Grant number: 883286
Project duration: Sep 2020 – Aug 2022
Project Coordinator: Joe Gorman, SINTEF

Horizon 2020: Secure societies
SU-INFRA02-2019
Security for smart and safe cities, including for public spaces
Project Type: Innovation Action

http://www.impetus-project.eu

IMPETUS Project Deliverable: D4.1

Data analytics and ingestion-time access control initial report

Dissemination Status: Public

Editor: Claudio Agostino Ardagna

Authors: Marco Anisetti, Annalisa Appice, Claudio Agostino Ardagna, Alessandro Balestrucci, Nicola Bena, Chiara Braghin, Michelangelo Ceci, Ernesto Damiani, Marco De Monte, Nicola Di Mauro, Stefano Ferilli, Corrado Loglisci, Donato Malerba, Francesca Mazzia, Costantino Mele, Paolo Mignone, Andrea Rizzello Bortone (CINI)

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

Grant number: 883286
Project duration: Sep 2020 – Aug 2022
Project Coordinator: Joe Gorman, SINTEF

Horizon 2020: Secure societies
SU-INFRA02-2019
Security for smart and safe cities, including for public spaces
Project Type: Innovation Action

About IMPETUS

IMPETUS (Intelligent Management of Processes, Ethics and Technology for Urban Safety) is a Horizon 2020 Research and Innovation project that provides city authorities with new means to improve the security of public spaces in smart cities, and so help protect citizens. It delivers an advanced, technology-based solution that helps operational personnel, based on data gathered from multiple sources, to work closely with each other and with state-of-the art tools to detect threats and make well-informed decisions about how to deal with them.

IMPETUS provides a solution that brings together:

- *Technology*: leverage the power of Internet of Things, Artificial Intelligence and Big Data to provide powerful tools that help operational personnel manage physical and cyber security in smart cities.
- *Ethics*: Balance potentially conflicting needs to collect, transform and share large amounts of data with the imperative of ensuring protection of data privacy and respect for other ethical concerns - all in the context of ensuring benefits to society.
- *Processes*: Define the steps that operational personnel must take, and the assessments they need to make, for effective decision making and coordination - fully aligned with their individual context and the powerful support offered by the technology.

Technological results are complemented by a set of practitioner's guides providing guidelines, documentation and training materials in the areas of operations, ethical/legal issues and cybersecurity.

IMPETUS places great emphasis on taking full and proper account of ethical and legal issues. This is reflected in the way project work is carried out, the nature of the project's results and the restrictions imposed on their use, and the inclusion of external advisors on these issues in project management.

The cities of Oslo (Norway) and Padova (Italy) have been selected as the site of practical trials of the IMPETUS solution during the project lifetime, but the longer-term goal is to achieve adoption much more widely.

The work is carried out by a consortium of 17 partners from 11 different EU Member States and Associated Countries. It brings together 5 research institutions, 7 specialist industrial and SME companies, 3 NGOs and 2 local government authorities (the trial sites). The consortium is complemented by the Community of Safe and Secure Cities (COSSEC) – a group established by the project to provide feedback on the IMPETUS solution as it is being developed and tested.

The project started in September 2020 with a planned duration of 30 months.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

Grant number:	883286	Horizon 2020: Secure societies
Project duration:	Sep 2020 – Aug 2022	SU-INFRA02-2019
Project Coordinator:	Joe Gorman, SINTEF	Security for smart and safe cities, including for public spaces
		Project Type: Innovation Action

For more information

Project Coordinator:	Joe Gorman, SINTEF:	joe.gorman@sintef.no
Dissemination Manager:	Snježana Knezić, TIEMS:	snjezana.knezic@gmail.com

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

Executive Summary

Background and objectives: Big data techniques and solutions are pushing towards a data-driven ecosystem where accurate decisions are supported by enhanced analytics and data management. The complexity of the big data ecosystem is amplified by the complexity of the corresponding infrastructures/tools, on one side, and by the data themselves that are intrinsically heterogeneous, collected at high rates from different sources, and with different formats in dynamic and collaborative environments. This scenario introduces the need to rethink and redesign conventional data tools and methods used to capture, store, manage and analyze data, while reviving the conflict between the need of protecting and sharing data. So far, performance and scalability requirements have been mainly addressed, but no proper considerations have been given to the problem of balancing data access and data sharing taking into account the peculiarities of big data systems. Similarly, algorithms for anomaly detection, e.g., Growing Hierarchical Self-Organizing Map (GH-SOM), require multiple iterations over the input dataset, thus making it intractable on larger datasets. Furthermore, the GH-SOM algorithm is designed to handle datasets composed of numerical attributes only. This aspect represents an important limitation, as most modern real-world datasets are characterized by mixed attributes, numerical and categorical. In addition, most anomaly detection algorithms do not guarantee output that is interpretable for the user but merely provide feedback in the form of a Boolean response, i.e., normal or anomalous.

The central objective of Deliverable D4.1 is to provide a big data solution in the smart city domain, supporting the definition and execution of big data analytics for anomaly detection and event classification. To this aim, this deliverable presents the data engine and the data management approach at the basis of these analytics, posing particular attention on data access and protection. It also defines novel analytics algorithms for anomaly detection and event classification.

Approach: The deliverable incrementally presents the proposed solution, which is an extension of the traditional Apache-based big data engine. The big data engine is enriched with an ingestion-based access control system that enforces non-functional requirements (e.g., privacy) on data collected at the partner cities. The data protection approach is driven by an extensive investigation of the key security requirements for big data governance. It is based on data annotations and secure data transformations performed at ingestion time, that is, it is platform-independent and could potentially be implemented in any big data environments. The big data engine is finally complemented with new algorithms for the construction of anomaly detection and event classification models, which guarantees large-scale efficient and distributed computation on time series data.

Results and conclusions: The big data solution introduced in this deliverable is an improvement with respect to the current state-of-art in the context of data preparation and analysis, as well as privacy protection. The data governance methodology in this deliverable introduces a common and shared access control layer, whereas in existing ad hoc solutions each service is monitored for data governance compliance. The proposed approach supports a city-level data governance strategy that can be applied to all services independently or to the whole smart city ecosystem by simply changing or adding new policies, while in case of an ad hoc solution every single service should be updated according to the new data governance strategy. Likewise, the data analytics methodology introduces the possibility of handling datasets composed of mixed attributes, making it ideal for many real-world issues, including the financial and cyber-security domains. Moreover, the proposed approach supports interpretability of the output for the end-user, by introducing a feature ranking, to better understand whether the identified anomaly represents a threat or not

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

Table of Contents

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

Table of Figures

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

List of Tables

List of Abbreviations

Table 1: List of Abbreviations

Abbreviation	Explanation
ABAC	Attribute Based Access Control
ANN	Artificial Neural Network
API	Application Programming Interface
AUC	Area Under the Curve
BMU	Best Matching Unit
DAG	Directed Acyclic Graph
ELT	Extract, Load, Transform
ETL	Extract, Transform, Load
GDPR	General Data Protection Regulation
GH-SOM	Growing Hierarchical Self-Organizing Map
HDFS	Hadoop Distributed File System
HiveQL	Hive Query Language
HQL	Hibernate Query Language
JDBC	Java™ database connectivity
LSH	Locality Sensitive Hashing
MQE	Mean Quantization Error
NILU	Norwegian Institute for Air Research
NoSQL	Not only SQL
ODBC	Open Database Connectivity
OLAP	Online Analytical Processing
PKI	Public Key Infrastructure
RDBMS	Relational Database Management System
RDD	Resilient Distributed Datasets
REST	Representational state transfer
SOM	Self-Organizing Map
SQL	Structured Query Language
UI	User Interface
URL	Uniform Service Locators
XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language
YARN	Hadoop Yet Another Resource Negotiator

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

List of Definitions

Table 2: List of Definitions

Term	**Definition/explanation**
Distributed system	A software system distributed over logically separated networking nodes with the main goal to distribute data and processing in a reliable way.
Architecture	A set of software components organized in a logical view to emphasize their dependencies.
Pipeline	A sequence of tasks to achieve a specific goal. Each task can be parallelized on multiple executors improving the performance. Pipelines are suitable jobs for processing-oriented distributed systems.
Software ecosystem	A set of tools, normally loosely coupled, designed to work in synergy. They are integrated in a software architecture by means of specific configurations.
Data Lake	An ecosystem of data management tools (software components) to handle data peculiarities like the absence of a predefined structure. Every tool involved in the data lake should have some degree of distribution to fit the need of parallelism in computing pipelines.
Data sanitization	Data sanitization refers to a plethora of solutions and processes aimed to protect sensitive data in a document, message, database. It includes anonymization, generalization, suppression, masking, to name but a few.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

1 About this deliverable

1.1 Why would I want to read this deliverable?

The main objective of deliverable D4.1 is to describe the approaches and methodologies at the basis of big data analytics for anomaly detection and event classification.

The ability to analyse large volumes of data is important to increase the safety and security of smart cities, boosting the adoption of smart processes even in critical domains such as healthcare and transportation. This deliverable provides an overview of big data approaches and methodologies at three layers:

i) infrastructure layer, presenting the data engine that supports big data analytics;
ii) data management layer, presenting how access to data is monitored and enforced at ingestion time;
iii) data analytics layer, presenting the algorithms for anomaly detection and event classification.

D4.1 presents the benefits a solution for big data analysis can bring to a scenario related to public space safety and security. The deliverable presents an end-to-end journey through a big data ecosystem focusing on all aspects at the basis of a proper big data analysis process in the smart city domain. It poses particular attention to the components for data preparation and analysis, as well as privacy protection.

When considering data management, it is important to recall that the conflict between the need of protecting and sharing data is hampering the spread of big data applications. Security and privacy assurance is required to protect data owners, while data access and sharing are fundamental to implement smart big data solutions. In this context, access control systems can assume a central role in balancing data protection and data sharing. However, existing solutions are not general and scalable enough to address the software and technological complexity of big data ecosystems, being unable to support such a dynamic and collaborative environment. D4.1 proposes an organic and coherent solution where traditional data management components are enriched with an ingestion-based access control system that enforces access to data in a distributed, multi-party big data environment. The approach presented in this deliverable is based on data annotations and secure data transformations performed at ingestion time, and implemented in a smart city domain using a traditional Apache-based big data engine.

When considering data analysis, it is important to recall that the research and development community is making a quantum leap towards increasingly accurate and precise algorithms. To keep pace with the evolving and pervasive environment surrounding EU citizens, it is necessary to design and implement new algorithms especially in the context of anomaly detection and event classification. Being able to pre-emptively identify anomalies and recognize specific events is fundamental to increase the safety of our cities. An increasing request for complex event classification and for smart and adaptive anomaly detection is therefore raising. This deliverable addresses this need by presenting novel algorithms for anomaly detection and event classification.

1.2 Intended readership/users

The primary audience of the deliverable is practicians, specialist, researchers, policy officers, safety and security operators with interest in big data analysis and big data management.

The audience can then be split into:

- Internal audience: all the Consortium Partners. In particular, partners in charge of platform and tools development can be interested in the architectural approach at the basis of the data management and analytics solutions; partners responsible for the specification of the privacy-preserving mechanisms can be interested in the data management approach with particular reference to the ingestion-based access control mechanism and its data transformations; partners involved in the ethics framework can be interested in the data management approach.
- External audience: anyone interested in the adoption of the project results and, in particular, in the data management approach and the data analytics algorithms for anomaly detection and event classification. More in details, external audience can include other R&D projects, system developers and engineers, data

scientist interested in the technical aspects of data management, access control, and data analytics. In addition, EU stakeholders (e.g., EU officers and policy makers) can be interested in the data management approach discussed in this deliverable. Finally, even SMEs involved in the domain of data management, data protection and data analytics, can be interested in this deliverable.

The deliverable (or parts of it) may also be of interest to a wider group: anyone with an interest in data management and transformation to support collection, analysis and manipulation of data, and in data analysis based on anomaly detection and event classification.

1.3 Structure

The document is organized in three main chapters, as follows.

Chapter 2 describes the Apache-based big data engine supporting big data analytics, on one side, and access control and ingestion functionalities, on the other side. This chapter first describes a conceptual view of the engine (Section 2.1). It then discusses an architectural view of the engine (Section 2.2), including data and resource management, Access and Audit, and Processing layers. Section 2.3 presents the architectural data flow.

Chapter 3 focuses on data ingestion and ingestion-time access control. It first discusses the approach to data ingestion of structured and unstructured data (Section 3.1). It then presents data annotations for data governance (Section 3.2) and data transformations for privacy protection (Section 3.3). It finally presents an attribute-based access control methodology that is applied at ingestion time (Section 3.4), with a discussion on how it is mapped on the data engine (Section 3.5) and a walkthrough scenario (Section 3.6).

Chapter 0 presents the data analytics algorithms. It first describes solutions for anomaly detection (Section 4.1). It then presents solutions for event classification (Section 4.2).

Chapter 5 finally draws our conclusions and presents next steps that will be reported in D4.2 "Data analytics and ingestion-time access control final report".

1.4 Other deliverables that may be of interest

D4.1 is in relation with the following deliverables:

- D1.2. "Requirements for public safety solutions", providing requirements for data analytics and ingestion-time access control.
- D3.1 "Secure Smart City Tool Development initial report", providing a description of the functionalities of the algorithms of the Physical Threat Intelligence tool that are used as a starting point for the definition of enhanced big data analytics in this deliverable.
- D4.2 "Data analytics and ingestion-time access control final report" will extend the discussion and approach presented in this deliverable.
- D5.2 "Initial mechanisms to preserve privacy in the secure smart city", describing privacy-enhancing technologies that can be adopted in the data transformation process at the basis of the ingestion-time access control solution in this deliverable.

D4.1 is also indirectly related to D7.1 "Validation Plan", presenting the process for validating the solutions provided by IMPETUS platform and D6.1 "Initial Concepts of Operation", developing strategies and operating guidelines, to support organizations in integrating the platform into the management of security events.

1.5 Synergy with other projects/initiatives

No synergy.

2 Data engine

This chapter presents an overview of the architecture of the big data engine supporting i) data ingestion (Section 3.1), ii) data sanitization and access management (Section 3.2, 3.3 and 3.4), and iii) data processing (Chapter 4) on data coming from the city pilots. In the following, we first present the conceptual view of our engine for non-expert readers (Section 2.1); we then present the architectural view with technical details on the working of the big data engine (Section 2.2); we finally present the architectural data flow (Section 2.3).

2.1 Conceptual view

Figure 1 shows a conceptual and abstract view of our big data engine architecture. It implements a *data management process*, that is, the process of collecting, storing, using, and maintaining data in a secure, efficient, and cost-effective way. The process includes all practices devoted to the protection of data security and privacy, as well as all activities needed to prepare data for a proper analytics process. It is composed of 5 main building blocks as follows.

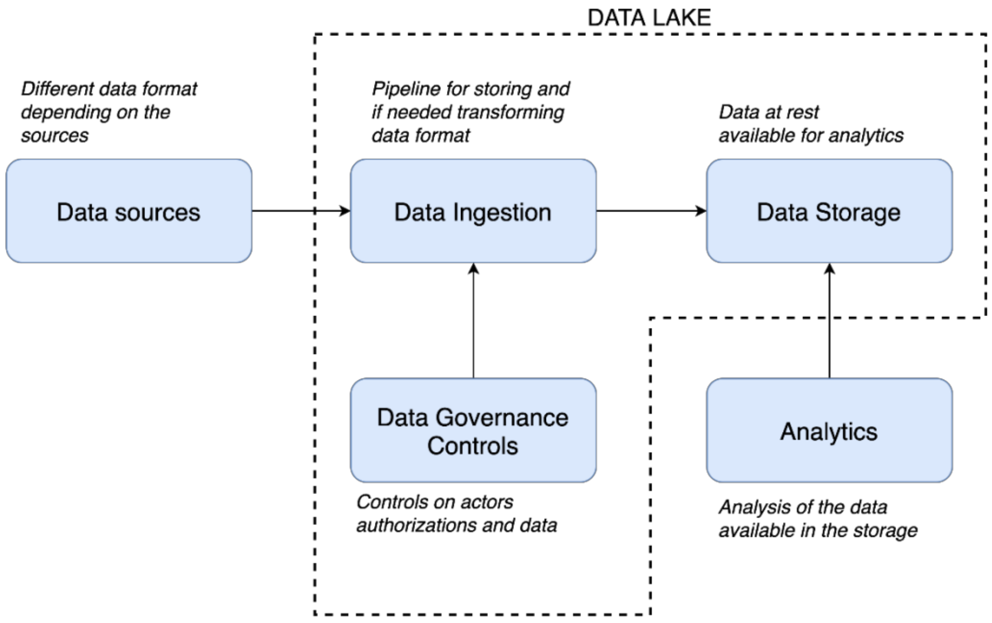

DATA LAKE
Different data format depending on the sources
Data sources
Pipeline for storing and if needed transforming data format
Data Ingestion
Data at rest available for analytics
Data Storage
Data Governance Controls
Controls on actors authorizations and data
Analytics
Analysis of the data available in the storage

Figure 1: Conceptual view of the big data engine.

- *Data sources:* Physical data collection points, where relevant data are collected.
- *Data ingestion:* The procedure that stores data within the data storage. Depending on data sources and the format of the collected data, data ingestion procedure may require a data format transformation that fits the data storage target of the ingestion process. Given the nature of the data to be ingested, it is normally implemented as a pipeline where i) the initial task collects data from the data sources, ii) the following tasks transform data, if needed, iii) the last task saves data to the landing data storage.
- *Data storage:* The component responsible for storing data. It is part of the data lake and preserves confidentiality, integrity, and availability of data at rest. It provides data for analytics.
- *Data governance controls:* The set of controls (e.g., processes, policies, standards, metrics) supporting an efficient and effective access to and usage of data. It is part of the data lake and preserves confidentiality, integrity, and availability of data at rest. It provides access to data for analytics.
- *Analytics:* A pipeline that implements a specific analysis on the data in the data storage having the scope to extract value. Analytics can include statistical analysis, data mining analysis or machine learning analysis, to name but a few.

We note that the core of the view in Figure 1, that is, data ingestion, data storage, data governance controls, implements the data lake, an ecosystem of data management tools and components that supports the management of the entire data life cycle from collection to analysis. We also note that, given the nature of the data to be ingested (possibly confidential data) and the heterogeneity of data sources, data ingestion is augmented with data governance controls on actors (services or users triggering data ingestion for analytics) and data. The latter can require additional control-oriented transformation tasks, which are implemented using the advanced ingestion-time access control presented in Chapter 3.

The *data management process* described in Figure 1 is composed of the following steps: i) data are collected from a variety of data sources, each of which with a possible different format; ii) data are forwarded to the ingestion pipeline that specifies the activities needed to transform and reconcile the different data formats; iii) data ingestion is subject to data governance, where ad hoc controls are implemented to monitor actors' activities and data exchanges and operations; iv) ingested data are stored in a data store for further analytics activities; v) analytics are performed on stored data and produce results that can be stored in the data storage as well. The process is implemented in a parallel and distributed system built on top of the big data engine described in the following of this section. Technical details on the entire data governance process and the analytics are available in Chapter 3 and Chapter 0, respectively.

2.2 Architectural view

Figure 2 shows the architecture of our Apache-based big data engine.[1] It is composed of three main types of components as follows. *Data and Resource Management* components focused on data collection and transformation prior to their storage for analytics. They are at the basis of data ingestion. *Access Control and Audit* components focused on data sanitization and access management. *Processing* components focused on the analytics of data collected at ingestion time. All components are presented in detail in Sections 2.2.1, 2.2.2, 2.2.3.

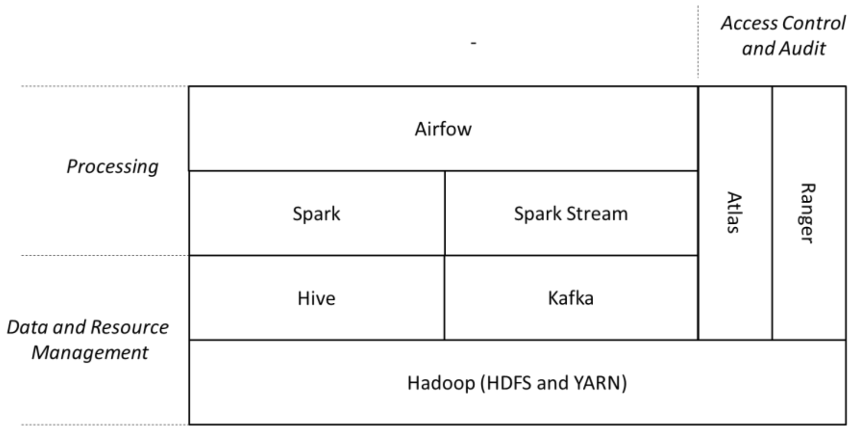


Figure 2: Big data engine architecture.

2.2.1 Data and resource management

Data and resource management are fundamental functionalities for any kind of distributed systems, including big data engines. In the following, we describe the components managing data storage (Hadoop Distributed File System - HDFS and Hive), ingestion, buffer storage and messaging (Kafka) and computational resources (Hadoop Yet Another Resource Negotiator - YARN).

[1] The big data Apache ecosystem offers some of the most prominent and used solutions in big data industry. Success is due both to the open source nature of all the tools, which can run on commodity hardware in existing data centers or on cloud infrastructures, and to their efficiency.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

2.2.1.1 HDFS and YARN

The Hadoop framework is designed to store and process large amount of data on clusters made of commodity hardware. It constitutes the lowest layer of our architecture, and provides storage (i.e., HDFS) and resource management (i.e., YARN) capabilities. It also provides the Map-Reduce processing model suitable for processing analytics that does not suffer from data intensive activities. It is based on work done by Google in the early 2000s [1][2]; the core idea is to distribute the data as they are initially stored, lowering the need to move data between nodes for the initial processing. Each node can then perform computations on the data it stores, thus reducing the need of communications (i.e., "shared nothing" paradigm where data are spread among nodes in advance). When data are loaded in the system, they are divided into blocks (64MB or 128MB) to ease the distribution of data and the computation of differences. Processing tasks are divided in two phases i) map tasks that insist on small portions of data and are executed where data are stored, ii) reduce tasks that combine data from different nodes to produce the final results. A master node allocates tasks to each processing node. Master node is also responsible for failure detection and reassigns tasks to a different node. It can also redundantly execute the same task to avoid restart or re-allocation.

Hadoop consists of: i) Hadoop Common that contains libraries and modules, ii) HDFS that contains all the components of the Hadoop distributed file system, iii) YARN that includes the components for resource allocation and negotiation, and iv) Hadoop MapReduce that provides a programming model for large scale data processing and components supporting it.

HDFS

HDFS is a distributed file system, written in Java, based on blocks to be distributed across nodes. Each block is replicated multiple times in different nodes to guarantee resilience. HDFS adopts an approach where files are written once and read many times; it is optimized for streaming reads.

More in detail, files, when loaded into HDFS, are split into blocks that are then distributed across nodes at load time. Different blocks from the same file are stored on different nodes and replicated for a given number of times. The master node called "NameNode" keeps track of the blocks–file relation (see Figure 3).

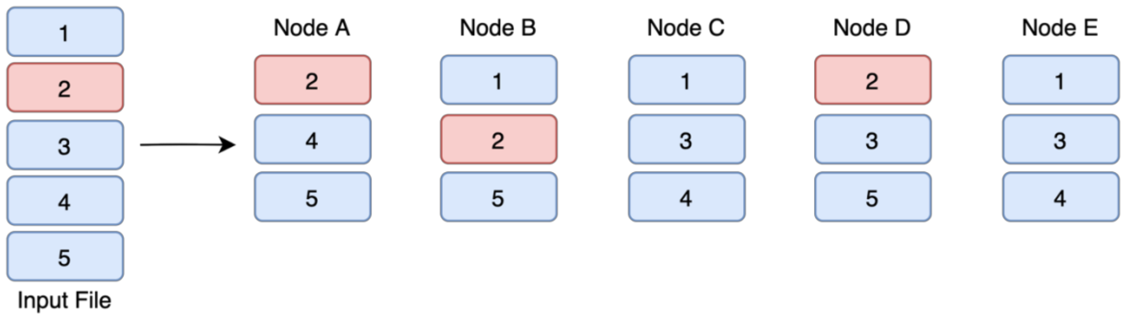


Figure 3: File distribution.

NameNode is also used to retrieve data, determining which blocks make up a file and on which data nodes those blocks are stored. Figure 4 shows a more detailed view on HDFS architecture.

- **Client:** It writes or reads files on HDFS.
- **NameNode:** It holds the metadata for HDFS. Stores the HDFS data blocks in FsImage file. Any updates to the file system (add/remove blocks) are marked on EditLog, without the need to change FsImage file. The modifications are triggered at checkpoint time by the Secondary NameNode.
- **Secondary NameNode:** It performs housekeeping functions for the NameNode. Periodically reads the EditLog and applies the changes to the FsImage file bringing it up to date. It supports fast restart of NameNode if needed.
- **DataNode:** It stores the HDFS data blocks.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

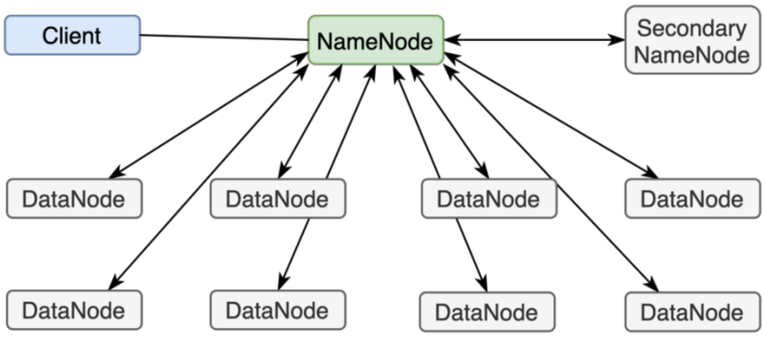


Figure 4: HDFS Architecture.

Hadoop MapReduce

Hadoop MapReduce component provides processing capabilities to Hadoop framework executing MapReduce tasks on the cluster of HDFS nodes. Figure 5 shows the architecture of MapReduce applied to Hadoop HDFS cluster.

- **JobTracker:** The master responsible to manage MapReduce jobs. It determines the execution plan for the job and assigns individual tasks.
- **TaskTracker:** The slave that performs the job. Monitors individual Map and Reduce tasks and keeps track of the performance of an individual mapper or reducer.
- **Map tasks:** Tasks that execute split and map.
- **Reduce tasks:** Tasks that execute shuffle and reduce.

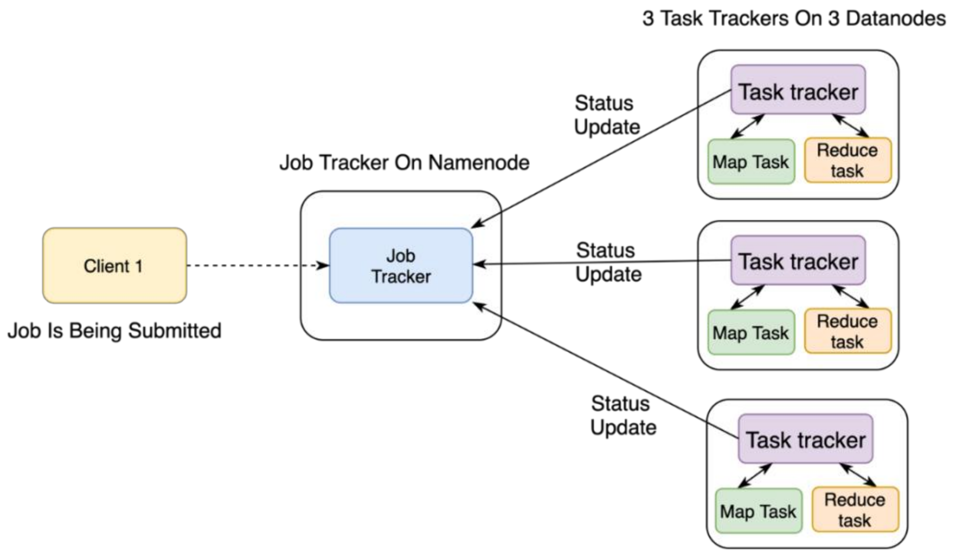


Figure 5: MapReduce architecture applied to HDFS cluster.

For every job submitted for execution in the system, there is one Job Tracker that resides on Namenode and there are multiple Task Trackers which reside on Datanode.

Task tracker is responsible to send the progress report to the Job Tracker and periodically send 'heartbeat' signals. In the event of a task failure, the job tracker can reschedule it on a different Task Tracker.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

Hadoop YARN

Hadoop YARN is the resource manager of the Hadoop cluster. It was introduced in Hadoop v2 as a generic platform to run any distributed applications (including MapReduce ones). YARN splits the functionalities of resource management and job scheduling/monitoring into separate components: i) Resource Manager focused on global resources management and ii) Application Master focused on single application (single job or workflow of jobs).

Figure 6 shows the YARN architecture. Resource manager is made of two subcomponents: Scheduler and Application Manager. The Scheduler allocates resources for applications. The Applications Manager manages job submission negotiations for the first container where the Application Master is executed.

The Application Master negotiates resources with the Scheduler for the application. It also tracks the status and monitor progresses.

Another component is the Node Manager, which launches and manages containers on a node. Containers execute tasks as specified by the Application Master.

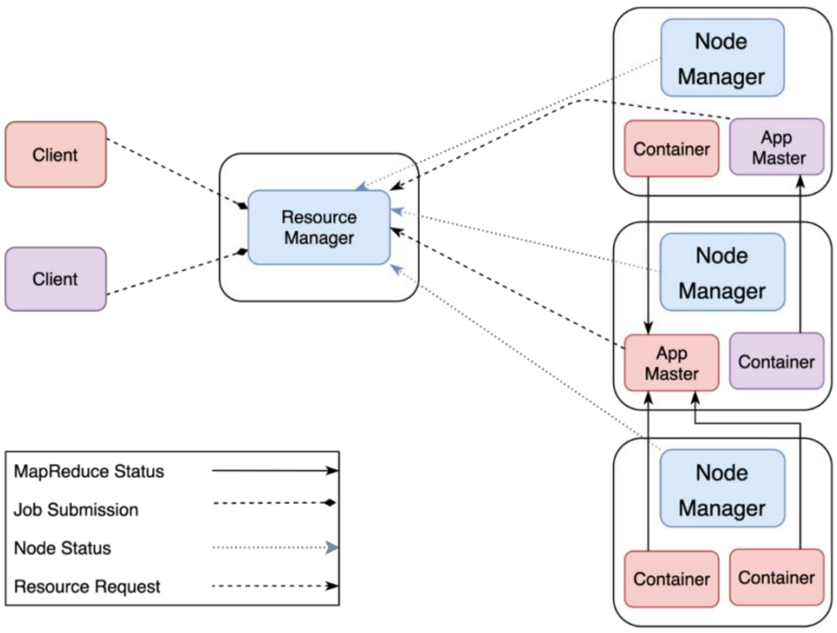


Figure 6: YARN architecture.

The steps to submit an application to a YARN cluster are discussed in the following.

1. A client submits an application to the YARN Resource Manager.
2. The Application Manager negotiates a container and bootstraps the Application Master instance for the application.
3. The Application Master registers with the Resource Manager and requests containers via Scheduler (RAMs and CPUs).
4. The Application Master communicates with Node Managers to launch the containers it has been granted with.
5. The Application Master manages application execution.
6. The Application Master reports completion of the application to the Resource Manager.
7. The Application Master unregisters with the Resource Manager, which then cleans up the Application Master container.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

2.2.1.2 Kafka

Kafka is the component responsible for stream ingestion. It is also integrated with Apache Atlas (see Section 2.2.2.2) to support its communications with components that perform data lineage and tag-related notification. Kafka is a distributed streaming platform that implements a producer/consumer paradigm with i) high scalability via partitions, ii) fault tolerance via replication and iii) a high level of parallelism and decoupling between data producers and data consumers.

Kafka uses a topic as a user-defined category with which messages are published. Producers publish messages to one or more topics, while consumers subscribe to topics and process the published messages. Kafka cluster consists of one or more brokers (servers) that manage the persistence (with specific retention periods, if needed) and replication of messages. Each broker has a unique ID and can be responsible for partitions of one or more topic logs.

Kafka brokers leverage on ZooKeeper to i) manage and coordinate the cluster, ii) elect a broker to deal with client requests for an individual partition of a topic.

Topics may have multiple writers and readers. Producers can also add a key to a message to let them be placed to the same partition. Without keys, messages are written to partitions in a round robin fashion.

Figure 7 shows the simplified architecture with Kafka cluster and ZooKeeper as an external service.

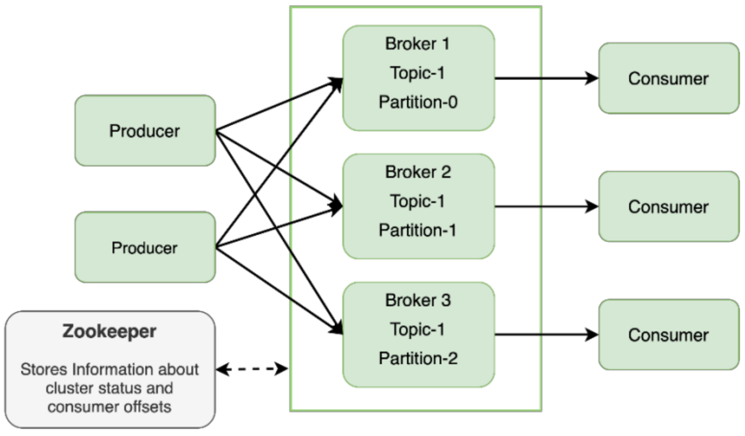

Producer
Producer
Zookeeper
Stores Information about cluster status and consumer offsets
Broker 1 Topic-1 Partition-0
Broker 2 Topic-1 Partition-1
Broker 3 Topic-1 Partition-2
Consumer
Consumer
Consumer

Figure 7: Architecture with Kafka cluster and ZooKeeper.

At a more abstract level, Kafka topics can be seen as channels through which data are streamed. Topics organize and structure messages and are identified by unique names within a Kafka cluster. In Kafka clusters, topics are divided into partitions, and partitions are replicated across brokers.

2.2.1.3 Hive

Hive is a structured data warehouse based on Hadoop, capable to manage large data sets that reside in a distributed storage. It uses a SQL-like language called HiveQL and leverages on storage and resource management of Hadoop. A Hive query is first converted into Map Reduce and then processed by Hadoop or Spark. Hive was developed by Facebook to reduce the work of MapReduce programs writing data. Figure 8 shows the Hive architecture. Hive supports thrift client, JDBC client and ODBC client. Hive services are:

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

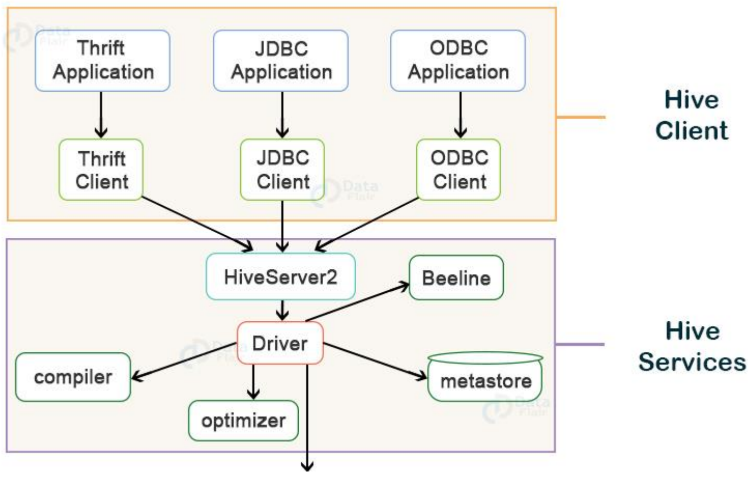


Figure 8: Hive architecture.

- **HiveServer2**: The second version of HiveServer service that supports more clients. It enables clients to execute queries on hive.
- **Driver:** It receives the HiveQL statements to execute them by creating sessions and monitoring the progress. It stores the metadata of HiveQL execution on the Metastore. The driver is also the collection point for data or query results retrieved after the reduce operation of the underline MapReduce engine.
- **Metastore:** It stores the metadata information about i) the structure of tables (e.g., column name and type), ii) partition metadata for the driver to track distributed data sets over the cluster.
- **Compiler:** It performs compilation of the HiveQL query, to an execution plan in terms of tasks and steps to be executed by Hadoop MapReduce. The compiler converts the query to an abstract syntax tree to check for compile-time errors and then to a Directed Acyclic Graph (DAG) that can be mapped on MapReduce stages and tasks.
- **Optimizer:** It works on the DAG to optimize performance and scalability.
- **Beeline:** A CLI used to submit queries and commands.

After compilation and optimization, the Hive executor executes the tasks interacting with the JobTracker of Hadoop. It checks for the pipelining of the tasks that handles dependency and tasks ordering. Currently Hive also permits to interact with Apache Tez, an executor engine that is an additional layer on top of YARN.

2.2.2 Access control and audit

Access Control and Audit are key functionalities for a big data engine. In the following, we describe components that handle authorization/access to data and access audit (Ranger) and support data annotation for data governance and audit (Atlas).

2.2.2.1 Apache Ranger

Apache Ranger offers centralized authorization and auditing across Hadoop components via specific plugins. It permits to design attribute-based access policies using resources classification or tags, and to specify data transformations. It is part of the access management and audit layer. It also offers audit capabilities of policies outcomes. Figure 9 shows the Apache Ranger Architecture. It also describes external but crucial components that interacts with Ranger, like Solr as audit searching engine, HDFS for hosting audit logs, Atlas for tags, LDAP/AD/OS for authentication. We note that HDFS and Solr are also mentioned in the architecture since they are involved in Ranger activities via specific plugins.

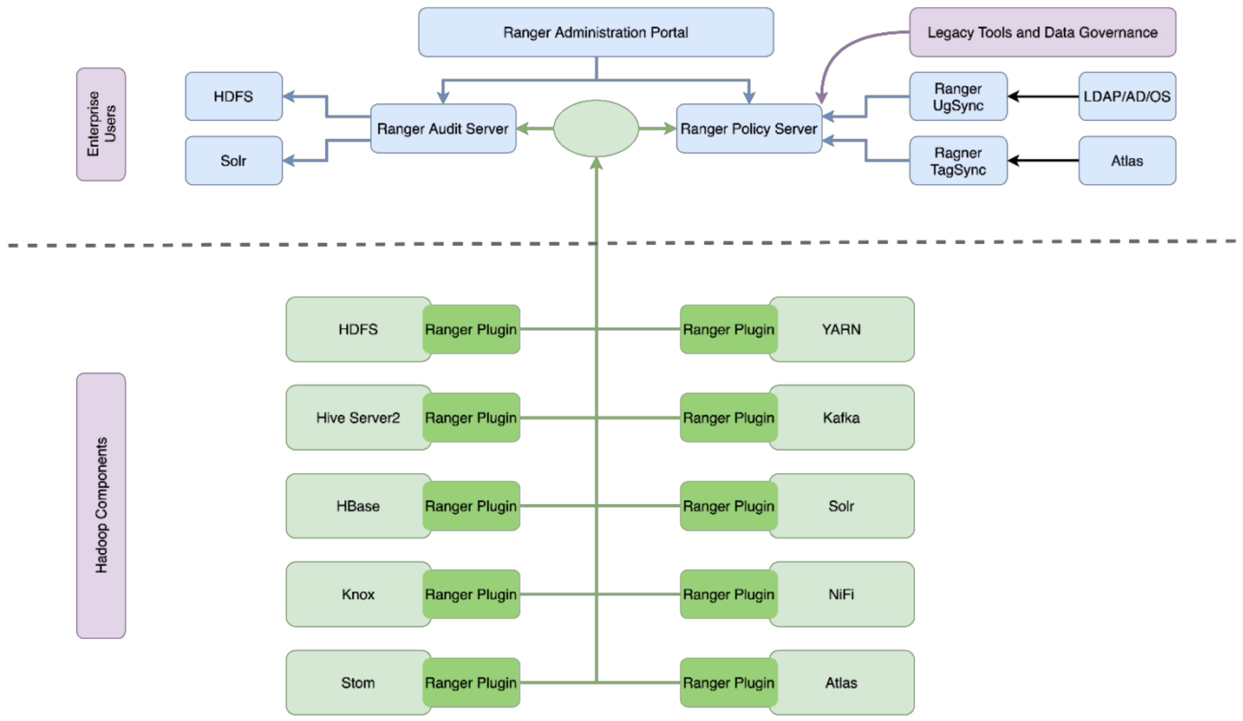


Figure 9: Apache Ranger Architecture and external components.

Ranger is composed of a set of services including i) Admin Portal, ii) Policy Server, iii) User/Group Sync Service, iv) Audit Service and v) Tag Sync Service.

- **The Admin Portal** is the central interface for the administration of Ranger. It provides UI for policy, audit, and the API module for interoperability.
- **The Policy Server** allows admin users to define/update policy details. It supports both allow and deny policies that can be associated with different security zones with distinct sets of resources. Security zones separate resource policies into different zones, to simplify administration of security policies and limit the policy checks during authorization against certain resources. Zones enable multiple zone administrators to setup different policies. One resource can only be assigned to one zone.
- **User/Group Sync** is a synchronization utility used to pull users and groups from different sources to be stored in Ranger policy DB for policy definition.
- **The Audit Service** permits to audit policy results. It is configured to enable audit on specific policies. Audit logs are stored to HDFS by default.
- **The Tag Sync Service** is used to separate resource-classification from access-authorization. It uses Atlas to manage metadata. It is driven by events handled by Atlas. Tag-based policies are evaluated before the resource-based ones.

Ranger interacts with other Hadoop components via plugins. The Ranger plugin loads policies from Admin Server regularly and cache them locally. It acts as the authorization module for the component that evaluates user requests against security policies. It is also involved in audit event triggering.

The plugin-based approach makes the architecture extensible. It supports authorization and auditing for new components.

2.2.2.2 Apache Atlas

Apache Atlas provides a metadata repository with a flexible type system to capture schema and metadata of multiple components (e.g., Hive and HDFS). It supports the data annotation process, tag policy specification in Ranger, and data lineage and provenance. It also provides metadata search functionalities based on attributes. In terms of capabilities, it provides: i) data classification, ii) centralized auditing, iii) search and lineage, iv) security and policy engine. To support data classification, it provides taxonomy and data annotations; it also automates the capture of relationships between data sets, sources, targets, and derivation processes. Atlas offers centralized auditing to capture security access information for every process and data interaction. In terms of search and lineage capabilities, Atlas permits to explore the data classification and audit information, search for features, locate relevant data and audit events, visualize data set lineage and operational security, and provenance-related information.

Atlas implements the layered architecture described in Figure 10.

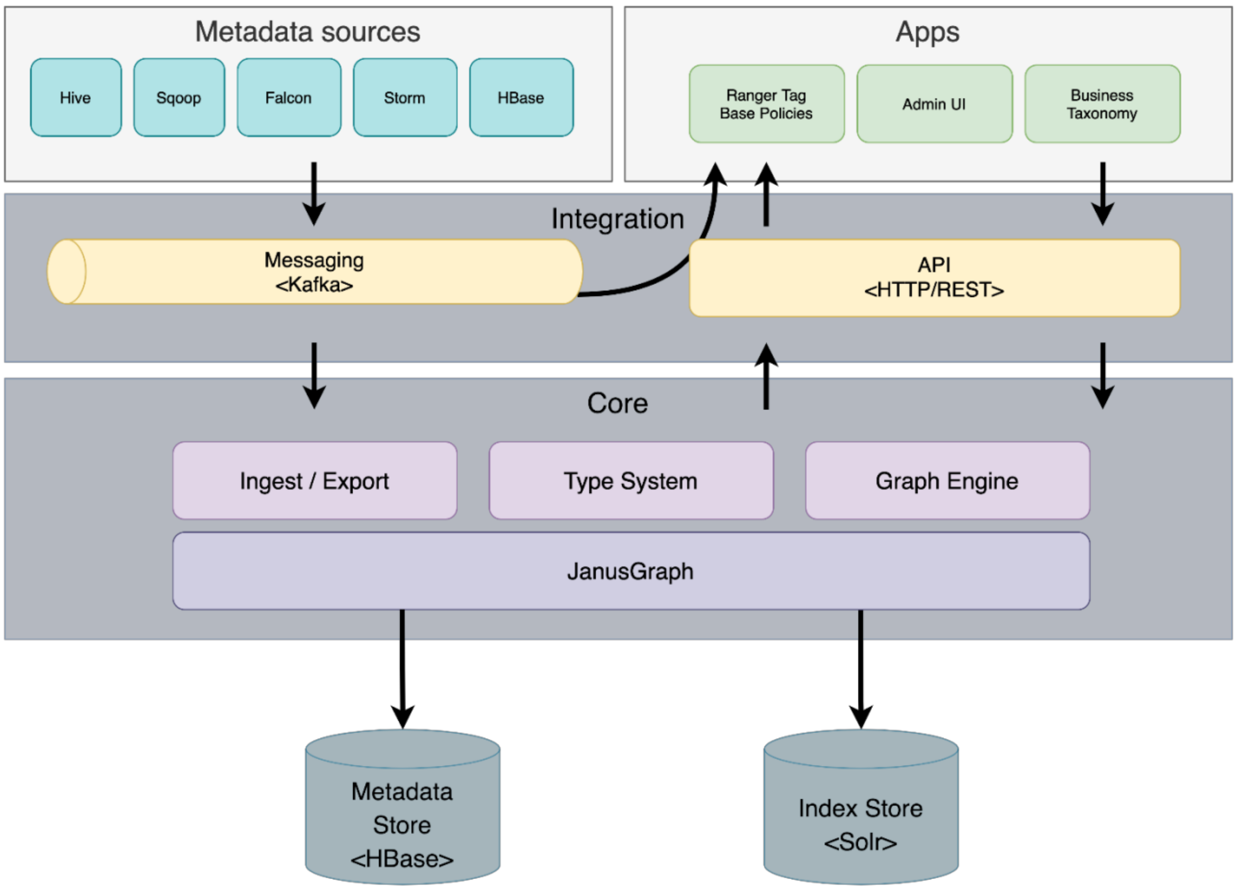


Figure 10: Architecture of Apache Atlas.

The core layer includes: i) the type system, ii) the ingest/export component and iii) the graph engine. The type system permits to define models for the metadata objects in terms of definitions (called types) and instances of types (called entities). It permits to maps relations between entities. The Graph engine is the engine used to manage metadata that can be added and exported via ingest/export services. Atlas interacts with the rest of the ecosystem in two ways through the integration layer: i) REST API as the primary mechanism to create delete and update entities, and to query and discover types and entities, ii) Kafka messaging to communicate metadata and events with other data-oriented Atlas supported services like Hive, HDFS, and Spark in our architecture.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

The Application layer of Atlas includes: i) A web admin UI interface, ii) Ranger tag-based policies, which consume metadata as tags to verify its authorization policies.

2.2.3 Processing

Parallel distributed processing of data is one of the distinguishing factors underpinning the big data engine adoption. We describe the components that support in memory processing of batch data (Spark), processing of streams/micro-batch (Spark stream), and orchestration/scheduling of processing pipelines (Airflow).

2.2.3.1 Apache Spark

Apache Spark is an advanced, unified analytics engine for large-scale data processing. It is capable to leverage on Hadoop, though compared to the Hadoop MapReduce it i) is focused on in-memory processing of cached data, ii) offers a more expressive computing model not limited to Map and Reduce instructions, and iii) achieves fault tolerance via re-execution and lineage instead of replication. It also provides the support of tools like Spark SQL and Mlib.

In addition, it supports more programming languages than Hadoop MapReduce.

The core idea of Spark is to provide a more expressive computing system (not limited to map reduce model). It exploits system memory to avoid saving intermediate results to disk and cache data for repetitive queries. It also achieves fault-tolerance by re-execution instead of replication. Figure 11 shows the Spark architecture, whose components work as follows.

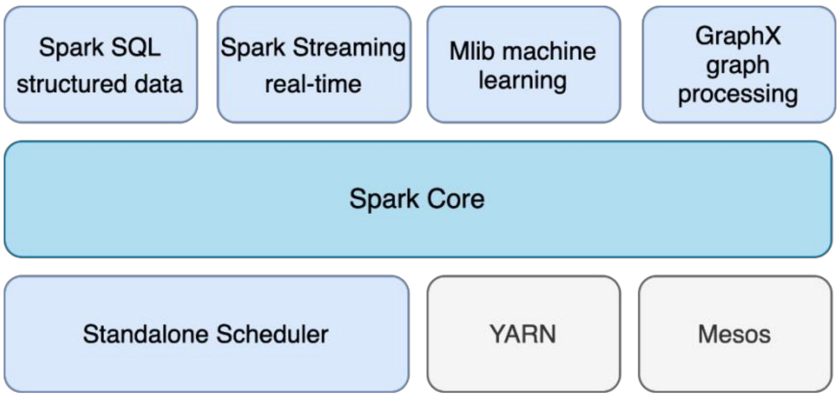


Figure 11: Apache Spark architecture.

- **Spark Core:** It provides an execution platform for all the Spark applications. It is responsible for fault recovery, scheduling, distribution and monitoring jobs, memory management and interaction with storage systems.
- **Spark SQL:** It enables users to run SQL/HQL queries. Spark SQL permits to process structured and semi-structured data.
- **Spark Streaming:** It enables a powerful interactive and data analytics application. The live streams are converted into micro-batches that are executed on top of Spark core.
- **Spark Mllib:** It is the library containing efficient and high-quality Machine learning algorithms written in Spark.
- **Spark GraphX:** It is the graph computation engine built on top of Apache Spark that enables to process graph data at scale.

In the Spark Architecture, YARN and Mesos are mentioned as alternative means to allocate resources for the Spark computations. If undefined, Spark can use its own Spark cluster manager.

A Spark program (i.e., driver program) first creates a SparkContext object that i) tells Spark how and where to access a cluster, ii) connect to several types of cluster managers (e.g., YARN). Cluster manager allocates resources across applications (workers node). Each worker node contains a Spark executor that is responsible to run computations and access the data storage.

To provide in-memory processing, Spark uses a peculiar data abstraction called Resilient Distributed Datasets (RDDs). RDD is a Fault-Tolerant Abstraction for In-Memory Cluster Computing. It is a collection of partitioned elements (e.g., tuples) that represent records of the data. It is i) resilient in terms of fault-tolerance, since it is able to recompute missing or damaged data partition due to node failures, ii) distributed since data reside on multiple nodes in a cluster.

RDD is stored in RAM memory. They are immutable, no changes once created, they can only be transformed by means of transformations to new RDDs (functional programming paradigm). RDD are subjects to lazy evaluation. The data inside RDD are not available or transformed until an action triggers the execution. RDDs are typed values and can be stored and retrieved directly from the memory without the need to access the disk. The operation applied to RDD refers to the entire data set and not to just one element. In Spark, operations are of two types, as follows.

- **Transformation:** Functions that return a new RDD. No evaluation is done when a transformation is called. The transformation function just takes an RDD and returns one or more RDDs. Examples of functions are map, filter, flatMap, groupByKey, reduceByKey, aggregateByKey.
- **Action:** It evaluates and returns a new value. When an Action function is called on a RDD object, all the data processing queries are computed at that time and the result value is returned. Examples of actions are reduce, collect, count, first, take, countByKey, foreach.

Figure 12 shows Spark RDD and operations.

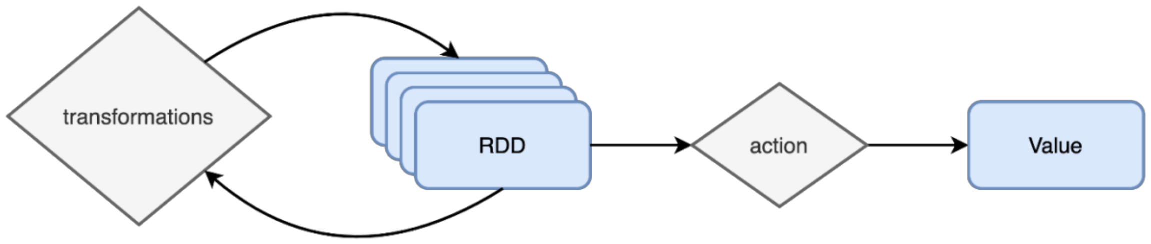


Figure 12: The operations in Spark RDD.

Spark transformations use lazy evaluation. No computation is performed but Spark remembers the set of transformations to be applied so that it can optimize the calculations and recover from failures. Spark implements two types of transformations: *narrow transformation*, where an output RDD has partitions with records that originate from a single partition in the parent RDD (e.g., Map, FlatMap, MapPartition, Filter), and *wide transformation* (i.e., shuffle), where output data of a single partition may originate from multiple partitions of the parent RDD (e.g., groupByKey, reduceByKey, Join, Intersect).

Spark actions are RDD operations that produce non-RDD values returning the final result of RDD computations. It triggers execution using lineage graph to load the data into original RDD, carries out all intermediate transformations and returns results to the driver program or writes it out to the file system. Examples of actions are: first, take, reduce, collect, count.

RDD and operations constitute the Spark DAG (Directed Acyclic Graph), where vertices represent the RDDs and edges represent the operations to be applied on RDD. On the calling of an action, the DAG is submitted to the DAG scheduler, which further splits the graph into the stages of the task. Stages are sequences of RDDs, which do not have a shuffle in between. For each stage, Spark creates a task for each partition in the new RDD, serializes the tasks, schedules and ships tasks to workers. DAG is therefore used to optimize the execution plan (e.g., minimize shuffling data around).

In addition to RDD, thanks to SparkSQL, DataFrame and Dataset can be used as structured data in Spark programs. DataFrame/Dataset provide a single interface for efficiently working with structured data including Apache Hive, Parquet, and JSON files.

SparkSQL integrates relational processing with Spark functional programming to offer connection between RDD and relational tables.

DataFrame is a distributed data set built on top of RDDs and organized in named columns. It is similar to a relational database, but it is immutable once constructed, permits to track lineage, enables distributed computations. It can be generated reading from a file, parallelizing a collection list, transforming an existing DataFrame and in general applying transformations or actions.

To access the DataFrames either SQL Context or Hive Context is needed:

- **SQLContext:** It is the entry point for working with structured data (rows and columns) in Apache Spark. It permits the creation of DataFrame objects as well as the execution of SQL queries.
- **Hive Context:** It works with Hive tables, a descendant of SQLContext. Hive Context provides a richer functionality than SQLContext.

DataFrames are more advanced than RDDs. They have i) Custom Memory Management with no garbage collection overhead (the data are stored in a binary format and the schema of the memory is known), ii) Optimized Execution plan for the execution of a query (final execution takes place on RDDs).

DataSet is an extension of DataFrame; it provides strongly-typed, immutable collection of objects that map to the relational schema. It is possible to convert the Type-safe data set to an “untyped” DataFrame.

2.2.3.2 Apache Spark streaming

Spark streaming enables large-scale stream processing fully integrated with Spark batch and interactive processing. It provides a simple batch-like API that implements a type of stream processing based on micro-batch. It can interact with Kafka, Flume, and the like, to obtain data streams. Figure 13 shows how Spark Streaming works.

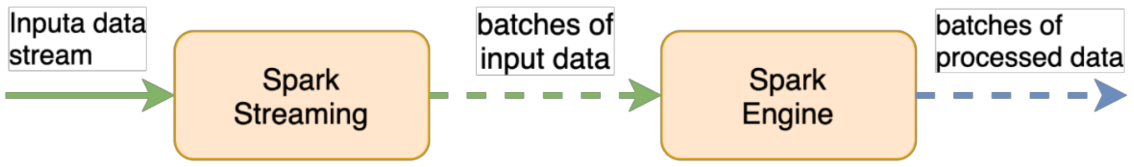


Figure 13: Apache Spark streaming workflow.

Spark Streaming receives live input data streams and divides the data into batches. The Spark engine processes each batch as RDDs to generate the final stream of results (again small batches). Batches of input data are replicated in memory of multiple worker nodes to obtain fault-tolerant processing.

More specifically, the main data abstraction of Spark Streaming is DStream, which is a sequence of RDDs representing a stream of data. Transformations modify DStream to another DStream using standard RDD operations or stateful operations. The processing of each batch has no dependency on the data of previous batches.

Stateless transformations can combine data from many DStreams within each time step. Stateful transformations use data or intermediate results from previous batches and compute the result of the current batch. Stateful transformations are operations on DStreams that track data across time. For instance, windowed operations that act over a sliding window of time periods.

2.2.3.3 Apache Airflow

Apache Airflow is an orchestrator of processing pipelines aimed to provide scheduling and monitoring capabilities. Airflow pipelines, differently from other orchestrators, are not designed with a metalanguage (e.g., XML), but directly implemented in Python. This permits better dynamic pipeline generation. The pipeline is modelled as a DAG of tasks. Tasks can be Spark jobs; Airflow can easily manage failures or complex and dynamic workflows of Spark jobs.

Figure 14 shows the Airflow architecture.

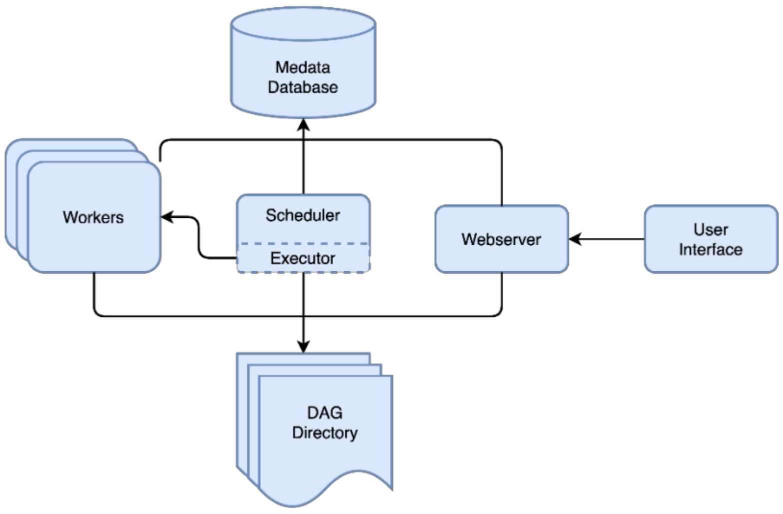


Figure 14: Airflow architecture.

More in detail, Airflow Architecture is based on the following components:

- Scheduler. The most important component of Airflow. It orchestrates DAGs and the corresponding tasks based on interdependencies and fair scheduling between DAGs. To execute a DAG it submits tasks to the executor.
- Executor. It handles running tasks. By default, Airflow runs everything inside the scheduler, but it also manages workers or external executors such as Celery and Kubernetes.
- Webserver: It constitutes the UI of Airflow. It provides interfaces to inspect, trigger and debug DAGs and tasks.
- DAG Directory: It contains DAGs definition. It is primarily accessed by the scheduler and executor.
- Metadata database: It stores metadata about DAGs like runs and other Airflow configurations. It is used primarily by the scheduler and executor to store DAG-related information.

In Airflow, a workflow (i.e., a DAG in execution) can be triggered manually, by external triggers, or by a scheduler. It can also be modified, even if scheduled, and either the modified version takes place at the next scheduling event or (using the backfilling mechanism) the modified version (or part of it) is executed back in time on previous data if available.

2.3 Architectural data flow

The big data engine in this deliverable implements a data flow composed of two sequential procedures: i) ingestion procedure, involving data sources and ingestion pipelines, ii) analytics procedure, involving analytics and visualization pipelines. Figure 15 shows the two procedures in terms of data processing pipelines, that is, a sequence of processing tasks that can be parallelized to exploit the big data engine capabilities.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

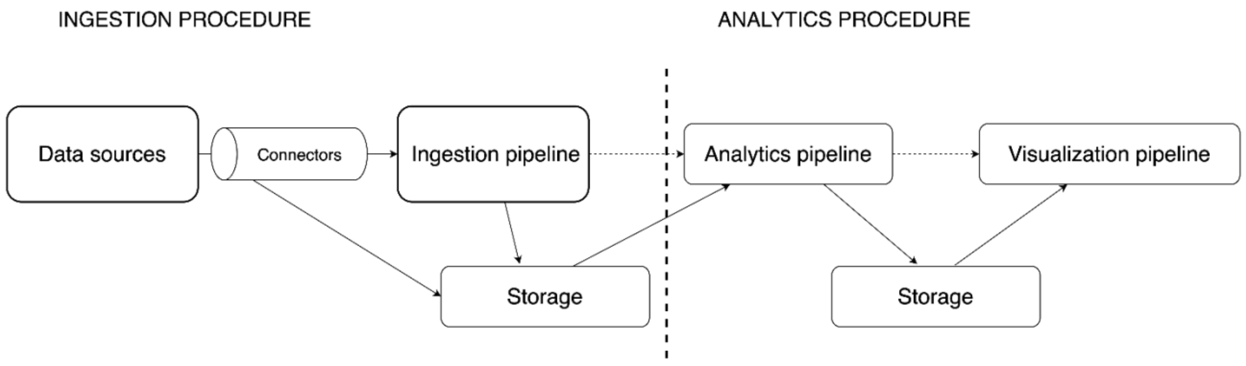


Figure 15: Architectural data flow.

Depending on the goal, pipelines can be roughly classified as i) ingestion pipelines, they capture and transform data with the purpose of saving them for further analysis. Ingestion pipelines deal with different data formats (i.e., structured, unstructured, and semi-structured); ii) analytics pipelines, they execute specific analysis on data, which may include data preparation tasks (e.g., normalization, cleaning, selections) to make data suitable for specific analytics, iii) visualization pipelines, they present the output of an analytics to users. Pipelines are implemented using processing components and executed with the support of the resource management components.

Considering the ingestion procedure:

- *Data Sources* are the source of row data. They are connected to the ingestion pipeline via specific connectors (e.g., Kafka queue, API, specific ingestion tools).
- *Ingestion pipeline* is a pipeline responsible to transform data flows prior to storing them. Data can be either ingested in batches, in real-time (stream), or using a hybrid combination of the two patterns. In the first case, the ingestion pipeline periodically collects (via a pull or push approach) groups of data. In the second case, data are continuously sent and treated by the ingestion pipeline as soon as they are collected. The hybrid way, called micro-batch, groups data in very small batches to enhance the data ingestion and corresponding analytics.
- *Data storage* is a distributed medium where data are permanently stored. Data storage behaves differently depending on the data format, that is, structured or unstructured data. The two data formats correspond to different storage solutions, different query languages, and different types of operations that can be performed on data. Structured data have an expected format and schema, and are generally stored in RDBMS where they are analyzed, managed, and queried. Unstructured data can come in many formats (e.g., email, social media posts, images/video) and are commonly stored in schemaless NoSQL datastores. In our architecture, data storage is implemented by Hive and HDFS.

Considering the analytics procedure:

- *Analytics pipeline* is responsible to retrieve some values from the data after ingestion (e.g., build a ML model, or apply a model for prediction). A very simple analytics pipeline can be expressed as queries to get insights and provide simple analysis (e.g., aggregations).
- *Visualization pipeline* is a pipeline designed to prepare data to be graphically visualised. The considered data are the outcome of an analytics or any aggregations on the data in the storage. To this aim, it usually includes some processing tasks suitable to lower the dimension of the plot or aggregate data in a specific manner.

Figure 15 also depicts a streaming pipeline integrating ingestion and analytics procedures into a single pipeline for prediction (dashed arrows) and a typical batch processing or model creation analytics (solid arrows).

We note that the ingestion pipeline is the middleware between data sources and analytics pipeline, and is then the most suitable point to implement our access control approach.

2.3.1 Data flow

The architecture in Figure 15 describes the different phases of the data management process in Section 2.1 and the corresponding data flow. Each phase is driven by a big data pipeline where structure and frequency of the data flow have primary importance.

Concerning the structure of the data flow, the type of data (structured, unstructured and semi-structured) affects the ingestion procedure and how data need to be structurally transformed (if needed).

Concerning frequency of the data flow, *batch*, *stream* (i.e., real-time flow), *micro-batch* are considered with the primary scope of defining how to analyse them. More specifically:

> **Batch –** Data are collected in large windows of time (usually ranging from 1 hour up to few days). The time to process them is not a critical factor. Processing data in batches introduces latencies between storage and data availability in analytics or visualization pipelines. If latencies are a negligible factor, dealing with data in batches is not an issue. Batch ingestion is normally obtained via API, file ingestion or specific ingestion tools. Batch analytics are the most traditional way to process data that derive from traditional data mining approaches, where batches of data are used to train and test a model.
>
> *Mapping on engine:* Ingestion via Hive or HDFS and analytics with Spark.
>
> **Stream –** Data come to ingestion or to analytics/visualization pipelines as soon as they are produced; in general, data in stream format are produced and dispatched one at a time (i.e., as a unit). In such a format, the latency to perform analytics and output results is a critical factor. Data streaming and real-time processing are strongly connected concepts. The videos coming from a surveillance camera or (system) log entries are just a common example of data stream. Streams are normally treated by end-to-end pipelines (dashed arrows in Figure 15). Stream ingestion is normally obtained via queue systems and it is in line with the analytics to keep the real time constraint.
>
> *Mapping on engine:* Ingestion via Kafka and analytics with Spark-stream.
>
> **Micro-batch –** it is a hybrid format between stream (under the aspect of velocity in data collection) and batch (aggregated data are sent in small files). Micro-batches are used when real-time processing is not a critical issue but, of course, it is not expected to wait hours for their processing. This kind of data can be obtained by dividing a batch in many smaller units. Micro-batch ingestion is obtained similarly to the stream one with buffered queue or scheduled API calls, but with the scope of storing the data prior to analytics procedure. Micro-batch analytics work similarly to the stream one, but more focused on incremental modelling and continuous predictions
>
> *Mapping on engine:* Ingestion via Kafka, HDFS and analytics with Spark-stream.

2.3.2 Architectural view on data ingestion procedure

Data ingestion procedures are composed of three steps: i) *extract*, the process of pulling the source data from the original data source (played by the data source connectors), ii) *transform*, the process of changing the structure of data to integrate with the target data system (ingestion pipeline in Figure 15), and iii) *load*, the process of depositing the data at rest into a data storage (storage in Figure 15).

The ordering of these steps defines the two main approaches: ETL (Extract, Transform, Load) and ELT (Extract, Load, Transform). ETL is typical for data warehouses and Online Analytical Processing (OLAP). In this case, data are sent in a temporary staging area prior to being transformed to a structured form. ELT is emerging as the prevalent scheme for big data analytics: data are stored immediately into a data lake storage system (dotted arrow in Figure 15), without any structural transformations.

In terms of data governance, ETL supports a priori compliance to regulations; non-compliant data are in fact never stored in the data warehouse. ETL ingestion pipelines may require modification in case the data structure

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

does not support a new type of analysis. On the contrary, ELT is focused on ingesting row data as they become available; data are transformed only when needed.

Our solution (presented in Chapter 3) is based on an advanced ETL schema where i) data transformation is the result of a policy enforcement, and ii) the target of phase load of the data ingestion procedure is the data lake infrastructure. The scope of this advanced ETL method is to enforce access control at ingestion time before an access request is received.

2.3.3 Architectural view on data analytics procedure

Data Analytics procedure analyses i) data after ingestion with the goal of building models or generating analysis or predictions (analytics pipeline), ii) data to produce suitable views for visualization and reporting (visualization pipeline).

Two aspects of analytics pipeline must be considered: batch analytics for model creation and stream analytics for prediction.

In the past, large amounts of data were mainly collected in batch formats and the technologies were not suitable for real-time data ingestion/processing. To manage batches, existing systems mainly relied on the ETL architecture. Subsequently, systems have been designed with architectures called Kappa (see Figure 16) that has been developed purely to handle data in stream mode. But this was also its limitation, since no batches of data could be collected using Kappa architectures. The introduction of Lambda architectures (see Figure 17) has overcome the limitations of Kappa architectures.

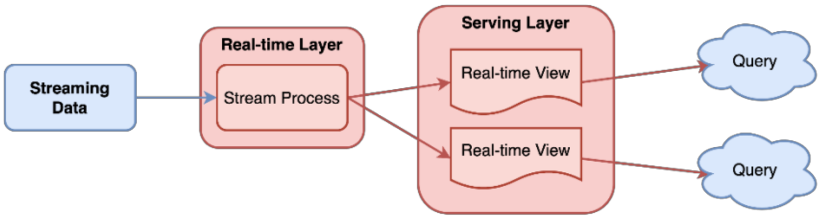


Figure 16: Kappa architecture.

The definition of the Kappa Architecture is attributed to Jay Kreps.[2] The underlying idea is to handle both real-time data processing and continuous reprocessing in a single stream processing engine. Kappa Architecture is a software architecture pattern and can be assumed as a simplification of the more complex Lambda Architecture. In the Kappa architecture, data are ingested as streams only and processed in a so-called "*real-time layer*", whereas the "*serving layer*" provides optimized responses (*real-time views*) to queries.

[2] http://milinda.pathirage.org/kappa-architecture.com/

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

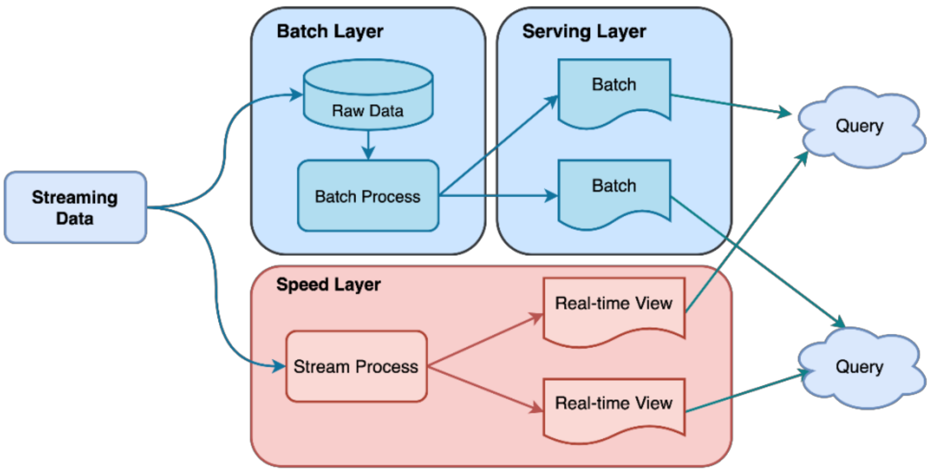


Figure 17: Lambda architecture.

Attributed to an idea of Nathan Marz, the Lambda architecture (the approach used in our solution) is the most applied architecture dealing with stream ingestion and processing. Lambda architecture is composed of three layers: batch layer, speed layer and serving layer. Batch layer deals with very large amount of data (main feature of data batch ingestion modality). The *batch layer* stores the incoming data and, by means of *batch processes* (performed at some interval of long duration), computes the batch views. On the other hand, the *speed layer* processes data streams in real time to provide *real-time views* on the most recent data. The *serving layer* makes the views produced by the previous two layers available to answer queries from the analytics processes. Despite Figure 17 shows that *serving layer* just handles those views outputted by the *batch layer*, in practice its utilities also cover real-time views.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

3 Data ingestion and access control

This chapter extends the data architecture in Figure 15 to achieve full data governance on the ingested data (see Section 3.4.2). More specifically, we propose an access control-based data governance approach that works at ingestion time and supports sanitization and transformation of data prior to the storage in the data lake.[3] This approach, based on data sanitization and transformation, would enable the support for GDPR compliance requirements at ingestion time. It is based on an advanced ETL schema (see Section 2.3.2 for details) focused on balancing the flexibility of ELT with the control capability of traditional ETL where *i)* data transformation is the result of an access control policy enforcement and *ii)* the loading target is the data lake infrastructure.

Our advanced ETL schema enforces access control policies at ingestion time before any access request is received. Access control policies are designed and enforced to guarantee that the data stored in the data lake are compliant with the policies defined for specific services/actors (possibly mandated by laws and regulations) and directly accessible with no delay when requested. To this aim, data are first annotated with labels modelling their peculiarities in terms of risk, sensitivity, or privacy; policies are then enforced on data according to these annotations; sanitized data are finally stored in the data lake and then used for the analytics in Chapter 0.

3.1 Data ingestion procedure

This section presents the details of our data ingestion procedure depicted in Figure 15. In particular, we detail *i)* data sources and connectors*,* describing methodologies for collecting data from different sources, *ii)* ingestion pipeline, detailing the ingestion pipeline tasks, and *iii)* ingestion storage, describing different storing solution for ingestion procedure.

3.1.1 Data sources and connectors

The process of data extraction is called data sourcing or data extraction (E in ETL or ELT). It is the process of extracting data from several external or internal data sources composing an IT infrastructure/ecosystem. It is performed by connectors that are specifically tailored for the peculiarities of the data sources. Connectors can be roughly classified based on two kind of communications they support: client/server (or request/response) and publish/subscribe. Client/Server is mostly related to a synchronous communication, whereas Publish/Subscribe refers to an asynchronous way of data shipping.

The request/response mechanism is traditionally associated to the concept of Application Programmable Interface (API). API is an interface that defines how two software or services communicate. APIs hide the underlying implementation by providing the developer with a selected and well-defined set of functionalities. There are two kinds of implementation usually followed to realize (Web 2.0) API: SOAP (i.e., Simple Object Access Protocol) and REST (i.e., Representational State Transfer). Some differences between these two ways of implementation are listed below:[4]

- SOAP is a protocol, whereas REST is an architectural pattern.
- SOAP uses service interfaces to expose its functionality to client applications, while REST uses Uniform Service Locators (URL) to access to the components.
- SOAP needs more bandwidth for its usage, whereas REST does not need much bandwidth.
- SOAP only works with XML formats, whereas REST works with plain text (e.g., XML, HTML, JSON).
- SOAP cannot use of REST, whereas REST can use SOAP.

Nowadays, REST (see Figure 18) is the most used approach to implement APIs. Another peculiarity of REST is that the application and functionality are divided into web resources [3].

[3] Part of the content of Chapter 3 has been presented in a paper titled "Dynamic and Scalable Enforcement of Access Control Policies for Big Data" accepted for publication at ACM MEDES 2021 [30].
[4] https://www.guru99.com/comparison-between-web-services.html

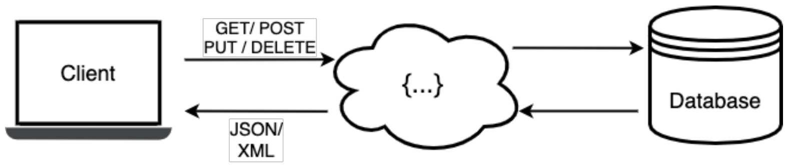


Figure 18: REST paradigm in action.

The publish/subscribe mechanism is increasingly applied due to its suitability in fully distributed architectures to decouple communications between independent components. It permits asynchronous communication supporting, in some cases, components (data producer and data consumer) having different velocity. This approach is often used in microservices-based architectures to create a communication bus. In this context the data that need to be transferred are (technically) called *events*, while their flow is called event-streaming. Publishers of events can be databases, sensors, mobile devices, cloud services and so on; practically, our data sources.
Events are classified and durably stored in topics. A topic is similar to a bucket, and events are the items in that bucket. A topic can have zero or many producers (i.e., data sources) and zero to many consumers (i.e., applications, tools, or storage devices). Apache Kafka is the most largely used tool for publish/subscribe mechanism (see Section 2.2.1.2 for further architectural details) based on tags and capable to handle message persistency.

3.1.2 Ingestion pipeline

The ingestion pipeline is composed of a sequence of tasks starting from data extraction (using connectors in Section 3.1.1), data transformation (in most of the cases format transformation) and data storing (to store data on a specific medium/storage service). Figure 19 shows a fine-grained ingestion pipeline structure that extends the ingestion pipeline in Figure 15.

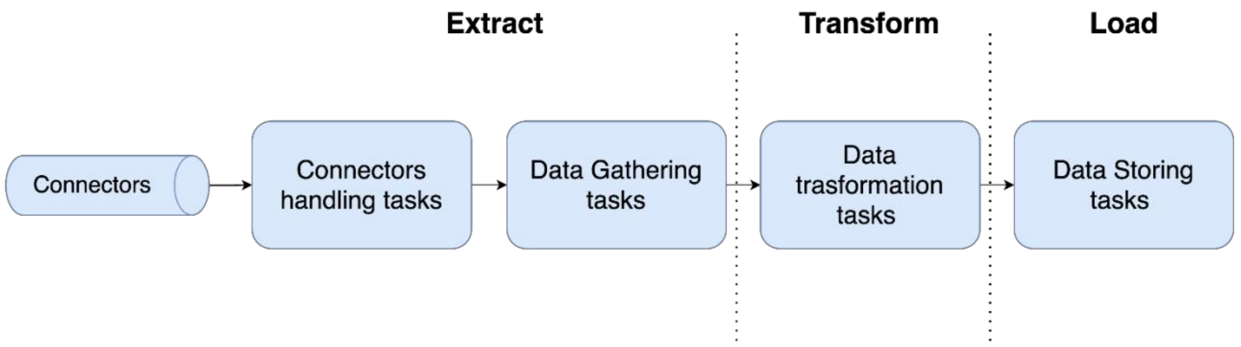


Figure 19: Fine-grained ingestion pipeline structure.

The ingestion pipeline should consider the following aspects:

- Data is extracted in its raw format and with producer speed. Incoming data are received in their respective raw format with no restrictions on data extension or size. *Connectors handling tasks* should be capable to deal with such format/size and also the velocity of the data producers. For instance, if a buffering is needed to collect a set of data prior to transform them, such buffering should be implemented at connectors level. Similarly, if row data are provided with a specific protocol, such protocol should be interpreted prior to the transformation and just after the *connectors* by the *data gathering tasks*.
- Data can require transformation or augmentation prior to be ingested. Since the data are received in raw format, some pre-processing is often necessary at this stage. Pre-processing is performed by means of *data transformation tasks*. Data transformation may involve changes of data format to match ingestion

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

storage capabilities and/or augmentations to enrich ingested data with contextual information of by merging different data sources.

3.1.3 Ingestion storage

According to their nature (i.e., structured, unstructured and semi-structured), data can be finally stored on ad-hoc designed mediums (i.e., NoSql, RDBMS, HDFS, ecc,) in a persistent fashion. The storage procedure is triggered by the Data Storing tasks and concludes every ingestion pipelines.
To store structured data a solution based on relational DBMS is adopted. As anticipated, the data before storage are pre-processed, hence transformed or augmented. In our engine, structured data are stored by means of Apache Hive (described in Section 2.2.1.3). This means that the *Data Storing tasks* handle procedures to connect to Hive and the final checks on the data format to guarantee absence of incompatibilities (e.g., data types) between the storing technologies, in this case Hive, and the data after transformation.

As far as unstructured and semi-structured data are concerned (e.g., documents, folders, images and so on) our engine adopts HDFS. We recall that HDFS is a distributed, scalable, and portable file system that suites well in dealing with big data issues (see Section 2.2.1.1).

3.2 Data annotation

Data annotation extends the data transformation tasks (in Figure 19) of the ingestion pipeline adding annotation capabilities. More in detail, it extends the augmentation capabilities of data transformation tasks by adding additional meta information on the data.

The term data annotation usually refers to the process of adding relevant and further information (called metadata) to a single/set of data. Data annotation or (meta)tagging are often used as interchangeable terms. A tag is a label consisting of a key and an optional value that is assigned to a resource. In cloud scenarios, tagging is widely used to perform more sophisticated filtering or reporting on resources.

Data annotations have two main advantages:

1. metadata tagging permits to identify, organize, and extract value out of the (possibly unstructured) raw data ingested. For example, it is possible to capture document semantics through tags.
2. Tagging enables to categorize resources in different ways (e.g., by purpose, owner, environment, or other criteria), encompassing conventional database schema information. Moreover, relationships between attributes of different data sets can be indicated explicitly as tags, without the need of some sort of, possibly computational expensive, data normalization.

When using a tagging solution, it is fundamental to devise and agree on a consistent set of tag keys, called vocabulary. However, the choice of the tags and how to apply them to resources depends on the specific use case or working environment. For example, tags can be used to express different data aspects: type (e.g., video, image, log, tuple, file), details on the content, origin/author (e.g., sensor or city), environmental context (e.g., public office, school, gathering place, street), purpose of usage (e.g., monitoring, surveillance, threat prevention, viability), operational requirements, or data sensitivity.

Table 3: Tagging use cases. provides an example of possible vocabularies to be used in different use cases.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

Table 3: Tagging use cases.

Use case	Description	Tag Key and possible values
Data classification	User-defined sensitivity of data	*DataClassification:* Public, Secret, Top Secret
Protected health info	Health data created, received, stored or transmitted by HIPAA-covered entities (e.g., demographic data, medical histories, test results, insurance information)	*PHI*
Privacy classification	Any information that permits the identity of an individual to be directly or indirectly inferred	*PrivacyClassification*: PII-id, PII-quasi-id, PII-Sensitive
Disaster recovery	Business criticality of the application, workload, or service	*DR*: Mission-critical, Critical, Essential
Business criticality	Business impact of the resource	*Criticality*: Low, Medium, High
Environment	Deployment environment of the workload or service	*Env*: Prod, Dev, QA, Stage, Test
Roles	Roles of employees in a hospital	*Role*: IT admin, Doctor, Nurse, Radiologist, Cardiologist, ...
Owner name	Owner of the workload or service	*Owner*: mrossi@company.com
Type	Type of document	*Type*: JPG, PDF, TXT, XML, HTML, …
Content	Content of document	*Content*: FacePic, EnvPic, ...
Creation date	Creation date of the resource	*CreationDate*: 2023-10-15
Source	Origin of data	*SensorID*: Camera123

For instance, use case *Privacy classification* should be used every time data privacy must be protected, that is, every time personal data are involved. Data privacy generally means the ability of a person to determine when, how, and to what extent personal information can be shared with or communicated to others.

According to the General Data Protection Regulation (GDPR),[5] personal data are any information relating to an identified or identifiable natural person (called data subject). Based on this definition, and on the technological advances that have improved data collection and analytics, personal data (Personally Identifiable Information - PII) fall into three categories:

(i) explicit identifiers, that is, any information that directly identifies individuals with certainty, such as national ID, assurance number, phone number, passport number;
(ii) quasi-identifiers, a set of data attributes that could jointly or uniquely identify a subject when combined with publicly available data, such as ZIP code, date of birth, and address;
(iii) sensitive information, data related to a person that do not permit direct identification. If linked to an individual, they reveal sensitive aspects of the individual private life.

The tagging vocabulary for use case *Privacy classification* then consists of a tag key *PrivacyClassification*, with three possible values (`PII-id`, `PII-quasi-id`, `PII-Sensitive`), representing the three categories.

[5] https://gdpr-info.eu/

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

3.3 Security and privacy-oriented data transformations

Security and privacy-aware transformations are special types of ingestion pipeline's transformation tasks that are focused on compliance to regulations and standards rather than simple format conversion. They need to have information about the semantics of the ingested data and this is normally provided via ad hoc design of pipeline (unfeasible in the dynamic context we are considering) or via data annotation (see Section 3.2). In the following, we present a taxonomy of security- and privacy-oriented transformations that are of interest in the context of our solution.

A taxonomy of data transformations collects all functions used to sanitize the ingested data. Data transformations are categorised depending on the security property they aim to guarantee.

Data transformations for confidentiality (or integrity). The conventional security mechanisms used to protect data are encryption or hashing. With the advent of multi-tenant distributed clouds and collaborative machine learning environments, ad hoc encryption schemes have been proposed, such as:

- **Attribute/based encryption (ABE) [4] [5].** It is a public-key encryption scheme where the key pair of a user and the ciphertext are dependent on a set of attributes. The decryption of a ciphertext is possible only if the set of attributes of the user wanting to decrypt the ciphertext matches the attributes of the ciphertext.
- **Identity/based encryption (IBE) [6].** It is an alternative to public key encryption, aiming to simplify key management in a certificate-based Public Key Infrastructure (PKI) by using human identities such as email or IP addresses as public keys.
- **Homomorphic encryption [7].** It is a form of encryption that permits performing computations on encrypted data without first decrypting it. The resulting computations are left in an encrypted form which, when decrypted, results in an identical output to the one obtained when performing computations on cleartext data.

Data transformations for anonymity. Data anonymization is the general term used to describe the process of hiding identifiable information that may lead to personal identification, so that users described by such data remain anonymous. Main anonymization techniques are described as follows:

- **Suppression.** It removes an entire part of data, for example by replacing original values with a symbol that represents a null character. In case of structured data, a column or an entire tuple can be removed, or substituted with a null character. For example, a licence plate number can be substituted by the ###### sequence. Since it consists in removing all data, it affects the usability.
- **Generalization.** It replaces the value with a less specific but semantically consistent value. Generalization is typically applied to structured data, however not all attributes are suitable for it. For example, numerical values can be easily generalized into a fixed range of values (e.g., age), whereas a user ID can hardly be generalized.
- **Masking and encryption.** They change characters to different characters, making the value inconceivable. They are a general case of truncation and of pre-fix preserving. Masking and encryption techniques make data useless for analysis. In addition, they are computationally inefficient: in the first case, values must be checked before changing them; in the second case, encryption has a cost.
- **Distortion.** It maps the value to a new/different value, obtained by using a hash function or a cryptographic primitive. Similarly to masking and encryption, distortion make data useless for analysis. Specific techniques (e.g., distance-preserving hashing) can preserve a given level of quality useful for analysis.
- **Swapping.** It rearranges variables randomly within a specific column in a relational database. It maintains the overall distribution of values, reducing the precision of data and their analysis. It is computationally efficient.

Data anonymization is frequently used to preserve privacy. In this case, it is important to identify which techniques should be applied to each PII identifier, quasi-identifier and sensitive data, since significant data should be retained as much as possible for analysis, but not at the cost of data leakage. The *k-anonymity* method [8] uses a combination of generalization, masking and suppression over an original data set to achieve *k*-anonymity, that is, to obtain an anonymized data set where each tuple is similar to at least other *k-1* tuples on the potentially identifying attributes that have been anonymized.

It is important to note that anonymization is meant to remove identifiable information forbidding later re-identification of a person. By contrast, pseudonymization manipulates data making them not-identifying data; original data however can be later recovered using additional information. Our approach can support both anonymization and pseudonimization.

3.4 Ingestion-time access control

This section presents the details of our access control system working at ingestion time, its functionalities, and some implementations details. The idea is to dynamically augment the ingestion pipeline with security and privacy-oriented transformation tasks instrumented by policies and data annotations, and triggered by policy enforcement.

In the following, we first describe the requirements for access control at ingestion time, we then describe in detail our access control policies.

3.4.1 Access control requirements

The big data scenario adds lots of complexity to data governance, especially from a *data protection* perspective. Data protection is a key requirement for ICT applications dealing with high volume of data from multiple data sources, structured and unstructured data, and a complex ICT infrastructure either on cloud or on premises owned by a coalition of different organizations (i.e., cities and operators). It is even more key in scenarios where most of the collected data include sensitive or personal information. Given the software and technological complexity of the big data ecosystems, implementing an access control system is not trivial.

Historically, companies have been handling data protection by adopting access control policies. Unfortunately, access control models developed in the 2000s for Service-Oriented Architectures fail to adequately manage the creation, use and dissemination of big data. They can be both ineffective and inefficient when applied to today's dynamic coalitions, where: *i)* partners join without necessarily integrating their cloud-based or on-premises ICT infrastructures, *ii)* collaborative processes are carried out involving multi-party data collection and analytics, *iii)* there are continuous changes in the security space structure of temporary coalitions, with many parameters for access right decisions unknown at policy writing time. Although, given its ability to support highly flexible and dynamic forms of data protection to business-critical data, Attribute Based Access Control (ABAC) [9] is currently adopted in big data projects as a common underlying model, the dynamic and decentralized nature of big data asks for new solutions. Current solutions are neither general nor scalable, since they are either platform-dependent or coarse-grained [10], and they often do not consider end-to-end security. Platform-specific approaches are designed for one system only (e.g., MongoDB, Hadoop), and leverage on native access control features of the protected platform. However, Hadoop's access control presents limitations, such as the complexity of deployment and the consumption of resources. Some recent proposals, like Federated Access Control Reference Model (FACRM) [11] [12] [13] are specifically tailored to the Apache Hadoop stack. On the other hand, platform-independent approaches have the advantage of being more general than platform-specific solutions. However, available platforms either model resources to be accessed as monolithic files (e.g., Microsoft DAC) or lack scalability. In addition, existing database-oriented approaches mainly leverage on recent research efforts for defining a unifying query language for NoSQL datastores (e.g., JSONiq and SQL++) [14], and therefore only focus on a particular type of analytic pipelines. The work of Hu et al. [15] can be seen as a first step towards a generalized access control model for big data processing frameworks, adaptable to Hadoop environment. However, the paper discusses the issues only from a high-level architectural point of view, without discussing a tangible solution.

A proper big data access control system must accomplish technical peculiarities of big data systems [16] [17], pointing to scenarios where huge amount of data are diverse, come at high rates and must be proven to be trustworthy, as clarified by the 5V storyline [18]. On top of this, big data systems are composed of an ecosystem of services, mainly from third parties, increasing the governance complexity under multiple perspectives. Rigid control, even if possible, does not fit the need to fully exploit the big data processing capabilities, while loosely control is not acceptable in many application contexts. New access control requirements then emerge as follows:

R1: Access to data must be evaluated before data analytics take place.

R2: Authorization should be the primary focus. Authentication is assumed to be managed by a separate and integrated module to guarantee federations within big data ecosystem.

R3: Access control enforcement must protect data during their entire life cycle. Access control policies must be enforced at each phase of the analytics process, guaranteeing that data are properly protected and shared only to authorized users and for authorized operations.

R4: Access control policies must support the specification of context-based access conditions, that is, access rights might depend on the run-time characteristics of the big data system (e.g., in the smart city scenario, a traffic control system should implement some form of adaptive signalling to mitigate dangerous situations during emergencies such as a car accident).

R5: Access control enforcement should support fine-grained access control, dealing with both structured and unstructured data. When structured data are considered, policies can refer to a single cell, a column, a tuple or an entire table of structured data. When unstructured data are considered, policies can specify the portion of the unstructured file whose access need to be regulated.

R6: Access control enforcement should not use data ownership as the only attribute to define access rights, but rather being applied at ingestion time on the basis of the evolving state of resources and individuals within a specific big data context.

R7: Access control should protect the privacy of sensitive data.

R8: Access control enforcement should be driven by dynamic and contextual annotations on data.

R9: Access control enforcement should be highly efficient and scalable to accomplish the increasing cardinality of data and rate of requests.

3.4.2 Access control policies

The novel ingestion procedure in Figure 19 is enriched with *access control enforcement capabilities* and performs data-model transformations of the ingested resources. The transformations depend on the state of the collaboration and the specific target where data are routed (separation of duties). Our approach makes policies verifiable and adaptable to the evolving state of resources and individuals within a specific big data context. We note that implementing access control at ingestion time can lead to data duplication in case a specific data source is ingested by two different actors or routed to different targets having different access control policies to comply with. However, in a big data context, such duplication can be considered negligible compared to the advantages obtained in terms of data governance and analytics performance.

Our methodology builds on and extends the policy structure of traditional attributed-based access control systems, whose policies rules are expressed as a tuple of the form

```
<subject, action, object, [context], [obligation], deny/allow>
```

where an optional obligation specifies an action that must be performed before the policy is enforced and the context also includes the purpose of the access. Obligations are in most of the cases actions that are not directly associated with the controlled data. In collaborative big data scenarios, however, data are shared among large coalitions of different organizations or operation teams in a city, but it is still important to ensure that sensitive data are properly protected and only exposed to the level required for specific business needs. In this context, deny access to data is the very last decision to take. Deny means no analytics, and then no value. In this paper, we put forward the idea that in the big data context preemptive data transformations are more suitable that denying data access. Indeed, we propose a platform-independent methodology still based on the common eXtensible Access Control Markup Language (XACML) architecture, where policy rules are of the form

```
<subject, action, object, context, data transformation>
```

where:

- the `subject` is the actor or the service who makes a request to access a certain object. It can be expressed either as a unique identifier or as a role/group the subject belongs to; we note that in our case the subject can be a *** as the specific target where data are routed;
- the `action` declares the operation (e.g., read, write, update, delete, select) on the object for which permission is requested;
- the `object` defines the data, system component or service to be accessed;
- the `context` is a set of attributes (independent of a particular subject, object or action) that are relevant to the authorization decision;
- the `data transformation` is a type of obligation that sanitizes data resources to guarantee a minimum level of access to data.

We note that, deny access to data can happen in extreme cases when data transformations delete all data resources.

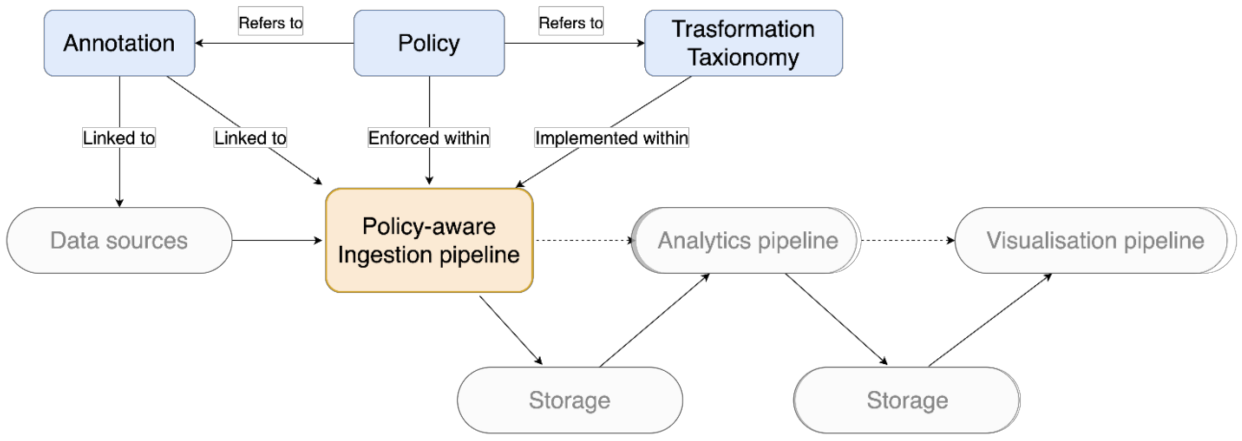


Figure 20: The ingestion-time access control methodology.

Figure 20 shows the big data processing system in Figure 15 extended with our policy-aware ingestion pipeline that extends the ingestion pipeline with the ability to enforce access control policies and transforming data. The policy-aware ingestion pipeline builds on the following building blocks.

1. *Data annotation* (Section 3.2) as the process where data sources are enriched with contextual information and metadata on data sensitivity/peculiarities. Such a process indicates to the ingestion pipeline the proper way of dealing with the data and drives data transformation.
2. *Taxonomy of data transformations* (Section 3.3) as a library of transformations that can be executed on data resources. It includes pruning, reshaping, or encrypting/decrypting the different resource parts. We support both unstructured raw data and multiple data models, including popular service/big data models, such as encoded JPEG image files, CSV files, JSON, HDFS, and XML. Each entry in the taxonomy is mapped to a transformation function at ingestion layer.
3. Policies expressing the need to enforce certain transformations based on attributes and actors executing the ingestion process.

Figure 21 shows more details of how the access control system enriches the ingestion pipeline in Figure 19. Specifically, it shows that annotations can be applied to i) data sources specified in the *Connectors handling tasks*, ii) data prior and after transformation in the *Data Transformation tasks*, and iii) data while moved to storage with the *Data Storing tasks*. We note that the last annotation before the data storage is used either to support data extraction on the data lake for further ingestion procedures or data control prior to the analytics. It

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

also shows that security and privacy-oriented tasks take places prior to the storing tasks and that their definition and application are driven by taxonomies of transformations and policies.

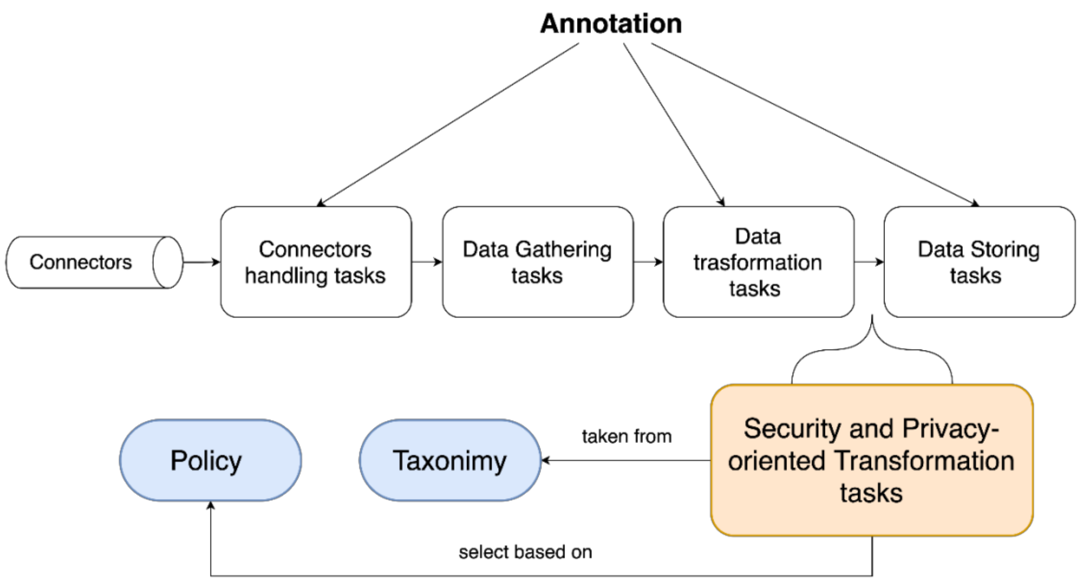


Figure 21: Details about ingestion pipeline structure.

Examples of data annotations and transformations for privacy-preserving access control

The data annotation process can be exploited to facilitate the design and implementation of security policies (e.g., multi-level or role-based policies), or to limit access to specific environments, users, applications (e.g., development, test, quality assurance or production). In case of unstructured data, tagging also helps in defining policies at different levels of granularity.

Consider for example a traffic monitoring service regulating accesses to a Low Emission Zone in a smart city scenario. Data sources are made of videos of city traffic. At ingestion time, the ingestion pipeline enriches videos (unstructured data) with vehicles' information (structured data). Our data annotation refers both to data sources and to the data generated by the ingestion pipeline, with the aim to preserve users' privacy. More specifically, the tagging vocabulary we consider contains:

- a set of tags to be associated to the snapshots metadata taken by the video camera to add information such as *ViolationTime*, indicating the time the picture was taken, *CameraID* to be used to retrieve the street controlled by the camera, *Type*=`JPG` and *Content*=`PlateID` specifying that the JPG file contains a picture with a given plate number;
- the privacy classification tags introduced in Table 3, to be assigned both to the *Content* of the snapshot file and to the vehicles' information that includes owner's personal attributes (e.g., license plate number, owner and home address).

Table 4 shows the privacy classification tags applied to the vehicles' information table. Since the table contains PII information, the columns of owner's personal attributes are labeled as `PII-id` or `PII-quasi-id` during the annotation phase and transformed by the ingestion pipeline to preserve privacy. The result is shown in Table 5.

Table 4: Result of data annotation process on structured data.

PII-ID	PII-SEMI-ID	PII-SENSITIVE	PII-ID	PII-SENSITIVE
plate_id	**owner_LN**	**owner_FN**	**owner_BD**	**Address**
M-X2495	Brown	Alice	1971-01-01	11, Grant Avenue
B-AR667	Green	Bob	1968-08-16	345, Market Street
B-A7813	White	Charlie	1950-03-03	2, Van Ness Avenue
ALZ-1234	Black	Trudy	2001-01-02	23, Alemany Boulevard

Table 5: Vehicle's owner data after the transformation (i.e., anonymization).

plate_id	owner_FN	owner_BD	address
M-xxxxx	Alice	1971-01-01	Grant Avenue
B-xxxxx	Bob	1968-01-01	Market Street
B-xxxxx	Charlie	1950-01-01	Van Ness Avenue
ALZ-xxxxx	Trudy	2001-01-01	Alemany Boulevard

More in detail, different anonymization techniques have been applied to the different columns:

- The first column records the plate number. In the example, we followed the German plates' format, where the first 1-3 letters are for the Region and City (e.g., M stands for Munich, B for Berlin and ALZ for Alzenau in Bavaria), then there are the state or city seals, followed by random letters or numbers. Plate ids have been only partially masked by substituting the free sequence of letters and numbers, to maintain the information of the possible origin of the vehicle.
- The column with the last name of the owner has been suppressed for obvious reasons. On the contrary, in the example, we maintained the first name of the vehicle's owner (for possible gender statistics).
- Generalization has been applied to the column with the birth date of the vehicle's owner, where only the year value corresponds to the original one.
- The last column shows another example of generalization applied to the street address, where the street number has been removed.

3.5 Mapping on the data engine

This section presents the implementation details and how the data ingestion process is technically integrated within our engine in Section 2. Specifically, we detail the data ingestion process from a technical standpoint by referring to those technologies and frameworks used in each specific step from the sourcing to the storage. We recall that all components and tools in our engine have been discussed in Chapter 2.

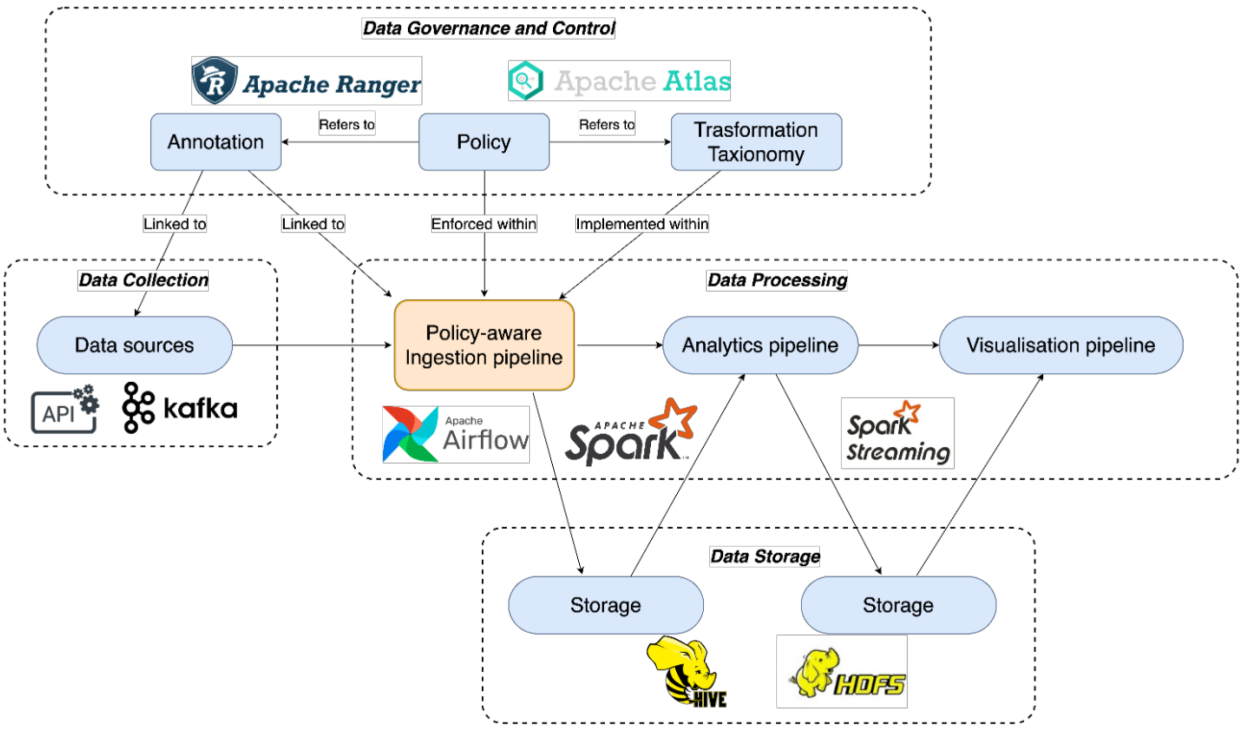


Figure 22: Technologies and frameworks in data ingestion and access control.

Figure 22 extends Figure 20 showing the link between our methodology and the technologies adopted in our big data engine (see Figure 2). Note that a single technology can be used to support different entities in our methodology. For instance, Spark is used to support all computations carried out in the pipelines.

In the following, we specify the different areas in Figure 22 and present how they are implemented in our engine.

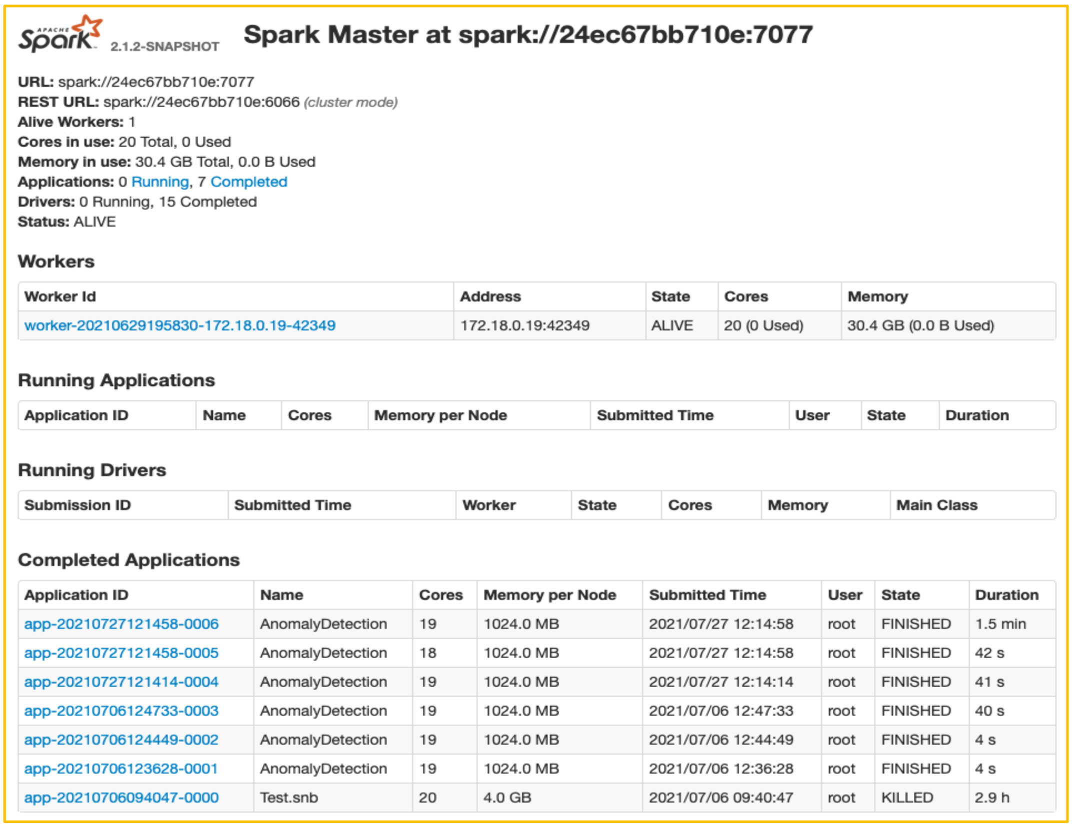
3.5.1 Area data collection

As mentioned in Section 2.3.1, there are three modes to insert data produced by sources (e.g., sensor stations) or other providers (e.g., cities): batch, micro-batch and streaming. In the case of data ingested in batch form, the data collection process is executed by means of communication interface offered by the system (API in Figure 22). Through these connectors, it is possible to technically input chunks of new data into the platform. Two are the possible ways to perform the data collection: active (i.e., by directly picking up data -- event-based) or passive (i.e., by a regular data receiving process -- time-based). The APIs hide the underlying implementation by offering to users only a selected and well-defined set of functionalities.

In the case of data ingested in streams, a Kafka-based queue mechanism is used. Kafka queues are used to identify the different types of sources (e.g., devices, cities) and data types (e.g., images, tuples, logs, videos). Each producer sends data towards a specific "queue", that is, appropriately labelled to distinguish them from each other in terms of meaning, content, context, and so on.

3.5.2 Area data processing

It includes all the pipelines from ingestion to visualization. In our engine the pipelines are written and executed by Spark or Spark stream (see Section 2.2.3) depending on the nature of the analysis to be carried out. Spark is capable to trace the execution of different jobs detailing the execution status. Figure 23 shows the Spark dashboard of our platform where the task "AnomalyDetection" was executed multiple times for testing purposes. It also details the structure of the cluster of Spark nodes.


Spark 2.1.2-SNAPSHOT **Spark Master at spark://24ec67bb710e:7077**

URL: spark://24ec67bb710e:7077
REST URL: spark://24ec67bb710e:6066 *(cluster mode)*
Alive Workers: 1
Cores in use: 20 Total, 0 Used
Memory in use: 30.4 GB Total, 0.0 B Used
Applications: 0 Running, 7 Completed
Drivers: 0 Running, 15 Completed
Status: ALIVE

Workers

Worker Id	Address	State	Cores	Memory
worker-20210629195830-172.18.0.19-42349	172.18.0.19:42349	ALIVE	20 (0 Used)	30.4 GB (0.0 B Used)

Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

Running Drivers

Submission ID	Submitted Time	Worker	State	Cores	Memory	Main Class

Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20210727121458-0006	AnomalyDetection	19	1024.0 MB	2021/07/27 12:14:58	root	FINISHED	1.5 min
app-20210727121458-0005	AnomalyDetection	18	1024.0 MB	2021/07/27 12:14:58	root	FINISHED	42 s
app-20210727121414-0004	AnomalyDetection	19	1024.0 MB	2021/07/27 12:14:14	root	FINISHED	41 s
app-20210706124733-0003	AnomalyDetection	19	1024.0 MB	2021/07/06 12:47:33	root	FINISHED	40 s
app-20210706124449-0002	AnomalyDetection	19	1024.0 MB	2021/07/06 12:44:49	root	FINISHED	4 s
app-20210706123628-0001	AnomalyDetection	19	1024.0 MB	2021/07/06 12:36:28	root	FINISHED	4 s
app-20210706094047-0000	Test.snb	20	4.0 GB	2021/07/06 09:40:47	root	KILLED	2.9 h

Figure 23: Spark tasks execution overview (dashboard).

Pipelines are then orchestrated together using Airflow that is capable to schedule their execution. For instance, to meet Lambda architecture requirements, Airflow can schedule a batch model processing pipeline aimed to update the model when enough data are available on the data lake. It can also schedule execution of the continuous prediction pipeline and synchronise it with visualization pipeline. In our scenario, Airflow is also used to orchestrate single tasks in the pipeline when needed, for instance, in case of very complex analytics pipelines. Figure 24 shows Airflow dashboard with a DAG where ingestion, analytics and visualization pipelines are put in sequence mapping the architecture of Figure 22.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

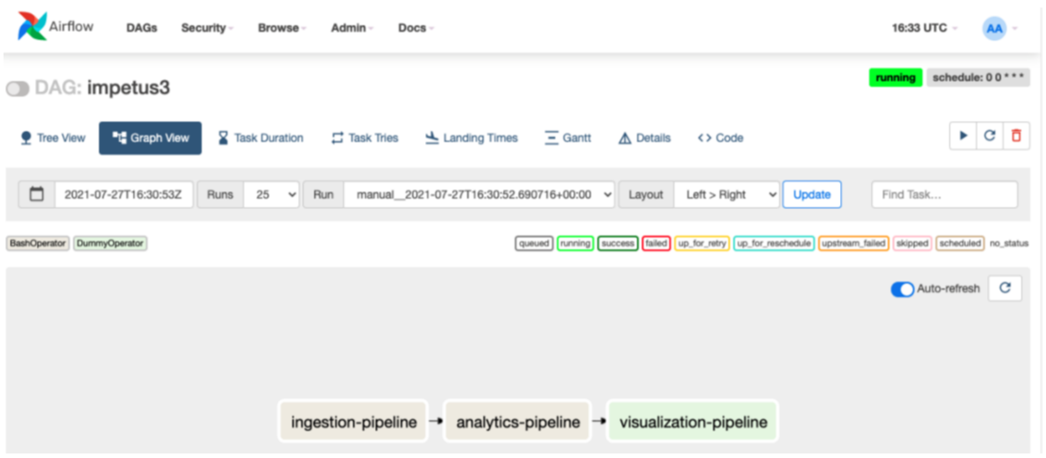


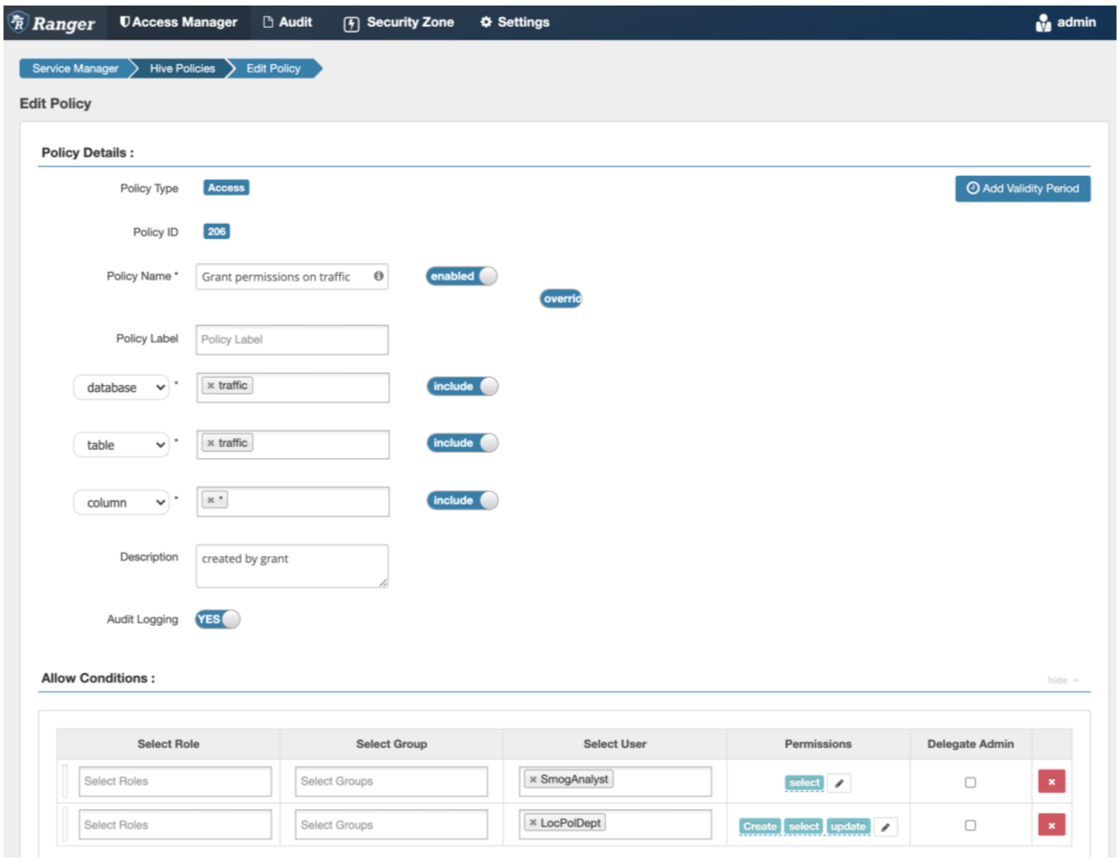
Figure 24: Pipelines workflow (by airflow UI).

3.5.3 Area data governance and control

This area contains technologies to support policy-aware ingestion pipelines (see Sections 3.2, 3.3, 3.4). More specifically, we adopted Apache Ranger for the definition, assignment, and enforcement of access control policies and Apache Atlas for annotations instrumenting the policies.

Figure 25 shows the Apache Ranger user interface to define an access control policy named "grant select user1 on traffic". In particular, from the "Policy Details" panel, we can see that such a policy targets all columns of table *traffic* in the database called *traffic*. In the lower panel, section "Allow conditions" specifies the permissions granted to the users as follows.

- *SmogAnalyst* can perform only select operations.
- *LocPolDept* can perform create, select, and update operations.



Ranger | Access Manager | Audit | Security Zone | Settings | admin

Service Manager > Hive Policies > Edit Policy

Edit Policy

Policy Details :

Policy Type	Access
Policy ID	206
Policy Name *	Grant permissions on traffic (enabled) (override)
Policy Label	Policy Label
database *	× traffic (include)
table *	× traffic (include)
column *	× * (include)
Description	created by grant
Audit Logging	YES

Add Validity Period

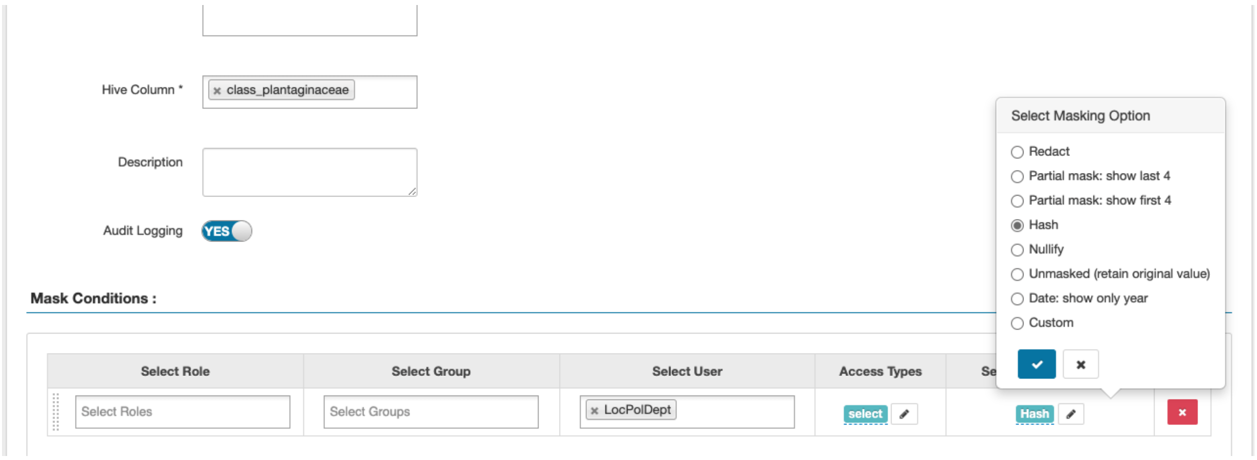
Allow Conditions :

Select Role	Select Group	Select User	Permissions	Delegate Admin	
Select Roles	Select Groups	× SmogAnalyst	select	☐	×
Select Roles	Select Groups	× LocPolDept	Create select update	☐	×

Figure 25: Definition of an Access policy.

Apache Ranger can access to different security and privacy-oriented transformation tasks and trigger their execution.

Figure 26 shows where it is possible to define the Apache Ranger transformation actions. Note that custom transformations can be defined and added to this list.



Hive Column *	× class_plantaginaceae
Description	
Audit Logging	YES

Select Masking Option

- ( ) Redact
- ( ) Partial mask: show last 4
- ( ) Partial mask: show first 4
- (●) Hash
- ( ) Nullify
- ( ) Unmasked (retain original value)
- ( ) Date: show only year
- ( ) Custom

Mask Conditions :

Select Role	Select Group	Select User	Access Types	Se	
Select Roles	Select Groups	× LocPolDept	select	Hash	×

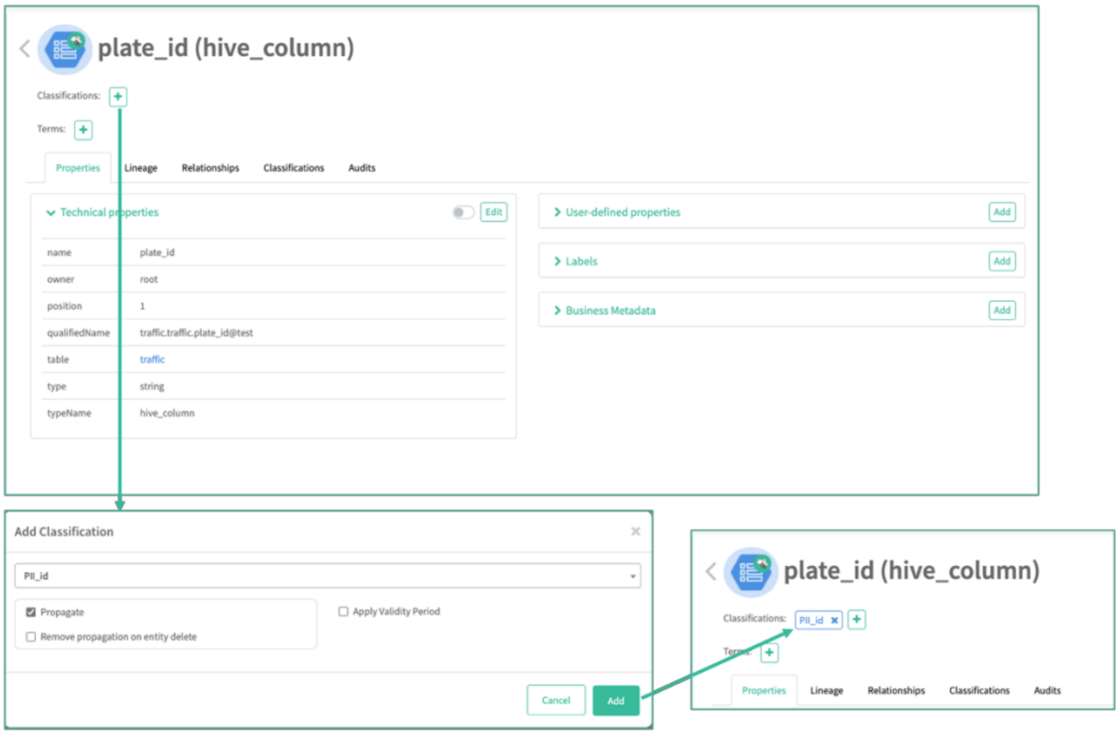
Figure 26: Apache Ranger transformation actions.

Practically speaking, Ranger is not capable to add a transformation task directly into the ingestion pipeline as needed, but it interacts with storage level technologies asking to execute the task prior to storing the outcomes of the ingestion pipeline.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

We note that all data coming from the ingestion pipeline (both batch and stream modes) are managed by the mechanisms in the area Data Governance and Control but not all undergo data sanitization techniques (e.g. data masking, hashing).

Apache Atlas is used to implement the *annotation* process by means of tags. Figure 27 shows the Atlas user interface suitable to define tags that are used during the data annotation phase. We note that Atlas internally uses Kafka to exchange messages using specific topics.


plate_id (hive_column)

Classifications: +

Terms: +

Properties | Lineage | Relationships | Classifications | Audits

Technical properties

name	plate_id
owner	root
position	1
qualifiedName	traffic.traffic.plate_id@test
table	traffic
type	string
typeName	hive_column

User-defined properties — Add

Labels — Add

Business Metadata — Add

Add Classification

PII_id

☑ Propagate

☐ Remove propagation on entity delete

☐ Apply Validity Period

Cancel | Add

plate_id (hive_column)

Classifications: PII_id ✖ +

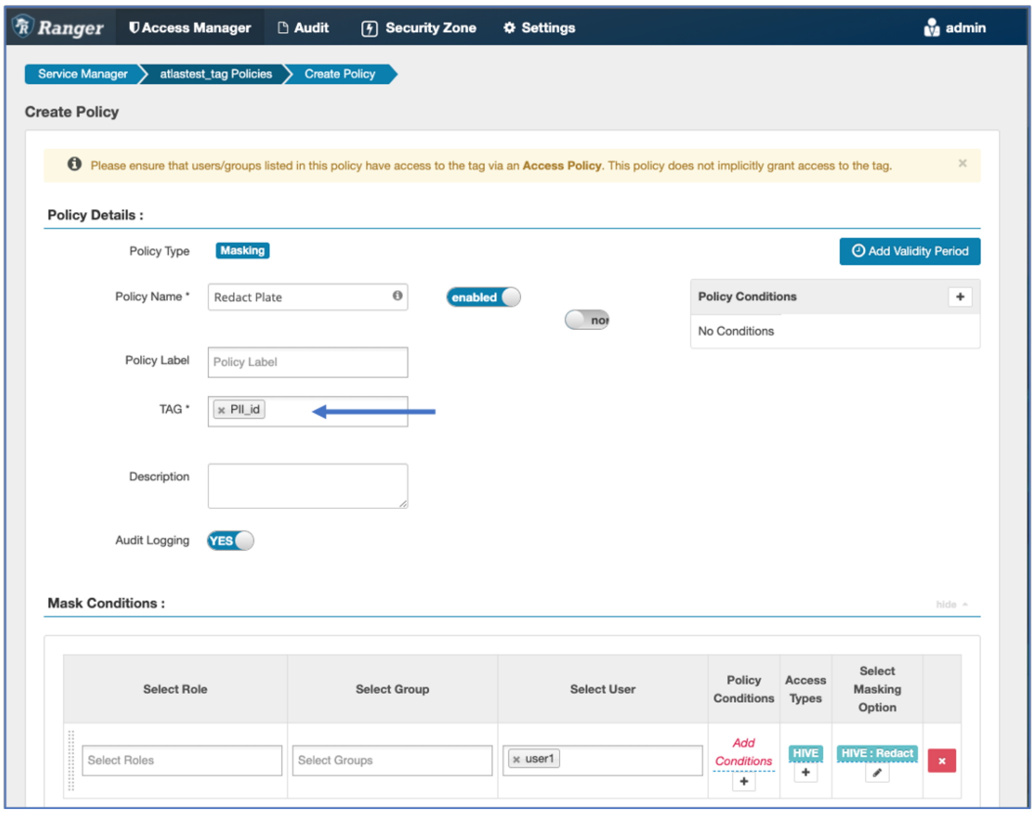
Terms: +

Properties | Lineage | Relationships | Classifications | Audits

Figure 27: Atlas tag definition.

The tags are then used in the framework of Apache Ranger tag-based policies and Apache Atlas notifies the relevant components about tags when needed.

Figure 28 shows a tag-based policy in Ranger using the tags defined in Figure 27. Atlas will also provide data lineage to improve the auditability of data flows.



Create Policy

Please ensure that users/groups listed in this policy have access to the tag via an **Access Policy**. This policy does not implicitly grant access to the tag.

Policy Details :

Policy Type: Masking

Policy Name *: Redact Plate — enabled

Policy Label: Policy Label

TAG *: × PII_id

Description:

Audit Logging: YES

Policy Conditions: No Conditions

Mask Conditions :

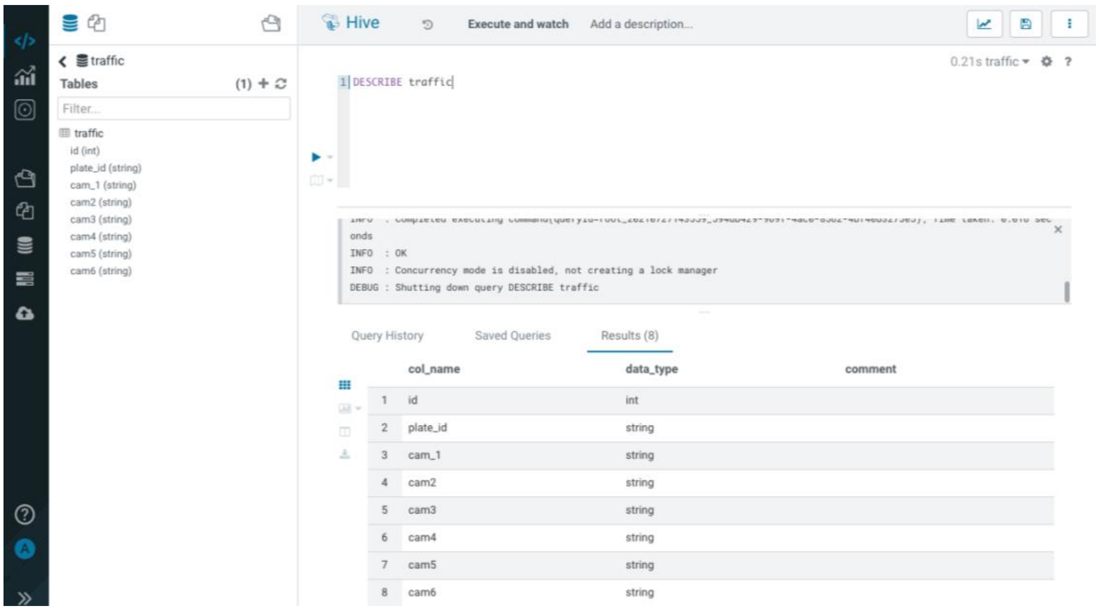
Select Role	Select Group	Select User	Policy Conditions	Access Types	Select Masking Option	
Select Roles	Select Groups	× user1	Add Conditions	HIVE	HIVE : Redact	×

Figure 28: Tag-based masking policy by using the tag defined in Figure 27

3.5.4 Area data storage

Once the data are treated by the "policy-aware ingestion pipeline" module, it can be stored in the storage components of the data lake (e.g., Hive or HDFS) and made available for the analytics and visualization pipelines. Technologies in area data storage are primarily focused on data availability, fault tolerance, and reliable distribution of data on the nodes of the cluster. They are indirectly involved in the processing procedure because nodes are normally involved in the processing of the local data, but when reduction is needed, prior to processing, the data should be redistributed or organized. This indiscernible relation between storage and processing is partially overcome by in memory processing of Spark, but still exists while workflows of different pipelines are considered. Figure 29 shows the Hive dashboard, where the schema of a structured data set (namely, *traffic*) is defined; Figure 30 shows the HDFS dashboard, where a folder is defined to contain unstructured data.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.



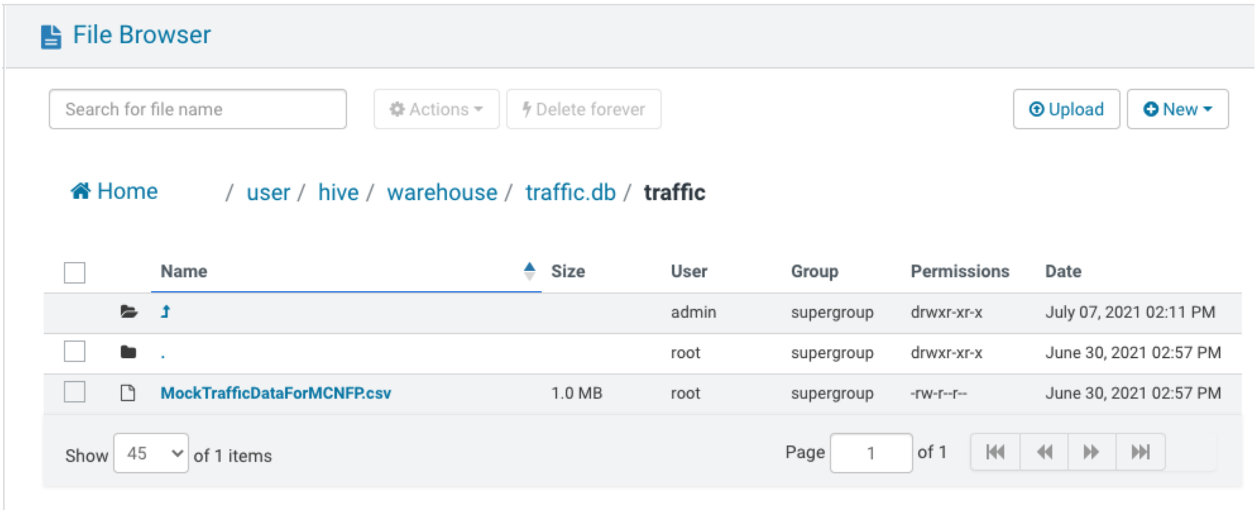
traffic
id (int)
plate_id (string)
cam_1 (string)
cam2 (string)
cam3 (string)
cam4 (string)
cam5 (string)
cam6 (string)

```
DESCRIBE traffic
```

```
INFO  : Completed executing command(queryId=[illegible]); Time taken: [illegible] sec
onds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
DEBUG : Shutting down query DESCRIBE traffic
```

	col_name	data_type	comment
1	id	int	
2	plate_id	string	
3	cam_1	string	
4	cam2	string	
5	cam3	string	
6	cam4	string	
7	cam5	string	
8	cam6	string	

Figure 29: Hive data set schema.



Home / user / hive / warehouse / traffic.db / traffic

Name	Size	User	Group	Permissions	Date
↥		admin	supergroup	drwxr-xr-x	July 07, 2021 02:11 PM
.		root	supergroup	drwxr-xr-x	June 30, 2021 02:57 PM
MockTrafficDataForMCNFP.csv	1.0 MB	root	supergroup	-rw-r--r--	June 30, 2021 02:57 PM

Show 45 of 1 items   Page 1 of 1

Figure 30: HDFS overview.

3.6 Walkthrough scenario

To see the full ingestion-time methodology at work, let us consider an extended version of the smart city scenario introduced in Section 3.4.2, where a *Smog Analysis* service is considered in addition to the *Traffic Monitoring* service regulating accesses to Low Emission Zones. Both services are based on the same data sources and ingestion pipeline. In the following, we first provide a description of the ingestion pipeline and then how our methodology applies to the two services. We also present some preliminary performance results.

3.6.1 Ingestion pipeline and policies

Services *Smog Analysis* and *Traffic Monitoring* share the same ingestion pipeline that includes video processing and data annotations already discussed in Sections 3.1 and 3.2. The outcome of this ingestion procedure is stored in the data lake of the city to be used by the two services independently. We note that, though the pipeline is the

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

same, it is executed by the two different services independently, thus providing the separation of duties characterizing our approach. Each service implements its own set of access control policies with different data transformations, leading to different enforcement results that are at the basis of our data governance approach.

The service *Traffic Monitoring* monitors accesses to low emission zones and issues a fine when an unauthorized access is registered by a vehicle not belonging to the set of authorized license plates (e.g., ambulances, residents, public transport). The service implements a simple visualization pipeline loading data after ingestion and showing vehicles metadata. Metadata are obtained using the plate number extracted from the video within a relative set of snapshots, with the scope of showing them to the service user (i.e., a member of the local police department). For this service, we consider two scenarios as follows.

- Normal scenario: authorized accesses are anonymized; unauthorized accesses are selected and the plate number is kept in clear to later issue a fine.
- Emergency scenario: the restrictions put in place in the normal scenario must be relaxed. A different policy applies without the application of the anonymization transformation, allowing service users to get all the details of vehicles that entered the Low Emission Zones during emergency. We note that the data lake contains plain data just for the emergency period, when the emergency is over the ordinary policies are restored.

In the following, we present the normal and emergency policies, respectively, as described by Apache Ranger with the form `<subject, action, object, context, data transformation>`:

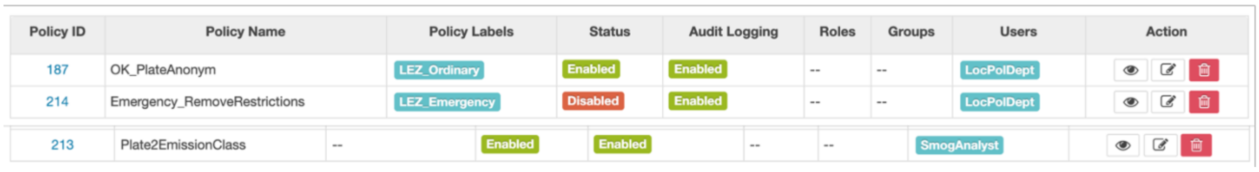
```
<LocPolDept, READ, Pii-id, {LEZ_Ordinary}, OK_PlateAnonym>

<LocPolDept, READ, Pii-id, {LEZ_Emergency}, Ø>
```

The service *Smog Analysis* combines information about air quality and traffic counting to create deeper insights about the connection between traffic regulation and air pollution. It uses the same ingestion pipeline, but the policy associated with the smog service is more restrictive in terms of privacy (e.g., video completely removed) and the transformation on the plate number is a replacement with the corresponding pollution class of the vehicle. In this case, the policy is of the form:

```
<SmogAnalyst, READ, Pii-id, -, Plate2EmissionClass>
```

Figure 31 shows the Apache Ranger Dashboard with the three defined policies.

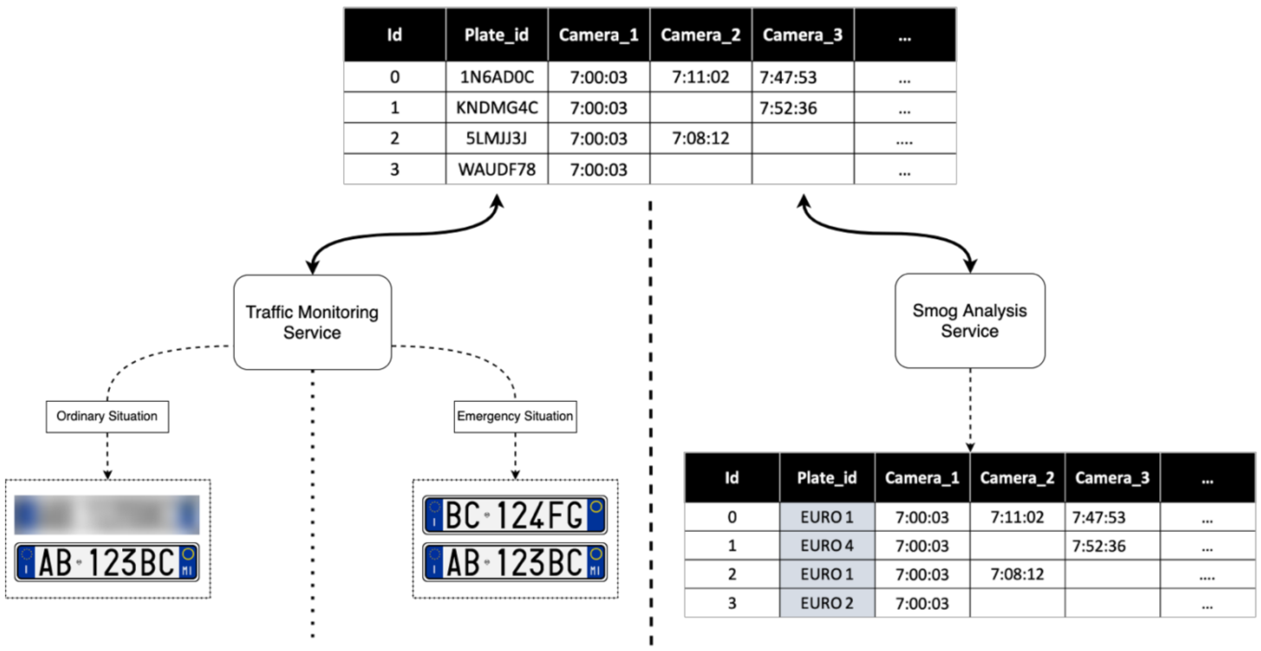


Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users	Action
187	OK_PlateAnonym	LEZ_Ordinary	Enabled	Enabled	--	--	LocPolDept	👁 ✎ 🗑
214	Emergency_RemoveRestrictions	LEZ_Emergency	Disabled	Enabled	--	--	LocPolDept	👁 ✎ 🗑
213	Plate2EmissionClass	--	Enabled	Enabled	--	--	SmogAnalyst	👁 ✎ 🗑

Figure 31: List of our policies in Apache Ranger.

3.6.2 Enforcement

Figure 32 shows the example with the two services in action and an excerpt of the ingested data. Both services can be implemented using an ad hoc end-to-end analytic approach; however, the level of data governance, as well as the flexibility provided, would not be comparable. With our solution, there is a common data governance based on an access control system, while in an ad hoc solution each service should be monitored for data governance compliance. In addition, a city-level data governance strategy can be applied to all the services or to the whole smart city ecosystem by simply changing or adding new policies, while in case of an ad hoc solution it requires to change every single service impacted by the new data governance strategy.



Id	Plate_id	Camera_1	Camera_2	Camera_3	...
0	1N6AD0C	7:00:03	7:11:02	7:47:53	...
1	KNDMG4C	7:00:03		7:52:36	...
2	5LMJJ3J	7:00:03	7:08:12		....
3	WAUDF78	7:00:03			...

Id	Plate_id	Camera_1	Camera_2	Camera_3	...
0	EURO 1	7:00:03	7:11:02	7:47:53	...
1	EURO 4	7:00:03		7:52:36	...
2	EURO 1	7:00:03	7:08:12		....
3	EURO 2	7:00:03			...

Figure 32: Data transformations for Traffic Monitoring and Smog Analysis services.

3.6.3 Performance considerations

We deployed the above system in our big data engine in Chapter 2 having one single node made of an Ubuntu 20.04.2 LTS machine with Intel Xeon E5-2620 - 2.095GHZ CPU and 32 GB of RAM. To evaluate the performance impact of our methodology, we used the Mock Traffic data set (https://www.kaggle.com/mathfour/mock-traffic-data) and extended it generating other entries to reach 20,000 records. We used the policies and services defined in this section and executed the ingestion procedure measuring the time needed to ingest data with and without policy enforcement. Figure 33 shows the performance results. We note that in the experiment without policies Atlas and Ranger were not fully disabled, therefore communication and notifications of incoming data between them were considered. The performance difference observed in Figure 33 represents just the policy enforcement time. The latter was not impacted by increasing the number of records due to the scaling capabilities of our architecture. We note that the enforcement time is reasonably negligible given the reduced set of policies and the limited effort requested by the transformations specified in the policies.

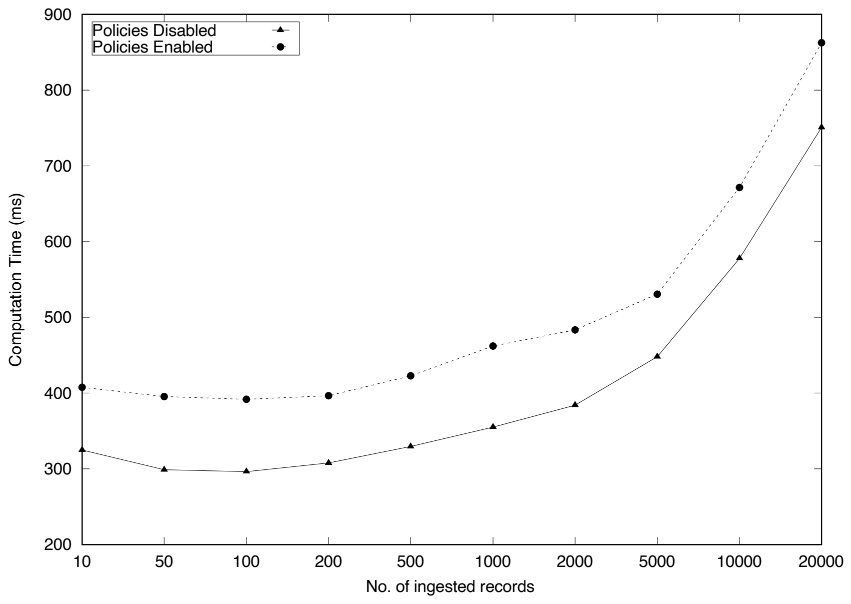


Figure 33: Comparing execution time with and without our policy enforcement.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

4 Data Analytics

This chapter presents an overview of the data analytics pipeline to be performed on data coming from the city pilots, that is, anomaly detection and event classification.

4.1 Anomaly detection

This section provides an overview of the anomaly detection pipeline.

Anomaly detection is a machine learning task which refers to the problem of identifying data that do not conform to patterns observed in historical data. These patterns represent the expected behaviour in normal conditions. Therefore, anomaly detection is usually performed through a data-driven algorithm to construct a model which will be able to detect a specific measurement/object/instance/observation as anomalous with respect to the historical data already seen. Anomaly detection is a very general task that finds applications in many real-domain scenarios such as fraud detection for credit cards, insurance or health care, intrusion detection for cyber-security, fault detection in safety critical systems, and military surveillance for enemy activities [19].

Since the task is based on machine learning algorithms, it is possible to visualize what happens geometrically to the considered observations under analysis. Figure 34 illustrates from a geometrical point of view how the anomalies could be represented in a 2-dimensional data set. Specifically, there are two normal regions, N_1 and N_2, since most observations lie in these regions. The points that are sufficiently far away from the regions, e.g., points o_1 and o_2, and points in region O_3, are considered as anomalies.

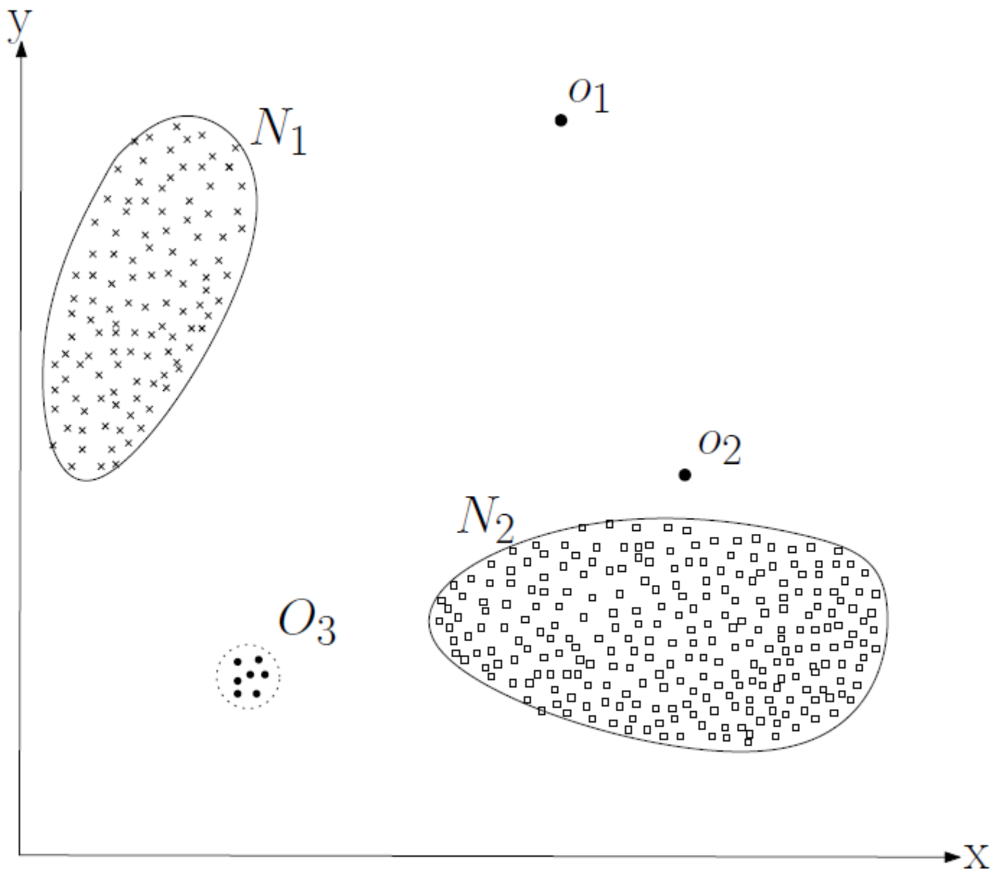


Figure 34: Anomalies in a 2-dimensional data set [19].

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

Anomaly detection is usually performed by training a model (henceforth anomaly detector) that is capable of catching anomalies from data. We identified three possible phases to train an accurate anomaly detector: the i) initial phase, ii) update phase, and iii) identification phase.

At the initial phase, a weak predictive model is constructed, that represents a “coin-flip” function with low predictive power, that needs to be trained to improve the performance. To this aim, at this phase, a batch-learning approach is considered to overcome the weakness of the starting function. The algorithm in this phase is at the "initial state".

After the first stage, the anomaly detector performs better than the previous starting function and it could be ready to take the sensor data as input to identify possible anomalies. However, since the distributions could vary also in normal cases (e.g., during the weekend CO2 in the air could be lower than during working days), an additional update phase was introduced, that aims to enable further training of the anomaly detector with the possibility to catch also non-anomalous variations in distributions. Furthermore, the update phase helps also to use a previous pre-trained model with possible reduction of the training time. The algorithm in this phase is at the "update state".

In the identification phase, the anomaly detector is ready to process sensor data to detect possible anomalies. At this phase, the anomaly detector could be placed in production to work actively with the real scenario data, since it shows better performance. The algorithm in this phase is at the "identification state". A general schema of the anomaly detection is shown in Figure 35.

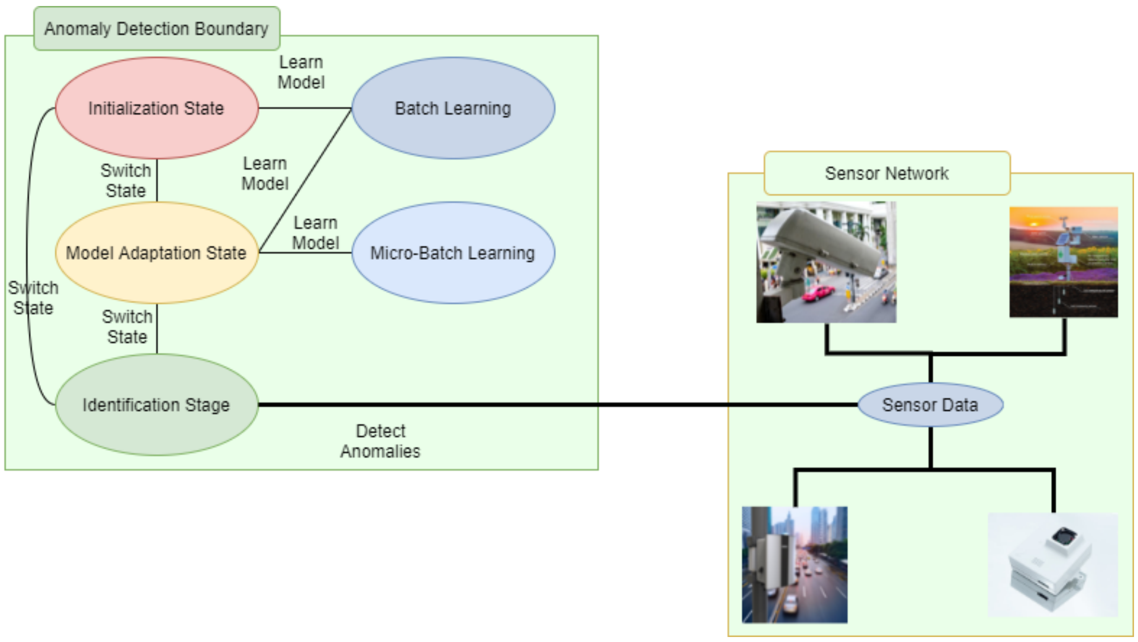


Figure 35: Anomaly detection for the identification of possible anomalies from sensor data.

The anomaly detector can handle all the possible sensor data that presents spatial (e.g., GPS coordinates) and temporal information (e.g., timestamp) and a set of descriptive variables that are acquired by the specific sensor for the monitoring of the city. For instance, independently from the type of the sensor (e.g., traffic cameras, air pollution), the anomaly detector acts with the same approach. Indeed, the anomaly detector works with values usually indicating the level of something: pedestrians’ concentration, traffic level, temperature, humidity, level of PM2.5, level of CO2, and so forth, that are automatically captured and transmitted by the sensor network. Therefore, in the real scenario, the anomaly detector will analyse the data coming from different sensors and it will be able to judge the data as a normal or anomalous case.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

4.1.1 Self-Organizing Map (SOM)

Self-Organizing Maps (SOMs) [20] are a particular type of Artificial Neural Network (ANN) that are trained using an unsupervised learning, to produce a low-dimensional (generally a two-dimensional one) representation of the input space of a data collection, also referred to as a map. For this reason, Self-Organizing Map is also considered a dimensionality reduction method. As a result, SOMs make it easier to analyse and visualize high-dimensional data. A peculiarity of SOMs is that the produced map allows to preserve the topological properties of the data, with the use of a neighbourhood function. What mainly distinguishes SOMs from ANNs is the fact that SOMs use competitive learning, as opposed to the error-correction learning, such as the backpropagation method.

The typical structure of a SOM is shown in Figure 36. The input layer consists of a set of input vectors, i.e., the observations used to train the map. The feature map is composed of a grid of nodes (called neurons), with lateral connections. Grids used by a SOM are generally characterized by a low dimensionality, typically two or three dimensions. In the feature map, each input vector is connected to all neurons, while each neuron in the map is associated with a weight vector, having the same size as the input vectors. Consequently, the neuron can be considered an element belonging to the input space. However, while the input vectors are fixed points within the map space, the neurons, during the training phase, will tend to move closer to the input vectors, preserving the topology induced by the map space.

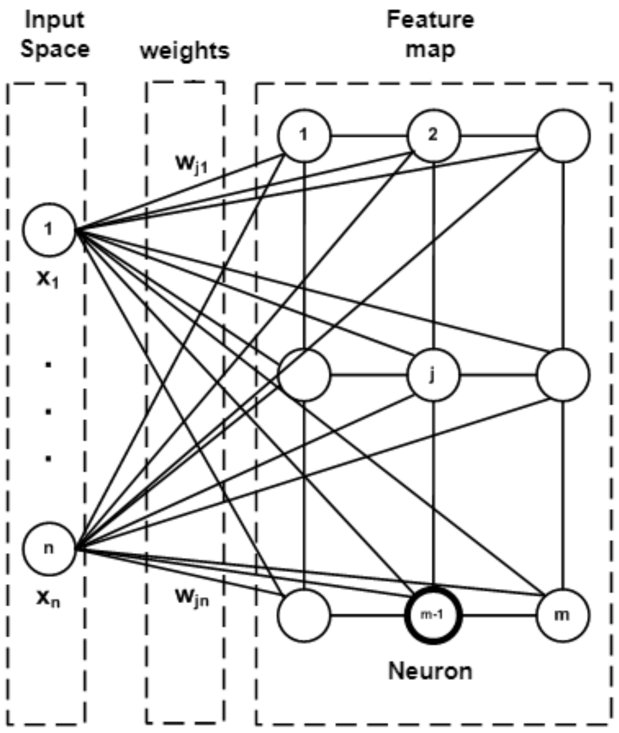


Figure 36: A SOM architecture with its input space, connection weights, and feature map [21].

The training process of a SOM is very reminiscent of the information organization processes of the human brain, in particular in the cerebral cortex, in which different sensory information (e.g., visual and auditory) is handled in separate parts of the same. The following illustrates the process of training of a Self-organizing map.

Let D be a data collection composed of n input vectors, defined as follows:

$$D = \{d_1, d_2, \dots, d_n\}$$

and let W be the set of the weight vectors, each of which is associated with a neuron in the feature map.

$$W = \{w_1, w_2, \dots, w_n\}$$

Weight vectors are initially initialised at small random numbers. An input vector from D is then acquired, and the Euclidean distance between the input vector and all the neurons is calculated, in order to find the most similar neuron to the input vector. This particular neuron is called the Best Matching Unit (BMU). Subsequently, the BMU and all its nearest neurons are brought closer to the input vector. The size and structure of the neighbourhood depend on the specific neighbourhood function used, denoted by N_f. The procedure is repeated for each input vector belonging to D, for a determined number of iterations t.

The equation for updating a neuron i, with a weight vector w_i, at time $t+1$, is given below.

$$w_i(t+1) = w(t) + \alpha(t) \cdot N_f(t) \cdot \left(d_k - w(t)\right)$$

where $N_f(t)$ represent respectively the neighborhood function at time t, $\alpha(t)$ is a scalar factor that defines the size of the correction, while d_k is an input vector belonging to D.

In the mapping phase, Euclidean distance between the new input vector and weight vectors is computed, in order to find the closest neuron, which could be used to classify the new data.

4.1.2 Growing Hierarchical SOM (GH-SOM)

In recent years, different variants of the SOM architecture have been proposed, one of the most popular is the Growing Hierarchical Self-Organizing Map (GH-SOM) [22]. This architecture overcomes some of the limitations of the SOM architecture. First, the SOM consists of a fixed feature map, where the number of neurons must be defined before the training phase. Second, hierarchical relationships between the input data are not clearly distinguishable in a SOM, within the map space. For these reasons, the structure of a GH-SOM dynamically grows into a hierarchy, according to the input data used during the training process.

GH-SOM introduced a metric called Mean Quantization Error (MQE). The MQE of a neuron defines the deviation of the neuron with respect to the input data that chose it as a BMU. The MQE of a SOM can thus be defined as the average MQE of all the neurons in the feature map.

During the training of a GH-SOM, the MQE of a neuron m_0 at level-0 is computed (mqe_0), with respect to all the input vectors $x_{(\cdot,i)}$, as defined below.

$$mqe_0 = \frac{1}{n} \sum_{x_{(\cdot,i)} \in C_n} \left\| m_0 - x_{(\cdot,i)} \right\|$$

where C_n is the set of the input vectors.

The first SOM is created at level-1 of the hierarchy, generally consisting of a 2×2 feature map. This first SOM is trained using the classical procedure for training a SOM. After the training, the MQE for the SOM (MQE_m) is calculated. A high value for MQE_m means that the map m is not able to map the input space well and, for this reason, more neurons are required, in order to better approximate the input space. This condition is governed by the τ_1 criterion:

$$MQE_m < \tau_1 \cdot mqe_p \text{ , with } 0 \leq \tau_1 \leq 1$$

where mqe_p defines the MQE of the parent neuron that wants to expand the map m. At this point, the map starts to grow, until the τ_1 criterion is satisfied. In order to grow, the neuron with the highest MQE is identified in the map, referred to as the error neuron e. After that, its most dissimilar connected neighbor is selected and a new column or a new row is inserted into the feature map, allowing new neurons to be added to the map. The new weights vectors are initialized with the average of the weight vectors of their adjacent neuron neighbours.

Figure 37 shows the process of insertion of a new row or column in a SOM architecture.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

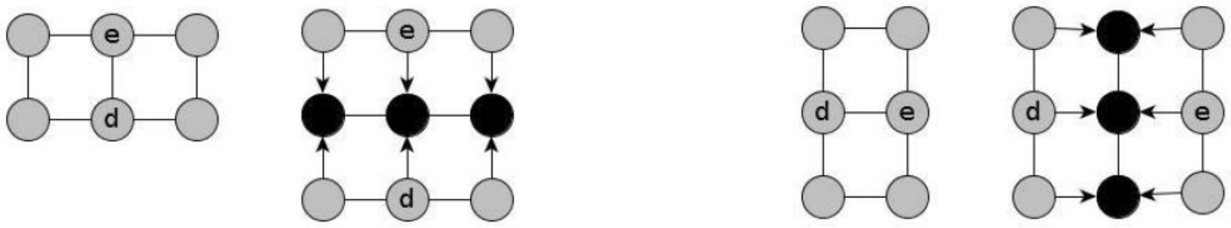


Figure 37: Insertion of a new row (left) or a new column (right) in a SOM architecture [23].

This procedure produces an update SOM, that is trained and analyzed, according to the τ_1 criterion. The process of growth and training continue, until the τ_1 condition is satisfied. When the τ_1 condition is satisfied, neurons in the map are analyzed, according to the τ_2 criterion:

$$mqe_k < \tau_2 \cdot mqe_0 \text{ , with } 0 \leq \tau_2 \leq 1$$

Neurons that do not satisfy the τ_2 condition are expanded into new SOMs, to the next level of the hierarchy that is being generated. The new SOMs repeat the same process of training, growing and hierarchical expansion of the first SOM, previously defined at the level-1 of the hierarchy. When all the neurons in the lowest level of the hierarchy satisfy the τ_2 criterion, the training of a GH-SOM is completed. At the end, the GH-SOM takes on a hierarchical structure, consisting of multiple SOMs organized in different hierarchical levels, such that each SOM represents the input data at a finer granularity with respect to the parent SOM.

The structure of a GH-SOM, at the end of the training process, is presented in Figure 38.

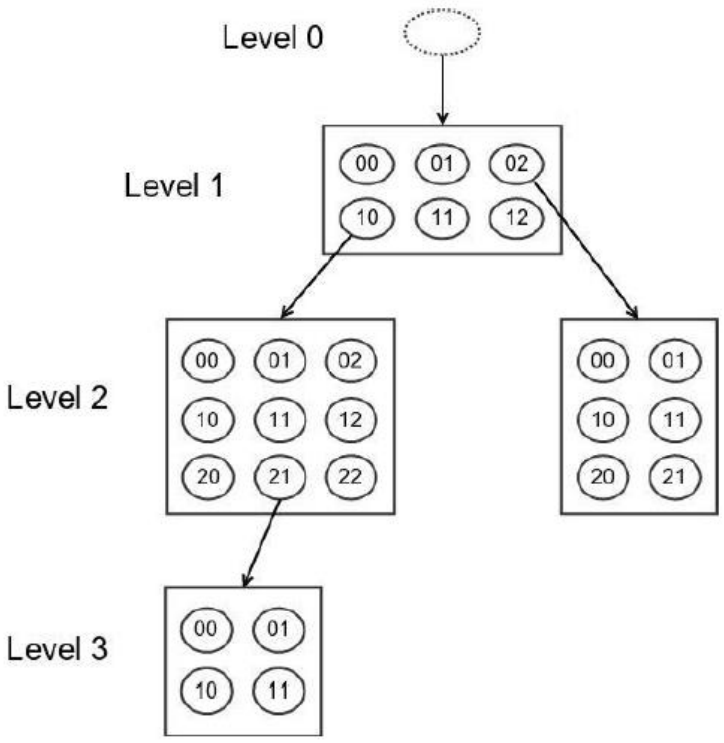


Figure 38: Structure of a GH-SOM after the training process [23].

4.1.3 Spark-GHSOM

This section presents the tool used for the anomaly detection pipeline.

The algorithm proposed in [23] is called Spark-GHSOM, that is able to overcome the classical GH-SOM issues. Specifically, GH-SOM requires multiple iterations over the input data set, making it intractable on large data sets. Furthermore, the conventional GH-SOM algorithm is capable of handling only numeric attributes. These limitations impact most of the modern real-world data sets that are characterized by mixed numerical and categorical attributes. Spark-GHSOM extends GH-SOM by exploiting the Spark platform to process massive data sets in a distributed manner. Moreover, Spark-GHSOM leverages the distance hierarchy approach to improve the optimization function of GH-SOM, in order to handle mixed-attribute data sets. Spark-GHSOM was tested with respect to accuracy, scalability and descriptive power and the results demonstrate the superior predictive and descriptive capabilities and applicability to large-scale data sets. In the project scope, in order to cope with the anomaly detection task, Spark-GHSOM has been extended, in order to perform anomaly detection that could be able to produce interpretable output easily readable also for the SOC operators and the end users.

The first step in Spark-GHSOM is to replace the MQE function, used during the training of a GH-SOM, with a dissimilarity measure that is able to handle both numerical and categorical attributes. To this scope, variance was chosen to solve this issue:

$$var_0 = \sum_{l=1}^{L} \left(1^{numerical(l)} \cdot var(l) + 1^{categorical(l)} \cdot \frac{u_2(l)}{2} \right)$$

where L is the total number of the input attributes, $1^{numerical(l)}$ is equal to 1 if the attribute l is numerical, 0 otherwise. The same applies for $1^{categorical(l)}$, in the case of a categorical attribute l.

The coefficient of unalikeability $u_2(l)$, for a categorical attribute l, is defined as:

$$u_2(l) = \sum_{i \in domain(l)} p_i(1 - p_i)$$

and p_i is defined as follows:

$$p_i = \frac{frequency(l_i, C_n)}{n}$$

where l_i is the i-th value of the attribute l and $frequency(l_i, C_n)$ is the absolute frequency of l_i, for the attribute l in C_n, the set of all the n input instances.

The training process of the Spark-GHSOM follows the classical process of the GH-SOM described in the previous paragraph, except for the use of a different function for the calculation of the distance between the input vector and the neurons of the feature map, since the Euclidean distance is not computable on categorical attributes. For this reason, the hierarchical distance was chosen.

The hierarchy obtained can thus be used to solve an anomaly detection task. In particular, when a new input vector is supplied to the hierarchy, the tool looks for the SOM that succeeds in better approximating the input data (that is, the SOM with the shortest distance with respect to the input vector). Once found, it is used to carry out the prediction for the new input data, based on the distance between the input vector and the neurons in the map.

As data distributions tend to change over time, it may be necessary to update the knowledge of the anomaly detector using more recent data. For this reason, Spark-GHSOM for anomaly detection provides the possibility to update the weights vectors of the neurons while keeping the generated hierarchy unchanged. This process can be particularly useful if end users do not have enough time or data availability to train a new anomaly detector from scratch. Consequently, having a pre-trained model already available, it is possible to provide the model with a micro-batch of data, in order to update the knowledge extracted by the model and adapt it to the user's needs.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

4.1.4 Interpretability

This section provides a description of how the output of Spark-GHSOM has been made interpretable and its advantages.

The anomaly detector could produce different types of output depending on the level of detail. The simplest approach provides feedback for the current data in the form of a Boolean response. This kind of output could support raising an alert if the response is equal to "anomaly" (see Figure 39 as an example).

The described approach has the advantage that is simple to handle and transmits the prediction as a binary variable (e.g., anomaly/normal, 0/1, true/false). However, the drawback of this approach is that it makes difficult for the end user interpreting the raised alert/anomaly. Therefore, a more informative approach could be considered by combining the previous one with a ranking of the variables with their importance, indicating the contribution to catch the anomaly (see Figure 40).

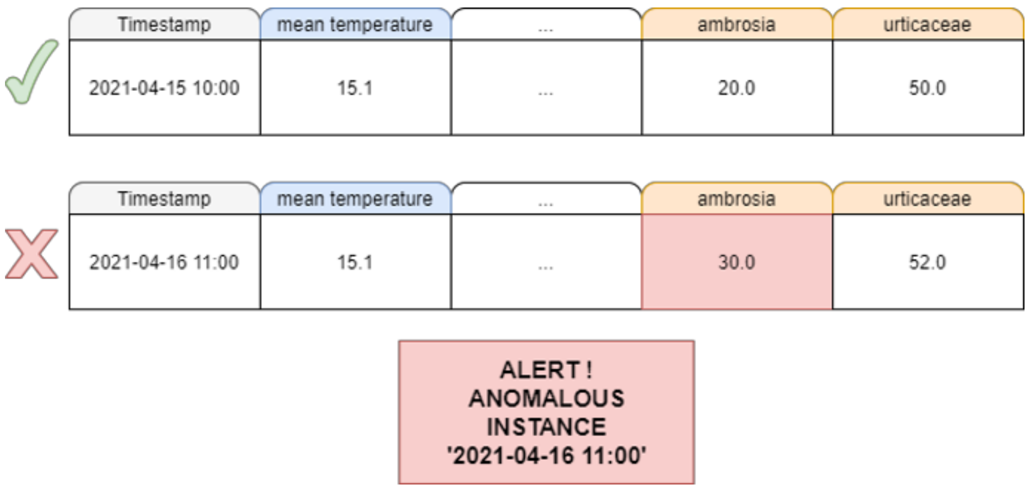


Figure 39: When an anomaly will be detected, the system will raise an alert indicating the timestamp and GPS coordinates.

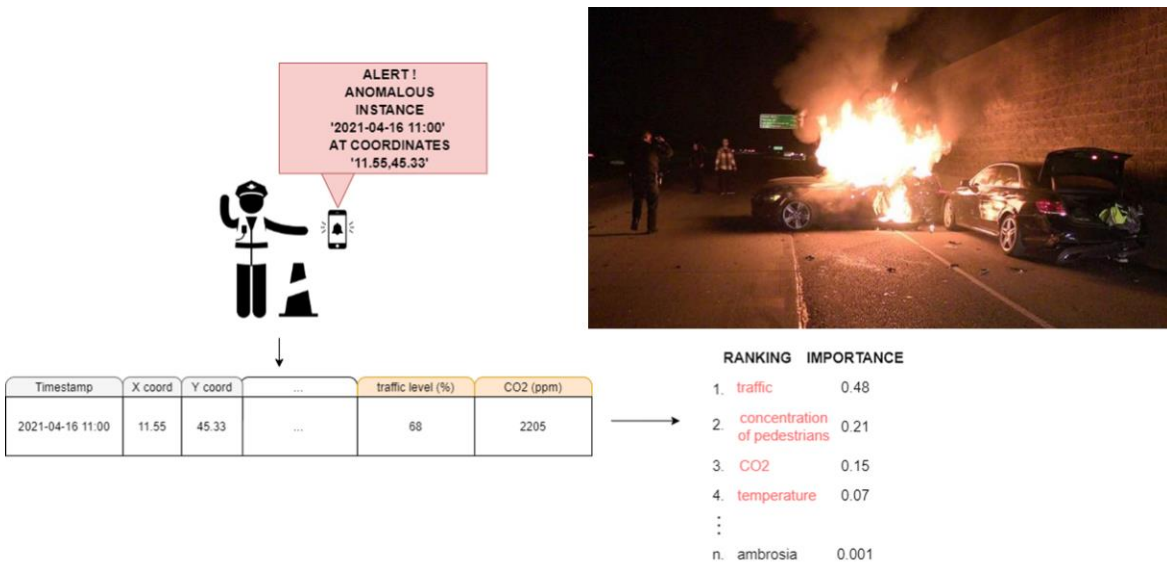


Figure 40: When an anomaly will be detected, the system will raise an alert and will provide a feature ranking according to the features importance in detecting the anomaly.

Feature ranking is a ranking of the entire features set, of which the data collection is composed, ordered with respect to the feature importance, each associated with a particular feature.

Feature importance is a numerical value between 0 and 1, which expresses how anomalous the value expressed by the feature is with respect to the data collection, such that the sum of all the features importance in the feature ranking is equal to 1.

The importance score is determined starting from a distance function between the current data under analysis and the anomaly detector model. The distance function is impacted by a predefined threshold (called threshold factor, that helps to control the sensibility of the algorithm). An anomaly will be detected if that distance is too high according to the previous data collection already seen for training. A graphical example is shown in Figure 41. This approach helps to better understand if the set of current measurements represent an anomaly. In this way, the operator would be able to interpret if the current scenario is representing a threat.

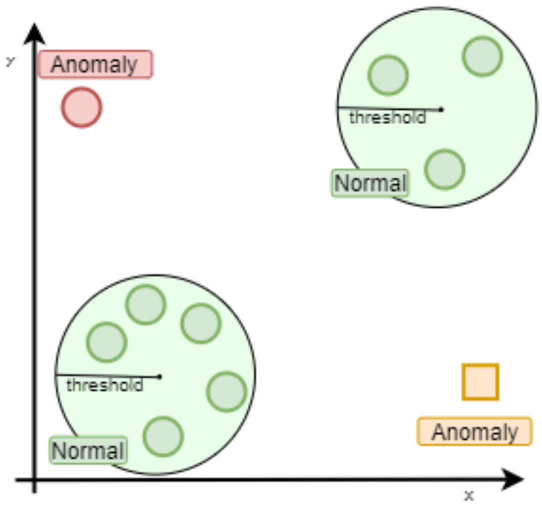
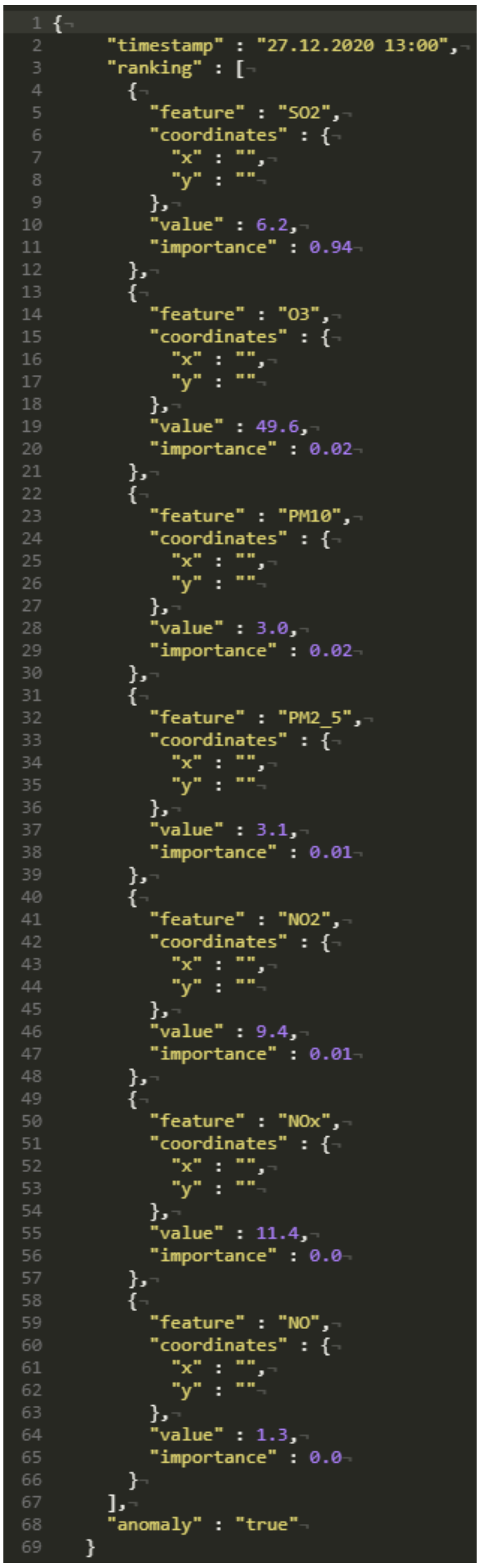


Figure 41: An example with two variables of analysis that correspond to two sensor measurements.

Consequently, for each new instance to be classified, the tool will provide the following output:

- Identification (i.e., normal or anomalous case).
- GPS coordinates of the sensors involved, if the observation is considered abnormal from the tool.
- Feature ranking of the instance.

An example of the output format from the anomaly detector, on atmospheric particulate data from the city of Oslo, is provided in Figure 42.



```
{
    "timestamp" : "27.12.2020 13:00",
    "ranking" : [
      {
        "feature" : "SO2",
        "coordinates" : {
          "x" : "",
          "y" : ""
        },
        "value" : 6.2,
        "importance" : 0.94
      },
      {
        "feature" : "O3",
        "coordinates" : {
          "x" : "",
          "y" : ""
        },
        "value" : 49.6,
        "importance" : 0.02
      },
      {
        "feature" : "PM10",
        "coordinates" : {
          "x" : "",
          "y" : ""
        },
        "value" : 3.0,
        "importance" : 0.02
      },
      {
        "feature" : "PM2_5",
        "coordinates" : {
          "x" : "",
          "y" : ""
        },
        "value" : 3.1,
        "importance" : 0.01
      },
      {
        "feature" : "NO2",
        "coordinates" : {
          "x" : "",
          "y" : ""
        },
        "value" : 9.4,
        "importance" : 0.01
      },
      {
        "feature" : "NOx",
        "coordinates" : {
          "x" : "",
          "y" : ""
        },
        "value" : 11.4,
        "importance" : 0.0
      },
      {
        "feature" : "NO",
        "coordinates" : {
          "x" : "",
          "y" : ""
        },
        "value" : 1.3,
        "importance" : 0.0
      }
    ],
    "anomaly" : "true"
  }
```

Figure 42: Output format of the anomaly detector.

4.1.5 Case studies

This section presents two different case studies, using data coming from the pilot cities of Padua and Oslo, respectively. Each case study was conducted using the supervised and unsupervised anomaly detection approaches. In a supervised anomaly detection task, the anomaly detector is trained only on labelled data, using the normal cases, and is evaluated on unseen data. In an unsupervised anomaly detection task instead, data are not labelled. For this reason, the anomaly detector is trained making an important assumption, that most of the data in the training set are normal cases.

That said, it is possible to infer that in the case of unlabelled data, it is necessary to use an unsupervised approach, while in the case of labelled data, both described approaches could be valid. However, even if having a greater flexibility, the main disadvantage of the unsupervised approach lies in the difficulty in performing a quantitative analysis, due to the lack of data labelling.

Padua – Data set description

The data extracted contain daily environmental information from the city of Padua, referring to the period that goes from 2014 to 2019, collected as batch. The data set consists of a sample per day, excluding those days when measurements were not present. Data were extracted from the ARPA Veneto website [24], Open Data section, where ARPA is an Italian regional agency that carries out environmental control activities, such as the analysis and measurement of air and water quality.

The information extracted includes the following measurements:

- Date of measurement (dd/mm/yyyy).
- Air temperature at 2 metres (°C), respectively mean, min and max temperature.
- Total precipitation on the day (mm).
- Humidity at 2 metres (%), respectively min and max humidity.
- Concentrations of allergenic pollen in the air (granules/m^3) per family, for a total of 26 families.

Figure 43Figure 43 shows the distribution of the meteorological information collected in the data set.

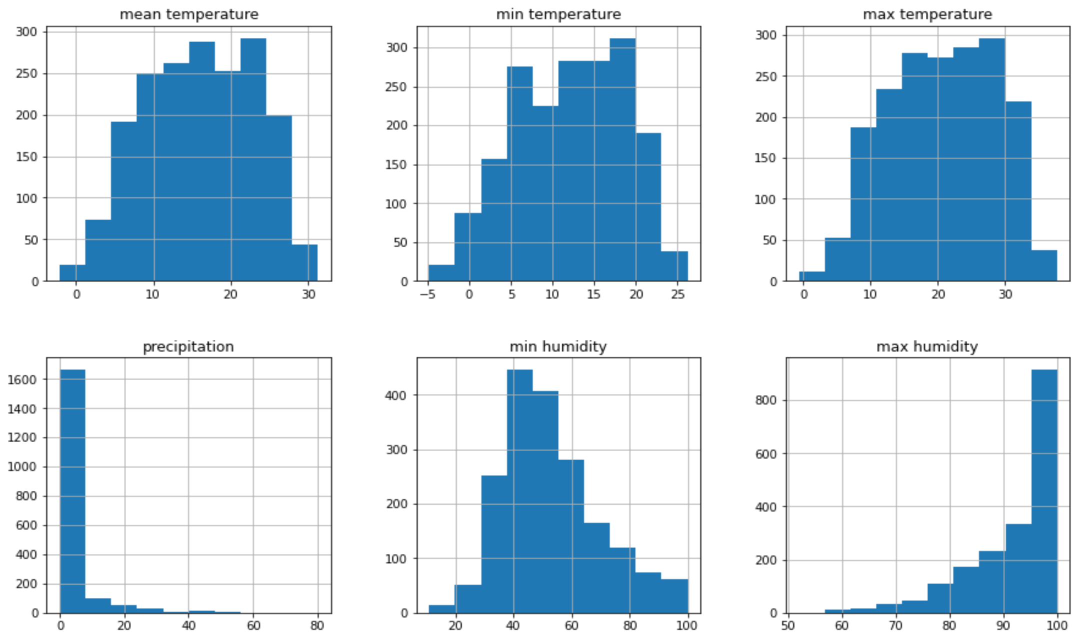


Figure 43: Data distributions for the attributes mean, min and max temperature, precipitation, min and max humidity.

From the charts is clearly visible that the temperature attributes, as well as the min humidity, follow a normal distribution, while distributions of precipitation and max humidity are asymmetrical.

Padua – Supervised Anomaly Detection

The data set presented in the previous section is clearly unlabelled, as it is not possible to determine with certainty whether a given observation belonging to the data set is anomalous or not. For this reason, the focus was on determining whether a given attribute belonging to the data set had some anomalous values or not. The choice fell on the ambrosia pollen, which is a very common pollen and above all aggressive, as even small amounts can trigger strong allergic reactions. Furthermore, as shown in Figure 44, high ambrosia concentrations occurred only in rare cases in the period considered, so that the "high" class could be considered as the "anomaly" class and the remaining classes could be aggregated into the "normal" class.

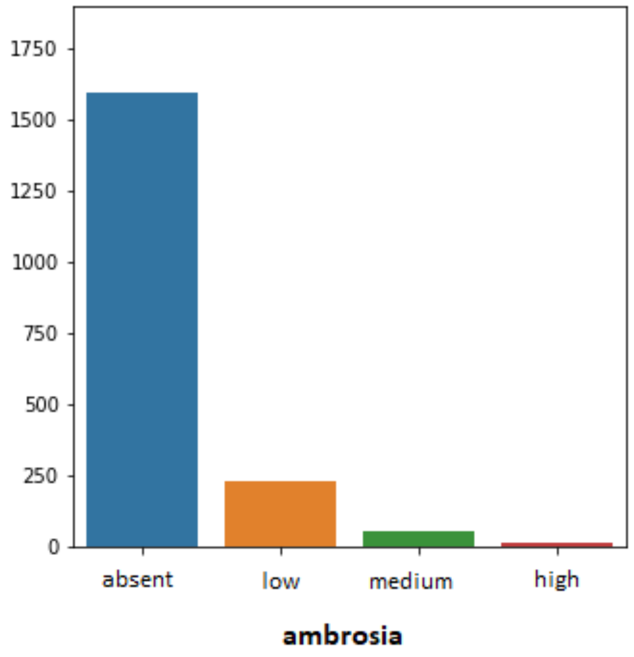


Figure 44: Ragweed pollen concentrations over the specified period, grouped into four concentration classes.

Table 6 illustrates the results of the conducted experiment, where the τ_1 and τ_2 parameters were set to 0.7 and 1.0, respectively. A description of the columns in the table is provided below:

Input parameters

- Epochs: number of passes of the entire training data set the algorithm has completed.
- Threshold factor: parameter governing the distance within which an instance is considered an anomaly.

Output parameters

- True Normal (TN): number of correctly classified instances of class "normal".
- False Anomaly (FA): number of instances of class "normal" that are classified as "anomaly".
- False Normal (FN): number of instances of class "anomaly" that are classified as "normal".
- True Anomaly (TA): number of correctly classified instances of class "anomaly".

As can be guessed, the main objective of an anomaly detector is to minimise the number of False Normal and False Anomaly as well as maximise the number of True Normal and True Anomaly. Between False Normal and False Anomaly, we give higher priority to minimizing False Normal cases, which may represent the most dangerous situation.

Table 6: Results of the Padua supervised anomaly detection experiment.

epochs	thresh. factor	true normal	false anomaly	false normal	true anomaly
10	6.0	371	21	2	1
10	5.5	369	23	1	2
10	5.0	363	29	1	2
10	4.5	355	37	0	3
10	4.0	342	50	0	3
10	3.5	322	70	0	3
15	6.0	385	7	3	0
15	5.5	384	8	1	2
15	5.0	378	14	1	2
15	4.5	374	18	0	3
15	4.0	364	28	0	3
15	3.5	352	40	0	3

As Table 6 shows, many of the configurations adopted were able to correctly detect all the anomalies present within the test set. In particular, the best performing configuration uses 15 epochs and a threshold factor equal to 4.5. This configuration, as well as correctly identifying all the anomalies, allows the lowest number of false anomalies to be returned, equal to 18.

From the experimental results, the following conclusions can be drawn:

- The decrease in the threshold factor allows more anomalies to be correctly classified, however, as expected, the number of false anomalies (also known as “false alarms”) also increases.
- Increasing the number of epochs reduces the number of false anomalies identified, thereby improving the overall performance.

Padua – Unsupervised Anomaly Detection

Unlike the previous experiment, in unsupervised anomaly detection is very complex to carry out a quantitative analysis. For this reason, in this experiment, it is preferable to use a qualitative approach, trying to draw some conclusions from the feature ranking returned by the tool.

The first step is to find, for each feature, if possible, reference tables containing concentration classes. An example of a reference table for the precipitation attribute is provided in Table 7.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

Table 7: Concentration classes for the precipitation attribute.

rainfall intensity	mm/6h	mm/12h	mm/24h
weak	0 - 5	0 - 10	0 - 15
moderate	5 - 15	10 - 30	15 - 45
strong	15 - 30	30 - 60	45 - 90
very strong	> 30	> 60	> 90

Each class in the reference table is associated with an "anomaly weight" in the range $[0,1]$, where values 0 and 1 are associated with the least and most dangerous classes, respectively. Table 8 shows the anomaly weights for the precipitation attribute, associated with each class in the reference table.

Table 8: Anomaly weight for each concentration class of the precipitation attribute.

rainfall intensity	mm/6h	mm/12h	mm/24h
weak	0.00	0.00	0.00
moderate	0.33	0.33	0.33
strong	0.66	0.66	0.66
very strong	0.99	0.99	0.99

Given k, the k-th position in the feature ranking, the anomaly quantifier at the k-th position is defined as follows:

$$anomaly\ quantifier@k = \sum_{i=1}^{k} w_i$$

where w_i is the anomaly weight of the attribute in the i-th position in the feature ranking, with $i \leq k$, if the reference table of the attribute in the i-th position is defined. Thus, using the anomaly quantifier, it is possible to identify instances containing more dangerous values than the remaining observations.

By keeping track of the value assumed by the anomaly quantifier at each position k of the feature ranking, it is possible to graphically represent the trend of the anomaly quantifier. For instances identified as anomalous by the tool, we expect to see curves with a high slope in the first positions of the feature ranking that gradually stabilize, until they no longer grow.

Figure 45 shows the trend of the anomaly quantifier, for each instance identified as anomalous by the tool. From a quantitative point of view, it is possible to calculate the area under the curve (AUC) for each curve in the graph, using the composite trapezoidal rule. The trapezoidal rule is a technique for approximating the definite integral of a function $f(x)$, by approximating the region under the graph of the function as a trapezoid and calculating its area. Figure 46 illustrates how the composite trapezoidal rule works.

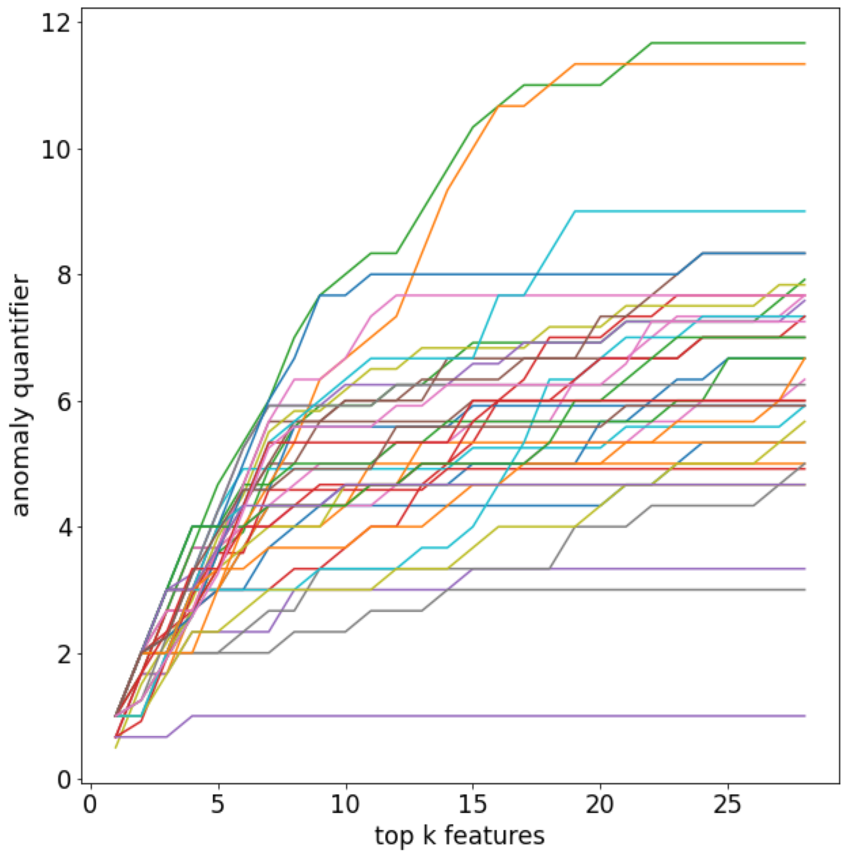


Figure 45: Anomaly quantifier value for each position in the feature ranking, for instances identified as anomalous by the tool.

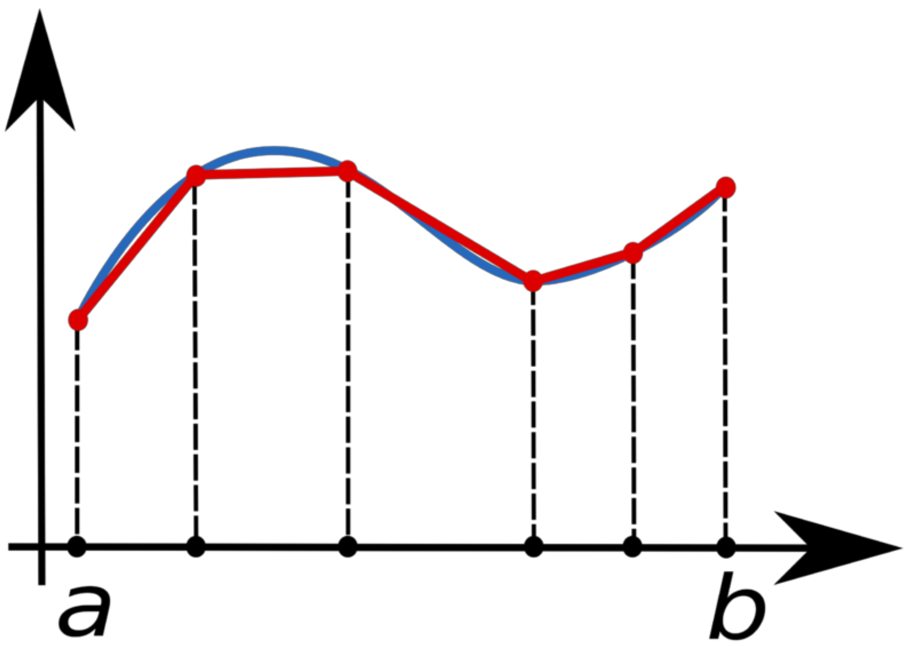


Figure 46: Composite trapezoidal rule.

To follow the procedure described in the previous paragraph, it was first necessary to normalize the x-axis and y-axis in the interval [0,1], then the AUC of each curve was calculated by considering as points the values of the anomaly quantifier at each k-th position in the feature ranking. The AUC mean was 0.41, with a standard deviation of 0.13, respectively. The AUC mean, therefore, lies slightly below the AUC of the bisector of the first and third quadrants, which in the interval [0,1] is characterized by an AUC of 0.5.

Let us consider the curve characterized by the highest AUC, which identifies the day on which more dangerous values occurred than on the remaining days, considering the descriptive features involved. The day identified is the 4 April 2014, as shown in Figure 47.

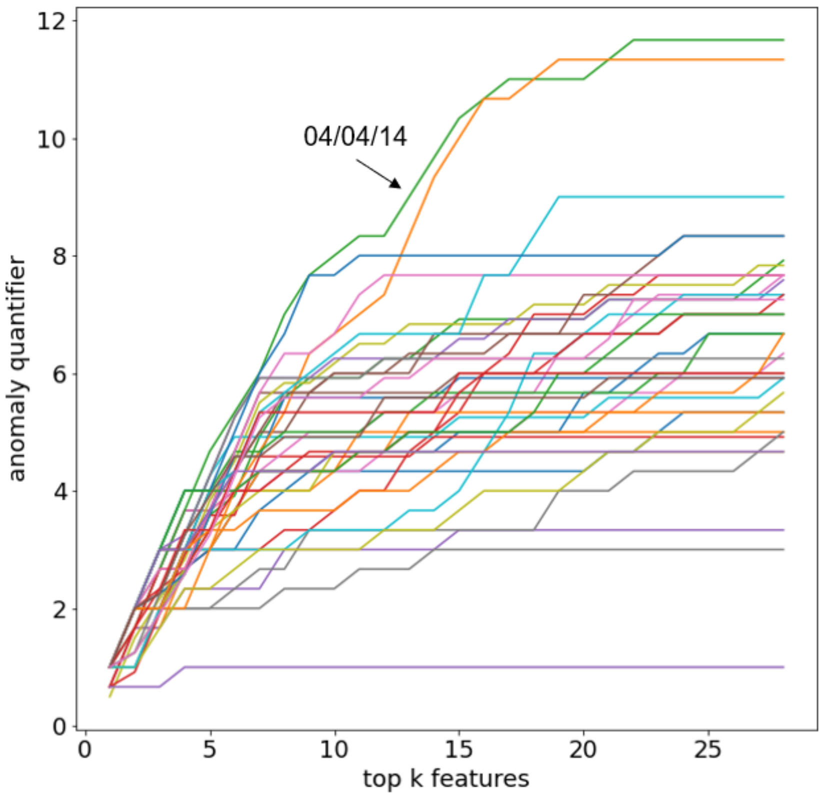


Figure 47: Anomaly quantifier curve of 4 April 2014.

The objective is to understand whether the feature ranking returned by the tool is consistent with the dangerous values recorded on that day.

The top 10 feature ranking for the identified day is provided below, together with the anomaly weights:

1. Urticaceae [1.00]
2. Salix [0.67]
3. Cupressaceae / Taxaceae [1.00]
4. Fagaceae [1.00]
5. Quercus [1.00]
6. Pinaceae [0.67]
7. Corylaceae [0.67]
8. Platanaceae [1.00]
9. Betulaceae [0.67]
10. Fagus Sylvatica [0.33]

The features listed in the top positions of the feature ranking describe pollen families, which should not be surprising as high pollen concentrations mainly occur during the spring period.

Figures Figure 48, Figure 49, Figure 50, Figure 51 show respectively the annual concentrations of pollens Urticaceae, Cupressaceae / Taxaceae, Corylaceae, and Betulaceae, relative to the year 2014. These graphs have been acquired from the ARPA report on allergenic fungal pollens and spores of the Veneto region [25].

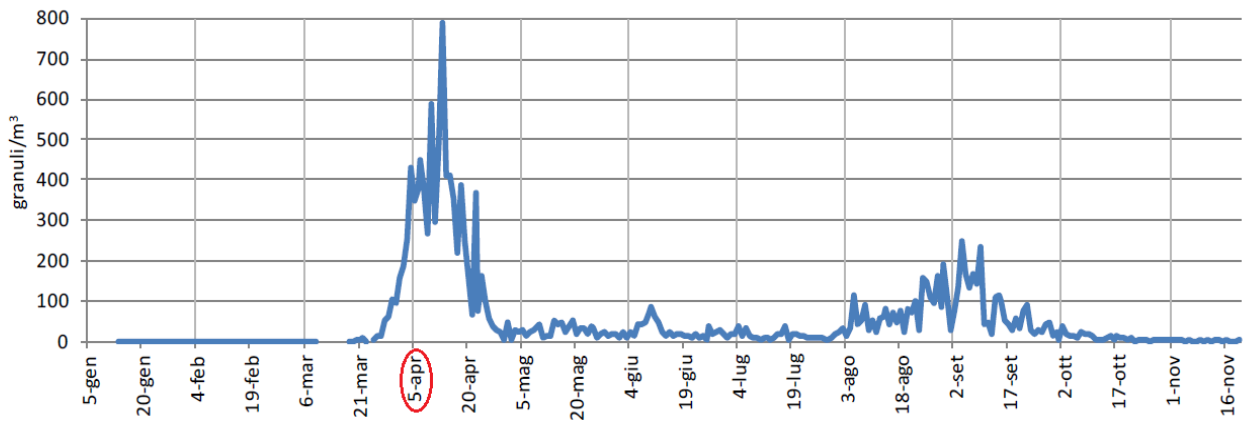


Figure 48: Average daily pollen concentration of Urticaceae, year 2014 **[**25**]**.

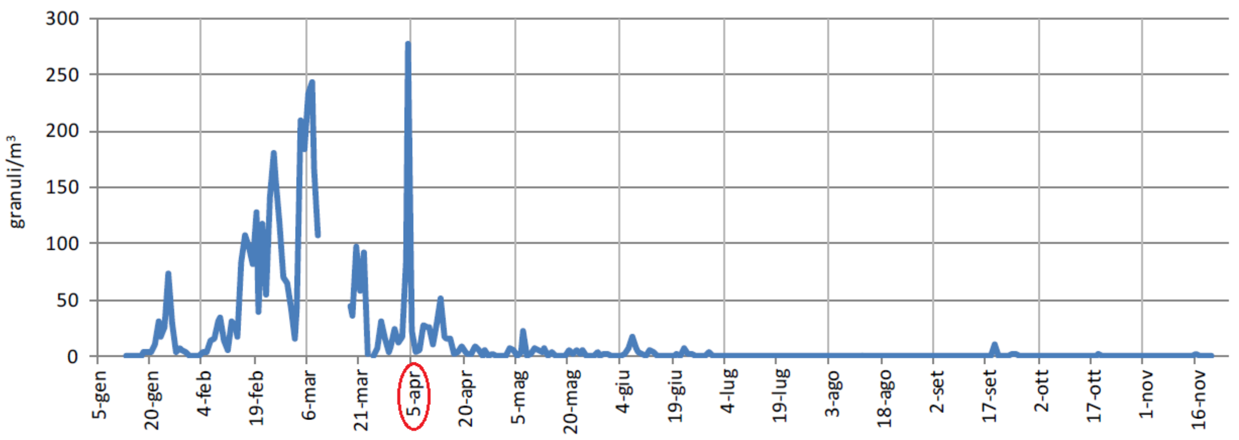


Figure 49: Average daily pollen concentration of Cupressaceae/Taxaceae, year 2014 **[**25**]**.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

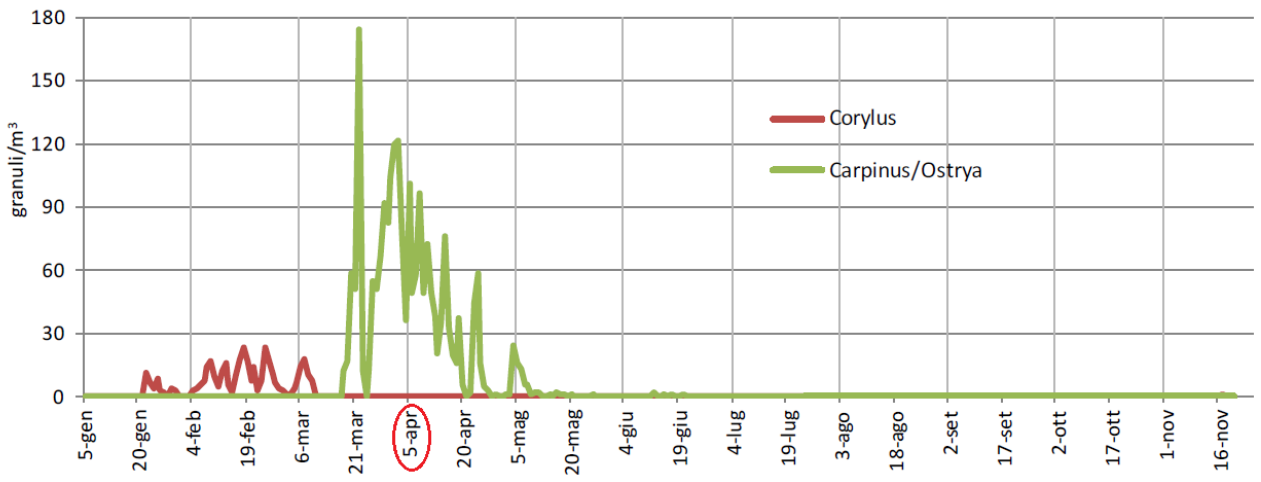


Figure 50: Average daily pollen concentration of Corylaceae - Corylus and Carpinus/Ostrya, year 2014 **[**25**]**.

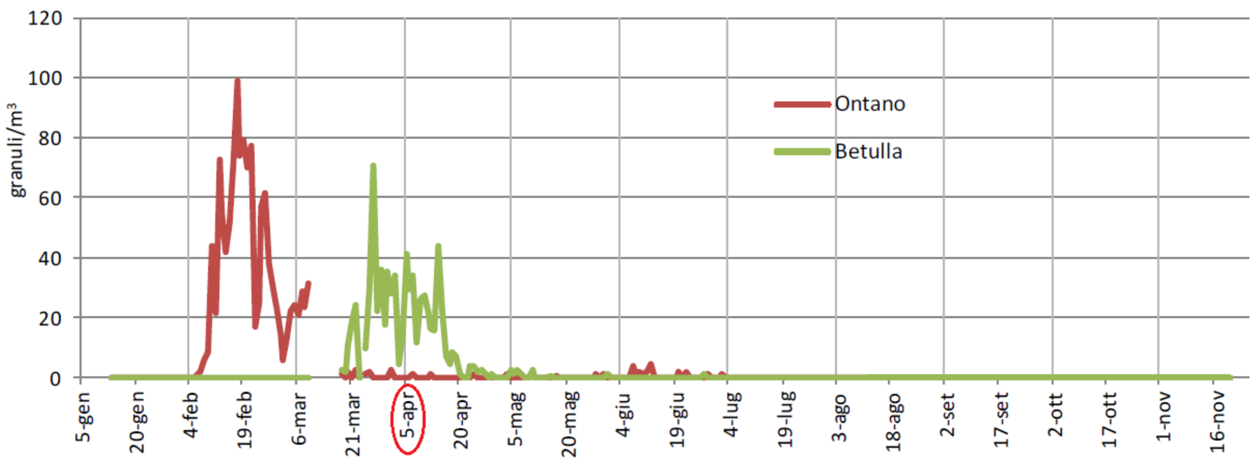


Figure 51: Average daily pollen concentration of Betulaceae - Alnus and Betula, year 2014 **[**25**]**.

From the graphs listed, it is clearly visible that high concentrations of the described pollens occurred simultaneously on 4 April 2014. Furthermore, considering that the graphs are represented using different scales, the feature ranking associated with 4 April 2014 seems to be consistent with the trend assumed by the charts in that period.

Oslo – Data set description

The data extracted from the city of Oslo contain hourly air quality and public transports information, considering the time interval 16:00 – 17:00 on 14 September 2021, collected as batches. Each instance in the data set records information about the public transport route at a given time instant. Air quality monitoring data were collected using the API provided by the Norwegian Institute for Air Research (NILU) [26], while the public transport data were collected using the API provided by the public transport company ENTUR [27].

The information extracted from the two sources includes the following measurements:

- Date of measurement (yyyy-mm-dd hh:mm:ss).
- Longitude coordinate of the vehicle (-180 to 180).
- Latitude coordinate of the vehicle (-90 to 90).
- Distance in meters between the previous stop (or current, if located at stop) and the next stop.
- Name describing the origin of the departure.
- Name describing the destination of the departure.
- Boolean value describing whether the service is a headway transport service.
- Boolean value describing whether the vehicle is affected by traffic congestions or other circumstances which may lead to further delays.

The list of stations and pollutants collected from the city of Oslo is provided below:

- Alnabru (NO, NO2, NOx, PM10, PM2.5 µg/m^3).
- Bryn Skole (NO, NO2, NOx, PM10, PM2.5 µg/m^3).
- Bygdøy Alle (NO, NO2, NOx, PM1, PM10, PM2.5 µg/m^3).
- E6 Alna senter (NO, NO2, NOx, PM10, PM2.5 µg/m^3).
- Hjortnes (NO, NO2, NOx, PM10, PM2.5 µg/m^3).
- Kirkeveien (CO mg/m^3, NO, NO2, NOx, PM10, PM2.5 µg/m^3).
- Loallmenningen (NO, NO2, NOx, PM1, PM10, PM2.5 µg/m^3).
- Manglerud (NO, NO2, NOx, PM10, PM2.5 µg/m^3).
- Rv 4 - Aker sykehus (NO, NO2, NOx, PM10, PM2.5 µg/m^3).
- Skøyen (PM10, PM2.5 µg/m^3).
- Smestad (NO, NO2, NOx, PM10, PM2.5 µg/m^3).
- Sofienbergparken (NO, NO2, NOx, O3, PM10, PM2.5, SO2 µg/m^3).
- Spikersuppa (PM10, PM2.5 µg/m^3).
- Vahl skole (PM10, PM2.5 µg/m^3).

Consequently, in this experiment, it will be necessary to manage the presence of numerical and categorical attributes. However, as already illustrated in Section 4.1.3, this situation does not consist problem as Spark-GHSOM can handle data sets consisting of mixed attributes.

Oslo – Supervised Anomaly Detection

As already described in the previous supervised experiment, an attribute must be identified to be used as the target variable, since the data set is not labelled. The choice fell on the attribute that denotes the presence or absence of traffic congestion, considering the lack of traffic as the normal situation and traffic congestion as the anomalous case.

Table 9 illustrates the results of the conducted experiment, where the τ_1 and τ_2 parameters were set to 0.7 and 1.0, respectively. A description of the columns in the table is provided below:

Input parameters

- Epochs: number of passes of the entire training data set the algorithm has completed.
- Threshold factor: parameter governing the distance within which an instance is considered an anomaly.

Output parameters

- True Normal (TN): number of correctly classified instances of class "normal".
- False Anomaly (FA): number of instances of class "normal" that are classified as "anomaly".
- False Normal (FN): number of instances of class "anomaly" that are classified as "normal".
- True Anomaly (TA): number of correctly classified instances of class "anomaly".

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

As can be guessed, the main objective of an anomaly detector is to minimise the number of False Normal and False Anomaly as well as maximise the number of True Normal and True Anomaly. Between False Normal and False Anomaly, we give higher priority to minimizing False Normal cases, which may represent the most dangerous situation.

Table 9: Results of the Oslo supervised anomaly detection experiment.

epochs	thresh. factor	true normal	false anomaly	false normal	true anomaly
10	6.0	192	4	1	0
10	5.5	192	4	1	0
10	5.0	192	4	1	0
10	4.5	192	4	1	0
10	4.0	192	4	1	0
10	3.5	189	7	1	0
15	6.0	192	4	1	0
15	5.5	192	4	1	0
15	5.0	192	4	1	0
15	4.5	192	4	1	0
15	4.0	192	4	1	0
15	3.5	189	7	1	0

As Table 9 shows, none of the configurations used were able to correctly classify the anomalous instance. Consequently, the descriptive attributes adopted may not be suitable for the prediction of road traffic anomalies. However, the anomaly detector was able to correctly classify most of the instances considered as normal situations, except for few cases where false anomalies were found by the tool.

The following conclusions can be drawn from the experimental results:

- Decreasing the threshold factor led to the discovery of more false anomalies, however did not increase the number of true anomalies.
- Increasing the number of epochs did not change the experimental results.

Oslo – Unsupervised Anomaly Detection

Similar to the Padua unsupervised experiment, this experiment aims to carry out a quantitative analysis through an accurate evaluation of the feature ranking returned by the tool.

In addition, from the experimental results, it could be interesting to understand whether the false anomalies identified in the previous experiment constitute a system error or whether they are true anomalies, but not related to road traffic congestion issues. For this reason, it was decided to reuse one of the configurations adopted in the previous experiment, namely the configuration with the number of epochs set at 10 and the threshold factor set at 4.0.

The experimental results show that, again, the system identified a total of four potential anomalies. In particular, by consulting the feature rankings of these potential anomalies, it emerged that the first two positions are occupied by latitude and longitude attributes respectively, as at those particular time instants the geographical coordinates were not correctly recorded. In this scenario, therefore, the providential intervention of the tool could avoid potential threats to public safety.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

In conclusion, it is possible to state that the anomalies identified by the system may constitute potential threats to public safety. Furthermore, by analyzing the feature ranking, it was possible to conclude that the potential anomalies identified in the previous supervised experiment, even if classified as false anomalies as they are not strictly related to traffic congestion, do constitute potential threats.

4.2 Event classification

This section provides an overview of the event classification pipeline.

Starting from a predefined set of threats (e.g., fire, car accident, attack with guns), similarly to the anomaly detection task, the event classification task aims to classify the current unclassified sensor data under analysis as a particular threat or as a normal case. Therefore, in addition to the anomaly detector, the event classifier would be able to also indicate the type of the threat under analysis. For this task, similarly to the anomaly detection, a training phase and an event identification phase are foreseen. The main difference with the anomaly detection task is that usually the classification of an object/event is guided by the learning of a predictive model in a supervised manner. This means that the data set used for the training of the model must be annotated by describing the possible threats for the real scenario. However, this could be challenging to obtain. To overcome this problem, unsupervised algorithms could be also considered for the classification task as for the anomaly detection. These algorithms are usually less accurate than the supervised ones, since they exploit less informative data avoiding to consider predefined classes.

4.2.1 Density-based clustering

This section provides a description of clustering and density-based clustering.

Clustering (also called Cluster analysis) is an unsupervised task in which objects in a data collection are grouped together into different groups, called clusters, according to the similar characteristics of the objects with respect to the remaining ones in the data collection.

The most popular clustering algorithm is k-means, in which k cluster centroids are identified and the objects in the data collection are assigned to the near cluster centroid. A possible drawback of centroid-based clustering is that the algorithm is unable to handle outliers, since all the objects in the data collection are assigned to a specific cluster, even if they do not belong in any of them. As a consequence, in the anomaly detection, it may be difficult to identify possible anomalies using this method.

In density-based clustering instead, “dense” clusters of points are identified, making possible to learn arbitrarily shaped clusters and thus identify outliers in the data collection. Figure 52 shows the clusters identified by the DBSCAN algorithm, a popular density-based clustering algorithm. The grey points were identified as outliers and, consequently, they were not associated with any cluster.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

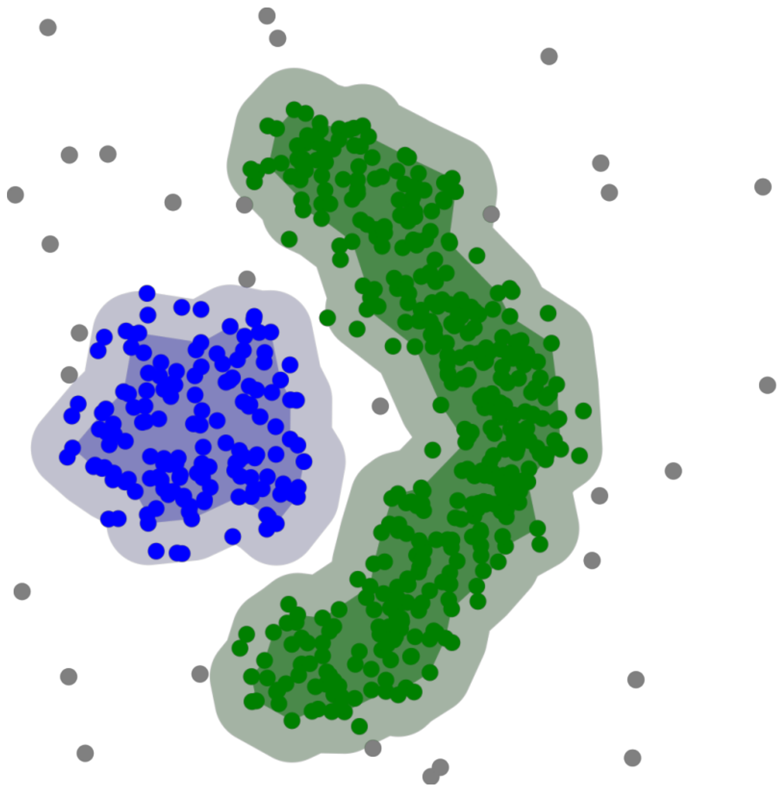


Figure 52: Density based clustering with DBSCAN [29].

4.2.2 DENCAST

This section presents the tool used for the event classification.

The method proposed in [28], called DENCAST, could be used for the event classification, since it is able to perform the unsupervised task of clustering also for the classification. It represents a novel distributed algorithm implemented in Apache Spark, which performs density-based clustering and exploits the identified clusters to solve both single-target and multi-target regression tasks (and thus, solves complex tasks such as time series prediction). Contrary to existing distributed methods, it does not require a final merging step (usually performed on a single machine) and is able to handle large-scale, high-dimensional data by taking advantage of Locality Sensitive Hashing (LSH).

Given a data collection, consisting of n labelled objects, a distributed variant of LSH is applied, to approximate a neighbourhood graph. The obtained neighbourhood graph is composed of a node for each labelled object and undirected edges (represented as pair of nodes $< u , v >$) for those nodes that appear similar enough, according to the data representation obtained after the application of LSH and for a given threshold, called *minPts*. From this moment, the method will just make use of the neighbourhood graph, that can be considered an approximation of the objects in the data collection and their distances. At this point, the obtained clusters could be exploited to classify new data. The entire process is summarized in Figure 53.

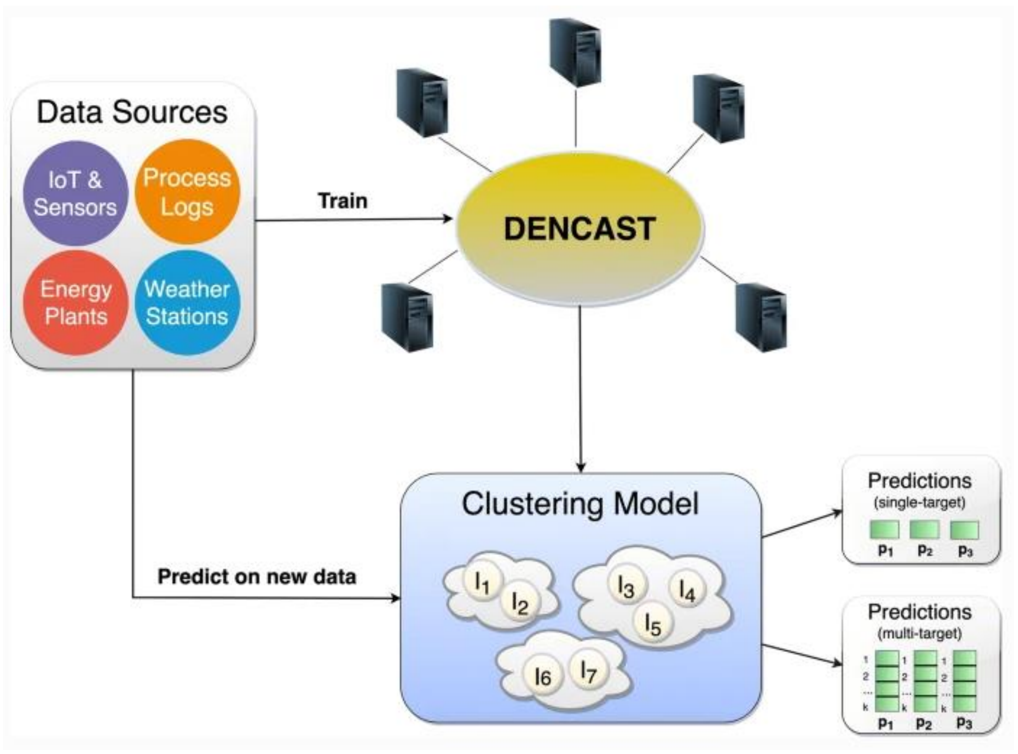


Figure 53: Overview of the DENCAST architecture **[**28**]**.

5 Conclusions and next steps

The deliverable presented both the theoretical and technological details of an innovative big data solution for the smart city domain, supporting analytics for anomaly detection and event classification, and posing particular attention to data management and protection. It proposed a solution where traditional data management components are enriched with an ingestion-based access control system that enforces access to data in a distributed, multi-party big data environment, before the execution of analytics pipelines.

The proposed solution was deployed on an Apache-based big data engine extended with an ingestion-time access control approach to achieve full data governance on the ingested data driving analytics for anomaly detection and event classification. The entire data management and analytics functionalities have been tested independently in a simplified smart city scenario, showing the feasibility of the approach.

5.1 Novelties and strengths of the proposed solution

The peculiarities of big data systems (i.e., ecosystem of services, mainly from third parties, where huge amount of diverse data are collected at high rates and must be proven to be trustworthy) increase both the data management and the analytics complexity under multiple perspectives.

With respect to the data protection and management problem, the methodology presented in this deliverable addresses the issues of access control models developed in the 2000s for Service-Oriented Architectures, which failed to adequately manage the creation, use and dissemination of big data. Existing models can be both ineffective and inefficient when applied to today's dynamic coalitions, where collaborative processes are carried out involving multi-party data collection and analytics, and there are continuous changes in the security space structure of temporary coalitions, with many parameters for access right decisions unknown at policy writing time. The attribute-based access control methodology presented in this deliverable works at data ingestion time. In particular, the new ingestion procedure performs data-model transformations of the ingested resources that depend on the state of the collaboration and the specific target where data are routed (separation of duties). This approach makes policies verifiable and adaptable to the evolving state of resources and individuals within a specific big data context. It is based on an advanced ETL schema where i) data transformation is the result of a policy enforcement and ii) the target of the load phase of the data ingestion procedure is the data lake infrastructure. The scope of this advanced ETL is to enforce access control at ingestion time before any access request is received. Therefore, a city-level data governance strategy can be applied to all the services or to the whole smart city ecosystem by simply changing or adding new policies, while in case of an ad hoc solution it requires to change every single service impacted by the new data governance strategy.

With respect to the data analytics problem, the methodology presented in this deliverable can handle large-scale and high dimensional data, without incurring computational bottlenecks as the algorithm proposed runs on multiple computational nodes, in a distributed fashion. Moreover, as most real-world datasets are characterized by numerical and categorical attributes, the proposed approach can handle mixed-attribute datasets, thus overcoming the limitations of those algorithms that can run on numerical datasets only. The knowledge acquired by the algorithm can be updated directly by the user, thus avoiding the instantiation of a new model, with a consequent gain in training time. Finally, the output of the tool has been made interpretable for the user by introducing a feature ranking, to better understand if the identified anomaly represents a threat or not.

5.2 Future work

In the final report due at M22, we plan to describe further improvements in the second year of the project, taking advantage of the close collaboration with some tasks and WPs. In particular:

1. The acceptance pilots and live exercises planned and the process for validating IMPETUS platform described in D7.1 "Validation Plan" will be used to test the solution in this deliverable and to have a feedback from SoC operators.
2. We will work towards a complete integration of the solutions to data management and data analytics in this deliverable, as well as within the whole IMPETUS platform according to the requests from city pilots.

3. The contributions of WP5 and especially of D5.2 “Initial mechanisms to preserve privacy in the secure smart city” will be used to refine privacy-enhancing technologies that could be adopted in the data transformation process at the basis of the ingestion-time access control solution in this deliverable.
4. Additional algorithms based on the idea of predictive clustering will be proposed.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

6 References

[1] S. Ghemawat, H. Gobioff and S.-T. Leung, "The Google file system," in *ACM symposium on Operating systems principles*, 2003.

[2] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM,* vol. 51, no. 1, pp. 107--113, 2008.

[3] R. T. Fielding, Architectural styles and the design of network-based software architectures, Irvine: University of California, 2000.

[4] L. Sweeney and P. Samarati, "Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression," SRI International, 1998.

[5] V. C. Hu et al., "Guide to attribute based access control (abac) definition and considerations," *NIST special publication,* vol. 800, no. 162, pp. 1--54, 2013.

[6] P. Colombo and E. Ferrari, "Access control technologies for Big Data management systems: literature review and future trends," *Cybersecurity,* vol. 2, no. 1, pp. 1--13, 2019.

[7] F. M. Awaysheh, M. Alazab, M. Gupta, T. F. Pena and J. C. Cabaleiro, "Next-generation big data federation access control: A reference model," vol. 108, pp. 726-741, 2020.

[8] M. Gupta, F. Patwa and R. Sandhu, "Object-Tagged RBAC Model for the Hadoop Ecosystem," in *Data and Applications Security and Privacy*, 2017.

[9] M. Gupta, F. Patwa and R. Sandhu, "An Attribute-Based Access Control Model for Secure Big Data Processing in Hadoop Ecosystem," in *Workshop on Attribute-Based Access Control (ABAC'18)*, New York (US), 2018.

[10] P. Colombo and E. Ferrari, "Towards a Unifying Attribute Based Access Control Approach for NoSQL Datastores," in *International Conference on Data Engineering (ICDE)*, 2017.

[11] V. Hu, T. Grance, D. F. Ferraiolo and K. D. R., "An Access Control Scheme for Big Data Processing," in *International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, Miami, FL (US), 2014.

[12] M. M. Ben Aissa, L. Sfaxi and R. Robbana, "DECIDE: A New Decisional Big Data Methodology for a Better Data Governance," in *European, Mediterranean, and Middle Eastern Conference on Information Systems (EMCIS 2020)*, 2020.

[13] A. Al-Badi, A. Tarhini and A. I. Khan, "Exploring Big Data Governance Frameworks," *Procedia Computer Science,* no. 141, pp. 271-277, 2018.

[14] Y. Demchenko et al., "Addressing big data issues in scientific data infrastructure," in *collaboration technologies and systems (CTS)*, 2013.

[15] V. Chandola, A. Banerjee and A. Kumar, "Anomaly detection: A survey," vol. 41, no. 3, 2009.

[16] T. Kohonen, "Essentials of the self-organizing map," vol. 37, pp. 52-65, 2013.

[17] E. Araujo, C. R. Silva and D. J. B. S. Sampaio, "Video target tracking by using competitive neural networks," vol. 4, no. 8, pp. 420-431, 2008.

[18] M. Dittenbach, D. Merkl and A. Rauber, "The growing hierarchical self-organizing map," *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks,* vol. 6, pp. 15-19, 200.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

[19] A. Malondkar, R. Corizzo, I. Kiringa, M. Ceci and N. Japkowicz, "Spark-GHSOM: Growing Hierarchical Self-Organizing Map for large scale mixed attribute datasets," *Information Sciences,* vol. 496, pp. 572-591, 2019.

[20] "Arpa Veneto, Open Data," [Online]. Available: https://www.arpa.veneto.it/dati-ambientali/open-data.

[21] "ARPA Veneto report on allergenic fungal pollens and spores," 2014. [Online].

[22] "Norwegian Institute for Air Research (NILU)," [Online]. Available: https://www.home-assistant.io/integrations/nilu/.

[23] "ENTUR API documentation," [Online]. Available: https://developer.entur.org/pages-real-time-intro.

[24] "https://en.wikipedia.org/wiki/DBSCAN," [Online].

[25] R. Corizzo, G. Pio, M. Ceci and D. Malerba, "DENCAST: Distributed Density-Based Clustering for Multi-Target Regression," *Journal of Big Data.*

[26] V. Goyal et al., "Attribute-based encryption for fine-grained access control of encrypted data.," in *13th ACM conference on Computer and communications security*, 2006.

[27] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual international conference on the theory and applications of cryptographic techniques.*, Berlin, 2005.

[28] X. Boyen and B. Waters, "Anonymous hierarchical identity-based encryption (without random oracles).," in *Annual International Cryptology Conference*, Berlin, 2006.

[29] F. Armknecht et al., "A Guide to Fully Homomorphic Encryption," in *IACR Cryptol. ePrint Arch*, 2015.

[30] M. Anisetti, C. Ardagna, C. Braghin, E. Damiani, A. Balestrucci and A. Polimeno, "Dynamic and Scalable Enforcement of Access Control Policies for Big Data," *ACM MEDES,* 2021.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

7 APPENDIX A: Mapping to requirements in D1.2

The big data solution for the smart city domain presented in this deliverable has been designed according to the IMPETUS platform requirements in D1.2. Table 10 contains a subset of the requirements from D1.2 that are relevant for this deliverable and describes how our solution addresses them.

Table 10: Mapping to requirements in D1.2.

ID	Requirement	How it is addressed
1	An Access Control methodology should handle access rights and data control.	The proposed solution implements a data governance approach based on an access control methodology that works at ingestion time. The access control methodology supports data sanitization and transformations. They are applied prior to the storage of data in the data lake and the execution of big data analytics of the big data engine tool in WP4.
3	The Access Control methodology should support sanitization policies that are enforced at ingestion time before an access to data is granted.	The proposed solution supports sanitization and transformation of data prior to their storage in the data lake and their use by big data analytics of the big data engine tool in WP4.
46	The change detector should raise an alert when sensor data do not follow an expected behavior, according to historical data for any variable under observation	The proposed algorithm can train a model that catches anomalies in the data collected by the sensors, with respect to historical data. The identified anomalies are used to raise alerts for further investigation.
47	The event classifier should raise an alert when sensor data represents a previously defined class of threat	The proposed algorithm can train a predictive model that identifies specific threats from data. Unlike the anomaly detection task, this task needs further information on the predefined classes of threats encoded in data (at least for a subsample) to train the predictive model.
58	The Access Control methodology and the ingestion process must use standard interfaces.	The proposed solution is implemented within a traditional Apache-based big data engine. The ingestion process is built on either REST interfaces or Kafka queues, and can connect to different types data sources.
60	The platform operators should manage Access Control policies.	The proposed solution permits to customize access control policies using the Apache Ranger component, driving the behavior of the tool for large-scale data analytics in WP4.
61	The ingestion process should support ingestion of structured and unstructured data.	The proposed solution supports ingestion of both structured and unstructured data using an Apache-based big data engine.
62	Data analytics should be executed on ingested data.	The proposed access control-based data governance approach is applied prior to the storage of data in the data lake and their use by big data analytics of the big data engine tool in WP4. Access control enforcement and data sanitization/transformations are applied on ingested data, which are then fed into the data analytics algorithms.

63	The Ingestion process must support standard ingestion workflows on data measured and collected by the pilot cities.	The proposed solution implements a generic approach that can support ingestion from different data sources including the ones of IMPETUS pilot cities.
115	The IMPETUS project must develop approaches and methodologies for large-scale data analytics and visualization, as well as implement access control requirements to ensure the security of the data.	The proposed solution supports large-scale data analytics for anomaly detection and event classification. The execution of the data analytics is mediated by the access control methodology at the basis of the data governance approach in WP4.

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

Members of the IMPETUS consortium

SINTEF	SINTEF, Strindvegen 4, Trondheim, Norway, https://www.sintef.no	Joe Gorman joe.gorman@sintef.no
Institut Mines-Télécom	Institut Mines Telecom, 19 place Marguerite Perey, 91120 Palaiseau, France, https://www.imt.fr	Joaquin Garcia-Alfaro joaquin.garcia_alfaro@telecom-sudparis.eu
UNÎMES	Université de Nimes, Rue du Docteur Georges Salan CS 13019 30021 Nîmes Cedex 1, France, https://www.unimes.fr	Axelle Cadiere axelle.cadiere@unimes.fr
cini consorzio interuniversitario nazionale per l'informatica	Consorzio Interuniversitario Nazionale per l'Informatica, Via Ariosto, 25, 00185 – Roma, Italy, https://www.consorzio-cini.it	Donato Malerba donato.malerba@uniba.it
Università degli Studi di Padova	University of Padova, Via 8 Febbraio, 2 - 35122 Padova, Italy, https://www.unipd.it	Giuseppe Maschio giuseppe.maschio@unipd.it
Entrepreneurship Development Centre for BIOTECHNOLOGY and MEDICINE	Biotehnoloogia ja Meditsiini Ettevõtluse Arendamise Sihtasutus, Tiigi 61b, 50410 Tartu, Estonia, https://biopark.ee	Sven Parkel sven@biopark.ee
SIMAVI Software Imagination & Vision	SIMAVI, Complex Victoria Park, Corp C4, Etaj 2, Șos. București – Ploiești, nr. 73 – 81, Sector 1, București, Romania, https://www.simavi.ro	Gabriel Nicola Gabriel.Nicola@simavi.ro Monica Florea Monica.Florea@simavi.ro
THALES	Thales Nederland BV, Zuidelijke Havenweg 40, 7554 RR Hengelo, Netherlands, https://www.thalesgroup.com/en/countries/europe/netherlands	Johan de Heer johan.deheer@nl.thalesgroup.com
CINEDIT INTELLIGENT VIDEO ANALYTICS	Cinedit VA GmbH, Poststrasse 21, 8634 Hombrechtikon, Switzerland, https://www.cinedit.com	Joachim Levy j@cinedit.com

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.

	Insikt Intelligence, Calle Huelva 106, 9-4, 08020 Barcelona, Spain, https://www.insiktintelligence.com	Dana Tantu dana@insiktintelligence.com
	Sixgill, Derech Menachem Begin 132 Azrieli Tower, Triangle Building, 42nd Floor, Tel Aviv, 6701101, Israel, https://www.cybersixgill.com	Benjamin Preminger benjamin@cybersixgill.com Ron Shamir ron@cybersixgill.com
	XM Cyber, Galgalei ha-Plada St 11, Herzliya, Israel https://www.xmcyber.com	Lior Barak lior.barak@xmcyber.com Menachem Shafran menachem.shafran@xmcyber.com
	City of Padova, via del Municipio, 1 - 35122 Padova Italy, https://www.padovanet.it	Enrico Fiorentin fiorentine@comune.padova.it Stefano Baraldi Baraldis@comune.padova.it
	City of Oslo, Grensen 13, 0159 Oslo, Norway, https://www.oslo.kommune.no	Osman Ibrahim osman.ibrahim@ber.oslo.kommune.no
	Institute for Security Policies, Kruge 9, 10000 Zagreb, Croatia, http://insigpol.hr	Krunoslav Katic krunoslav.katic@insigpol.hr
	International Emergency Management Society, Rue Des Deux Eglises 39, 1000 Brussels, Belgium, https://www.tiems.info	K. Harald Drager khdrager@online.no
	Unismart – Fondazione Università degli Studi di Padova, Via VIII febbraio, 2 - 35122 Padova, Italy, https://www.unismart.it	Alberto Da Re alberto.dare@unismart.it

The research leading to these results has received funding from Horizon 2020, the European Union's Framework Programme for Research and Innovation (H2020) under grant agreement n° 883286.