A Software Architecture for Extreme-ScaLe
Big-Data AnalyticS in Fog CompuTIng Ecosystems

# D3.1 Software architecture requirements and integration plan

## Version 1.0

### Document Information

| | |
|---|---|
| **Contract Number** | 825473 |
| **Project Website** | https://elastic-project.eu/ |
| **Contractual Deadline** | M6, May 2019 |
| **Dissemination Level** | PU |
| **Nature** | R |
| **Author(s)** | Eduardo Quiñones, Sara Royuela (BSC) |
| **Contributor(s)** | Usman Wajid, John Beattie (ICE); Cristóvão Cordeiro (SIX); Cristina Zubia (IKL); Luis Miguel Pinho, Luis Nogueira, António Barros (ISEP) |
| **Reviewer(s)** | Cristóvão Cordeiro (SIX) |
| **Keywords** | software development ecosystem, software architecture, integration plan |

***Notices:*** *The ELASTIC project has received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement Nº 825473.*

# Change Log

| Version | Author | Description of Change |
|---|---|---|
| V0.1 | Eduardo Quiñones, Sara Royuela (BSC) | Initial Draft |
| V0.2 | Usman Wajid, John Beattie (ICE) | Updated information about distributed data analytics |
| V0.3 | Jürgen Assfalg (FLO) | Small corrections |
| V0.4 | Cristóvão Cordeiro (SIX) | Nuvla and NuvlaBox description |
| V0.5 | Cristina Zubia (IKL) | KonnektBox, Monitor and Data Router description |
| V0.6 | Luis Miguel Pinho, Luis Nogueira, António Barros (ISEP) | Small corrections, NFR technical requirement and NFR tool |
| V1.0 | Eduardo Quiñones (BSC) | Ready for submission to the EC |
| | | *(Final Change Log entries reserved for releases to the EC)* |
| | | |
| | | |
| | | |

# Table of contents

# 1. Executive Summary

This deliverable covers the work done during the first phase of the project within WP3. The deliverable spans 6 months of work and handles the work done in Task 3.1 "*Software architecture requirements specification*" to reach milestone MS1.

This deliverable describes the **ELASTIC software development ecosystem** upon which the ELASTIC use-case will be developed, deployed and executed. Concretely, it identifies the set of software components and tools that will form the ELASTIC ecosystem, and it provides a short description of each component.

One of the key features of the ELASTIC ecosystem will be its capability to instantiate multiple software architecture configurations, incorporating different software components. The objective of this feature is to cover different system necessities to properly coordinate edge and cloud resources, while guaranteeing the non-functional requirements (NFR) imposed by the cyber-physical interactions. This key feature requires each software component to define a clear interface, described in this deliverable, to ensure a seamless integration among them.

Finally, this deliverable also provides the *development and integration plan* for the ELASTIC project. It includes the tools, techniques and methodologies that will be shared among all partners in order to ensure the quality of the final product.

The first milestone of Task 3.1 has been carried out successfully and all objectives of MS1 have been reached and documented in this deliverable.

# 2. The ELASTIC Software Development Ecosystem

## 2.1 Overview

ELASTIC addresses each of the requirements defined in Section 3 by developing a software development ecosystem incorporating software components from multiple computing areas, including distributed data analytics, embedded computing, internet of things (IoT), cyber-physical systems (CPS), software engineering, high-performance computing (HPC), and edge and cloud technologies. These unique (and heterogeneous) combination of software components will enable to efficiently distribute extreme-scale big-data analytics across the compute continuum, from edge to cloud, and provide guarantees on the non-functional requirements of the system.

Figure 1 shows the overall ELASTIC software development ecosystem including the main software components. Concretely, the ELASTIC ecosystem will consist of:

- The **distributed data analytics platform (WP2)** upon which the data analytics methods implemented in the ELASTIC use-cases (WP1) will execute. This platform will also incorporate any interface that the software components may require to expose to the programmer.
- The **orchestrator layer (WP3)**, which will incorporate the COMPSs run-time, will be the responsible of implementing the key innovation of ELASTIC: a new *elasticity concept* capable of efficiently distributing the computation (provided by the data analytics platform) across the compute continuum (through the hybrid fog computing platform), while fulfilling the system's NFR.

- The orchestration layer will consider the input from the **NFR tool (WP4)**, that will (statically) analyse the internal structure of application implementing the data analytics and (dynamically) monitor the execution of the system using the hybrid fog computing platform capabilities.
- The **hybrid fog computing platform (WP5)** will be in charge of deploying the computation across the compute continuum based on the distribution provided by the orchestration layer. The deployment will be performed by *Nuvla* which will consider two interfaces at the edge level: the *NuvlaBox* and the *KonnektBox*. The former will only support microservices provided by Docker (or similar); the latter will support both microservices and monolithic native services provided by Linux (or similar). Finally, the hybrid fog computing platform will incorporate a data router service to feed the data analytics methods with data coming from the geographically distributed sensors. Here, *dataClay* will be used to implement a storage distributed system to guarantee that the data is available across the compute continuum.

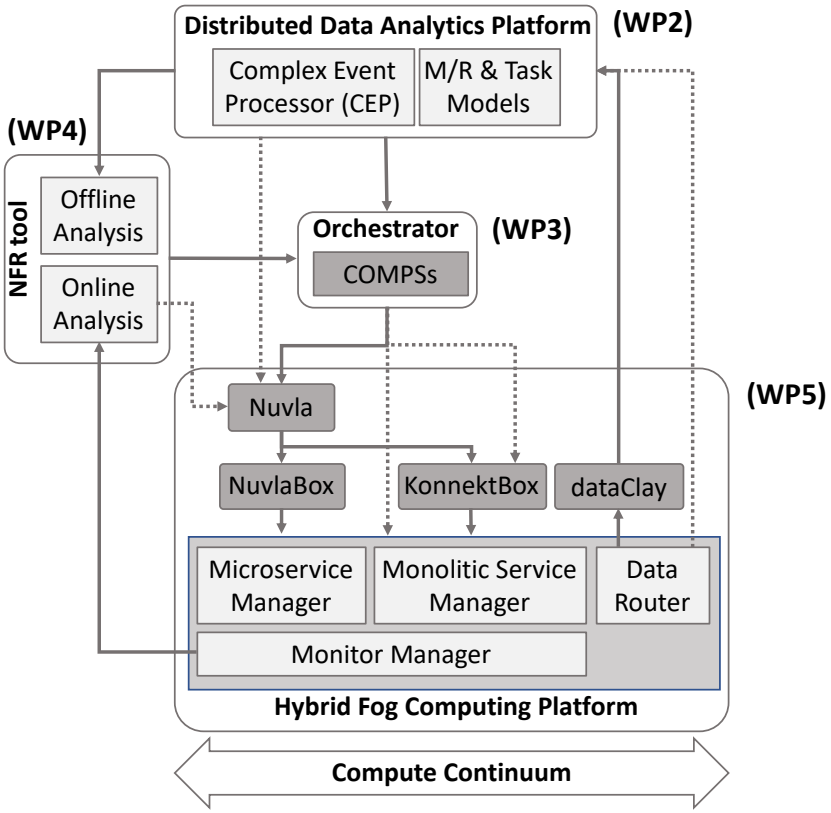


*Figure 1. ELASTIC Software Development Ecosystem.*

In the figure, the solid arrows represent the interaction among all the software components intended to be integrated within the complete ELASTIC software architecture. However, as stated above, the ELASTIC ecosystem will allow to instantiate other software architectures formed by a subset of software components. This is represented with the dashed arrows in the figure. For example, the software architecture could be instantiated without COMPSs, Nuvla or dataClay. In this case, these other instantiations will not include all the functionalities intended to be developed by ELASTIC. An example of a software architecture not featuring all ELASTIC functionalities is a system in which the data does not need to be distributed across the compute continuum but resides in a centralized resource. In this case, the

Data Router can bypass dataClay. In this case however, no distributed storage features will be included in the resultant software architecture.

Overall, the principle behind this flexible ecosystem is to enable the creation of multiple instantiations of the ELASTIC software architecture to cover different requirements of the potential "users" of the ELASTIC technology. Moreover, this principle will facilitate the interoperability, portability, and scalability of the ELASTIC software components and so it will increase the potential exploitation opportunities in the near future.

## 2.2 ELASTIC Software Components

ELASTIC has carefully selected the software components that will form the ELASTIC ecosystem, prioritizing those owned by the ELASTIC partners or offered as open-source with a large community behind. By doing so, we envision to reduce the time-to-market and maximize exploitation opportunities of the ELASTIC ecosystem. Table 1 identifies the set of software components that will be included in the ELASTIC ecosystem, the WP in which the component will be evaluated, the owner and the license.

*Table 1. ELASTIC Software Components.*

| Software Component | | WP | Owner | License |
|---|---|---|---|---|
| Distributed Data Analytics Platform | COMPSs | WP3 | BSC | Open-source |
| | Flink | WP2 | Apache Software Foundation | Open-source |
| | Spark | | | Open-source |
| | Kafka | | | Open-source |
| Orchestrator | COMPSs | WP3 | BSC | Open-source |
| NFR tool | Static Analysis tools | WP4 | ISEP | Open-source |
| | Run-time analysis tools | | ISEP | Open-source |
| Hybrid Fog Computing Platform | Nuvla/NuvlaBox | WP5 | SixSQ | Open-source |
| | KonnektBox | | IKL | Proprietary |
| | dataClay | | BSC | Open-source |
| | Kubernetes | | Linux Foundation | Open-source |
| | Docker | | Docker Inc. | Open-source |

In the next subsections, the software components are described in detail. Further details on components that belong to specific WPs may be found in the respective design/requirement document of the WP for the first phase of the project, i.e., within deliverables D2.5 [1], D4.2 [2], and D5.1 [3]. The COMPSs component is presented in this deliverable in Section 5.

## 2.3 Distributed Data Analytics Platform

This section describes the software components included in the ELASTIC software development ecosystem to implement data analytics applications. See Deliverable D2.5 [1] for further information.

### 2.3.1 Map/Reduce and Task-based APIs

The ELASTIC ecosystem will incorporate two well-known programming models capable of exploiting complementary forms of parallelism: Map/Reduce and Task-based.

The **Map/Reduce (M/R) programming model** is a common model for distributing the same computation into parallel units for operating over very large datasets. Originally developed by Google, it has numerous implementations, including Apache Spark [4], RISELAB's PyWren [5] and others. Generally, it consists of two categories of operations: (1) *map* operations, which run the same computation on each data record/chunk, and (2) *reduce* operations, which combine all the records/chunks, or subsets thereof, into a single result, or a new dataset, respectively. Many common tasks can be carried out with M/R operations. The M/R programming model only exploits *structured parallelism*. The ELASTIC software components that will support this programming model are Spark and COMPSs, both described in subsections 2.3.2 and 0 respectively.

The **Task-based programming model** is used for distributing independent, asynchronous and parallel units of computation for operating over very large datasets. The model may support the definition of fine-grain synchronization mechanisms among tasks by means of data dependencies: a task with an *input* dependency on a data element cannot start its execution until any previous task with an *output* dependency over the same data element completes. The task-based model can exploit both, *structured* and *unstructured* parallelism. As result, the task-based model allows supporting the M/R model with two sequential tasks to implement the *map* and *reduce* primitives. COMPSs [6] and RISELAB's Ray [7] are two distributed programming frameworks that support the task-based model. The ELASTIC software component that will support this programming model is COMPSs, described in subsection 0.

### 2.3.2 Spark

Apache Spark [4] is an open-source project for fast data processing and cluster computing. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Spark facilitates the implementation of both iterative algorithms, which visit their data set multiple times in a loop, and interactive/exploratory data analysis, i.e., the repeated database-style querying of data. Apache Spark uses memory and can use a disk for processing. Spark is able to execute batch-processing jobs between 10 to 100 times faster than the Hadoop-based M/R framework

Every Spark application consists of a driver program that runs the user's main function and executes various parallel operations on a cluster. The main abstraction Spark provides is a resilient distributed dataset (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. A second abstraction in Spark is shared variables that can be used in parallel operations. By default, when Spark runs a function in parallel as a set of tasks on different nodes, it ships a copy of each variable used in the function to each task.

*Table 2. Sample methods from Spark*

| Description | API |
|---|---|
| Chaining the resilient distributed dataset (RDD) transformation operations to realise the MapReduce functionality | `JavaPairRDD<String, Integer> counts = lines.flatMap(...).map(...). reduceByKey(...);` |

The parallel processing enabled by Spark is supported by the COMPS distributed framework used in the ELASTIC platform implementation.

### 2.3.3 COMPSs

COMPSs [6] is a distributed development framework developed at BSC and used in HPC environments to distribute workloads transparently across multiple computing nodes with regards to the underlying infrastructure. In cloud and big-data environments, COMPSs provides scalability and elasticity features allowing the dynamic provision of resources.

COMPSs supports the two programming models presented in Section 2.3.1 into a unified development environment. The M/R syntax used is compatible with the one provided by Apache Spark (see Table 2). The conceptual description of the main task-based API primitive is summarized in Table 3. The M/R and task-based API primitives will be then transformed to a set of API calls to the COMPSs run-time that will orchestrate the distribution computation across the compute continuum, while fulfilling the NFR, transparently to the programmer.

*Table 3. Conceptual description of the task-based API provided by COMPSs.*

| Programming model | API Primitives |
|---|---|
| Task-based | `task [(<dataset>=[IN\|OUT\|INOUT] return=<dataset>)]` |

### 2.3.4 Flink

**Apache Flink** [8] is an open-source and distributed stream processing framework that provides support for real-time complex event processing. Flink offers tremendous capabilities to run real-time data processing pipelines in a fault-tolerant way at a scale of millions of events per second. Minimising resource use resources at single millisecond latencies is also considered a strength of Flink. These features make Flink a suitable candidate for use in the distribute data analytic platform – specially based on the emphasis on the non-functional requirements in the ELASTIC ecosystem. Moreover, Flink has been designed to perform computations at in-memory speed and run in all common cluster environments, which make it use-able in the task-based programming model in the ELASTIC ecosystem.

*Table 4. Some of the key methods from Flink.*

| Programming model | API Primitives |
|---|---|
| Setting up a sample data stream (from a file) | `final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();`<br>`DataStream<String> text = env.readTextFile("file:///path/to/file");` |
| Setting up a transformation function on a data stream | `DataStream<String> input = ...;`<br>`DataStream<Integer> parsed = input.map (new MapFunction<String, Integer>() {`<br>`  @Override`<br>`  public Integer map(String value) {`<br>`  return Integer.parseInt(value);`<br>`  }`<br>`});` |

### 2.3.5 Kafka

Apache Kafka [9] is an open-source stream-processing software platform, project aims to provide a unified, high-throughput, low-latency platform for handling real-

time data feeds. Its storage layer is essentially a "massively scalable pub/sub message queue designed as a distributed transaction log, making it highly suitable for processing streaming data from heterogeneous sources in the ELASTIC ecosystem.

Kafka stores key-value messages that come from arbitrarily many processes called producers. The data can be partitioned into different "partitions" within different "topics". Within a partition, messages are strictly ordered by their offsets (the position of a message within a partition), and indexed and stored together with a timestamp. Other processes called "consumers" can read messages from partitions. For stream processing, Kafka offers the Streams API that allows writing Java applications that consume data from Kafka and write results back to Kafka. In the ELASTIC Distributed Analytic platform, Apache Kafka will work in conjunction with other stream processing systems such as Apache Flink and Apache Spark.

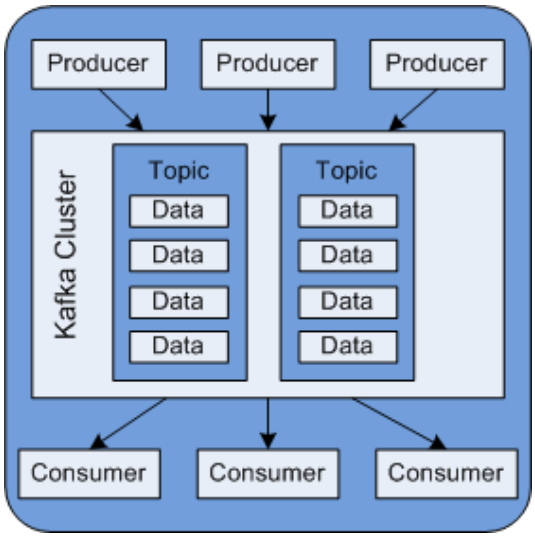


*Table 5. Some of the key methods from Kafka*

| Description | API |
|---|---|
| Create a topic named "test" | `kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic test` |
| Show the list of topics | `kafka-topics.sh --list --bootstrap-server localhost:9092` |
| Post messages on the "test" topic | `kafka-console-producer.sh --broker-list localhost:9092 --topic test` |
| Consume messages | `kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning` |

## 2.4 Orchestrator Platform

Within ELASTIC, the COMPSs run-time will implement one of the key innovations of ELASTIC, i.e., a new elasticity concept to distribute the computation among the compute continuum while fulfilling the NFR imposed by the system.

COMPSs supports two interfaces to communicate with the different cloud technologies for resource and container management. The *resource management interface* provides methods for creating, consulting and destroying resources. The specific methods are represented as a generic Cloud Connector and implemented with the functionalities presented in Table 6. The abstract methods are further implemented in the specific subclasses that implement four different connectors: (1) rOCCI (ruby Open Cloud Computing Interface) [10], (2) JClouds [11], (3) Docker [12], and (4) Mesos [13].

*Table 6. Generic Cloud Connector of COMPSs runtime.*

| Description | API |
|---|---|
| Create a virtual machine (VM) with the given description and properties | `public abstract Object` **`create`**`(HardwareDescription hd, SoftwareDescription sd, Map<String, String> prop) throws ConnException;` |
| Wait until the VM with id @id is created | `public abstract VirtualResource` **`waitUntilCreation`**`(Object id) throws ConnException;` |
| Destroy the VM with id @id | `public abstract void` **`destroy`**`(Object id);` |
| Return the time slot | `public long` **`getTimeSlot`**`()` |
| Return the price per time slot | `public abstract float` **`getPriceSlot`**`(VirtualResource virtualResource);` |
| Close the connector interface | `public abstract void` **`close`**`();` |

The *container management interface*, created within the Phase 1 of the project to better support the interaction with the rest of software components, allows to distribute the computation on containers deployed across the continuum. The methods to handle the deployment process are represented as an abstract Container Starter, with the methods listed in Table 7.

*Table 7: Abstract Container Starter of COMPSs Runtime*

| Description | API |
|---|---|
| Runs the necessary process to launch the container in a node and returns the container id. | `protected abstract String` **`distribute`**`(String master, Integer port) throws InitNodeException;` |
| Stops the remote container | `protected abstract void` **`stopContainer`**`(String user, String name, String containerId);` |
| Removes the container from the remote host | `Protected abstract void` **`deleteContainer`**`(String user, String name, String containerId);` |

Moreover, we have extended the definition of the available resources, specified in COMPSs through the resources and project XML files (see Section 5 for further details). Concretely, we have included the **`ContainerNode`** field to enable container deployment, following the formatting as shown below.

```
<ContainerNode Name="localhost">
    <Engine>docker</Engine>
    <ImageName>compss/compss-app:1.0</ImageName>
</ContainerNode>
```

Finally, COMPSs supports the distribution of parallel programming models included in the SDKs provided by the parallel embedded architectures of the edge components to further exploit the performance capabilities these architectures. Concretely, it supports OpenMP OpenMP [14], OmpSs [15] and CUDA [16].

Section 5 provides a detailed description of COMPSs.

## 2.5 NFR Tool

The ELASTIC architecture includes the Non-Functional Requirements (NFR) tool, which will (statically) analyse the internal structure of application implementing the data analytics and (dynamically) monitor the execution of the system using the hybrid fog computing platform capabilities.

This tool will therefore execute in two different contexts:

1. As a stand-alone tool, to be used before deployment to determine the system configuration (allocation of components to nodes) which enables to fulfil the non-functional requirements of the applications; and
2. As a runtime component which will analyse the monitoring data related to the services' execution, detecting actual execution changes that violate the service level agreements, triggering changes by the orchestration layer.

Detailed information about the requirements and functionalities of the NFR Tool can be found in deliverable D4.2 [17].

### 2.5.1 Offline Tool

Table 8 presents the interface of the NFR Offline tool. The tool can be used to analyse a single property in isolation, using the optional --NFR parameter. The other inputs of the tool are the files with the platform description, the application services and the QoS parameters.

*Table 8. NFR Offline tool interface*

| Description | API |
|---|---|
| Analyze a single property | analyze [--NFR XXX] --platform-description platform.xml --application-services services.xml --QoS-parameters qos.xml --output system_configuration.xml |

### 2.5.2 Online Monitor

During execution, the NFR tool provides two types of objects. The first, is the SLA Manager which enables a single property to be analysed in isolation. Table 9 presents the generic interface of the SLA Manager, where Platform_Information denotes the configuration of the ELASTIC nodes, Services the currently executing services, QoS_Parameters defines the services' desired level of service, and Runtime_Information indicates the monitored information acquired during runtime. This analysis will output a possible new system configuration, describing the set of required changes to the current system configuration or an empty set if the system is currently satisfying with the required QoS levels. A simple feasibility check is also available.

*Table 9. Isolated NFR analysis API*

| Description | API |
|---|---|
| Create a System Configuration | SystemConfiguration system = createConfiguration(Platform_Information, Services,QoS_Parameters) |
| Isolated NFR property feasibility check, when a change is detected in a service execution | Boolean feasible = checkFeasibility(Runtime_Information) |
| Isolated NFR property analysis, returning the (possible empty) set of required changes to the current system configuration | SystemConfiguration changes = isolatedAnalysis(Runtime_Information) |
| Configuration Change Information | SystemConfiguration system = changeConfiguration(Service, QoS_Parameters) |

The second provides a holistic NFR analysis, where individual SLA Managers are iteratively used by the Global Manager, considering the potential trade-offs between performance, predictability, energy-efficiency, communication quality and security. The API of the Global Manager is described in Table 10, being similar to the one in Table 9.

*Table 10. Holistic NFR analysis API*

| Description | API |
|---|---|
| Create a System Configuration | SystemConfiguration system = createConfiguration(Platform_Information, Services,QoS_Parameters) |
| Holistic NFR property feasibility check, when a change is detected in a service execution | Boolean feasible = checkFeasibility(Runtime_Information) |
| Holistic NFR property analysis, returning the (possible empty) set of required changes to the current system configuration | SystemConfiguration changes = holisticAnalysis(Runtime_Information) |
| Configuration Change Information | SystemConfiguration system = changeConfiguration(Service, QoS_Parameters) |

A note that the SLA Managers when used in the holistic phase, will implement analysis which already take into consideration the multiple non-functional properties.

## 2.6 Hybrid Fog Computing Architecture

The ELASTIC project proposes a hybrid fog-computing architecture based on standard open-source reusable components. The dependences between the components running on the fog is reduced to a minimum in order to allow the hot-plug and dynamic reconfiguration of the services offered by the platform. The hybrid architecture allows the use of dynamic applications in form of microservices (containers) or native applications (monolithic).

Detailed information about the software component included in the fog computing architecture can be found in deliverable D5.1 [3].

### 2.6.1 Nuvla and NuvlaBox

Nuvla is a secured edge-to-cloud management platform, enabling near data processing, targeting a number of industries, including AI, smart city, big data and Industry 4.0. In ELASTIC, Nuvla will provide the edge control and application provisioning capabilities for both the cloud and edge infrastructures. Nuvla also brings a 'data ledger' service component which allows the user to register, search and find datasets from a wide range of infrastructures, across the cloud and edge continuum. This can then be used to build higher level placement heuristics.

Nuvla is open source under Apache 2.0 and it is also available for on-premise installations.

Alongside Nuvla, there is the NuvlaBox, an open source software distribution which turns x86 and ARM hardware platforms into edge devices. The NuvlaBox can be controlled remotely from Nuvla, providing a platform to deploy and monitor applications packaged as containers on the edge and in the cloud. The NuvlaBox also comes with clustering capabilities, which are highly appreciated by the ELASTIC use cases, as the need for additional computation power varies over time.

Throughout the project, Nuvla and NuvlaBox will also be extended to provide self-contained management capabilities - an edge-to-edge solution - which will allow for a faster, more secure and cloud-independent way to manage and provision workloads at the edge.

#### 1.1.1.1 Nuvla API

Nuvla exposes a rich API heavily inspired by the DMTF CIMI cloud industry standard[1]. All infrastructure resources are modelled and represented as JSON documents which are uniquely identified by URIs. Each resource representation has a globally unique ID attribute of type URI which acts as a reference to itself.

The API is self-discoverable and follows a RESTful HTTP-based protocol, providing language-agnostic integration with other software components. The standard GET, POST, PUT and DELETE HTTP methods are available, covering all basic Search (or Query) and CRUD (Create, Read, Update, and Delete) operations, plus the possibility to add custom resource operations which are mapped into POST requests.

The API server also includes additional intelligence to support authentication-like "user" and "session" resources, which are used when providing access control lists, for a fine-grained authorization model. At the moment, the API supports a wide set

[1] https://www.dmtf.org/sites/default/files/standards/documents/DSP0263_2.0.0.pdf

of authentication mechanisms, from internal user/password and API key/secret, to social and even federated authentication (via GitHub, eduGAIN, etc.).

The interface specification provides advanced features for manipulating results when searching resource collections. All the resource selection parameters are specified as HTTP query parameters. These are specified directly within the URL when using the HTTP GET method.

Table 11 list the management API to be used by adjoining ELASTIC components in order to achieve a successful and secure software integration.

*Table 11. API syntax exposed by Nuvla.*

| **Method** | **Path** (included in /api/) | **Description** |
|---|---|---|
| **get** | *cloud-entry-point* | Self-discovery entry point, with a list of all the resources available in the system |
| | *<resource-name>* | Get the collection "resource-name" with a list of all individual resources within |
| | *<resource-name>/<uuid>* | Get a specific resource, identified by a UUID |
| | *<resource-name>?$filter=<expression>* | Filter a collection with a mathematical "expression" compliant with the EBNF grammar defined in the CIMI specification |
| | *<resource-name>?$orderby=<attr>:[asc\|desc]* | Sort a collection by an "attribute" |
| **post** | *<resource-name>* | Passed together with a payload, it creates a new resource |
| | *<resource-name>/<action>* | Triggers a custom action for that resource. It also accepts payloads, depending of the action specification |
| **put** | *<resource-name>/<uuid>* | Edits the resource with the UUID "uuid" and updates its attributes based on the request's payload |
| **delete** | *<resource-name>/<uuid>* | Deletes the resource with the specified UUID |

### 2.6.2 KonnektBox

The KonnektBox is the fog-computing platform of the IKERLAN KONNEKT® architecture, which is the in-house developed solution for industrial digitalisation of products and services, composed by KonnektCloud and KonnektBox itself. The KonnektBox uses a proprietary protocol (via MQTT) to manage its workloads and services. If the solution uses a VPN, Docker Swarm or Kubernetes can be implemented and an administration port can be opened in order to perform services administration. Any IT infrastructures which provide the technologies mentioned in D5.1 [3] can be associated with Nuvla, to allow the management of workloads into non-NuvlaBox devices. For example, any Docker Swarm infrastructure can be added to Nuvla and have its microservices managed from there. So, Nuvla could see KonnektBox as a secure EndPoint (using Docker Swarm).

*Table 12. Microservice manager's Python API description[2].*

| Description | API |
|---|---|
| Run a Swarm container on Docker Engine | `def swarm(image, command=None, **kwargs)` |
| Create a discovery token | `def create(image, **kwargs)` |
| List the nodes in a Docker cluster | `def list(**kwargs)` |
| Create a Swarm manager | `def manage(image, repository, tag=None, **kwargs)` |
| Create a Swarm node | `def join(image, repository=None, tag=None, **kwargs)` |

The docker node CLI utility allows users to run various commands to manage nodes in a swarm, for example, listing the nodes in a swarm, updating nodes, and removing nodes from the swarm [50].

#### 2.6.2.1 KonnektBox microservice manager

KonnektBox microservice manager provides a way to deal with containers using OS-level virtualisation, which eases the deployment of programs and the reconfiguration and dynamic fog-cloud service execution required by the project, whilst ensuring a lightweight virtualization mechanism. The manager must provide an API to deal with the creation, run, stop, deletion etc. of the containers. Table 4 presents the basic API. Docker is a well-known known alternative and it offers an API that can be called through command line, HTTP, Python or GO.

*Table 13. Microservice manager's Python API description[3].*

| Description | API |
|---|---|
| Runs a container | `def run(image, command=None, **kwargs)` |
| Stops a container | `def stop(image, **kwargs)` |
| Prints the logs of a container | `def logs(image, **kwargs)` |
| Lists all available containers | `def list(**kwargs)` |
| Pulls a container | `def pull(image, repository, tag=None, **kwargs)` |
| Commits a container | `def commit(image, repository=None, tag=None, **kwargs)` |

#### 2.6.2.1 KonnektBox monolitic service manager

In the KonnektBox monolithic service manager operating system calls are going to be used to execute any task type, directly from the application or via the microservice manager. Those OS system calls are: 1) process management (fork, exit, wait), 2) memory management, 3) file operations (file open, read, write, close), 4) input/output device management, including communications. Those system calls are

[2] Based on the Docker swarm API: https://docs.docker.com/swarm/reference/
[3] Based on the Docker API (https://docker-py.readthedocs.io/en/stable/containers.html)

to be used via the compiler (C, C++), via interpreter (Python) or via the Java virtual machine (Java).

In ELASTIC we will provide a higher level API, to be used by the orchestrator for the task management, presented in *Table 14*.

*Table 14. Monolitic service manager's Python API description.*

| Description | API |
|---|---|
| Runs a native application | `def run(app, command=None, **kwargs)` |
| Stops a native application | `def stop(app, **kwargs)` |
| Prints the logs of a native application | `def logs(app, **kwargs)` |
| Lists all available native applications | `def list(**kwargs)` |
| Pulls a native application | `def pull(app, repository, tag=None, **kwargs)` |

### 2.6.3 Monitor manager

The monitor manager will be the responsible for the collection of metrics of the current computing node (as for example the CPU, GPU, memory usage or the functioning temperature, humidity or connectivity). Additionally, the monitor manager will expose the characteristics of the computing node (e.g. hardware type, minimum latency, security level…). Those metrics and characteristics are required by the Orchestrator to select the appropriate computing node to execute a task. Therefore, there will be a daemon running in every computing node and it will expose an API to collect the data to the orchestrator.

Nowadays, there are simple alternatives like CAdvisor which analyses resource usage and performance characteristics of running containers, or more complex and complete like Prometheus which provides full metric analytics.

Certain parts of the KonnektBox monitoring daemon will be used in this component.

*Table 15. Microservice manager's Python API description[4].*

| Description | API |
|---|---|
| Gets the usage from a time interval of different measurements, such as CPU, GPU, memory | `def get_statistics(time_interval, measurements, **kwargs)` |
| Gets hardware resources | `def get_hw(characteristics=All, **kwargs)` |
| Gets latency, security level… and other NFR requirements values | `def get_features(characteristics=All, **kwargs)` |

[4] Based on the Docker API (https://docker-py.readthedocs.io/en/stable/containers.html)

### 2.6.4 Data router

Data router middleware will be responsible for the data communication and storage, created by the sensors, which is going to be later processed by the distributed data analytics platform. This communication can occur through the traditional send and receive interfaces, or, alternatively, publish-subscribe pattern can be used. An example of publish-subscribe pattern is the MQTT standard protocol. The implementation of this approach requires a message broker, accessible from both the publisher and the subscriber.

Some developments of KonnektBox will be integrated in ELASTIC Data Router.This middleware should be compatible with dataClay.

*Table 16. Send/receive tentative Python API description.*

| Description | API |
|---|---|
| Sends a message to a receiver | `def send(message, receiver)` |
| Receives a message from a sender | `def receive(message, sender)` |
| Sends a message to all nodes | `def broadcast(message)` |
| Sends a message to some receivers | `def multicast(message, receiver)` |

*Table 17. Publish-subscribe tentative Python API description.*

| Description | API |
|---|---|
| Publishes a message to a topic | `def publish(topic, message)` |
| Subscribes to a topic | `def subscribe(topic)` |
| Waits and receives messages from a topic that it is previously subscribed | `def poll(timeout)` |

### 2.6.5 dataClay

dataClay is a distributed storage platform developed at BSC and used in HPC environments to maintain user-defined data consistency and visibility. dataClay enables different computing resources located at different levels of the compute continuum, from edge to cloud, to define the visibility of their data. To do so, it enforces visibility scopes for each piece of data (at any granularity), ranging from local, visible to a subset of consumers, or global. This tuneable visibility is key to share data while at the same time make the system secure and scalable. dataClay will enable to guarantee the data consistency among the different computing resources (as defined by the user through data scoping), liberating the programmer from the burden of implementing explicit data transfers across the compute continuum.

The dataClay interface allows programmers to store objects using the same model that they use in the application, thus avoiding time consuming transformations between the persistent and non-persistent models. In addition, dataClay simplifies and optimizes the idea of moving computation close to the data by enabling the execution of methods in the same node where a given object is located. Furthermore, dataClay code intrusion is considered minimum, since it does not require adding a vast number of code lines. Table 18 presents the Python API of

dataClay. An equivalent API using the CamelCase naming convention is also offered for Java source codes.

*Table 18. DataClay Python API description.*

| Description | API |
|---|---|
| Stores an object in dataClay and assigns an OID to it.<br>An optional alias may be provided to identify the object (additionally to its OID). | `def make_persistent (self, alias=None, backend_id=None, recursive=True)` |
| Removes the alias linked to an object. If this object is not referenced starting from a root object, the garbage collector will remove it from the system. | `def delete_alias (cls, alias)` |
| Federates the current object with external dataClay. | `def federate (self, dataclay_name=None, recursive=True)` |

Deliverable D5.1 [3] provides a detailed description of dataClay.

# 3. Requirements of the ELASTIC Software Architecture

## 3.1 Market Analysis: The Strategic Research and Innovation Agenda (SRIA)

The SRIA[5], defined by the Big Data Value Association (BDVA)[6], collects the needs and concerns of the big data stakeholder ecosystem as a result of several workshop, conferences, surveys, etc. Below, we list the main challenges defined in the SRIA:

- **Privacy and security guarantees**: Data from different sources and locations make difficult tracking of security and privacy of data. This is one of the main concerns of the stakeholder ecosystem.
- **Cost management**: Companies need to foresee the cost of Big Data projects. This is a big challenge due to quick scalability and vast amount of data to process. Providers are supplying new monetization models to offer cost-effective alternatives to customers.
- **Scalability and performance**: Coping with the process, storage and analysis of vast amount of data collected from geographically distributed data sources make necessary new approach of software tools to face this extreme-scale analytics challenge.
- **Integration with existing models**: Companies are concerned about lack of operability of the new Big Data solutions with existing systems, what can lead to additional costs, lack of skills, increase of time to market, etc. In this sense, is important to ensure interoperability by using de-facto market standards.
- **New and varied big data capabilities**: Each sector has different business objectives so they will require different analytic needs. This includes the

[5] http://bdva.eu/sites/default/files/EuropeanBigDataValuePartnership_SRIA__v3_0.pdf#overlay-context=downloads%3Fq%3Ddownloads

[6] The BDVA is an industry-led organisation representing European large and SME industry and research organizations.

fulfilment Big data analytics solutions providers need to address these customers' requirements in all domains, being real-time, energy efficiency and security the most important ones to be fulfilled in the smart city domain. These requirements

- **Lack of big data skills:** Shorten the time-to-market is a great challenge for Solution providers, so new Big Data tools have to be highly programmable.

*Figure 2* shows the occurrence of each of the needs and concerns of the big data stakeholder ecosystem presented above.

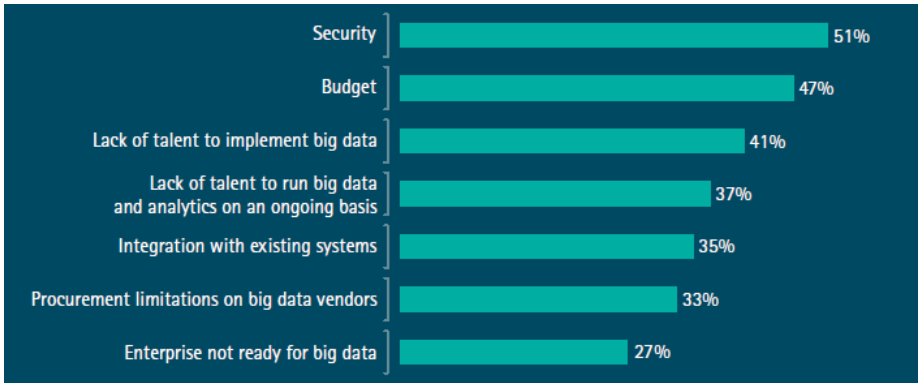


*Figure 2. Main challenges to implement Big Data in the companies*[7]

## 3.2 ELASTIC Software Architecture Business Goals (BG)

Based on the previous market analysis conducted by the BDVA, we have identified six BGs that the ELASTIC software architecture has to incorporate.

| ID | BG1 |
|---|---|
| Name | **Interoperability** |
| Type | Business Goal |
| Descrip-tion | ELASTIC will ensure integration and interoperability by incorporating de-facto market standards software components. This is a key feature for a wider uptake of the ELASTIC solution and for exploitation purposes. Main interests will be those applied in fog computing, smart cities and railway domains, and possible contributions to standardization are also addressed. |
| Rationale | Ensure integration and interoperability of ELASTIC software components with existing solutions. |
| Involved Stakehol. | IoT/Cloud/Edge Providers, Big data analytics solutions providers. |

| ID | BG2 |
|---|---|
| Name | **Ease of use** |
| Type | Business Goal |
| Descrip-tion | One of the main goals of the ELASTIC solution is to help programmers in the development of big data applications in real-time scenarios. To do this, ELASTIC will reduce the time of manually distributing the computation along the continuum, by means of a software architecture able to do this task in an efficient automatic way. |

[7] https://es.scribd.com/document/308550795/Accenture-Big-Data-POV

| | |
|---|---|
| **Rationale** | Reduce time to market of deploying and executing new extreme-scale big data services and additional costs of training. |
| **Involved Stakehol.** | Big data analytics solutions providers, Innovative SMEs of big data solutions, Edge providers, SMEs in all sectors, IoT solution providers. |

| **ID** | **BG3** |
|---|---|
| **Name** | **Scalability and high performance** |
| **Type** | Business Goal |
| **Descrip-tion** | ELASTIC aims to develop a novel software development ecosystem to efficiently distribute computation along the compute continuum. To ensure high performance, ELASTIC will adopt architectures from HPC, and parallel embedded computing domains. |
| **Rationale** | The elaboration and knowledge extraction (including data-in-motion and data-at-rest analytics) of data coming from geographically distributed sources. |
| **Involved Stakehol.** | Big data analytics solutions providers, Innovative SMEs of big data solutions, SMEs in all sectors. |

| **ID** | **BG4** |
|---|---|
| **Name** | **Non-functional Requirements** |
| **Type** | Business Goal |
| **Descrip-tion** | The ELASTIC software architecture will provide guarantees about the non-functional requirements inherited from the cyber-physical interaction of the smart city domain. The "level of guarantee" will depend on the "criticality-level" of the service implemented and the computing devices upon which these services will be deployed. In this sense, edge-based devices will provide stronger guarantees than cloud ones. |
| **Rationale** | Enable the use of ELASTIC technology in environments with non-functional requirements, e.g. railway. |
| **Involved Stakehol.** | Big data analytics solutions providers, Innovative SMEs of big data solutions, SMEs in all sectors, Academics and Scientists. |

| **ID** | **BG5** |
|---|---|
| **Name** | **IT infrastructure cost reduction** |
| **Type** | Business Goal |
| **Descrip-tion** | The ELASTIC software architecture has the objective of enhancing the level of data latency and throughput as those obtained using a cloud-based infrastructure but at lower cost. |
| **Rationale** | Make workload and data process distribution between Cloud and Edge at lower cost while maintaining or enhancing the productivity, compared to cloud-only solutions. |
| **Involved Stakehol.** | Big data analytics solutions providers, Innovative SMEs of big data solutions, SMEs in all sectors, Academics and Scientists. |

| **ID** | **BG6** |
|---|---|
| **Name** | **Privacy and Security** |
| **Type** | Business Goal |

| | |
|---|---|
| **Description** | Regarding privacy, all data collected by the IoT sensors connected to the ELASTIC software architecture will be anonymized before being processed and stored. This anonymization will fulfil current Italian regulations regarding privacy and security of citizens. These regulations require to guarantee that there will be no means to: (1) recognize the specific pedestrian/vehicle and (2) trace the path of the anonymize object.<br>Regarding security, the ELASTIC software architecture will incorporate all mechanisms to guarantee the correct operation of the smart city and the connected tramways, including active thread avoidance mechanisms. |
| **Rationale** | Ensure privacy and security of business and personal data collected and processed by the ELASTIC software architecture. |
| **Involved Stakehol.** | All stakeholders. |

## 3.3 Technical Requirements (TR) of the ELASTIC Software Development Ecosystem

The six BG identified in the previous section results in four technical requirements described below.

| **ID** | **REQ-SWARCH-TR1** |
|---|---|
| **Name** | **Increase Software Productivity** |
| **Type** | Technical Requirement |
| **Description** | ELASTIC aims to promote productivity regarding the development and execution of extreme-scale data analytics services, in terms of programmability, portability and performance. With such purpose ELASTIC promotes the serverless concept that provides an execution model where the management of resources is completely hidden from the application developer. |
| **Rationale** | **Programmability.** The developer is responsible for: (1) defining the functionality of the extreme-scale data analytics service as units of code exposed to the outside world, and (2) defining the parallelism exposed by the service, by means of programming models, i.e., task-based and M/R. This will allow the software architecture to efficiently manage the underlying computing resources, and to hide the complexity of the compute continuum to the programmer. This possibility boosts the performance of the whole system, possibly slightly degraded by the use of multiple abstraction layers due to its greater response latency compared to systems continuously running on a dedicated server. This is a further step towards the definition of what the application does and not how the application does.<br>**Portability.** The use of the serverless execution model and well-known programming models to express the data-analytics functionality, i.e., task-based, M/R, streaming, and the parallelism exposed by the application enables to execute the same application in multiple platforms without any performance loss. The underlying run-time systems included in the ELASTIC software development ecosystem are responsible for dealing with the internals of each specific technology with the objective of maximizing the performance capabilities, including edge and cloud. |

| | |
|---|---|
| | **Performance.** The ELASTIC software architecture will be responsible for distributing the data analytics execution across the compute continuum. The ELASTIC software architecture will include the proper data analytics and programming models to exploit the capabilities of the architectures where the final functionalities will ride on. A number of parallel programming models will be supported in the ELASTIC software architecture as well, i.e., OpenMP, OpenACC or CUDA, which can exploit the benefits of each final architecture (multi-core, NVIDIA GPU, GPGPU, etc.) in a customized manner as evidence that performance is a very important aspect within the ELASTIC architecture. |

| **ID** | **REQ-SWARCH-TR2** |
|---|---|
| **Name** | **Fulfilment of Non-Functional Requirements** |
| **Type** | Technical Requirement |
| **Descrip-tion** | The ELASTIC software architecture aims to support the non-functional requirements usually found in cyber-physical systems, such as real-time processing, energy-efficiency, communication quality of service, security and safety. |
| **Rationale** | In cyber-physical systems, the interaction between the computing system and its environment needs to cope with the non-functional requirements inherited from the application domain. Control loops require guaranteed response times, sensors and embedded computers require energy efficiency. At the same time, while processing time and energetic cost of computation is reduced as data analytics is moved to the cloud, the end-to-end communication delay and the performance of the system (in terms of latency) increases. Moreover, as computation is moved to the cloud, the required level of security increases to minimise potential attacks, which may end up affecting the safety assurance levels, hindering the execution and data exchange among edge and cloud resources.<br>The ELASTIC project will include these non-functional properties as first class entities in the software architecture. Developers will be able to specify quality-of-service requirements, which can span from critical guaranteed requirements to best-effort approaches, depending on the criticality of the application function. The component deployment will be either statically or dynamically performed, to be able to provide the required quality-of-service. Critical guarantees will be provided through static deployment of resources, whilst softer requirements will be coped with a mix of static and dynamic adaptation approaches. |

| **ID** | **REQ-SWARCH-TR3** |
|---|---|
| **Name** | **Enable Flexibility and Elasticity** |
| **Type** | Technical Requirement |
| **Descrip-tion** | The ELASTIC project aims to develop a novel *elasticity concept,* defined as the ability to exploit the underlying infrastructure to automatically adapt the computing capabilities provided by the pool of resources to the current workload, while guaranteeing the non-functional requirements. |
| **Rationale** | ELASTIC will exploit elasticity at different computing levels: (a) *horizontal elasticity* across the same computing level, i.e., at edge or at cloud, allowing to dynamically incorporate new compute resources |

| | |
|---|---|
| | to the analytics workloads; and (b) *vertical elasticity* at different computing levels, i.e. across edge and cloud, allowing to resize existing computing capabilities by accessing higher computing levels. In both cases, the ELASTIC software architecture will guarantee the fulfilment of the non-functional requirements. |

| **ID** | **REQ-SWARCH-TR4** |
|---|---|
| **Name** | **Privacy and Security Mechanisms to Guarantee the Legal Framework** |
| **Type** | Technical Requirement |
| **Descrip-tion** | ELASTIC software architecture will incorporate the mechanisms needed to fulfil the Italian and European regulations[8] regarding privacy in the collection of data from Florence citizens, and regarding security with respect to the potential threads that can affect the integrity of the extreme scale data analytics services. |
| **Rationale** | Regarding privacy ELASTIC will incorporate mechanisms at the edge side to anonymise the information collected from the sensors installed in the municipality of Florence and tramway vehicle sensors.<br>Regarding security, it will include the required isolation mechanisms on the network-side to avoid outside attackers or unauthorized users. |

Each TR addresses a BG as defined in Table 19.

*Table 19. Relation between business goals and technical goals of the ELASTIC project.*

| Business Goals | Technical Goals |
|---|---|
| BG1. Interoperability | REQ-SWARCH-TR1. Increase Software Productivity |
| BG2. Easy-to-use | |
| BG3. Scalability and Performance | |
| BG4. Real-time Requirements | REQ-SWARCH-TR2. Fulfilment of Non-Functional Requirements |
| BG5. IT Infrastructure Cost Reduction | REQ-SWARCH-TR3. Enable Flexibility and Elasticity |
| BG6. Privacy and Security | REQ-SWARCH-TR4. Privacy and Security Mechanisms to Guarantee the Legal Framework<br>REQ-SWARCH-TR2. Fulfilment of Non-Functional Requirements |

# 4. Software Development and Integration Plan

The ELASTIC project involves a distributed team of several people from different institutions and areas of expertise. This section defines the development and

[8] REGULATION (EU) 2016/679 OF THE EUROPEAN PARLAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural person; NATIONAL REGULATION (IT) "CODE FOR THE PROTECTION OF PERSONAL DATA" - DLGS 196/2003, “Smart Road Decree”, Italian Ministry of Transportation – 28/02/2018 - that refers to national regulation, and legislative decree shortly published containing provisions for the adaptation of national legislation to the Regulation (EU) 2016/679 - GDPR.

integration processes to be followed during the execution of the project, inspired in the Scrum methodology [18].

Scrum is an iterative and incremental framework for managing product development. It defines a flexible, holistic product development strategy where a development team works as a unit to reach a common goal, and enables teams to self-organize by encouraging close collaboration of all team members. The Scrum process is divided in Sprints. A sprint is a timeboxed effort restricted to a specific duration. Each sprint starts with a planning event that identifies the work to be done and makes an estimated forecast for the sprint goal. Each sprint ends with a sprint review and sprint retrospective to identify lessons and improvements for the next sprints.

The remainder of this section is organized as follows: Section 4.1 introduces the development and integration, as well as quality assurance processes for the ELASTIC project; Section 4.2 defines the infrastructure to be used in the project to enable all teams to share and coordinate information, and Section 4.3 describes the standards and guidelines to facilitate the usage of the infrastructure.

## 4.1 Processes

This section introduces two main processes: (1) the development and integration process, and (2) the quality assurance process. The former focuses on providing means for a continuous development approach that reduces risks and facilitates the building and releasing procedures, and the latter focuses on guaranteeing high quality development results. These processes are specified in the following subsections, including the activities related to each of the processes.

### 4.1.1 Development and Integration Process

The ELASTIC project is an aggregation of different components, as defined in the ELASTIC software ecosystem, which cooperate via specified interfaces to provide different functionalities (see Section **Error! Reference source not found.**). The s oftware architecture integration process consists in the combination of all software components into one unique ecosystem, ensuring that all components work as defined in the functional requirements and, the software architecture as a whole, provided the desired functionalities.

The ELASTIC project consists of four different phases with a milestone at the end of each phase. Figure 3 shows the integration process of the different components in the ELASTIC ecosystem considering the different phases and all components. This integration follows a tree structure and is done incrementally. Each step of the integration (represented as a diamond) has a *leader* or *master* (represented by the color of the diamond, as described in the legend), which is the responsible for that particular part of the integration.

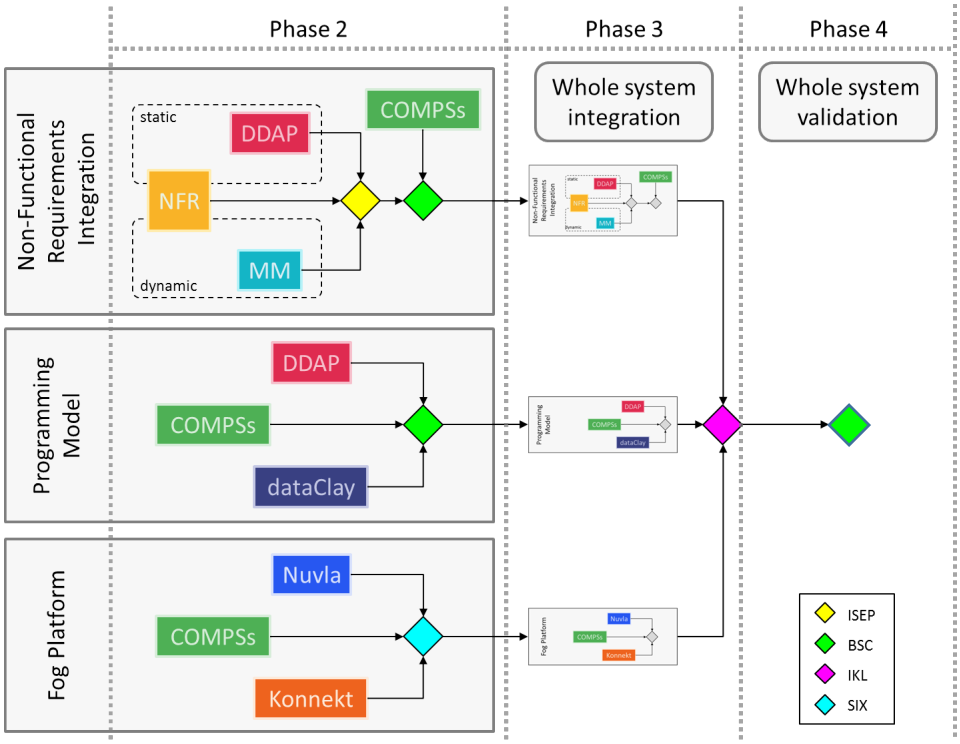


*Figure 3. Tree-like diagram of the integration of the ELASTIC ecosystem components.*

**Phase 1.** During this phase, the set of non-functional requirements has been defined to energy, security, real-time and communications instability. As a result, the API for each component has also been defined in order to allow a proper communication between with the components. This phase is currently finishing by the development of all the deliverables, including the current document, that are to be finished by Milestone 1 of the project.

**Phase 2.** During this phase, the different components will be developed based on APIs defined in phase 1 and considering each NFRs separately (i.e., the effects that one NFR may have on another is not taken into account). Furthermore, the NFR tool will be integrated with the Distributed Data Analytics Platform (DDAP) and the Monitor Manager (MM), and the COMPSs orchestrator with be integrated on one hand with the DDAP and dataClay, and on the other hand with Nuvla and Konnekt. All these integrations will also consider each NFR separately. The functionalities of each components, as well as the functionalities of the integrated components will be verified by using unit tests.

At the end of this phase (Milestone 2), each team has to accomplish the functional requirements of the corresponding component. The different teams forming the project will work following a Scrum based approach where different *sprints* of approximately one month of duration will be defined.

**Phase 3.** This phase will be implemented in two stages: the first stage will repeat the same integration as performed in phase 2, but taking into account all NFRs at the same time; the second stage will need the coordination of all teams to integrate all components taking into account all NFR at a global level. Each integration activity will have a leader (or *master* in the Scrum terminology) responsible for the integration. Furthermore, each integration will be validated by means of both unit tests and regression tests.

**Phase 4.** During the last phase of the project, all software components of the ELASTIC software ecosystem must be validated. By means of the micro-validations carried out during the integration, unexpected situations are minimized at this point. Overall, this phase will validate and verify the whole system regarding the different use-cases defined in the project as well as the synthetic benchmark described in Section **Error! Not a valid bookmark self-reference.**0. All bugs and issues reported during the previous phase will be addressed by the end of this phase, leaving the project free from known errors, hence validating the functional and non-functional requirements expected for both, the components separately, and the system as a whole.

In order to have a general overview, Figure 4 shows the Gantt chart of the full integration plan containing a high-level description of the different tasks included in the integration plan, as well as the different milestones of the project as defined in the DoA [19].

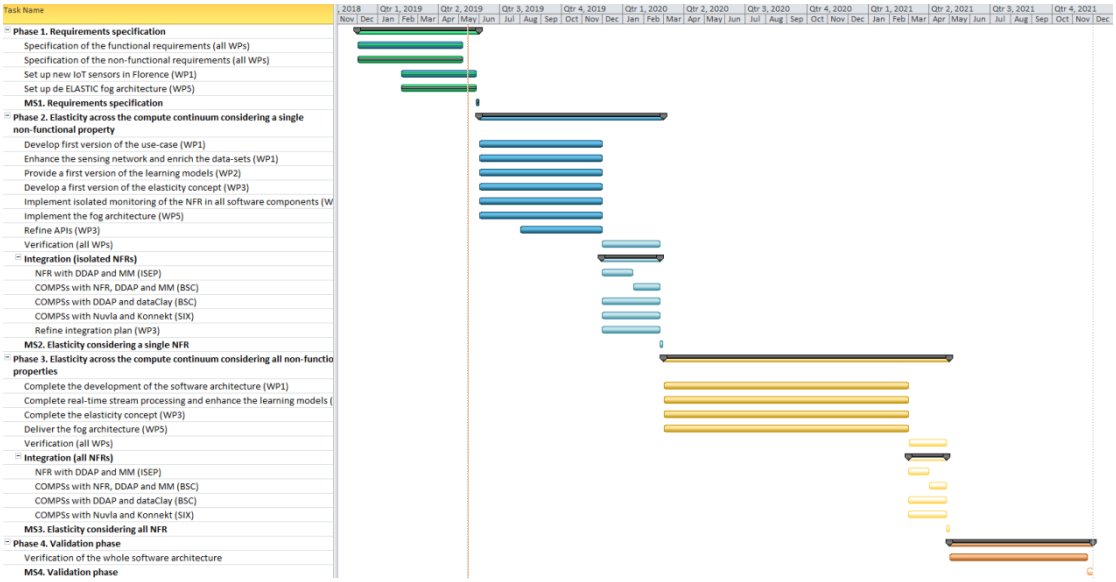


*Figure 4. Gantt chart of the development and integration plan of the ELASTIC project.*

### 4.1.2 Quality Assurance Process

The ELASTIC software developers' team is a highly distributed team of teams. This scenario requires taking special care of the communication and interconnection among the different teams. With such a purpose, several activities will be carried at, so the project provides high quality standards in terms of faults and compliance with specified behavior. These activities will occur at different stages of the development process. Some of them are continuous and some others are recurring. This section summarizes the quality assurance activities performed in the development and integration process of the ELASTIC project.

#### 4.1.2.1 Scrum-based Methodology

The Scrum methodology reduces risks and removes dependencies between releases and integration activities, allowing the synchronization of the different integration parts and the final validation. An important aspect included in the methodology is the recognition that there will be unpredictable challenges. For that reason, Scrum adopts an evidence-based empirical approach focusing on how to maximize the

team's ability to deliver quickly, to respond to emerging requirements, and to adapt to evolving technologies and changes in the project conditions.

In Phases 1 and 2, each team in ELASTIC will define its own sprints regarding contents and duration, because each team is to work in isolation. After that, in Phases 3 and 4, all teams in the project will have to define a common structure for the sprints, according the duration and contents of each sprint.

#### 4.1.2.2 Unit Testing

During Phases 1 and 2, a synthetic benchmark (see Section 0) will be developed and used together with unit tests in order to validate the functionalities implemented in each component. These unit tests will allow determine if individual units of code are correct, simplifying the later integration and facilitating future changes. Each time a new functionality is to be added to the software ecosystem, the entire unit test suite should be executed, ensuring that all new and existing tests run successfully upon code check in. To enhance the quality of the unit tests, a series of guidelines shall be used, e.g., name tests properly, keep tests small and fast, cover boundary cases, and prepare tests for code failing.

Initially, unit tests are to be tested manually by each component's developers. A system such as JUnit [20] (for Java) or GoogleTest [21] (for C/C++) can be used with such a purpose, depending on the base language used for the particular component. In a second stage, in order for the unit testing to be useful, the tests are to be performed automatically by a continuous integration system, as explained in Section 4.2.2.2.

#### 4.1.2.3 Regression Tests

During the integration, changes to the implementation may be needed. In such case, regression tests shall be run each time a modification is implemented in any part of the software ecosystem, so it can be determined whether the changes break anything that worked prior to the change. Regression tests, overall, consist on rerunning previous tests any time a modification is performed, as well as writing new tests when necessary. Adequate coverage is paramount when conducting regression tests. For that reason, a series of strategies and good practices must be followed, e.g., check possible side effects when fixing bugs, write regression tests for each bug fixed, and remove redundant tests.

#### 4.1.2.4 Bug and Issue Tracking

Bug and issue trackers allow managing lists of bugs and issues. This kind of application is used to create, update and resolve project issues, providing the platform to maintain a knowledge base that includes information that may help to resolve the issues. During the integration phase this is a key aspect to ensure the correct collaboration and communication among the different teams that participate in the integration of each particular component. In general, issue trackers have proven to be an effective lightweight task management system, much more useful than real-time messaging o emailing.

For this task, the ELASTIC project will use GitLab [22]. Further details are included in Section 4.2.3.1.

## 4.2 Infrastructure

This section describes the tools and platforms that we have identified to potentially be used within the ELASTIC project. Infrastructure decisions are based on consortium agreements and are triggered by common development standards as well as in particular the quality assurance and integration processes described in Section 0.

### 4.2.1 Development Platform

This section defines the platforms that will be used for the development of the ELASTIC project.

#### 4.2.1.1 Integrated Development Environments (IDEs)

We will select free and open source IDEs because this kind of platform provides several benefits:

- Code completion or code insight: IDEs recognize language's keywords and function names. This knowledge is typically used to highlight typographic errors, suggest a list of available functions based on the appropriate situation, or offer a function's definition from the official documentation.
- Resource management: IDEs manage resources such as libraries and header files, hence being aware of any required resource missing. By using this feature, errors can be spotted at the development stage and not later, in the compile or build stage.
- Debugging tools: IDEs allow thoroughly testing applications before release by means of assigning variable values at certain points, connecting different data repositories, or accepting different run-time parameters.
- Compile and build: IDEs allow automatic translation from high-level language to object code for languages that require a compile or build stage.

Some of the IDEs that may be used in the ELASTIC project are listed as follows:

- Eclipse [23] comprises a Rich Client Platform (RCP) for developing general purpose applications, and includes a powerful plug-in system. Components such as COMPSs will be developed using this framework. Other IDEs will be considered as well, depending on each partner requirements.
- Atom [24] is a free and open-source text and source code editor with support for plugins written in Node.js, and embedded Git Control, developed by GitHub. It provides a desktop application easy to extend, where most of the extending packages have free software licenses.

#### 4.2.1.2 Software Configuration Management

Software configuration management systems provide means for distributed teams to work collaboratively together on shared documents. Within ELASTIC, development will be carried out using two different such platforms:

- Apache Subversion (SVN) [25]. SVN is a tool for source code versioning. It comprises a centralized repository for collaboration, and also allows managing different development lines in so called branches. SVN provides online versioning of directories, files and metadata. Other functionalities offered by SVN include conflict resolution and merging. SVN will be used in ELASTIC for sharing documentation such as deliverables, technical reports, papers and posters, among others.

- Git [26]. Git is a free and open source distributed version control system able to handle very large projects with speed and efficiency, because it has a tiny footprint and includes features like cheap local branching, convenient staging areas, and multiple workflows. Git will be used in ELASTIC for sharing code source. It is particularly convenient within the ELASTIC project, where a very distributed team develops in parallel several components, because its support for submodules Git submodules allow to treat different projects as separate, yet still be able to use one from within the other.

#### 4.2.1.3 Instant Messaging and Transparency

Communication is of paramount importance in the development and integration process, particularly, when working in remote teams composed of many people. For such a reason, not only communication but also transparency is needed, because everybody must be aware of what is happening in all sides in order to participate or even plan their own work.

Slack [27] is a team communication tool that provides many benefits. The most significant to the project are listed below:

- Integrates all team communications in one place. Furthermore, the communications can be segmented into channels, organized by topics, and different users can be assigned to each channel, depending on the visibility the channel must have.
- Integrates other web services, e.g., GitHub, for notification and viewing code check-ins, and Dropbox and Google Drive, for file sharing.
- All content is searchable from one search box. Communications between several people can lead to large amounts of information that is later hard to find. Slack search filter options narrow the search on the conversations to specific channels, persons, or many other filters.
- Code snippets sharing. Slacks supports sharing code snippets with specific syntax highlighting. The platform also supports other members to download it, view it in a raw mode, or leave comments and modifications.

### 4.2.2 Integration Platform

This section defines the platforms that will be used for the integration of the ELASTIC project.

#### 4.2.2.1 Automated Build System

Different components of the ELASTIC project will use the Apache Maven [28] automated build system. Apache Maven is a project management and comprehension tool that manages project building, reporting and documentation from a central place. The primary goals of this platform are: (1) making the build process easy, (2) providing a uniform build system, (3) providing quality project information, (4) providing guidelines for best practices development, and (5) allowing transparent migration to new features.

Within the ELASTIC project, Maven will be used, at least, for the development and integration of COMPSs features. Since it is a Java tool, it also requires the Java SDK.

#### 4.2.2.2 Continuous Integration System

A continuous integration system, such as Jenkins [29] will be used to automatize the integration of the software components as well as for testing. Such a system focuses on two jobs:

1. Building and testing software projects continuously. Jenkins provides a flexible continuous integration system, making it easy for developers to integrate changes to the project, and making it easy for users to obtain a fresh build. The automated, continuous build increases the productivity.
2. Monitoring executions of externally-run jobs. This includes jobs such as *cron* jobs or jobs that are run on remote machines. The results of these jobs are kept by Jenkins, as well as sent to developers by email. Any form of checking these results allows developers to notice when something is wrong faster and easier than using traditional testing mechanisms.

### 4.2.3 Quality Assurance Tools

This section covers the quality assurance tools used within the ELASTIC project to provide means to test and control code and thus system quality.

#### 4.2.3.1 Issue Tracking Tool

The ELASTIC project will use GitLab [22] to track issues and feature requests. This application enables lean and agile project management from basic issue tracking to scrum project management. Specifically, it allows:

- Manage and track issues: (1) collaborate and define specific business needs, (2) track effort, size, complexity, and priority of resolution, and (3) eliminate silos and enable cross-functional engagement.
- Visualize work with issue boards: (1) visualize the status of work across the lifecycle, (2) manage, assign and track the flow of work, and (3) enable Kanban and Scrum styles of agile delivery.
- Maintain traceability through the DevOps Pipeline: (1) link issues with actual code change needed to resolve issues, (2) visualize and track the status of builds, testing, security scans, and delivery, and (3) enable entire team to share a common understanding of status.

## 4.3 Standards and Guidelines

Development guidelines provide a basic set of rules to enforce consistent and standardized coding practices. These guidelines are even more vital in a distributed software development project with teams at geographically separated locations. The guidelines in this section assure code quality and complement the processes defined in Section 0.

### 4.3.1 Design Patterns

Within ELASTIC, developers will use design patterns when applicable. Design patterns [30] are time-tested solutions to recurring design problems and offer several benefits:

1. Provide solution to issues in software development using a proven solution.
2. Design patterns make communication between designers more efficient.
3. Facilitate program comprehension.

### 4.3.2 Code Comments

Code comments help to explain and describe the actions of a certain block of code, describing behaviors that cannot otherwise be clearly expressed in the source language and easing comprehension. ELASTIC developers will comment crucial parts in the source code to help other developers understand their code.

In spite of numerous benefits of having properly commented source code, comments can be misguiding if not used properly. Thus a few points worth consideration while writing comments are:

1. Comments can get out of sync with the code if people change the code without updating the comments. Thus, comments should always change together with code.
2. Good comments are hard to write and time consuming, but pay off in long term.
3. Adding comments can be counter-productive if the information provided by them is not relevant to the part of code where they are provided. Hence, inline comments should describe the next line of code.

### 4.3.3 Programming Style

Programming style is a set of rules or guidelines used when writing the source code. These guidelines include elements common to a large number of programming styles such as the layout of the source code, including indentation, the use of white space around operators and keywords, the capitalization of keywords and variable names, the style and spelling of user-defined identifiers, such as function, procedure and variable names; and the use and style of comments.

Since the ELASTIC project will include several components that are already under development and follow their respective programming styles, developers in the frame of the ELASTIC project will follow these styles. For those parts of code which purpose is integrating different components of the ELASTIC ecosystem, the involved partners will define the programming style together the APIs specified in Section **Error! Reference source not found.**.

# 5. The COMPSs Software Component

## 5.1 General Description

COMPSs will be the main software component that will form the *orchestrator layer* of the ELASTIC ecosystem (see Figure 1).

COMPSs provides a complete framework, composed by a *programming model* and a *runtime system*, enabling the development of parallel applications for distributed infrastructures at a very low cost because of two main reasons: (1) the model is based on sequential programming on top of popular programming languages (i.e., Java, Python and C/C++), meaning that users do not have to deal with the typical duties of parallelization and distribution (e.g., thread creation, data distribution, fault tolerance, etc.); and (2) the model abstracts the application from the underlying distributed infrastructure, hence COMPSs programs do not include any detail that could tie them to a particular platform (e.g., deployment or resource manager) boosting portability among diverse infrastructures and enabling execution in a fog environment.

COMPSs applications are composed of three parts: (1) the *main program*, which is the code that is executed sequentially and contains calls to the user-selected methods that are to be executed as asynchronous tasks; (2) the *remote methods*, also called *Core Elements* (CEs), which are the implementations of the tasks; and (3) the *annotated interface*, which declares methods to be run as remote tasks along with metadata needed by the runtime to properly schedule tasks (e.g., the priorities of the tasks, the dependences among tasks, etc.).

## 5.2 Run-time Internals

Currently, the COMPSs runtime is organised in a *master-worker structure* (depicted in Figure 5):

(a) The *master* part executes in the resource where the application is launched, i.e., where the main program runs, and is responsible for steering the parallelisation of the application, as well as for implementing most of the features of the runtime concerning task processing and data management.

(b) The *worker* side is in charge of responding to task requests coming from the master, although in some designs such as clusters it also has data transfer capabilities, and it can be transient (i.e., a new runtime process is started every time a task request arrives) or persistent (i.e., a process remains in the resource all along the application lifetime).

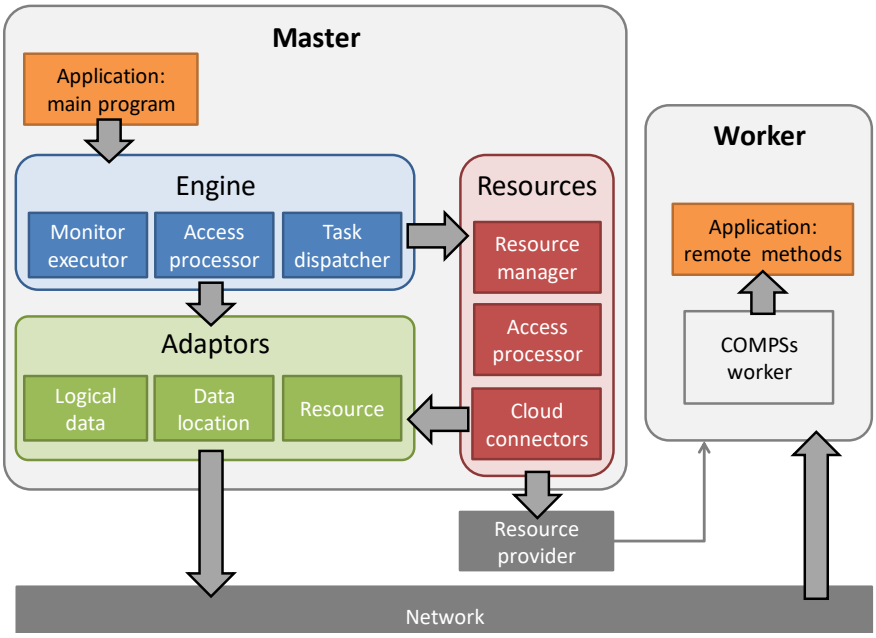


*Figure 5. Single COMPSs framework overview.*

Within the ELASTIC project, different COMPSs workflows may coexist (in different edge nodes and in the cloud), meaning that devices that behave as master in a

COMPSs framework, may behave as worker in a different COMPSs framework, leading to a *worker-worker* structure.

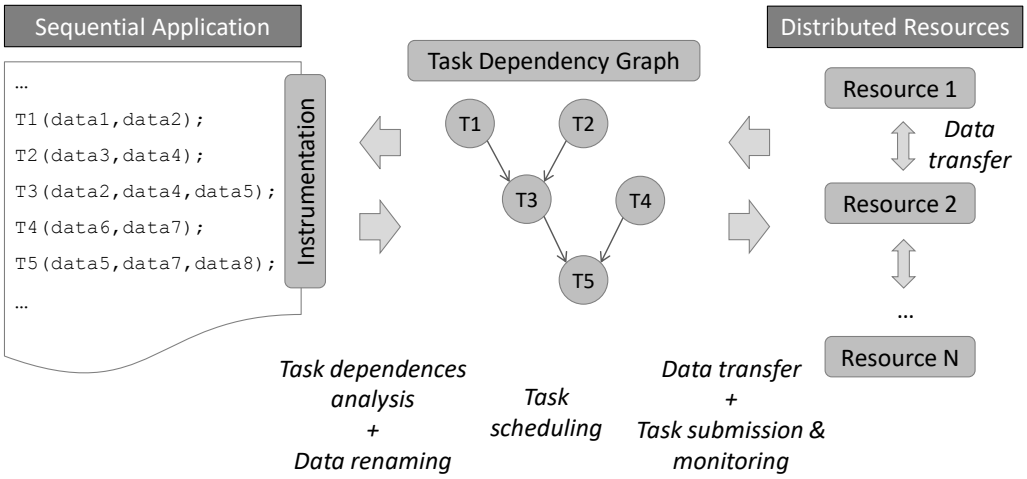


*Figure 6. Execution pipeline of a COMPSs application.*

The execution of a COMPSs program (summarized in Figure 6) involves the following stages:

1. *Instrumentation* of the main program. The first phase of the execution of a COMPSs application consists of two parts: first, the methods selected by the programmer are replaced by the asynchronous creation of their associated tasks, and then the data accesses specified for these methods are checked in order to ensure the sequential memory consistency.
2. *Data dependence analysis*. As the main program runs, the runtime receives task creation requests. The data consumed and produced by the task is used by the data dependence analysis mechanism, which dynamically builds a *Task Dependency Graph* whose nodes are tasks, and whose arrows symbolise the dependences. This graph represents the workflow of the application and imposes what can and cannot be run concurrently.
3. *Data renaming*. In order to expose more parallelism in the applications, data causing Write-after-Read (WaR) and Write-after-Write (WaW) dependences is renamed. The runtime keeps track of all data accessed by the application and the versions of this data created after the renaming process; hence it can guarantee the sequential memory consistency of the application.
4. *Task scheduling*. A task remains in the Task Dependency Graph until all its predecessors have completed, hence its dependences are solved. Then, using the list of worker resources the runtime is provided with, if the runtime is able to find an available resource, the task is scheduled. Otherwise, the task is added to a queue of pending tasks waiting for a free resource. Different policies allow mapping tasks to resources (e.g., based on data locality, in a round robin fashion, etc.). Additionally, there is a *pre-scheduling* mechanism is offered in the runtime in order to send the data needed by a task before the task can be executed, overlapping computation and communication. This way, when the processor where the task is to be executed gets free, the task can be submitted without waiting for any transfer.

5. *Task submission*, *execution* and *monitoring*. Once a task is ready to be executed, its input data is transferred and the target resource is free, then the master runtime asynchronously submits the task and registers the notifications coming from the worker resource informing about the completion of the task. In the worker resource, the worker part of the runtime is in charge of executing the task. Furthermore, the master runtime implements a fault-tolerant mechanism that allows for retrying the submission either in the same resource or in a different one. Finally, when the task completes, the runtime removes it from the Task Dependency Graph.

For the integration of COMPSs into the ELASTIC ecosystem, information about the NFR must be added to the scheduler. This will allow meeting the NFR expected by the user with the properties provided by the different components of the fog environment in terms of time, energy, security and communication features. In this regard, COMPSs applications, particularly the tasks that are to be executed asynchronously, shall be enriched with information to better characterise the NFR (e.g., priority, deadline, period, energy budget, safety integrity level) when necessary. Furthermore, the different resources shall provide timing properties (current workload) as well as the programming model supported. With this information, an enhanced version of the COMPSs scheduler shall be able to meet the requirements of the ELASTIC use-cases with the properties of the resources.

## 5.3 Interface with the Underlying Computing Devices

An important feature of COMPSs is the capability to execute applications transparently with regards the underlying infrastructure. With such a purpose, COMPSs implements the interaction between the runtime and the computational resources (i.e., physical resources or VMs) by means of different *adaptors*, each implementing the specific providers APIs. This mechanism makes possible the execution of computational loads on fog environments without the need of adapting the code, hence providing scalability and elasticity properties. Currently, there are two adaptors implemented: (1) *Non-blocking I/O*, *NIO*, which offers high performance in secured environments, and (2) *GAT*, which offers interoperability with diverse kinds of Grid middleware.

Together with the adaptors, the COMPSs runtime uses *connectors* to communicate with the cloud managers. Each connector implements the interaction of the runtime with a given provider's API, supporting four basic operations: (1) ask for the price of a certain VM in the provider, (2) get the time needed to create a VM, (3) create a new VM and (4) terminate a VM. This design allows connectors to abstract the runtime from the particular API of each provider and facilitates the addition of new connectors for other providers. Currently COMPSs implements four different connectors: (1) rOCCI (ruby Open Cloud Computing Interface) [10], (2) JClouds [11], (3) Docker [12], and (4) Mesos [13]. Although COMPSs provides many features for managing the cloud transparently, there is an important aspect that the framework does not take into account: the *NFR* imposed by the applications that will run on top of the ELASTIC ecosystem. For this reason, COMPSs will interact with both, the *NFR tool* and the *hybrid fog computing platform* (see Figure 1). In that regard, a new COMPSs connector will be implemented.

Flexibility is one of the main features of COMPSs. With that in mind, COMPSs has been integrated with several programming models in order to better exploit the capabilities of the different target architectures. This means the COMPSs programmers can define computing elements in several programming languages: (a)

OpenCL [31], for GPGPU programming [6], (b) OmpSs [15] for CPU, GPU and Cluster programming, or (c) MPI [32], for Cluster programming. The integration between COMPSs and OmpSs is particularly interesting, because OmpSs further integrates other programming languages such as CUDA [16], OpenCL and MPI [32]. This means that COMPSs not only allows for flexible, programmable and portable programming of both edge and cloud devices, but also supports a performance-aware environment where specific programming models can be used to exploit the special features of each target architecture.

# 6. Acronyms and Abbreviations

- CPS – Cyber-Physical System
- DoA – Description of Action (Annex 1 of the Grant Agreement)
- HPC – High Performance Computing
- IoT – Internet of Things
- M/R – Map / Reduce
- MS – Milestones
- NFR – Non-Functional Requirements
- VM – Virtual Machine
- WP – Work Package
- TR – Technical Requirement
- BG – Business Goal

# 7. Bibliography

[1] ELASTIC, “D2.5. Distributed data analytics platform requirements,” MS1, 2019.

[2] ELASTIC, “D4.2. Non-functional properties analysis and constraints specification,” MS1, 2019.

[3] ELASTIC, “D5.1. General requirements of the fog architecture,” MS1, 2019.

[4] Apache, “Apache Spark, https://spark.apache.org”.

[5] U. B. Riselab, “Pywren, https://rise.cs.berkeley.edu/projects/pywren/,” 2019.

[6] F. Lordan, R. M. Badia and W.-M. Hwu, “Enabling GPU Support for the COMPSs-Mobile Framework,” in *International Workshop on Accelerator Programming Using Directives*, 2017.

[7] U. B. RiseLab, “Ray, https://rise.cs.berkeley.edu/projects/ray/,” 2019.

[8] Apache, “Flink, https://flink.apache.org,” 2019.

[9] Apache, “Kafka, https://kafka.apache.org,” 2019.

[10] “A Ruby OCCI Framework,” 2013. [Online]. Available: https://github.com/ffeldhaus/rOCCI.

[11] “Apache jclouds®,” 2018. [Online]. Available: https://jclouds.apache.org/.

[12] “Docker,” 2018. [Online]. Available: https://www.docker.com/.

[13] “Apache Mesos,” 2018. [Online]. Available: http://mesos.apache.org.

[14] "OpenMP," 2018. [Online]. Available: www.openmp.org.

[15] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell and J. Planas, “OmpSs: a Proposal for Programming Heterogeneous Multi-Core Architectures,” *Parallel Processing Letters,* vol. 21, no. 2, pp. 173-193, 2011.

[16] NVIDIA, “CUDA C Programming Guide,” 2018. [Online]. Available: https://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.

[17] E. Consortium, “Deliverable 4.2. Non-functional properties analysis and constraints specification,” May 2019.

[18] K. Schwaber and M. Beedle, Agile Software Development with Scrum, vol. 1, Prentice Hall Upper Saddle River, 2002.

[19] *Grant Agreement Description of Action (DoA),* 2017.

[20] "JUnit 5," 2018. [Online]. Available: junit.org/junit5/.

[21] "Google Test," 2018. [Online]. Available: https://github.com/google/googletest.

[22] GitLab, "The first single application for the entire DevOps lifecycle," 2019. [Online]. Available: https://about.gitlab.com/.

[23] The Eclipse Foundation, "Enabling Open Innovation & Collaboratio," 2018. [Online]. Available: http://www.eclipse.org/.

[24] "Atom," 2018. [Online]. Available: https://atom.io/.

[25] "Apache Subversion," 2018. [Online]. Available: https://subversion.apache.org/.

[26] "Git," 2018. [Online]. Available: https://git-scm.com/.

[27] Slack, "Where work happens," 2019. [Online]. Available: https://slack.com/.

[28] "The Apache Maven Project," 2018. [Online]. Available: https://maven.apache.org/.

[29] "Jenkins," 2018. [Online]. Available: https://jenkins.io/.

[30] E. Gamma, R. Helm, J. Vlissides and R. E. Johnson, "Design Patterns Elements of Reusable Object-Oriented Software," in *Addison-Wesley*, 2000.

[31] J. E. Stone, D. Gohara and G. Shi, “OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems,” *Computing in Science & Engineering,* vol. 12, no. 3, pp. 66-73.

[32] “MPI: A Message-Passing Interface Standard, version 3.1,” 2015. [Online]. Available: https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

[33] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, 2008.

[34] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rose, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker and I. Stoica, "Apache Spark: a Unified Engine for Big Data Processing," *Communications of the ACM,* vol. 59, no. 11, pp. 56-65, 2016.

[35] E. Jonas, S. Venkataraman, I. Stoica and B. Recht, "Occupy the cloud: distributed computing for the 99%," in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017.

[36] F. Lordan, E. Tejedor, J. Ejarque, R. Rafanell, J. Álvarez, F. Marozzo, D. Lezzi, R. Sirvent, D. Talia and R. M. Badia, "ServiceSs: an interoperable programming framework for the Cloud," *Journal of Grid Computing,* vol. 12, no. 1, p. 67–91, 2014.

[37] R. M. Badia, J. Conejero, C. Diaz, J. Ejarque, D. Lezzi, F. Lordan, C. Ramon-Cortes and R. Sirvent, "Comp Superscalar, an Interoperable Programming Framework," *SoftwareX,* vol. 3, pp. 32-36, 2015.

[38] "ELASTIC Grant Agreement Description of Action (DoA)," 2018.