This project has received funding from the European Union's Horizon 2020 research and innovation programme under **grant agreement No 780355**
Ref. Ares(2020)4252801 - 14/08/2020

FANDANGO DELIVERABLE

Deliverable No.:	D5.3
Deliverable Title:	FANDANGO platform setup defining process
Project Acronym:	FANDANGO
Project Full Title:	FAke News discovery and propagation from big Data and Artificial iNtelliGence Operations
Grant Agreement No.:	780355
Work Package No.:	5
Work Package Name:	The FANDANGO software stack
Responsible Author(s):	ENG, Sindice, LIVETECH, VRT, CERTH, CIVIO
Date:	31.07.2020
Status:	v1.0 - Final
Deliverable type:	REPORT
Distribution:	PUBLIC

REVISION HISTORY

VERSION	DATE	MODIFIED BY	COMMENTS
V0.1	06/06/2020	Ghedin Alberto (ENG)	First draft
V0.2	25/06/2020	Panagiotis Stalidis, Thodoris Semertzidis (CERTH), David Martín (UPM), Camila Garcia (LvT), Neil Byrne (Siren)	First Internal revision
V0.3	17/07/2020	David Martín (UPM)	Internal review
V1.0	31/07/2020	Ghedin Alberto (ENG)	Final Version

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ABBREVIATION	DESCRIPTION
H2020	Horizon 2020
EC	European Commission
WP	Work Package
EU	European Union
K8s	Kubernetes
VM	Virtual Machine
IP	Internet Protocol address
Elastic	Elasticsearch
URL	Uniform Resource Locator
REST	Representational State Transfer
NER	Named Entity Recognition

EXECUTIVE SUMMARY

This document is a deliverable of the FANDANGO project funded by the European Union's Horizon 2020 (H2020) research and innovation program under grant agreement No 780355.

Up to February 2020 (M26), the project proceeded according to schedule and foreseen scientific and technical objectives. Starting from the very beginning of March 2020 (M27), however, COVID-19 situation has rapidly impacted project activities, causing delays. In particular, the impact of COVID-19 situation has delayed the preparation and submission of this deliverable, expected in M30.

In accordance with the consortium decision to adopt an iterative prototyping methodology in carrying out research and development activities, this deliverable contains also the updated revisions of FANDANGO Architecture (cf. D5.1), Data Model and Components (cf. D3.1).

Several iterative prototyping cycles leaded to the setup of a first full-featured implementation of FANDANGO that has been used to run the first pilot.

The implementation and testing of the different prototypes led to an architectural revision focused on Kubernetes (k8s), Kafka and Elasticsearch. In particular, the various components have been translated from a Cluster architecture based on Hadoop / Spark to a cluster architecture based on Kubernetes. The logical design and data flow have been slightly modified to gain better performances, as will be seen in the next chapters, but the infrastructure has been profoundly changed.

The parallel execution paradigm has remained unchanged and with K8s the service can scale to several instance to handle load peaks.

Requirements reported in deliverable "D2.3 User requirements" has been transformed in D5.1 in technical functionalities and then matched with the best technologies to satisfy them. During the development and the integrations between components we have tested different technologies and we decided to focus the infrastructure around Kafka, Kubernetes and Elasticsearch, so the mapping defined in D5.1 has been revisited. This decision, explained in next chapter, facilitates the portability, and optimizes the infrastructure. The following table [Table 1: User requirements mapping] reports the requirements associated with the chosen tool fulfil them.

REQUIREMENT	TOOL
Ingest data from web sites, claims, open data	Custom Crawlers
Analyze news article to find information that can help user to detect malicious contents	TensorFlow, ML libraries
Analyze videos or images for malicious modification detection	TensorFlow
Graph analysis for authors and publisher analytics	Neo4J
Visualization tool for statistics about current state of data in the system and graph visualization	Siren Investigate
Web application that permit user to retrieve useful information about a specific article	Python, Flask
Claim verification tool	Siren Investigate

Table 1: User requirements mapping

The decision to exclude Hadoop impacted also on non-functional requirement, in particular on the storage.

REQUIREMENT	TOOL
Intermediate storage that allows stream processing	Apache Kafka
Final storage for fast data retrieving and comparisons	Elasticsearch
Raw data storage	Azure Blob Storage, Elasticsearch

Table 2: Non-functional requirements mapping

1. INTRODUCTION

This document is closely connected with D5.1, were the FANDANGO reference architecture was defined. D5.1 was focused on the description of the architecture from the point of view of the software tools used. In the current document we maintained the same structure of D5.1 but with a deeper level of detail, describing the individual components implemented and the relationships between them. In addition, the FANDANGO project is following an iterative prototyping development approach, consequently some software tools were excluded in favour of other products that better adapt to the characteristics of the workloads implemented.

After the first pilot execution using a full featured implementation of the FANDANGO platform, it has been set up the migration of docker images and standalone software installed in VMs, to a Kubernetes cluster which orchestrates both the services developed in the project and the various control tools.

The part relating to storages, in particular Siren and Kafka, remained in the IaaS version both because, in this case, they do not benefit greatly from the features offered by K8s, and because, due to their nature of data storage, it is more suitable to remain standalone installations.

The document, after a brief introduction to Kubernetes, will list the Kubernetes deployments and describe the main features such as input/output and their implementation.

2. DATA FLOW

The aim of this chapter is to give an overview of the system, focusing on data transformations, without going into the implementation details.

The Fandango project aims to address the aggressive emergence of fake news, post-truths, and misinformation, providing a web app and web services (SaaS) to support professionals with the following high-level features:

- misleading detection and scoring, providing a set of misleading scores (based on Big Data analysis techniques) of the different relevant components of News: Text, Authors, Source and Media
- Support user data investigation, through an interactive exploration of open data and verified claim databases.

To implement these features, we get input data from a selected list of web sites, verified claims sources and open data repositories. Web sites represent the list of monitored sources used to retrieve news to be scored by FANDANGO's analyzers. Open data repositories are databases of trusted information that FANDANGO users can use to check articles content and retrieve additional information for specific topics, like immigration or climate change. Finally, Claims are fact-checked claims, debunked by fact-checkers. Data are gathered by web crawlers, software programs that browse specific internet web sites to retrieve their content. This is the ingestion module of the project.

Data collected from websites need to be processed and analyzed in a pipeline, while open data and claims are stored directly into the final database since they do not need to be processed. A graphical representation of the current, revised data flow is available in [Figure 2: Data flow diagram].

The first choice to perform ingestion operation was Apache Nifi (https://nifi.apache.org/). It offers a set of functions to design reach data flows that retrieves data from the web and allows custom programs to execute in his components. NiFi has been integrated from the first version of the platform and it has been a crucial solution to perform these operations. When we decided to move on K8s to optimize the infrastructure and containers management, we have divided the various components implemented with NiFi into specific containers in order to isolate the logical functions in atomic micro-services and to simplify the software architecture by delegating the orchestration of all the components to K8s.

Raw data pass through Apache Kafka (https://kafka.apache.org/), a queue-based storage that allows data to be retrieved from different applications. Data is are accessible by each analytic component allowing to share data and information between the elaborations.

Once data is stored in the system a process will elaborate the information, categorizing the content. Raw data is divided extracting text, images videos and metadata, which will be managed accordingly. Metadata includes additional information like authors, source, publish time and so on. Kafka's storage is organized in topic, distinct queues for different type of data. Each type of information (text, image, video, metadata) will be saved in related topic.

The data processing is then delegated to the various analysis processors, delivered in containers with specific program languages and libraries. Consider that the infrastructure allows the scaling of the containers according to the stress of the running instances.

The output of the analysis processors is combined using KSQL and is saved into Elasticsearch (https://www.elastic.co/products/elasticsearch).

In the Data Flow described in D5.1 and in D3.1 the offline ingestion process works as follows [see Figure 1: D5.1 Proposed Data Flow]:

1. the components of the news were extracted (by pre-processing)
2. the news components were analyzed by the analysis engines
3. when the engines completed the analysis, they published the result into the specific Kafka topic
4. the aggregator waited, in a 24-hour window, for all the engines analysis to compose complete result
5. the aggregator wrote the result in elastic

This process performance was heavily affected by faults and/or different throughputs of analysis engines, causing loss of data if at least one of the engine takes more than 24 hours to provide its result, causing the eviction of the news in the aggregation phase.

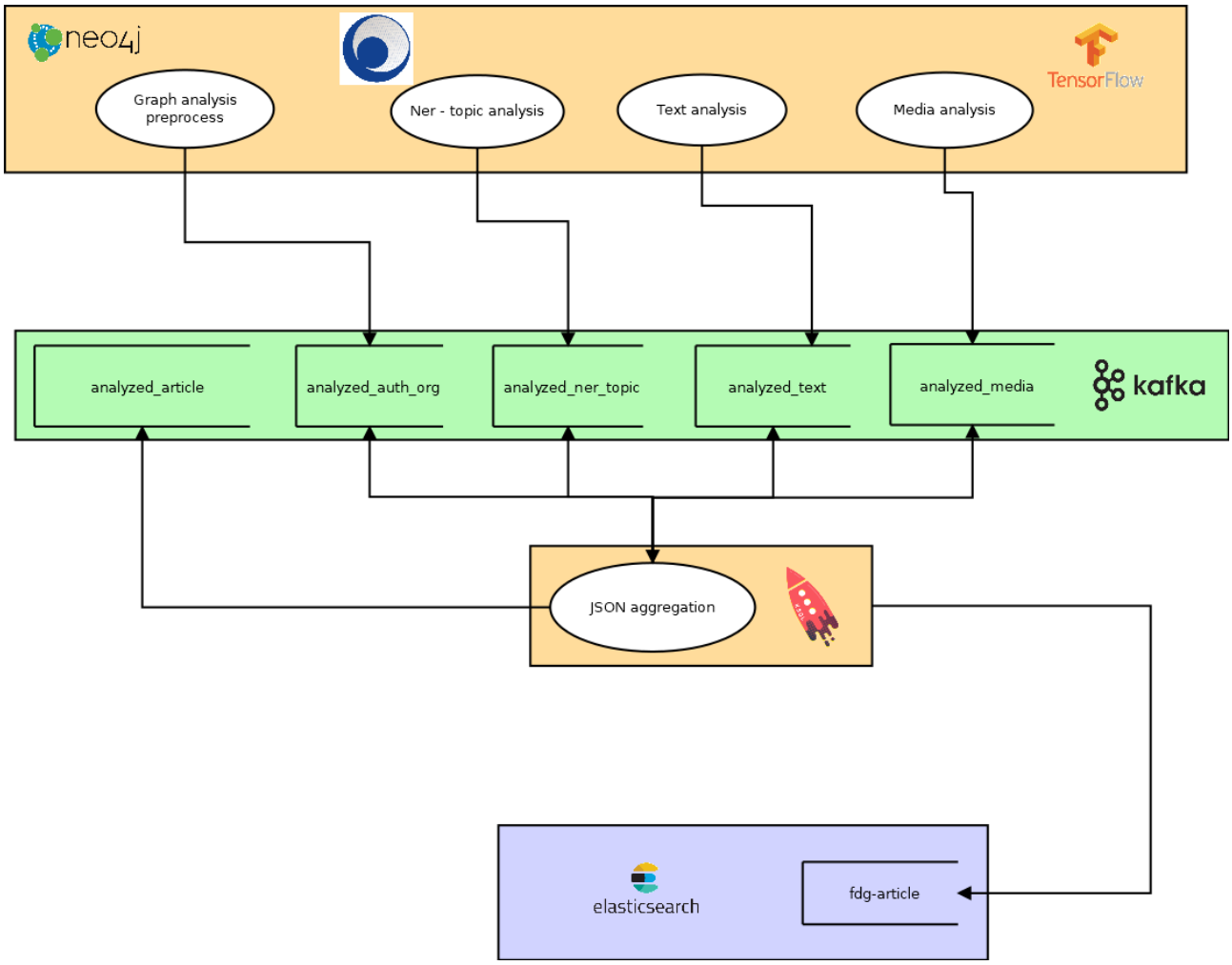

Figure 1: D5.1 Proposed Data Flow

Hence, the ingestion process has been revised to remove this potential cause of data loss while preserving the analyzers to run in parallel and asynchronously. In detail, with the current, revised data flow, the ingestion process performs the following steps (see Figure 2: Data flow diagram):

1. crawled data are preprocessed
2. each specific engine analyse its specific element of news (i.e. text, authors, source, media; for example, the multimedia analysis engine analyses the set of images)

3. each specific engine writes on its respective specific index in Elastic one or more documents related to the current analysis (for example if the news contains 5 images, the multimedia analysis engine creates 5 documents in the fdg-media index) and get the IDs of the docs returned from Elastic
4. the specific engine then creates a document with the returned IDs of the documents that will contain the results of the analysis, once completed (in our example, with the IDs of the 5 documents related to image analysis), writes the document in the Kafka topic and begins the analysis
5. the aggregator collects the documents with the IDs produced by the engines
6. once all the documents provided by the engines are collected, the aggregator builds and writes in Elastic a document that joins all collected IDs (it is worth noting that in this revised version, the aggregator collect the IDs of the analysis results instead of the results)
7. when an engine completes its specific analysis, it updates its relative index (for example the multimedia analysis engine updates the fdg-media index) and notifies to the fusion score service that the analysis has been completed
8. when all analyses have been completed, the fusion score service calculate and updates the doc with the fusion score

From the data model point of view, the big change is that the final doc (in the index fdg-article) now contains the IDs of the analysis (and therefore of the result), not the value of the analysis result.

The project main data storage, Elasticsearch, allows fast access to data via an advanced indexing as well as a search engine that allows full text queries, fundamental for claim verification and rapid user interaction with the database. From Elasticsearch are made available for final user to be interrogated and satisfy user requirement specified.

Siren Investigate (https://siren.io/) will allows the final user to have access at data statistics and information graph. With this tool final user will be provided with the necessary information that is employed to check the propagation and the correlation of fake news or a topic, and to alert regarding suspicious news.

On the other hand, FANDANGO web application is the other visualization tool provided by the project. In this scenario, a certain user will use the web app to investigate reliability of specific articles with the possibility of selecting part of the text and verify a particular claim. These features will support the final decision of the user when analyzing specific news, providing all information extracted from the analysis with additional advice retrieved from analytical computation of historical data.

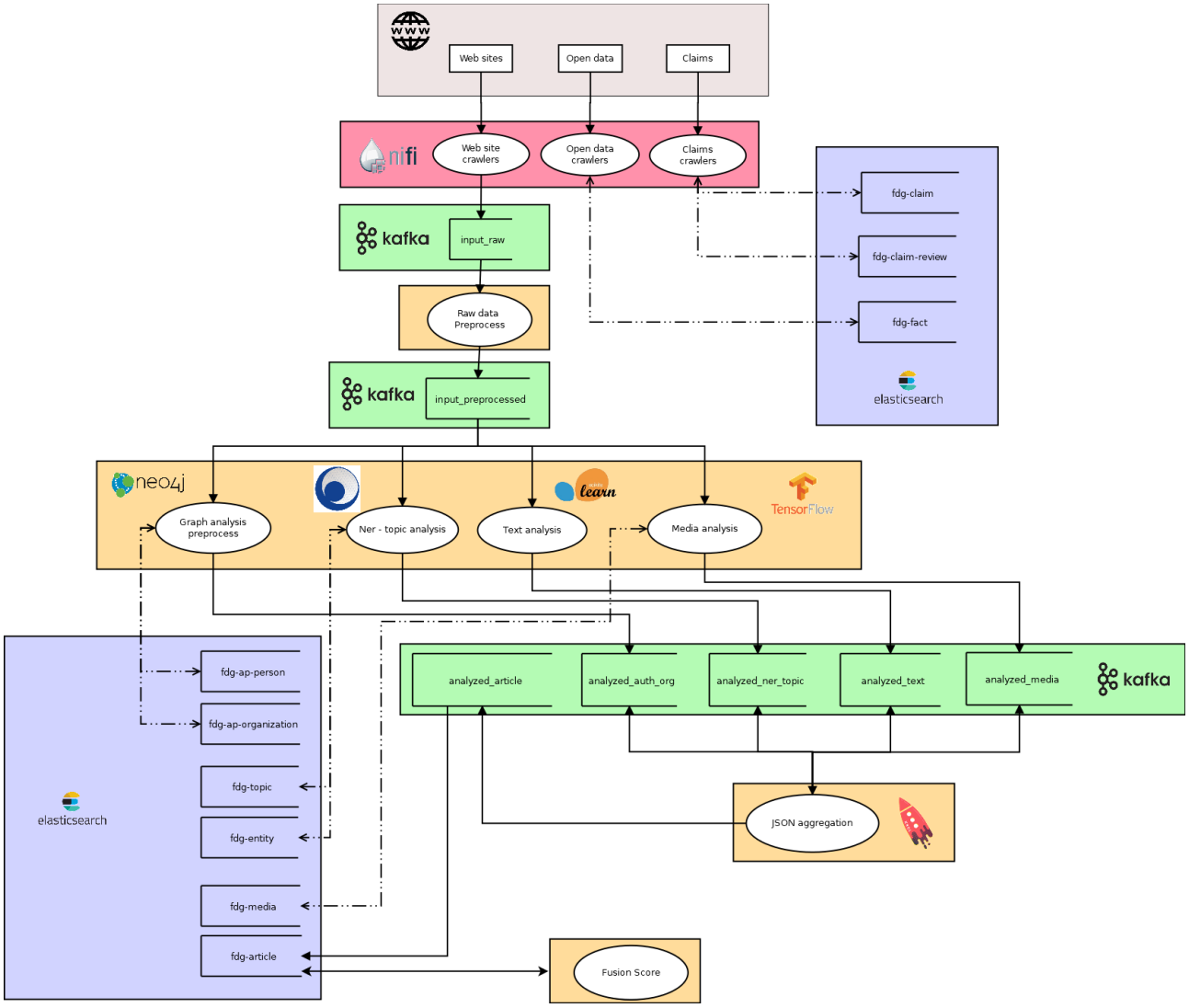

Figure 2: Data flow diagram

3. ARCHITECTURE OVERVIEW

As described in the introduction the infrastructure has been set up on a Kubernetes cluster.

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem.

Kubernetes provides a framework to run distributed systems resiliently. It takes care of scaling and failover for application, provides deployment patterns, and more.

Kubernetes provides:

- **Service discovery and load balancing**
  Kubernetes can expose a container either using the DNS name or using their own IP address (in this case using functionalities provided by Azure Cloud). If the amount of traffic to a container is considerable large, Kubernetes can load balance and distribute the network traffic so that the deployment is stable.
- **Storage orchestration**
  Kubernetes allows to automatically mount several types of storage system, such as local storages, public cloud providers, and more.
- **Automated rollouts and rollbacks**
  It's possible to describe the desired state for the deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate (this is crucial and useful when new version of software is deployed)
- **Automatic bin packing**
  The Kubernetes cluster is composed by several nodes that are used to run containerized tasks. Each task can be configured to use a specific amount of resources such as CPU and memory (RAM). Kubernetes can fit containers in the nodes to make the best use of the resources.
- **Self-healing**
  Kubernetes restarts containers that fail, replaces containers, kills containers that do not respond and does not advertise them to clients until they are ready to serve.

Reference: (Kubernetees Offical Documentation)

3.1. KUBERNETES IN FANDANGO

In Fandango K8s is used as orchestrator for services (pre-processing, analysis, monitoring, etc...).

Here a representation of new infrastructure with K8s:

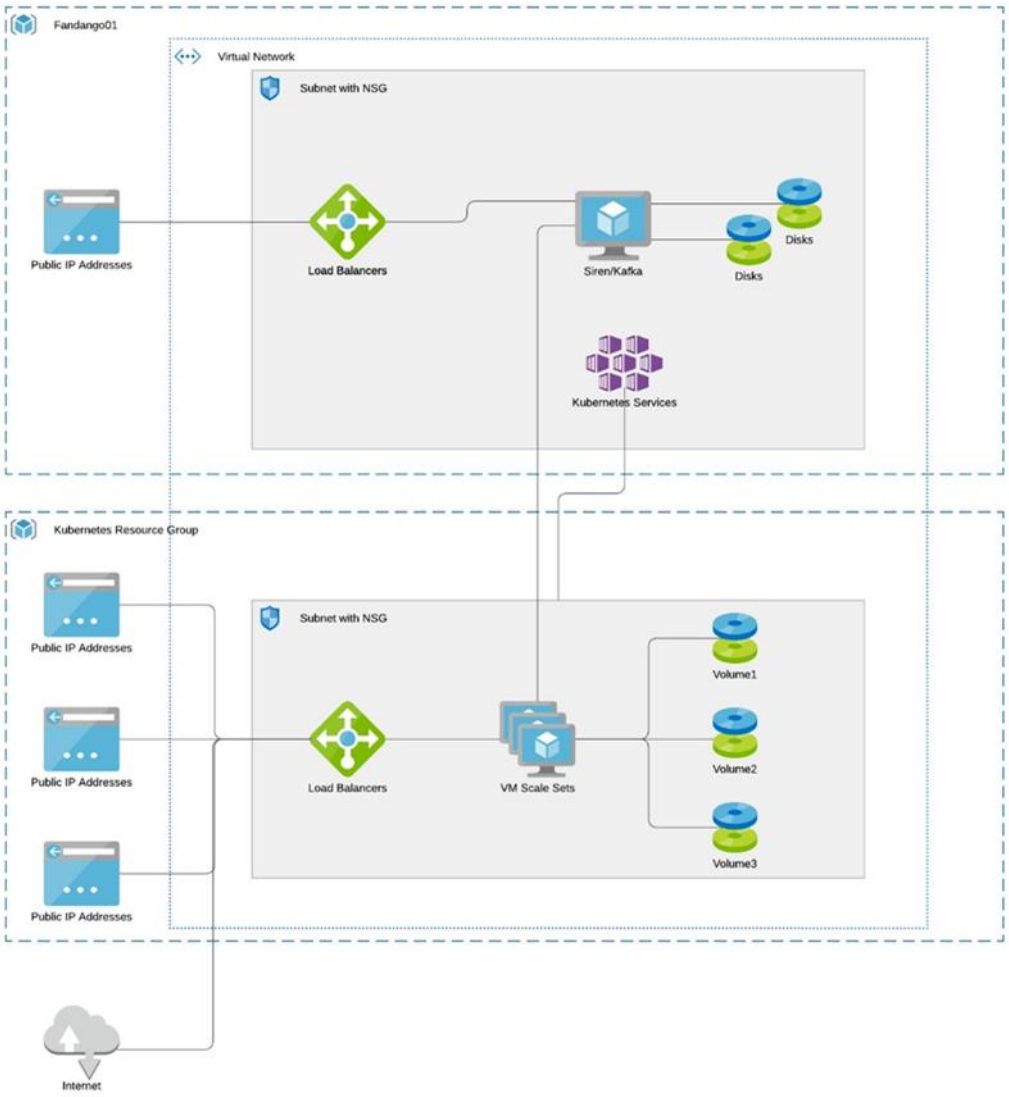

Figure 3: Fandango Infrastructure

The figure above shows two connected clusters. The main cluster consists of an infrastructure built to host virtual machines and storaging services (IaaS mode). The cluster contains virtual network, balancer, security group, ip and all the components necessary to keep the services isolated from internet, to ensure the security of internal data and the interoperability of internal components. The second cluster, which actually belongs to a subnet of the main cluster, contains the Kubernetes infrastructure (PaaS mode).

Even if the K8s cluster provides public IP address for the hosted services, the access is blocked by the firewall. All the comunications between the internet to the K8s cluster pass through a reverse proxy (an nginx).

3.2. KUBERNETES CONCEPTS

Kubernetes contains several abstractions that represent the state of the system: deployed containerized applications and workloads, their associated network and disk resources and other information about what your cluster is doing. These abstractions are represented by objects in the Kubernetes API.

3.2.1. POD

A *Pod* is the basic execution unit of a Kubernetes application, the smallest and simplest unit in the Kubernetes object model that you create or deploy. A Pod represents processes running on the cluster (for example an analyzer or KSQL).

A Pod encapsulates an application's container (or, in some cases, multiple containers), storage resources, a unique network identity (IP address), as well as options that govern how the container(s) should run. A Pod represents a unit of deployment: *a single instance of an application in Kubernetes*, which might consist of either a single container or a small number of containers that are tightly coupled and that share resources.

3.2.2. SERVICE

Each Pod gets its own IP address, however for a service (for example graph-analysis), the set of Pods running in one moment in time could be different from the set of Pods running that application a moment later (pods can be evicted and recreated, so the linked IP can change).

This leads to a problem: if some set of Pods (for example analysis pods) provides functionality to other Pods (for example the wen interface pods) inside the cluster, how do the frontends can communicate to the backend if the entry point (the IP) can change?

In Kubernetes, a Service is an abstraction which defines a logical set of Pods and a policy by which to access them (this pattern is called a micro-service). The set of Pods targeted by a Service is usually determined by a selector.

For example, consider the image-processing backend which is running with three replicas. While the actual Pods that compose the backend set may change, the web interface clients should not need to be aware of that, nor should they need to keep track of the set of backends themselves.

The Service abstraction enables this decoupling.

3.2.3. VOLUME

Docker also has a concept of volumes, though it is somewhat looser and less managed. In Docker, a volume is simply a directory on disk or in another Container. Lifetimes are not managed and until very recently there were only local-disk-backed volumes.

A Kubernetes volume, on the other hand, has an explicit lifetime - the same as the Pod that encloses it. Consequently, a volume outlives any Containers that run within the Pod, and data is preserved across Container restarts. Of course, when a Pod ceases to exist, the volume will cease to exist, too. Perhaps more importantly than this, Kubernetes supports many types of volumes, and a Pod can use any number of them simultaneously.

At its core, a volume is just a directory, possibly with some data in it, which is accessible to the Containers in a Pod. How that directory comes to be, the medium that backs it, and their contents are determined by the particular volume type used.

To employ a volume, a Pod specifies what volumes to provide for the Pod and where to mount those into Containers.

A process in a container sees a filesystem view composed from their Docker image and volumes.

3.2.4. REPLICASET

A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.

3.2.5. DEPLOYMENT

A *Deployment* provides declarative updates for Pods and ReplicaSets. It contains the configurations of the containers, pods, replica sets and volumes.

3.3. FANDANGO KUBERNETES INFRASTRUCTURE

The logical infrastructure described in [Figure 2: Data flow diagram] has been implemented in Kubernetes using the objects described in the previous paragraphs. For all the analysis services and the data interaction components (KafkaHQ, KSQL) it has been defined a set of k8s configuration files that describes:

- deployments: that configure the container (replica set, ports, image, variables, specific image configurations)
- service: that configure the service to access the functionality
- volume claim: the definition of the volumes

In the following images we represent the architecture implemented in K8s and hosted in our cluster for the online and offline process (to simplify we show only the deployment. Consider that each deployment logically contains the definitions of replica set, pod, service, volume). Most deployments serve both the online and offline process, in the following chapters we go in details of each of them, describing where they are used.

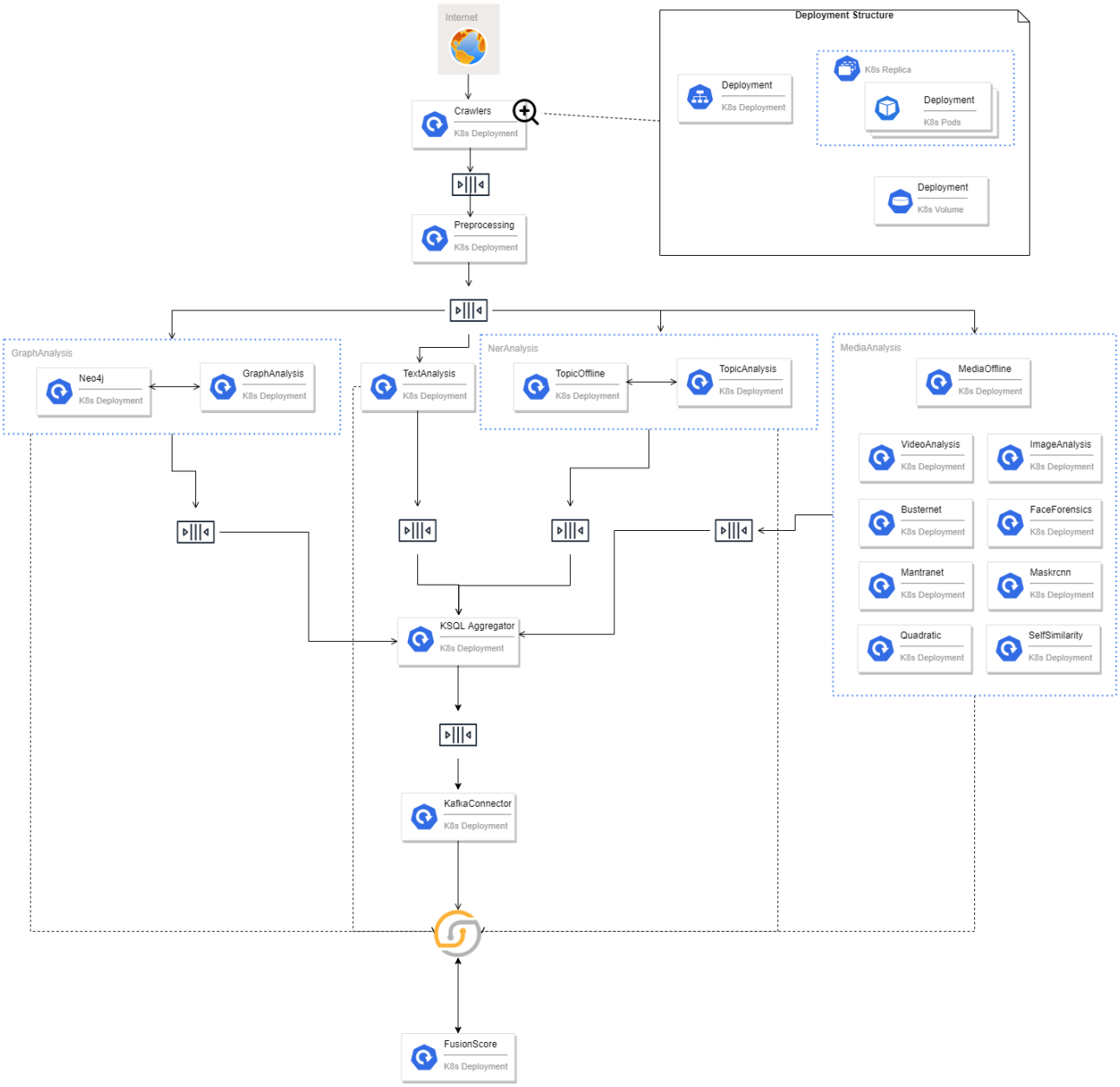

Figure 4: Offline process on K8s

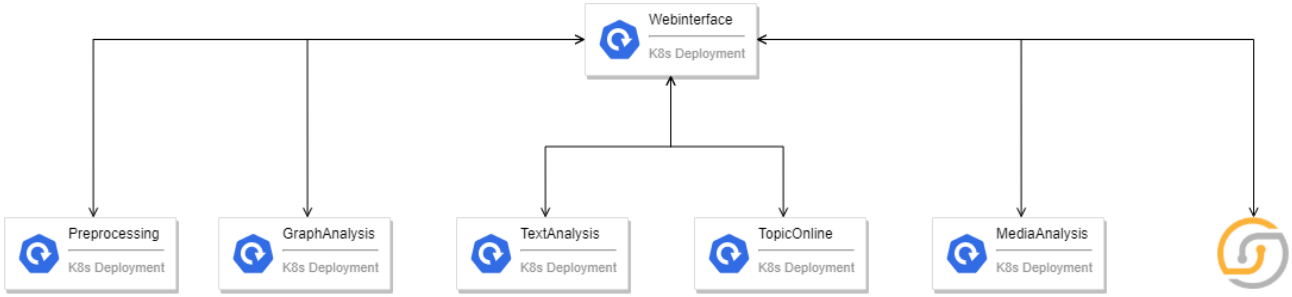

Figure 5: Online Process on K8s

The artifacts have been created translating the docker-compose configurations files. This procedure has been done by the infrastructure team in a semi-automatic way to reduce the risk to lose configurations, and to have uniform structures. The step performed are:

- review the docker-compose files of all the containers
- execute the kompose command to have a first mapping from docker-compose files and k8s *.yaml files
- fix the conversion problems.

4. INFRASTRUCTURE TESTS

The transition to the new K8s architecture took several weeks of testing, both to check the infrastructure and the integration between the different components.

As previously described, the Kubernetes distribution used in the project is AKS. This platform simplifies the delivery of the cluster and, after studying and defining the architecture, the infrastructure can be created with a single command. The problems encountered in the testing phase were mainly related to the type of components and their usage in a cost-effective way. In particular, the architecture implemented required some components that belong to the free tier, but anomalous behaviors were found and took some time to find a solution.

The translation of the configuration files from the docker compose to the k8s yaml required a long work of testing on the individual components and on the integration of the various services. This phase was done first in local, using minikube, and finally in the cloud infrastructure. This last step was done first by the team responsible for the infrastructure, then by the developers and finally integrated in group sessions.

The business logic is included within the individual containers, consequently it did not require particular changes, but it was necessary to review some details to adapt them to the orchestrator's logic. An example is error handling. In Kubernetes it is important that in case of temporary errors (such as a disconnection from Kafka or from a service), the container exits with an error message so that the orchestrator sets it as in error state, dismisses it and creates another instance. Environment variables have also been revised and other minor changes have been made for containers.

The last phase was the tuning. The components were integrated into K8s and stress tests (load and execution time) were carried out so that the architecture could bear the load under normal conditions. The result was cluster sizing and memory/cpu quotas configurations for individual pods.

Going into the detail of the single components, the testing process is generally composed of these phases:

- unit tests of the individual software components implemented during the developments
- integration test of the containers in local environment
- integration test of the containers in the docker integration environment (Azure)
- integration test of the different modules within the Kubernetes environment (Azure)
- user acceptance test of the individual functions within the UAT Kubernetes environment (Azure)

The open points have been registered in GIT in the Issues section.

5. INFRASTRUCTURE ENVIRONMENT

The infrastructure is hosted in Azure Cloud. The introduction on Azure Cloud Service is available in "D5.1 Reference Architecture description", so in this chapter we present the implementation details.

The actual implementation is well presented in the figure [Figure 3: Fandango Infrastructure], where you can identify two separated Resource Groups. Here the page in azure portal:

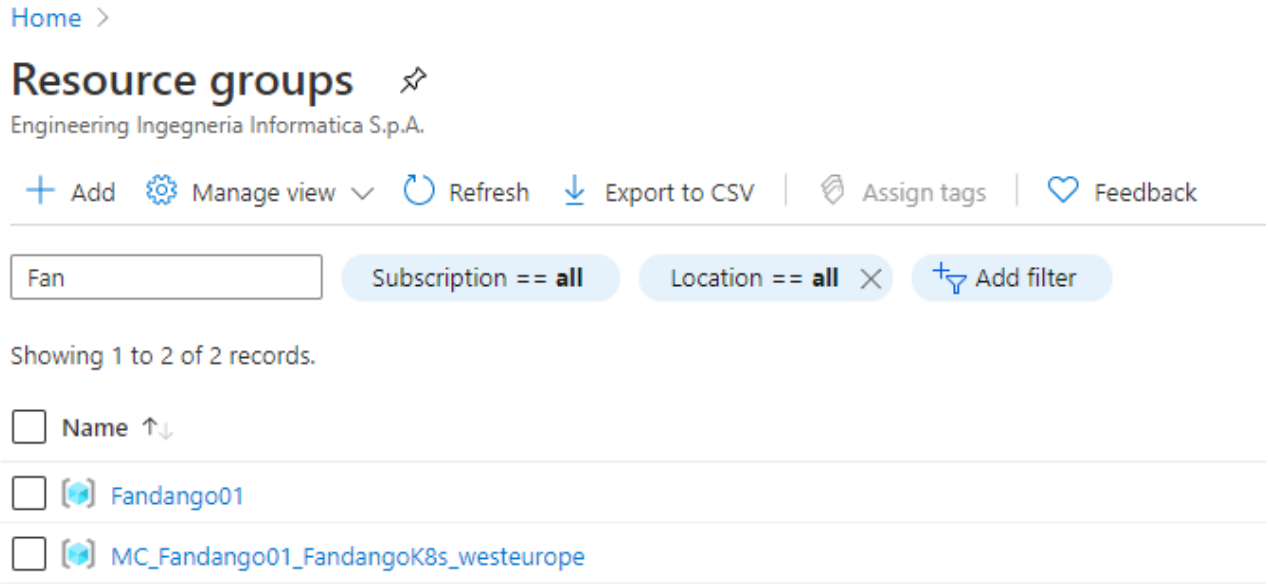

Figure 6: Azure Resource Groups

Fandango01 is the parent resource group and contains the following objects:

Field	Type
fandango01	Container registry
Fandango01	Recovery Services vault
Fandango01IP	Public IP address
Fandango01LB	Load balancer
Fandango01Network	Virtual network
Fandango01NSG	Network security group
FandangoAvailSet01	Availability set
FandangoEdge01	Virtual machine
fandangoedge01364	Network interface
FandangoEdge01_DataDisk	Disk
FandangoEdge01_OsDisk_1	Disk
FandangoEdge02	Virtual machine
fandangoedge02331	Network interface
FandangoEdge02_DataDisk	Disk
FandangoEdge02_OsDisk_1	Disk
FandangoGPU01	Virtual machine
fandangogpu01104	Network interface
FandangoGPU01_DataDisk_0	Disk

FandangoGPU01_OsDisk_1	Disk
FandangoK8s	Kubernetes service
fandangostorage	Storage account
StartAzureV2Vm (Startup/StartAzureV2Vm)	Runbook
Startup	Automation Account

Table 3: Fandango01 Resource Group's elements

From the table is possible to identify tree cluster of resources:

- General resources like:
  - Network: to connect all the objects.
  - Public Ip Address: the point to access to the network. The IP is linked to the load balancer.
  - Load Balancer: the load balancer in this case acts as NAT redirecting the requests to the correct endpoint inside the virtual network.
  - Network Security Group: this object acts as firewall and enable or deny the access to the network. For example, all the internal services are not exposed to the outside, but they can be accessed just inside the virtual network.
  - Availability set: An Availability Set distributes highly available workloads across multiple Fault Domains, thereby eliminating any single point of failure. Unless the entire data center is down, workload will keep running.
  - Container Registry: Contains the custom containers.
  - Storage Account: This is an object store used to store raw data.
  - Runbook and Automation Account: these resources are used to automate the startup of the VMs
- VM resources like:
  - Virtual machines: Azure VM are the engine for the executions. We currently have 3 VM:
    - FandangoEdge01: Machine for integration tests with docker. It also hosts a node of Kafka.
    - FandangoEnge02: Machine with Elastic and Kafka.
    - FandangoGPU: a very powerful machine to test the Media services. In case of necessity to speed up the computational capabilities of the cluster the machine can be started and used.
  - Disks: each machine has one OS disk for the Operative system, and a Data disk.
  - Network Interface: the link to the virtual network.
- Kubernetes Service: the PaaS service for Azure Kubernetes (AKS).

MC_Fandango01_FandangoK8s_westeurope is the resource group linked to the AKS service. It contains the following resources:

Field	Type
aks-agentpool-36482182-nsg	Network security group

aks-nodepool1-36482182-vmss	Virtual machine scale set
kubernetes	Load balancer
kubernetes-a019ec8ba247d41d7b08dddc87b74873	Public IP address
kubernetes-a4b7c574b2c9846b187503c686d7d828	Public IP address
kubernetes-a4f9e163a9e4e40a8b383b312d5dce59	Public IP address
kubernetes-a52f561648259424c94e2798d1e42ef6	Public IP address
kubernetes-a72cb3c5a192a4b8ab29d064b1732b01	Public IP address
kubernetes-a8ceff0ff7e2c456e9f41c23023f479d	Public IP address
kubernetes-aa029f1b9ee6f4e9dbe83677f0886be6	Public IP address
kubernetes-aa28d74ae34e147f8b4eadeeafcd1117	Public IP address
kubernetes-ab23d70601e644de2989431bb464bb7f	Public IP address
kubernetes-abd434a1c360849debc6f5a709015d7c	Public IP address
kubernetes-ac487fe6d278243a89a03bedf1b280af	Public IP address
kubernetes-acd3cb70a0bf3449bb07d77c5a054ece	Public IP address
kubernetes-adce2a3f6b8fd4336b116456b299ae45	Public IP address
kubernetes-ae5917dd197d04549ae9ad400f7171e9	Public IP address
kubernetes-ae75c7f16907141398153da28c153164	Public IP address
kubernetes-aeb8ea944cae845e5a96dd2bda66e9a5	Public IP address
kubernetes-aed7097817d3e411da827f6d4f45becc	Public IP address
kubernetes-aedf38483a3a64c0a8abd6ff81d32c59	Public IP address
kubernetes-dynamic-pvc-24164671-6baf-4a34-bc9b-d09922e2599a	Disk
kubernetes-dynamic-pvc-31d0906e-bf94-45de-9f20-3437438f801e	Disk
kubernetes-dynamic-pvc-541afd4d-a940-4349-b825-96215dafde56	Disk
kubernetes-dynamic-pvc-912e7b76-f4b1-4775-817c-876afd1f54ab	Disk
kubernetes-dynamic-pvc-94885532-8ad5-43b7-9a6d-779b2adb7d39	Disk
kubernetes-dynamic-pvc-9fadb817-40a6-4350-8b2f-660b37d76bd4	Disk
kubernetes-dynamic-pvc-d3285acd-f1a1-445b-bac3-93d795a312ed	Disk
kubernetes-dynamic-pvc-f385852e-e58d-4fd9-9f45-45787b0c2a36	Disk

Table 4: MC_Fandango01_FandangoK8s_westeurope Resource Group's elements

It contains:

- Network Security Group (NSG): this NSG belongs to the subnetwork of Kubernetes. This security group is secondary to the Fandango01NSG, so if you want to give internet access to an IP present in this subnet, you must define a rule for both NSGs.
- Load Balancer: in this case the load balancer acts as balancer of requests to the Virtual Machine scale set.
- VM scale set: it is a scalable set of machines. AKS has a dynamically configurable set of nodes on which it can deploy pods based on load. In this moment, the set consists of three machines with 16 GB of RAM and 8 cores each, but in case of excessive load it is configured to reach up to 5 VM. The set is configured to release the VMs overnight.
- Public Ips: public Ips connected to the K8s services
- Disks: volumes associated to the PODs

The architecture is scalable on two levels:

- The VM scale set automatically scale in and out depending on the load of the provided VMs
- The PODs can scale according the load of the existing PODs of same Replica Set

All these configurations are configurable and then the system react according the load (the metrics are calculated on the average load in a minute)

6. COMPONENT DETAILS

In this chapter the individual software components described in the figures [Figure 4: Offline process on K8s] and [Figure 5: Online Process on K8s] will be detailed.

6.1. CRAWLERS

The Crawler service is the component which provides data to the infrastructure from different sources. They are implemented as an always-on container that analyses the web and captures news from a set of configurable sources.

These data come from three main groups of sources: articles from news sites, reviews of statements by fact-checking agencies and facts from archives of open data.

The first group of data are extracted from web pages (in HTML format) as structured information. Transforming un-structed data into structured, presents several caveats. Relevant information can be spread in any part of the document, can be encoded in any format or it can be missing altogether. The main parts that are checked and verified before passing the information on are the title, author(s), publisher(s), date of publication and/or modification and media. Moreover, to have clean data for the later stages, an initial check for advertisements and irrelevant content is performed to exclude it from both the text body and the media files. Finally, if an exact date is missing from the article, the date and time of crawling is assumed since this is the earliest time of existence that can be verified from the system.

The second data source for the Fandango platform are the reviews provided by the fact-checking sites. Although the information provided by the fact checkers on their websites is unstructured, a set of sites have followed Google's initiative for verified news. These fact checkers also offer a review summary in a crawler-compatible json format that follows the schema.org claimReview scheme. Fandango crawlers collect only reviews that follow the specifications so that users can find reviews relevant to their query, quickly. To easily compare reviews, the provided scores are normalized before storage to the same range of values.

The third source of information for Fandango platform is open datasets. Open data are silos of raw information offered to the public. Each of the databases follows an internal structure for the data they provide, so a common analysis of data from different sources is practically impossible. Index creation is performed by several source-specific crawlers that aim to extract the type of information provided in each data source in each database and the spatial and temporal scope of the data. Therefore, the user can quickly detect factual data about the query it is running.

The Crawler implementation is based on the following three containers:

Functionality	News Crawler
Technology	K8s/Docker
Containers	certh_news_crawlers
Language	Python
Input-online	API(Table 42: crawling and preprocessing output Article Crawler)
Input-offline	Scheduler
Output-online	API(Article Crawler)
Output-offline	Kafka (input-raw)
Notes	

Table 5: News Crawler Implementation

Functionality	Claim Crawler
Technology	K8s/Docker
Containers	certh_claim_crawlers
Language	Python
Input-online	
Input-offline	Internet
Output-online	
Output-offline	*Elastic (Table 36: fdg-article index* fdg-claim)
Notes	

Table 6: Claim Crawler Implementation

Functionality	Fact Crawler
Technology	K8s/Docker
Containers	certh_fact_crawlers
Language	Python
Input-online	
Input-offline	Internet
Output-online	
Output-offline	Elastic (fdg-fact)
Notes	

Table 7: Fact Crawler Implementation

6.1.1. COMPONENT TEST

To verify the functionality of the crawlers, several Unitest were executed using different kind of inputs to evaluate the performance of the components. More specifically, these set of tests were performed independently for both Offline and Online processes since the way they communicate to the rest of the component differs according to each pipeline. After the components were integrated on the platform, we arranged integration tests performed by developers and infrastructure manager in order to check the behavior in the K8s cluster.

6.2. PRE-PROCESSING

The main goal of this service is to clean the data delivered by crawlers and to provide an appropriate format for the information collected before performing the analysis. To do so, different procedures are needed including machine / deep learning techniques to ensure the quality of different input fields such as authors, domains, language or dates. More specifically, the set of crawlers collect all the authors that are assumed to be associated to the article. However, in many scenarios, they are unreachable, or the crawlers are not capable to distinguish between the author's name and other unnecessary tags associated with them on the website. Hence, this service contains a deep learning model which can distinguish between human names in a sentence. This family of models is known as the Named Entity Recognition (NER) models. Additionally, to increase the precision of pre-processing, filters based on regular expressions were used.

Language detection is also required since in some cases, crawlers cannot acquire it directly from the website, so the pre-processing module must face it. To do this, the service uses a deep learning model for automatic language identification (ALI) based on the title and text of the article in cases where crawlers are unable to collect it.

Furthermore, using the URL and the source domain of the article, the pre-processing extracts the publisher associated to the article by either using WHOIS IP or text matching procedures (when WHOIS is not capable of analyzing the domain) with a large list of sources in order to retrieve the desired value.

Additional methods are used to discard those articles whose text or title fields are empty probably because they are not interesting articles but other types of sources that crawlers have collected (linked docs, ADs, login pages).

A similar procedure is performed for the media content retrieved by the crawlers; in some cases, non-related images such as logos or banners may appear and must be removed in order to avoid troubles when training the image analyzer. Therefore, using the aspect ratio and the number of pixels of the image, the pre-processing decides whether a certain image must be discarded or not.

Finally, the dates are also reviewed to avoid unexpected formats. The format used in this case is based on Universal Coordinated Time (UTC) to mitigate the problems associated with the time zone of each country.

Functionality	Preprocessing
Technology	K8s/Docker
Containers	fandango_preprocessing
Language	Python
Input-online	*API (Table 42: crawling and preprocessing output*

6.3. ARTICLE CRAWLER

6.3.1. END POINT

Method	Endpoint
POST	api/get_article

Table 43: crawlers end-point

6.3.2. INPUT

Field	Type	Description
URL	string	URL to an article

Table 44: crawlers input

6.3.3. OUTPUT

Field	Type	Description
identifier	string	The unique identifier given to the article
title	string	The extracted title of the article
description	string	Extended title of the article

	summary	string	A summary of the article contents	
	text	string	The full text body of the article	
	keywords	list of string	A list of keywords provided by the publisher	
	language	string	The detected language the article is written in	
	authors	list of string	A list of authors for the article provided by the publisher	
	date_created	datetime	The date and time in UTC that the request was created	
	date_modified	datetime	The date and time in UTC that the article was last modified (according to the publisher)	
	date_published	datetime	The date and time that the article was first published (according to the publisher)	
	publish_date_estimated	string	A "yes"/"no" of if the publish date was estimated by Fandango (or provided by the publisher)	
	top_image	string	The URL to the most prominent image in the article	
	images	list of string	A list of URLs to all the image media files in the article (may contain advertisements)	
	videos	list of string	A list of URLs to all the video files in the article	
	URL	string	The URL of the article	
	source_domain	string	The source domain of the downloaded article (publisher)	
	spider	string	The name of the spider that was used to download the article (internal debugging)	
	texthash	string	A hash of the full text (for duplicate detection)	

Table 45: crawlers output

Preprocessing)

Input-offline	Kafka (input-raw)
Output-online	*API (Table 42: crawling and preprocessing output*

6.4. ARTICLE CRAWLER

6.4.1. END POINT

Method	Endpoint
POST	api/get_article

Table 43: crawlers end-point

6.4.2. INPUT

Field	Type	Description
URL	string	URL to an article

Table 44: crawlers input

6.4.3. OUTPUT

Field	Type	Description
identifier	string	The unique identifier given to the article
title	string	The extracted title of the article
description	string	Extended title of the article
summary	string	A summary of the article contents
text	string	The full text body of the article
keywords	list of string	A list of keywords provided by the publisher
language	string	The detected language the article is written in
authors	list of string	A list of authors for the article provided by the publish
date_created	datetime	The date and time in UTC that the request was create
date_modified	datetime	The date and time in UTC that the article was last mod (according to the publisher)
date_published	datetime	The date and time that the article was first published (according to the publisher)
publish_date_estimated	string	A "yes"/"no" of if the publish date was estimated by Fandango (or provided by the publisher)
top_image	string	The URL to the most prominent image in the article
images	list of string	A list of URLs to all the image media files in the article contain advertisements)
videos	list of string	A list of URLs to all the video files in the article
URL	string	The URL of the article
source_domain	string	The source domain of the downloaded article (publish
spider	string	The name of the spider that was used to download th article (internal debugging)
texthash	string	A hash of the full text (for duplicate detection)

Table 45: crawlers output

Preprocessing)

Output-offline	Kafka (input_preprocessed)
Notes	

Table 8: Preprocessing Implementation

6.4.4. COMPONENT TEST

To verify the functionality of the preprocessing several Unitest were executed using different kind of inputs to evaluate the performance of the component. More specifically, these set of tests were performed independently for both Offline and Online processes since the way they communicate to the rest of the component differs according to each pipeline. All the aforementioned tests were conducted in a development environment allocated at UPM's machines.

After the component where integrated on the platform we arranged integration tests performed by developers and infrastructure manager to check the behavior in the K8s cluster.

6.5. GRAPH ANALYSIS

The main goal of this component is to measure the level of credibility of a certain source by analyzing the most intrinsic information regarding such source including the domain, the suffix, the media type or even the behavior of the source or publisher throughout social networks such as Twitter.
The higher the score indicator, the higher the confidence level for a certain entity (either author or publisher). This service is distributed using a multithreading implementation to speed up separate operations.
First, the service reads the documents from the Kafka input_preprocessed queue and ingests the authors and organizations in Elasticsearch ensuring that they are not duplicates. Subsequently, Elasticsearch provides the set of identifiers for all the ingested items and, when identifiers are available, the associated objects are incorporated into the graph and the further analysis is launched…
The approach uses the so-called Neo4j as the graph framework for computing the scores that are used to measure the trustworthiness of both publishers and authors. Among the diverse features used for calculating the scores, the articleRank and the Eigen Centrality graph algorithms were selected to measure the rank or the power of every single node in the network. Using this information together with metrics of trustworthiness of the articles associated with them (from the text analyzer), it is possible to have a potential metric of trustworthiness.
Moreover, some metrics regarding the suffix of the domain are collected in order to infer the score based on this specific pattern of the URL (for instance to verify whether the domain corresponds to an educational source or a commercial one). On the other hand, UPM has collected a large list regarding the media type of some sources to weight the trustworthiness depending on this parameter (press agencies, newspapers, magazines etc.).
To improve the scoring, UPM analyses the Twitter account of the source in order to extract some features such as the level of activity of the account, the verification of the account or the level of popularity of the account based on the number of friends, followers or posted tweets.
Finally, by combining the aforementioned metrics, this service generates a scoring metric which can be explainable to end-users and it is objective according to the selected criteria of the analysis.
After analyzing the publisher, the authors are scored by combining the information of the publisher associated to them as well as the information provided by the scores of the articles which they have written until the moment of the current analysis.

Once the scores are calculated, the service updates them in Elasticsearch in the appropriate index.

Functionality	Graph Analysis
Technology	K8s/Docker
Containers	fandango_source_credibility
Language	Python

Input-online	API (Table 61: image get output Graph Analyzer)
Input-offline	Kafka (input_preprocessed)
Output-online	API (Table 61: image get output Graph Analyzer)
Output-offline	Kafka (analyzed_auth_org)
Notes	

Table 9: Graph Analysis Implementation

Functionality	Graph
Technology	K8s/Docker
Containers	neo4j
Language	External component integrated and configured for FANDANFO usage
Input-online	Invoked directly from Graph Analysis container (BOLT protocol)
Input-offline	Invoked directly from Graph Analysis container (BOLT protocol)
Output-online	Invoked directly from Graph Analysis container (BOLT protocol)
Output-offline	Invoked directly from Graph Analysis container (BOLT protocol)
Notes	

Table 10: Neo4J container

6.5.1. COMPONENT TEST

To verify the functionality of the graph analysis several Unitest were executed using different kind of inputs to evaluate the performance of the component. More specifically, these set of tests were performed independently for both Offline and Online processes since the way they communicate to the rest of the component differs according to each pipeline.

Moreover, since this service depends on Neo4j, several tests were conducted to consolidate the connection to this external component. This operation has been critical on K8s infrastructure due to the previous requirement between graph analysis container and Neo4j.

6.6. TEXT ANALYSIS AND WEB INTERFACE

This component predicts whenever an article is reliable or not by analyzing its text and title, returning a number between 0 and 1, where 1 is fully reliable and 0 the opposite. Text analysis component is based on a deep learning classifier, trained and validated with thousands of labelled articles annotated in 3 different ways. The classifier must handle 4 different languages and different articles topics.

Functionality	Text Analysis
Technology	K8s/Docker

<table>
<tr><td>Containers</td><td>fandango-fake-news</td></tr>
<tr><td>Language</td><td>Python/js/HTML</td></tr>
<tr><td>Input-online</td><td>API (Topic Analyzer, Table 61: image get output
Graph Analyzer, Table 55: video get output
Image Analyzer, Video Analyzer, Table 42: crawling and preprocessing output
Article Crawler, Table 45: crawlers output
Preprocessing)</td></tr>
<tr><td>Input-offline</td><td>Kafka (input_preprocessed)</td></tr>
<tr><td>Output-online</td><td>GUI/Elastic(Table 34: fdg-fusion-score index

6.7. FDG-TEXT-SCORE
<table><tr><th>Field</th><th>Type</th><th>Description</th></tr><tr><td>identifier</td><td>keyword</td><td>The identifier of the performed analy</td></tr><tr><td>textScore</td><td>float</td><td>It is a number to indicate the reliabili</td></tr><tr><td>relevance</td><td>float</td><td>It indicates the weight given to te calculation on basic statistics.</td></tr><tr><td>timestamp</td><td>date</td><td>Execution date</td></tr></table>Table 35: fdg-text-score index

fdg-article)</td></tr>
<tr><td>Output-offline</td><td>Kafka (analyzed_text)</td></tr>
<tr><td>Notes</td><td></td></tr>
</table>

Table 11: Text Analysis and Web Interface Implementation

The same container hosts the Fandango user interface as well. The GUI is composed by 5 different pages connected with a specific function, it has been thought and designed, reminding to some of the most famous search engines interface, in order to be an easy-to-use tool for final users. This configuration makes easier integration since a service script is visible only to its owner and just an output and input format must be defined a priori. The communication between the web interface and the service is proxied by web server's proxy pass roles.
Fandango web interface communicates with the back-end side (using HTTP protocol) which in turn manages information exchanges among micro-services.

6.7.1. COMPONENT TEST

The text analyzer has been verified through Unitests in local development environment. After the components were integrated on the platform, several integration tests were performed by developers and infrastructure manager to check the behavior in the K8s cluster.

All API's called by the UI have been tested first, using Postman application and integrated once the correct version of each one of them was checked.

The GUI has been tested by users in the UAT phase and in several usability test sessions.

6.8. MULTIMEDIA ANALYSIS

The multimedia engine performs analysis on video and image components of the news. The implemented analysis can be divided into two main categories by modality, namely descriptors and manipulation detectors for images and videos. The descriptors aim to provide the other components of the platform information about the visual content of the media in the articles, such as objects that appear in an image or the location where a video was shot. On the other hand, the manipulation detectors examine the available media to detect their authenticity and predict the probability that the multimedia object has been altered from its original form. The forecast for an image has details per pixel and for a video per frame and per pixel.

To provide a versatile system, we have implemented three media analysis components: one to perform image similarity analysis, one to perform video manipulation analysis and one to perform all other image analyses. The image analysis component relies to other individual components by forwarding analysis requests, therefore each component can be updated at will, without affecting other analysis types. An important consideration for this decision was the fact that individual analyzers can be implemented using different, usually conflicting, libraries.

The following tables describes the containers that perform the image analysis (the engines).

Functionality	Busternet Detection
Technology	K8s/Docker
Containers	busternet
Language	Python/TensorFlow
Input-online	Invoked by “Image Analysis” functionality
Input-offline	Invoked by “Image Analysis” functionality
Output-online	Invoked by “Image Analysis” functionality
Output-offline	Invoked by “Image Analysis” functionality
Notes	

Table 12: Burstnet Detection Image Analysis Implementation

Functionality	Face-forensics
Technology	K8s/Docker
Containers	face_recognition
Language	Python/TensorFlow
Input-online	Invoked by “Image Analysis” functionality
Input-offline	Invoked by “Image Analysis” functionality
Output-online	Invoked by “Image Analysis” functionality
Output-offline	Invoked by “Image Analysis” functionality
Notes	

Table 13:Face Forensics Image Analysis Implementation

Functionality	Mantranet Detection
Technology	K8s/Docker
Containers	mantranet
Language	Python/TensorFlow
Input-online	Invoked by “Image Analysis” functionality
Input-offline	Invoked by “Image Analysis” functionality
Output-online	Invoked by “Image Analysis” functionality

Output-offline	Invoked by “Image Analysis” functionality
Notes	

Table 14: Mantranet Detection Image Analysis Implementation

Functionality	Maskrcnn-detection
Technology	K8s/Docker
Containers	maskrcnn
Language	Python/TensorFlow
Input-online	Invoked by “Image Analysis” functionality
Input-offline	Invoked by “Image Analysis” functionality
Output-online	Invoked by “Image Analysis” functionality
Output-offline	Invoked by “Image Analysis” functionality
Notes	

Table 15: Maskrcnn Image Analysis Implementation

Functionality	Quadratic Detection
Technology	K8s/Docker
Containers	quadratic
Language	Python/TensorFlow
Input-online	Invoked by “Image Analysis” functionality
Input-offline	Invoked by “Image Analysis” functionality
Output-online	Invoked by “Image Analysis” functionality
Output-offline	Invoked by “Image Analysis” functionality
Notes	

Table 16:Quadratic detection Image Analysis Implementation

Functionality	Self Similarity
Technology	K8s/Docker
Containers	self_similarity
Language	Python/TensorFlow
Input-online	Invoked by “Image Analysis” functionality
Input-offline	Invoked by “Image Analysis” functionality
Output-online	Invoked by “Image Analysis” functionality
Output-offline	Invoked by “Image Analysis” functionality
Notes	

Table 17: Self Similarity Image Analysis Implementation

The image similarity module depends on the currently stored images in the system and therefore the produced results cannot be stored. It is more practical to use a synchronous process than to store the results and invalidate them after a small amount of time.

The image and video analysis modules are both resource and time consuming and as such they are implemented to work asynchronously. When called, they produce and return an identifier (promise) of where the results are going to be stored in the fdg-media index of elastic. While the results can be retrieved directly from the database, a helper endpoint is also offered to retrieve the results without prior knowledge of the database conventions. Additionally, to avoid analyzing the same media file multiple times, the

identifier is based on the media URI, thus, if a media object has already been analyzed or is scheduled for analysis, the existing analysis is used.

The three services described above are summarized in the following tables:

Functionality	Image Analysis
Technology	K8s/Docker
Containers	certh_image_analysis_service
Language	Python
Input-online	Invoked by “Online Multimedia” functionality
Input-offline	Invoked by “Offline Multimedia” functionality
Output-online	Invoked by “Online Multimedia” functionality
Output-offline	Invoked by “Offline Multimedia” functionality
Notes	

Table 18: Image Analysis Implementation

Functionality	Video Analysis
Technology	K8s/Docker
Containers	certh_video_analysis_service
Language	Python
Input-online	Invoked by “Online Multimedia” functionality
Input-offline	Invoked by “Offline Multimedia” functionality
Output-online	Invoked by “Online Multimedia” functionality
Output-offline	Invoked by “Offline Multimedia” functionality
Notes	

Table 19: Video Analysis Implementation

Functionality	Image Similarity
Technology	K8s/Docker
Containers	certh_image_similarity_service
Language	Python
Input-online	Invoked by “Online Multimedia” functionality
Input-offline	Invoked by “Offline Multimedia” functionality
Output-online	Invoked by “Online Multimedia” functionality
Output-offline	Invoked by “Offline Multimedia” functionality
Notes	

Table 20: Image Similarity Analysis Implementation

The video and image analyzers have been implemented as engines for analysis and can be shared between the online and offline processes. Two scripts were created, one to monitor the relevant Kafka queue and one to listen for requests from the UI, allowing for separation between engine and dispatching.

The following container acts as dispatcher for the offline process (get data news form Kafka and send to analyzers):

Functionality	Offline Multimedia
Technology	K8s/Docker
Containers	certh_offline_media_service

Language	Python
Input-online	
Input-offline	Kafka (text_preprocessed)
Output-online	
Output-offline	Kafka (analyzed_multimedia)
Notes	

Table 21: Offline Multimedia Implementation

The following container act as dispatcher for the online:

Functionality	Online Multimedia
Technology	K8s/Docker
Containers	certh_online_services
Language	Python

Input-online

*API (**Errore. L'origine riferimento non è stata trovata.**, Table 42: crawling and preprocessing output*

6.9. ARTICLE CRAWLER

6.9.1. END POINT

Method	Endpoint
POST	api/get_article

Table 43: crawlers end-point

6.9.2. INPUT

Field	Type	Description
URL	string	URL to an article

Table 44: crawlers input

6.9.3. OUTPUT

Field	Type	Description
identifier	string	The unique identifier given to the article
title	string	The extracted title of the article
description	string	Extended title of the article
summary	string	A summary of the article contents
text	string	The full text body of the article
keywords	list of string	A list of keywords provided by the publisher
language	string	The detected language the article is written in
authors	list of string	A list of authors for the article provided by the publish
date_created	datetime	The date and time in UTC that the request was created

date_modified	datetime	The date and time in UTC that the article was last modified (according to the publisher)	
date_published	datetime	The date and time that the article was first published (according to the publisher)	
publish_date_estimated	string	A "yes"/"no" of if the publish date was estimated by Fandango (or provided by the publisher)	
top_image	string	The URL to the most prominent image in the article	
images	list of string	A list of URLs to all the image media files in the article (may contain advertisements)	
videos	list of string	A list of URLs to all the video files in the article	
URL	string	The URL of the article	
source_domain	string	The source domain of the downloaded article (publisher)	
spider	string	The name of the spider that was used to download the article (internal debugging)	
texthash	string	A hash of the full text (for duplicate detection)	

Table 45: crawlers output

6.10. PREPROCESSING

6.10.1. END POINT

Method	Endpoint	Note
POST	api/preprocessing/offline/start	Offline proce
POST	api/preprocessing/online/preprocess_article	Online proce
POST	api/preprocessing/manual_annotation/preprocess_annotation	Manual anno process

Table 46: preprocessing end-points

6.10.1.1. INPUT

Field	Type	Description
message	string	Message response
status	integer	Code response
data	struct	Raw article object

Table 47: preprocessing input

6.10.1.2. INPUT MANUAL ANNOTATION

Field	Type	Description
title	string	Title of the article
text	string	Article body

	URL	string	URL of the article
	authors	struct	List of author's names

Table 48: manual annotation input

6.10.1.3. OUTPUT

Field	Type	Description
message	string	Message response
status	integer	Code response
data	struct	Pre-processed article object

Table 49: preprocessing output

Video Analyzer)

Input-offline

Output-online

*API (**Errore. L'origine riferimento non è stata trovata.**, Table 42: crawling and preprocessing output*

6.11. ARTICLE CRAWLER

6.11.1. END POINT

Method	Endpoint
POST	api/get_article

Table 43: crawlers end-point

6.11.2. INPUT

Field	Type	Description
URL	string	URL to an article

Table 44: crawlers input

6.11.3. OUTPUT

Field	Type	Description
identifier	string	The unique identifier given to the article
title	string	The extracted title of the article
description	string	Extended title of the article
summary	string	A summary of the article contents
text	string	The full text body of the article
keywords	list of string	A list of keywords provided by the publisher
language	string	The detected language the article is written in

authors	list of string	A list of authors for the article provided by the publisher	
date_created	datetime	The date and time in UTC that the request was created	
date_modified	datetime	The date and time in UTC that the article was last modified (according to the publisher)	
date_published	datetime	The date and time that the article was first published (according to the publisher)	
publish_date_estimated	string	A "yes"/"no" of if the publish date was estimated by Fandango (or provided by the publisher)	
top_image	string	The URL to the most prominent image in the article	
images	list of string	A list of URLs to all the image media files in the article (may contain advertisements)	
videos	list of string	A list of URLs to all the video files in the article	
URL	string	The URL of the article	
source_domain	string	The source domain of the downloaded article (publisher)	
spider	string	The name of the spider that was used to download the article (internal debugging)	
texthash	string	A hash of the full text (for duplicate detection)	

Table 45: crawlers output

6.12. PREPROCESSING

6.12.1. END POINT

Method	Endpoint	Note
POST	api/preprocessing/offline/start	Offline proce
POST	api/preprocessing/online/preprocess_article	Online proce
POST	api/preprocessing/manual_annotation/preprocess_annotation	Manual annc process

Table 46: preprocessing end-points

6.12.1.1. INPUT

Field	Type	Description
message	string	Message response
status	integer	Code response
data	struct	Raw article object

Table 47: preprocessing input

6.12.1.2. INPUT MANUAL ANNOTATION

Field	Type	Description

	title	string	Title of the article
	text	string	Article body
	URL	string	URL of the article
	authors	struct	List of author's names

Table 48: manual annotation input

6.12.1.3. OUTPUT

Field	Type	Description
message	string	Message response
status	integer	Code response
data	struct	Pre-processed article object

Table 49: preprocessing output

Video Analyzer)

Output-offline	
Notes	

Table 22: Online Multimedia Implementation

6.12.2. COMPONENT TEST

To verify the functionality of the media components, several Unitests were executed using different kind of inputs to evaluate the performance of the components. More specifically, the cases of unavailability or wrong encoding of the media object and the unavailability of the database were validated. Additionally, the correct communication between the components was validated.

After the components were integrated on the platform, we arranged integration tests performed by developers and infrastructure manager in order to check the behavior in the K8s cluster.

6.13. TOPICS ANALYSIS

The topics and the named entity component are responsible for providing the pipeline with information about the content included in the text of an article, claim or fact.

The input needed for the service is the clean text used in the article and the detected language in which the article is written. The output of the service consists of two lists of identifiers, the one corresponding to all the entities detected in the text and the other to the main topics that are discussed in the text based on the machine learning algorithm that has been developed and is described in document D4.1.

As the Multimedia analysis also in this case the engine is separated from the entry points (offline and online). The following image contains the core of the analysis:

Functionality	Topic Service
Technology	K8s/Docker

Containers	certh_topic_extraction_service
Language	Python/TensorFlow
Input-online	Invoked by “Online Multimedia” and “Offline Topics” functionality
Input-offline	Invoked by “Online Multimedia” and “Offline Topics” functionality
Output-online	Invoked by “Online Multimedia” and “Offline Topics” functionality
Output-offline	Invoked by “Online Multimedia” and “Offline Topics” functionality
Notes	

Table 23: Topic Service Implementation

The offline process has been implemented by the following container:

Functionality	Offline Topics
Technology	K8s/Docker
Containers	certh_offline_ner_service
Language	Python/TensorFlow
Input-online	
Input-offline	Kafka (text_preprocessed)
Output-online	
Output-offline	Kafka (analyzed_ner_topic)
Notes	

Table 24: Topic Analysis Manager Implementation

The online process has been included in the functionality “Online Multimedia”.

6.13.1. COMPONENT TEST

The functionality of the topic analyzer component was verified through Unitests. After the components were integrated on the platform, we arranged integration tests performed by developers and infrastructure manager to check the behavior in the K8s cluster.

6.14. FUSION SCORE ANALYSIS

The fusion score analysis component is responsible for calculating a final indicator of trustworthiness for the entire article. The input needed for the service is the identifier of the article in the database. The component communicates with the database and retrieves all the analyses that have been performed by the other components. The different outcomes are stored in a separate index in the database, along with metadata about the date and time that the analysis took place and which individual scores were considered for the scoring. The service returns a verification that the requested analysis had been started.

The description of the container is presented in the following table:

Functionality	Fusion Score Service
Technology	K8s/Docker
Containers	certh_fusion_score_service

Language	Python/TensorFlow
Input-online	“All other components”
Input-offline	“All other components”
Output-online	
Output-offline	
Notes	

Table 25:Fusion Score Service Implementation

6.14.1. COMPONENT TEST

The functionality of the fusion score analyzer component was verified through Unitests. The major consideration for these tests is the stability of the service regardless of if the analysis scores are stored in the old specification of the platform (synchronous) or the new (asynchronous) so that no data are lost during the transition. After the components were integrated on the platform, we arranged integration tests performed by developers and infrastructure manager to check the behavior in the K8s cluster.

6.15. AGGREGATOR AND KAFKA-CONNECTOR

The aggregator is a composition of two containers: fandango_kafka_elastic and ksql.

KSQL, is an extension of Kafka provided by Confluent that allows streaming query from and to different Kafka Topics. KSQL runs *continuous queries*, transformations that run continuously as new data passes through them, on streams of data in Kafka topics. In contrast, queries over a relational database are *one-time queries,* run once to completion over a data set, as in a SELECT statement on finite rows in a database.

The next graph illustrates a stream example:

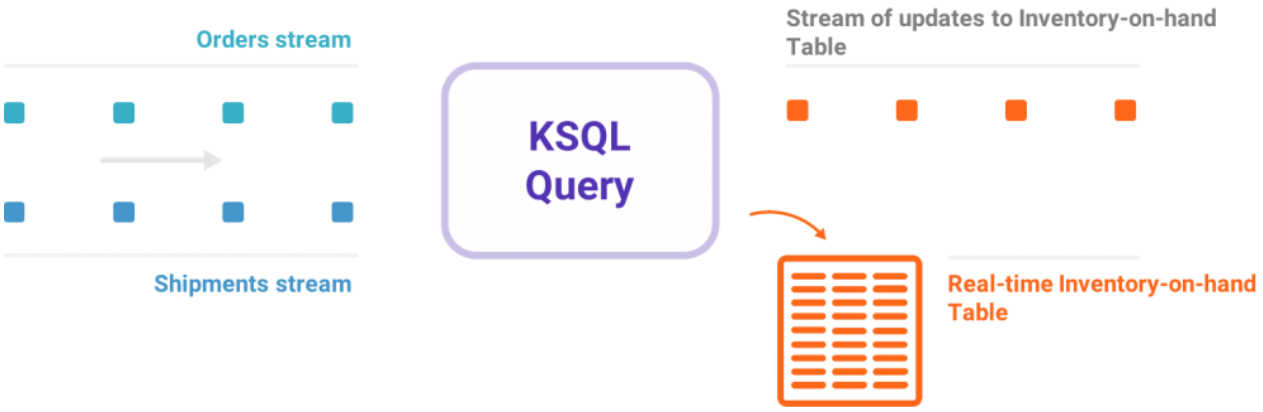

Figure 7: KSQL Behaviour

In Fandango we use KSQL to join the different result of the analysis and create a final joined topic. Here the graph of the queries:

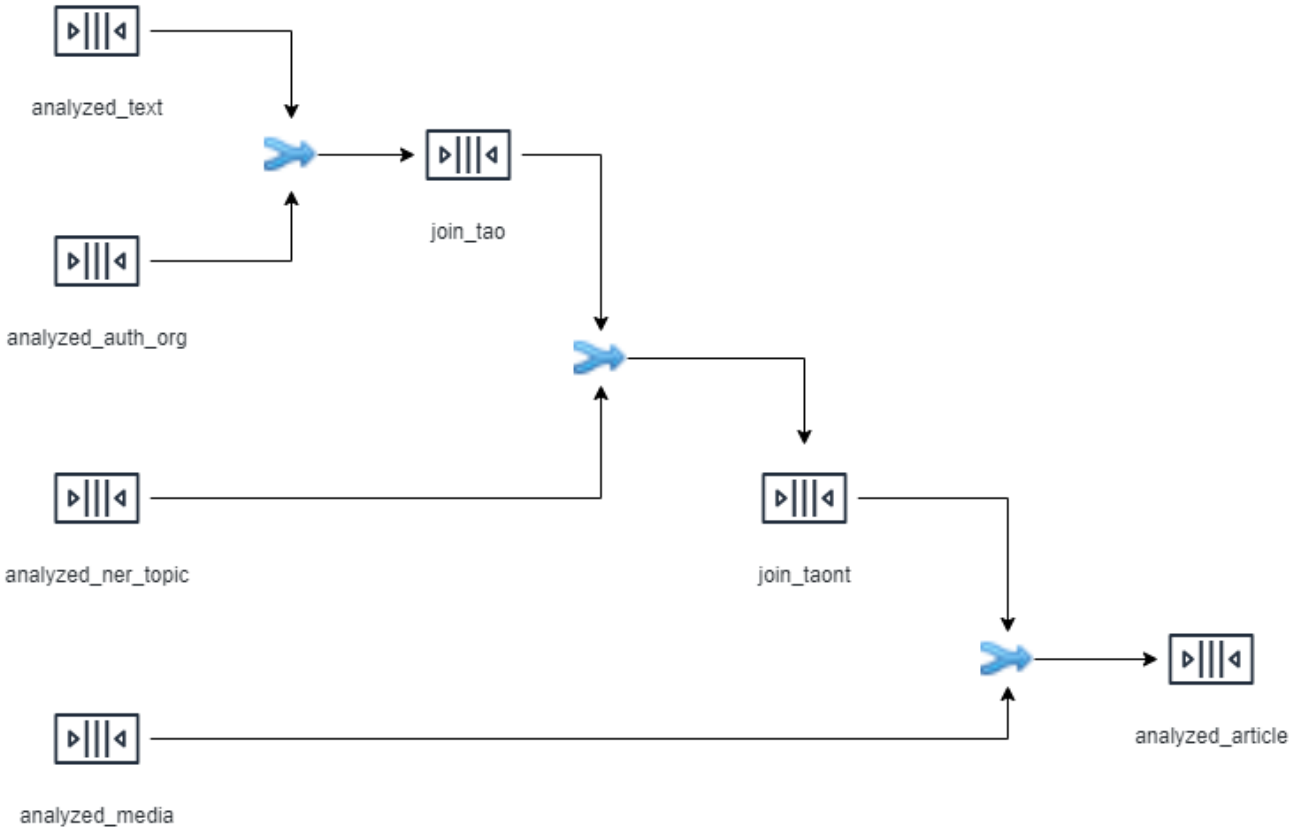

Figure 8: Aggregator Details

The result is then moved to Elastic using a consumer written in Python and hosted by the container fandango_kafka_elastic. We tested other solutions to write into elastic, in particular the component Kafka Connect (released by Confluent). However, it was too much resource consuming, so a light connector in Python was finally decided to be implemented.

The following two tables contains the details of the containers:

Functionality	Aggregator
Technology	K8s/Docker
Containers	ksql
Language	-
Input-online	
Input-offline	Kafka (analyzed-media, analyzed-text, analyzed-auth-org, analyzed-ner-topic)
Output-online	
Output-offline	Kafka (analyzed-article)
Notes	

Table 26: Aggregator Implementation

Functionality	Kafka Connector

<table>
<tr><td>Technology</td><td>K8s/Docker</td></tr>
<tr><td>Containers</td><td>fandango_kafka_elastic</td></tr>
<tr><td>Language</td><td>python</td></tr>
<tr><td>Input-online</td><td></td></tr>
<tr><td>Input-offline</td><td>Kafka (analyzed-article)</td></tr>
<tr><td>Output-online</td><td></td></tr>
<tr><td>Output-offline</td><td>Elastic (Table 34: fdg-fusion-score index

6.16. FDG-TEXT-SCORE
<table><tr><th>Field</th><th>Type</th><th>Description</th></tr><tr><td>identifier</td><td>keyword</td><td>The identifier of the performed analy</td></tr><tr><td>textScore</td><td>float</td><td>It is a number to indicate the reliabil</td></tr><tr><td>relevance</td><td>float</td><td>It indicates the weight given to te calculation on basic statistics.</td></tr><tr><td>timestamp</td><td>date</td><td>Execution date</td></tr></table>Table 35: fdg-text-score index

fdg-article)</td></tr>
<tr><td>Notes</td><td></td></tr>
</table>

Table 27: Kafka-Elastic Connection Implementation

6.16.1. COMPONENT TEST

The functionality followed the same test process of all the components in the infrastructure as defined in chapter [Infrastructure Tests].

6.17. KAFKA

As mentioned in the introduction paragraphs, FANDANGO has an architecture stream oriented, as it constantly captures and processes the data flow captured by the crawlers. Kafka is the component that coordinate all the Data Flow phases. More specifically, Kafka allows the ingestion, processing and storage tools to communicate via a distributed and reliable queue. This ensures high performance, component decoupling and reliability.

Further details regarding Kafka can be found in D5.1.

6.18. SIREN

Siren is the other component installed in a virtual machine outside of the K8s cluster.

In FANDANGO, Siren platforms is used to deliver fluid investigation capabilities to users trying to identify the trustworthiness of articles and claims. The main functionalities based on Siren platform are:

- Analytics about news articles with full text search capabilities and dashboarding, providing contextual analysis and drill-down mechanics to users investigating a group of articles
- Analytics about claims with full text search capabilities and dashboarding, providing contextual analysis and drill-down mechanics to users investigating certain claims
- Analytics for open data with full text search capabilities and dashboarding to improve user access to official information.
- Identification of related articles and claims that will support users by providing a bigger picture of the current scenario they are investigating.
- Knowledge-graph visualization to quickly identify related subjects, inter-connectivity, relevance, geographical spread and time-line progression of all aspects concerning article and claims publications.

The communication between Siren and the other components pass through Elastic: each analysis components writes in Elastic (see D3.1 for details on data model) and Siren load data from the indexes and build the interface.

In addition, Siren provides the article and claim similarity service:

Functionality	Article and Claim similarity service
Technology	Siren
Input-online	*API (Table 79: similar article output*

6.19. SIMILAR CLAIM, TOPIC ANALYZER

6.19.1. END POINT

Method	Endpoint
POST	api/topic_analysis

Table 65: topic post end-point

6.19.1.1. INPUT

Field	Type	Description
Identifier	string	Identifier of the article
articleBody	string	The text to be analyzed
Headline	string	Article headline

Table 66: topic post input

6.19.1.2. OUTPUT

Field	Type	Description
identifier	string	Identifier of the article
status	string	Current status of the analysis (200/400/500)
message	string	Explanation of status

Table 67: topic post output

6.19.2. END POINT

Method	Endpoint
GET	api/topic_analysis

Table 68: topic get end-point

6.19.2.1. INPUT

Field	Type	Description
identifier	string	Identifier of the article

Table 69: topic get input

6.19.2.2. OUTPUT

Field	Type	Description
identifier	string	Identifier of the article
about	list of string	A list of keywords to describe the text
topic	list of string	A list of identifiers to topics that match the input text
mentions	list of string	A list of identifiers of all the named entities that were detected in the text
status	string	Current status of the analysis (200/400/500)
message	string	Explanation of status

Table 70: topic get output

6.20. FUSION SCORE

6.20.1. END POINT

Method	Endpoint
POST	api/fusion_score

Table 71: fusion score post end-point

6.20.1.1. INPUT

Field	Type	Description
Identifier	string	The article to be analyzed

Table 72: fusion score post input

6.20.1.2. OUTPUT

Field	Type	Description
status	string	Current status of the analysis (200/400/500)
message	string	Explanation of status

Table 73: fusion score post output

6.20.2. END POINT

Method	Endpoint
GET	api/ fusion_score

Table 74: fusion score post end-point

6.20.2.1. INPUT

Field	Type	Description
identifier	string	Identifier of the article

Table 75: fusion score input

6.20.2.2. OUTPUT

Field	Type	Description
identifier	string	Identifier of the article
calculatedRating	integer	Score given from the analyser (percentage of authenticity)
calculatedRatingDetail	string	Reasoning of why the score was given

Table 76: fusion score output

Similar Article)

Input-offline

Output-online

API (Table 79: similar article output

6.21. SIMILAR CLAIM, TOPIC ANALYZER

6.21.1. END POINT

Method	Endpoint
POST	api/topic_analysis

Table 65: topic post end-point

6.21.1.1. INPUT

Field	Type	Description
Identifier	string	Identifier of the article
articleBody	string	The text to be analyzed
Headline	string	Article headline

Table 66: topic post input

6.21.1.2. OUTPUT

Field	Type	Description
identifier	string	Identifier of the article
status	string	Current status of the analysis (200/400/500)
message	string	Explanation of status

Table 67: topic post output

6.21.2. END POINT

Method	Endpoint
GET	api/topic_analysis

Table 68: topic get end-point

6.21.2.1. INPUT

Field	Type	Description
identifier	string	Identifier of the article

Table 69: topic get input

6.21.2.2. OUTPUT

Field	Type	Description
identifier	string	Identifier of the article
about	list of string	A list of keywords to describe the text
topic	list of string	A list of identifiers to topics that match the input text
mentions	list of string	A list of identifiers of all the named entities that were detected in the text
status	string	Current status of the analysis (200/400/500)
message	string	Explanation of status

Table 70: topic get output

6.22. FUSION SCORE

6.22.1. END POINT

Method	Endpoint
POST	api/fusion_score

Table 71: fusion score post end-point

6.22.1.1. INPUT

Field	Type	Description
Identifier	string	The article to be analyzed

Table 72: fusion score post input

6.22.1.2. OUTPUT

Field	Type	Description
status	string	Current status of the analysis (200/400/500)
message	string	Explanation of status

Table 73: fusion score post output

6.22.2. END POINT

Method	Endpoint
GET	api/ fusion_score

Table 74: fusion score post end-point

6.22.2.1. INPUT

Field	Type	Description
identifier	string	Identifier of the article

Table 75: fusion score input

6.22.2.2. OUTPUT

Field	Type	Description
identifier	string	Identifier of the article
calculatedRating	integer	Score given from the analyser (percentage of authenticity)
calculatedRatingDetail	string	Reasoning of why the score was given

Table 76: fusion score output

Similar Article)

Output-offline	
Notes	Siren's similarity services leverage Elasticsearch's "More Like This" advanced search api. "More Like This" employs a tf-idf approach to search to return search results containing greater contextual similarity to the user's querying document.

Table 28: Similarity Services Implementation

6.22.3. COMPONENT TEST

The services have been extensively tested by front loading test indices with documents with known content, then searching multiple documents with varying relevance against the test indices to verify accuracy.

This accuracy is observed to increase dramatically as the number of documents stored in each index (article and claim) increases.

The front-end implementations and customizations have been tested following the approach defined in chapter [Infrastructure Tests]. More specifically, considering the nature of the implementations, it has been stressed the Acceptance Tests by Final Users.

7. ELASTIC INDEXES

This chapter contains the details of the actual structure of the Indexes in Elastic and it's an evolution of the document FANDANGO_D3.1API. Consider that the project is still on going, so there could be modification in the last period.

7.1. FDG-MEDIA

Field	Type	Description
identifier	text	Unique identifier for the analysis
contentUrl	text	URL where the analysed media was retrieved
status	text	Current status of the analysis (start/download/analyzing/end/error)
type	text	Type of the media file (image/video)
encodingFormat	text	Encoding format of the media
contentSize	text	Size of the media file
uploadDate	text	Date the media file was uploaded
display	text	where the overlaid image or video will be stored

Table 29: fdg-media index

7.2. FDG-TOPIC

Field	Type	Description
identifier	text	Unique identifier for the analysis
contentUrl	text	URL where the analysed media was retrieved
status	text	Current status of the analysis (start/download/analyzing/end/error)
type	text	Type of the media file (image/video)
encodingFormat	text	Encoding format of the media
contentSize	text	Size of the media file
uploadDate	text	Date the media file was uploaded

Table 30: fdg-topic index

7.3. FDG-ENTITY

Field	Type	Description
identifier	text	Unique identifier for the analysis
contentUrl	text	URL where the analysed media was retrieved
status	text	Current status of the analysis (start/download/analysing/end/error)
type	text	Type of the media file (image/video)

encodingFormat	text	Encoding format of the media
contentSize	text	Size of the media file
uploadDate	text	Date the media file was uploaded

Table 31: fdg-entity index

7.4. FDG-AP-PERSON

Field	Type	Description
Identifier	Text	Unique alphanumeric identifier of the item.
Name	Text	The name of the person.
URL	Text	URL of the person.
Nationality	Text	Nationality of the person.
Bias	Text	
jobTitle	Text	The job title of the person.
gender	Text	Gender of the person.
affiliation	array of text (FK fdg-ap-organization)	An organization that this person is affiliated with.
trustworthiness	float	Score of how trustworthy the author is
relevance	Float	How relevant the trustworthiness is

Table 32: fdg-ap-person index

7.5. FDG-AP-ORGANIZATION

Field	Type	Description
identifier	Text	The alphanumeric identifier of an organization.
name	Text	The name of the organization.
URL	Text	URL of the organization.
nationality	Text	Nationality of the organization.
country	Text	Country of the organization.
bias	Text	
parentOrganization	text (FK - fdg-ap-organization)	The larger organization that this organization is a sub-organization of, if any.
trustworthiness	Float	Score of how trustworthy the organization is
relevance	Float	How relevant the trustworthiness is

Table 33: fdg-ap-organization index

7.6. FDG-FUSION-SCORE

Field	Type	Description
identifier	keyword	The identifier of the performed analysis

analyzer	text	The analyzer that executed the analysis
calculatedRating	float	Rating of trustworthiness calculated by FANDANGO for the article.
calculatedRatingDetail	text	Rating of trustworthiness calculated by FANDANGO broken down into different evaluated categories.
timestamp	date	Execution date

Table 34: fdg-fusion-score index

7.7. FDG-TEXT-SCORE

Field	Type	Description
identifier	keyword	The identifier of the performed analysis
textScore	float	It is a number to indicate the reliability of an article
relevance	float	It indicates the weight given to text score based on calculation on basic statistics.
timestamp	date	Execution date

Table 35: fdg-text-score index

7.8. FDG-ARTICLE

Field	Type	Description
identifier	Text	The alfanumeric identifier of an article.
headline	Text	Headline of the article.
articleBody	Text	The actual body of the article.
URL	Text	The URL of article
sourceDomain	Text	The source of article
inLanguage	Text	The text that indicate the language
publishDateEstimated	Text	
dateCreated	Date	The date on which the article was created, or the item was added to a DataFeed.
dateModified	Date	The date on which the Article was most recently modified, or when the item's entry was modified within a DataFeed.
datePublished	Date	Date of first broadcast/publication.
author	array of text (FK - fdg-ap-person or fdg-ap-organization)	The author of this content.
publisher	array of text (FK - fdg-ap-person or fdg-ap-organization)	The publisher of the article.
about	array of text (FK - Topic)	Topics to which the article talks about.
mentions	array of text (FK - Entity)	Entities mentioned by the article.

contains	array of text (FK - Media)	Media files like audio, videos and images that are part of the article.

Table 36: fdg-article index

7.9. FDG-CLAIM

Field	Type	Description
identifier	text	The alfaNumeric identifier of a claim.
appearance	array of text (FK - Article)	Indicates an occurrence of a claim in some article.
firstAppearance	text (FK - Article)	Indicates the first known occurrence of a claim in some article.
text	text	The textual content of this claim.
dateCreated	date	The date on which the claim was created, or the item was added to a DataFeed.
dateModified	date	The date on which the claim was most recently modified, or when the item's entry was modified within a DataFeed.
datePublished	date	Date of first broadcast/publication.
about	array of text (FK - Topic)	Topics that the claim talks about.
mentions	array of text (FK - Entity)	Entities mentioned by the claim.
possiblyRelatesTo	array of text (FK - Fact)	Facts that are possibly related to the claim.

Table 37: fdg-claim index

7.10. FDG-CLAIM-REVIEW

Field	Type	Description
identifier	text	The alfaNumeric identifier of a claim review.
claimReviewed	text	A short summary of the specific claims reviewed in a ClaimReview.
reviewAspect	text	This review is relevant to this part or facet of the claim reviewed.
reviewBody	text	The actual body of the review.
dateCreated	date	The date on which the claim was created, or the item was added to a DataFeed.
dateModified	date	The date on which the claim was most recently modified, or when the item's entry was modified within a DataFeed.
datePublished	date	Date of first broadcast/publication.
aggregateRating	number	The overall rating of this review, based on a collection of reviews or ratings, of it.
reviewRating	number	The review rating value
itemReviewed	text (PK - Claim)	The claim that is being reviewed/rated.
references	array of text(FK - Fact)	Facts referenced in this review.
URL	Text	The URL of claim

Table 38: fdg-claim-review index

7.11. FDG-FACT

Field	Type	Description
identifier	text	The alfanumerical identifier of a fact.
Name	text	The name of the fact.
Text	text	The textual content of this fact.
URL	Text	URL of the fact.
About	array of text (FK - Topic)	Topics that the fact talks about.
mentions	array of text (FK - Entity)	Entities mentioned by the fact.
dateCreated	date	The date on which the fact was created, or the item was added to a DataFeed.
dateModified	date	The date on which the fact was most recently modified, or when the item's entry was modified within a DataFeed.
datePublished	date	Date of first broadcast/publication.
temporalCoverageStart	date	The start date and time of the fact.
temporalCoverageEnd	date	The end date and time of the
spatialCoverage	text	The spatialCoverage of a fact indicates the place(s) which are the focus of the content.

Table 39: fdg-fact index

8. API

This chapter contains the details of the actual definition of the APIs exposed by the services and it contains an evolution of the document FANDANGO_D3.1API and it's important for this deliverable because it represent input/output for the processes described in the previous chapters. Consider that the project is still on going, so there could be modification in the last period.

8.1. CRAWLING AND PREPROCESSING

This API is used for the communication between the webserver and the GUI. It is used from the GUI to ask the backend to execute all processing phases (crawl, preprocess, analyze and write result on Elastic), and give back the result to display.

8.1.1. END POINT

Method	Endpoint
POST	api/crawl_and_preprocessing

Table 40: crawling and preprocessing end-point

8.1.2. INPUT

Field	Type	Description
URL	string	article URL to crawl, preprocess and get the analysis result

Table 41: crawling and preprocessing input

8.1.3. OUTPUT

Field	Type	Description
news_preprocessed	object	It contains all the information to fulfil all
Opendata	object	list of author identifiers

Table 42: crawling and preprocessing output

8.2. ARTICLE CRAWLER

8.2.1. END POINT

Method	Endpoint
POST	api/get_article

Table 43: crawlers end-point

8.2.2. INPUT

Field	Type	Description
URL	string	URL to an article

Table 44: crawlers input

8.2.3. OUTPUT

Field	Type	Description
identifier	string	The unique identifier given to the article
title	string	The extracted title of the article
description	string	Extended title of the article
summary	string	A summary of the article contents
text	string	The full text body of the article
keywords	list of string	A list of keywords provided by the publisher
language	string	The detected language the article is written in
authors	list of string	A list of authors for the article provided by the publisher
date_created	datetime	The date and time in UTC that the request was created
date_modified	datetime	The date and time in UTC that the article was last modified (according to the publisher)
date_published	datetime	The date and time that the article was first published (according to the publisher)
publish_date_estimated	string	A "yes"/"no" of if the publish date was estimated by Fandango (or provided by the publisher)
top_image	string	The URL to the most prominent image in the article
images	list of string	A list of URLs to all the image media files in the article (may contain advertisements)
videos	list of string	A list of URLs to all the video files in the article
URL	string	The URL of the article
source_domain	string	The source domain of the downloaded article (publisher)
spider	string	The name of the spider that was used to download the article (internal debugging)
texthash	string	A hash of the full text (for duplicate detection)

Table 45: crawlers output

8.3. PREPROCESSING

8.3.1. END POINT

Method	Endpoint	Note
POST	api/preprocessing/offline/start	Offline process
POST	api/preprocessing/online/preprocess_article	Online process
POST	api/preprocessing/manual_annotation/preprocess_annotation	Manual annotation process

Table 46: preprocessing end-points

8.3.1.1. INPUT

Field	Type	Description
message	string	Message response
status	integer	Code response
data	struct	Raw article object

Table 47: preprocessing input

8.3.1.2. INPUT MANUAL ANNOTATION

Field	Type	Description
title	string	Title of the article
text	string	Article body
URL	string	URL of the article
authors	struct	List of author's names

Table 48: manual annotation input

8.3.1.3. OUTPUT

Field	Type	Description
message	string	Message response
status	integer	Code response
data	struct	Pre-processed article object

Table 49: preprocessing output

8.4. VIDEO ANALYZER

8.4.1. END POINT

Method	Endpoint
POST	api/video_analysis

Table 50: video post end-point

8.4.1.1. INPUT

Field	Type	Description
URL	string	The URL to a video file (youtubedl)
video_id	string	Desired unique identifier for the analysis (optional)
force	boolean	Indication that the analysis should be performed even if it has been performed already (default: False)
first_frame	integer	First frame to be analyzed (default: first frame of video)

last_frame	integer	Last frame to be analyzed (default: last frame of video)
Fps	integer	Frames per second to analyze (default: 3 frames per second)

Table 51: video post input

8.4.1.2. OUTPUT

Field	Type	Description
identifier	string	Unique identifier for the analysis
status	string	Current status of the analysis (200/400/500)
message	string	Explanation of status

Table 52: video post output

8.4.2. END POINT

Method	Endpoint
GET	api/video_analysis

Table 53: video get end-point

8.4.2.1. INPUT

Field	Type	Description
identifier	string	Unique identifier of the analysis

Table 54: video get input

8.4.2.2. OUTPUT

Field	Type	Description
identifier	string	Unique identifier for the analysis
contentUrl	string	URL where the analyzed video can be downloaded
status	string	Current status of the analysis (start/download/analyzing/end/error)
type	string	Type of the media file (video)
encodingFormat	string	Encoding format of the video
contentSize	string	Size of the video file
uploadDate	string	Date the video file was uploaded
calculatedRating	integer	Score given from the analyzer (percentage of authenticity)
calculatedRatingDetail	string	Reasoning of why the score was given
progress	float	Percentage of analysis that has been performed

Table 55: video get output

8.5. IMAGE ANALYZER

8.5.1. END POINT

Method	Endpoint
POST	api/image_analysis

Table 56: image post end-point

8.5.1.1. INPUT

Field	Type	Description
URL	string	The URL to an image file
image	base64	An image file to analyze

Table 57: image post input

8.5.1.2. OUTPUT

Field	Type	Description
identifier	string	Unique identifier for the analysis
status	string	Current status of the analysis (200/400/500)
message	string	Explanation of status

Table 58: image post output

8.5.2. END POINT

Method	Endpoint
GET	api/image_analysis

Table 59: image get end-point

8.5.2.1. INPUT

Field	Type	Description
identifier	string	Unique identifier of the analysis

Table 60: image get input

8.5.2.2. OUTPUT

Field	Type	Description
identifier	string	Unique identifier for the analysis
contentUrl	string	URL where the analyzed image can be downloaded
status	string	Current status of the analysis (start/download/analyzing/end/error)
type	string	Type of the media file (image)
encodingFormat	string	Encoding format of the image
contentSize	string	Size of the image file
uploadDate	string	Date the image file was uploaded

calculatedRating	integer	Score given from the analyzer (percentage of authenticity)
calculatedRatingDetail	string	Reasoning of why the score was given
progress	float	Percentage of analysis that has been performed

Table 61: image get output

8.6. GRAPH ANALYZER

8.6.1. ENDPOINT

Method	Endpoint	Note
POST	api/graph_analysis/offline/start	Offline process
POST	api/graph_analysis/online/analyze_article	Online process

Table 62: graph end-points

8.6.2. INPUT

Field	Type	Description
data	struct	Output of the preprocessing service

Table 63: graph input

8.6.3. OUTPUT

Field	Type	Description
message	string	Message response
status	integer	Code response
data	struct	Output of the graph analysis

Table 64: graph output

8.7. TOPIC ANALYZER

8.7.1. END POINT

Method	Endpoint
POST	api/topic_analysis

Table 65: topic post end-point

8.7.1.1. INPUT

Field	Type	Description
Identifier	string	Identifier of the article
articleBody	string	The text to be analyzed
Headline	string	Article headline

Table 66: topic post input

8.7.1.2. OUTPUT

Field	Type	Description
identifier	string	Identifier of the article
status	string	Current status of the analysis (200/400/500)
message	string	Explanation of status

Table 67: topic post output

8.7.2. END POINT

Method	Endpoint
GET	api/topic_analysis

Table 68: topic get end-point

8.7.2.1. INPUT

Field	Type	Description
identifier	string	Identifier of the article

Table 69: topic get input

8.7.2.2. OUTPUT

Field	Type	Description
identifier	string	Identifier of the article
about	list of string	A list of keywords to describe the text
topic	list of string	A list of identifiers to topics that match the input text
mentions	list of string	A list of identifiers of all the named entities that were detected in the text
status	string	Current status of the analysis (200/400/500)
message	string	Explanation of status

Table 70: topic get output

8.8. FUSION SCORE

8.8.1. END POINT

Method	Endpoint
POST	api/fusion_score

Table 71: fusion score post end-point

8.8.1.1. INPUT

Field	Type	Description
Identifier	string	The article to be analyzed

Table 72: fusion score post input

8.8.1.2. OUTPUT

Field	Type	Description
status	string	Current status of the analysis (200/400/500)
message	string	Explanation of status

Table 73: fusion score post output

8.8.2. END POINT

Method	Endpoint
GET	api/ fusion_score

Table 74: fusion score post end-point

8.8.2.1. INPUT

Field	Type	Description
identifier	string	Identifier of the article

Table 75: fusion score input

8.8.2.2. OUTPUT

Field	Type	Description
identifier	string	Identifier of the article
calculatedRating	integer	Score given from the analyser (percentage of authenticity)
calculatedRatingDetail	string	Reasoning of why the score was given

Table 76: fusion score output

8.9. SIMILAR ARTICLE

8.9.1. END POINT

Method	Endpoint
POST	/siren/FindSimilarArticles

Table 77: similar article end-point

8.9.2. INPUT

Field	Type	Description
identifier	string	uid of article

Table 78: similar article input

8.9.3. OUTPUT

Field	Type	Description
identifier	string	uid of searched article

results	array of objects	resulting articles
results.article	object	each resulting similar article
identifier	string	
articleBody	string	
author	object	corresponding entity from fdg-ap-person
about	array of strings	
inLanguage	string	
dateModified	string	
calculatedRatingDetail	object of numbers	
URL	String	
datePublished	String	
calculatedRating	number	
contains	array of strings	
dateCreated	String	
publishDateEstimated	String	
mentions	array of strings	
topic	String	
publisher	object	corresponding entity from fdg-ap-roganization
sourceDomain	String	
headline	String	

Table 79: similar article output

8.10. SIMILAR CLAIM

8.10.1. END POINT

Method	Endpoint
POST	/siren/FindSimilarClaims

Table 80: similar claim end-point

8.10.2. INPUT

Field	Type	Description
identifier	string	uid of claim
text	string	free text input by user
topics	array of strings	a list of keywords provided/selected by user

Table 81: similar claim input

8.10.3. OUTPUT

Field	Type	Description
identifier	String	uid of claim from which a user may have searched
topics	array of strings	topics specified in input search
results	array of objects	A list of Similar Claims
results.claim	object	Each individual Claim object
firstAppearance	string	
identifier	string	
claimReviews	array of objects	
about	array of strings	
calculatedRatingDetail	object	
calculatedRatingDetail.authorRating	number	
calculatedRatingDetail.publishRating	number	
dateModified	string	
calculatedRating	number	
datePublished	string	
appearance	array of strings	
dateCreated	string	
mentions	array of strings	
text	string	
possiblyRelatesTo	array of strings	

Table 82: similar claim output

9. CONCLUSION

Several iterative prototyping cycles leaded to the setup of a first full-featured implementation of FANDANGO that has been used to run the first pilot.

The implementation and testing of the different prototypes led to an architectural revision focused on Kubernetes (k8s), Kafka and Elasticsearch. In particular, the various components have been translated from a Cluster architecture based on Hadoop / Spark to a cluster architecture based on Kubernetes.

In particular, the introduction of K8s required a revision of the initial cluster structure with a view to containers. This involved the revision of the algorithms favouring a microservice-like horizontal scaling rather than a parallelization at the execution engine level (Spark). The consequence is that some components such as NiFi and Spark have been excluded from the infrastructure.

Even if the logical design and data flow have been slightly modified to gain better performances, the infrastructure has been profoundly changed and both the virtualized HW infrastructure (IaaS) and the PaaS platform have been described in detail in the previous sections of this document.

The ingestion process has been revised to remove potential cause of data loss while preserving the analyzers to run in parallel and asynchronously. This revision has impacted the implementation of the components involved into the ingestion process, specifically in terms of Data Model and APIs. All these changes have been reported into this document.

The last infrastructure released has been considered consolidated and definitive. Due to the iterative method adopted, the components might suffer of slightly modifications that will have no impact on the architecture.

Any future changes will be reported in following deliverables.