. The approval service is not detailed but exposes REST APIs for managing approvals.

Listing 1. OPA Policy

```
package hadoop

default allow := false

# Allow deletion if user is the data owner and has
   high clearance
allow if {
  input.action == "delete_dataset"
  user_is_data_owner
  user_has_high_clearance
  not approval_required["delete_dataset"]
}

# Require approval for high-risk deletions
approval_required["delete_dataset"] if {
  data.resource_attrs[input.resource].criticality
  == "high"
}

# Partial evaluation: OPA signals approval
   requirement instead of enforcing it
approval_metadata := {
  "requester": input.user,
  "action": input.action,
  "resource": input.resource,
  "approver": data.approvers[input.action],
  "time_limit": data.approval_time_limits[input.
  action]
} if approval_required["delete_dataset"]

# User role checks
user_is_data_owner if data.user_attrs[input.user].
  role == "data_owner"
user_has_high_clearance if data.user_attrs[input.
  user].clearance == "high"
```

VII. Evaluation and Comparison

The evaluation of access control systems follows the guidelines set out in NIST IR 7874 [25], with a primary focus on administration and enforcement. Performance and support are not considered, as they fall outside the scope of this prototype. Administration is crucial for managing costs, efficiency, and ease of operation within access control systems. While OPA logs decisions, it lacks built-in audit trails and automated privilege discovery, which limits its auditing capabilities. Nevertheless, Rego's flexibility streamlines privilege assignment, although the absence of a user interface adds complexity. One of OPA's key strengths lies in its policy writing capabilities, facilitated by an expressive syntax. Policy updates are efficiently managed through OPAL, and while privilege delegation is supported, it requires custom policy definitions. OPA integrates effectively with various applications and protocols, enabling multi-host access control and rule enforcement across different system layers.

In terms of enforcement, OPA allows for complex rule combinations, ensuring robust policy application. Implicit bypassing is not permitted, though predefined emergency exceptions can be configured. The principle of least privilege is enforced via attribute-based policies, though careful policy design is necessary to avoid privilege escalation. Additionally, OPA enforces separation of duties, mitigating access conflicts and ensuring compliance with security constraints through fine-grained policy definitions. Conflict resolution is achieved through rule priorities and evaluation order, ensuring consistent decision-making.

We assessed our model by classifying evaluation criteria as critical, optional, or supplementary according to their relevance to the case study. As shown in Table II, the model satisfies all critical criteria and supports several essential optional and supplementary features.

We compared our approach with U-XACML [26], [27], BigUCON, and RQ-UCON [12], [13], which extend access control with usage control principles. U-XACML, based on XACML, employs a centralized Policy Decision Point (PDP), which limits scalability and creates bottlenecks in cloud-native environments. In contrast, BigUCON and RQ-UCON enhance Hadoop security with fine-grained usage control, but their focus is limited to attribute-based constraints, without incorporating organizational structures or approval-based enforcement.

Our approach introduces three key advancements: First, it integrates hierarchical approval through organization-based validation, ensuring that critical access requests undergo multi-step authorization within the organization. Second, it strengthens security by incorporating policy review and approval mechanisms, leveraging OPA's policy-as-code paradigm to facilitate systematic policy definition, review, and enforcement, ensuring explicit auditability of access decisions. Finally, unlike U-XACML's centralized PDP, our model adopts a decentralized policy enforcement strategy, distributing decision-making closer to access points, which improves latency, fault tolerance, and scalability, particularly in cloud-native environments.

As shown in Table II, our model successfully meets all critical evaluation criteria while supporting key optional and supplementary features. By integrating organizational validation, approval-based enforcement, and decentralized decision-making, it enhances security, flexibility, and adaptability for dynamic, cloud-native infrastructures requiring hierarchical and policy-driven access control.

VIII. Discussion

The proposed access control model enhances security in cloud-native Hadoop environments by integrating stateful authorization with an approval-based mechanism that also enforces Separation of Duties (SoD). Traditional access control models, such as RBAC and ABAC, lack the adaptability required for dynamic, containerized deployments. By leveraging Usage Control (UCON) principles and multi-step authorization, our approach enables fine-grained, context-aware security enforcement that aligns with the evolving nature of cloud-native architectures. The integration with Open Policy Agent (OPA) ensures compatibility with modern cloud infrastructures, providing a scalable and flexible solution for securing distributed Big Data workflows while mitigating risks associated with privilege escalation and insider threats through enforced approval steps.

However, the proposed model has certain limitations. A key limitation is that it does not explicitly define how critical actions requiring approval should be designated. While one could leverage risk-based calculations, such as those proposed in RQ-UCON [13], to dynamically classify high-risk actions, this paper does not prescribe a specific methodology for selecting these actions. The effectiveness of the approach, therefore, depends on external risk assessment strategies or predefined organizational policies. For instance, in a previous work [28], we proposed a method for detecting suspicious behavior using entropy-based calculations. Such an approach could be integrated into the current model to dynamically identify critical actions based on deviations in observed behavior, but this is not explored in this paper.

While this model introduces approval as an additional security control, its implementation falls outside the scope of this work. This contribution nevertheless extends the approval framework we established in our request-based access control paper [29].

Additionally, we do not address secured service-to-service management in this work. We believe this can be handled using service meshes [30].

Technically, the OPA agent could directly interact with the approval service, as described with the interaction of PDP and PXP introduced by [31], but this approach complicates policy testing. Instead, we opted for partial evaluation, where OPA returns a partially evaluated decision to the enforcer (the service), which then calls the approval service for further evaluation. This separation improves flexibility and maintainability while ensuring dynamic approval handling.

Despite these limitations, the proposed model provides a foundational framework for enhancing security in cloud-native Big Data systems. It offers a balance between flexibility, scalability, and security enforcement, laying the groundwork for future research into adaptive authorization mechanisms and efficient approval workflows.

IX. Conclusion and Future Work

The aim of this work was to model access control for the Hadoop ecosystem in cloud-native environments. To do this, we first reviewed container-based architectures within the Hadoop ecosystem, highlighting the evolution and associated security challenges. This informed the design of a conceptual access control model tailored to modern technologies. We then proposed a logical architecture based on the Open Policy Agent, integrating usage control and multi-step authorization to secure privileged actions. Our experimental implementation with Docker, Apache Kafka, and Apache Flink demonstrated the model's feasibility in a dynamic, distributed context.

The results confirm that traditional access control approaches fall short in dynamic, containerized Big Data environments. In contrast, our proposed model offers a more adaptive and granular mechanism by combining stateful usage control with organizational approvals. This approach improves both the security posture and operational oversight of cloud-native Hadoop platforms, particularly by mitigating risks related to unbounded privileges and rogue container deployment.

Two key directions for future work have emerged. The first is extending the implementation to Kubernetes, enabling us

TABLE II
COMPARISON OF OUR MODEL WITH U-XACML, BIGUCON, AND RQ-UCON BASED ON NIST CRITERIA.

Metric	Type	Our Model	U-XACML	BigUCON	RQ-UCON
Administration					
Audit	Supplementary	x	x	x	x
Privilege/Capability Discovery	Supplementary	x	x	x	x
Ease of Privilege Assignment	Optional	✓	✓	x	x
Syntactic and Semantic Support for Access Rules Specification	Critical	✓	✓	✓	✓
Policy Management	Supplementary	✓	x	x	x
Delegation of Administrative Capabilities	Supplementary	x	✓	x	x
Configuration Flexibility within Existing Systems	Supplementary	✓	x	x	x
Horizontal Control Scope	Optional	✓	✓	✓	✓
Vertical Control Scope	Optional	✓	✓	✓	✓
Enforcement					
Policy Combination, Composition, and Constraints	Critical	✓	✓	✓	✓
Bypass Prevention	Supplementary	x	x	x	x
Support for the Principle of Least Privilege	Optional	✓	✓	✓	✓
Separation of Duties (SoD)	Critical	✓	✓	✓	✓
Security (Containment and Constraints)	Critical	✓	✓	✓	✓
Conflict Resolution or Prevention	Critical	✓	✓	✓	✓
Operational/Situational Awareness	Optional	✓	✓	✓	✓
Control Granularity	Critical	✓	✓	✓	✓
Expression Properties (Policy/Model)	Critical	✓	✓	✓	✓
Adaptability to Policy Implementation and Evolution	Critical	✓	✓	✓	✓
Partial Evaluation of Access Requests	Supplementary	✓	x	x	x
Reactive Policy Updates	Supplementary	✓	x	x	x
Approval of Critical Actions	Supplementary	✓	x	x	x

to assess the full potential of the model — from deployment control using OPA Gatekeeper to runtime access to Hadoop services. The second is enhancing the approval service by integrating real-time detection of suspicious behaviors, thereby reducing reliance on manual approval and improving responsiveness in dynamic threat contexts.

REFERENCES

[1] de Souza Granh and R. G. Duarte, "Hadoop," pp. 913–917, 2019. [Online]. Available: https://doi.org/10.1007/978-3-319-77525-8_36

[2] *Container Technology*. Singapore: Springer Nature Singapore, 2023, pp. 295–342. [Online]. Available: https://doi.org/10.1007/978-981-19-3026-3_7

[3] K. S. Murugiah Souppaya, John Morello, "Application container security guide," *NIST*, September 2017. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-190

[4] M. Gupta, F. Patwa, and R. Sandhu, "Object-tagged rbac model for the hadoop ecosystem," 06 2017, pp. 63–81. [Online]. Available: https://doi.org/10.1007/978-3-319-61176-1_4

[5] G. Maanak, P. Farhan, and S. Ravi, "An attribute-based access control model for secure big data processing in hadoop ecosystem," in *Proceedings of the Third ACM Workshop on Attribute-Based Access Control*, ser. ABAC'18. New York, NY, USA: Association for Computing Machinery, 2018, p. 13–24. [Online]. Available: https://doi.org/10.1145/3180457.3180463

[6] H. Ait, H. Belhadaoui, F. H. Reda, and O. Malassé, "A role-attribute based access control model for dynamic access control in hadoop ecosystem," *IAENG International Journal of Computer Science*, vol. 50, 03 2023. [Online]. Available: https://www.iaeng.org/IJCS/issues_v50/issue_1/IJCS_50_1_21.pdf

[7] T. Rugendo and A. Kahonge, "Access control model for container based virtual environments," *International Journal of Computer Applications*, vol. 174, pp. 21–29, 02 2021. [Online]. Available: https://doi.org/10.5120/ijca2021921091

[8] G. Rostami, "Role-based access control (rbac) authorization in kubernetes," *J. ICT Stand.*, vol. 11, no. 3, pp. 237–260, 2023. [Online]. Available: https://doi.org/10.13052/jicts2245-800X.1132

[9] Z. B. T. Ramaswamy Chandramouli (NIST), "A zero trust architecture model for access control in cloud-native applications in multi-cloud environments," *CSRC, NIST*, September 2023. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-207A

[10] S. Bahloul, K. Bessaoud, and M. Abid, *H-RCBAC: Hadoop Access Control Based on Roles and Content*, 01 2022, pp. 423–437. [Online]. Available: https://doi.org/10.1007/978-981-16-3637-0_30

[11] H. A. Idar, K. Aissaoui, H. Belhadaoui, and R. F. Hilali, "Dynamic data sensitivity access control in hadoop platform," in *2018 IEEE 5th International Congress on Information Science and Technology (CiSt)*, 2018, pp. 105–109. [Online]. Available: https://doi.org/10.1109/CIST.2018.8596381

[12] G. Baldi, Y. Diaz-Tellez, T. Dimitrakos, F. Martinelli, C. Michailidou, P. Mori, O. Osliak, and A. Saracino, "Session-dependent usage control for big data," *J. Internet Serv. Inf. Secur.*, vol. 10, no. 3, pp. 76–92, 2020. [Online]. Available: https://doi.org/10.22667/JISIS.2020.08.31.076

[13] R. Jiang, X. Chen, Y. Yu, Y. Zhang, and W. Ding, "Risk and ucon-based access control model for healthcare big data," *J. Big Data*, vol. 10, no. 1, p. 104, 2023. [Online]. Available: https://doi.org/10.1186/s40537-023-00783-8

[14] S. Baccouri and T. Abdellatif, "Big-abac: Leveraging big data for adaptive, scalable, and context-aware access control," *CMES - Computer Modeling in Engineering and Sciences*, vol. 143, no. 1, pp. 1071–1093, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1526149225001080

[15] R. Wang, C. Li, K. Zhang, and B. Tu, "Zero-trust based dynamic

access control for cloud computing," *Cybersecur.*, vol. 8, no. 1, p. 12, 2025. [Online]. Available: https://doi.org/10.1186/s42400-024-00320-x

[16] M. A. Poltavtseva, A. A. Podorov, and E. B. Aleksandrova, "Verification of access control in big data systems using temporal logics," *Autom. Control. Comput. Sci.*, vol. 58, no. 8, pp. 1311–1317, 2024. [Online]. Available: https://doi.org/10.3103/S0146411624700974

[17] A. Lazouski, F. Martinelli, and P. Mori, "Usage control in computer security: A survey," *Computer Science Review*, vol. 4, pp. 81–99, 05 2010. [Online]. Available: https://doi.org/10.1016/j.cosrev.2010.02.002

[18] T. R. Rao, P. Mitra, R. Bhatt, and A. Goswami, "The big data system, components, tools, and technologies: a survey," *Knowl. Inf. Syst.*, vol. 60, no. 3, pp. 1165–1245, 2019. [Online]. Available: https://doi.org/10.1007/s10115-018-1248-0

[19] C. N. C. Foundation, "Cloud native," 2024, [Online; accessed 15-July-2024]. [Online]. Available: https://www.cncf.io/about/who-we-are/

[20] Uber, "Containerizing apache hadoop infrastructure at uber," 2021, [Online; accessed 15-July-2024]. [Online]. Available: https://www.uber.com/en-TZ/blog/hadoop-container-blog/

[21] Comcast, "Yarn on kubernetes," 2024, [Online; accessed 15-July-2024]. [Online]. Available: https://github.com/Comcast/kube-yarn

[22] C. Bettini, *Temporal Access Control*. Cham: Springer Nature Switzerland, 2025, pp. 2594–2596. [Online]. Available: https://doi.org/10.1007/978-3-030-71522-9_831

[23] X. Zhang, F. Parisi-Presicce, R. Sandhu, and J. Park, "Formal model and policy specification of usage control," *ACM Trans. Inf. Syst. Secur.*, vol. 8, no. 4, p. 351–387, nov 2005. [Online]. Available: https://doi.org/10.1145/1108906.1108908

[24] C. N. C. Foundation, "Open policy agent (OPA)," 2024, [Online; accessed 15-July-2024]. [Online]. Available: https://www.cncf.io/projects/open-policy-agent-opa/

[25] V. Hu and K. Scarfone, "Guidelines for access control system evaluation metrics," 09 2012. [Online]. Available: http://dx.doi.org/10.6028/NIST.IR.7874

[26] F. Martinelli, I. Matteucci, P. Mori, and A. Saracino, "Enforcement of u-xacml history-based usage control policy," in *Security and Trust Management*, G. Barthe, E. Markatos, and P. Samarati, Eds. Cham: Springer International Publishing, 2016, pp. 64–81. [Online]. Available: https://doi.org/10.1007/978-3-319-46598-2_5

[27] V. Gkioulos and Rizos, "Enhancing usage control for performance: An architecture for systems of systems," pp. 69–84, 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-12786-2_5

[28] B. M. A. Quimatio and F. Tsognong, "Horbac optimization based on suspicious behavior detection using information theory," *SN Comput. Sci.*, vol. 2, no. 2, p. 121, 2021. [Online]. Available: https://doi.org/10.1007/s42979-021-00515-w

[29] B. M. Azanguezet Quimatio, F. Tsognong, and M. J. Nkenlifack, "A correct-by-construction model for request-based access control," in *Soft Computing and Its Engineering Applications*, K. K. Patel, K. Santosh, G. Gomes de Oliveira, A. Patel, and A. Ghosh, Eds. Cham: Springer Nature Switzerland, 2025, pp. 44–57. [Online]. Available: https://doi.org/10.1007/978-3-031-88039-1_4

[30] K. V. Palavesam, M. V. Krishnamoorthy, and A. S M, "A comparative study of service mesh implementations in kubernetes for multi-cluster management," *Journal of Advances in Mathematics and Computer Science*, vol. 40, no. 1, pp. 1–16, Jan. 2025. [Online]. Available: https://journaljamcs.com/index.php/JAMCS/article/view/1958

[31] C. Jung and J. Dörr, *Data Usage Control*. Cham: Springer International Publishing, 2022, pp. 129–146. [Online]. Available: https://doi.org/10.1007/978-3-030-93975-5_8

Fidèle Tsognong is a dedicated researcher currently pursuing his Ph.D. in Computer Science, building upon his Master's degree from the University of Dschang, Cameroon. His research is deeply rooted in the critical domain of cybersecurity, with a specialized focus on the intricacies of Identity and Access Management (IAM), Security Information and Event Management (SIEM), and access control mechanisms. Driven by the rapid evolution of modern technological landscapes, his work emphasizes the security challenges inherent in Big Data analytics and cloud-native environments. He aims to develop robust and adaptable access control solutions that can effectively safeguard sensitive information in these dynamic and increasingly complex systems.

Dr. Benoit Martin Azanguezet received his Ph.D. in Computer Science from the University of Dschang, Cameroon, in 2016. He is currently a lecturer at the same institution. His research focuses on two key areas: information security and databases. In the field of information security, his work aims to enhance organizational security, with recent efforts concentrated on developing robust access control models. In databases, he specializes in both relational and NoSQL models. Dr. Azanguezet has contributed to many research papers, reflecting his extensive work in these domains.

Prof. Marcellin Julius Nkenlifack is the Head of the Department of Mathematics-Computer Science (Faculty of Science, University of Dschang). He is an expert for the ICT and Artificial Intelligence Commission of the National Technology Development Committee (CNDT). He is a representative of African researchers (Central Africa) on the CARI Permanent Committee (African Conference on Research in Computer Science and Applied Mathematics). He has been a visiting researcher at: the Galilee Institute –Paris13, SUPELEC –Rennes, Cheikh Anta Diop University –Dakar, Felix Houphouet Boigny University –Abidjan, CFI-CIRAD –Brazzaville. He is the author of about fifty articles in international journals. He has successfully led numerous research projects funded by various international organizations such as the "Silicon Valley Community Foundation". He has received several international awards for his scientific achievements.