Technologies Used in BIG-ABAC

sondes

February 2025

1 BIG-ABAC Implementation

BIG-ABAC integrates distributed technologies for real-time access control. The implementation consists of the following components:

a. Node.js for Simulating PIPs Node.js is used to generate real-time data streams from simulated IoT devices, such as patient vitals and ambulance status. These streams are published to Kafka topics, ensuring continuous contextual updates.

b. Apache Kafka 3.8.0 for Real-Time Data Streaming Kafka ensures real-time ingestion and distribution of contextual data and policy updates, providing high throughput and low latency. It enables event-driven decision-making and dynamic policy enforcement.

c. Kafka Streams for Context Handler Kafka Streams processes contextual updates, detects significant operational changes, and publishes alerts to the *Context Change* topic. This enables real-time adaptation of access control policies.

d. Load Balancing with Kafka and Ranger To prevent bottlenecks, access requests are distributed across multiple Policy Decision Point (PDP) instances.

- *Kafka topic partitioning*: The *Access_Request_Topic* is configured with multiple partitions to allow concurrent processing.
- *Kafka consumer groups*: Multiple PDP instances are grouped as consumers to ensure automatic request distribution.
- *Nginx API Gateway*: Routes incoming access requests to the available PDP instances.

e. Node.js and MongoDB v7.0.12 for PAP and Data Lake The Policy Administration Point (PAP) dynamically manages policies and updates, storing logs in MongoDB for traceability.

- *MongoDB Atlas*: Stores historical policy decisions and access logs for auditing and compliance.
- *Node.js REST API*: Handles policy management requests and synchronizes updates with the Policy Change topic.

f. Distributed Caching for Ranger Policies To reduce redundant database queries and accelerate policy evaluations, frequently accessed policies are cached.

- *Apache Ranger in-memory caching*: PDP instances store frequently used policies locally to reduce database lookups.
- *Redis Cluster*: A distributed caching system ensures low-latency access to policy rules across multiple PDP nodes.

g. Apache Ranger 2.4.0 for PDP Apache Ranger serves as the Policy Decision Point (PDP), evaluating access requests based on real-time policy and context updates.

- *Kafka consumer integration*: Ranger PDP listens to *Policy Change* updates for dynamic policy adjustments.
- *REST API integration*: Policy updates are applied dynamically via HTTP PUT requests to the Ranger API.

h. Parallel Processing for Ranger PDP Requests To handle high-concurrency scenarios, Ranger PDP processes multiple access requests simultaneously.

- *Multi-threaded PDP processing*: Each PDP instance spawns multiple worker threads to handle concurrent access requests.
- *Kafka consumer multi-threading*: PDP instances utilize multi-threaded Kafka consumers to parallelize incoming request handling.

i. Ranger Solr 8.11.2 for Search and Auditing Solr indexes policy logs and access events, enabling fast retrieval and forensic analysis.

j. PostgreSQL 12.15 for Policy Storage PostgreSQL stores structured access control policies, ensuring transactional integrity and secure rule management.

k. Node.js for PEP The Policy Enforcement Point (PEP) processes access requests and forwards them to the PDP, ensuring immediate enforcement of decisions.

l. AnyLogic 8.8.3 for Simulation Modeling AnyLogic is used to model and analyze the scalability and real-time adaptability of BIG-ABAC in simulated emergency scenarios.

1.0.1 Error Handling and Exception Management

To ensure the resilience and fault tolerance of BIG-ABAC, structured error-handling mechanisms are integrated across all stages of policy updates, access evaluations, and enforcement processes. The Policy Administration Point (PAP) validates policy configurations before publishing updates to the "Policy Change" topic in Apache Kafka, preventing malformed policies from propagating. If an error occurs during policy retrieval or enforcement, the Policy Decision Point (PDP), implemented with Apache Ranger, enforces the last known valid policy while logging errors in MongoDB for forensic analysis.

Implementation Details

- **Apache Kafka & Ranger Logs**
  - **Policy Update Errors**: If Kafka fails to deliver policy updates, Apache Ranger falls back to the most recent successfully applied policy stored in PostgreSQL.
  - **Context Processing Errors**: Kafka's dead-letter queue (DLQ) captures malformed or unprocessable context updates, allowing administrators to inspect and resolve issues manually.
- **PEP Timeout Handling & Retries**
  - The Policy Enforcement Point (PEP), integrated with Node.js, implements timeout mechanisms for REST API requests sent to Apache Ranger. If a request times out, PEP retries the request before logging an exception.
  - Failed access requests are logged in MongoDB, ensuring all anomalies are tracked for auditing.
- **Health Monitoring & Self-Healing**
  - Kafka's built-in monitoring (Cruise Control) detects consumer failures, automatically redistributing load to healthy instances.
  - Prometheus & Grafana dashboards provide real-time monitoring of Kafka topics, policy update latencies, and access control decision latencies, triggering alerts for unusual patterns.

Version Control and Software Updates To maintain system integrity and ensure seamless policy updates, BIG-ABAC implements automated version control and software update mechanisms. Policy changes are dynamically applied through Apache Ranger without requiring system downtime, ensuring uninterrupted access control enforcement.

- **Policy Versioning and Rollback**
  - Each policy version is stored in MongoDB, enabling administrators to revert to previous configurations if inconsistencies or errors occur.
  - Policy versioning supports compliance tracking and detailed audit trails for forensic analysis.
- **Automated Deployment Strategies**
  - Containerized updates using Docker and Kubernetes ensure controlled deployment of the Policy Administration Point (PAP), Policy Decision Point (PDP), and Policy Enforcement Point (PEP) components.
  - Updates are staged in a test environment before being applied to production, minimizing disruptions.
- **Reliable Policy Propagation**
  - Kafka enforces message ordering and guarantees that only validated policies are applied.
  - Policy updates are distributed through the "Policy Change" topic, ensuring that all PDP instances receive consistent and synchronized updates.

2 Conclusion

By integrating automated version tracking, rollback capabilities, and seamless updates, BIG-ABAC ensures continuous policy enforcement without downtime, maintaining system stability in high-demand environments. The use of distributed caching, load balancing, and parallel processing optimizes performance, allowing the system to efficiently handle high-concurrency access requests with minimal latency.

These mechanisms collectively reinforce fault tolerance, compliance, and real-time adaptability, ensuring that access control decisions remain accurate and reliable even under failure conditions. The error-handling framework, including Kafka's dead-letter queue (DLQ) and Apache Ranger's failover mechanisms, prevents policy inconsistencies while providing auditability through MongoDB's structured logging.

With its scalable, event-driven architecture, BIG-ABAC dynamically processes contextual updates and policy modifications, making it well-suited for critical domains such as emergency healthcare, smart cities, and industrial IoT. By leveraging cutting-edge technologies, including Kafka, Apache Ranger, MongoDB, and Kubernetes, BIG-ABAC delivers a resilient, high-performance, and future-proof access control framework capable of adapting to evolving security and compliance requirements.