Integrated Framework for Predictive and Collaborative Security of Financial Infrastructures

Start Date of Project: 2018-05-01

Duration: 36 months

D5.1 Report on Integrated BigData Infrastructure - I

Deliverable Details	
Deliverable Number	D5.1
Deliverable Title	Report on Integrated BigData Infrastructure - I
Revision Number	3.0
Author(s)	GFT
Due Date	28/02/2019
Delivered Date	
Reviewed by	CNR, IBM
Dissemination Level	PU
EC Project Officer	Christoph CASTEX

No.	**Contributing Partner**
1.	GFT (author)
2.	CNR (reviewer)
3.	IBM (contributor & reviewer)
4.	HPE (contributor)
5.	INNOV (contributor)
6.	ORT (contributor)
7.	FBK (contributor)

Document Status

☒ draft

☒ Consortium reviewed

☒ WP leader accepted

☒ Project coordinator accepted

This project has received funding from the European Union's Horizon 2020 research and innovation programme 2014-2020 under grant agreement No 786727

Revision History

Version	By	Date	Changes
1.0	GFT	01/02/2019	Initial ToC, bullet points, Lorem Ipsum examples
1.1	GFT	18/02/2019	Complete draft ready for Peer Review
2.0	CNR, IBM	22/02/2019	Document amended and commented after Peer review
2.1	GFT	25/02/2019	Complete document ready for QA review
3.0	GFT	28/02/2019	Final version for submission

Executive Summary

FINSEC BIG DATA INFRASTRUCTURE aims to provide a complete stack comprising an infrastructure management system to support the analysis and prediction algorithms of the FINSEC platform, managing the historical and real data of the security, the knowledge base and the description of the assets, being completely scalable, adaptable to runtime and high performance.

In this way, FINSEC BIG DATA INFRASTRUCTURE will respond to the needs of big data operations and data intensive applications. The system will base all infrastructure management decisions on data analysis, monitoring data from implementations and the logic derived from data operations that govern and influence storage, compute and network resources, as well as their interdependencies. In addition to the infrastructure management system, "Data as a Service" will be offered to data providers, decision makers, private and public organizations. Approaches to data cleansing, data layout and efficient storage, combined with seamless data analysis, will be holistically implemented across multiple data stores and locations.

In order to provide the information required for better management of the infrastructure, FINSEC BIG DATA INFRASTRUCTURE will provide a set of basic services, such as the CRUD DB interface, which facilitates the analysis and sizing of data-driven applications in terms of service forecasting required data, their interdependencies with the micro-services application and the underlying resources required. This will allow the identification of data properties of their applications and their data requirements, thus enabling FINISEC BIG DATA INFRASTRUCTURE to perform the deployment with specific performance and quality assurance. In addition, a data toolkit will allow data scientists to assimilate their data analysis functions and specify their preferences and constraints, which will be exploited by the infrastructure management system for resources and data management. Finally, a process modelling framework will be provided to enable function-based modelling of processes, which will be mapped to an automated analysis of the process of a concrete technical level.

The aforementioned key results of FINSEC BIG DATA INFRASTRUCTURE are reflected in a set of main constituent elements in the corresponding general stack architecture. This deliverable describes the key functionalities of the general architecture, the interactions between the main building blocks and their components, while providing a first version of the interiors of these components regarding the research approaches to be implemented during the project. Further detailed information and specifications of the components will be provided through the relevant design reports and specification of the relevant project work packages.

TABLE OF CONTENTS

List of Tables

List of figures

1. Introduction

This document describes the solutions to integrate the predictive and collaborative data-driven machine learning algorithms with a Big Data infrastructure specifically designed for the FINSEC Platform, called in the following FINSEC BIG DATA INFRASTRUCTURE.

The FINSEC BIG DATA INFRASTRUCTURE starts from the business requirements and use cases analyzed in the previous tasks of the projects (T2.1, T2.3 and T2.4) and is specifically designed to be hosted in the cloud in order to work seamlessly with the other applications of the Reference Architecture (D2.4) to enable the delivery of security services based on the SECaaS paradigm.

The design involves the methods for integration of the data-driven components of the project's security toolbox (SIEM, CCTV, Anomaly Detection, Predictive Analytics, RAE, PenTest) over the same infrastructures, along with the integration of the analytics infrastructure of the project developed in other WPs.

1.1. Background

The new data-driven industrial revolution highlights the need for big data technologies, to unlock the potential in various application domains (e.g. finance, security, transportation, healthcare, etc). In this context, big data analytics frameworks exploit several underlying infrastructure and cluster management systems.

However, these systems have not been designed and implemented in a "big data context", and they instead emphasise and address the computational needs and aspects of applications and services to be deployed. FINSEC BIG DATA INFRASTRUCTURE aims at addressing these challenges (depicted in *Figure 1*) through robust microservices that range from a scalable, runtime-adaptable infrastructure management system (to support analytics according to data aspects), to techniques for dimensioning big data applications, modelling and analysis of organizations, assets and physical logical infrastructure, as well as provisioning data-as-a-service, by exploiting a seamless big data framework. The *Figure 1* shows how a BigData infrastructure can face (by which means and with which benefits) the previously mentioned challenges.

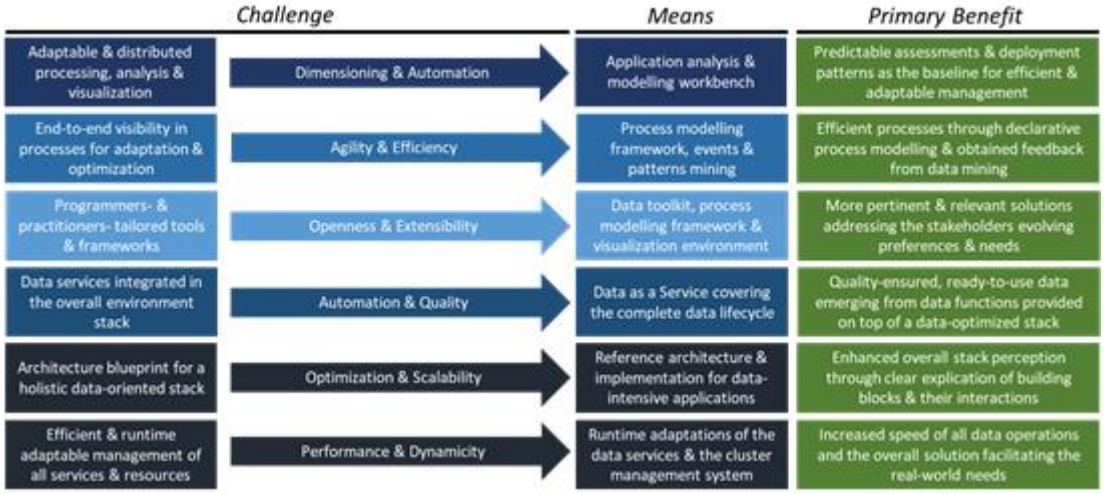

Figure 1: Technical challenges[1]

[1] GFT, BigDataStack, , D2.4, GA No 779747, 2018

1.2. Document structure

The structure of the document is aimed at covering in a complete way all the aspects of the FINSEC BIG DATA INFRASTRUCTURE, from the technological background laying behind the design work done to the integration of the data model into the infrastructure, passing through the functional and technical requirements coming from the outcomes of the previous activities within the project and the description of the design features of the infrastructure. The document is structured as follows: the current section provides the background and context of this task and deliverable, together with an overview of the relevant input from the previous tasks and the output for the following activities of the project; Section 2 provides State of the Art relevant to Big Data Infrastructure from both the database and application points of view; Section 3 is focused on an Analysis and Mapping of the Requirements from previous tasks and from different aspects of the project; Section 4 represents the core of the deliverable, and contains the FINSEC BIG DATA INFRASTRUCTURE design including the key provisions and the overall architecture and a description of the main architecture components; Section 5 deals with implementation and integration aspects of the infrastructure, with the details about its deployment for FINSEC purposes; conclusions are drawn in Section 6. The document is completed by ANNEX A about the data model that was thought for the project.

1.3. Relevant Task Input

The deliverable takes as input some information coming from the previous tasks within the project. The main input are coming from the definition of the Reference Architecture done in Task 2.5 and formalized in the Deliverable 2.4, which identifies the boundaries of the data layer, thus setting the data exchange interfaces between the FINSEC BIG DATA INFRASTRUCTURE and the other relevant building blocks of the project. Moreover, Task 2.4 dealing with the integrated data model (cyber + physical information and events) represents an input source for Task 5.1, giving details about the nature of the data to be exchanged and stored by the infrastructure. Finally, Section 3 explains how some requirements for the FINSEC BIG DATA INFRASTRUCTURE come from the definition of the predictive security done in Task 2.3.

1.4. Relevant Task Output

The main outcome of this deliverable, will be the definition of a detailed design architecture and the choice of the technologies for the BigData infrastructure. This kind of information will be needed in the following of Task 5.1, where the infrastructure will be implemented and integrated within M18, with the preparation of Deliverable 5.2. The infrastructure will be integrated together with all the other modules of the FINSEC platform in Task 5.3 (probes, application, services, infrastructures, etc.) and tested during WP6 pilot activities.

2. State of the Art

Infrastructural technologies are the core of the Big Data ecosystem. For decades, enterprises relied on relational databases for processing structured data. However, the volume, velocity and variety of data mean that relational databases often cannot deliver the performance and latency required to handle large, complex data. Thus, new database solutions have been emerged in order to provide advantages in terms of performance, scalability, and suitability for Big Data environments. Among these solutions, there are NoSQL databases, NewSQL databases, and file storage systems like HDFS and GFS detailed in the following sections.

Taking into account the FINSEC requirements arising from the previous phases of the project, in particular the work of WP2 in Tasks T2.3, Task T2.4 and related deliverables D2.3 and D2.4, and

considering the FINSEC DATA MODEL for the Cyber Physical Threat Intelligence (CPTI) in the financial sector addressed by the FINSEC Platform, in the following sections the most relevant state of the art solutions to support complex and extensible data models and applications will be analysed.

Moreover, solutions for managing large quantities of data, their ingestion and Big Data infrastructure will be analysed to form the background on top of which the specific FINSEC BIGDATA INFRASTRUCTURE will be built.

2.1. NoSQL Solutions

One of the key advances in resolving problem of big data has been the emergence of NoSQL as an alternative database technology. A very flexible and schema-less data model, horizontal scalability, distributed architectures, and the use of languages and interfaces that are “not only” SQL typically characterize this technology. NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS. The NoSQL data model does not guarantee ACID properties (Atomicity, Consistency, Isolation and Durability) but instead it guarantees BASE properties (Basically Available, Soft state, Eventual consistency). In addition, it is in compliance with the CAP (Consistency, Availability, Partition tolerance) theorem. NoSQL databases have many data models: Key-Value, Document, Column, Graph and Multi-model. We focus on document databases because it stores data as documents that are based on a specific encoding such as JSON, BSON, XML, etc. Although they differ in their data model, all NoSQL databases allow a relatively simple storage of unstructured, distributed data and achieve high scalability. They are best adapted for applications that don’t use a fixed schema, or don’t require ACID operations, and for intensive read and update OLTP (On-Line Transaction Processing) workloads.

A. **MongoDB** is an open source, document oriented database that is written in C++. In MongoDB, the documents are mainly stored in BSON (Binary JSON) format which is efficient both in storage space and scan speed when compared to JSON. MongoDB defines its own query language. Queries can be performed with complex criteria, conditions, sorting, embedded documents, etc. It is also possible to use indexing, like in relational databases, which allows performing faster queries. In addition, MongoDB supports MapReduce, which allows complex aggregations across documents. The changes to a single-document are guaranteed to be atomic. The addition of multi-document ACID transactions in MongoDB 4.0 makes it the only open source data platform to combine the speed, flexibility, and power of the document model with ACID data integrity guarantees. Through snapshot isolation, transactions provide a globally consistent view of data, and enforce all-or-nothing execution to maintain data integrity. MongoDB uses the GridFS, a distributed file system that stores big files in the form of chunks or parts. Database Sharding can also be applied to allow distribution across multiple systems for horizontal scalability. Although, MongoDB still supports the master-slave replication, replica sets are recommended for new production deployments to replicate data in a cluster. Replica sets are usually used for data redundancy, automated failover, read scaling, server maintenance without downtime, and disaster recovery. In general, MongoDB is an excellent choice for projects that deal with massive volumes of data and significant scale-out requirements where high performance is critical. It also helps in situations where data is too complicated and heterogeneous to be modelled in a relational schema or enable real-time analytics. MongoDB is frequently used for large scalable applications like mobile apps, content management, real-time analytics, and applications involving the Internet of Things.

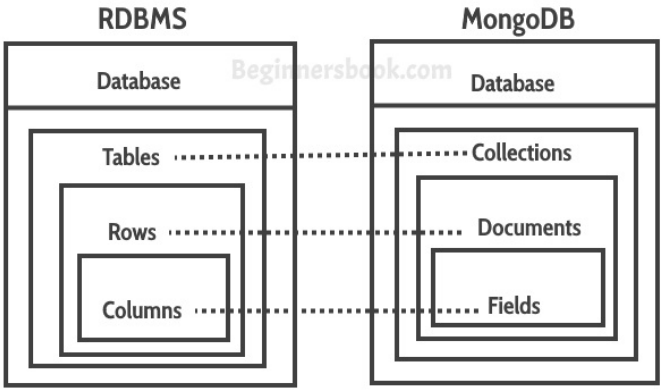

Figure 2: MongoDB architecture

B. **Elasticsearch** is a highly scalable open-source full-text search and analytics engine. It is designed to store, search, and analyze big volumes of data quickly and in near real time. It was built on top of Apache Lucene, which is an open source search engine library. Elasticsearch provides a rich, flexible, query language based on JSON called Query DSL (Domain Specific Language) which allows user to build much more complicated, robust queries. An index in Elasticsearch is similar to a database in a RDBMS, it can store different types of documents, update them, and search for them. Each document is a JSON object which consists of zero or more fields, where each field is either a primitive type or a more complex structure. A document has a Document type associated with it, however, all documents are schema-free, which means that two documents of the same type can have different sets of fields. In order to store a large amount of data that can exceed the hardware limits of a single node, Elasticsearch provides the ability to subdivide the index into multiple pieces called shards. Moreover, Elasticsearch allows to make one or more copies of index's shards into what are called replica shards. Replication is important for better availability and performance in case a shard/node fails.

   Elasticsearch is based on REST architecture and provides API endpoints to not only perform CRUD operations over HTTP API calls, but also to enable users to perform cluster monitoring tasks using REST APIs. REST endpoints also enable users to make changes to clusters and indices settings dynamically. Elasticsearch operations such as reading or writing data usually take less than a second to complete which lets Elasticsearch a good choice for near real-time use cases such as application monitoring and anomaly detection.

C. **CouchDB** is an Open Source NoSQL Database implemented in concurrency-oriented language Erlang. It utilizes JSON to store data and JavaScript as its query language. It uses a B-tree index, updated during data modifications. These modifications have ACID properties on the document level and the use of MVCC (Multi-Version Concurrency Control) enables readers to never block. CouchDB's document manipulation uses optimistic locks by updating an append-only B-tree for data storage, meaning that data must be periodically compressed. This compression, in spite of maintaining availability, may hinder performance. Regarding fault-tolerant replication mechanisms, CouchDB supports both master-slave and master-master replication that can be used between different instances of CouchDB or on a single instance. Scaling in CouchDB is achieved by replicating data, a process which is performed asynchronously. It does not natively support sharding/partitioning. Consistency is guaranteed in the form of strengthened eventual consistency, and conflict resolution is performed by selecting the most up to date version (the application layer can later try to merge conflicting changes, if possible, back into the document).

D. **Couchbase** is an open source, distributed, NoSQL document-oriented database for latency sensitive, interactive, always-on (24x7) applications. It is derived from CouchDB and Membase databases. Couchbase documents are stored as JSON. Data are distributed across nodes and replicated with a master-master model. The greatest differences compared to CouchDB it's the introduction of N1QL, a query language that improves data transformation and manipulation, and a Memcached-based caching technology that best suits real time access. Memcached allows to cache data in RAM memory across the cluster nodes what increases the performance for real-time requirements. The data is also persisted on disk. Couchbase provides real time data processing by using Kafka, Storm, and Sqoop components and it is scalable thanks to its identical nodes and automatic sharding.

2.2. NewSQL Solutions

NewSQL is a technology that aims at making current relational SQL more scalable. It's an attempt to combine NoSQL and SQL. SQL provides ACID properties but isn't fast enough when it comes to concurrency. NoSQL aims at Brewers CAP theorem but doesn't necessarily provide ACID properties. NewSQL tries to provide relational DBMS that has same scalability as NoSQL for OLTP while still providing ACID properties.

The general features of NewSQL technologies are well described by VoltDB, which represents a general paradigm for this family of solutions.

VoltDB is an in-memory database, which depends on the main memory for data storage. VoltDB is an ACID relational database that uses a shared-nothing architecture, ensuring that the data is always correct and available. The data is organized into memory partitions, and transactions are sent by clients connected to the database. VoltDB uses horizontal scalability to increase the capacity of the nodes of the existing database, or the number of nodes in a shared-nothing cluster. For high availability VoltDB uses partitions which are transparently replicated across multiple servers. If one fails all data remains available and consistent for continuum operation. Memory performance with durability on the disk is possible with the VoltDB snapshot. The snapshot is a complete copy of the database at a certain point in time that is written on the disk. VoltDB uses asynchronous replication on the WAN (Wide Area Network) for loss recovery. The remote copy is a read-only while it is not considered to be the primary database. Voltdb is a great choice for use cases where very high performance and predictably low latency are critical as well as where accurate counting/accounting is important, such as in policy enforcement, personalization, fraud/anomaly detection, and other request-response style fast-decisioning and fast data pipeline applications.

2.3. Comparative table of relevant data base solutions

Table 1: comparison between DB solutions

Name	***Couchbase***	***MemSQL***	***MongoDB***	***Elasticsearch***	***VoltDB***
Description	JSON-based document store derived from CouchDB with a Memcached compatible interface. Originally called Membase.	MySQL wire-compliant distributed RDBMS that combines an in-memory row-oriented and a disc-based column-oriented storage.	NoSQL database, a document store.	A distributed, RESTful search and analytics engine based on Apache Lucene. Elasticsearch allows the combination of many types of searches such as structured, unstructured, geo, and metric	Distributed In-Memory NewSQL RDBMS Used for OLTP applications with a high frequency of relatively simple transactions that can hold all their data in memory.

Primary database model	Document store	Relational DBMS	Document store	Search engine	Relational DBMS
Secondary database models		Document store Key-value store	Key-value store	Document store	Key-value store
Website	www.couchbase.com	www.memsql.com	www.mongodb.com	www.elastic.co/products/elasticsearch	www.voltdb.com
documentation	docs.couchbase.com	docs.memsql.com	docs.mongodb.com/manual	www.elastic.co/guide/en/elasticsearch/reference/current/index.html	docs.voltdb.com
Developer	Couchbase, Inc.	MemSQL Inc.	MongoDB, Inc.	Elastic	VoltDB Inc.
Initial release	2011	2013	2009	2010	2010
Current release	6.0.0, October 2018	6.7, November 2018	4.0.5, December 2018	6.6.0, January 2019	8.4, January 2019
License	Open Source, Apache version 2	Commercial. Free developer edition available.	Open Source, MongoDB Inc.'s Server Side Public License v1. Prior versions were published under GNU AGPL v3.0. Commercial licenses are also available.	Open Source, Apache Version 2; Elastic License	Open Source, AGPL for Community Edition. Commercial license for Enterprise, AWS, and Pro Editions
Cloud Service	Yes	Yes	Yes (MongoDB Atlas DBaaS)	Yes	Yes
Cloud-based only	no	no	no	no	no
Implementation language	C, C++, Go and Erlang	C++	C++	Java	Java, C++
Server operating systems	Linux, OS X, Windows	Linux 64 bit	Linux, OS X, Solaris, Windows	All OS with a Java VM	Linux, OS X
Data scheme	schema-free	Yes	Although schema-free, documents of the same collection often follow the same structure. Optionally impose all or part of a schema by defining a JSON schema.	schema-free, Flexible type definitions. Once a type is defined, it is persistent	Yes
XML support	no	no	no	no	

Secondary indexes	Yes	Yes	Yes	Yes, All search fields are automatically indexed	Yes
SQL Support	SQL-like query language (N1QL)	yes, but no triggers and foreign keys	Read-only SQL queries via the MongoDB Connector for BI	SQL-like query language	Yes, only a subset of SQL 99
APIs and other access methods	Memcached protocol RESTful HTTP API (only for server administration)	JDBC ODBC	proprietary protocol using JSON	Java API RESTful HTTP/JSON API	Java API RESTful HTTP/JSON API JDBC
Supported programming languages	.Net C Clojure ColdFusion Erlang Go Java JavaScript (Node.js) Perl PHP Python Ruby Scala Tcl	Bash C C# Java JavaScript (Node.js) Python	C C# C++ Erlang Haskell Java JavaScript Perl PHP Python Ruby Scala Through unofficial drivers : Actionscript Clojure ColdFusion D Dart Delphi Go Groovy Lisp Lua MatLab PowerShell Prolog R Smalltalk	.Net Groovy Java JavaScript Perl PHP Python Ruby	C# C++ Go Java JavaScript (Node.js) PHP Python Erlang (Not officially Supported)
Server-side scripts (Stored Procedures)	View functions in JavaScript	yes	JavaScript	Yes	Java
Triggers	Yes, via the TAP protocol	No	No	Yes, using the 'percolation' feature	No
Partitioning methods	Sharding	Sharding	Sharding	Sharding	Sharding
Replication methods	Master-master replication (including cross data center replication) Master-slave replication	Master-slave replication	Master-slave replication	Yes	Master-master replication Master-slave replication

MapReduce	Yes	No	Yes	ES-Hadoop Connector	No
Consistency concepts	Eventual Consistency Immediate Consistency (selectable on a per-operation basis)	Immediate Consistency only within each node	Eventual Consistency Immediate Consistency (can be individually decided for each write operation)	Eventual Consistency, Synchronous doc based replication. Get by ID may show delays up to 1 sec. Configurable write consistency: one, quorum, all	Strong Consistency
Referential integrity (Foreign keys)	No	No	No (typically not used, however similar functionality with DBRef possible)	No	No (FOREIGN KEY constraints are not supported)
Transaction concepts	No, atomic operations possible only within a single document	ACID (Only isolation level: READ COMMITED)	Multi-document ACID Transactions with snapshot isolation	No	ACID (Transactions are executed single-threaded within stored procedures)
Concurrency	yes	yes, multi-version concurrency control (MVCC)	yes	Yes	Yes, Data access is serialized by the server
Durability	yes	yes	yes (Optional)	Yes	Yes, Snapshots and command logging
In-memory capabilities		yes	Yes, In-memory storage engine introduced with MongoDB version 3.2	Memcached and Redis integration	
User concepts	User and Administrator separation with password-based and LDAP integrated Authentication	fine grained access rights according to SQL-standard	Access rights for users and roles		Users and roles with access to stored procedures
Real-time Capabilities					Yes

Distributed Storage Systems

File storage systems are another solution to deal with large volume of data in distributed environments. The major ones are Google File Storage (GFS) and Hadoop Data File Storage (HDFS).

A. **GFS** is a scalable distributed file system developed by Google to meet the needs of its large distributed data-intensive applications. It is designed for environments that are prone to failures, that manipulate huge data files by frequent read/append operations, and that need to process data in batch rather than in real-time. Thus, it is highly fault-tolerant and reliable, and emphasizes on high throughput rather than low latency.

B. **HDFS** is an open source implementation of GFS. It is part of the Apache Hadoop, an open source framework for distributed storage and distributed processing of large data sets (see below for further details). The biggest clusters implementing Hadoop are composed of 45 000 machines and store up to 25 petabyte of data.

2.4. BIG DATA TOOLS AND APPLICATIONS

Other basic tools and applications will be evaluated in order to be integrated into the FINSEC BIG DATA INFRASTRUCTURE. The following subsections present the more relevant ones.

2.5. Apache Hadoop Framework

Apache Hadoop (http://hadoop.apache.org/) is a collection of open-source software utilities to manage a large network of computers to solve problems involving massive amounts of data and computation. Hadoop is one of the most important frameworks for working with Big Data. Hadoop biggest strength is scalability: it upgrades from working on a single node to thousands of nodes (each offering local computation and storage) without any issue in a seamless manner, allowing the distributed processing of large data sets across clusters of computers using simple programming models. Rather than rely on hardware to deliver high-availability, the framework itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

The framework is an actual set of different modules:

- Hadoop Common: The common utilities that support the other Hadoop modules.
- Hadoop Distributed File System (HDFS): A distributed file system that provides high-throughput access to application data.
- Hadoop YARN: A framework for job scheduling and cluster resource management.
- Hadoop MapReduce: A YARN-based system for parallel processing of large data sets
- Hadoop Ozone: An object store for Hadoop.

Within the major advantages provided by Hadoop it is possible to highlight:

- The framework allows the user to quickly write and test distributed systems. It is efficient, and it automatic distributes the data and work across the machines and in turn, utilizes the underlying parallelism of the CPU cores.
- Hadoop does not rely on hardware to provide fault-tolerance and high availability (FTHA), rather Hadoop library itself has been designed to detect and handle failures at the application layer.
- Servers can be added or removed from the cluster dynamically and Hadoop continues to operate without interruption.
- Hadoop (apart from being open source) is compatible on all the platforms since it is Java based.

Finally, many other Hadoop-related projects have been developed at Apache, which actually realize an ecosystem of tools that further enrich the Hadoop capabilities.

2.6. Apache Storm framework

Apache Storm (http://storm.apache.org) is a distributed real-time big data-processing system. Storm is designed to process vast amount of data in a fault-tolerant and horizontal scalable method. It is a streaming data framework that has the capability of highest ingestion rates. Though Storm is stateless, it manages distributed environment and cluster state via Apache ZooKeeper (https://zookeeper.apache.org/). It is simple and enables to execute all kinds of manipulations on real-time data in parallel. Apache Storm is continuing to be a leader in real-time data analytics. Storm is easy to setup, operate and it guarantees that every message will be processed through the topology at least once.

Storm has many use cases: real-time analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. Storm is fast: a benchmark clocked it at over a million tuples processed per second per node. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.

Within the major advantages provided by Storm it is possible to identify:

- Storm is open source, robust, and user friendly. It could be utilized in small companies as well as large corporations.
- Storm is fault tolerant, flexible, reliable, and supports any programming language.
- Allows real-time stream processing.
- Storm is unbelievably fast because it has enormous power of processing the data.
- Storm can keep up the performance even under increasing load by adding resources linearly. It is highly scalable.
- Storm performs data refresh and end-to-end delivery response in seconds or minutes depends upon the problem. It has very low latency.
- Storm has operational intelligence.
- Storm provides guaranteed data processing even if any of the connected nodes in the cluster die or messages are lost.

It's also relevant to briefly compare the features provided by Hadoop and Storm frameworks. Basically they're used for analysing big data. Both of them complement each other and differ in some aspects. Apache Storm does all the operations except persistency, while Hadoop is good at everything but lags in real-time computation. Storm is designed for Real-time stream processing and it's stateless, while Hadoop for Batch processing and it's stateful. A Storm streaming process can access tens of thousands messages per second on cluster, while Hadoop HDFS uses MapReduce framework to process vast amount of data that takes minutes or hours. Storm topology runs until shutdown by the user or an unexpected unrecoverable failure, while Hadoop MapReduce jobs are executed in a sequential order and completed eventually. Finally, both are distributed and fault-tolerant.

2.7. $H_2$0.AI Framework

H2O.ai (https://www.h2o.ai/) is a framework focused on bringing AI in general and Machine/Deep Learning in particular to businesses through software. Its flagship product is H2O, the leading open source platform that makes it easy for financial services, insurance companies, and healthcare companies to deploy AI and deep learning to solve complex problems. There are several organizations and data scientists using H20.ai for different objectives such as predictive maintenance and operational intelligence.

Using in-memory compression, H2O handles billions of data rows in-memory, even with a small cluster. To make it easier and to create complete analytic workflows, H2O's platform includes interfaces for R, Python, Scala, Java, JSON, and CoffeeScript/JavaScript, as well as a built-in web interface, Flow.

H2O is designed to run in standalone mode, on Hadoop, or within a Spark Cluster, and typically deploys within minutes. It also includes many common machine learning algorithms, such as:

- generalized linear modeling (linear regression, logistic regression, etc.),
- Naive Bayes,
- Principal components analysis,
- k-means clustering

H2O implements best-in-class algorithms at scale, such as distributed random forest, gradient boosting, and deep learning. H2O also includes a Stacked Ensembles method, which finds the optimal combination of a collection of prediction algorithms using a process known as ”stacking.” With H2O, one can build thousands of models and compare the results to get the best predictions.

H2O is nurturing a grassroots movement of physicists, mathematicians, and computer scientists to herald the new wave of discovery with data science by collaborating closely with academic researchers and industrial data scientists. Stanford university giants Stephen Boyd, Trevor Hastie, and Rob Tibshirani advise the H2O team on building scalable machine learning algorithms allowing to improve the performance and results.

2.8. Apache Spark framework

Apache Spark is a recent state-of-the-art map-reduce technology that provides 100x speedup compared with Hadoop. It supports three different execution modes: batch, streaming and structured streaming (Spark Streaming). The batch mode is the most basic and provides API to process a batch of data. Streaming and spark streaming supports creating a streaming pipeline, where the input data arrives as a stream (e.g., Kafka stream) and processed as micro-batches. The main difference between Spark streaming and Structured Spark streaming is that the latter provides API to handle out-of-order events and represents a stream as an infinite table, while the former creates micro-batches to be processed sequentially by the pipeline.

Apache Spark is strongly supported both by academy and industry with a rich ecosystem for processing data from different data sources. It includes Machine Learning library and Graph analytics that simplify creation of scalable analytics by leveraging already implemented algorithms.

The core of Spark is based on Resilient Distributed Data (RDD) and DAG scheduler that provides fault-tolerance in case of cluster nodes failures. Spark provides a higher level Dataframe data structure as an abstraction of tables. Spark application can be developed in Java, Scala, Python or R.

2.9. Apache Kafka

Apache Kafka is an open-source stream-processing software platform aiming to provide a unified, high-throughput, low-latency platform for handling real-time data feeds.

One of the main capabilities of the platform is to publish streams of data or records, like message queue; moreover, it is able to store streams of records in a fault-tolerant way, thus making it a good candidate for security applications in the financial sector. The streams of records can be processed as well. Generally, Kafka is used to build real-time applications or real-time data pipelines between applications.

From a hardware point of view, Kafka runs on a cluster of servers, managing records composed by a key, a value, and a timestamp. The platform adopts 4 different typologies of APIs, allowing the applications to interact with it: the Producer API (used to publish streams of records), the Consumer API (to process them), the Streams API (to transform input streams of records into output records) and the Connector API (to connect Kafka platform to existing applications or data systems, such as Databases).

Clients for Kafka are available in many different languages.

Table 2: comparison between BigData management technologies

Name	***Hadoop***	***Storm***	***H2O***	***Spark***	***Kafka***
Description	Set of tools for management of computer networks for processing large amounts of data.Highly scalable.	Distributed BigData processing system in real time; open source and user friendly, flexible, supports any programming language.	Open source platform for financial services and insurance companies to deploy AI to solve complex problems.	Map-reduce technology speeding up the process if compared with the other ones. Compatible with many different environments.	Open-source stream-processing software platform to handle real-time data streams.
Relevance	Relevant to FINSEC because of its fault tolerance (distributed environment). It is compatible with all the platforms.	FINSEC needs to manage huge amounts of data in a robust manner, with analytics and machine learning functionalities. The fault-tolerant features of Storm make it fully relevant for a security application.	Specific for financial services and insurance companies. Interfaces available with many different environments and a web interface. Includes different machine learning algorithms.	Creation of micro-batches of data starting from a data stream. Includes a machine learning library. Fault tolerance in case a node fails. Compatible with many different environments.	Key-Value based records. Relevant to FINSEC because of its fault tolerance (distributed environment). Compatible with many different environments.
Support	BigData storage (distributed file system, HDFS); BigData processing (Data Collector, YARN);	In FINSEC dataflow, it is relevant for BigData processing (Data Collector), analytics and AI.	In FINSEC dataflow, it is relevant for BigData processing (Data Collector), analytics and AI, Predictive Security.	Data Collector: probes giving input data as a stream that needs to be processed as a batch once stored in the Data Tier. Analytics and AI.	Data Collector: probes giving input data as a stream that needs to be processed
Roadmap	*To be evaluated and integrated in future releases of FINSEC BIG DATA INFRASTRUCTURE depending on specific technical requirements from other tools*				

3. Analysis and Mapping of Requirements for Big Data Infrastructure

3.1. Mapping of requirements from FINSEC Reference Architecture

The FINSEC Reference Architecture (RA) has been designed and detailed in Deliverable 2.4 which represents one of the main input source to define the functionalities, the structure and the technological solutions for the Big Data Infrastructure.

The high-level architecture describes the logical view of the FINSEC Platform to support processes. It presents the different components required to manage the transformation and harmonisation of the valuable datasets from the respective data sources, to their semantic annotation and the metadata generation, to their exploration with dynamic queries and view creation on top of them. Furthermore it offers advanced analytics including state of the art big data focused algorithm execution and sophisticated processing, complemented by advanced visualizations of the analysis results.

Additionally, the FINSEC Reference Architecture outlines all the key features of the logical view for FINSEC platform. In order to have a clearer overview of the drivers for Big Data Infrastructure from the RA, the logical view is depicted in the following diagram.

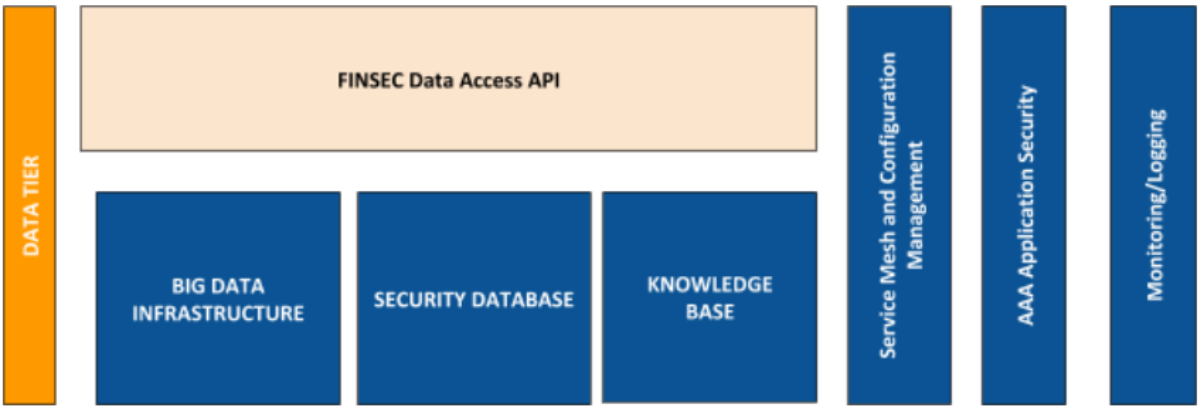

Figure 3: FINSEC Data Tier Architecture

The design of the FINSEC BIG DATA INFRASTRUCTURE was driven by the results of the thorough analysis of all the technical requirements that was conducted with the aim of addressing the goals and the expectations of the FINSEC BIG DATA INFRASTRUCTURE stakeholders. Moreover, the design of the FINSEC BIG DATA INFRASTRUCTURE high-level architecture facilitated the realisation of the designed workflows that enable the data-driven innovation in the FinTech domain as envisioned by the consortium.

A key concept in the FINSEC project is the definition of a consistent Data Model outlined in Deliverable D2.3 and following in internal documents. See APPENDIX (REF FINSEC Reference Data Model).

The whole FINSEC Platform can be conceived as an “intelligent engine” capable of transforming observed data from the physical and digital world (physical-cyber infrastructure) into Threat Intelligence. The information produced will be referred to as **Cyber and Physical Threat Intelligence (CPTI)**. In the same way that Cyber Threat Intelligence (CTI) is valuable information exchanged in the Cyber Security Domain, the CPTI produced in the FinTech sector is the added-value information

produced by the platform which could be exchanged (in-out) between Financial Organizations and Security Organizations (CERT/CSIRT like).

The FINSEC Platform can be considered a transformer of information that, at any step, correlates and aggregates more information along the way (from the data collection to presentation) using observed data and other information gathered from the asset model, the knowledge base using the machine learning analytics and prediction algorithms.

A consistent Data Model is used to represent this information transformation. The FINSEC approach is based on existing standards suited to describe events and threats in either the Physical or the Cyber domain and to extend with the missing part. The FINSEC Data Model is built on objects that can be described with sequences of key-values in an endlessly extensible way.

A consequence is that the data determine and shape the architecture as much as the functionality. The key-values maps naturally into a JSON representation and the JSON representation fits seamlessly into a NoSQL database technology.

Therefore the technology solutions derive from the requirements, the functional specifications and the mapping on the Reference Architecture into the following schema:

Table 3: Mapping of FINSEC requirements from RA

Requirement	Functional Specifications	Mapping in Reference Architecture	Technology solutions provided by Big Data Infrastructure
Ingestion of Physical & Logical Incidents	Import of Data from Physical and Logical Edge/Probes	Data Collector into Security Database	NoSQL DB (MongoDB) and Distributed File System
Knowledge Base and Infrastructure Asset	Import from external KB and Asset Modeling	Knowledge Base import Risk Assessment Engine	NoSQL DB (MongoDB) and Distributed File System
Incident and Threat Detection	Real time Analytics	Security Services and Tools	NoSQL DB (MongoDB) and Distributed File System
Predictive Security	Machine learning on Big Data	Predictive Analytics	Query Engine (Elasticsearch) and Distributed File System
Presentation	Present Data and Threats as user interface	Dashboard	Custom GUI (KIBANA on top Elasticsearch)
Collaboration	Export of Physical Logical Threat Intelligence	Collaboration Tool	Query Engine (Elasticsearch)

4. FINSEC BIG DATA INFRASTRUCTURE

The following section will introduce the specific solutions for the FINSEC BIG DATA INFRASTRUCTURE, a service-oriented application implementing the concept of Data As A Service (DaaS), to provide support to other applications/microservices internal and external to the FINSEC Platform. The FINSEC BIG DATA INFRASTRUCTURE will be based on a set of technologies that meet the business and functional requirements of the project stated in the requirements tasks and analyzed before in the document. To this end, the FINSEC BIG DATA INFRASTRUCTURE supports a data-driven flow from data entry to visualization. The envisioned "core service stack" is depicted in *Figure 4*, which captures the key features of the FINSEC BIG DATA INFRASTRUCTURE.

The FINSEC BIG DATA INFRASTRUCTURE is an efficient and optimised infrastructure management, including all aspects of management for the computing, storage and networking resources, as described before.

The FINSEC BIG DATA INFRASTRUCTURE exploits the underlying core service of Data-driven Infrastructure Management System, to provide a service core for Data as a Service (see figure) in a performant, efficient and scalable way. Data as a Service will incorporate a set of technologies addressing the complete data path: modelling and representation, cleaning, aggregation, and data processing (including seamless analytics, real-time and process mining). Distributed storage will be realised through a layer enabling data to be fragmented/stored according to different access patterns and allowing the efficient expression of that data for database storage and subsequent retrieval. Advanced modelling will be provided to enable the definition of flexible schemas for both data. These schemas will be then utilised by the introduced seamless data analytics framework, which analyses data in a holistic way across multiple data stores and locations, and operates on data irrespective of where and when it arrives at the platform. A cross-stream analytic engine will be provided that can be executed in distributed environments. The engine will consider the latencies across data centres, the locality of data sources and data sinks, and produce a partitioned topology that will maximise the performance.

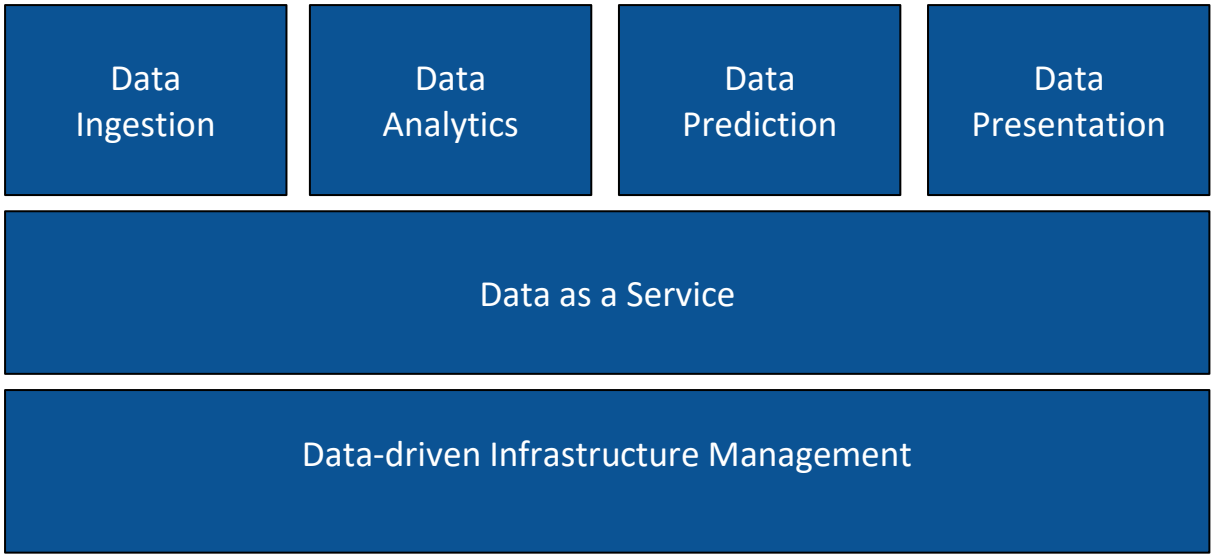

Figure 4: Core Services

The core service for ***Data Ingestion*** *aims at openness and extensibility*. The service will allow the *ingestion of data objects* and the *definition of analytics*, providing at the same time *"hints" towards the infrastructure/cluster management system for the optimised management* of these analytics tasks.

Furthermore, the service will allow probes and analytics engines to *specify requirements and preferences* both for the *infrastructure management* (e.g. application requirements) and for the *data management* (e.g. data quality goals, incremental analytics, information aggregation "levels", etc.).

The core service for ***Data Analytics*** service will provide a *framework allowing for flexible modelling of process analytics* to enable their execution. Functionality-based process modelling will then be concretised to technical-level process mining analytics, while a feedback loop will be implemented towards overall process optimisation and adaptation.

The core service for ***Data Prediction*** aims at enabling the applications of predicting algorithms with the required data services, their interdependencies with the application micro-services and the necessary underlying resources.

Finally, the core service for ***Data Presentation***, going beyond the visualization of data and analytics, outcomes to *adaptable visualisations in an automated way,* according to application analysis and data semantics. Visualizations will cover a wide range of aspects (interlinked if required) besides *data analytics*, such as *computing, storage and networking infrastructure data*, *data sources* information, and *data operations* outcomes (e.g. cleaning outcomes, aggregation outcomes, etc.). Moreover, the FINSEC BIG DATA INFRASTRUCTURE *visualisations will be incremental*, thus providing data analytics results as they are produced.

4.1. Operation of the FINSEC BIG DATA INFRASTRUCTURE

The envisioned operation of FINSEC BIG DATA INFRASTRUCTURE is reflected in five main phases as depicted in *Figure 5* (and further detailed in the following sub-sections): Entry, Modelling, Analytics and Prediction and Presentation.

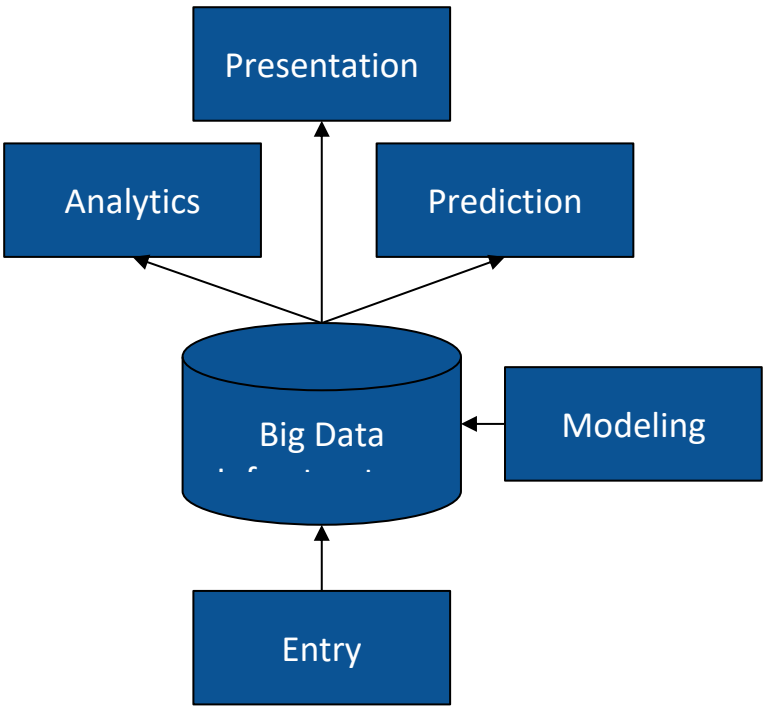

Figure 5: Phases for BigData infrastructure

4.1.1. Entry Phase

During the entry phase, PROBES ingest their data through a DATA COLLECTOR. Thus, the Entry Phase consists of three discrete steps:

- Probes ingest their data in the FINSEC BIG DATA INFRASTRUCTURE-supported data stores through a unified API. They can directly choose if they want to store (non-) relational data or use the FINSEC BIG DATA INFRASTRUCTURE's object storage services. Moreover, historical data can periodically move from the operational database to object storage, keeping only recent data on the database and providing a backup mechanism. Streaming data can also be processed, leveraging the FINSEC BIG DATA INFRASTRUCTURE implementation.
- Given the stored data, Analytics can design processes utilising the intuitive graphical user interface provided by the Process Modelling framework, and the available list of "generic" processes (e.g. customer segmentation process). Overall they compile a business workflow, ready to be mapped to concrete tasks that will be executed. This mapping is performed by a mechanism incorporated in the Process Modelling framework, which is called Process Mapping.
- Based on the outcomes of process, the data services (which constitute the corresponding business workflow) are made available to the analytics through the API. The analytics modules can specify preferences for specific data, for example, how a data service should treat missing values or ingest a complete algorithm in the case this has not been mapped (and as a result made available) by the Process Mapping mechanism.
- The output of the Entry Phase is a set of documents (that includes all relevant information for the application graph with concrete "executable" services) that is passed to the analytics engine in order to identify the resource needs for the services and as a result for the overall application.

4.1.2. Modelling Phase

During the modelling phase, Risk Assessment Engines create their models and stores them into the FINSEC BIG DATA INFRASTRUCTURE. The business processes and the assets will be design by utilising the functionalities of the Process Modelling framework in order to describe the overall business workflows, while analytics can specify their preferences and pose their constraints through the Data service interface.

4.1.3. Analytics Phase

The analytic phase of FINSEC BIG DATA INFRASTRUCTURE aims at optimizing the provision of data services and data-intensive applications by understanding not only their data-related requirements (e.g. related data sources, storage needs, etc.) but also the data services requirements across the data path (e.g. the resources needed for effective data representation, aggregation, etc.) and the interdependencies between application components and data services (both included as processes through the process modelling approach described in the previous paragraph). In this context, dimensioning includes a two-step approach that is realised through the FINSEC BIG DATA INFRASTRUCTURE:

- In the first step, the input from the Data Collection is used to support the composite application (consisting of a set of micro-services) needs with relation to the required data services.
- The second step is to support these identified/required data services, as well as all the application components, regarding their infrastructure resource needs.

4.1.4. Prediction Phase

The prediction of FINSEC BIG DATA INFRASTRUCTURE aims at orchestrating the optimum deployment patterns and practices for both data services and applications. The need for such optimisation emerges from the fact that all services to be deployed have interdependencies that need to be considered, to obtain a practical deployment, as well as to account for the user's preferences (e.g. minimisation of cost).

To this end, the deployment approach of FINSEC BIG DATA INFRASTRUCTURE includes a two-step phase and is realised through the mechanism of the FINSEC BIG DATA INFRASTRUCTURE management system:

- Deployment receives the dimensioning information for components and decides on the overall optimum deployment pattern through a ranking mechanism.
- Following the identification and analysis of the interrelations and the impact of the components, in terms of computation, storage and networking resources (considering data characteristics such as volumes, application components and data services I/O rates, legal constraints, etc.), optimum deployment patterns will be compiled.

4.1.5. Presentation Phase

The presentation service of FINSEC BIG DATA INFRASTRUCTURE is realised through different components (Dashboard, Collaboration Modules) and aims at the visualization and transport of the complete data resources, in an optimised way for data-intensive applications.

4.2. FINSEC Big Data Infrastructure Architecture

According to D2.4, the **Data Tier** is the logical layer where information are stored and is organized into different storage infrastructures, providing consisting data access API to all other modules. Quoting D2.4

> *The **Data Tier** provides an infrastructure to serve data that follow in the **FINSEC REFERENCE DATA MODEL** (defined by the project in tasks T2.3 and T2.4). It provides access in read/write via a Data Access API, exposed by an ad-hoc service of the platform (Data Manager). This module exposes convenient data access and manipulation functions to clients, is responsible for ensuring validation of input data against the data model and abstracts away the actual underlying DB engine(s), which can be changed without affecting upper-level services.*

As depicted in the *Figure 3*, the F**INSEC BIG DATA INFRASTRUCTURE** will provide a ***complete infrastructure big data management system*** accomplished by a seamless and consistent API interface.

The Data Tier will be used as a service from other applications both external and internal to the FINSEC Platform, outlined in D2.3. In this respect it should be considered as a Security Data as a Service and could also be a general module used in other contest. Therefore it is designed with generality by design.

Conceptually, data flows into the data tier from the external world, coming from probes and other applications/microservices (e.g. the import module of a compatible Knowledge Base of Cyber Threat Information). These modules produces Security Data that will be consumed from other modules in the platform, for further elaboration. However the complex nature of the data, the potentially large quantities to be stored and the need to process the data for analytics, predictions and visualization

requires into the data tier the presence of a specialized search engine that scales on a big data infrastructure.

The FINSEC BIG DATA INFRASTRUCTURE is designed, in the following, as a modular architecture composed of multiple key components, where each component was designed with a clear business context, scope and set of functionalities. As the project matured after the initial version of the high-level architecture, additional functionalities were designed and introduced in the platform. Moreover, as a result of the comprehensive analysis of the feedback received by the end-users of the platform from the released versions of the platform, a series of adjustments and refinements were introduced in the components of the platform in order to better address the identified requirements, but also to facilitate the implementation of the functionalities of the platform.

The following picture depicts the illustrated scenario.

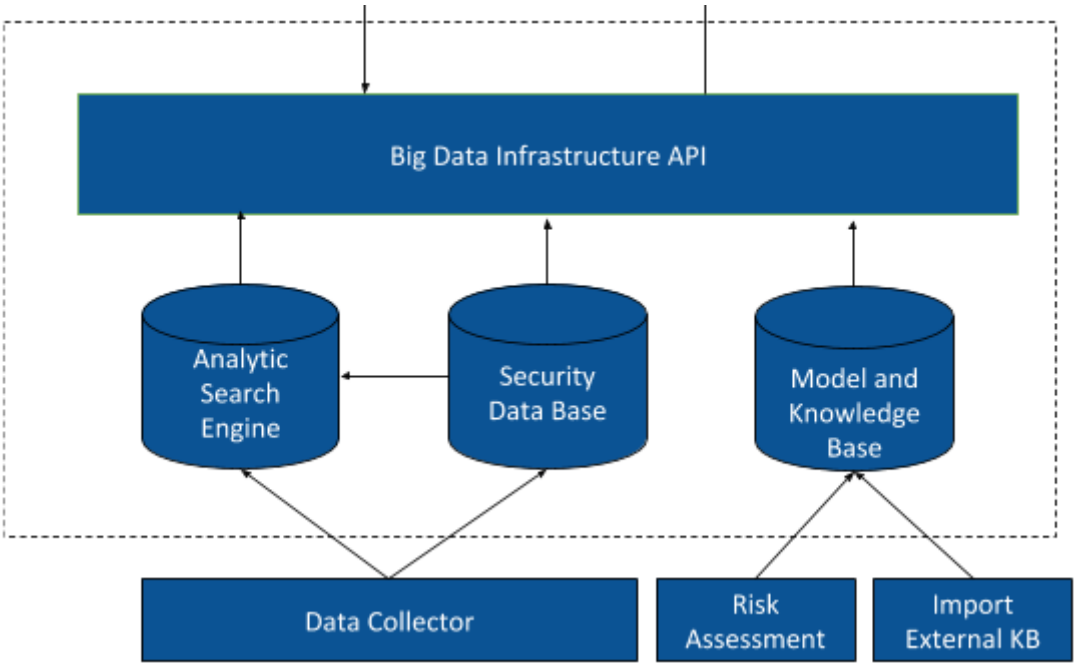

Figure 6: FINSEC BigData Architecture

The depicted FINSEC BIG DATA INFRASTRUCTURE high-level architecture, incorporates all the adjustments and refinements that were introduced in the course of development of the FINSEC BIG DATA INFRASTRUCTURE platform. This architecture will drive the implementation and the release of the FINSEC BIG DATA INFRASTRUCTURE Minimal Viable Platform (MVP).

In the MVP, all probes will feed the Data Collector which in turn will use the ingestion/create methods of the FINSEC BIG DATA INFRASTRUCTURE Security Database.

4.3. Building Block Structure and Functionalities

4.3.1. Security Database (MONGODB)

The FINSEC BIG DATA INFRASTRUCTURE provides a NoSQL database to store data coming from the field via the probes and Data Collector. The actual choice according to the analysis presented in the previous sections is to use a NoSQL database such as MongoDB.

Moreover MongoDB also offers GridFS specification for storing and retrieving large files that exceed the BSON-document size limit of 16MB.

In MongoDB, databases hold collections of documents. Data is stored in these documents in a binary representation known as Binary JSON (BSON). Every document has a unique key, “id”, in a collection. Collections in MongoDB have Dynamic schemas. Thus, a different “shapes” of documents can be stored within a single collection.

GridFS stores large binary files by dividing the files into smaller files called “chunks” and saving each of them as a separate document. GridFS limits a default chunk size to 255 kB, thus enabling efficient file handling operations regardless of the file size.

As shown in the *Figure 7*, GridFS uses two collections to save a file to a database: fs.chunks and fs.files . The first one contains the binary file divided into 255kB chunks while the other collection contains the metadata for the document.

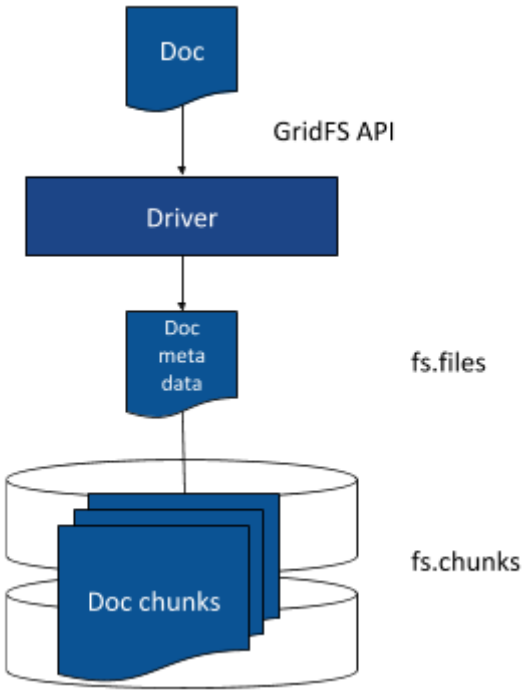

Figure 7: GridFS structure

There are several reasons that might lead an organization to the decision of storing the binary data in the same system as the metadata instead of storing it in a separate repository. These include:

- The resulting application will have a simpler architecture: one system for all types of data;
- Document metadata can be expressed using the rich flexible document structure, and documents can be retrieved using all the flexibility of MongoDB’s query language;
- MongoDB’s high availability (replica sets) and scalability (sharding) infrastructure can be leveraged for binary data as well as the structured data;
- One consistent security model for authenticating and authorizing access to the metadata and files;
- GridFS doesn’t have the limitations of some file systems, like number of documents per

directory, or file naming rules.

Using MongoDB GridFS as a solution for handling unstructured data is a better option because data is stored inside the database, which is highly scalable and designed for horizontal partitioning. Another reason in favor of using MongoDB GridFS as a solution to this problem is that horizontal partitioning combined with replication capabilities enables use of simultaneous reading of data from several database servers that serve as replication servers. The simultaneous reading capability greatly increases throughput performance of the system.

4.3.2. Security DB API

The Security DB of the FINSEC BIG DATA INFRASTRUCTURE will supports four basic operations, Create, Read, Update and Delete (CRUD). These operations can be used for inserting and retrieving data formatted according to the FINSEC Data Model (FINSTIX see appendix).
The Security DB will be implemented in MongoDB and the rest of example will use the API Compass. The rest of this section, presents an overview on these operations.

1) Create Operation

In FINSEC BIG DATA INFRASTRUCTURE will, insert operations target a single collection. All write operations are atomic on the level of a single document.

FINSEC BIG DATA INFRASTRUCTURE will provide two methods to insert a document into a collection:

```
db.finstix.insertOne()  # Inserts one document
db.finstix.insertMany() # Inserts multiple documents
```

Example:

```
db.finstix.insertOne(    // Collection
{                        // FINSTIX Document
   "type": "x-finstix",
   "id": "x-finstix--998fcca5-06c7-4c5d-98d7-3c966599fc94",
   "created": "2019-02-06T18:13:36.140Z",
   "modified": "2019-02-06T18:13:36.140Z",
   "subtype": "Physical",
   "description": "Attack to Person",
   "parameter": "Confidentiality 50%",
   "narrative": "attack with knife and gun",
   "priority": "7",
   "organization": "Wirecard",
   "asset": "ATM #6789",
   "position": "45.490946+9.228516",
   "probe": "FUJITSU CCTV"
}
)
```

2) Read Operation

Read operations retrieve documents from collections.

To query a collection for documents, FINSEC BIG DATA INFRASTRUCTURE will provide basic methods to query/find an object into a collection:

```
db.finstix.find(query, projection)
```

Where query specifies the criteria of the documents to return, and projection specifies the fields to return in the documents that match the query criteria.

Example:

```
db.finstix.find( // find finstix object in Collection
   {created: $gt: 2018-01-01},    // query criteria
    asset: ,
    address: 1}
   ).limit(5)
```

The query uses $gt to return,events created after a certain date and uses the method limit(5) to set the maximum number of returned documents to 5.

3) Update Operation

Update operations modify existing documents in a collection. FINSEC BIG DATA INFRASTRUCTURE will provides the following methods to update documents of a collection:

```
db.finstix.updateOne()  # Modifies one document
db.finstix.updateMany() # Modifies multiple documents
db.finstix.replaceOne() # Replaces one document
```

4) Delete Operation

Delete operations remove documents from a collection. FINSEC BIG DATA INFRASTRUCTURE will provides the following methods to delete documents of a collection :

```
db.finstix.deleteOne() Deletes one document
db.finstix.deleteMany() Deletes multiple documents
```

Scalable Infrastructure

In order to meet the needs of applications with large data sets and high throughput requirements, the FINSEC BIG DATA INFRASTRUCTURE implemented with MongoDB, provides horizontal scale-out for databases using a technique called sharding. Sharding allows MongoDB deployments to scale beyond the limitations of a single server and it does this without adding complexity to the application. It automatically divides and distributes data across multiple servers or shards. Each shard is backed by a replica set to provide always-on availability and workload isolation. To respond to workload demand, nodes can be added or removed from the cluster in real time, and MongoDB will automatically rebalance the data accordingly, without manual intervention. MongoDB supports sharding through the configuration of a sharded cluster, which is composed of three major components: Shards, Query routers and Configuration servers.

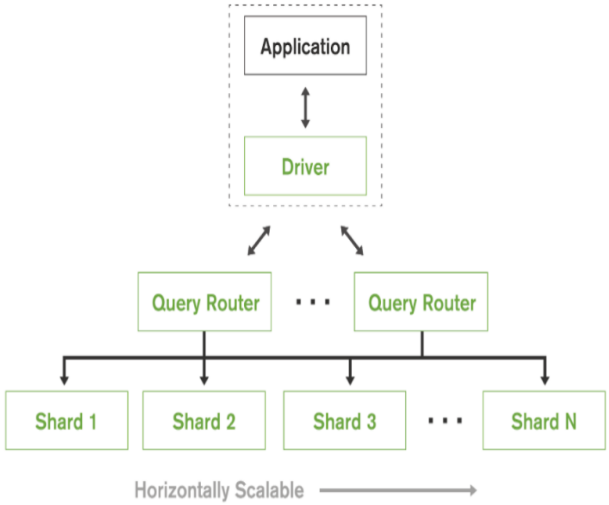

Figure 8: Automatic sharding for horizontal scale-out

In the sharded cluster, each shard contains a subset of the sharded data and can be deployed as a replica set. The query router directs requests from the application to the appropriate shard and also returns the result back to the client. These are "mongos" instances and there can be more than one in a cluster. Multiple mongos instances reduce the request load from client. The third component in a sharded cluster is the configuration servers. They have metadata about all the shards and hence, help the query router to direct different operations to specific shards. *Figure 9* shows an overview of a sharded cluster in MongoDB.

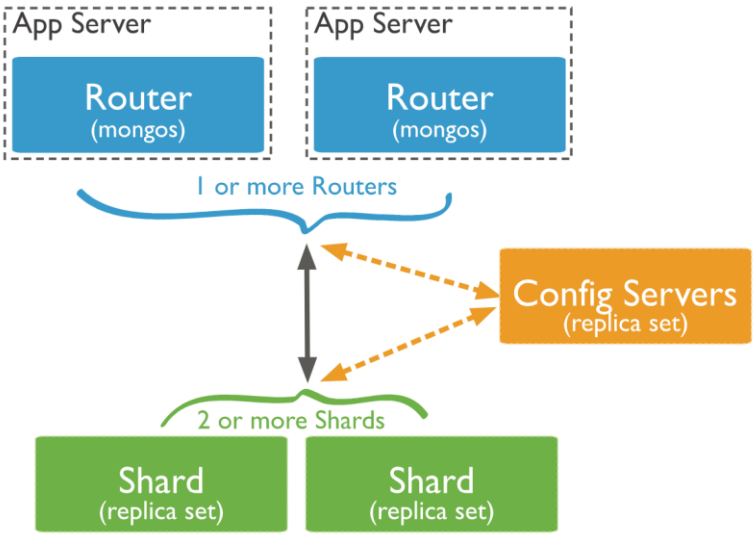

Figure 9: A sharded cluster in MongoDB

Sharding in MongoDB is on collection level. Data of a collection is partitioned by a shard key. Shard keys are indexed fields existing in every document in the collection. Shard key values are divided into groups, also known as chunks and distributed evenly across the shards. MongoDB offers multiple sharding policies that enable developers and administrators to distribute data across a cluster according to query patterns or data locality. As a result, MongoDB delivers much higher scalability across a diverse set of workloads:

- **Ranged Sharding**. Documents are partitioned across shards according to the shard key value.

Documents with shard key values close to one another are likely to be co-located on the same shard. This approach is well suited for applications that need to optimize range based queries, such as co-locating data for all customers in a specific region on a specific shard.

- **Hashed Sharding**. Documents are distributed according to an MD5 hash of the shard key value. This approach guarantees a uniform distribution of writes across shards, which is often optimal for ingesting streams of time-series and event data.

- **Zoned Sharding**. allows precise control over where data is physically stored in a cluster. This allows developers to accommodate a range of application needs. For instance, controlling data placement by geographic region for latency and governance requirements, or by hardware configuration and application feature to meet a specific class of service.

4.3.3. ANALYTICS SEARCH ENGINE

The FINSEC BIG DATA INFRASTRUCTURE provides an analytic search engine to work on stored data coming from probes and the Security Database. The actual choice according to the analysis presented in the previous sections is Elasticsearch. Thus, Elasticsearch complements the NoSQL database (MongoDB) as a search engine to allow scalable and near real-time search on the data. The advantages of such a solution is the ability to reliably store documents and perform simple full-text search queries in MongoDB along with the extensive functionalities, customizability and speed for performing complex full-text search queries in Elasticsearch.

Elasticsearch was initially developed as a system for full text search in large volumes of unstructured data. At present, Elasticsearch is a full-fledged analytical system with various capabilities. Its main strengths are exceptional and reliable speed, very high customizability and outstanding flexibility.

In order to accomplish sophisticated searches, Elasticsearch provides the two key features Query and Filter. There exist multiple types of queries, the most important being Full-text, Term, Match, and Prefix. The Geo filter is also one of most important features of Elasticsearch. Since Elasticsearch is built on top of Lucene, Elasticsearch makes use of all features provided by Lucene and extends them by providing additional features. The Query Domain Specific Language (Query DSL) can be used in order to support the creation of advanced queries of Elasticsearch. Elasticsearch uses the Inverted index structure for allowing fast full-text searches.

The search in Elasticsearch is near real-time. This means that although documents are indexed immediately after they are successfully added to an index, they do not appear in the search results until the index is refreshed. Elasticsearch does not refresh the indices after each update. Instead it makes the use of a specified time interval, also called refresh interval, to perform this operation. By default, the refresh interval is one second. Since refreshing is costly in terms of disk I/O, it can affect the indexing performance. For that reason, increasing the refresh interval before updating a large number of documents is useful. Elasticsearch provides a Search API supports GET and POST methods and enables to search across multiple indices. However, more complex searches can be accomplished by using the Query DSL which allows for using Queries and Filters.

In the context of data analysis, Elasticsearch is used together with other components such as Logstash and Kibana, and plays the role of data indexing and storage.

Scalability of the search Engine:

As Elasticsearch has a distributed architecture it enables to scale up to thousands of servers and accommodate petabytes of data. It provides the ability to subdivide the index into multiple pieces called shards, and its structure is optimized for fast and efficient full-text searching. Shards come in two types, master and replica. The master shard allows both read and write operations, while the replica is read only, and is an exact copy of the master. Such a structure ensures the stability of the system, since in the event of a master failure, the replica becomes a master.

The advantage of sharding feature is that it allows horizontal scaling of the content volume and improves the performance of Elasticsearch by providing parallel operations across various shards that are distributed on nodes. Each primary and replica shard is built of multiple segments. Elasticsearch makes the use of segment merging for reducing the number of segments in order to allow faster searching and for reducing the size of the index because of removing deleted documents when the merge is finalized. *Figure 10* illustrates an example of an Elasticsearch cluster. The cluster consists of two nodes with four primary shards and four replica shards. As shown in the figure, replica shards reside on different nodes than the primary shards in order to help in case of primary shard failure and for reasons of load balancing of incoming requests (i.e., Replica shard 1, which belongs to the Primary shard 1, resides on Node 2, whereas the primary resides on Node 1).

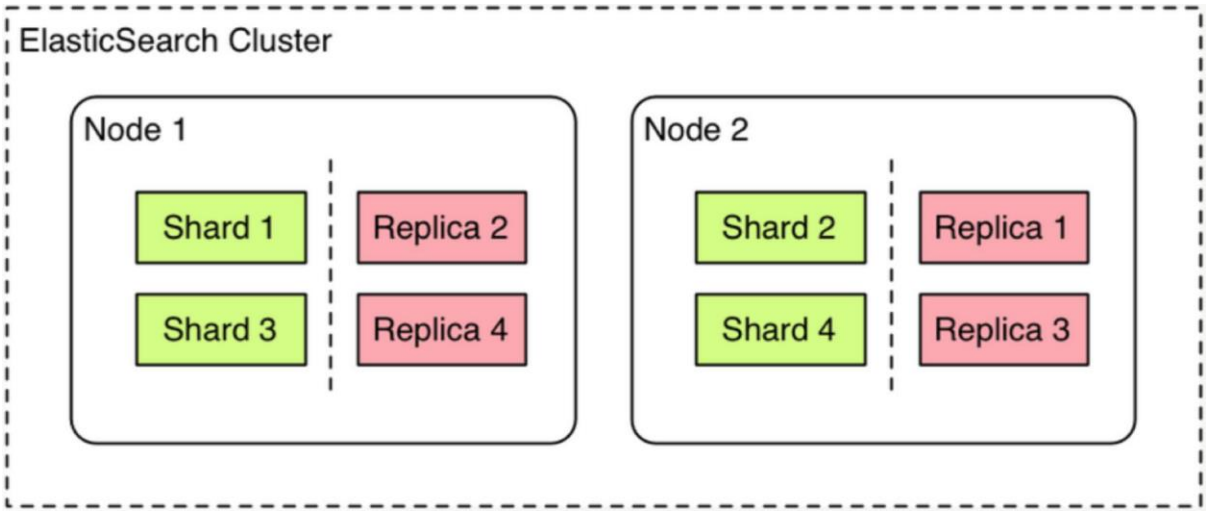

Figure 10: Example of an Elasticsearch cluster

4.3.4. Synchronization with Mongodb

There are different ways to synchronize data from MongoDB to Elasticsearch (e.g., Logstash, Mongoosastic, Transporter, Monstache, etc) but one easy option is to use Mongo-Connector.

Mongo-Connector is a generic connection system that can be used for connecting MongoDB to search engines such as Solr or Elasticsearch for more advanced search. It copies the documents stored in MongoDB to the target system. Afterwards, it constantly performs updates on the target system to keep MongoDB and the target synchronized. The connector supports both Sharded Clusters and standalone Replica Sets, hiding the internal complexities such as rollbacks and chunk migrations.

Figure 11 illustrates how data can be synchronized using mongo-connector, which requires mongoDB to run in **replica-set mode**. It synchronizes data in mongoDB to the target then tails the mongoDB oplog, keeping up with operations in mongoDB in real-time. A package named "**elastic2_doc_manager**" is also required in order to write data to Elasticsearch.

From version 2.0, mongo-connector can also replicate files stored in GridFS to Elasticsearch using the attachment mapping type.

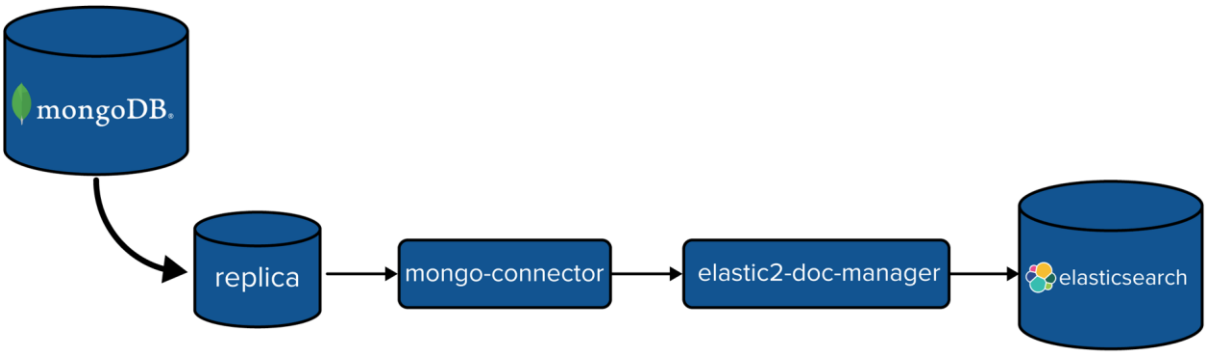

Figure 11: data synchronization through mongo-connector

4.3.5. ELASTICSEARCH BASIC OPERATIONS

The usage of the RESTful API is quite simple: it expects JSON encoded parameters, and can be accessed using HTTP. The returned results are also encoded in JSON. The RESTful API supports requests in order to manage the index, check the server health, update the node, search data, and manage the cluster. Since REST is built upon HTTP protocol, it supports all methods of HTTP like GET, PUT, POST, DELETE, and so on. By default, Elasticsearch does not provide any authentication or authorization method to its REST API. However, the chargeable Elasticsearch plugin Shield provides functionalities for encrypting communications and a role-based access control. The REST API provides the speeding up of atomic operations with the Bulk API. It allows to make multiple create, read, update, and delete requests of documents at once. The rest of this section, presents an overview on basic CRUD APIs.

Index API

The index API adds or updates a typed JSON document in a specific index, making it searchable. The following example inserts the JSON document into the "finstix" index, under a type called " x-finstix " with an id of "x-finstix—998fcca5-06c7-4c5d-98d7-3c966599fc94" :

```
PUT finstix/x-finstix/x-finstix—998fcca5-06c7-4c5d-98d7-3c966599fc94

{    "created": "2019-02-06T18:13:36.140Z",
     "modified": "2019-02-06T18:13:36.140Z",
     "subtype": "Physical",
     "description": "Attack to Person",
     "parameter": "Confidentiality 50%",
     "narrative": "attack with knife and gun",
     "priority": "7",
     "organization": "Wirecard",
     "asset": "ATM #6789",
     "position": "45.490946+9.228516",
     "probe": "FUJITSU CCTV"
}
```

Get API

The get API allows to get a typed JSON document from the index based on its id. The following example gets a JSON document from an index called finstix, under a type called x-finstix, with id valued x-finstix—998fcca5-06c7-4c5d-98d7-3c966599fc94:

```
GET finstix/x-finstix/x-finstix—998fcca5-06c7-4c5d-98d7-3c966599fc94
```

Delete API

The delete API allows to delete a typed JSON document from a specific index based on its id. The following example deletes the JSON document from an index called finstix, under a type called x-finstix, with id x-finstix—998fcca5-06c7-4c5d-98d7-3c966599fc94:

```
DELETE /finstix/x-finstix/x-finstix—998fcca5-06c7-4c5d-98d7-3c966599fc94
```

Update API

The update API allows to update a document based on a script provided. The operation gets the document from the index, runs the script, and indexes back the result. The following example add a new field to the document:

```
POST finstix/x-finstix/x-finstix—998fcca5-06c7-4c5d-98d7-3c966599fc94/_update
{
    "script" : "ctx._source.new_field = 'value_of_new_field'"
}
```

Queries and Filters

Elasticsearch provides a REST API and clients for several programming languages that support a flexible query language named 'Query DSL'. Although, it is denoted as a "Query" DSL, it also contains a "Filter" DSL. A search can be performed in two ways: in a form of a query or in a form of a filter. The main difference between them is that a query calculates a relevance score of the returned documents whereas the filter does not. Due to this, and the fact that filter can be cached, searching via filters is faster than via queries. A filter asks a yes/no question of every documents, whereas the query also asks the question: 'How well does this document match?' In Elasticsearch, there are different types of queries, like basic queries, compound queries, full-text search queries, and pattern queries, to name a few. Basic queries allow for searching for a part of the index. Furthermore, they allow for nesting other queries inside the basic query. Compound queries allow combining multiple queries or filters inside them. Full-text search queries support full-text searching, analyzing their content and providing Lucene query syntax. Last but not least, pattern queries support various wildcards in queries. Basic queries include e.g., term, match, and indices queries. The match query can be also categorized to the group of full-text search queries. This also applies to the prefix query that can be included to the group of pattern queries.

Below is an example of query clauses being used in query and filter context in the **search API**. This query will match documents where all of the following conditions are met:

The **description** field contains the word attack.

The **narrative** field contains the word gun.

The **subtype** field contains the exact word Physical.

The **created** field contains a date from 06 Feb 2019 onwards.

```
GET /_search
{
  "query": {
       "bool": {
       "must": [
         { "match": { "description":   "attack"        }},
            { "match": { "narrative": "gun" }}
       ],
       "filter": [
```

```
        { "term":  { "subtype": "Physical" }},
        { "range": { "created": { "gte": "2019-02-06" }}}
         ]
       }
  }
}
```

4.5. FINSEC BIG DATA INFRASTRUCTURE TECHNICAL INTERFACE

As an example, the following table give the technical specification of the RestAPI interface to the Entry module into the FINSEC Big Data INFRASTRUCTURE.

Table 4: Create DB interface

Technical Interface	
Reference Code	DB01#01
Function	Upload the dataset
Subsystems	MongoDB- DATABASE
Type, State	RESTful-API
Endpoint URI	<server url:9009>/ingest/
Input Data	FINSTIX JSON format
Output Data	200 OK - NOT OK

Table 5: Create DB interface

Technical Interface	
Reference Code	DB01#02
Function	Upload the dataset
Subsystems	MongoDB- DATABASE
Type, State	RESTful-API
Endpoint URI	<server url:9009>/ingestmulti/
Input Data	FINSTIX JSON format
Output Data	200 OK - NOT OK

Table 6: Create DB interface

Technical Interface	
Reference Code	DB01#03
Function	Upload the dataset
Subsystems	MongoDB- DATABASE
Type, State	RESTful-API
Endpoint URI	<server url:9009>/find/
Input Data	FINSTIX JSON format
Output Data	200 OK - NOT OK

5. Implementation and Integration Aspects

5.1. Deployment Infrastructure

FINSEC will support the deployment of the Big Data Infrastructure components as part of the FINSEC Core, which will be available as a Kubernetes application that can be installed both on-premises and in the cloud.

Kubernetes is the leading open source container orchestration platform and allows to leverage all the advantages of a modern container-based solution, including ease of distribution via prepackaged Docker images, automated deployment of complex systems using manifests and simple horizontal scalability. Furthermore, using Kubernetes as the deployment platform allows to abstract from the actual underlying infrastructure, which may be virtual or physical, on-premises or cloud-based. Most public cloud providers offer managed Kubernetes clusters, including Amazon EKS, Azure AKS, Google GKE and DigitalOcean Kubernetes. These allow to provision a ready-to-use Kubernetes cluster with no configuration effort. On-premises Kubernetes solutions include the community-supported open source version, which can be installed with tools such as kubeadm or Kubespray, and commercially supported distributions, such as Red Hat OpenShift Container Platform.

Two important requirements for Big Data solutions are scalability and storage. Horizontal scalability is easy to achieve in a container cluster, since the number of desired instances for each container can be changed dynamically with API commands or even auto-scaling rules, e.g. based on CPU utilization. Regarding storage, while early versions of container platforms did not have good support for persistent storage, recent versions of Kubernetes can use a wide range of storage plugins for persistent volumes, from traditional Fibre Channel LUNs and iSCSI, to distributed storage such as GlusterFS and block storage from the main cloud providers. Several NoSQL and Big Data components, including MongoDB and Elasticsearch, publish official ready-to-use Docker images, which make it easy to deploy them on a container cluster.

Regarding hosting in the cloud or on-premises, FINSEC will support both scenarios, as shown in *Figure 12*. The integrated development and testing environment will be hosted on the Digital Ocean public cloud, leveraging their managed Kubernetes offering. This will allow flexibility in allocating resources on an as-needed basis and avoid concerns about physical infrastructure provisioning and management. On the other hand, pilots will have the option to choose between a cloud plus edge deployment or a fully on-premises deployment. In the first scenario, the FINSEC Core for the pilot will be deployed in a dedicated environment in the Digital Ocean cloud, with only probes running in the edge, i.e. the pilot organization data center. This will benefit from fully delegating the deployment of the FINSEC Core to the FINSEC DevOps tools, thanks to tight integration on the same cloud platform. In the second scenario, all FINSEC components will be hosted on the pilot organization own infrastructure. This will still benefit from the automated deployment using Kubernetes manifests, but it will require setting up a local Kubernetes cluster (if not already available) and possibly tuning the deployment configuration to adapt to the local infrastructure, such as using proper drivers for persistent storage.

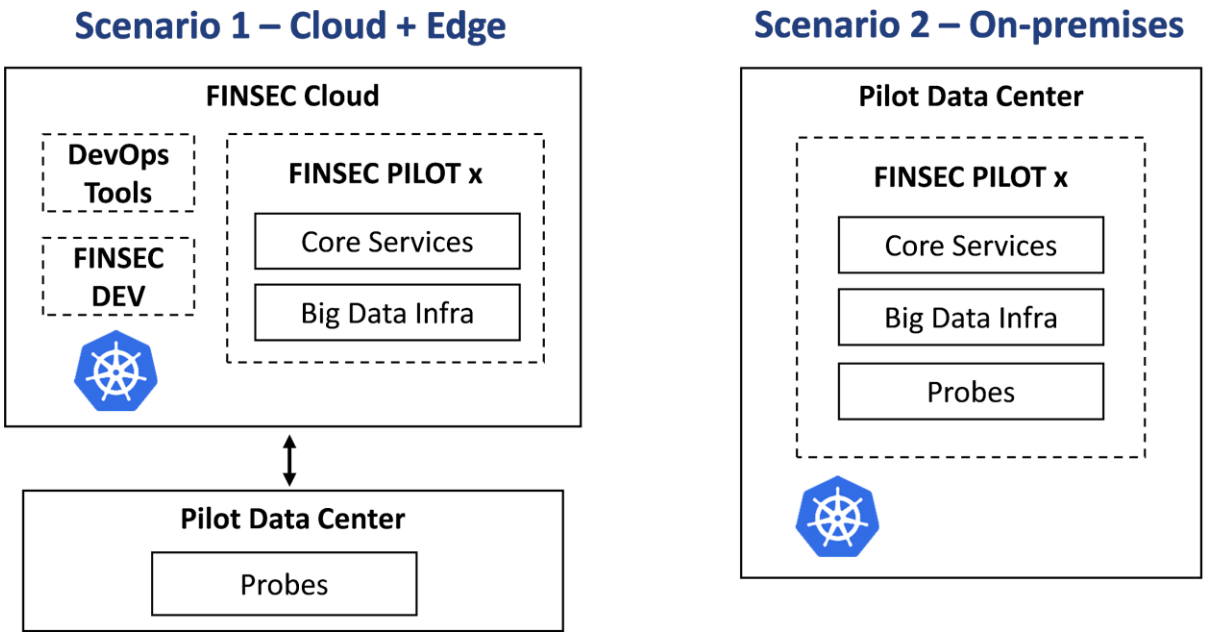

Figure 12: Deployment scenarios

5.2. FINSEC technologies to be integrated with the BigData Infrastructure

The integration includes a seven-step process:

- Based on the deployment phase, outcomes regarding the optimised deployment pattern, computing resources are reserved and allocated.
- According to the allocated computing resources, storage resources are also reserved and allocated. It should be noted that storage resources are distributed.
- Data-driven networking functions are compiled and deployed to facilitate the diverse networking needs between different computing and storage resources.
- The application components and data services are deployed and orchestrated based on "combined" data and application-aware deployment patterns. An envisioned orchestrator mechanism will compile the corresponding orchestration rules according to the deployment patterns and the reserved computing, storage and network resources.
- Data analytics tasks will be distributed across the different data stores to perform the corresponding analytics, while orchestration of application components and data services is also performed.
- Monitoring data is collected and evaluated for the resources (computing, storage and network), application components and data services and functions (e.g. query execution status).
- Runtime adaptations take place for all elements of the environment including resource re-allocation, storage and analytics re-distribution, re-compilation of network functions and deployment patterns.

5.2.1. Anomaly Detection

Anomaly detection is a family of analytics techniques that learn typical properties of the system and reports significant deviations from the typical system's properties as outliers. Anomaly detection is frequently used in the state-of-the-art Intrusion Detection Systems (IDSs) because it provides a protection of the system from new zero-day attacks whenever these attacks deviate from typical behaviors of the system. Another advantage of Anomaly detection techniques is that they don't require a balanced training set in which both malicious and benign events are equally represented. These techniques are a better fit for real industrial system where malicious events are much more rare than benign events.

There is a wide range of Anomaly Detection techniques including statistical methods, clustering methods, time series analysis and recent techniques based on deep neural network. In FINSEC we will deploy a number scalable adaptive Anomaly Detection analytics as a cloud service. The architecture of Anomaly Detection component is depicted below.

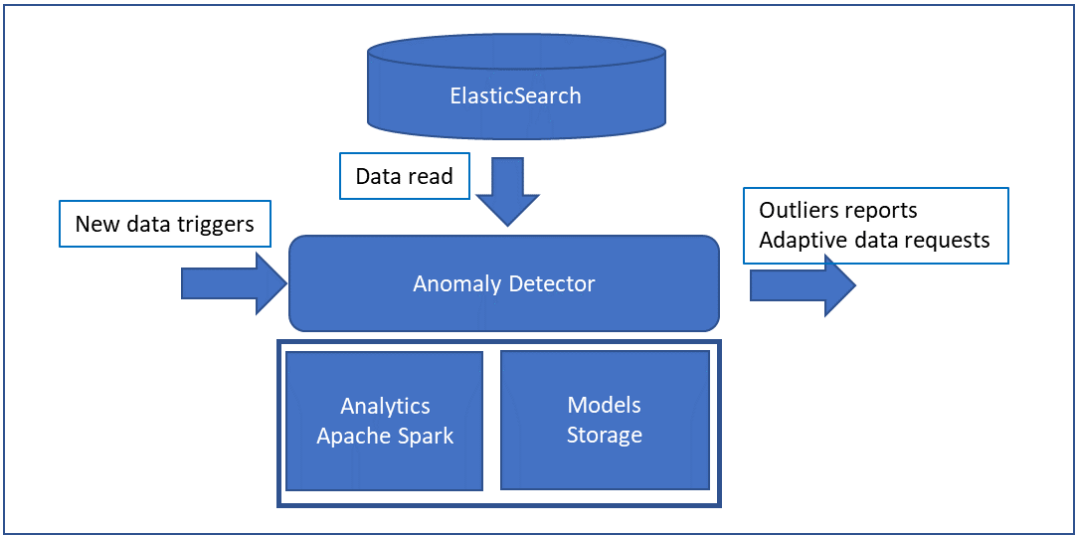

Figure 13: Anomaly Detection component architecture

For scalability we will use an Apache Spark platform which is the most recent state-of-the-art of map-reduce platform with a rich set of machine learning libraries optimized for Big Data setups. The development will be python based to allow to leverage and evaluate publicly available state-of-the-art machine learning techniques as part of in the rich python eco-system. The data will be periodically read from ElasticSearch or other data sources according to received data triggers and processed both for training and analysis in Spark. The trained models will be stored in a persistent storage e.g., HDFS. The output of anomaly detector will be the reports of the detected outliers or adaptive data requests. The data triggers and output of anomaly detector will be distributed using Kafka protocol.

For adaptivity we will use a number of techniques, including online training techniques. For these methods the models are adaptively updated for new system inputs. A simple example of such a technique is the exponential smoothing average which enables continuous updates of mean values and the corresponding standard deviation values of system features. A more advanced example will be estimation of conditional density of next observation of a signal given a previous time window either by directly estimating some signal statistics or by applying LSTM deep learning techniques.

Another example of an adaptive algorithm is an alert budgeting system. An alert budgeting system aims to adaptively set thresholds above which alerts are generated. Alert budgeting systems will

automatically adapt the thresholds according to the recent system behavior to make sure that in average the security officer doesn't need to handle more than a predefined amount of alerts. This method adapts to the nature of the data and has the added benefit of allowing the security officer to configure an 'easy to grasp' parameter like "The amount of alerts that can be handled by a human operator during a day" and not an obscure threshold level number.

FINSEC Anomaly detection analytics will be tuned, trained and validated using data provided by FINSEC partners. Anomaly detection component will report detected outliers along with an anomaly score and an additional contextual info of the triggered outlier.

6. Conclusions

The present document has provided the report of the activities done so far within Task 5.1 on the definition and design of FINSEC BIG DATA INFRASTRUCTURE. An overview of the technological state of the art was given at the beginning of the deliverable, with the comparison between NoSQL and NewSQL solutions. According to the need of scalability, a NoSQl solution is identified as the best one. MongoDB is the most suitable technology among the analysed ones, and it will be supported by analytics engine Elasticsearch, supporting queries for data search within the database on JSON-based data, as foreseen by the proposed FINSTIX data model.

The BigData infrastructure is thought in a Data as a Service perspective, where a set of technologies are integrated together to address the complete data path: modelling and representation, cleaning, aggregation, and data processing, and then the operation of FINSEC BIG DATA INFRASTRUCTURE is reflected in four main phases, namely Entry, Modeling, Analytics and Prediction and Presentation.

The infrastructure is a modular architecture, whose deployment and integration will be based on the Kubernetes technology, which is able to support storage functionalities as well.

The following activities within Task 5.1 will be focused on the consolidation of the proposed design and on the implementation, integration and deployment of the FINSEC BIG DATA INFRASTRUCTURE. Deliverable 5.2 (to be submitted in M18) will represent the final report for this task on the results of the infrastructure implementation.

ANNEX A - FINSTIX Data Model

FINSTIX BASIC CONCEPTS

1. The FINSEC Data Model is an extension of STIX2
2. It will be called FINSTIX
3. FINSTIX extends STIX2 into the physical and logical domain
4. FINSTIX Data Model basic object is a sequence of key-values that can be passed as JSON
5. FINSTIX Data Model general object is an aggregate of more objects and relations still expressed in JSON
6. FINSTIX will be also specific including information relevant to the financial sector
7. FINSTIX defines other objects and relations to STIX2 to cope with the correlation of physical and logical data
8. Probes generate Observed Data, Events, Incidents, Logs (observed data) according to the FINSTIX Data Model
9. Data Collectors (DC) have the function to gather data from probes normalizing, sanitizing, prioritizing and storing CPTI into the Data Layer. In other words, a DC knows the syntax-semantic and add or subtract further information to the FINSTIX objects passing through.
10. FINSTIX objects coming from Data Collector are stored into the DATA LAYER of the FINSEC platform.
11. Asset Model (AM) and Knowledge Base (KB) are represented with FINSTIX objects as well.
12. The Analytics/Predictive algorithms use events, observed data, the Knowledge base and Asset Models to produce Cyber Physical Threat Intelligence (CPTI vs CTI).

Table 7: FINSTIX objects

Source	Type	Description with difference with STIX	Key-Value (JSON like)
ALL	GENERIC	The generic object. All objects MUST have the basic properties described on the right.	{ "type": "object type", "id": "typexxx--8e2e2d2b-17d4-4cbf-938f-98ee46b3cd3f", "created": "e.g. 2019-02-06T20:03:00.000Z", "modified": "e.g 2019-02-08T18:13:36.140Z", "subtype": "Physical or Logical or both", "name": "name of object", "description": "short description" "parameter": "",

The research leading to these results has received funding from the European Union's Horizon 2020 Research and Innovation Programme, under grant agreement no 700071.

			"narrative": "long description ", "priority": "1-10", "organization": "Company name", "asset": "asset name", "position": "polar coordinates", "probe": "Probe name" }

PROBE	Observed Data	Conveys information observed on infrastructure both logical and physical assets.
		
		Typically used to contain:
		● Log
		● Events
		● Incidents
		for both logical and physical assets.
		
		In FINSTIX, and in our specific case and situation, one can manipulate and modify the three attributes **"first_observed"**, "**last_observed**" and "**number_observed**" to enlarge the collected data volume when necessary allowing more accuracy when processing the data.
			{
			"type": "object type",
			"id": "typexxx--uuid",
			"created": "2018-04-06T20:03:00.000Z",
			"modified": "2019-02-06T18:13:36.140Z",
			"subtype": "Physical or Logical or both",
			"name": "Probe name",
			"description": "Lorem Ipsum"
			"parameter": "",
			"narrative": "attack with knife and gun",
			"priority": "7",
			"organization": "organization--uuid",
			"asset": "asset--uuid",
			"position": "45.490946+9.228516",
			"probe": "FUJITSU CCTV"
			"first_observed": "1",
			"last_observed": "2",
			"number_observed" : "123",
			}

PROBE	Probe	A new object to describe probe	{
			"probe": "log-probe-123456",
			"asset": "assetname",
			}

PROBE	Probe Configuration	A new object to contain specific configuration of probe	{
			TBD
			}

PROBE	Indicator	Contains a pattern that can be used to detect suspicious or malicious cyber or physical activity.	{
			TBD
			}

PROBE	Intrusion Set	A grouped set of adversarial behaviors and resources with common properties believed to be orchestrated by a single threat actor.	{ "type": "intrusion-set", "id": "intrusion-set--uuid", "created_by_ref": "identity--uuid", "created": "2016-04-06T20:03:48.000Z",

		Attack Resource Level is an open vocabulary that captures the general level of resources that a threat actor, intrusion set, or campaign might have access to. It ranges from individual, club, contest, team, organization, to government. The Attack Motivation open vocabulary describes the Threat Actor or Intrusion Set's motivation. It ranges from “accidental”, “coercion”, “dominance”, “ideology”, “notoriety”, “organizational-gain”, “personal-gain”, “personal-satisfaction”, “revenge”, to “unpredictable”.	"modified": "2016-04-06T20:03:48.000Z", "name": "Intrusion set name", "description": "Intrusion set description", “first_seen”: "2016-04-06T20:03:48.000Z", “last_seen”: "2016-04-06T20:03:48.000Z", “resource_level”: “attack-resource-level-ov entry”, "aliases": [...], "goals": [...], “primary_motivation”: “attack-motivation-ov entry”, “secondary_motivations”: [attack-motivation-ov entries] }

INFRASTRUCTURE KBMODEL	Identity	Individuals or groups, as well as classes of individuals.
		
		The Identity Class open vocabulary describes the type of entity that the Identity represents: individual, group, organization, class (e.g. the Domain Administrators in a system), unknown.
			{
			"type": "identity",
			“id”: "identity--uuid",
			"created_by_ref": "identity--uuid",
			"created": "2016-04-06T20:03:00.000Z",
			"modified": "2016-04-06T20:03:00.000Z",
			"name": "Identity name",
			“description”: “Identity description”,
			"identity_class": "identity-class-ov entry”,
			“contact_information”: “Contacts”
			}

INFRASTRUCTURE KBMODEL	Organization	Organizations, or groups, organizations, or groups.	{
			"type": "organization",
			“id”: "organization--uuid",
			"created_by_ref": "identity--uuid",
			"created": "2016-04-06T20:03:00.000Z",
			"modified": "2016-04-06T20:03:00.000Z",
			"name": "Organization name",
			“description”: “Organization description”,
			“contact_information”: “Contacts”
			}

INFRASTRUCTURE KBMODEL	Asset	Asset of the Logical or Physical Infrastructure, like a PC, Application, ATM, etc. Keys follow ENISA Taxonomy for Logical (see figure 3 in D2.3) Keys follow FINSEC Taxonomy for Physical	{ "type": "asset", "id": "asset--uuid" "subtype": "Physical or Logical or both", “organization”: “organization--uuid”, ... standard ... "credential”: “pass, key, bio info, … auth”,

			"hardware": "["external": "..."], "["internal": "..."], "firmware": "["external": "..."], "["internal": "..."], "Logical Op": "Update functions… ", "Physical Op": "Update functions…, ", "User Info": "Behavior data, ", "User Health": "Safety against, ", "User Property": "Physical prop, Virtual Prop", ... }

INFRASTRUCTURE KBMODEL	Tool	STIX Legacy - legitimate software that can be used by threat actors to perform attacks.	{
			"type": "tool",
			"id": "tool--uuid",
			"created_by_ref": "identity--uuid",
			"created": "2016-04-06T20:03:48.000Z",
			"modified": "2016-04-06T20:03:48.000Z",
			"labels": [...],
			"name": "Tool name",
			"description": "Tool description",
			"tool_version": "Tool version",
			"kill_chain_phases": [...]
			}

THREAT KB	Attack Pattern	A type of Tactics, Techniques, and Procedures (TTP) that describes ways threat actors attempt to compromise targets.
		
		This is an extension of STIX to consider physical attacks
			{
			"type": "attack-pattern",
			"id": "attack-pattern--uuid",
			"created": "2018-04-06T20:03:00.000Z",
			"modified": "2019-02-06T18:13:36.140Z",
			"external_references": [...],
			"name": "Attack pattern name",
			"description": "Attack pattern description",
			"kill-chain-phases": [...]
			}

THREAT KB	Campaign	A grouping of adversarial behaviors that describes a set of malicious activities or attacks that occur over a period of time against a specific set of targets. This is an extension of STIX to consider physical attacks	{ "type": "campaign", "id": "campaign--uuid", "created": "2018-04-06T20:03:00.000Z", "modified": "2019-02-06T18:13:36.140Z", "name": "Campaign name", "description": "Campaign description", "aliases": [...], "first_seen": "2018-04-06T20:03:00.000Z", "last_seen": "2019-02-06T18:13:36.140Z",

			“objective”: “Campaign primary goal” }

THREAT KB	Malware	A type of TTP, also known as malicious code and malicious software, used to compromise the confidentiality, integrity, or availability of a victim’s data or system.	{
			"type": "???",
			"id": "",
			"created": "2016-05-12T08:17:27.000Z",
			"modified": "2016-05-12T08:17:27.000Z",
			"name": "",
			"description": "Malware description",
			"labels": [...],
			“kill-chain-phases”: [...]
			}

THREAT KB	***Tampering***	An attack to infrastructure aimed to modified the physical elements-
		
		This is an extension of STIX to consider specific physical attacks .
			{
			"type": "tampering",
			"id": "tampering--uuid",
			"created": "2016-05-12T08:17:27.000Z",
			"modified": "2016-05-12T08:17:27.000Z",
			"name": "ATM tampering",
			"description": "Tampering description",
			missing
			}

THREAT KB	Threat Actor	Individuals, groups, or organizations believed to be operating with malicious intent.
		
		The Threat Actor Role open vocabulary describes the different roles that a threat actor can play: agent, director, independent, infrastructure-architect, infrastructure-operator, malware-author, sponsor.
		
		The Threat Actor Sophistication open vocabulary captures the skill level of a threat actor: none, minimal, intermediate, advanced, expert, innovator, strategic.
		
		Attack Resource Level is an open vocabulary that captures the general level of resources that a threat actor, intrusion set, or campaign might have access to.
			{
			"type": "threat-actor",
			"id": "threat-actor--uuid",
			"created_by_ref": "identity--uuid",
			"created": "2016-04-06T20:03:48.000Z",
			"modified": "2016-04-06T20:03:48.000Z",
			"labels": [...],
			"name": "Threat Actor name",
			"description": "Threat Actor description",
			"aliases": [...],
			"roles": [threat-actor-role-ov entries],
			"goals": [...],
			"sophistication": "threat-actor-sophistication-ov entry",
			"resource_level": "attack-resource-level-ov entry",
			"primary_motivation": "attack-motivation-ov entry",
			“secondary_motivations”: [attack-motivation-ov entries],
			“personal_motivations”: [attack-motivation-ov entries]
			}

		It ranges from individual, club, contest, team, organization, to government. The Attack Motivation open vocabulary describes the Threat Actor or Intrusion Set's motivation. It ranges from “accidental”, “coercion”, “dominance”, “ideology”, “notoriety”, “organizational-gain”, “personal-gain”, “personal-satisfaction”, “revenge”, to “unpredictable”.	

ACTION	Course of Action	An action taken to either prevent an attack or respond to an attack.
		See Countermeasures in Fig 6. of D2.3
			{
			"type": "course-of-action",
			"id": "course-of-action--uuid",
			"created_by_ref": "identity--uuid",
			"created": "2016-04-06T20:03:48.000Z",
			"modified": "2016-04-06T20:03:48.000Z",
			"name": "Course of action name",
			"description": "Course of action description"
			}

PRESENTATION	CP Threat Intelligence	Cyber Physical Threat Intelligence
		
		Keys follow ENISA Taxonomy (*see figure 5 in D2.3*)
			{
			"type": "cpti",
			"id": "cpti--uuid",
			"asset": "asset--uuid",
			“attack-pattern”: “attack-pattern--uuid”,
			..
			“nefarious activity”: [
			“firmware_mod”: [...],
			“remote_firmware_att”: [...],
			“attack_persistence”: “Firmware modification/Bootkit”,
			“info_access”],
			“eavesdropping”: [...],
			“physical_att”: [...],
			“damage”: [...],
			“failures”: [...],
			“outages: [...],
			“legal”: [...]
			}

COLLABORATION	Report	Collections of threat intelligence focused on one or more topics, such as a description of a threat actor, malware, physical attack technique, including contextual details.	{
			TBD
			}

The Cyber Physical Threat Intelligence (CPTI) is the principal object since it collects and is enriched by threat information as soon as they are gathered from the probes and processed by the Predictive Analytics module. One or more CPTI objects are used to generate the output of the intelligence process, that is a report about ongoing or possible future attacks on one or more assets belonging to the infrastructure. The report can be accessed through the FINSEC dashboard and shared through the FINSEC collaborative module.

FINSTIX Relationships

FINSTIX basic objects will be put in relations with KB Objects while being transformed in the Engine.

The different FINSTIX objects can be related through either direct key-value relationships. Starting from the infrastructure, each asset is owned by a unique organization, thus it contains a key-value relationship that links itself to an organization object. In addition, each probe is related to a single asset, for this reason the probe object contains a key-value relationship that points at the related asset. Moreover, an observed data is related to a specific asset, then it presents a key-value reference to specify the particular asset. Finally, the Cyber Physical Threat Intelligence (CPTI) contains a key-value reference to the threat target asset.

In general, the Threat Knowledge Base objects are related through the Relationship Objects defined in STIX (see *Table 8*).

Table 8: STIX Relationship Objects

Source	Relationship Type	Target
attack-pattern	targets	identity, vulnerability

	uses	malware, tool
campaign	attributed-to	intrusion-set, threat-actor
	targets	identity, vulnerability
	uses	attack-pattern, malware, tool
course-of-action	mitigates	attack-pattern, malware, tool, vulnerability
indicator	indicates	attack-pattern, campaign, intrusion-set, malware, threat-actor, tool
intrusion-set	attributed-to	threat-actor
	targets	identity, vulnerability
	uses	attack-pattern, malware, tool
malware	targets	identity, vulnerability
	uses	tool
	variant-of	malware
threat-actor	attributed-to	identity
	impersonates	identity
	targets	identity, vulnerability
	uses	attack-patter, malware, tool
tool	targets	identity, vulnerability

References

GFT, Deliverable 2.4, H2020 BigDataStack, GA No 779747, 2018

Sandy Ryza, Uri Laserson, Josh Wills, Sean Owen, Advanced Analytics with Spark, O'Reilly Media, April 2015

Krishna Sankar, Holden Karau, Fast Data Processing with Spark - Second Edition, Packt, March 2015

Venkat Ankam, Big Data Analytics with Spark and Hadoop, Packt, September 2016

Nick Pentreath, Machine Learning with Spark, Packt, February 2015

Cloudera, Apache Kafka Guide, February 2019

Michael Mühlbeyer, Introduction to Apache Kafka, Trivadis, July 2018

Neha Narkhede, Gwen Shapira and Todd Palino, Kafka: the Definitive Guide, O'Reilly

Manish Kumar, Chanchal Singh, Building Data Streaming Applications with Apache Kafka

Radu Gheorghe, Matthew Lee Hinman, Roy Russo, Elasticsearch in action, Manning

Clinton Gormley, Zachary Tong, Elasticsearch The Definitive Guide, O'Reilly

Elasticsearch MongoDB integration:

https://researchcenter.paloaltonetworks.com/2017/11/engineers-at-work-enhancing-aperture-elasticsearch-mongodb/

https://www.nuxeo.com/fr/resources/supercharging-your-content-management-stack-with-mongodb-and-elasticsearch/

https://kafka.apache.org/documentation/

The research leading to these results has received funding from the European Union's Horizon 2020 Research and Innovation Programme, under grant agreement no 700071.