Developing Data-Intensive Cloud
Applications with Iterative Quality
Enhancements

Final assessment report and impact analysis

Deliverable 6.4

Deliverable:	D6.4
Title:	Final assessment report and impact analysis
Editor(s):	Vasilis Papanikolaou (ATC)
Contributor(s):	Danilo Ardagna (PMI), Laurie-Anne PARANT (NETF), Ismael Torres (PRO), Christophe Joubert (PRO), Elisabetta Di Nitto (PMI), Youssef RIDENE (NETF), Gabriel Iuhasz (IEAT), Joas Yannick KINOUANI (NETF), Matej Artač (XLAB), Tadej Borovšak (XLAB), Giuliano Casale (IMP), Pooyan Jamshidi (IMP), Yifan Zhai (IMP), Chen Li (IMP), Lulai Zhu (IMP) Marcello M. Bersani (PMI), Francesco Marconi (PMI), Michele Guerriero (PMI), Damian Andrew Tamburri (PMI), Craig Sheridan (flexiOPS Limited), José Ignacio Requeno (ZAR), Jose Merseguer (ZAR)
Reviewers:	Giuliano Casale (IMP), Elisabetta Di Nitto (PMI)
Type (R/P/DEC):	Report
Version:	1.0
Date:	31-Jan-2018
Status:	Final Version
Dissemination level:	Public
Download page:	http://www.dice-h2020.eu/deliverables/
Copyright:	Copyright © 2018, DICE consortium – All rights reserved

DICE partners

ATC:	Athens Technology Centre
FLEXI:	flexiOPS Limited
IEAT:	Institutul e-Austria Timisoara
IMP:	Imperial College of Science, Technology & Medicine
NETF:	Netfective Technology SA
PMI:	Politecnico di Milano
PRO:	Prodevelop SL
XLAB:	XLAB razvoj programske opreme in svetovanje d.o.o.
ZAR:	Universidad De Zaragoza

The DICE project (February 2015-January 2018) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644869

Executive summary

DICE aims at defining a general-purpose methodology and toolset to define data-intensive cloud applications. This means that applications defined with DICE span different domains. To validate the productivity gains induced by DICE, three industrial use cases on data-intensive applications have been carried out through the project (News&Media, e-Government and Maritime operations domains).

The objective of D6.4 is to review the productivity gains of the DICE tools and to show what is to illustrate their performance in laboratory and real-case business scenarios. Validation and verification of these tools has been based on predefined KPIs set by the tool owners at the beginning of the project and achieved, or even surpassed, by the demonstrators.

Exploiting DICE tools to analyze, deploy, monitor, test and improve real Data Intensive Applications (DIAs), not only improves their overall quality, but also significantly increases the productivity of the demonstrators and their development and engineering teams. DICE tools have proven to be both user friendly and practical, while providing significant results mainly to early stage validation, process automation and monitoring activities. DICE tools can be obtained from: https://github.com/dice-project/.

Glossary

ADT	Anomaly Detection Tool
AIS	Automatic Identification System
API	Application Programming Interface
CEP	Complex Event Processor
CPU	Central Process Unit
DDSM	DICE Deployment Specific Model
DIA	Data-Intensive Application
DICE	Data-Intensive Cloud Applications with iterative quality enhancements
DICER	DICE Roll Out Tool
DMON	DICE Monitoring
DPIM	DICE Platform Independent Model
DTSM	DICE Technology Specific Model
FCO	Flexiant Cloud Orchestrator
GMF	Graphical Modelling Framework
IDE	Integrated Development Environment
MVP	Minimum Viable Product
POC	Proof of Concept
QA	Quality-Assessment
TCT	Trace Checking Tool
TOSCA	Topology and Orchestration Specification for Cloud Applications
UC	Use Case
UML	Unified Modelling Language
URL	Uniform Resource Locator
VM	Virtual Machine
VPC	Virtual Private Cloud
VTS	Vessel Traffic System

Table of contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

In this deliverable, we showcase and summarize validation and verification results achieved by the tool owners and by the demonstrators (News&Media, e-Government and Maritime operations domains) within DICE. As D6.4 is the only *public* deliverable of WP6, material from previous restricted deliverables (such as D6.1, D6.2 and D6.3) servers the primary purpose of exposing the use cases to the wide audience, in addition to presenting for the first time results obtained in the final year of the project.

Emphasis has been given to activities and actions that took place over the last 6 months of the project (M30-M36), however significant information from work done during the first 30 months of the project is also presented and highlighted when needed.

Adding to the above, the current deliverable provides detailed information regarding validation activities and progress made on DICE tools, that have not been reported before and took place from M30 to M36. A number of tools have undergone significant changes and improvements over this period, while additional validation activities took place either in a lab environment or within the use cases (demonstrators). For example, in some cases, tools already evaluated in previous months, have been evaluated again in more depth in the last semester, to test the new functionalities added to the tools in year 3. As other WPs dedicated to tool improvements (ie. WP2, 3 and 4) do not have a dedicated deliverable for M36, the current document also contains information regarding latest technical development concerning the tools.

All DICE tools have been validated by at least one demonstrator, while in some cases tools have been validated in more than one demonstrators, validating the ability of the DICE framework to act as a general-purpose development environment. The decision on the optimal number and type of demonstrators to use to validate a tool has been driven by the Big data technologies used in the demonstrators, for example tools focused on Apache Storm have been tested mainly in the News&Media demonstrator, while those focused on Apache Spark have been primarily validated on the tax fraud detection demonstrator.

Table 1: DICE tools validated in Use Case environment

DICE Tool	Current Status		
Tool Name	ATC	proDEVELOP	NETFECTIVE
UML Profile	🟦	🟦	🟦
DICE IDE	🟦	🟦	🟦
Deployment Modelling Tool (DICER)	🟦	🟦	🟦
Delivery Tool	🟦	🟦	🟦
Verification Tool	🟦	⬜	⬜
Simulation Tool	🟦	🟦	🟦
Optimization Tool	⬜	⬜	🟦
Monitoring Tool	🟦	🟦	🟦
Anomaly Detection Tool	⬜	🟦	⬜
Trace Checking Tool	🟦	⬜	⬜
Enhancement Tool	🟦	⬜	🟦
Quality Testing Tool	🟦	⬜	⬜
Configuration Optimization Tool	🟦	⬜	⬜
Fault Injection Tool	🟦	🟦	🟦

The following table summarises the KPIs achieved by the demonstrators.

Table 2: DICE Tools KPIS

DICE Tool	Validation KPI	KPI Target	Status of validation KPIs at M36 (Use Cases)	KPI achieved by at least one Demonstrator
Delivery tool (+ DICER)	Time from deployment modelling to deployment of the DIA	Reduction of 50+% of deployment times with respect to no-modelling approach	Reduction experienced by ATC > 70% Reduction experienced by NETF > 90% Reduction experienced by PRO >80%	Yes
Verification	Violations of properties on timing constraints identified by Verification tool	>=2	ATC: 4	Yes
Simulation	Prediction error of response time. Prediction error of throughput. Prediction error of resources utilization	<=30%	PRO: Response time: max. error 18% Throughput: max. error 5% Utilization: max. error 13.2% ATC: Utilization: max. errors below 20% Throughput: max.error 16.41%	Yes
Optimization	Cost prediction error	<=30%	NETF: The average percentage error cost estimate overall was 3%.	Yes
Monitoring	Time to configure the monitoring across the DIA; monitoring overhead	<=30% decrease; <5% overhead	ATC: Time to configure: 40% decrease PRO: Time to configure: 94% reduction For both case overhead is negligible, <5%.	Yes
Anomaly detection	False positives	<10%	PRO: 6,6% of false positives using the Anomaly detection tool	Yes
Trace checking	Number of refined parameters occurring in the model verified with Verification tool	>=1	ATC: 2	Yes
Enhancement	Number of anti-patterns detected in one demonstrator	>=1	ATC: Number of anti-patterns: 2	Yes
Quality testing	Manual time required in a test cycle	>30% reduction	ATC: Reduction per Test Cycle: 34,78%	Yes
Configuration optimization	Difference in latency or throughput compared to default config	>30%	ATC: Throughput (m/s): ~128% incr Latency (ms): ~86.6% decr	Yes
Fault injection	Manual time required in a test cycle	>30% reduction	PRO: 66,6% Time saving ATC: 20% time saving	Yes (on average)
DICE IDE	Number of DICE tools used in each demonstrator	4	ATC: 8 PRO: 7 NETF: 8	Yes

1.2 Work done on Containers

During the last semester of the project, considerable effort has been dedicated to support the use of **containers** within the DICE project. This is the main extension achieved for the framework in year 3 of the project. Although not a planned activity at the beginning of the project, given the relevance that this technology is acquiring and its potential benefits, upon suggestions from our reviewers, we considered to dedicate resources to support and validate it in one of the uses cases.

The reasoning for supporting containers was twofold. First, the exercise of adding the container support across the DICE tools stack acted as a further validation that showed how extensible the DICE tools are and what it takes to add a new technology paradigm. As discussed in D7.7, for example as a result of feedback obtained in public presentations of DICE, end users often want to know if a certain technology can be added to DICE, and the response is affirmative based on the experience reported in this deliverable. Second, the popularity of the containers in the industry has grown so high during the DICE project, that the need for their support would naturally occur from the users.

We therefore extended both the design part of the DICE tools as well as the runtime part. In the design part, the DICE profile has been modified to allow specifying containers at the DDSM level. The DICER tool has been extended to support the generation of blueprints from contained-based deployment models. The optimization tool has been used to identify the minimum cost configuration of the BULMA application (developed in collaboration between DICE and the EUBRA-BIGSEA project) which implements a predictive machine learning algorithm for Intelligent Transport Systems (bus trajectories identifications) and which exploits containers.

To enable the transfer to runtime, we extended the DICE TOSCA technology library with support of the Docker flavour of the containers. This means that the DICE Deployment Service is capable of deploying applications that include both components running in classical virtual machines combined with ones running within containers.

The Posidonia Operations use case (maritime operations) has been used to validate the automatic deployment of the solution with containers. To make use of good practices in handling containers, some components have been repackaged in order to better separate the functionality of the different services and have a container for each service.

Another technology that on the outside appears related to containers is the technology of **microkernels**. This is a considerably slimmed down version of the operating system, which lets specially prepared applications to run close to hardware. The technology is gaining traction in the industry and thus has a potential to catch up with containers. From the **MIKELANGELO** project comes **OSv** as a specific microkernel solution. XLAB being in both projects had a unique opportunity to include support for OSv into DICE deployment stack and to showcase a Big Data cluster deployment. PMI has extended the deployment modelling metamodels accordingly, further extending the possible reach and usage of the DICE toolset.

1.3 Document Contents

The current document includes the following sections:

- **Section 1** provides an overview of this current deliverable by presenting the overall structure and concept of it.
- **Section 2** is dedicated to all DICE tools and provides evidence and material regarding validation and verification activities which took place throughout the project's duration, while emphasis has been given to the period from M30 to M36.
- **Section 3** focuses on the three demonstrators (use cases) and provides information regarding their validation activities of DICE tools in their business environments while presenting the use cases as a whole to wider audience.
- **Section 4** is dedicated to drawing the high-level conclusions of this document.

2 DICE Tools

2.1 Deployment Modelling Tool (DICER)

DICER is a tool for enabling the model-driven deployment of data-intensive applications (DIAs) leveraging the Infrastructure-as-Code (IaC) paradigm. More specifically, **DICER adopts the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) standard** as an architecture deployment description formalism and is able to automatically generate IaC for DIAs in the form of TOSCA blueprints from DICE-stereotyped UML models. To this aim, DICER exploits the DICE-Profiles applied on UML Deployment Diagrams as well as the DIA-specific TOSCA library, both defined and developed in the scope of the DICE project.

While the technical advancements and the majority of features for the DICER tool were discussed previously in the context of deliverables D2.3, D2.4 and D2.5, the improvement work aiming at addressing the feedback received by project reviewers and by our advisory board members did not stop and addressed 3 major technical challenges, namely: (a) containerisation; (b) varied DB support; (c) monitoring-as-a-service. Following this work, the **DICER tool addressed new major additions to encompass the modelling of generic nodes including generic containers** which can now be modelled as standard DDSM constructs and generated as TOSCA blueprint stubs inside DICER-generated TOSCA IaC. Moreover, DICER is now **capable of building IaC** which includes the "monitorable" feature, such that the Deployment Service can deploy monitoring facilities specific to a certain node. Finally, the DICER exposed a limited support to data-intensive DB technology - in this respect, the newest version includes support to MongoDB as further technological package inside DICER modelling and automated deployment facilities.

The validation methodology for DICER employed a case-study research approach. More in particular, the DICER team (WP2 and WP5) involved the case-study owners on a series of controlled experiments aimed at understanding:

1. The chronometric gains of using DICER in action, controlling variables such as experience, background, and model-savviness of the observer;
2. The amount of IaC saved by employing DICER in action;

As described in Section 3, all case studies have evaluated the DICER, with PRO exploiting also the containerization approach, NETF the varied DB support and all the monitoring-as-a-service features. Evaluation results show a considerable **gain of 7x** for the times that are saved by using DICER in action instead of classical coding-intensive approaches. Moreover, the quantity of code saved with DICER is also considerable, **reaching about 9x times the amount of code needed in a classical approach**.

Table 3: Overview of the in-lab experiments with DICER

Blueprint	Used Technology	Max dep.	Num. Elems	Avg generation time [s]	Gen. LOC	Avg deploy. time [s]
Pi Calculus	Spark	3	3	1.18	294	442
WordCount	Zookeeper Storm	3	4	1.26	393	445
Hadoop cluster	HDFS YARN	4	5	1.31	646	550
WikiStats	Storm Zookeeper Cassandra	4	7	1.69	782	480
WikiStats (with trace-cheking service)	Storm Zookeeper Cassandra Spark HDFS	5	10	1.93	1175	675

For an in-depth assessment of the correctness of the infrastructural code generation as well as effectiveness of the tool in terms of possibility to compose different technologies in a simple way and in terms of reduction of lines of infrastructural code to be developed, we performed also some in-lab controlled experiments. More specifically, to verify that DICER works correctly without introducing a significant overhead, we have experimented with various DIAs that exploit different combinations of the technologies we currently support. Table I shows a summary of some of these experiments. In all the cases, we developed the deployment model for the corresponding DIA and we generated and deployed the resulting blueprint on our internal OpenStack Mitaka infrastructure. In all considered cases, the process led to a correct deployment fulfilling our expectations. The first column in **Error! Reference source not found.** ncludes the names we have assigned to the considered DIAs and the second lists the used technologies. From this column, the reader can see that we have experimented with an increasing combination of technologies to achieve deployments with a different level of complexity. The *Num. Elems* column reports the number of UML modelling elements used to model each DIA. The *Max dep.* column represents the length of the longest path in a blueprint's dependency graph, not counting virtual resources that are created almost instantly. For example, level 1 represents an architecture with services that are all peers and where none of the services depends on any other service. Level 2 represents a two-tier architecture, where a service (e.g., a database) needs to be deployed before another one (e.g., an application) can be deployed and configured. The larger is the value of *Max dep.* the larger is the complexity of the deployment procedure to be executed for the corresponding DIA as it requires a specific deployment order to be kept as well as a runtime configuration of the dependencies between the involved components. *Gen. LOC* reports on the number of Lines of Code of infrastructural software generated by DICER in each experiment. This number gives an idea on the amount of low level programming effort saved by DICER users. From the experiments we have conducted, it appears that this number depends on the number of modelling elements.

To quantify the time needed to generate and deploy blueprints, we repeated the experiments 10 times and com- puted the average duration for generating the blueprint (fifth column in the table) and for automatically deploying it through the deployment service (seventh column). As we expected, we can see that the blueprint generation time is significantly lower that the time needed to deploy all considered cases.

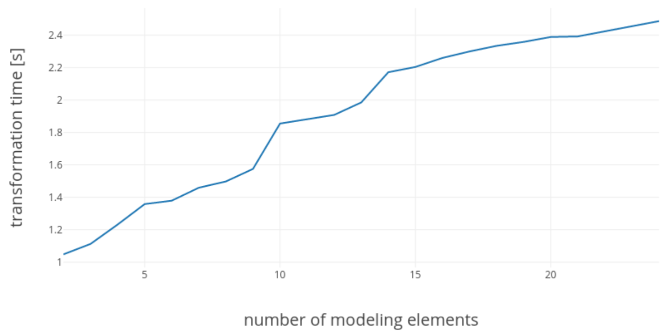


Figure 1: TOSCA blueprint generation avg times per number of modeling elements

To see if DICER was able to handle also cases more complex that the ones in the table, we carried out a scalability anal- ysis starting from a simple model with a single VMsCluster including one technology element and by gradually increasing the complexity up to a case that included, 8 VMsClusters, 8 distinct technology-specific elements (e.g. StormCluster, CassandraCluster, etc.) and 8 distinct DiaJobs. Such models were randomly generated and the amount of generated infrastructural code ranged from 150 lines, for the simplest model, to 2092 lines for the more complex one. We ran each single experiment of blueprint generation 30 times, averaging the resulting generation time. In all cases DICER behaved correctly and generated the corresponding executable bluprint. **Figure 1** plots the time needed for such generation that remains always within acceptable boundaries.

2.2 Delivery tool

Transitioning from design environment to the runtime involves having to follow the deployment diagram of the DIA by taking repetitive steps in an order prescribed by both the deployment diagram and the specifics of the DIA's components. An important DevOps tool makes these steps automatically, continuously and reliably. The DICE Deployment Service with the help of DICE TOSCA technology library and DICE Configuration Optimisation assumes the role of such a tool in the DICE Methodology.

At the end of M30, the delivery tools reached a phase that was feature complete: it supported all of the core DICE technologies, and the same blueprint could be deployed without change to any of the supported infrastructures: FCO, OpenStack or Amazon EC2. After that, we proceeded to **validate the tool when used on Amazon EC2** in order to assess the maturity of the tool for the purposes of the Posidonia Operations (described later in the report). We also validated the tool ability to adopt new platforms by extending the technology support to include **container and microkernel technologies**.

2.2.1 Validation in Amazon EC2

A good application orchestration needs to handle various levels of abstractions in a sensible way. The goal of modelling the deployments is to capture all the necessary details of an application, while at the same time it doesn't have to go too much into detail. It has to specify all the components of the DIA, their relationships and any configuration that deviates from the defaults. But when it comes to representing the hosting environment, i.e., the cloud provider's IaaS details, it is safe to assume that they all share common elements. For example, they all offer compute and networking capabilities. Therefore, the deployment representation can hide these details or move it away from the DIA's representation.

From the perspective of providing such an abstraction, an orchestrator (or, more specifically, the plug-in that implements specific technology and platform support) has to work out differences and peculiarities that do exist between providers. It is therefore up to the tool owner to perform the validation that the behavior of the deployment is comparable to the one observed previously in FCO and OpenStack.

Our approach in this validation was to assess the scenario involving first bootstrapping the delivery tool and then using it to make the first DIA deploy. The assumption therefore is that the user has a compiled Storm application[1] (i.e., a .jar file) and a blueprint for the DIA as a .yaml file describing the whole Spark cluster, and an Amazon Web Services (AWS) account with permissions to manage resources through the AWS's web API. The goal of the validation was to show that the DIA can be successfully deployed in the Amazon EC2 environment. The time needed for this to occur also needs to be reasonably short: within 3 hours for the first deployment, and less than 30 minutes for each redeployment.

Table 4: Timings for Amazon EC2 deployment of a Spark application

Task	Time required [minutes]
AWS account setup (manual)	30 min
Collecting AWS parameters (manual)	60 min
Running bootstrap steps (manual)	25 min
Cloudify Manager bootstrap (automated)	15 min
DICE Deployment Service bootstrap (automated)	10 min
DIA deployment (automated)	15 min
Total first deployment time	**2 h 35 min**
DIA undeployment (automated)	2 min
DIA deployment (automated)	15 min
Total time subsequent redeployment (per redeployment)	**17 min**

[1] Storm WikiStats: https://github.com/dice-project/DICE-WikiStats/tree/master/storm-wikistats

In our experiment, we followed the DICE Deployment Service Administration Guide[2] to set up the DICE Deployment Service. Then we first ran the deployment of the Spark application by submitting the blueprint to the DICE Deployment Service instance set up in the previous steps. The same step was for redeploying the same blueprint, but this includes first the process of clearing the previous deploy.

Table 4 shows the timings that we recorded during this process. The timings apply to an engineer with basic understanding of operating with command line interfaces, configuration files editing and AWS handling.

From the results we can see that we can obtain a working DIA runtime within approximately 2:30 h. At the same time this gives us a platform which lets us have new deploys of the application, complete with the entire Spark stack, in less than 20 minutes.

2.2.2 Container and microkernel support

The main purpose of the DICE Deployment Service has been to automate deployment and configuration of the DIAs by installing each of the needed component in a virtual machine[3]. In this approach, all the installation and composition happens on the fly according to appropriate Chef Cookbook recipes that are a part of DICE TOSCA technology library.

In the final 6 months of the project, we have also examined alternative approaches, which bundle parts of components ahead of time into an image. This image is then executed as a container in a micro-virtualised environment. The approach has become popular in the industry, because it enables simpler handling of individual units of processing and computation, which in themselves might be quite complex to set up. They are also easier to deploy in multi-cloud setting. DICE use case partner Prodevelop has recognised the usability of the Docker, a representative that is prominent in the container technologies community, for their internal development and testing processes. This was a valid motivation for us to consider extending support in the DICE TOSCA technology library to include Docker.

With our support for container technologies we also wanted to make a few more steps ahead. Thanks to collaboration with the MIKELANGELO[4] project, we also provide support for a microkernel technology called OS^v [5]. Microkernels are related to containers, as they provide a slimmed-down operating system that runs a component in a lightweight fashion. The added benefit of this technology is that it speeds up deployment of individual instances of the component, and it also lowers the overhead on memory and CPU that traditional operating systems apply to any component.

To both validate and demonstrate the support for new technologies, we revisited our city traffic use case [D5.3]. The use case is a part of collaboration with TIMON[6], and it collects and processes public data emitted from sensors placed at streets in Ljubljana and published by the public transport company. Focussing on the batch processing part of the use case, we introduced the following extensions and updates:

- We added a web user interface, packaged in a Docker image.
- We replaced the Spark worker nodes set up on virtual machines with Spark workers packaged as OS^v images.

Figure 2 shows the blueprint representation from Cloudify[7]. We should point out that in this case, the Docker images used in the blueprint need to be prepared ahead of time and uploaded to a Docker repository. Also, for the OS^v components, their images need to either be bundled with the blueprint upon

[2] Administration Guide: https://github.com/dice-project/DICE-Deployment-Service/blob/master/doc/AdminGuide.md
[3] Theoretically, bare metal deployments are also possible, but the use cases had no requirement for it so far.
[4] MIKELANGELO project: https://www.mikelangelo-project.eu/
[5] OS^v: http://osv.io/
[6] https://www.timon-project.eu/
[7] Cloudify: http://cloudify.co/

submission for deployment, or they need to be available in the target interface as an operating system image (e.g., an AMI on the Amazon EC2).

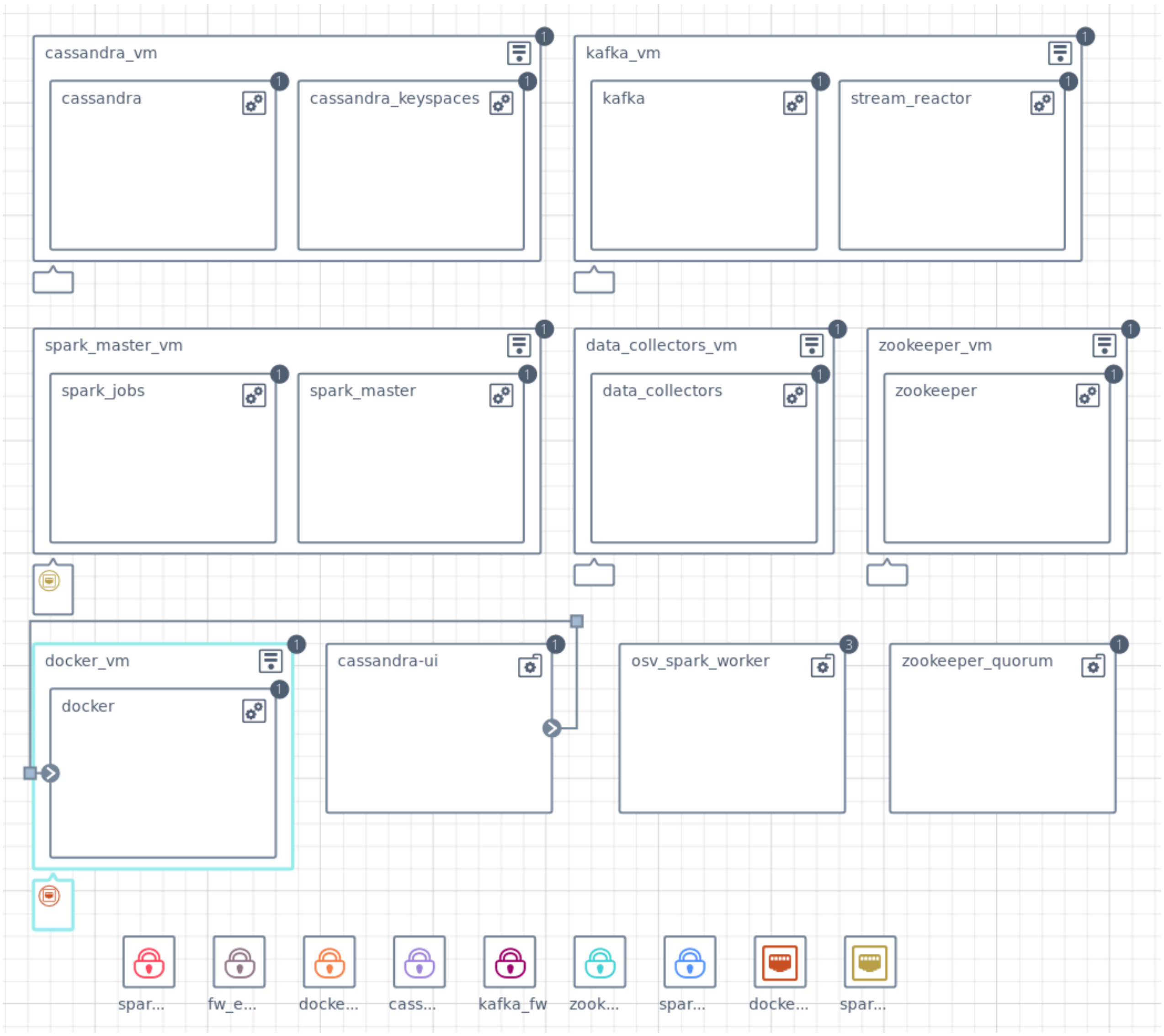


Figure 2: Screenshot of DICE Traffic Application blueprint as shown by Cloudify Web UI

The outcome of deploying this blueprint to the DICE Deployment Service is as expected: in the target environment (OpenStack or Amazon EC2) we obtain a working runtime of our use case DIA. This demonstrates that it is possible to freely mix various complementing technologies into the application, making composition of DIAs even more flexible.

We should point out that in the case of the OS^v components, preparing the microkernel editions requires special customizations of the source component in order for it to comply with the rules and capabilities of the OS^v operating system. This requires a certain effort of a specialized person, so it cannot be trivially applied to any application or component. But to see the benefit of doing this from the perspective of the DevOps, we compared the time of deployment of a blueprint using microkernels with the deployment time of a regular (i.e., virtual machines only) blueprint. **Figure 3** and **Figure 4** show the deployment sequences first of the blueprint where all Spark components are deployed as VMs, and second as the result of the deployment of the blueprint shown in the **Figure 2**. Each horizontal line shows the time that the orchestrator was active configuring an instance of a specific node template.

Deploying traffic application using prepackaged OSv components (we only replaced Apache Spark workers) reduced the deployment time from 750 to 500 seconds on average, a reduction by a third from the VM-only deployment time.

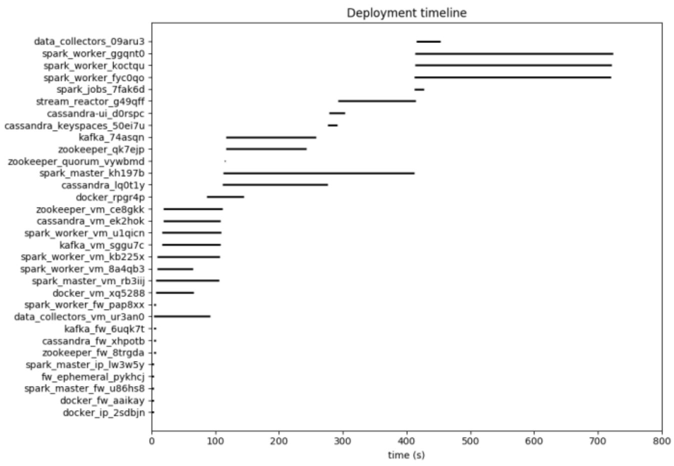


Figure 3: Timings for regular (VM-only) traffic application deployments

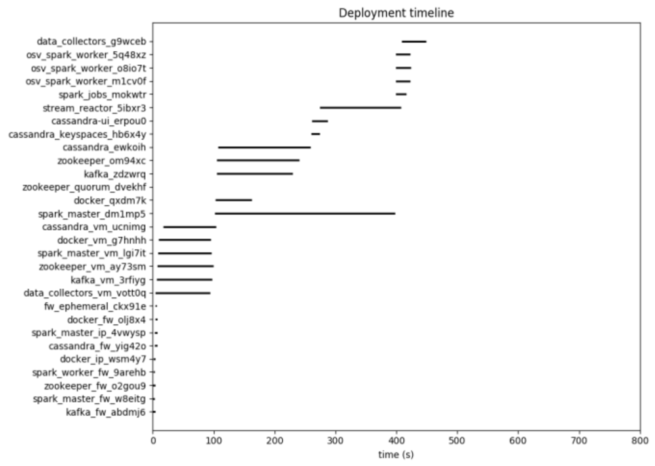


Figure 4: Timings of OSv traffic application deployments

2.2.3 Summary

The validation activities that we continued to perform by the project's M36 demonstrate the value and benefits of using the DICE Delivery tool. The originally supported core technologies already enabled composition and automated deployment of versatile and complex DIAs, with the possibility to supply custom components using a Script Runner node type. With the added support of Docker and OSv, the spectrum of possible DIAs is now even wider.

Our own internal validation process was also complemented with the feedback we obtained from some of the external users. This includes users from the use case partner groups in the consortium (who report their findings from their own perspective later in this document) and a small number of external users. From their experience and support queries we find that a certain step in the learning curve remains: the knowledge of handling system administration tasks. An unavoidable prerequisite for the DICE Delivery tool to work properly is that the target datacenter or cloud is properly configured in terms of the network, operating system images, etc. In a public platform such as the Amazon's EC2, it is possible to prescribe in the instructions the majority of the parameters, so in that case the overall user experience is higher. But in custom, private environment such as an OpenStack testbed, there are more possibilities of unforeseen misconfiguration that may block the deployment process.

Assuming that a customer employs a reasonably trained system administrator, then they can set up and configure the DICE Delivery tool from zero to a running environment in a few hours. The resulting services then enable quick deployments of the DIA blueprints. But the users and customers who are new to Cloud environment may need additional help before they can take full advantage of the tool. As a part of the work to improve and strengthen the DICE offering we therefore consider expanding on the documentation. However, this measure might get an opposite effect by potentially making the instructions alone too intimidating to the users. So as a countermeasure we plan to include additional checks of the user's input against the target environment.

2.3 Verification Tool

The DICE Verification Tool (D-VerT) is the tool enabling the formal analysis of safety properties on Data Intensive Applications (DIAs) from the DICE Platform.

D-VerT allows application designers to evaluate their design against safety properties, such as reachability of undesired configurations of the system, meeting of deadlines, and so on.

The analysis, which is mainly focused on the timing behaviour of the applications, follows a model-driven approach: applications are defined by DIA designers as DICE-profiled UML models through the Papyrus editor of the DICE Platform.

Once the model is complete and the needed configuration parameters are provided, D-VerT automatically generates the instance of the formal model corresponding to the UML model, and performs the formal analysis on it. The outcome of the analysis is then reported in the DICE IDE and is employed by the designer of an application for refining its model at design time, in case anomalies are detected.

The formal models are based on a Metric Temporal Logic formalism, and the analyses exploit Bounded Satisfiability Checking techniques.

Currently the tool supports the analysis of both streaming applications designed in Storm (buffer saturation analysis) and batch applications using the Spark technology (deadline feasibility analysis).

Since M30, various validation activities and refinement have been carried out. The main improvements regarded both the integration with the DICE Platform and the internals of the tool. In particular, some optimizations have been studied for the formal models in order to improve verification performances.

The target KPI for DICE Verification Tool indicates that the number of violations of properties on timing constraints identified by the tool must be >= 2, and this has been achieved on the considered case studies.

The following sections summarize the validation activities that have been performed for the Storm and Spark technologies on a number of use cases. The validation on the ATC case study is described in detail in **Section 3**.

2.3.1 Apache Storm

The Storm technology has been validated on various scenarios. The first validation activity, carried out on a simple topology having 2 spouts and 3 bolts, was presented in Deliverable D3.5 [9]. Next, the *FocusedCrawler* topology, with different configurations, was used as validation in [10]. In order to evaluate the performance and the scalability of the tool, we carried out many experiments on the presented topologies, by varying the topology parameters and the number of bolts considered. The following table shows time and memory consumptions statistics we collected.

Table 5: FocusedCrawler time and memory consumption statistics

Topology Configuration	N. of Bolts	Duration	Max Memory
focused-crawler-complete	8	2664s	448MB
focused-crawler-reduced-cfg-1	4	95s	142MB
focused-crawler-reduced-cfg-2	4	253s	195MB
focused-crawler-reduced-cfg-3	4	327s	215MB
focused-crawler-reduced-cfg-4	4	333s	206MB
focused-crawler-reduced-cfg-5	4	3184s	317MB
focused-crawler-reduced-cfg-6	4	1060s	229MB
focused-crawler-reduced-cfg-7	2	14s	79MB

While the previously mentioned validations were mostly focused on the evaluation of the formal model and the server component of the tool, [11] and [12] show the complete usage of D-VerT (from UML modeling to verification) for the iterative improvement of a *web crawler* topology. The tool identified 2 major violations in the topology, and helped the refactoring of the model.

Since M30, the tool was validated both on the ATC use case and on the Wikistats application.

As described with more details in **Section 3**, the formal analysis identified the presence of 4 violations in the *crawler* topology of the ATC use case, and allowed, by means of the iterative refinement of the model, for their resolution.

The Wikistats topology is an application performing analytics on web content from the Wikipedia website in a streaming fashion and stores the resulting statistics to a NoSQL database. We focused on the Storm topology and modeled it as a DICE-profiled UML Class Diagram using the Papyrus UML editor of the DICE Platform (**Figure 5**).

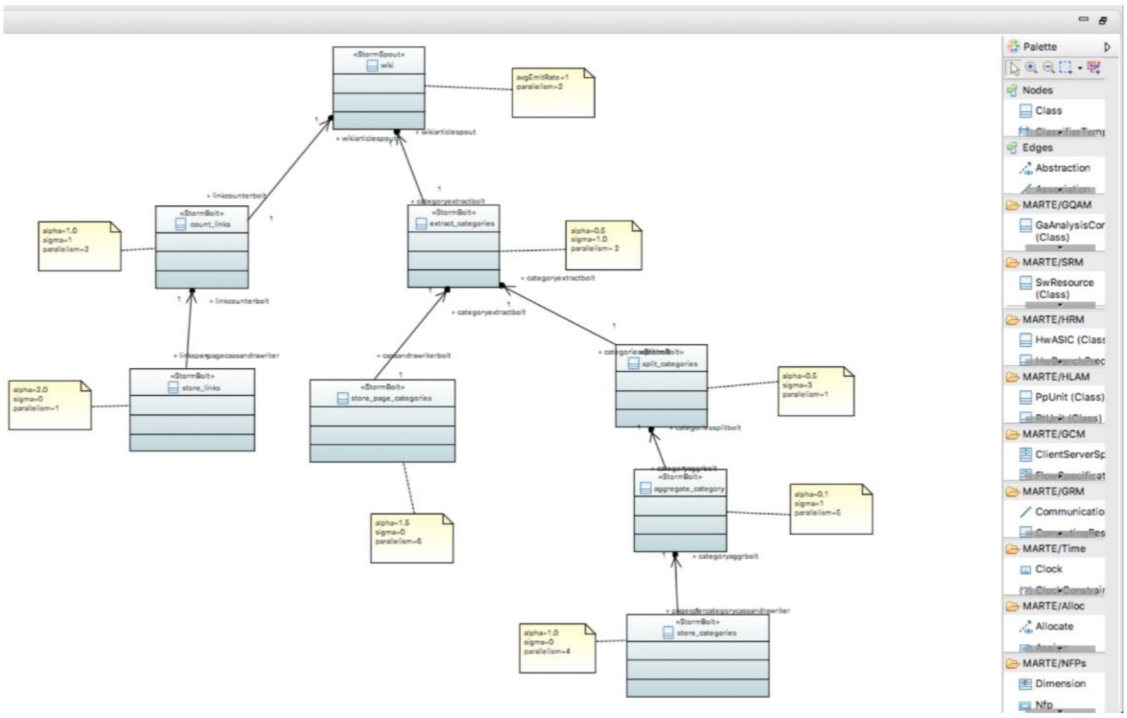
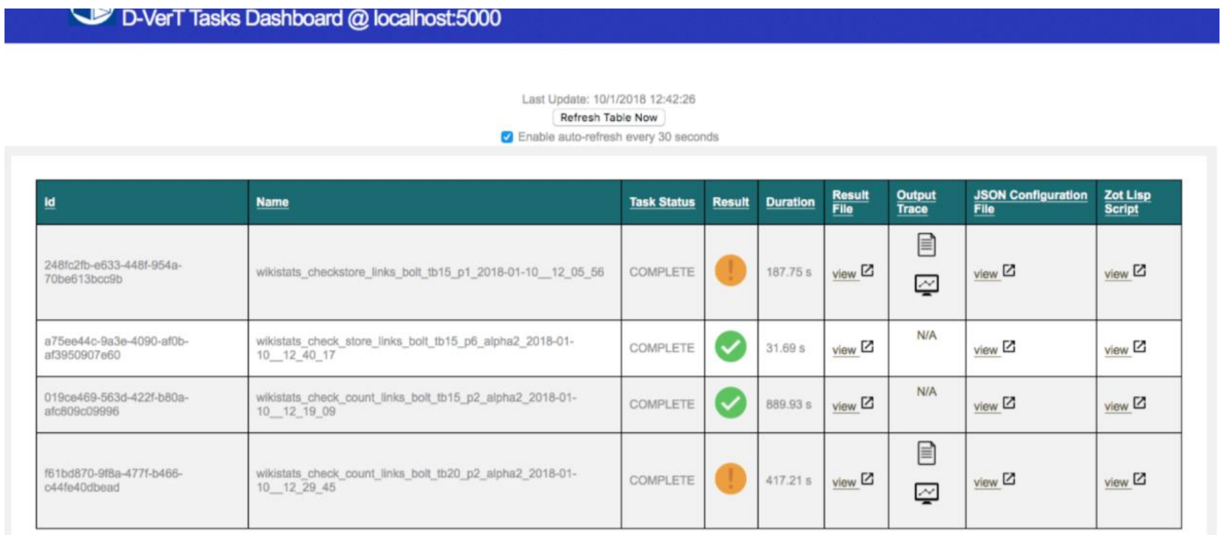


Figure 5: UML Class Diagram describing the Wikistats topology

Several verification tasks have been performed, focusing on the different nodes in the topology.

D-VerT identified a first violation in the *count_links* bolt, providing an execution trace in which the number of messages in its input buffer grows in an unbounded way. After a refinement of the component, consisting in the improvement of its execution time (*alpha* parameter) the issue was solved as the tool did not detect any further problem on the bolt. However, another violation was found in the *store_links* bolt. This outcome motivated a further modification in the topology model: by increasing the level of parallelism of the *store_links* bolt the verification task did not find any more issues in the component.


D-VerT Tasks Dashboard @ localhost:5000

Last Update: 10/1/2018 12:42:26

Refresh Table Now

Enable auto-refresh every 30 seconds

Id	Name	Task Status	Result	Duration	Result File	Output Trace	JSON Configuration File	Zot Lisp Script
248fc2fb-e633-448f-954a-70be613bcc9b	wikistats_checkstore_links_bolt_tb15_p1_2018-01-10__12_05_56	COMPLETE	!	187.75 s	view		view	view
a75ee44c-9a3e-4090-af0b-af3950907e60	wikistats_check_store_links_bolt_tb15_p6_alpha2_2018-01-10__12_40_17	COMPLETE	✓	31.69 s	view	N/A	view	view
019ce469-563d-422f-b80a-afc809c09996	wikistats_check_count_links_bolt_tb15_p2_alpha2_2018-01-10__12_19_09	COMPLETE	✓	889.93 s	view	N/A	view	view
f61bd870-9f8a-477f-b466-c44fe40dbead	wikistats_check_count_links_bolt_tb20_p2_alpha2_2018-01-10__12_29_45	COMPLETE	!	417.21 s	view		view	view

Figure 6: D-VerT dashboard showing verification outcomes for the Wikistats use case

Figure 6 shows the graphical interface through which it is possible to consult the status of the launched verification tasks. The execution times of the verification tool we collected during the validation on the Wikistats use case range from tens of seconds to tens of minutes, depending on the configuration.

2.3.2 Apache Spark

Validation activities for the Spark technology were first shown in deliverables D3.6 [13] and D3.7 [14], both considering a simple application performing a series of transformations on a text file.

Since M30, extensive experimental validations have been made on three different well-known applications: the simple SortByKey operation, the graph processing algorithm PageRank, and the clustering procedure K-Means.

The execution DAGs underlying the three applications have different sizes and different levels of parallelism, and have been executed and analyzed several times with different settings. This allowed us to evaluate both the accuracy of the formal model against realistic use cases and the scalability of the approach.

We performed the *deadline feasibility* check on all the settings of the different applications, varying the deadlines and recording the execution times.

On average, the tool was able to provide an outcome for feasible deadlines in less than a minute for the small-sized *SortByKey* examples, in a few minutes for the medium-sized *Pagerank* examples, and in tens of minutes (even hours) for *K-Means*, the example with the largest size. On the other hand, we registered extremely higher execution times for the feasibility analysis of deadlines that turned out to be unfeasible, but close to the minimum feasible deadline. In some of these cases, the tool took days you terminate the analysis. Since the difference between the analysis of feasible and unfeasible deadlines were so important, we decided to introduce a timeout to stop the analysis after an amount of time that (according to our experience and supported by experimental evidence) was reasonably high to conclude that the deadline is unfeasible.

Experiments have shown promising results in terms of the accuracy of the model. The percentage difference between the minimum feasible deadline found by the formal analysis and the actual (average) execution time of the application running on a cluster, was below the 10% on all the settings we considered. The complete experimental evaluation is available at [15].

After this activity, we implemented some refinements to the Spark formal models aiming at the reduction of the model size and the mitigation of the state space explosion problem. The final goal of these refinements is to reduce memory consumption and the execution time needed to perform verification. The first experiments show that, on average, the memory consumption halved with the new version of the model and the execution time decreases up to the 80% in some settings.

2.4 Simulation Tool

The DICE Simulation tool is a simulation-based tool for reliability and efficiency assessment of Data Intensive Applications (DIA). Simulations carried out by the Simulation tool are model-based simulations. For performing its model-based simulation task, the tool takes as input UML models annotated with the DICE profile. It uses Model-to-model (M2M) transformations, that transform the DIA execution scenarios represented in these profiled UML models into Petri net models. Then, the tool evaluates these Petri nets, and finally uses the results of the Petri net evaluation to obtain the expected performance and reliability values in the domain of DIA software models.

At the end of M30, the Simulation tool reached a phase that was feature complete. Currently, the Simulation tool computes the following metrics:

- **Performance assessment:** Utilization of hardware devices and execution threads, response time and throughput of the application.
- **Reliability assessment:** MTTF, availability and probability of continuous correct operation.

The target Key Performance Indicator (KPI) for the Simulation tool specifies that the tool must predict the performance metrics with less than 30% of error on average. The DICE Simulation tool satisfies this KPI.

The Simulation tool has been validated at DPIM and DTSM level for reliability and efficiency assessment of DIA. In particular, performance evaluation has been validated for Posidonia Operations and BluAge case studies at DPIM level, and News Orchestrator at DTSM level for Apache Storm (see Section 3). Besides, the Simulation tool has been also validated with synthetic data at DTSM level for Apache Hadoop [3,2], Apache Storm [5,2] and Apache Spark [6]. In the following paragraphs we detail the validation of every technology step by step. The Simulation tool is currently being validated for BluAge case study at Apache Spark level.

2.4.1 Apache Hadoop

The validation results for Hadoop MapReduce has been already presented in Section 6 of the DICE Deliverable 3.8 [1] and Section 3 of DICE Deliverable 3.1 [2]. The results have been published in [3] and are here summarized.

The experiments were executed on Amazon EC2 and CINECA, the Italian supercomputing center. The target version was Hadoop 2.6.0. We studied the performance for various configurations of the Hadoop MapReduce framework. That is, we analyzed the simulation errors for different number of mappers (nMaps), reducers (nRed), cores and users. We chose a set of SQL queries that were translated into MapReduce jobs using Apache Hive [4]. They were executed over a dataset of several files ranging from 250 GBytes to 1 TByte that were used as external tables. The metric that we measured was the system response time. The SQL queries are identified by the name R1-R5 in the first column of the table. A description of the SQL sentences can be consulted in [1]. The last column of the table represents the percentage of relative error between the estimated and real response times. The response times obtained by simulation (column T swn [ms]) are close to the response times obtained by real executions (column T [ms]) in the Hadoop MapReduce cluster for most situations. The simulation tool offers good estimations of the real execution of Hadoop MapReduce systems. For all cases, the relative error is lower than 17.81%; and reaches a relative error of 0.03% in the best cases. The validation results are summarized in the following table.

Table 6: Validation of the DICE Simulation Tool for Apache Hadoop MapReduce

Query	Users	Cores	Scale (GB)	nMaps	nRed	T (ms)	T SWN (ms)	% Error
R3	1	80	1000	1560	1009	1019973	1020294,84	0.03
R2	3	40	250	4	4	86023	119712.3	-17.81

2.4.2 Apache Storm

The validation results for Storm has been already presented in Section 3 of DICE Deliverable 3.1 [2]. The results have been published in [5].

The experiments were executed in a cluster with two workstations. All the workstations were characterized by Intel(R) Core(TM) i7-6700 CPUs (3.40GHz) with 8 cores, 32GBytes of RAM, a Gigabit Ethernet and Ubuntu Linux OS (version 14.04). The target version was Storm 1.0. We studied the performance for various configurations of the Storm framework. That is, we analyzed the simulation errors for different number of cores, number of threads per bolt and thread execution time.

The Storm example consists of two spouts (data generators/file readers) and three bolts (data processing). The Storm application executes a word count. We considered the utilization of each bolt as performance metric, i.e., the percentage of time that the threads associated to a bolt are active and processing tuples (see columns %BX Cap). The simulation tool offers good estimations of the real execution of Storm systems. For all cases, the relative error is lower than 15.7%; and reaches a relative error of 0.068% in the best cases (see columns %BX Error). We have varied the execution time of the bolts and the arrival rate of messages to the spouts. The validation results are summarized in the following table.

Table 7: Validation of the DICE Simulation Tool for Apache Storm

Experiment	Cores	% B1 Cap	% B2 Cap	% B3 Cap	% B1 Error	% B2 Error	% B3 Error
R5	11	39.4	38.7	17	0.068	3.275	4.18
R10	11	9.5	11.9	7.3	15.738	0.958	12.618

2.4.3 Apache Spark

The validation results for Spark has been already presented in Appendix D of DICE Deliverable 3.4 [6].

In order to validate the Simulation tool, we have worked with several publicly available libraries and benchmarks for Spark. Concretely, we presented results from the SparkPrimes example, a toy example presented in DICE Deliverable 3.4 that computes a set of prime numbers; and the spark-perf suite developed by Databricks [7], a performance testing framework for Apache Spark 1.0+ that covers the more important parts of the ecosystem (Spark Core RDD, SQL + DataFrames and machine learning with MLlib).

In particular for the spark-perf suite, we have evaluated the performance of several Spark Core operations. We have focused on analysing different combinations of count and aggregation operations (e.g., mapping and count, filtering and count, mapping and grouping by integer keys, mapping and grouping by string keys, mapping and reducing, etc.). The aforementioned examples were run with different job configurations. Mainly, we have parameterized: a) the number of assigned cores to each job; and b) the number of tasks per operation. The execution environment where the Spark applications were launched consists of a Spark Master (coordinator) and a set of six workstations for running the computations. The workstations are virtual machines deployed on the flexiOPS FCO cloud [8]. All the workstations were characterised by virtual AMD CPUs (4x2.60GHz) with 4GBytes of RAM, a Gigabit ethernet and Ubuntu Linux (version 14.04) OS.

The validation results are summarized in the following table. It shows the error rate in the response times predicted by our tool for the SparkPrimes and the spark-perf suite with respect to the real executions.

Particular cases of SparkPrimes have high errors, for instance, 32.2%. Some cases in the spark-perf suite reached up to 26.094% of error. We suspected that these errors could be caused by the exponential distribution that simulates the mean execution time of the tasks. Therefore, we used an alternative probability distribution (Erlang distribution) in order to provide a better accuracy.

We repeated the experiments using the Erlang distribution instead of the exponential one. On average, the errors with the Erlang distribution were 2.27% smaller for the SparkPrimes example; and we got a reduction of up to 12.87% for the spark-perf suite. More in detail, we observed that the Erlang distribution fits better than the exponential one when there are a few number of parallel tasks per operation with respect to the total number of cores in the cluster (numTasks < 200). Conversely, the exponential distribution fits better than the Erlang distribution when there are a high number of parallel tasks per operation with respect to the total number of cores in the cluster (numTasks >= 400). The internal behaviour of the Spark application master is faded out when managing large number of tasks.

Table 8: Validation of the DICE Simulation Tool for Apache Spark

Experiment	Cores	numTasks	% SparkPrimes Error	% spark-perf Error
R7	18	200	32.2	26.094
R9	18	800	2.94	0.013

2.5 Optimization Tool

D-SPACE4Cloud is the the optimization tool developed within the DICE framework to support the capacity planning process of shared Hadoop Cloud clusters for MapReduce, Spark and Storm applications with quality of service guarantees. In a nutshell, the tool implements a search space exploration able to determine the optimal virtual machine (VM) type, possibly from different providers, and the optimal number of instance replicas.

The underlying optimization problem is demonstrated to be NP-hard and it is solved heuristically, whereas job execution times are estimated via queueing network (QN) or Stochastic Well Formed Net (SWN) models by relying on the DICE Simulation tool model to model transformations. D-SPACE4Cloud implements an optimization mechanism that efficiently explores the space of possible configurations.

During the third year, D-SPACE4Cloud was extended to support Spark applications, which are the core of the NETF case study. In the last six months, we improved the tool GUI (an Eclipse plugin) and developed a new component, which relies on a very fast ad-hoc simulator PMI developed within the EUBRA-BIGSEA project[8] and which is able to derive optimization models directly from Spark logs. In this deliverable we summarise our validation results for the capacity planning of Spark applications in public clouds. The validation of the tool for MapReduce applications has been reported in[9]. The results of the validation for private clouds are reported in the DICE Deliverable D3.9.

To validate D-SPACE4Cloud, we aimed at assessing the quality of the optimal solution obtained using D-SPACE4Cloud. Assuming to optimize a given DIA and a deadline to meet, we focus on the response time measured in a real cluster provisioned according to the number of VMs determined by the optimization procedure, quantifying the relative gap as a metric of the optimizer accuracy. Formally:

$$\textbf{Error-deadline \%} = (D - T_{real})/T_{real} \qquad \textbf{(1)}$$

where D is the deadline and T_{real} the execution time measured on the real cluster, so that possible misses would yield a negative result.

[8] http://www.eubra-bigsea.eu
[9] M. Ciavotta, E. Gianniti, D. Ardagna: D-SPACE4Cloud: A Design Tool for Big Data Applications. ICA3PP 2016: 614-629

The second error metric we considered is the cost prediction error evaluated as percentage difference of the minimum cost cluster configuration identified by inspection able to fulfill a deadline set a priori and the cost of the cluster identified by D-SPACE4Cloud. Formally:

$$\textbf{Error-cost \%} = (C_{cluster} - C_{D\text{-}SPACE4Cloud}) / C_{D\text{-}SPACE4Cloud} \qquad (2)$$

Note that, this latter is the target KPI for the D-SPACE4Cloud tool validation, whose median value should be below 30%.

D-SPACE4Cloud has been validated by considering the NETF case study, the TPC-DS industry benchmark by deploying the DIAs on Microsoft Azure platform. Moreover, thanks to the collaboration with the EUBRA-BIGSEA project, PMI validated its tool by considering the BULMA application developed within EUBRA-BIGSEA deployed on Docker containers.

2.5.1 TPC-DS Case Study

The TPC Benchmark DS (TPC-DS) is a decision support benchmark that models several generally applicable aspects of a decision support system, including queries and data maintenance. Although the underlying business model of TPC-DS is a retail product supplier, the database schema, data population, queries, data maintenance model and implementation rules have been designed to be broadly representative of modern decision support systems. TPC-DS run on RDBMS but also on many Big Data environments like Hadoop and Spark.

The supporting schema contains vital business information, such as customer, order, and product data. TPC-DS is based on snowflakes schema. It consists of multiple dimensions and fact tables. Each dimension has a single surrogate key. The fact tables join with dimensions using each dimension table surrogate key. The experimental settings were the same as in the NEFT case study. For the validation we considered two queries Q26 and Q52 at 500 GB scale and, we considered six cases ran on D12v2 VMs, varying the deadlines. The table below summarizes results.

Table 9: TPC-DS D-Space4Cloud error metrics

Query	VM type	Time real (ms)	Simulator prediction time (ms)	Error-deadline %	Cluster real (nCores)	Cluster DS4C (nCores)	Error-cost %
Q26	D12v2	186,659	158, 287	15%	48	52	-8%
Q52	D12v2	188,860	150, 488	20%	48	52	-8%
Q26	D12v2	304,048	186, 066	39%	32	36	-11%
Q52	D12v2	263,034	175, 394	33%	32	36	-11%
Q26	D12v2	454,054	332,364	27%	20	24	-17%
Q52	D12v2	410,588	353,852	14%	20	24	-17%

First of all, none of the considered runs led to a deadline miss. Moreover the relative error is always below 20%, with a worst case result of 17% and the average settling at 12%. The tool is conservative and always allocates one additional VM. Overall we can conclude that the optimization tool is effective in identifying the minimum cost solution at design time, guaranteeing that deadlines are met as well.

2.5.2 BULMA Case Study

Thanks to the involvement of PMI in the EUBRA-BIGSEA project, DICE tools have been validated also on Docker container based deployments by considering the project BULMA application. The goal of BULMA is to provide high-quality integrated geospatial-temporal training data to support predictive machine learning algorithms of Intelligent Transport Systems applications and services.The problem faced by BULMA is the following. The task of identifying bus trajectories from the sequences of noisy geospatial-temporal data sources is known as a map-matching problem. It consists of performing the linkage between the bus GPS trajectories and their corresponding road segments (i.e., predefined trajectory or shapes) on a digital map. BULMA is a novel unsupervised technique capable of matching a bus trajectory with the *correct* shape, considering the cases in which there are multiple shapes for the same route (common cases in many Brazil cities, e.g., Curitiba and São Paulo). Furthermore, BULMA is able to detect bus trajectory deviations and mark them in its output. BULMA application has been run on Spark 1.6, where each executor was allocated to a Docker container with 2 cores and 4GB of memory. The data set considered includes the data of five days of bus GPS data gathered in Curitiba. Overall 4 different configurations up to 48 cores were considered and results are summarized in **Table 10**. Even if in this case one scenario (the one at 48 cores) led to a deadline miss, the accuracy of the simulation tools is in line with the previous analysed scenarios based on virtualized systems. The average absolute percentage cost error is 16% below the 30% median error envisioned for the target KPI.

Table 10: BULMA D-Space4Cloud error metrics

Simulator prediction time (ms)	Time real (ms)	Error-deadline %	Cluster real (nCores)	Cluster DS4C (nCores)	Error-cost %
723,876,859	724,858,500	15%	36	40	-10%
644,487,201	693,636,333	-20%	48	46	4%
1,039,256,814	1,126,044,750	39%	12	16	-25%
781,989,438	797,890,000	33%	24	32	-25%

2.6 Monitoring Tool

The DICE monitoring platform (DMon) is designed to be a scalable system with minimal collection overhead that is able to collect performance monitoring data from big data services underlying most data intensive applications. It currently supports: YARN, Spark, Storm, MongoDB, Cassandra, System metrics etc.

DMon has several core features meant to ease the workload of the end user. For example, DMon is able to automatically detect Storm topologies as well as some basic Spark and YARN services which aids in the setup process. It is also capable of automatically generating basic visualizations for all big data frameworks.

At M30 the monitoring tool is feature complete: monitoring metrics for all DICE supported technologies are collected, metric processing overhead is of loaded to DMon significantly reducing collection overhead, deployment is done automatically via blueprints by the DICE deployment service. Since M30 most of the work done was for performance improvements and minor bug fixes. As part of T.4.2 during the development of the Anomaly detection tool we have added an artefact repository to DMon. This repository is designed to hold all predictive models trained by the anomaly detection tool. This feature has been optimized in this period.

On of the main optimizations done in this period was applied to the WSGI (Web Server Gateway Interface) used by DMon. This was done to ensure the best possible availability of monitoring data. As we can see in **Figure 7** we have benchmarked a number of solutions such as; Tornado, Bjoern and Gunicorn using the Locust load testing tool. As a baseline we have also included the standard WSGI called Werkzeug.

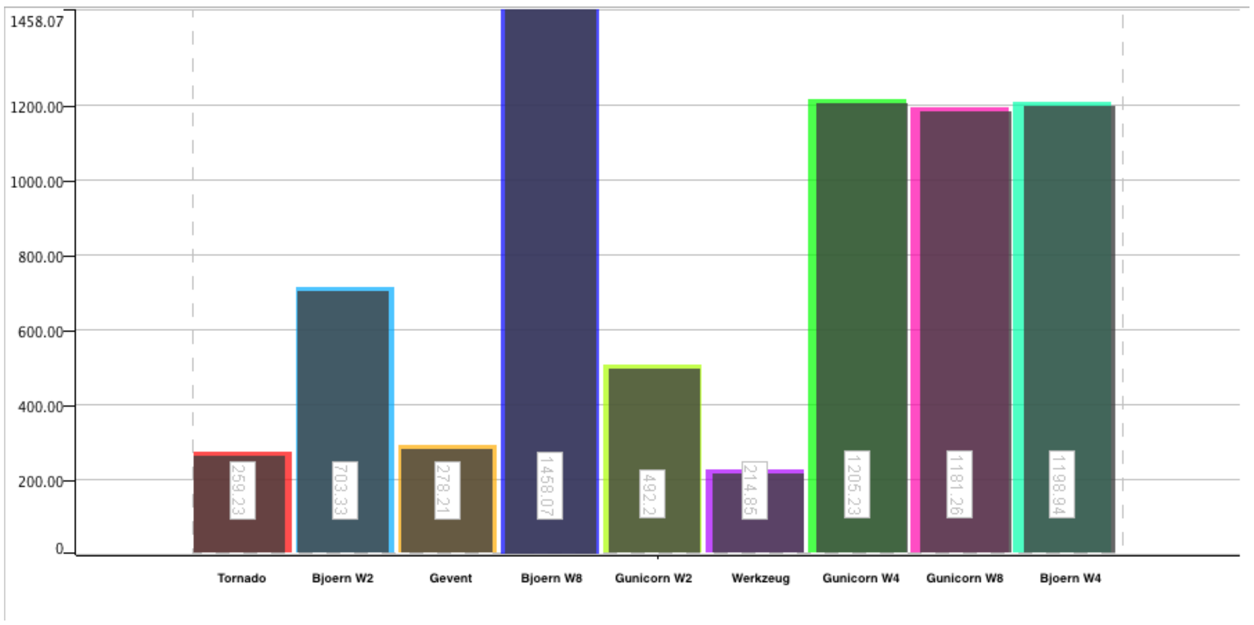


Figure 7: Benchmark with various WSGI setups for DMon

Another important modification was made to DMon which allows the monitoring of different versions of some of the Big Data frameworks supported by the DICE methodology. Of particular interest is Spark which has different ways of exporting performance metrics in versions 1.x than 2.x.

The monitoring tool fulfils all KPIs by virtue of; its automatic deployment by the Deployment service thus considerably reducing the time it takes to configure and the processing of monitoring data inside DMon all collection components (such as the *dmon-agent*) are tasked only with forwarding the metrics to the DMon controller.

2.7 Anomaly Detection Tool

The anomaly detection tool (ADT) is designed to detect contextual anomalies in performance metrics collected during DIA execution. It allows developers and designers to see how their application performs and notifies them in the case an anomaly occurs. In contrast to other solutions based on rules which mainly deal with a limited number of metrics (features) ADT can use the entire spectrum of features available during DIA execution.

It is designed to form a lambda type architecture together with the monitoring platform. DMon is the serving layer where all monitoring data is stored and queried as well as the location where trained models are stored. ADT has two modes of operation. First it has a training mode where models are trained and validated. The second mode is the prediction mode where trained models are instantiated and are used for detecting. These two modes represent the batch and speed layer respectively from a lambda architecture.

ADT was also used on a new use case dealing with fraud detection in credit card transactions. In this use case the data available is very imbalanced having an extremely low anomaly occurrence (well under 0.0005%). It is important to mention that the available data was synthetically generated based on real transaction features.

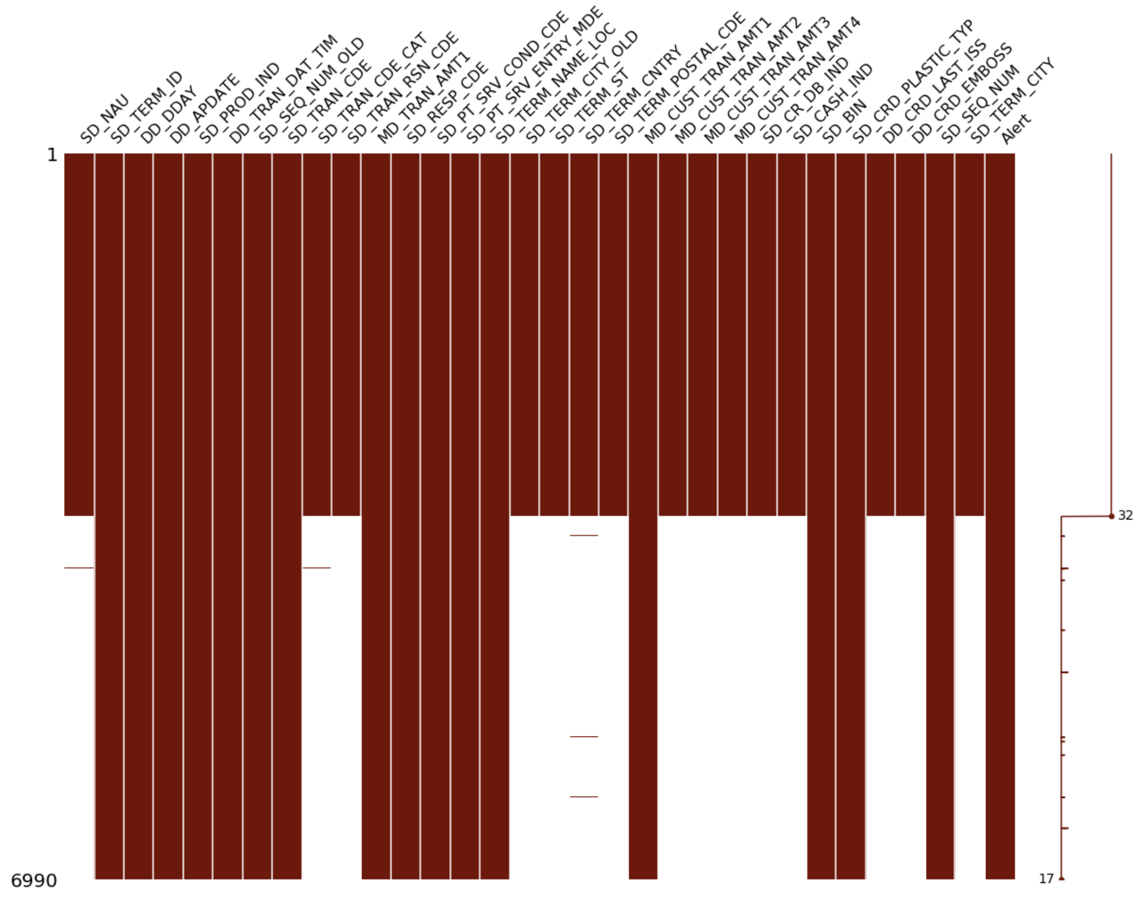


Figure 8: Dataset features and missing values

In the case of machine learning libraries such as XGBoost missing values are dealt with automatically by the underlying algorithm, it is able to learn which is the best imputation values. However, other methods are susceptible to these missing values in the data. This fact has lead us to include features in ADT which signal if missing values are detected. We can see from **Figure 8** some of the selected features for anomaly detection. Some of the features had missing values which is easily seen from figure **Figure 8**. From **Figure 9** we can see the ADT calculated Pearson correlations generated during the initial analysis of the available dataset.

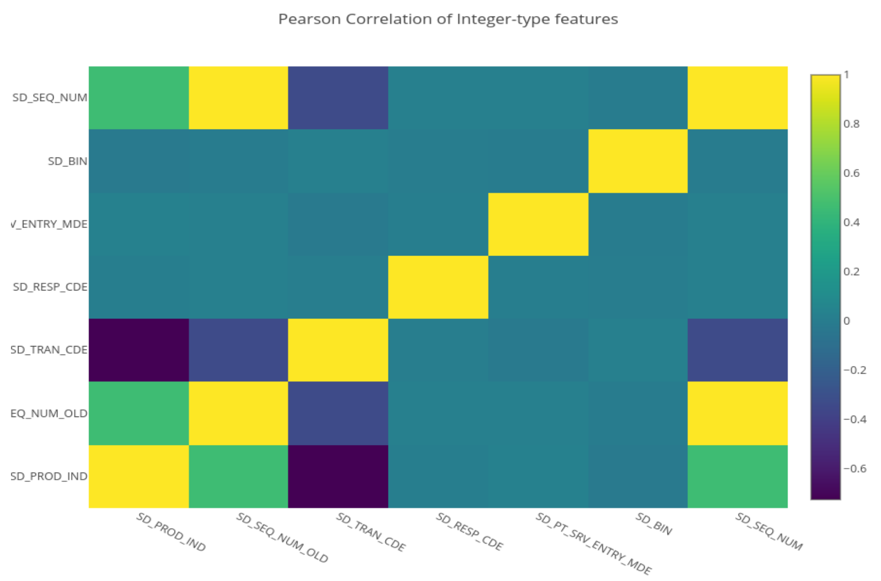


Figure 9: Pearson correlation for floating point and integer features

All supervised and unsupervised methods have been tested on this dataset their average performance is around 6% to 8% false positives. **Figure 10** shows a partial decision tree trained using ADT while **Figure 11** shows the training vs cross validation graph.

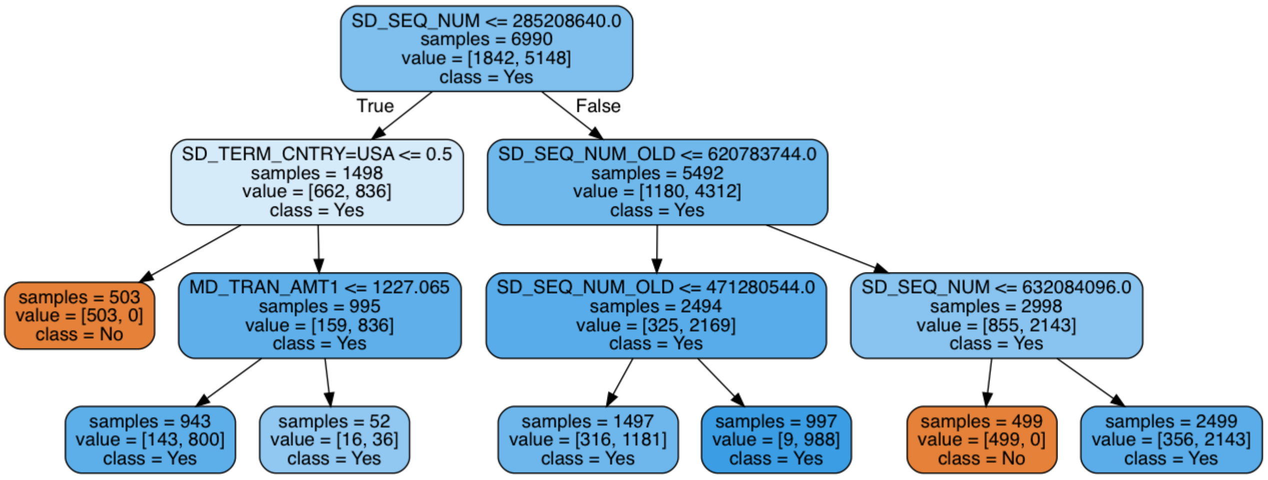


Figure 10: ADT created Decision Tree example

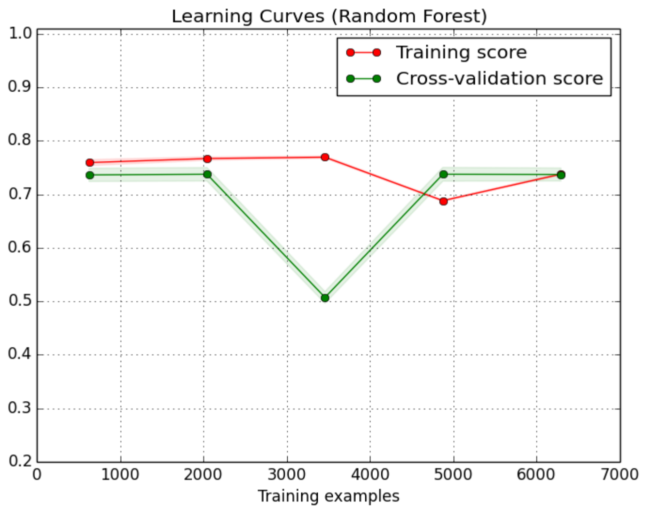


Figure 11: Training vs validation score for ADT Random Forest

This dataset has proven to be of significant challenge as most methods implemented in ADT where until this point tested on datasets which had a maximum of 50000 to 1 ratio with regard to anomalies while in this scenario the ratio is much closer to 1 million to 1. Because of this usage of pre-processing methods for unbalanced dataset such as SMOTE and ADASYN is of great interest. Preliminary integration of these methods have yielded very promising results ADT being able to detect anomalies detecting 7.8% false positives.

2.8 Trace checking Tool

DICE-TraCT is the DICE tool which enables the trace checking analysis of DIA logs. Trace checking is an approach for the analysis of system executions that are recorded as sequences of timestamped events to establish whether the system logs satisfy a property, usually specified in a logical language. Adding to the above, trace checking is a way to check the correctness of the ordering of the events occurring in the system and of the time delays between pairs of events. Moreover, it is useful when the aggregated data that are available from the monitoring system are not enough to conclude the correctness of the system executions with respect to some specific criteria. Additionally, trace checking is a possible technique to achieve this goal and can be used on purpose to extract information from the executions of a running application.

According to the DICE methodology, trace checking is performed after verification to allow for continuous model refinement.

Since M30, DICE-TraCT underwent various validation and refinement activities that were carried out by using realistic applications such as the well-known Wikistats and also NewsAsset, the use case scenario owned by ATC. The complete DICE-TraCT framework based on a client-server structure was tested under the Wikistats application through the analysis of the logs extracted by the monitoring platform. The implementation of the IDE concluded the activity of the DICE-TraCT tool. The next figure shows the result of the analysis of the Wikistats topology that is sent to DICE-TraCT IDE by the DICE-TraCT engine.

DICE-TraCT allows the users to implement specific monitoring solutions for the collection of information through the analysis of the application logs. In particular, DICE-TraCT enables the analysis of the logs to elicit the parameter values, related to certain specific features of the running application, that are necessary for the verification carried out by D-VerT. In our scenario, those parameters cannot be directly extracted from the monitoring platform of the framework used to implement the DIA, e.g., Storm. For instance, the parameter *sigma*, that characterizes the abstraction of the bolt functionality, can be calculated by means of the analysis of the logs with the trace-checking technique implemented by DICE-TraCT, whereas an estimation via the Storm monitoring service turns out to be unfeasible.

DICE-TraCT easily fulfils the requirements on the KPIs as it can extract all the parameter values necessary for the verification with D-VerT, for every component of the topology. DICE-TraCT, in fact, can provide the average emit rate of the all the spouts and both sigma and the average time required by the bolts for elaborating their inputs, for all the bolts in the topology.

2.9 Enhancement Tool

The DICE Enhancement tool is designed for iteratively enhancing the DIAs quality. Enhancement tool aims at providing a performance and reliability analysis of big data applications, updating UML models with analysis results, and proposing a refactoring of the design, if performance anti-patterns are detected.

At the end of M30, the Enhancement tool reached a phase that was feature complete:

- **DICE-FG:** Integrated a novel estimation algorithm for *hostDemand*, called est-le, that outperforms several state-of-the-art algorithms
- **DICE-APR**: Developed and improved two submodules, Tulsa and Anti-Pattern Detection and Refactoring (APDR).
    - **Tulsa**, a M2M transformation tool, transforms the design time model (i.e, UML model), which is annotated with runtime performance quality characteristics by DICE FG tool, into performance model, Layered Queueing Network (LQN) model. Tulsa has a standalone version which executes a series of transformation tasks specified in an Ant build file. A specific launch configuration can be invoked from the IDE run-configuration panel. The run-configuration in question invokes model-to-model transformation that parses the UML diagrams and returns a LQN model for performance anti-pattern detection.
    - **APDR** also has a standalone version. It detects the pre-defined performance anti-patterns (i.e., Infinite Wait and Excessive Calculation), formally defined by using Matlab scripts, of DIAs and provides refactoring suggestions to the designer. It supports Big Data technologies (e.g. Storm).

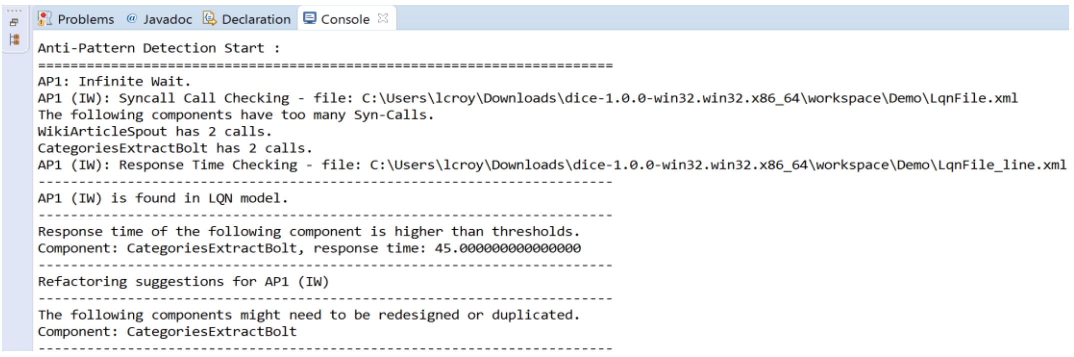
- **Enhancement Tool Plug-in**: DICE Enhancement tool (i.e., DICE-FG and DICE-APR) has been integrated in the DICE IDE as a plug-in by M30. It can be invoked through the Pop-up menu.

2.9.1 Lab validation

The Enhancement tools have been validated on the ATC and NETF studies and details can be found in the corresponding use case sections later in this deliverable. Additionally, we here report additionaly lab validation activities carried out in YR3.

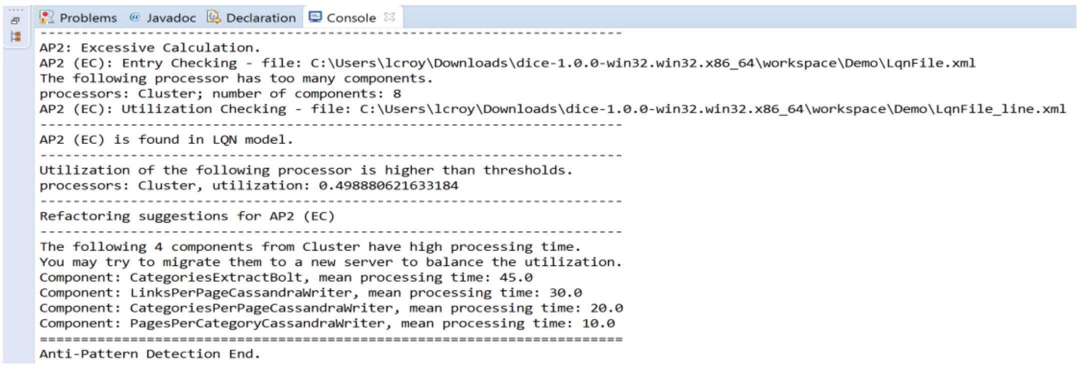
We modified a Storm application, WikiStats, to test our approach. The application processes and analyses web pages from the popular Wikimedia website in order to extract some statistics that are then stored into a database. The test environment is based on DICE IDE 1.0.0 and MATLAB Compiler Runtime (MCR) R2015a. The WikiStats application is composed of 8 components. We assume that the threshold on the number of components, which can be deployed on a server, is 2. The upper bound of the CPU utilization is 0.4. The maximum number of calls is 1 and the upper bound of response time is 4.0 sec.

The problem faced by WikiStats is the following. The task of providing a refactoring of the design if the performance anti-patterns (i.e, Infinite Wait and Excessive Calculation) are found in the WikiStats application. It consists of modelling, analysis, extraction, detection and refactoring. 1) Modelling. Two models (i.e., architecture model and performance model) need to be generated for the later processing. The user of DICE IDE builds the design time model in the form of UML by using Papyrus editor in the DICE IDE. The UML model of WikiStats is annotated with core tags (e.g., *hostdemand*) of stereotypes of MARTE and DICE profiles. Enhancement tool plugin helps to generate performance model (i.e., LQN) according to the corresponding UML model. 2) Analysis. Both the solved LQN model and the performance analysis results can be obtained with the help of LINE solver. The analysis results of CPU utilization of the cluster is 0.4989. 3) Extraction. Extracting the constraints (i.e., performance upper bound) to be used in the deployment refactoring. 4) Detection and refactoring. As results of the Excessive Calculation detection and refactoring, the threshold on the number of components is 2, the current deployment violates the bound. Furthermore, the utilization 0.4989 is greater than the threshold on CPU utilization (0.4). Thus the Excessive Calculation anti-pattern is found. As results of the Infinite Wait detection and refactoring, there are 2 components that have more than one function call but only one of them has the response time that is greater than the threshold (4.0 sec). Thus, the refactoring suggestions for those two anti-patterns are shown in **Figure 12** and Figure **13**, validating the ability of the tool to identify anti-patterns and provide suggestions.



```
Anti-Pattern Detection Start :
======================================================================
AP1: Infinite Wait.
AP1 (IW): Syncall Call Checking - file: C:\Users\lcroy\Downloads\dice-1.0.0-win32.win32.x86_64\workspace\Demo\LqnFile.xml
The following components have too many Syn-Calls.
WikiArticleSpout has 2 calls.
CategoriesExtractBolt has 2 calls.
AP1 (IW): Response Time Checking - file: C:\Users\lcroy\Downloads\dice-1.0.0-win32.win32.x86_64\workspace\Demo\LqnFile_line.xml
----------------------------------------------------------------------
AP1 (IW) is found in LQN model.
----------------------------------------------------------------------
Response time of the following component is higher than thresholds.
Component: CategoriesExtractBolt, response time: 45.000000000000000
----------------------------------------------------------------------
Refactoring suggestions for AP1 (IW)
----------------------------------------------------------------------
The following components might need to be redesigned or duplicated.
Component: CategoriesExtractBolt
----------------------------------------------------------------------
```

Figure 12: Infinite Wait detection and refactoring suggestion



```
-----------------------------------------------------------------
AP2: Excessive Calculation.
AP2 (EC): Entry Checking - file: C:\Users\lcroy\Downloads\dice-1.0.0-win32.win32.x86_64\workspace\Demo\LqnFile.xml
The following processor has too many components.
processors: Cluster; number of components: 8
AP2 (EC): Utilization Checking - file: C:\Users\lcroy\Downloads\dice-1.0.0-win32.win32.x86_64\workspace\Demo\LqnFile_line.xml
-----------------------------------------------------------------
AP2 (EC) is found in LQN model.
-----------------------------------------------------------------
Utilization of the following processor is higher than thresholds.
processors: Cluster, utilization: 0.498880621633184
-----------------------------------------------------------------
Refactoring suggestions for AP2 (EC)
-----------------------------------------------------------------
The following 4 components from Cluster have high processing time.
You may try to migrate them to a new server to balance the utilization.
Component: CategoriesExtractBolt, mean processing time: 45.0
Component: LinksPerPageCassandraWriter, mean processing time: 30.0
Component: CategoriesPerPageCassandraWriter, mean processing time: 20.0
Component: PagesPerCategoryCassandraWriter, mean processing time: 10.0
=================================================================
Anti-Pattern Detection End.
```

Figure 13: Excessive Calculation detection and refactoring suggestions

2.10 Quality testing Tool

The quality testing tool (QT) offers a Java library (QT-LIB) and a workload generation tool (QT-GEN) for stress testing data-intensive applications. The tool supports Apache Storm and Apache Kafka, and through the latter it can deliver data pipelines to several other platforms including Apache Spark. QT-GEN allows to generate a workload to be injected in the application, either by random number generation or through the fitting and sampling of a class of hidden Markov models on existing workload traces for the system. The latter is particularly useful to create new tests in systems that process commercial data, which would be expensive to buy, such as Twitter datasets. QT-LIB enacts the test by reading the workload produced by QT-GEN and sending JSON tuples on output streams at the frequency requested by the end user.

QT has been presented in deliverables D5.4 (Storm) and D5.5 (Kafka and Spark). On top of the experiments reported there, we have performed additional validations of the QT-LIB tool to establish its ability to harness the physical resources of the load testing machines. It was decided to focus on further validation of QT-LIB since QT-GEN has been the subject of a journal publication, see D5.4, therefore its maturity was deemed largely sufficient for practical use by third parties.

Typically, in stress testing exercises the machine that hosts the workload generator soon becomes a bottleneck, as the volume of data and outgoing requests grows. This is expected as the larger the host utilization, the largest the parallelism of the testing tool. It is however a problem when the testing tool is not able to saturate the host or the target, as this typically points to a software bottleneck due to incorrect programming of the tool or misconfiguration of the underpinning platforms. To ensure that QT was not affected by this issue, we carried out new experiments to verify the scalability of the library.

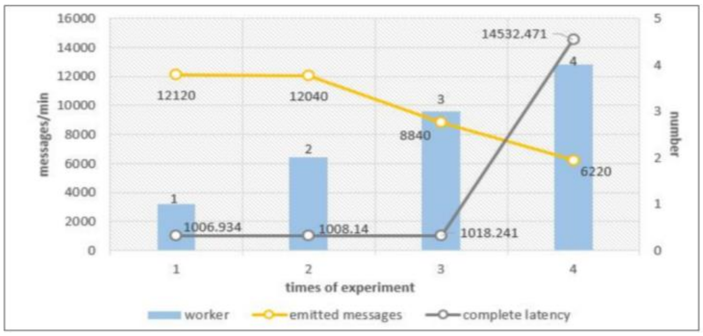
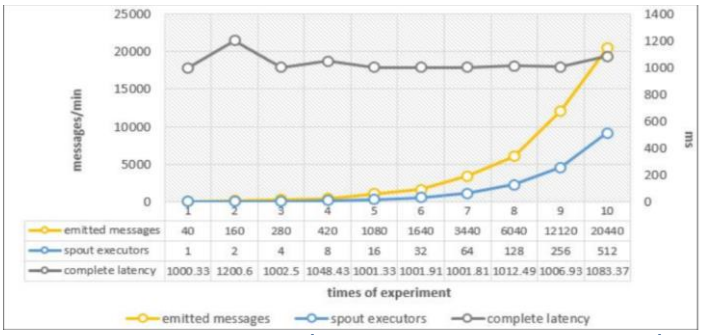


Figure 14: Further validation of QT-LIB load injector: saturation of target

Figure 14 illustrates one of the experiments we have carried out to establish the ability of QT-LIB to scale. We have run a basic Storm setup on a two-core machine, and injected load from QT-LIB. We increased the volume of messages emitted by QT-LIB and we verified that this was able to correctly generate the load on the target machine up to its saturation. In the figure, we see that the number of emitted messages per worker on Storm remains constant around 24,000 as the number of workers exceeds 2 (note that the values reported in the figure are per worker), suggesting that the Storm platform has indeed saturated.

We have then carried out extra experiments to establish the scalability of the QT-LIB framework on the host, which extend the ones presented in D5.4. **Figure 15** reports the results of these experiments. As we see, by increasing the parallelism of the QT-LIB injector in terms of number of spouts and emitted messages during the observation period, the system continues to produce correct results even with a parallelism of 512 spouts, which represents a rather large scenario for a small testbed like the one considered in our lab tests. This further confirms the absence of software bottlenecks in the QT tool.



	1	2	3	4	5	6	7	8	9	10
emitted messages	40	160	280	420	1080	1640	3440	6040	12120	20440
spout executors	1	2	4	8	16	32	64	128	256	512
complete latency	1000.33	1200.6	1002.5	1048.43	1001.33	1001.91	1001.81	1012.49	1006.93	1083.37

Figure 15: Further validation of QT-LIB load injector: saturation of host

2.11 Configuration optimization Tool

CO focuses on optimizing data-intensive application configurations based on a technique known as Bayesian Optimization, which is a machine learning methodology for black-box global optimization. In this methodology, the unknown response of a system to a new configuration is modelled using a Gaussian process (GP), an important class of machine learning models. GPs are used to predict the response of the platform to changes in configuration. GPs can take into account mean and confidence intervals associated with measurements, predict system behavior in unexplored configurations, and can be re-trained quickly to accommodate for new data. Experimental results indicate that Bayesian optimization method can be much faster on real systems than existing auto-tuning techniques. CO was first developed in deliverables D5.1 and later refined with transfer learning methods in deliverable D5.2, which bootstrap the method from an existing set of GPs based on an earlier release of the application. Later, in D5.3 we have integrated CO in the DICE IDE, allowing the user to specify configuration parameters within given ranges directly from the GUI.

CO has been extensively validated on various types of Big Data workloads and platforms, in particular Storm and Cassandra. In addition to the material included in D5.1 and D5.2, experiments have also been carried out to validate the performance of the method in relation to other blackbox methods and across multiple dimensions.

For example, **Figure 16** shows a comparison of the CO algorithm based on transfer learning, named TL4CO, with tree-based regression and polynomial fitting methods. Lower error values indicate solutions closer to the global optimum. The figure, overall, indicates that the CO methods are able to optimally configure a data-intensive application, in this case a Storm-based system, than existing methods in the state of the art.

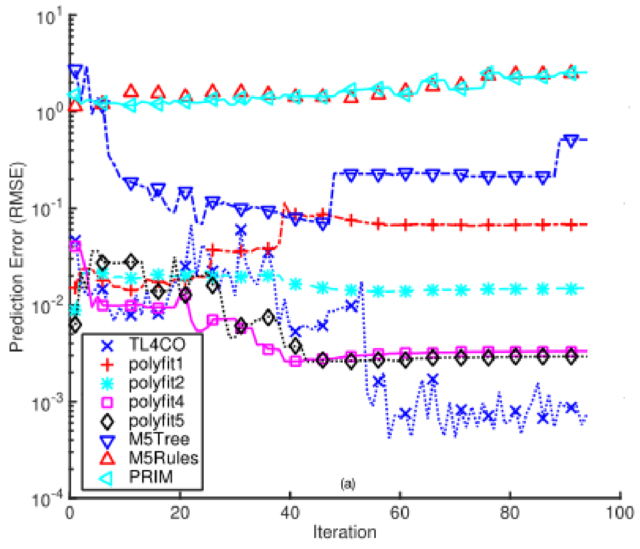


Figure 16: Further validation of configuration optimization tool

2.12 Fault injection Tool

The FIT allows users to generate faults on their Virtual Machines, giving them a means to test the resiliency of their installation. Using this approach the designers can use robust testing, highlighting vulnerable areas to inspect before it reaches a commercial environment. Users or application owners can test and understand their application design or deployment in the event of a cloud failure or outage, thus allowing for the mitigation of risk in advance of a cloud based deployment.

To validate the impact of FIT usage the common task manager program 'top' found in many Unix-like operating systems was used. Using the Linux 'top' command on the target VM the current state of its resources can then be seen.

Regarding CPU saturation, before running the CPU overload fault, the %Cpu usage is measured. While running the CPU overload the %Cpu quickly rises to near 100%. The stress command issued by the FIT would typically account for around 99.2% of the available CPU capacity.

Regarding memory saturation, the FIT 'stressmem' feature is called with designated parameters. The tool first connects via SSH to the VM and determined the OS version by checking the /etc/*-release for the version of the OS (Ubuntu in our case). It then looks for the memory stress tool suitable for Ubuntu, for example Memtester. If the tool is not found first the DICE FIT installs the tool along with dependencies. Finally, the FIT calls Memtester to saturate memory in the target node. Again, a standard monitoring tool such as 'top' (in the %MEM column) will show the 2GB (or whatever was specified in the memory size parameter of stressmem) RAM available to the VM being saturated.

The use of standard measurement tools already bundled with the OS means it is then easy for use cases to ensure the injected fault is having the desired effect.

2.13 DICE IDE

The pivotal tool of the project is the DICE IDE. It integrates the execution of the different DICE tools and it gives support to a new MDE methodology. The IDE is an integrated development environment tool for MDE where a designer can create models to describe data-intensive applications and their underpinning technology stack.

The IDE offers the ability to specify DIAs through UML models. From these models, the toolchain guides the developer through the different phases of quality analysis, formal verification being one of them.

The IDE is based on Eclipse Neon 4.6, which is the de-facto standard for the creation of software engineering models based on the MDE approach. The Eclipse IDE has been customized with suitable plug-ins that integrate the execution of the different tools, in order to minimize learning curves and simplify adoption. Not all tools are integrated in the same way. Several integration patterns, focusing on the Eclipse plugin architecture, have been defined. They allow the implementation and incorporation of application features very quickly. Moreover, creating custom versions of DIA applications are easier and without source code modifications need. DICE Tools are accessible through the DICE Tools menu.

2.13.1 IDE Global Architecture

The overall goal of the IDE is to become the main access gateway for all designers and developers willing to adopt and follow the proposed methodology for building DIA applications.

The IDE guides the developer through the methodology, based on tools' Cheat Sheets. It initially offers the ability to specify the data-intensive application through UML models stereotyped with performance profiles. From these models, the tool-chain guides the developer through the different phases of quality analysis (e.g., simulation and/or formal verification), deployment, testing, and acquisition of feedback data through monitoring data collection and successive data warehousing. Based on runtime data, an iterative quality enhancements tool-chain detects quality incidents and design anti-patterns. Feedbacks are then used to guide the developer through cycles of iterative quality enhancements.

2.13.2 Conclusions

Improvements provided by the use of the IDE in order to develop data-intensive applications are: (1) User-friendly IDE, (2) Support for most of the phases of software development cycle, (3) Plug-In Updates, (4) Integration with other quality tools, (5) IDE customizations, (6) Access to remote repositories, and (7) Advanced UML modelling.

The last version (v1.0.1) was published on January 11, 2018 and it is a minor revision of the first release version (v1.0.0) of the DICE IDE and includes the last version of the DICE tools. DICE IDE allows to update the different DICE tools through the option "check for updates" of the help menu.

The IDE has been used for the validation of all the DICE tools, since the access and interaction with the other tools are done through the IDE, moreover the IDE has been used in the development of the three industrial DIA use cases explained in this deliverable.

Internally, for each new version of the IDE a set of validations has been made to ensure that the integration with the other tools works properly. These validations consist in review the cheat sheets, configuration properties, menu entries and update site for each integrated tool.

3 Deployment and Impact of Tools in Use Cases Environment

All three demonstrators have been using and validating DICE tools in their development and business environment, throughout the whole project lifetime. All tools have been validated by at least one use case, while a large number of tools have been validated in more than one use case. Part of the material presented in this section is a summary of material already presented extensively in previous confidential deliverables D6.1 to D6.3, and we find necessary to comment on them in order to provide a complete overview to the reader.

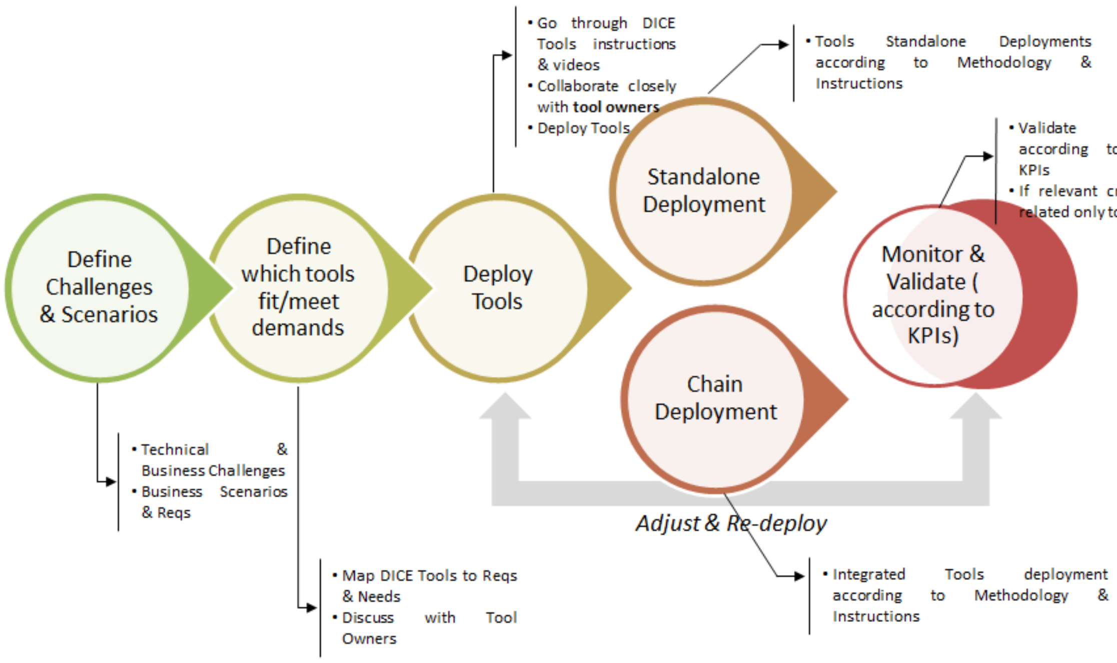

Define Challenges & Scenarios
Define which tools fit/meet demands
Deploy Tools
Standalone Deployment
Chain Deployment
Monitor & Validate (according to KPIs)
Adjust & Re-deploy
• Technical & Business Challenges
• Business Scenarios & Reqs
• Map DICE Tools to Reqs & Needs
• Discuss with Tool Owners
• Go through DICE Tools instructions & videos
• Collaborate closely with tool owners
• Deploy Tools
• Tools Standalone Deployments according to Methodology & Instructions
• Integrated Tools deployment according to Methodology & Instructions
• Validate Tools according to generic KPIs
• If relevant create KPIs related only to the UC

Figure 17: DICE tools deployment methodology

3.1 ATC Use Case

We are currently experiencing a social web explosion, which is giving the power of speech back to the citizens who had been practically deprived of this since the gradual explosion of the population that made it impossible for news to travel via the old channel of the ‘word-of-mouth’. We now live through the new phenomenon of the ‘e-word-of-mouth’, which is travelling in rapid paces and huge volumes through tweets, posts and blogs. This could be an opportunity for direct access to information coming from first hand sources, as it was happening centuries ago in small societies. The problem is that the scales are now very much different and all the sides have their say in this big bang of gossiping: truth and lies, positive and negative, genuine and fake. Verifying content in an easy, transparent and fast way is becoming more and more relevant, especially when taking into consideration (a) the sheer quantity of content found in Social Networks, and (b) the fact that a lot of content consists of hoaxes, rumours or deliberately misleading information (e.g. propaganda, fake news, spin). ATC has used DICE tools to develop the Trend Topic Detector Module of TruthNest, a unique web platform which can be used for assisting the end user to verify the validity of a post coming from the social media.

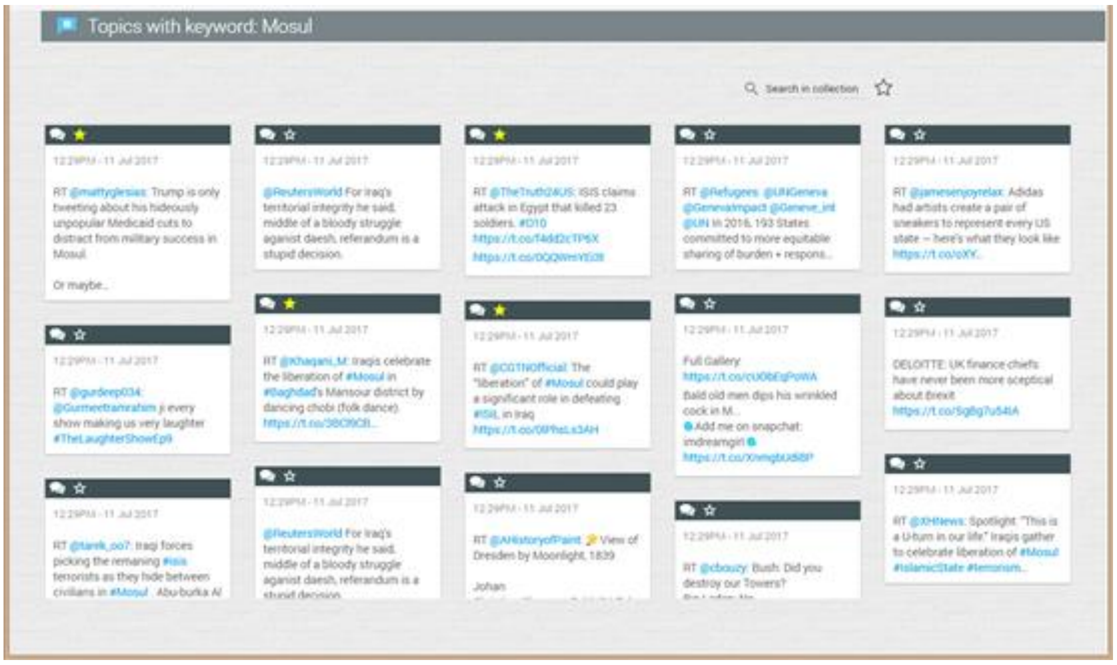
3.1.1 Use Case Scenarios

TruthNest is an online comprehensive tool that can promptly and accurately discover, analyse, and verify the credibility and truthfulness of reported events, news and multimedia content that emerge in social media in near real time (see **Figure 18**). The end user has the ability to verify the credibility of a single post within seconds by activating, with a single click, a series of analysis events for achieving the desired result.

More specific, TruthNest users will be able to bring in streams from social networks which will then be able to analyse and gain insights as to several dimensions of the verification process. In addition, they will also be able to create and monitor new "smart" streams from within TruthNest.

Figure 18: TruthNest Screenshot

An important module for TruthNest - which has been developed from scratch is the "Trend Topic Detector". The "Trend Topic Detector" provides to the end user a visualisation environment showing prominent trends that derive from social media, and, more specifically, from Twitter. What is critical to mention at this stage, is that only the "Trend Topic Detector" module has been developed by using DICE tools while the rest of TruthNest's components have been developed by using conventional tools and methodologies as these have been used by ATC's engineering and development team.



Topics with keyword: Mosul

Search in collection

12:29PM - 11 Jul 2017
RT @mattyglesias: Trump is only tweeting about his hideously unpopular Medicaid cuts to distract from military success in Mosul.

Or maybe...

12:29PM - 11 Jul 2017
RT @gurdeep034: @Gurmeetramrahim ji every show making us very laughter #TheLaughterShowEp9

12:29PM - 11 Jul 2017
RT @tarek_oo7: Iraqi forces picking the remaning #isis terrorists as they hide between civilians in #Mosul . Abu-burka Al

12:29PM - 11 Jul 2017
@ReutersWorld For Iraq's territorial integrity he said. middle of a bloody struggle aganist daesh. referandum is a stupid decision.

12:29PM - 11 Jul 2017
RT @Khaqani_M: Iraqis celebrate the liberation of #Mosul in #Baghdad's Mansour district by dancing chobi (folk dance). https://t.co/38O9CB...

12:29PM - 11 Jul 2017
@ReutersWorld For Iraq's territorial integrity he said. middle of a bloody struggle aganist daesh. referandum is a stupid decision.

12:29PM - 11 Jul 2017
RT @TheTruth24US: ISIS claims attack in Egypt that killed 23 soldiers. #D10 https://t.co/f4dd2cTP6X https://t.co/DQQWmYE08

12:29PM - 11 Jul 2017
RT @CGTNOfficial: The "liberation" of #Mosul could play a significant role in defeating #ISIL in Iraq https://t.co/08PheLs3AH

12:29PM - 11 Jul 2017
RT @AHistoryofPaint: View of Dresden by Moonlight, 1839

Johan

12:29PM - 11 Jul 2017
RT @Refugees: @UNGeneva @GenevaImpact @Geneve_int @UN In 2016, 193 States committed to more equitable sharing of burden + respons...

12:29PM - 11 Jul 2017
Full Gallery https://t.co/cU0bEgPoWA
Bald old men dips his wrinkled cock in M...
Add me on snapchat: imdreamgirl
https://t.co/XnmgbUd8P

12:29PM - 11 Jul 2017
RT @cbouzy: Bush: Did you destroy our Towers?

12:29PM - 11 Jul 2017
RT @jamesenjoyrelax: Adidas had artists create a pair of sneakers to represent every US state – here's what they look like https://t.co/oXY...

12:29PM - 11 Jul 2017
DELOITTE: UK finance chiefs have never been more sceptical about Brexit https://t.co/Sg8g7u54tA

12:29PM - 11 Jul 2017
RT @XHNews: Spotlight: "This is a U-turn in our life." Iraqis gather to celebrate liberation of #Mosul #IslamicState #terrorism...

Figure 19: Trend Topic detector

3.1.1.1 Trend Topic Detector

3.1.1.1.1 Description

The Trend Topic Detector is the heart of our News Orchestrator application and is centered around a clustering module. This creates clusters of tweets that relate to the search criteria submitted. The clusters are formulated by grouping the tweets found based on their common terms. The module tracks a percentage of the tweets posted onwards on Twitter, as the Twitter streaming API limitations impose. While it is restricted currently on Twitter stream API, it can take input from multiple social media (YouTube, Flickr and others) however it has not been implemented yet.

3.1.1.1.2 Architecture & Topology

The main pipeline of the clustering module is implemented as a Storm topology, where sequential bolts perform specific operations on the crawling procedure. These bolts include entity extraction (by using Stanford NER classifier) and minHash computation to estimate how similar the formulated sets are. The tweets terms are extracted and indexed in a running Solr instance. The most frequent terms are computed in a rolling window of 5 minutes and 20 clusters are formulated by default. A label (typically the text of a tweet) is assigned to each cluster. The results are stored in a Mongo database. The module is highly configurable and offers nearly real time computation of clusters.

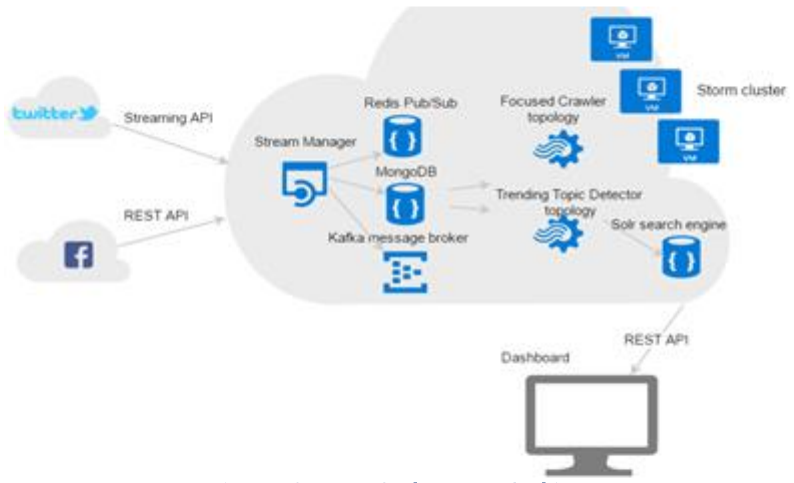


Figure 20: News Orchestrator Architecture

The main purpose of topic-detector part of focused crawler is the extraction of trending topics contained in items shared through social networks. By trending topics, we refer to frequent features (n-grams, named entities, hashtags) that exhibit an abnormal increase on the current timeslot compared to the previous ones.

The main pipeline of topic-detector is implemented as an Apache Storm topology, where the sequential bolts perform a specific operation on the detection procedure. The overall topology is depicted in the following figure.

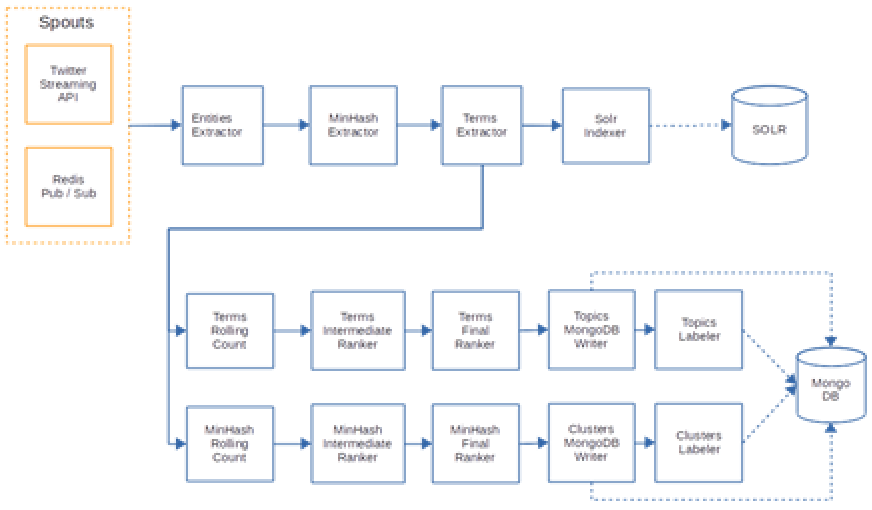


Figure 21: Trend Topic Detector Topology

The input stream consists of items: these refer to tweets published in Twitter. There are two spouts that are possible to inject items in the topology: a) one that listens to a Redis message broker following the Publish/Subscribe pattern, and b) one waiting for incoming items from Twitter's Streaming API. The items fed to the detector through Redis may be produced by any independent process. One possibility is to use the Stream Manager project.

The first bolt in the topology extracts named entities from the text of the messages injected in the topology by the spouts. For this procedure, Stanford NER 2extractor is used. The next bolt extracts a signature from the text based on the MinHash algorithm. Intuitively, items with similar text will have identical signatures. The next bolt extracts terms from the items i.e. n-grams and hashtags. These three bolts add the extracted data in the corresponding fields of the Item class. Solr indexer bolt used to index the items in a running instance of Solr. This index can be used afterwards to get the items relevant to an extracted topic.

At that point the topology is split into two parallel pipelines: the first one is used to extract trending topics based on trending terms, while the second used to cluster items based on minhash signature. Both of these pipelines use the same sequence of bolts, operate on a different field of the items (terms and minhash signature respectively).

The first bolt in these pipelines is the TermsRollingCountBolt. This bolt performs rolling counts of incoming objects (terms or minhash). Rolling means that by using a sliding window the bolt keeps track of statistics of an object to the current time window compared to the previous ones. At the end of each time window, the bolt emits a rolling count tuple per object, consisting of the object itself, its latest rolling count, a metric that indicates how trending is the object and the actual duration of the sliding window (just for testing as the length of the windows is constant).

As each object can emitted from multiple nodes in a distributed topology, the next two bolts used to aggregate and rank the objects, in a map-reduce fashion. The final ranking bolt emits a ranking of top objects (terms or minhash) for the current time window. Finally, each of the objects in the rank is stored in a MongoDB instance. The time stamp of the current window is also stored in order to keep track of the evolution of these objects over time. Finally, for each of the emitted top objects a label is extracted by using the text of the indexed items. Then by using either the terms or the minhash value a Solr query is performed. From this query, we get the most relevant and most popular item and we use its text as a label of the object.

3.1.2 Validation & Impact analysis

In the architecture described it is worth mentioning that although the Trend Topic Detector application deals with big streams of social networks data the use of Big Data technologies in the processing layer is quite limited. The idea is to re-engineer the architecture and introduce Big Data technologies where this is possible. By identifying and addressing quality-driven metrics we expect to isolate bottlenecks in the architecture and revise/redesign those parts by introducing Big Data technologies. More specifically, the revised architecture should satisfy the following requirements:

- High Availability
    - The system should be stable on a 24/7 basis
- Fault tolerance
    - The system should recover automatically in case of failure without losing significant data
- Performance
    - The system should be able to scale up in terms of throughput

The time behaviour of the Trend Topic Detector application is quite critical since the analysed information regarding the trending topics extracted should be indexed and exposed by the User Interface in almost real time, enhancing in this way the importance of the identified news topics. In order to achieve the above, we have applied a number of DICE tools in our environment, in order to validate and assess their results in a "real business" environment.

3.1.2.1 DICE IDE

DICE IDE has been used as the main access point of most DICE tools throughout the whole process of testing and validating DICE Tools. Using the IDE allowed us to minimize the learning curve we needed to cover for using the tools, as the IDE is based on Eclipse which was being already used by ATC engineers. Minimising the learning curve, allowed us to shift resources from training to production activities, which allowed us more time for application development, testing and productisation.

3.1.2.2 DICER and Deployment Service

The DICER tool allowed us to express the infrastructure needs and constraints for the Trend Topic Detector application and also to automatically generate deployment blueprints to be used on a cloud environment. We evaluated the usefulness of the DICER tool in terms of time saving, with regard to the time needed to setup the infrastructure manually from scratch, and the degree of automation that DICER offers. Note that in the total time that we computed for the DICER execution time we included the time needed by the DICE Deployment service to deploy the generated blueprint.

Table 11: DICER and Deployment Service KPI for the ATC use case

Manual setup time (mins)	DICER time (mins)	Time saving (%)	DICER automation (%)
230	40	82.6	72.5

The whole process was really fast and we achieved a time saving **almost 80% compared to the time we needed previousl**y when we were installing manually the Storm cluster and all of its dependencies (Zookeeper etc) as well as the Storm application and all the dependencies for the persistence layer. The fact that the TOSCA blueprints allow the refinement of the Storm-specific configuration parameters in advance is really convenient since we can experiment with different Storm cluster setups by applying another reconfiguration, resulting in a new testbed, until we reach the most efficient in terms of performance and throughput for our topologies. More detailed information can be found in **D6.2-Initial implementation and evaluation**.

3.1.2.3 Monitoring Tool

The ability to monitor Trend Topic Detector's deployed topologies is necessary in order to identify any possible bottleneck or even to optimize the performance and throughput by adding more parallelization for example. We have installed the Monitoring platform core modules, and we then used one of the features of latest Deployment Service release to automatically register the Storm cluster nodes on the core services of Monitoring platform during the infrastructure/application deployment phase. The monitoring was performed not only on a per (Stormcluster) node but also on a per application level which allowed us to distinguish between issues related to hardware specifications of the nodes and issues related to the application's internal mechanics like the size of internal message buffer of Storm. More detailed information can be found in **D6.2-Initial implementation and evaluation**.

Table 12: Monitoring Platform KPI for the ATC use case

Manual setup time (x #nodes)	Through DS
10 mins x 3 nodes = 30 mins	6 mins x 3 nodes = 18 mins

3.1.2.4 Quality Testing Tool

The Trend Topic Detector uses the Twitter's Streaming API which allows low latency real time access to Twitter's global stream of tweet data by making a long lived HTTP request and parsing the response incrementally. In that way, it avoids using the Twitter REST API where the API rate limit window duration is 15 minutes and the maximum number of allowed requests/call is up to 180 per time window. But the Streaming API also imposes some limitations regarding the size of the sampled tweets which is approximately ~1% of the total number of tweets published. The goal is to request a paid API from Twitter which would allow the Trend Topic Detector to crawl and analyse a bigger quota of the stream of published tweets. The logic was to run the Quality Testing tool for helping us in evaluating the maximum input rate that the trending topic detector topology can handle without drastically affecting the performance, mostly in terms of response time.

So, the goal of the experiment with quality testing was to stress test the capacity of the Trend Topic Detector topology. More specifically, we wanted to check how the topology is behaving in case of burst events (either political or social or economic) taking place in Twitter. Whenever such case happens, the social media users generate a significant high volume of messages and the DICE Quality Testing tool can help in simulating such heavy load conditions.

In order to validate the performance of the tool, a chrono assessment has been performed comparing the manual time that ATC engineers would need to perform manually the stress testing of the topology to the time that is required by the Quality Testing tool to perform a similar. As a result, we discovered that the reduction in time per test cycle is proportional to the number of experiments/iterations. More specific for:

- **Reduction of:** 27.027% for 4 experiments/iterations
- **Reduction of:** 34.782% for 5 experiments/iterations

More detailed information can be found in **D6.3-Consolidated Implementation and evaluation.**

3.1.2.5 Configuration Optimisation Tool

In general, a typical Storm deployment comprises of a variety of configuration properties that affect the behaviour of nimbus, supervisors as well as the topologies. It is a non-trivial task for a developer to select optimal values for the configuration properties in order to fine tune the Storm topology execution. This is usually a complicated task accomplished by experts in this area. We have applied the Configuration Optimization tool on the Trend Topic Detector topology, starting by making the topology reliable, that is to properly acknowledge the successfully processed tuples in order to have valid and consistent results regarding the performance metrics (throughput and response time) on each iteration of the tool. Also, we have created an offline feed Twitter replay script, to emit an already crawled corpus of Twitter items with a fixed rate for having a better control over the experiments. This has been done to avoid the sudden peaks in the input rate coming from the Twitter Streaming API (a common case when for example a burst event, political economical or other, happens), that could otherwise affect randomly the consistency of the measurements results on each iteration.

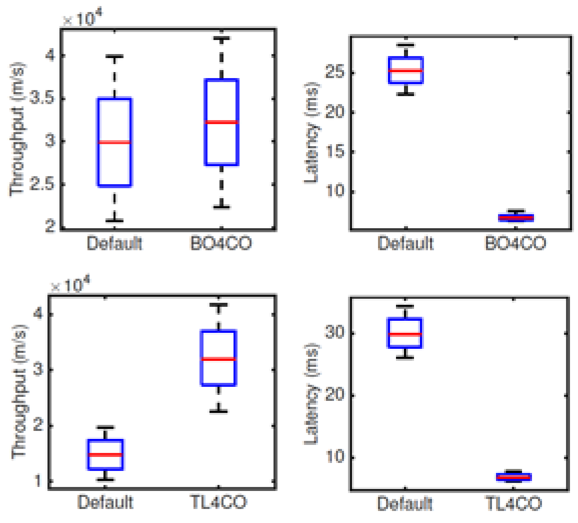


Figure 22: Configuration Optimisation Tool results

In order to evaluate the improvement by applying the Configuration Optimization tool on the Storm configuration that the Trend Topic Detector relies on, the ATC engineers have monitored the topology throughput as well as the latency. As it is shown in **Figure 22**, the impact on performance is more than twice compared to the default configuration which is a significant improvement. This achievement has been achieved after only 100 iterations which resulted in a total of 16 hours (100 * 10 minutes) of execution. The corresponding validation KPI has been fully addressed, resulting in a more than 30% improvement in both the throughput and the latency metrics. Adding to the above, by using this tool we have managed to do this without the need of a senior expert, but only with the support of a junior engineer, which allowed us to reduce resources thus shift them to other activities. More detailed information can be found in **D6.3-Consolidated Implementation and evaluation.**

Table 13: Configuration Optimsation Results in ATC Use Case

	Throughput (m/s)	Latency (ms)
Default config	1.4	30
TL4CO	3.2	4
	~128% incr	~86.6% decr

3.1.2.6 Fault Injection Tool

Since the Trend Topic Detector topologies deal with a cloud deployment it is quite critical to be able to test the consequences of faults early in the development phase in a controlled environment rather than once the application is in production. It is important for the Trend Topic Detector DIA to be comprised of reliable topologies due to the nature of the processing it performs: if for example there is a burst event at some time then losing some of the social networks messages due to network failures/repartitions could affect the quality of the trending topics identified at that period.

For the reasons above, we deployed the Fault Injection tool in order to test how the application behaves in terms of reliability and fault-tolerance and eliminate any single point of failure as possible. To this end, we used the Fault Injection tool to randomly stop/kill not only the various types of Storm processes (nimbus, supervisor and worker processes) but also a whole node of the Storm cluster, and we checked how those actions affect the proper execution of the topologies

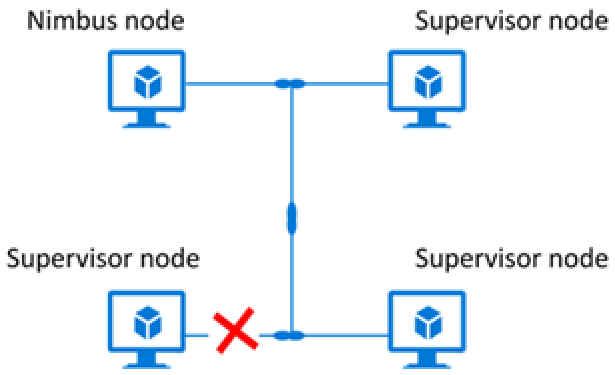


Figure 23: Fault injection tool indicative scenario

Finally, we used the Fault Injection tool to generate high memory and CPU usage for the Storm cluster VMs to simulate high memory/CPU load. The various processing bolts of the topology were not affected too much since none of them is high CPU-bound or memory greedy. Using the Fault Injection Tool allowed us to deploy a larger number of "faults" when compared to the manual procedure (20%), therefore be able to collect a larger number of results. However, we didn't manage to reach the desired KPI, set by the tool owner. **Additional validation activities are currently on going and will continue after M36**.

Table 14:Fault Injection Results

Average manual setup time per tool/fault(mins)	Total manual setup time (mins) *	FI setup time (mins) *	Time saving (%)
15	3*3*15=135	3*3*12=108	20

More detailed information can be found in **D6.2-Initial implementation and evaluation**.

3.1.2.7 Verification Tool

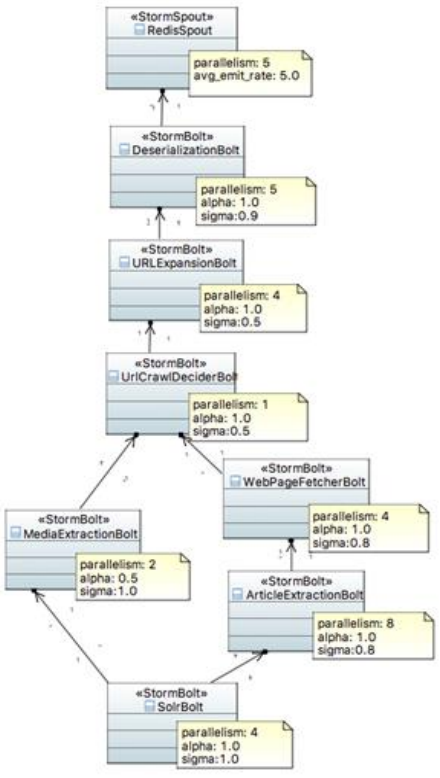


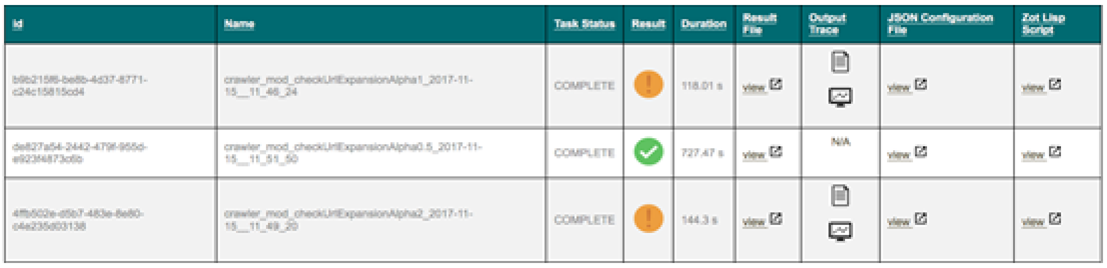
Figure 24: Class Diagram of the Crawler topology

The DICE Verification Tool (D-VerT) has been used in the Trend Topic Detector case study to perform formal analysis on the Crawler topology, and check for the presence of timing violations possibly leading to the saturation of the input buffer in some bolts. The topology has been designed at the DTSM level as a DICE – profiled UML Class Diagram, as shown in **Figure 24**.

D-VerT identified two violations on the UrlExpansionBolt component. After the first violation was found, a first refinement consisted in diminishing the value of alpha from 2 to 1. Even after this change the tool found an issue in the component, providing an execution trace showing an unbounded increase in the number of elements in the input buffer. By diminishing again the value of alpha to be 0.5, the issue was solved and the tool did not find any violating execution for that bolt. However, this improvement in the processing time of UrlExpansionBolt impacted on the bolt downstream, i.e., UrlCrawlDeciderBolt. In fact, a new violation was found for on the input buffer of *UrlCrawlDeciderBolt,* which turned out not being able to process the incoming flow of data in a timely manner. In this case the refinement consisted in increasing the level of parallelism of the bolt. A first increase, from 1 to 4, was not enough, as resulted in another violation, while, with a parallelism level of 6, no violation was found by the tool.

Table 15 shows the D-VerT dashboard for some of the configuration that underwent verification.

Table 15: D-VerT Tasks Dashboard



Id	Name	Task Status	Result	Duration	Result File	Output Trace	JSON Configuration File	Zot Lisp Script
b9b215f6-be8b-4d37-8771-c24c15815cd4	crawler_mod_checkUrlExpansionAlpha1_2017-11-15__11_46_24	COMPLETE	!	118.01 s	view		view	view
de827a54-2442-479f-955d-e923f4873c6b	crawler_mod_checkUrlExpansionAlpha0.5_2017-11-15__11_51_50	COMPLETE	✓	727.47 s	view	N/A	view	view
4ffb502e-d5b7-483e-8e80-c4e235d03138	crawler_mod_checkUrlExpansionAlpha2_2017-11-15__11_49_20	COMPLETE	!	144.3 s	view		view	view

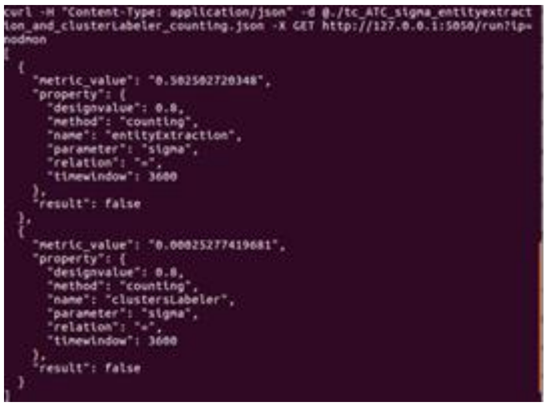
Adding to the above, in the following table we report the details of the most relevant verification tasks performed and described above. The highlighted *alpha* and *parallelism* refer to the current bolt under verification, and have to be considered as the only variations with respect to the initial configuration of the topology.

Table 16: Verification Tasks

Bolt under verification	*alpha*	*parallelism*	Outcome	Verification Time
UrlExpansionBolt	2.0	4	SAT (violation found)	118s
UrlExpansionBolt	1.0	4	SAT (violation found)	1435
UrlExpansionBolt	0.5	4	UNSAT (NO violation found)	7275s
UrlCrawlDeciderBolt	1.0	1	SAT (violation found)	770s
UrlCrawlDeciderBolt	1.0	4	SAT (violation found)	1483s
UrlCrawlDeciderBolt	1.0	6	UNSAT (NO violation found)	1461s

As a conclusion, we can report that by deploying the verification tool, we managed to easily **spot four (4) violations** of properties on timing constraints, which allowed us to apply all possible configurations, without the need of spending countless resources for spotting these violations.

3.1.2.8 Trace Checking Tool



```
curl -H "Content-Type: application/json" -d @./tc_ATC_sigma_entityextract
ion_and_clusterLabeler_counting.json -X GET http://127.0.0.1:5050/run?lp=
nodmon
[
  {
    "metric_value": "0.502502720348",
    "property": {
      "designvalue": 0.8,
      "method": "counting",
      "name": "entityExtraction",
      "parameter": "sigma",
      "relation": "<",
      "timewindow": 3600
    },
    "result": false
  },
  {
    "metric_value": "0.00025277419681",
    "property": {
      "designvalue": 0.8,
      "method": "counting",
      "name": "clustersLabeler",
      "parameter": "sigma",
      "relation": "<",
      "timewindow": 3600
    },
    "result": false
  }
]
```

Figure 25: Trace Checking Tool

The Trace Checking Tool has been employed for the validation and the refinement of the DICE-TraCT. By deploying the DICE-TraCT allowed us to implement specific monitoring solutions for the collection of information through the analysis of the application logs. More specific, the tool enabled us to analyse the logs to elicit the parameter values, related to certain specific features of the running application, that are necessary for the verification carried out by the Verification tool. In particular, the value of two parameters sigma for the Storm bolts and emit rate for the Storm spouts have been evaluated through the trace-checking analysis. The obtained values have been used in the Verification tool to refine the DTSM model of the application. The application logs have been analyzed in order to calculate the value of sigma for two bolts EntityExtraction and ClusterLabeler.

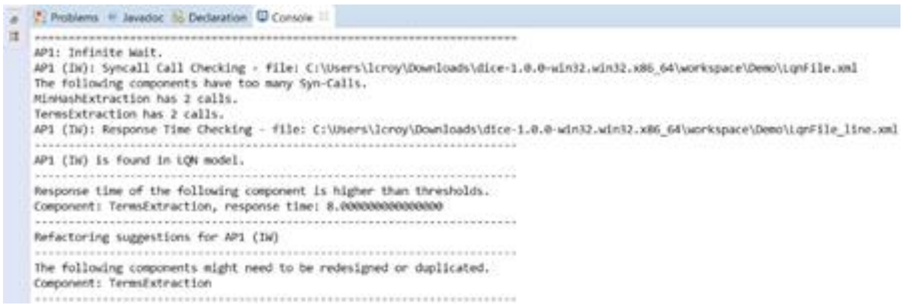
3.1.2.9 Simulation Tool

Scalability, bottleneck detection and simulation/predictive analysis are some of the core requirements for the News Orchestrator DIA. We have deployed the DICE Simulation tool in order to perform a performance assessment of our Storm based DIA that would allow us to predict the behavior of the system prior to the deployment of it on a production cloud environment. The results from deploying the tool, showed that for some of the bolts (approximately more than half) the prediction error is indeed very small, less than 10%, predicting quite accurately the capacity of the bolts. For ATC engineers, having a tool like the DICE Simulation tool in the stack of the tools is a significant advantage. Every new feature that is added in the News Orchestrator DIA may result in an undesired imbalance with regards to the performance of the system. Being able to validate the performance impact on the DIA prior to the actual deployment on a cloud infrastructure gives the flexibility to fine tune the topology configuration in advance and take corrective actions (i.e. scaling by increasing bolts parallelism) without wasting resources (costs and efforts) that would be otherwise required by an actual deployment on the cloud. More detailed information can be found in **D6.3-Consolidated Implementation and evaluation.**

3.1.2.10 Enhancement Tool

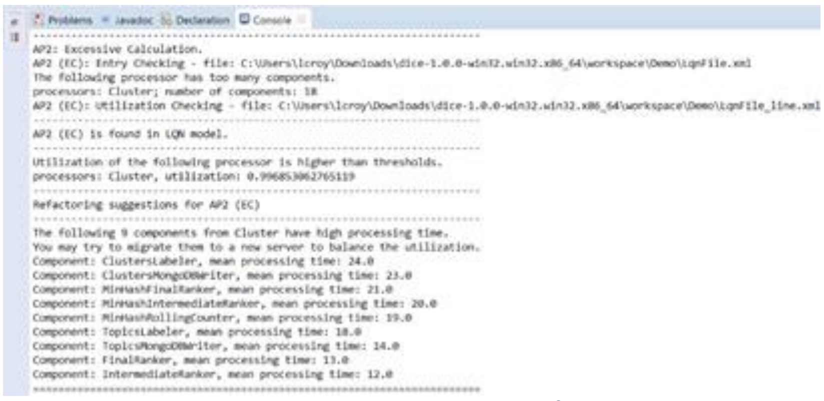
The main pipeline of topic-detector of ATC case is also tested on APR. The detector is implemented as a storm topology, where the sequential bolts perform a specific operation on the detection procedure. The test environment is based on DICE IDE 1.0.0 and MATLAB Compiler Runtime (MCR) R2015a. The ATC application is composed of 18 components. We assume that the threshold on the number of components, which can be deployed on a server, is 8. The upper bound of the CPU utilization is 0.8. The maximum number of call is 1 and the upper bound of response time is 7.0 sec.

The problem faced by ATC is the following. The task of providing a refactoring of the design if the performance anti-patterns (i.e, Infinite Wait and Excessive Calculation) are found in the ATC application. It consists of modelling, analysis, extraction, detection and refactoring. 1) Modelling. Two models (i.e., architecture model and performance model) need to be generated for the later processing. DICE IDE builds the design time model in the form of UML by using Papyrus editor. The UML model of ATC is annotated with core tags (e.g., hostdemand) of stereotypes of MARTE and DICE profiles. APR helps to generate performance model (i.e., LQN) according to the corresponding UML model. 2) Analysis. Both the solved LQN model and the performance analysis results can be obtained with the help of the LINE solver. The analysis results of CPU utilization of the cluster is 0.9968. 3) Extraction. Extracting the constraints (i.e., performance upper bound) to be used in the deployment refactoring. 4) Detection and refactoring. As results of the Excessive Calculation detection and refactoring, the threshold on the number of component is 8, the current deployment violates the bound. Furthermore, the utilization 0.9968 is greater than the threshold on CPU utilization (0.8). Thus the Excessive Calculation anti-pattern is found. As results of the Infinite Wait detection and refactoring, there are two component that have more than one function call but only one of them response time is great than the threshold (7.0 sec). Thus, the refactoring suggestions for those two anti-patterns are shown in **Figure 26** and **Figure 27**.



```
---------------------------------------------------------------------------
AP1: Infinite Wait.
AP1 (IW): Syncall Call Checking - file: C:\Users\lcroy\Downloads\dice-1.0.0-win32.win32.x86_64\workspace\Demo\LqnFile.xml
The following components have too many Syn-Calls.
MinHashExtraction has 2 calls.
TermsExtraction has 2 calls.
AP1 (IW): Response Time Checking - file: C:\Users\lcroy\Downloads\dice-1.0.0-win32.win32.x86_64\workspace\Demo\LqnFile_line.xml
---------------------------------------------------------------------------
AP1 (IW) is found in LQN model.
---------------------------------------------------------------------------
Response time of the following component is higher than thresholds.
Component: TermsExtraction, response time: 8.000000000000000
---------------------------------------------------------------------------
Refactoring suggestions for AP1 (IW)
---------------------------------------------------------------------------
The following components might need to be redesigned or duplicated.
Component: TermsExtraction
---------------------------------------------------------------------------
```

Figure 26: Infinite Wait detection and refactoring suggestion



```
----------------------------------------------------------------------
AP2: Excessive Calculation.
AP2 (EC): Entry Checking - file: C:\Users\lcroy\Downloads\dice-1.0.0-win32.win32.x86_64\workspace\Demo\LqnFile.xml
The following processor has too many components.
processors: Cluster; number of components: 18
AP2 (EC): Utilization Checking - file: C:\Users\lcroy\Downloads\dice-1.0.0-win32.win32.x86_64\workspace\Demo\LqnFile_line.xml
----------------------------------------------------------------------
AP2 (EC) is found in LQN model.
----------------------------------------------------------------------
Utilization of the following processor is higher than thresholds.
processors: Cluster, utilization: 0.996853062765119
----------------------------------------------------------------------
Refactoring suggestions for AP2 (EC)
----------------------------------------------------------------------
The following 9 components from Cluster have high processing time.
You may try to migrate them to a new server to balance the utilization.
Component: ClustersLabeler, mean processing time: 24.0
Component: ClustersMongoDBWriter, mean processing time: 23.0
Component: MinHashFinalRanker, mean processing time: 21.0
Component: MinHashIntermediateRanker, mean processing time: 20.0
Component: MinHashRollingCounter, mean processing time: 19.0
Component: TopicsLabeler, mean processing time: 18.0
Component: TopicsMongoDBWriter, mean processing time: 14.0
Component: FinalRanker, mean processing time: 13.0
Component: IntermediateRanker, mean processing time: 12.0
**********************************************************************
```

Figure 27: Excessive Calculation detection and refactoring suggestions

DICE APR can detect 2 performance anti-patterns in the Storm topology. DICE APR takes some performance indexes, e.g., response time and CPU utilization, as input parameters, and detects the application performance anti-patterns by mean of analyzing the solved performance model with performance bounds, and provides architecture refactoring suggestions. Tests executed on the Storm-based applications (e.g., Wikistats and ATC case) have shown that DICE APR tool is able to effectively detect performance anti-patterns. The tool enables the end user to quickly identify the performance issues of running Storm applications and apply the corresponding refactoring action (e.g., component redesign).

3.1.3 Discussion

Throughout the deployment of DICE tools during the last 36 months, we have managed to validate most of the tools in our development environment. Moreover, a number of DICE tools have provided a significant advantage by reducing our development time thus reducing our operational and development costs by a significant percentage or even allowing us to assign less experienced developers to complicated projects instead of wasting “expensive” senior developers time. DICE tools have also enabled us to develop and fine tune a critical module for us (Trend Topic Detector) which has evolved into a powerful tool, sitting in the heart of one of our newest and most innovative products ready to be deployed into the market.

3.2 PRO Use Case

Posidonia Operations is an Integrated Port Operation Management System highly customizable that allows a port to optimize its maritime operational activities related to the flow of vessels in the port service area, integrating all the relevant stakeholders and computer systems.

In technical terms, Posidonia Operations is a real-time and data intensive platform able to connect to AIS (Automatic Identification System), VTS (Vessel Traffic System) or radar, and automatically detect vessel operational events like port arrival, berthing, unberthing, bunkering operations, tugging, etc.

Posidonia Operations is a commercial software solution that is currently tracking maritime traffic in Spain, Italy, Portugal, Morocco and Tunisia, thus providing service to different port authorities and terminals.

The goals of creating this case study are adopting a more structured development policy (DevOps), reducing development/deployment costs and improve the quality of our software development process.

In the use case, the following scenarios are considered: deployment of Posidonia Operations on the cloud considering different parameters, support of different vessel traffic intensities, add new business rules (high CPU demand), run simulation scenario to evaluate performance and quality metrics.

3.2.1 Business goals

Three main business goals have been identified for the Posidonia Operations use case.

- **Lower deployment and operational costs.**

Posidonia Operations is offered in two deployment and operational modes: on-premises and on a virtual private cloud. When on-premises, having a methodology and tools to ease the deployment process will result in a shortened time to production, thus saving costs and resources. In the case of a virtual private cloud deployment, it is expected that the monitoring, analysis and iterative enhancement of our current solution will result in better hardware requirements specifications, which in the end are translated into lower operational costs.

- **Lower development costs.**

Posidonia Operations is defined as a "glocal" solution for maritime operations. By "glocal" we mean that it offers a global solution for maritime traffic processing and analysis that can be configured, customized and integrated according to local requirements. In addition, the solution operates in real-time making tasks like testing, integration, releasing, etc. more critical. By the application of the methodology explained in the book, these tasks are expected to be improved in the development process, thus resulting in shortened development lifecycles and lower development costs.

- **Improve the quality of service.**

Several quality and performance metrics have been considered of interest for the Posidonia Operations use case. Monitoring, predictive analysis or ensure reliability between successive versions will end in an iterative enhancement of the quality of service to our current customers.

3.2.2 System Architecture

Posidonia Operations is an integrated port operations management system. Its mission consists on "glocally" monitor vessels' positions in real time to improve and automatize port authorities operations. The below image shows the general architecture of Posidonia Operations. The architecture is based on independent Java processes that communicate with each other by means of a middleware layer that gives support to a Message Queue, a Publication and Subscription API and a set of Topics to exchange data among components.

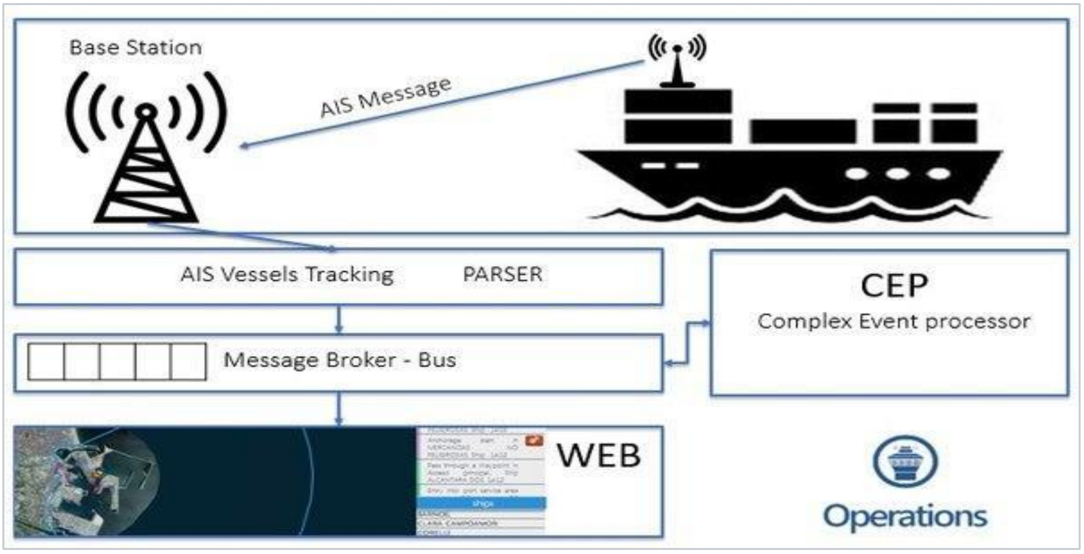
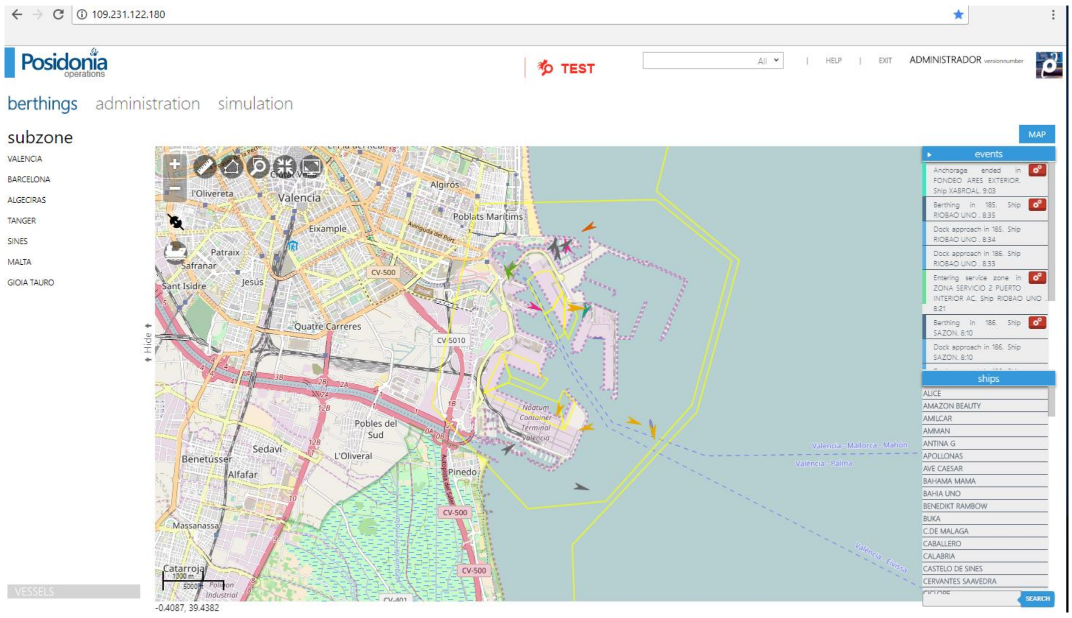


Figure 28: Posidonia Operations general architecture

An overview of the main components for Posidonia Operations would be:

- Vessels in the service area of a port send AIS messages that include their location and other metadata to a central station. (This is out of the scope of the architecture diagram)
- An AIS Receiver (a spout) receives those messages and emits them through a streaming channel (usually a TCP connection)

- The AIS Parser (a bolt) is connected to the streaming channel, parses the AIS messages into a middleware topic and publishes it to a Message Queue.
- Other components (bolts) subscribe to the Message Queue to receive messages for further processing. As an example, the Complex Event Processing engine receives AIS messages in order to detect patterns and emit events to a different Message Queue.
- The Posidonia Operations client "Web" allows to the employees of the port to have a visual tool that allows them to control the location of the vessels in real time. This website shows on a map the different vessels that are within the area of influence of a port, with a list of operations that are happening.



Anchorage ended in FONDEO ARES EXTERIOR. Ship XABROAL. 9:03
Berthing in 185. Ship RIOBAO UNO . 8:35
Dock approach in 185. Ship RIOBAO UNO . 8:34
Dock approach in 186. Ship RIOBAO UNO . 8:33
Entering service zone in ZONA SERVICIO 2 PUERTO INTERIOR AC. Ship RIOBAO UNO . 8:21
Berthing in 186. Ship SAZON. 8:10
Dock approach in 186. Ship SAZON. 8:10

ALICE
AMAZON BEAUTY
AMILCAR
AMMAN
ANTINA G
APOLLONAS
AVE CAESAR
BAHAMA MAMA
BAHIA UNO
BENEDIKT RAMBOW
BUKA
C.DE MALAGA
CABALLERO
CALABRIA
CASTELO DE SINES
CERVANTES SAAVEDRA

Figure 29: Posidonia Operations Client

3.2.3 Use Case Scenarios

There exists different usual scenarios where Posidonia Operations development lifecycle can benefit from the DICE Framework and DICE Methodology. These scenarios are a small subset of the possible ones but are representative of interesting situations and are based on our current experience delivering a data intensive application to port authorities and terminals.

3.2.3.1 Deployment Scenario

Currently Posidonia Operations can be deployed in two fashions:

- **On-premises:** The port authority provides its own infrastructure and the platform is deployed on Linux virtual machines
- **In the cloud:** Posidonia Operations is also offered as a SaaS for port terminals. In this case, we use the Amazon Virtual Private Cloud (VPC) to deploy an instance of Posidonia Operations that gives support to different port terminals.

Apart from this, Posidonia Operations configuration varies depending on the deployment environment:

- Hardware requirements (number of nodes, CPU, RAM, DISK) to deploy of Posidonia Operations on each port is based on the team experience. For each deployment, the hardware requirements are calculated manually by engineers, considering the estimation of the number of vessels and the complexity of the rules applies for each message to be analysed. DICE tools can help to tune automatically the appropriate hardware requirements for each deployment.

- Posidonia Operations deployment and configuration is done by a system administrator and a developer and it varies depending on the port authority. Although deployment and configuration is documented, the DICE tools can help to adopt a DevOps approach, where deployment and configuration can be modelled in order not only to better understand the system by different stakeholders, but also to automate some tasks.
- A DevOps approach can help to provide also test and simulation environments that will improve our development lifecycle.

3.2.3.2 Support vessels traffic increase for a given port

Posidonia Operations core functionality is based on analysing a real-time stream of messages that represent vessels positions to detect and emit events that occur on the real world (a berth, an anchorage, a bunkering, etc.).

Different factors can make the marine traffic of a port increase (or decrease), namely:

- Weather conditions
- Time of the day
- Season of the year
- Current port occupancy
- etc.

This means that the number of messages per second to be analysed is variable and can affect performance and reliability of the events detected if the system is not able to process the streaming data as it arrives. When this is not possible, messages are queued and this situation has to be avoided.

We currently have tools to increase the speed of the streaming data to validate the behaviour of the system in a test environment. However, the process of validating and tuning the system for a traffic increase is a tedious and time consuming process where DICE tools can help to improve our current solution.

3.2.3.3 Add new business rules (CEP rules) for different ports

Analysis of the streaming data is done by a Complex Event Processing engine. This engine can be considered as a “pattern matcher”. For each vessel position that arrives, it computes different conditions, that when satisfied produce an event.

The number of rules (computation) to be applied to each message can affect the overall performance of the system. Actually, the number and implementation of rules vary from one deployment to other.

DICE tools can help on different quality and performance metrics, simulation and predictive analysis, optimization, etc. in order to tune our current solution.

3.2.3.4 Give support to another port in the cloud instance of Posidonia Operations

Give support to another port (or terminal) in the cloud instance of Posidonia Operations usually means:

- Increase the streaming speed (more messages per second)
- Increase on computation needs (more CEP rules executed per second)
- Deployment and configuration of new artefacts and/or nodes

In this case, DICE tools can help improve Posidonia Operations also on estimating the monetary cost of introducing a new port on the cloud instance.

3.2.3.5 Run a simulation to validate performance and quality metrics among versions

CEP rules (business rules) evolve from one version of Posidonia Operations to another. That means that performance and quality of the overall solution could be affected by this situation among different versions. Some examples of validations we currently do (manually):

- Performance: New version of CEP rules don't introduce a performance penalty on the system
- Performance: New version of CEP rules don't produce queues
- Reliability: New version of CEP rules provide the same output as prior version (they both detect the same events)

One of the main issues of the current situation is that measuring the performance (system performance and quality of the data provided by the application) is done manually and it's very costly to obtain an objective quantification. By using DICE simulation tools, performance and reliability metrics can be predicted for different environment configurations, thus ensuring high quality versions and non-regression.

3.2.4 Validation & Impact analysis

The requirements of the Posidonia Operations use case can be classified in three types:

1. Assessment of the impact in performance after changes in software or conditions.
2. Automatic extraction of relevant performance metrics and KPIs
3. Automation of deployment

After analysing the DICE Tools, we have detected which tools would be interesting to use to achieve the requirements. This section explains how we used the tools for the use case.

3.2.4.1 DICE IDE

The DICE IDE integrates all the tools of the proposed platform and gives support to the DICE methodology. The IDE is an integrated development environment tool, based on Eclipse, for MDE where a designer can create models to describe data-intensive applications and their underpinning technology stack. The DICE IDE integrates the execution of the different tools, in order to minimize learning curves and simplify adoption. More detailed information can be found in **D6.2-Initial implementation and evaluation**.

3.2.4.2 DICER

The DICER tool allows us to generate the equivalent TOSCA Blueprint (deployment recipe) from the Posidonia use case DDSM created using the DICE IDE. This Blueprint is used by the deployment service to automatically deploy the Posidonia use case. With this tool you can obtain different blueprints for different configurations of the use case. The generated TOSCA blueprint file contains all the information specified in the DDSM Model and can be used to deploy the full use case. No manual changes were made to the generated TOSCA Blueprint for the deployment. More detailed information can be found in **D6.2-Initial implementation and evaluation**.

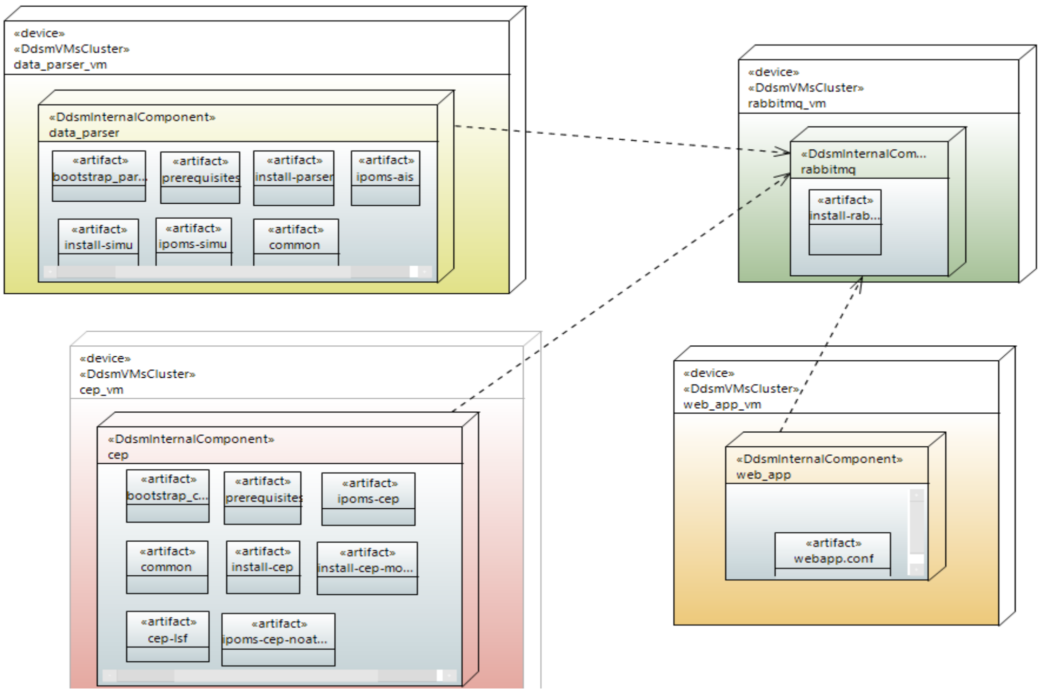
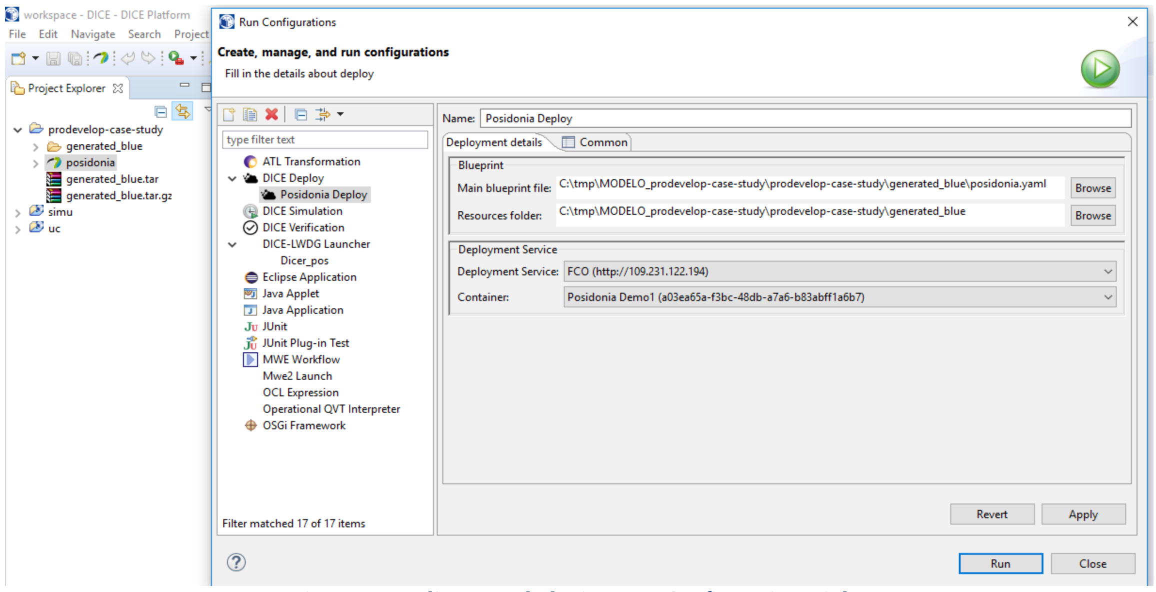


Figure 30: Posidonia use case DDSM

To deploy the solution on flexiOPS FCO, we used the new Delivery Tool plugin available in the DICE IDE “0.1.14” (see **Figure 31**).



Create, manage, and run configurations

Fill in the details about deploy

Name: Posidonia Deploy

Blueprint

Main blueprint file: C:\tmp\MODELO_prodevelop-case-study\prodevelop-case-study\generated_blue\posidonia.yaml

Resources folder: C:\tmp\MODELO_prodevelop-case-study\prodevelop-case-study\generated_blue

Deployment Service

Deployment Service: FCO (http://109.231.122.194)

Container: Posidonia Demo1 (a03ea65a-f3bc-48db-a7a6-b83abff1a6b7)

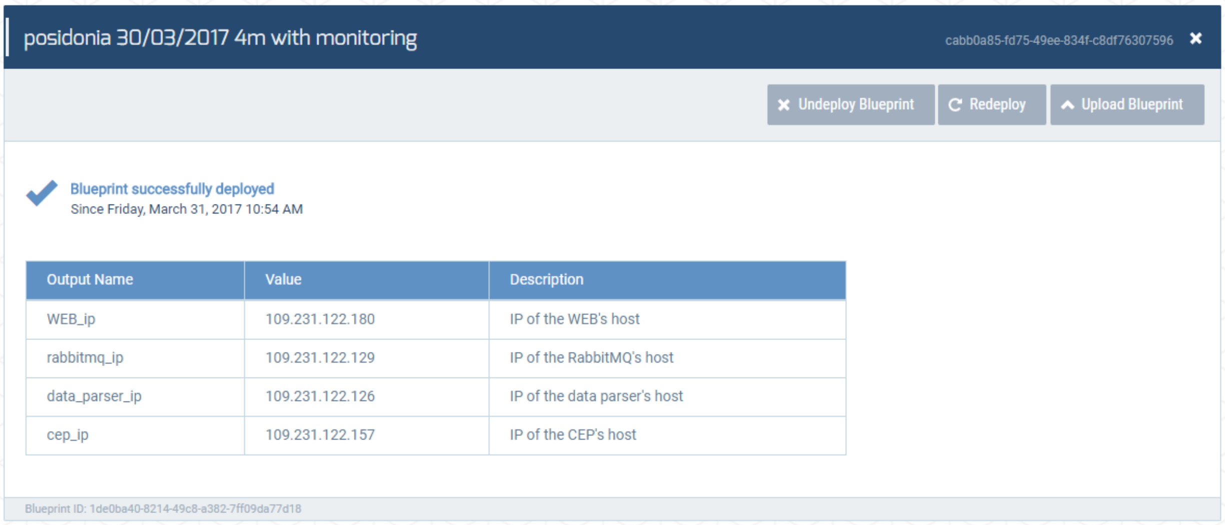
Figure 31: Delivery Tool plugin - Run Configuration Dialog

3.2.4.3 Deployment Service

Deploy the Posidonia Operations manually is quite time and cost consuming. Deployment Service is able to deploy on the cloud a configuration of the Posidonia Operations use case in few minutes. The last version of the deployment service works over FCO and Amazon AWS. To deploy a solution using the deployment service it is needed to provide as an input the TOSCA Blueprint of it. This blueprint is obtained using the DICER tool.

Using Deployment service, the deployment is faster (only few minutes), you can deploy different configurations of your solution, and it does not require system administrator experts because the deployment is automatized.

Figure 32 shows the four machines created by the Delivery Tool as it was specified in the DDSM model and the corresponding TOSCA blueprint. If we access to the Posidonia web client (id: 109.231.122.180), we can see a map of the Valencia's Port with the current vessels in its area of action (see Figure xx). The vessels are shown because all other Components are running properly and are providing the information that is shown into the web client.


posidonia 30/03/2017 4m with monitoring
cabb0a85-fd75-49ee-834f-c8df76307596
Undeploy Blueprint
Redeploy
Upload Blueprint

Blueprint successfully deployed
Since Friday, March 31, 2017 10:54 AM

Output Name	Value	Description
WEB_ip	109.231.122.180	IP of the WEB's host
rabbitmq_ip	109.231.122.129	IP of the RabbitMQ's host
data_parser_ip	109.231.122.126	IP of the data parser's host
cep_ip	109.231.122.157	IP of the CEP's host

Blueprint ID: 1de0ba40-8214-49c8-a382-7ff09da77d18

Figure 32: Posidonia use case - Deployed machines

In the last year of the DICE project, the DICER and deployment tool included support for deployments on the Amazon Web Services (AWS). We have used the last versions of these tools to deploy Posidonia Operations on AWS with the support of the XLAB.

To be able to carry out automatic deployments on amazon, it is necessary to install a set of software tools. Specifically, you should install Cloudify Manager and Deployment service in your instance of AWS EC2.

The same DDSM used to deploy the use case on FCO was used to deploy the uses case on Amazon. The only change made was to modify the name of the "Image" used to display "posidonia web client", which due to its complexity, we opted to use an image of the machine instead of installing it from scratch using the deployment service. In the Amazon deployment, we used exactly the same strategy and configuration that we had used for the FCO deployment.

The image of the "Operation web client" was created manually in a local environment using virtual box software, after that we used an import facility provided by AWS to import the ISO image.

Some problems arose when trying to use the new image during the deployment due to permission issues, because the deployment service was not able to access to the machine created from the image. Moreover, we had to modify the scripts used to configure the machine to correct issues related with the remote configuration of the "Posidonia web Client"

Once the AWS is configured and tested, the time required to deploy the full use case on AWS is between 12-15 minutes.

Table 17: DICER and Deployment Service KPI for the Prodevelop use case

	without DICE (min)	With DICE (min)	Time saving(%)	Automation with DICE (%)
First Deployment	600-1200	300	50%-75%	100%
Next deployments	600-1200	20	97,3%- 98,6%	100%

3.2.4.4 Monitoring Tool

This tool allows us to obtain metrics in real time for a running instance of the Posidonia Operations: Generic hardware performance metrics (CPU and memory consumption, disk usage, etc.) and Specific use case metrics, such as, number of events detected, location of the events, execution time per rule, messages per second, etc.

The reported results, provided by the monitoring platform, allow the architect and developers to eventually update the DDSM and/or DPIM models to achieve a better performance. Another interesting point is that the Monitoring Tools facilitates the integration with the Anomaly Detection Tool and the Trace Checking Tool, because both tools use the information stored in the monitoring as a data input.

Figure 33 shows one of the dashboards created for the Posidonia use case. This dashboard contains 3 visualizations related with the "rules/events" detected by the CEP component:

- "**avg micros of rules**" shows a linear graph with the average execution cost of the different rules in the last minutes group by minutes.
- "**Pie Chart Rules (occurrences)**" shows a chart distribution with the number of rules executed in a period group by type of rule.
- "**Pie Chart Rules (sum of microseconds)**" shows a chart distribution with the cost of the rules executed in a period group by type of rule.

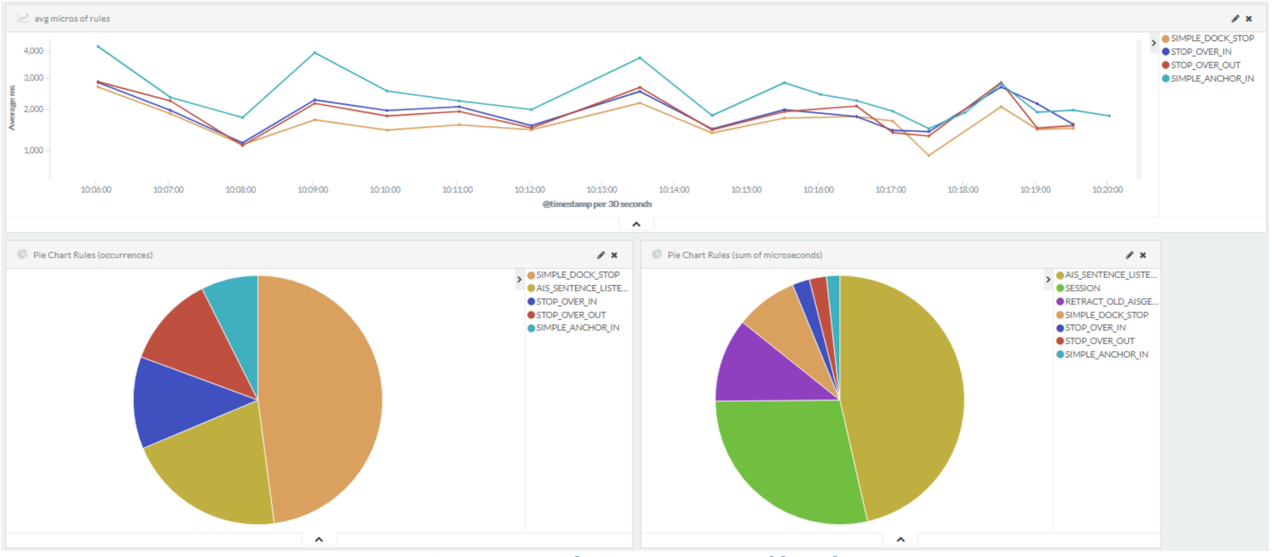


Figure 33: Posidonia use case - Dashboard 2

Figure 34 shows the "dashboard 3" created for the Posidonia use case. This dashboard contains 3 visualizations related with the location of "rules/events" detected by the CEP component:

- "**map**" shows a map with the exact location of the rules detected in a period of time. Each point of the map represents the exact location where they occurred. The bigger the point is, more events occurred in that location.
- "**amount of events in time**" shows the number of event that occurs each minute group by type of event.
- "**total events**" counts the number of events analysed by the Posidonia use case system.

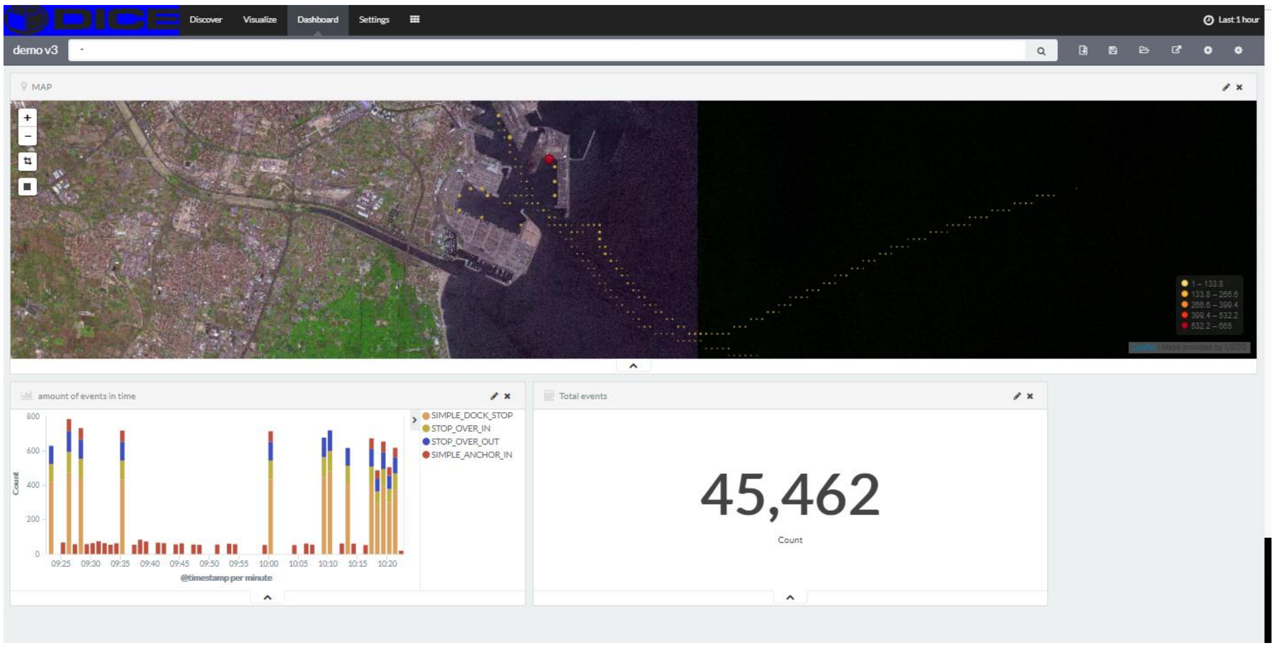


Figure 34: Posidonia use case - D-MON Dashboard 3

This tool allows us to obtain metrics for a running instance of Posidonia Operations:

- Generic hardware performance metrics (CPU and memory consumption, disk usage, etc.)
- Specific use case metrics, such as, number of events detected, location of the events, execution time per rule, messages per second, etc.

The reported results allow the architect and developers to eventually update the DDSM and/or DPIM models to achieve a better performance.

Another interesting point is that the Monitoring Tools facilitates the integration with the Anomaly Detection Tool and the Trace Checking Tool, because both tools use the information stored in the monitoring as a data input. More detailed information can be found in **D6.2-Initial implementation and evaluation**.

3.2.4.5 Fault Injection Tool

DICE Fault Injection Tool (FIT) is used to generate faults within virtual machines. In Posidonia use case, this tool is useful to check how the CEP component behaves against a high CPU load. To observe the behaviour of the system, we use the Monitoring Tool that contains a specific visualization for the system load.

Although the Fault Injection Tools can launch other types of faults, for Posidonia use case, only the High CPU fault is relevant to evaluate the response of the system to this situation. It's important validate that the System continues working and no event is lost when a high load happens.

Posidonia use case behaves well against high CPU demands, and only a little increment in the required time to process events is observed. Moreover, no event is lost due to the high CPU demand and we can affirm that Posidonia use case is reliable against high CPU loads. More detailed information can be found in **D6.3-Consolidated Implementation and evaluation.**

3.2.4.6 Anomaly Detection Tool

Anomaly Detection Tool allows us to validate that the system works as is expected with the current rules and with the addition of new ones. That is, no events are lost, no false positives are given, the execution time is kept within a reasonable range or the order of events detected is adequate.

Anomaly Detection Tool is mostly used in the use case to detect anomalies related with the processing time required of the different events that the system analyses.

First of all, we studied the log file generated by the CEP component (see **Table 18**Table 1). The "component" column represents the type of rule, the "key" column represents the timestamp, the "method" column represents the function called, the "ms" represents the time need to evaluate the rule and the "Ship" column represent the identification of the vessels. We observed that each type or rule analysed has a similar execution cost (column "ms").

Table 18: Example of the Rules execution cost

Component	key	method	ms	ship
AIS_SENTENCE_LISTENER	2017-06-22T11:53:04.278Z	HANDLE_MESSAGE	209	211636100
SIMPLE_ANCHOR_OUT	2017-06-22T11:53:04.272Z	UPDATE_ACTIVE	710	305965000
STOP_OVER_OUT	2017-06-22T11:53:04.271Z	UPDATE_ACTIVE	652	305965000
SIMPLE_DOCK_STOP	2017-06-22T11:53:04.270Z	UPDATE_ACTIVE	293	305965000
STOP_OVER_IN	2017-06-22T11:53:04.270Z	UPDATE_ACTIVE	680	305965000
SIMPLE_DOCK_STOP	2017-06-22T11:53:04.269Z	UPDATE_ACTIVE	295	305965000
AIS_SENTENCE_LISTENER	2017-06-22T11:53:04.263Z	HANDLE_MESSAGE	9483	305965000
SIMPLE_ANCHOR_OUT	2017-06-22T11:53:04.262Z	UPDATE_ACTIVE	755	305965000
STOP_OVER_OUT	2017-06-22T11:53:04.261Z	UPDATE_ACTIVE	618	305965000
STOP_OVER_IN	2017-06-22T11:53:04.260Z	UPDATE_ACTIVE	686	305965000
SIMPLE_DOCK_STOP	2017-06-22T11:53:04.259Z	UPDATE_ACTIVE	303	305965000
AIS_SENTENCE_LISTENER	2017-06-22T11:53:04.246Z	HANDLE_MESSAGE	199	225366000
AIS_SENTENCE_LISTENER	2017-06-22T11:53:04.245Z	HANDLE_MESSAGE	766	224161160

To validate the use case, some anomaly detection methods have been used: AdaBoost, Decision Tree and Random Forest. For supervised learning methods labelled anomalies from application data instances are a prerequisite. The data sets must be labelled to create a viable training and validating data set. Once this is done the resulting predictive models can be easily applied at runtime. We manually label a dataset comprising over 4800 data points taking into account the considerations of the **Table 19**.

For validation purposes, we ran all supervised and unsupervised methods on this data set. We can see in the following table the results of the first validations. First, we ran a baseline where all methods had their parameters set to default values and saved both the score and the time it took to train a model (BScore, BTime). After that, we ran parameter optimization on all methods and executed a 10-fold cross validation with 30% of the dataset used for validation. We can see that the parameter optimization not only allow us to optimize the predictive performance but also the required training time (BScore and BTime for the baseline and FScore and FTime for the best performing).

Table 19: Anomaly Detection Experiments

Method	BScore	BTime	Param Search	CV Mean	CV STD	CV Time	Fscore	FTime
RF	0,68	0,061148882	185,4456398	99,98%	0,05 %	27,50044012	1	4,573200941
DT	0,53	0,004851103	5,034779072	99,97%	0,09 %	0,047837019	0,998913718	0,003602982
AD	0,51	0,235774994	62,63462806	100,00 %	0,00 %	0,415092945	1	0,05479002
NN	0,34	0,564079046	1771.0435	100.00 %	0.00 %	0.2695	1.0	0.03283

An interesting observation which can be made using ADT is the so called feature importance. It is in fact showing what the impact of each feature from the data set has on the classification model. Table XX shows the feature importance for the tree based classification methods. The Features column represent the name of the feature analysed and the columns AdaBoot, Decision Tree and Random Forest contain the impact of each feature in each method. The higher the number, the more important the feature.

One surprising fact evident in the **Table 20** is that although "ms" feature has quite an impact on the predictive model it is not the most representative.

Table 20: Feature Importance

Features	AdaBoost	Decision Tree	Random Forest
AIS_SENTENCE_LISTENER	0,1	0,193142068	0,153848567
RETRACT_OLD_AISGEOMDATA	0,1	0,000700426	0,005668693
SESSION	0,1	0,00990615	0,032303538
SIMPLE_ANCHOR_IN	0,1	0,052707564	0,196569172
SIMPLE_DOCK_START_OUT	0,1	0,003373742	0,035556067
SIMPLE_DOCK_STOP	0,1	0,091526082	0,208327863
STOP_OVER_IN	0,1	0,526665234	0,194793414
ms	0,3	0,121978734	0,172932687

The last validation experiment was done for Isolation Forest (ISF) unsupervised method. Because we have already labelled data we can run the unsupervised method and see if it identifies the correct anomalies. Of course, Isolation Forest is not able to distinguish between distinct types of anomalies. It can mark events as normal or anomalous however, this is enough to test the ratio of false positives to true positives.

Table 21 shows the performance of Isolation Forest. It shows the total manually labelled anomaly and detected anomalies count, the number of false positives and good anomalies detected. Percentage labelled denotes the percentage of anomalies from the original data set, percentage detected denotes the percentage of ISF detected anomalies. This shows that although ISF didn't detect all the anomalies (15.5 % from the original 22.4 %) it had a relatively small false positive count (58 anomalies yielding an accuracy of 93.4%). During the testing phase, it was evident that the more data you feed to ISF the more accurate it becomes. We tested the method with only 6500 events. It is possible to reduce the error significantly by adding more events.

Table 21: Feature Performance

Metric	CEP
Labelled anomalies	1447
Detected anomalies	999
False positives	58
Good Anomalies	941
Percentage labelled	22,4
Percentage detected	15,5
Accuracy	93,4

During this period, the Anomaly Detection Tool has been validated to detect anomalies related with the cost execution time of the different events that the CEP component analyses, this cost impact directly in the performance of the system

The ISF method did not detect all the anomalies (15.5 % from the original 22.4 %) but it had a relatively small false positive count (accuracy of 93.4%). It is possible in increase the accuracy of the method by considering a bigger set of data.

We achieved quality KPI "False positives", by having a 6,6% of false positives using the Anomaly Detection Tool. More detailed information can be found in **D6.3-Consolidated Implementation and evaluation.**

3.2.4.7 Simulation Tool

The vessels traffic increase for a given port directly affects the Posidonia Operations performance if the system is not properly sized. We have worked with the Simulation Tool and the Monitoring tool to define the dimension of the system and to monitor it in real time.

Analysis of the streaming data is done by a Complex Event Processing engine. This engine can be considered as a "pattern matcher", for each vessel position that arrives it computes different conditions, that when satisfied produce an event. The number of rules (computation) to be applied to each message can affect the overall performance of the system. Actually, the number and implementation of rules vary from one deployment to other. In recent months, a significant effort has been made to improve the quality and the validation of the use case

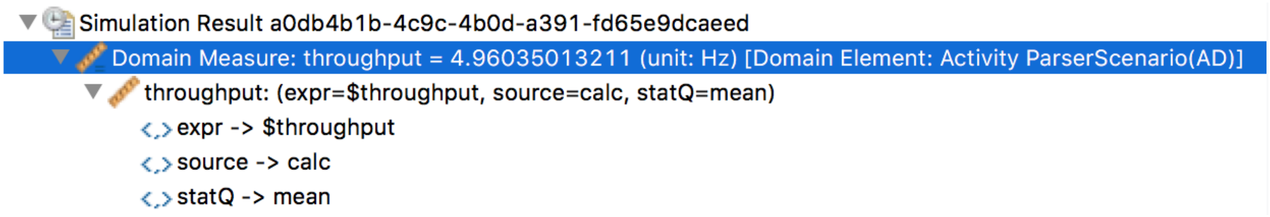
The input to the Simulation Tool is a DPIM model, which provides the relevant magnitudes of the application. In our use case, the relevant magnitudes are the number of CEPs, the number of rules used by each CEP and the computational cost associated to each rule, and the expected input message rate (messages per second).

The Simulation Tool will compute the estimated maximum throughput which can be handled by the application with the parameters specified. If the expected input message rate is higher than the maximum throughput, then the prediction is that the application will not be able to run properly in the expected circumstances.

This tools therefore provides:

- Prediction of the consequences of scaling the application at design time
- Iterative enhancement
- Comparison among multiple configurations

After executing the simulation tool, we get the maximum throughput of the system. In this case the maximum throughputs obtained are equal to the message rate specified as input parameter, so we can conclude that the AIS parser can support the configurations of 5, 6 and 7 messages per second provided:



```
▼ Simulation Result a0db4b1b-4c9c-4b0d-a391-fd65e9dcaeed
  ▼ Domain Measure: throughput = 4.96035013211 (unit: Hz) [Domain Element: Activity ParserScenario(AD)]
    ▼ throughput: (expr=$throughput, source=calc, statQ=mean)
        expr -> $throughput
        source -> calc
        statQ -> mean
```

Figure 35: Simulation system running

Several configurations of the simulation tool can be configured and launched at one time. With the simulation tool is possible to obtain a plot of the predicted throughput for the given configurations (see **Figure 36**).

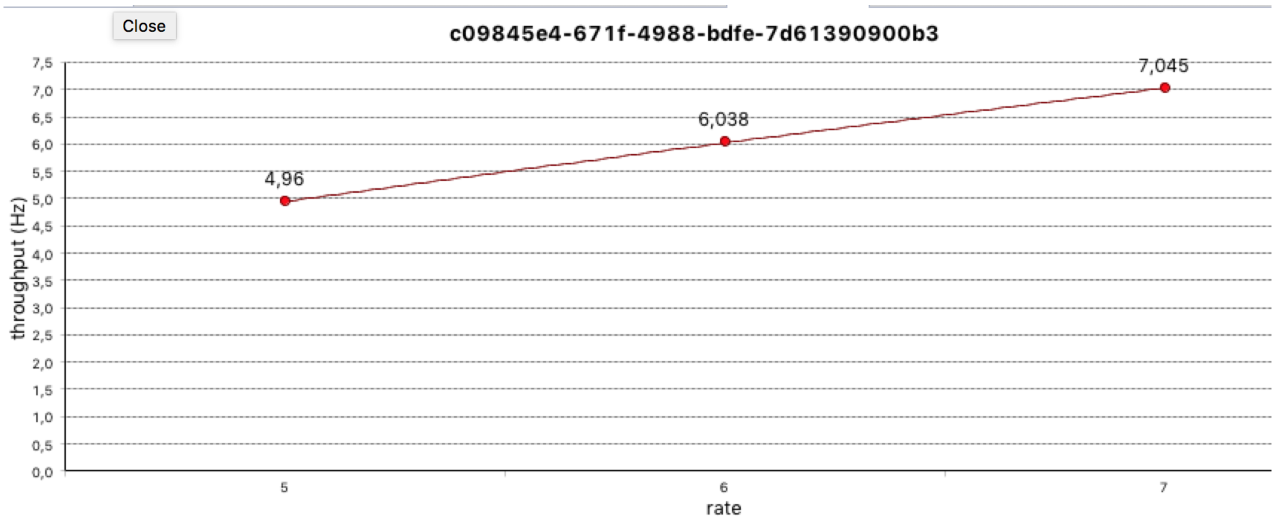


Figure 36: Throughput for different configurations

More detailed information can be found in D6.2-Initial implementation and evaluation.

3.2.4.8 Validation Activities on Containers

Posidonia use case has been used to validate the automatic deployment of the solution with containers. While containers can be used simply to encapsulate and isolate applications in a similar manner to virtual machines, they’re most effective when used as a fundamentally new way of packaging and architecting applications. Instead of large monolithic applications, application design will increasingly use architectures composed of small, single-function “Microservices”, independent services that communicate through network interfaces. This suits agile and DevOps approaches, and reduces the unintended effects associated with making changes in one part of a large monolithic program

To make use of good practices in handling containers, some components have been repackaged in order to better separate the functionality of the different services and have a container for each service "microservices".

Figure 37 contains the description of the Posidonia DDSM model using containers. We have used a mix-approach. The rabbit and the cep components are deployed using virtual machines and the parser component is deploy using containers.

We have selected this approach, because Message Broker and CEP components requires a lot of resources. For these components, it is better to have a full machine instead of using components. In addition to that, the parser component does not need so many resources, which motivates the use of containers for the deployment. This component is internally composed of two services “simulation” and “AIS-parser processor”:

- **Simulation**: The component used to simulate the AIS messages sent by vessels. In the production version, the messages are obtained from a real antenna placed in the port.
- **Parser**: The component that processes all the messages sent by the simulator and publishes the information into the broker.

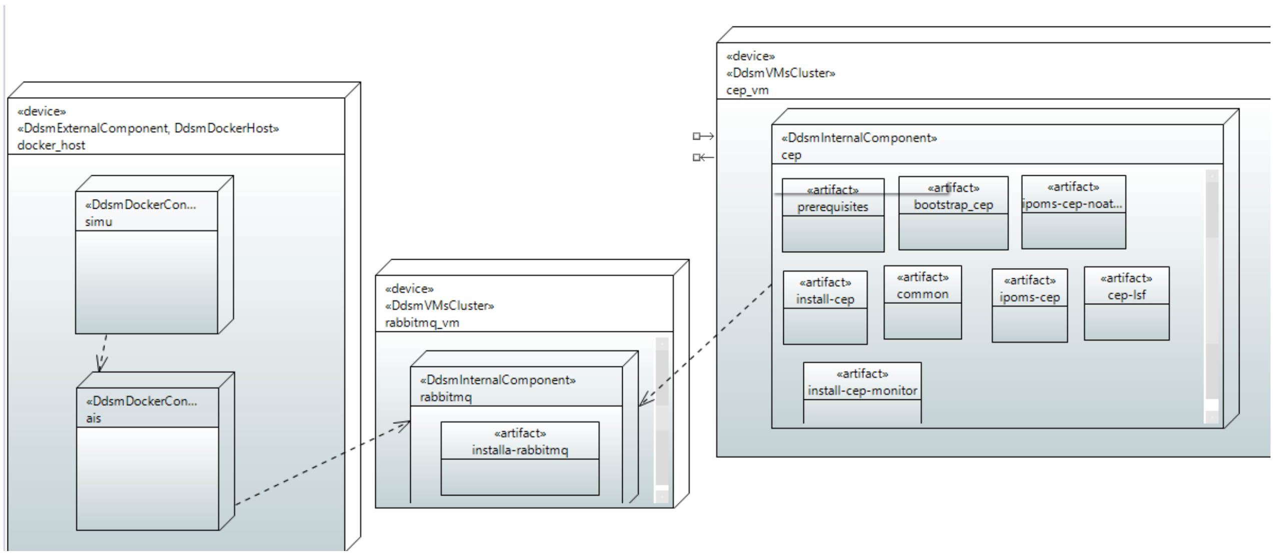


Figure 37: Posidonia DDSM Model using containers

In addition to modifying the model, Prodevelop had to redo the scripts to adapt them to the use of containers. Once the "blueprint" file of the model has been generated by the DICER tool, the deployment has been successfully completed in minutes with the new scripts.

The time required to deploy the solution using containers is between 20-30% less than the time required to deploy the initial solution without containers.

3.2.5 Discussion

After finalizing the use case we have used and validated most of the DICE tools. we can classify the results obtained in four categories:

a) Assessment of the impact in performance after changes in software or conditions

- Predict at design time the impact of changes in the software (number of rules, number of CEPs) and/or conditions (input message rate “Simulation Tool”, CPU overloads “Fault Injection Tool”)
- Detect some bottleneck and anomalies with the use of the Anomaly Detection Tool. In the following months, we will conclude the validation of the Trace Checking Tool and we will able to have more control over the Posidonia use case and detect more bottlenecks and performance issues.

b) Increase the Quality of the system

- Detect punctual performance problems with the use of the Anomaly Detection Tool.
- Detect errors in the CEP component. Detection of loss rules and false rules detection and delays in the detection of the rules compared to the real time in which the event happened.

c) Automatic extraction of relevant KPI

- Easy computation of application execution metrics (generic hardware metrics such as CPU, memory consumption, disk access, etc).

- Easy computation of application-specific metrics which have to do with computational cost of rules, number of messages processed per second, location of events on a map etc.
- Quantification of application performance in terms of percentage of port events correctly detected by the CEP(s).

d) Automation of deployment

- Much faster deployment.
- Possibility of deployment in different cloud providers and configurations.
- Possibility of using containers for the deployment in some components.
- Added new components to the use case that are automatically deployed.
- Automatic configuration of the monitoring tool after deployment.
- Increased number of test deployments for quality testing.
- Easily understandable system configuration due to use of diagrams

In order to validate the use case, some metrics were defined at the beginning of the project. The impact analysis of applying DICE to the use case can be summarized with the following metrics:

Table 22: Prodevelop KPIs Summary

KPI	Result
At most 30% ERROR in the computational cost estimation provided by the Simulation Tool for a given application configuration	This could be computed in a reverse way. For example: The Simulation Tool predicts that the current CEP configuration will be able to handle, at most, 2000 messages per minute. The true maximum input rate should be between 1400 and 2600.
At least 50% REDUCTION in deployment time.	Current time: 10 hours. Time with DICE: 20 minutes Reduction: 97.6%
At least 30% REDUCTION when configuring Monitoring compared to our current solution	Current time: 3 hours. Time with DICE: 10 minutes (almost transparently configured through Delivery Tool) Reduction: 94.5%
At least ONE application-specific quality metric	The metric is the correctness rate (percentage of events correctly detected by the CEP). It is provided by the Trace Checking Tool.
At least FOUR DICE tools used in a profitable way in our use case (apart from DICE profile and DICE IDE)	A set of DICE Tools has been validated in the use case: Monitoring Tool, Trace Checking Tool, Simulation Tool, Delivery Tool, DICER Tool, Anomaly Detection Tool and Fault Injection Tool
<10 % False positives	we achieved quality KPI “False positives”, by having a 6,6% of false positives using the Anomaly Detection Tool.

3.3 NETF Use Case

Within DICE, NETF built a new product (Big Blu) to demonstrate the capabilities of Big Data in e-government applications especially for **Tax fraud detection**. Therefore, compared to the other demonstrators, this use case shows the ability of DICE to support the development of a new application from scratch.

Fraud is a huge industry. It is impossible to quote an accurate cost of fraud, not least because much of it is still undetected, but The European Union has estimated the fiscal loss lost due to tax evasion to be of the order of 1 trillion euros per year. It’s no surprise, therefore, that the governments/organizations most hit by the crime have been stepping up their game in recent years with huge investment in fraud management technology. Many initiatives have been launched both at national and European levels aiming to combat frauds and build various solutions (see https://www.iota-tax.org/ and https://ec.europa.eu/taxation_customs/fight-against-tax-fraud-tax-evasion/missing-part_en).

With the availability of massive amounts of structured and unstructured information, data organization and analytics are a promising solution to find anomalies and uncover violations, and even predicting frauds that may happen in the future. Governments are increasingly using Big Data in multiple sectors to help their agencies manage their operations, and to improve the services they provide to citizens and businesses. In this case, Big Data has the potential to make tax agencies faster and more efficient. Unfortunately, most public organizations are still using legacy information systems that don't support the recent Big Data platforms and where the data is highly duplicated, scattered, even diluted (e.g., resulting from COBOL programming style) without any consolidated view, no correlation, no explicit semantics and with ineffective naming conventions. Documentation is often inadequate or missing. On top of that, the huge amount of digital data must be collected from all sorts of sources including non-classical databases and "flat files", sometimes with odd structuring, such as configuration files and comma-delimited text files.

The point to be made here is that migrating data from such systems into a modern platform is undoubtedly a tedious task, and the process can be risky, expensive and disruptive. Because of those limitations, Big Data plans are often held back by governments and companies. Using Big Data technologies implies an unavoidable step which consists on migrating the legacy database schema and data into a relational database. NETF released Blu Age (www.bluage.com)**,** a modernization tool set that automates the migration of databases and data allowing them to be standardized. This product allows for joint modernization of both the data format and the database schema. Making the transition to NoSQL is more common as more and more businesses are doing the same to keep up with the growing volumes of data to process.

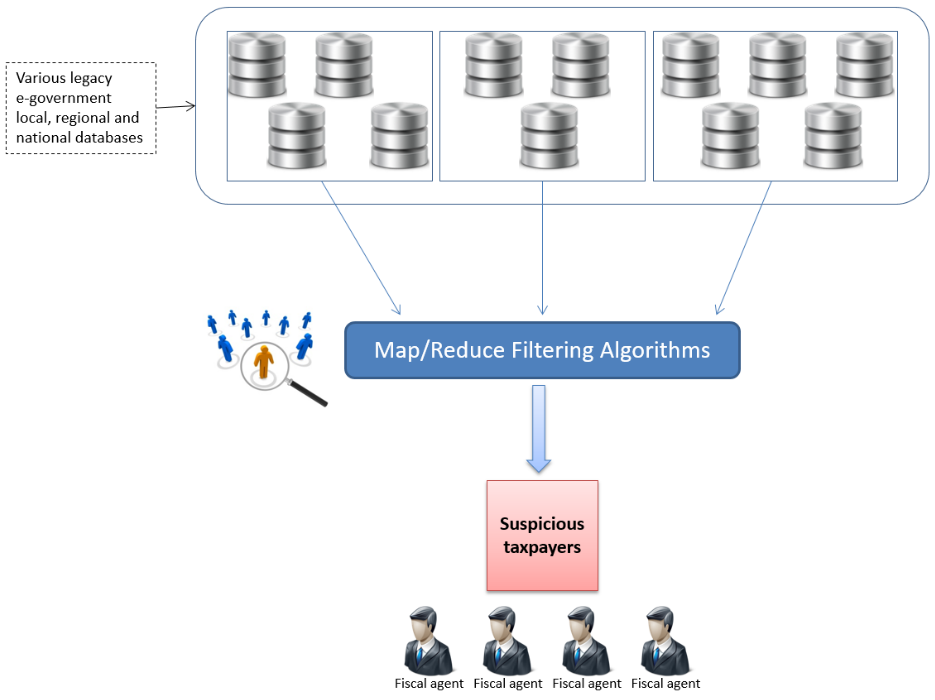

Various legacy e-government local, regional and national databases
Map/Reduce Filtering Algorithms
Suspicious taxpayers
Fiscal agent
Fiscal agent
Fiscal agent
Fiscal agent

Figure 38: Big Blu Concept

Large-scale data analytics is an effective way to detect and understand fraud on a massive scale. Automated systems scan vast customer databases looking for anomalous, unusual or suspicious activity in real time. Large teams of fraud analysts manually review flagged transactions. NETF believes that it's time for governments to start capitalizing on proven Big Data technologies which are already revolutionizing business efficiency across industries from healthcare to education and retail. But Fraud management requires a holistic approach, blending tactical and strategic solutions as with the state-of-the-art technology solutions and best practices in fraud strategy and operations. Traditional fraud detection has not been particularly successful, largely because it happens after the fact. New thinking needs to be injected into the strategies for dealing with fraud. Through the use of cutting-edge technology that brings transactional, analytical, and big data together in real time. Our approach aims to detect deviant behavior much earlier, enabling tax agencies to combat fraud in a much more effective way and at very low costs.

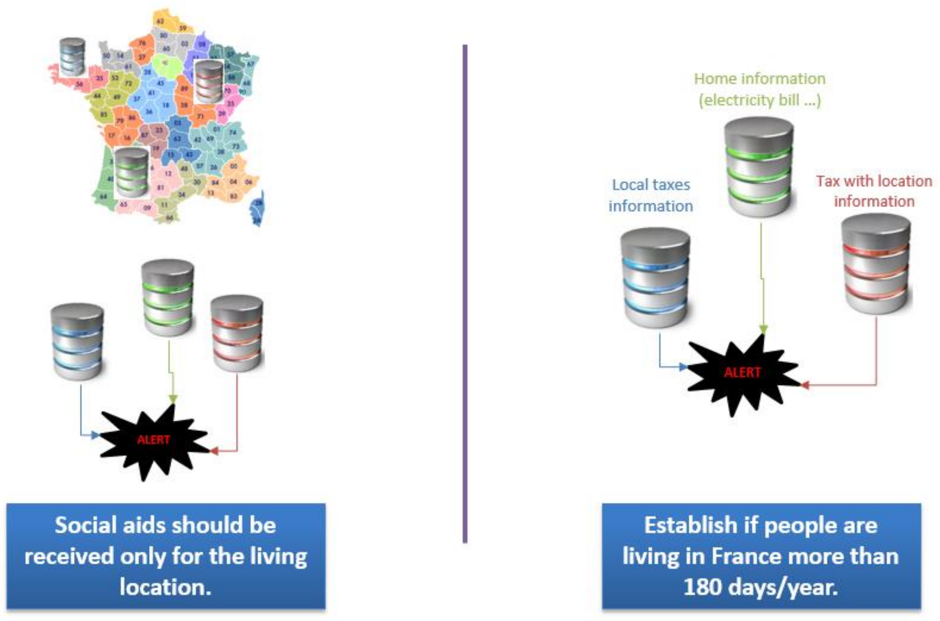

ALERT
Social aids should be received only for the living location.
Home information (electricity bill ...)
Local taxes information
Tax with location information
ALERT
Establish if people are living in France more than 180 days/year.

Figure 39: Big Blu Impact

Within Big Blu, we address both of temporal and physical fraud types. Temporal frauds can be related to changes compared to last years in incomes for example. The physical type is more related to the geographic situation of the data (local, regional or national). Here are two examples of frauds we are able to detect:

- Identifying taxpayers who are registered in different French departments in order to collect fraudulently social aids. For example, Big Blu can catch a nonexistent address or an address that just doesn't compute.
- The second example is related to the fictitious relocation of the taxpayer who improperly claim living abroad in order to not pay tax on income or wealth in France.

There are many other fraud types we studied and will be processed by Big Blu such as VAT pay back, ID theft, etc.

3.3.1 Use Case Scenarios

NETF produced a specific model of the description of a taxpayer. This model is based on the CERFA forms which are publically available on the Web at https://mon.service-public.fr/portail/app/cms/public/les_formulaires. Based on CERFA forms, we were able to build a huge model with hundreds of attributes describing a taxpayer (name, address, ID card number, married/single/..., birthday, car owned, social security number...). This intrinsically raises no privacy/confidentiality issues since the random data will not refer to real persons. NETF product processed millions of instances of the specific model.

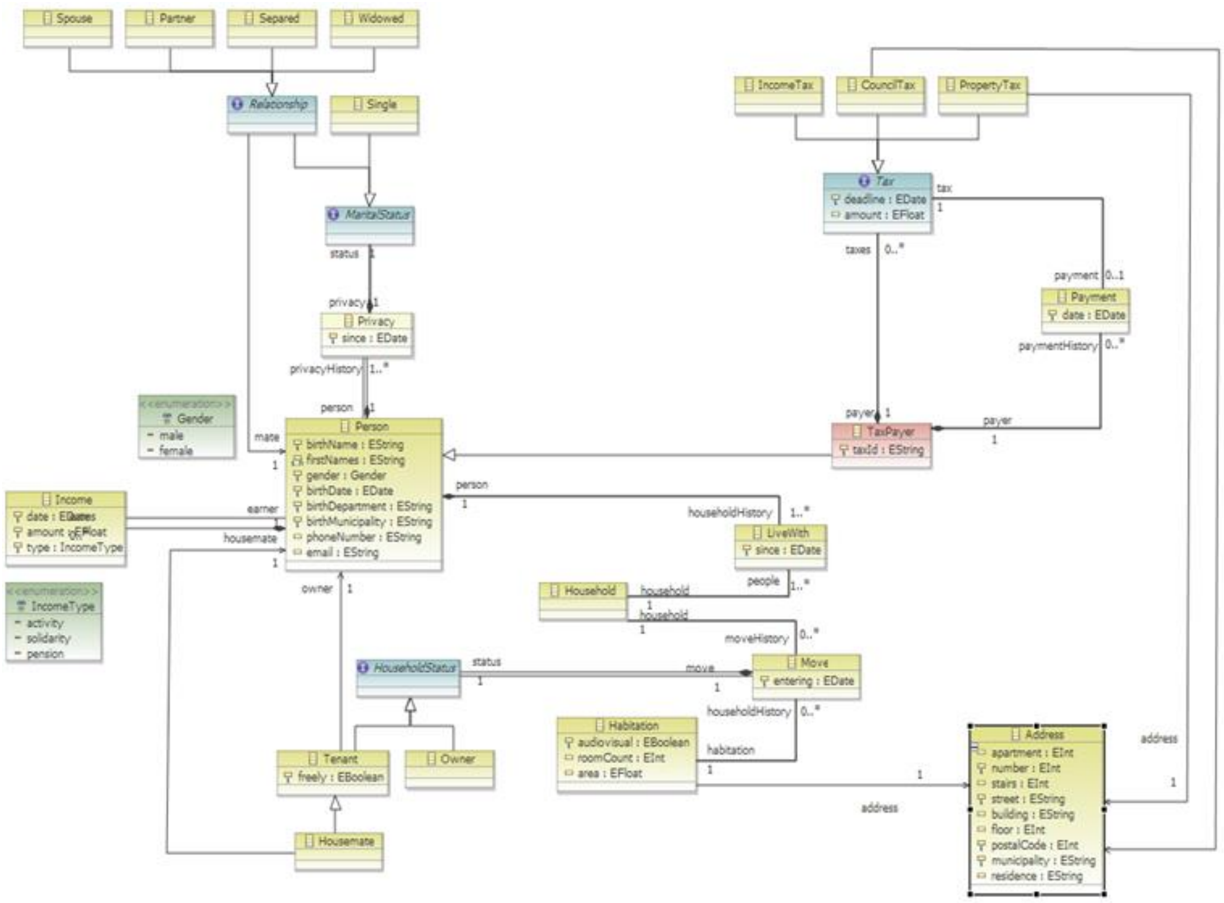


Figure 40: Big Blu Model

All instances were automatically generated using various algorithms. Each algorithm will generate specific text datasets on heterogeneous formats (flat files, legacy proprietary databases, etc). Therefore, it is expected that confidentiality constraints won’t be considered. The focus is the direct valorisation of existing legacy data by deducing new data with richer semantics. This amounts to processing ill-explained financial data whose interpretation may lead to further investigation towards tax fraud management. In this case, identifying fraudulent people and organisations (among the tremendous volume of data to be gather, correlate and queries to execute) is challenging. Big Blu consolidates data from different data stores and provides a technical infrastructure to help users move away from using data mining to visualize information toward forecasting and making up decision between suggested options, as in the spirit of Big Data applications.

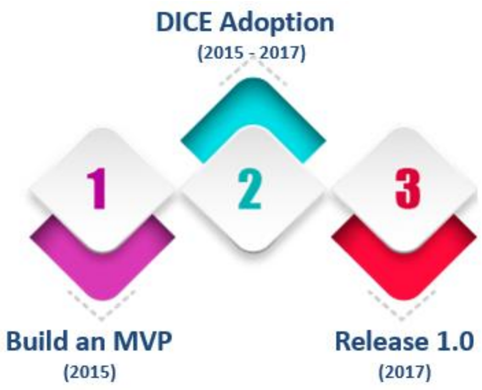


Figure 41: DICE Adoption

NETF built a simple but realistic plan for the development of the use case. This scenario is made of 3 core steps. We started our strategy by building a Minimum Viable Product. This MVP has been developed, deployed and tested on various configurations on a private Cloud. DICE allowed us to accelerate the creation of an initial prototype for Big Blue and make a technical and business case for the viability of the open source technologies used by DICE in the context of tax fraud detection. The success of this initial activity has lead the consortium to identify as one of the main benefits of DICE the possibility to cheaply build proof-of-concepts within organizations for their products. A more in-depth analysis of this has later been developed in D7.7.

The second step of our plan was to test and validate DICE tools within an adoption perspective. This work took place last year and allow us to identify the tools which will accelerate the third and last step which is the release of version 1.0 of Big Blu. This first release is stable and based on a scalable architecture which can be deployed on a production environment.

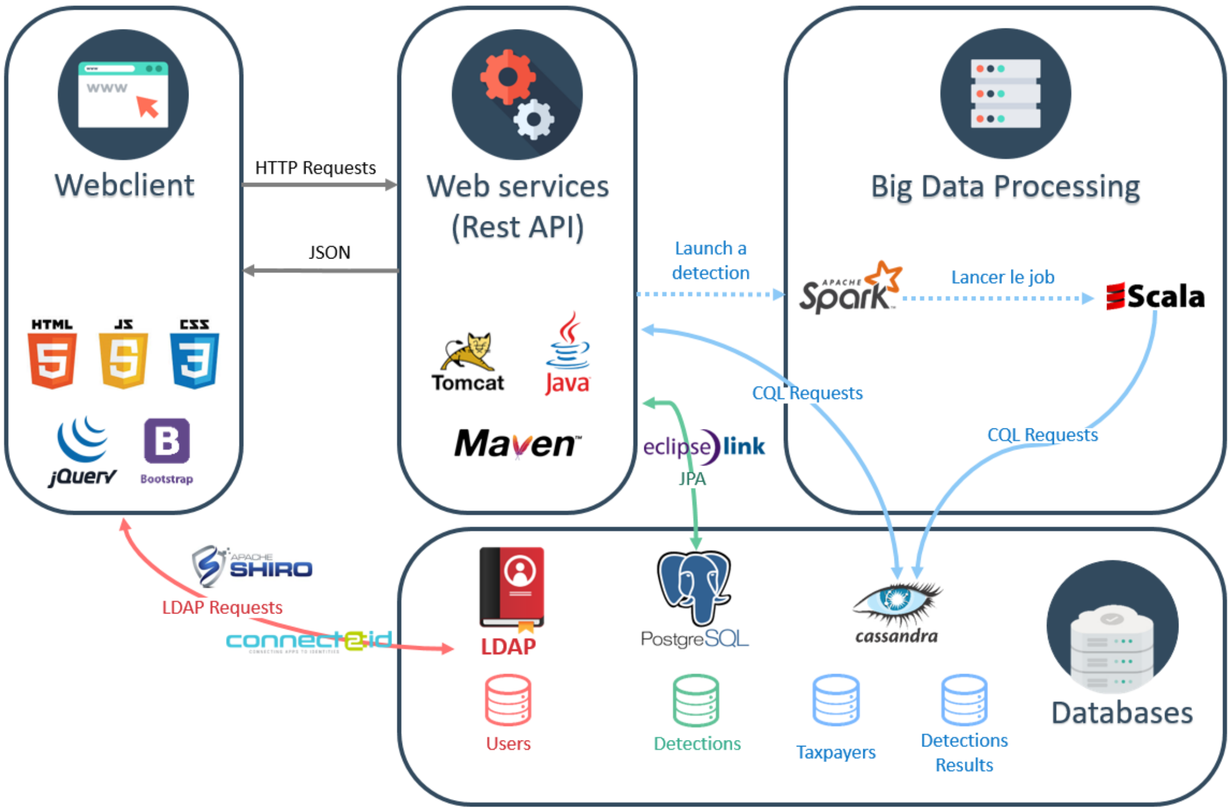
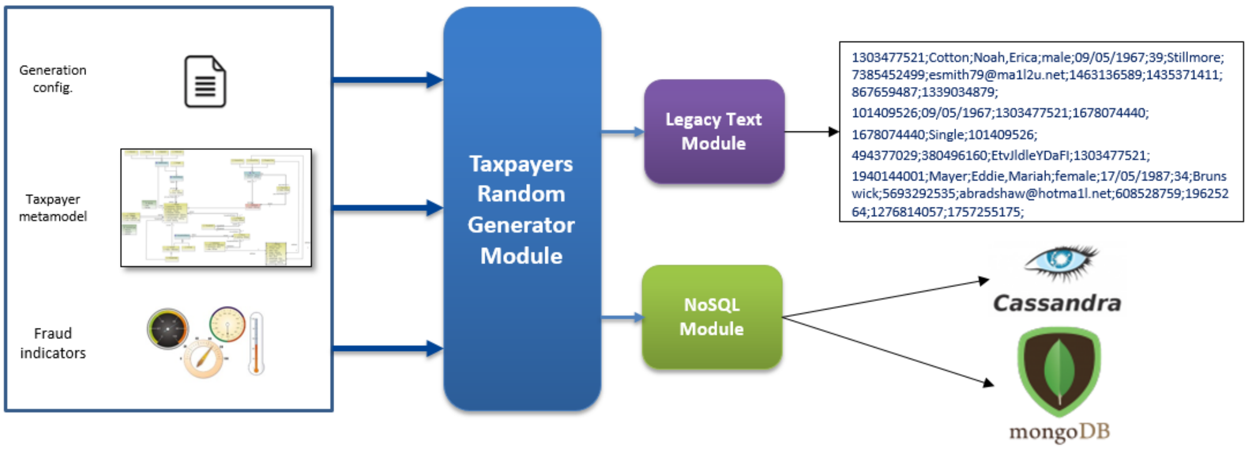


Figure 42: Big Blu Architecture

The initial version of Big Blu which was developed last year has been largely enriched with new features based on a new architectural choices. Big Blu is made of 3 main parts:

- The graphical user interface which is a web application developed using html, css and JavaScript programming languages. We recently added jQuery and Bootstrap in order to enhance the user experience. Our end-users will be tax agents working in various treasury departments. In order to provide them a user-friendly tool, we started building a solution with a rich GUI (Graphical User Interface). This latter includes menus, navigation/exploration pages, configuration tools, etc.
- The big data application itself which is based on Apache Cassandra for data management and Apache Spark for data processing. This part includes the biggest and most important part of the demonstrator. In fact, it includes the core feature of the tool which is in charge of the data processing in order to detect fraudulent conducts.
- And a webservice which makes the glue between the user-interface and the Data-Intensive Application.

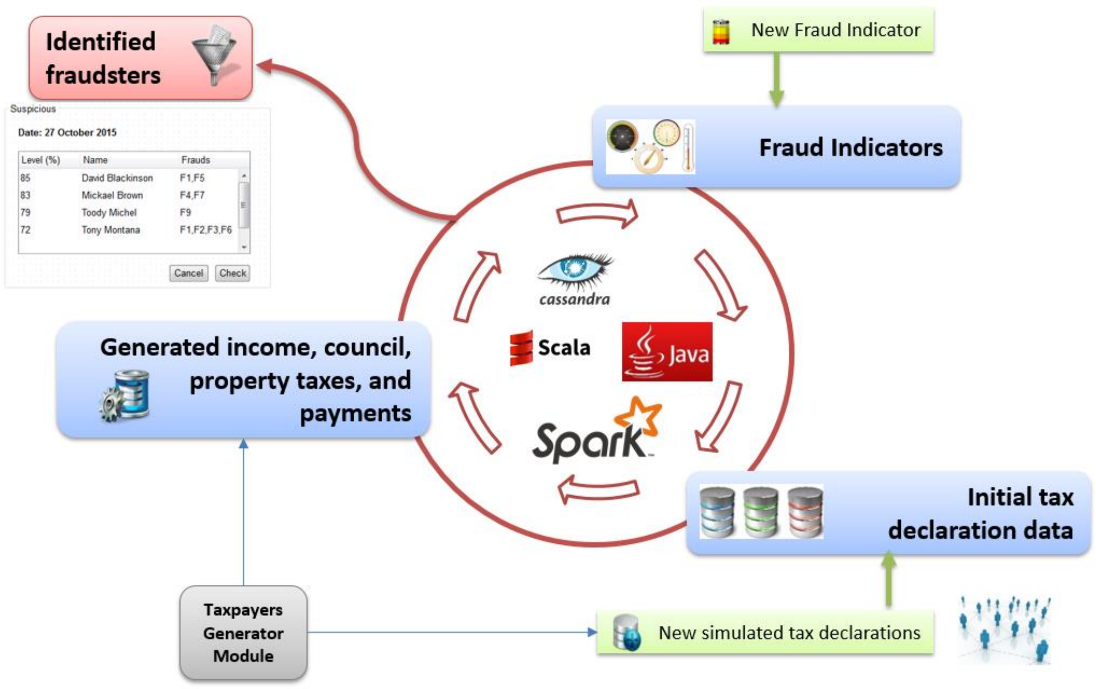
In order to avoid any privacy and/or confidentiality issue, we decided to process imaginary but realistic data. Thus, we implemented a piece of software we called "Taxpayers Random Generator Module" which is able to generate, according to our needs, information describing millions of taxpayers. These data are the main input for the whole product.



```
1303477521;Cotton;Noah,Erica;male;09/05/1967;39;Stillmore;
7385452499;esmith79@ma1l2u.net;1463136589;1435371411;
867659487;1339034879;
101409526;09/05/1967;1303477521;1678074440;
1678074440;Single;101409526;
494377029;380496160;EtvJldleYDaFI;1303477521;
1940144001;Mayer;Eddie,Mariah;female;17/05/1987;34;Bruns
wick;5693292535;abradshaw@hotma1l.net;608528759;196252
64;1276814057;1757255175;
```

Figure 43: Taxpayers Random Generator Module

For our use case, we opt for the Lambda Architecture which is actually a reference architecture for Batch and Real-Time processing in the Big Data ecosystem. This architecture relies on a Data Input Stream which is in our case a Cassandra Cluster filled with various kind of information related to taxpayers. So far, we built this Cluster and are now able to explore and query it to gather details and provide them to the processing unit.


Identified fraudsters
New Fraud Indicator
Fraud Indicators
cassandra
Scala
Java
Spark
Generated income, council, property taxes, and payments
Initial tax declaration data
Taxpayers Generator Module
New simulated tax declarations
Suspicious
Cancel
Check

Date: 27 October 2015

Level (%)	Name	Frauds
85	David Blackinson	F1,F5
83	Mickael Brown	F4,F7
79	Toody Michel	F9
72	Tony Montana	F1,F2,F3,F6

Figure 44: Taxpayers Random Generator Module Architecture

Our Big Data application made of these technologies is continuously running. The Cassandra databases are filled with taxpayers' detail, historical tax declarations, etc. The application will be performing computation on all data including new generated inputs. These data have to be processed using Fraud indicators which are a set of rules described by a domain expert. In the case of a new Fraud Indicator, we have to proceed to a new batch processing phase on all data. But we need to be able to answer any query using a merge between old batch results and new real-time computations. The user will be notified on the graphical user interface with the taxpayers who may be fraudulent. Provided results will mainly have a record of value among others and in no case lead to an automatic condemnation. The demonstrator will just facilitate the task of filtering and gathering data for fiscal agents in order to increase productivity.

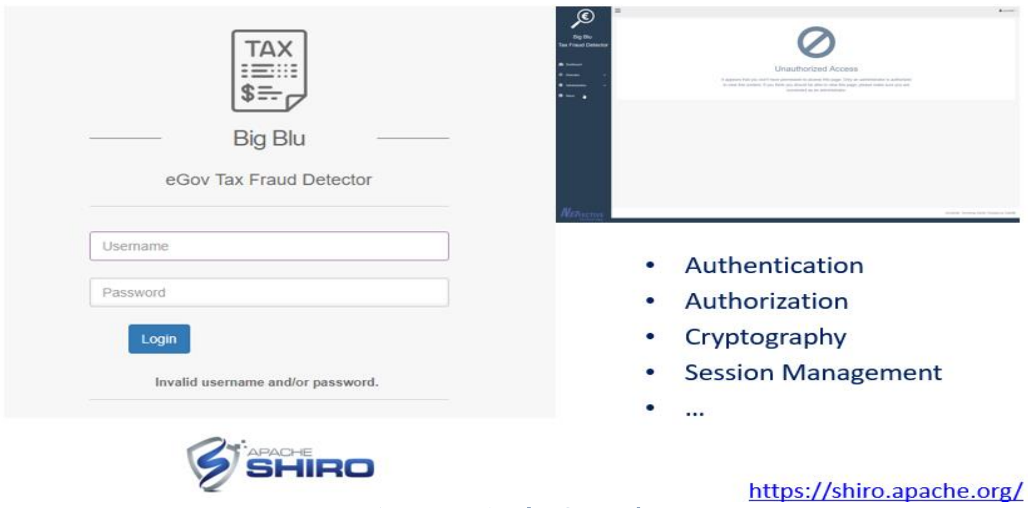


Figure 45: Big Blue Screenshot

The security and privacy is not an option for a product such Big Blu which is supposed to process personal and sensitive data. For this reason, we added an important security layer that performs authentication, authorization, cryptography, and session management for the application users. An administrator is in charge of managing (create/edit/delete) the user accounts. Regarding the privacy, we have been explicitly asked by some potential customers to focus on role specifications. The main goal is to be able to specify different roles according to the tax agent granted permissions. To address this point, we implemented and integrated the SecureUML profile which facilitates the specification of various roles and permissions and brings. This is what have been implemented using Apache Shiro (https://shiro.apache.org/). Moreover the details provided at the model level are used later by the Trace Checking tool while the application is running in order to check whether the authorizations are not violated.

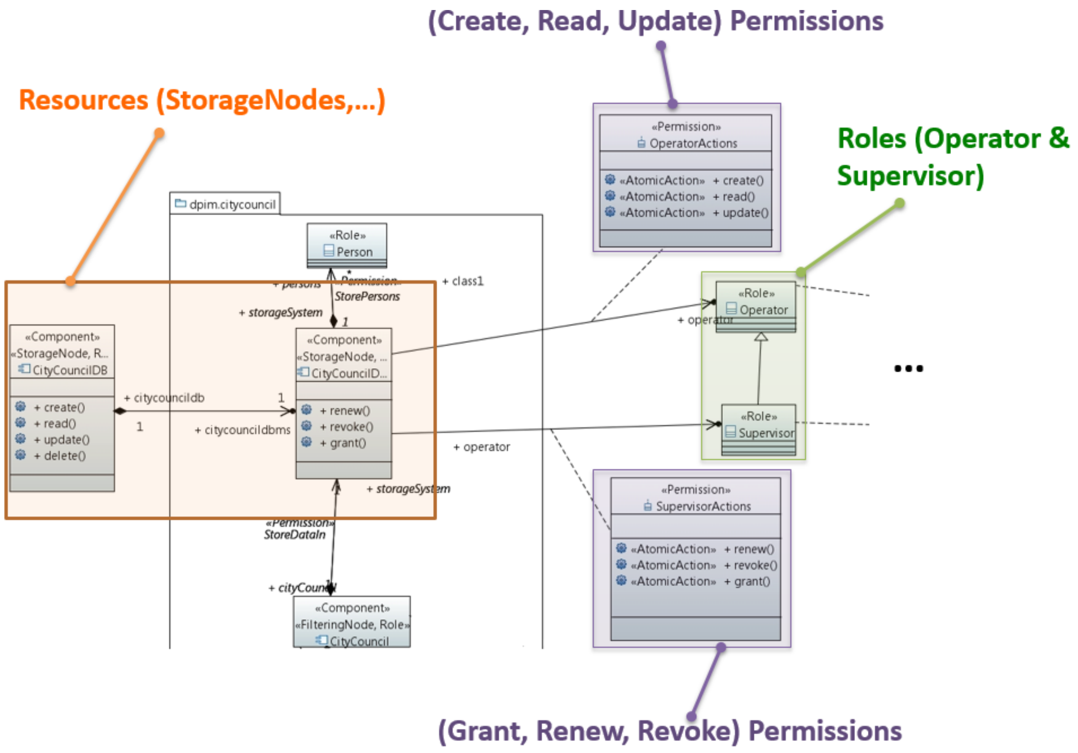
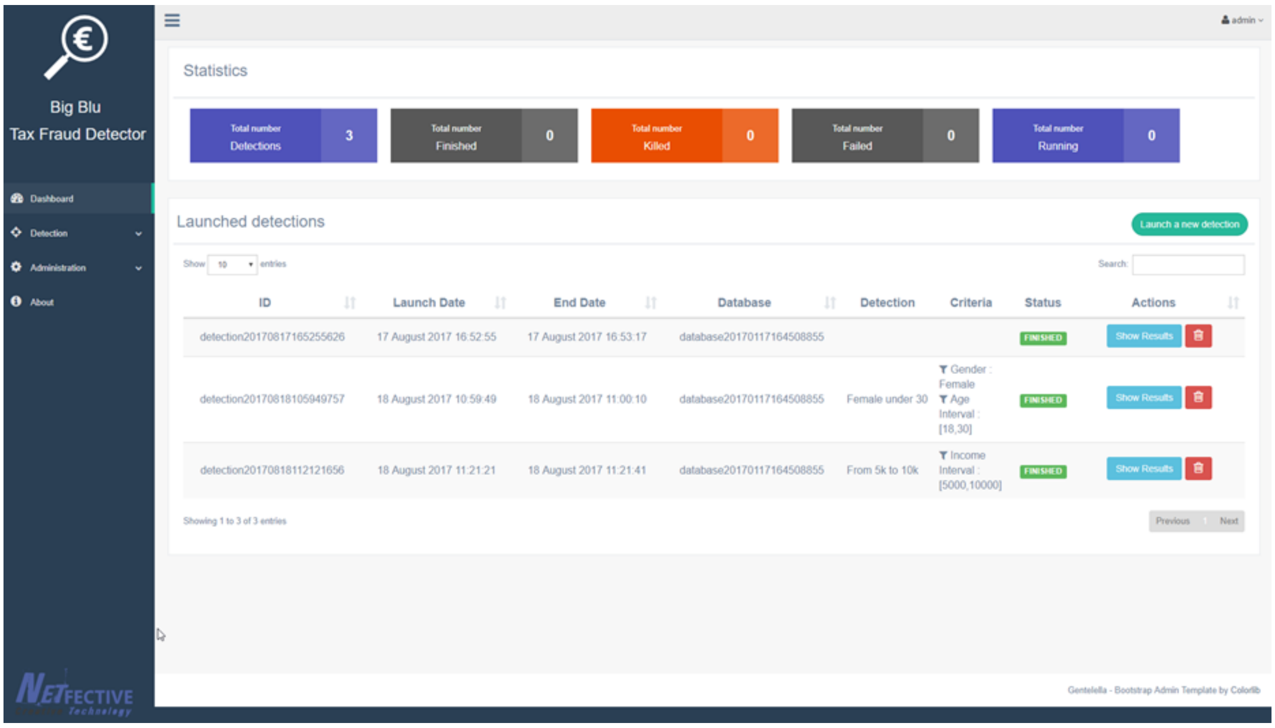

(Create, Read, Update) Permissions
Resources (StorageNodes,...)
Roles (Operator & Supervisor)
«Permission» OperatorActions
«AtomicAction» + create()
«AtomicAction» + read()
«AtomicAction» + update()
dpim.citycouncil
«Role» Person
+ persons
«Permission» StorePersons
+ class1
+ storageSystem
1
«Component» «StorageNode, R... CityCouncilDB
+ create()
+ read()
+ update()
+ delete()
«Component» «StorageNode, ... CityCouncilD...
+ renew()
+ revoke()
+ grant()
+ citycouncildb
1
1
+ citycouncildbms
+ operator
«Role» Operator
«Role» Supervisor
+ operator
+ storageSystem
«Permission» StoreDataIn
+ cityCouncil
«Component» «FilteringNode, Role» CityCouncil
«Permission» SupervisorActions
«AtomicAction» + renew()
«AtomicAction» + revoke()
«AtomicAction» + grant()
...
(Grant, Renew, Revoke) Permissions

Figure 46: Big Blu Model

In the following section, we will present some screenshots and discuss the main features offered by Big Blu.



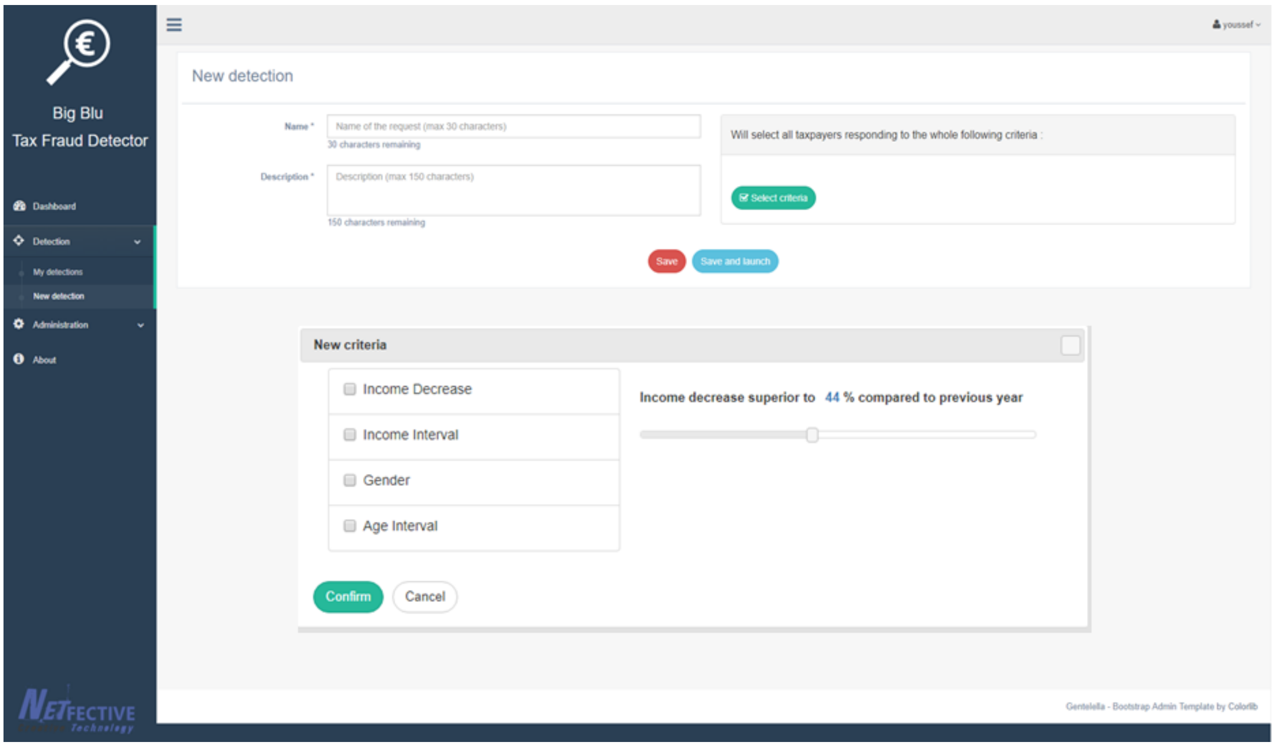
Launched detections

ID	Launch Date	End Date	Database	Detection	Criteria	Status	Actions
detection20170817165255626	17 August 2017 16:52:55	17 August 2017 16:53:17	database20170117164508855			FINISHED	Show Results
detection20170818105949757	18 August 2017 10:59:49	18 August 2017 11:00:10	database20170117164508855	Female under 30	Gender : Female Age Interval : [18,30]	FINISHED	Show Results
detection20170818112121656	18 August 2017 11:21:21	18 August 2017 11:21:41	database20170117164508855	From 5k to 10k	Income Interval : [5000,10000]	FINISHED	Show Results

Showing 1 to 3 of 3 entries

Figure 47: Big Blu Screenshot 1

This screen allows the user to launch a new fraud detection and see all the launched detections, check their status, access to their results and see main statistics. This is a private environment where users can only see their own detections.



New detection

Name * Name of the request (max 30 characters)
30 characters remaining

Description * Description (max 150 characters)
150 characters remaining

Will select all taxpayers responding to the whole following criteria :

New criteria

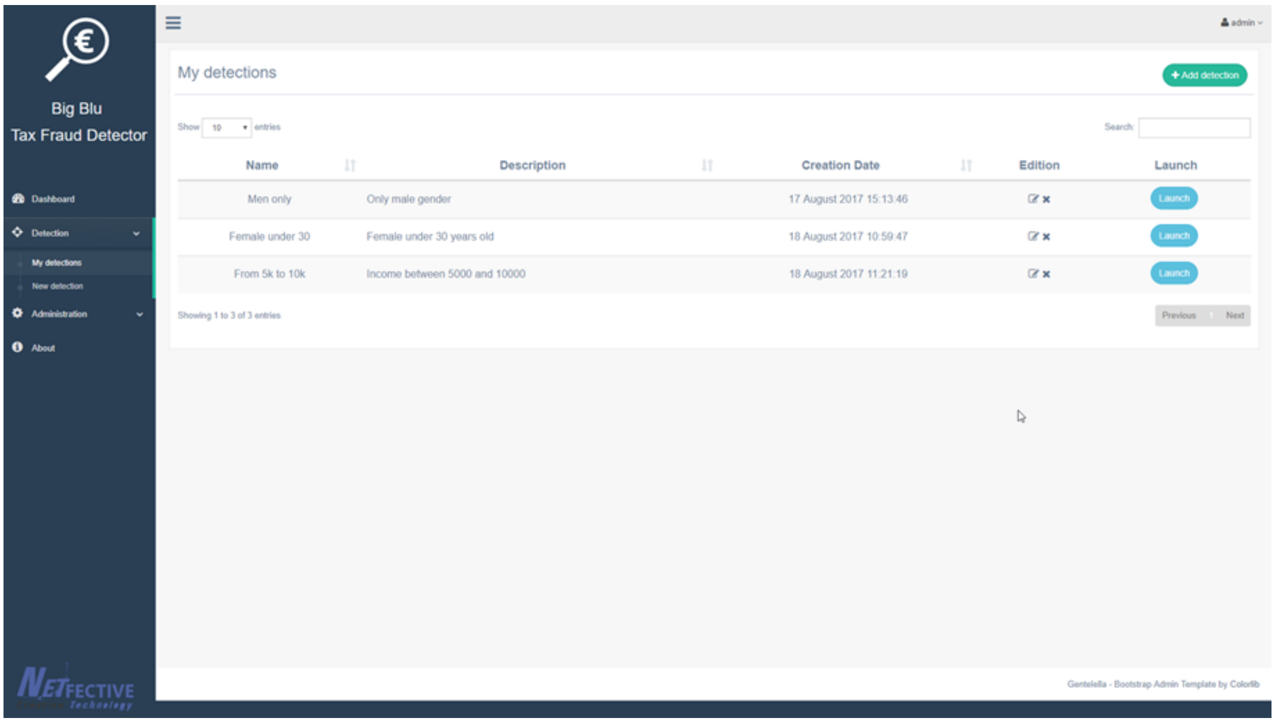
- Income Decrease
- Income Interval
- Gender
- Age Interval

Income decrease superior to 44 % compared to previous year

Figure 48: Big Blu Screenshot 2

In the last version of Big Blu, we give the possibility to users to create multiple detections. In fact, after multiple discussions with some potential customers, we decided to completely reconsider the way Fraud Indicators (FI) were initially managed. In fact, in the initial version of the prototype we built "static" Fraud

Indicators which cannot be enriched and/or personalized by end-users. This initial implementation was useful in order to validate the whole idea and the underlying technologies. But it has limitations mainly because it needs new development (advanced users) in order to add new FI. The new version of Big Blu allows users to build their own requests using all the available data about taxpayers in the database. Big Blu is a kind of requests engine.



My detections

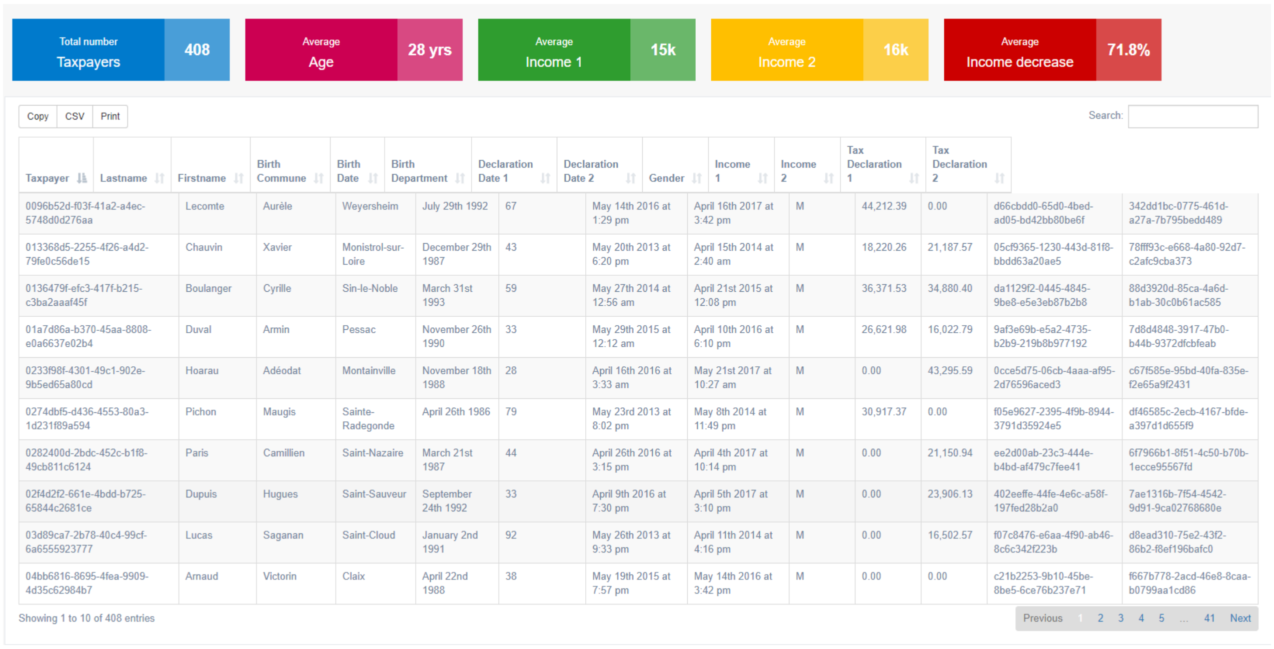
Name	Description	Creation Date	Edition	Launch
Men only	Only male gender	17 August 2017 15:13:46		Launch
Female under 30	Female under 30 years old	18 August 2017 10:59:47		Launch
From 5k to 10k	Income between 5000 and 10000	18 August 2017 11:21:19		Launch

Showing 1 to 3 of 3 entries

Previous Next

Figure 49: Big Blu Screenshot 3

Compared to the initial version, NETF completely changed the way detections are managed in order to let end-users customize their queries and build Fraud Indicators according to specific needs.



Taxpayer	Lastname	Firstname	Birth Commune	Birth Date	Birth Department	Declaration Date 1	Declaration Date 2	Gender	Income 1	Income 2	Tax Declaration 1	Tax Declaration 2	
0096b52d-f03f-41a2-a4ec-5748d0d276aa	Lecomte	Aurèle	Weyersheim	July 29th 1992	67	May 14th 2016 at 1:29 pm	April 16th 2017 at 3:42 pm	M	44,212.39	0.00	d66cbdd0-65d0-4bed-ad05-bd42bb80be6f	342dd1bc-0775-461d-a27a-7b795bedd489	
013368d5-2255-4f26-a4d2-79fe0c56de15	Chauvin	Xavier	Monistrol-sur-Loire	December 29th 1987	43	May 20th 2013 at 6:20 pm	April 15th 2014 at 2:40 am	M	18,220.26	21,187.57	05cf9365-1230-443d-81f8-bbdd63a20ae5	78fff93c-e668-4a80-92d7-c2afc9cba373	
0136479f-efc3-417f-b215-c3ba2aaaf45f	Boulanger	Cyrille	Sin-le-Noble	March 31st 1993	59	May 27th 2014 at 12:56 am	April 21st 2015 at 12:08 pm	M	36,371.53	34,880.40	da1129f2-0445-4845-9be8-e5e3eb87b2b8	88d3920d-85ca-4a6d-b1ab-30c0b61ac585	
01a7d86a-b370-45aa-8808-e0a6637e02b4	Duval	Armin	Pessac	November 26th 1990	33	May 29th 2015 at 12:12 am	April 10th 2016 at 6:10 pm	M	26,621.98	16,022.79	9af3e69b-e5a2-4735-b2b9-219b8b977192	7d8d4848-3917-47b0-b44b-9372dfcbfeab	
0233f98f-4301-49c1-902e-9b5ed65a80cd	Hoarau	Adéodat	Montainville	November 18th 1988	28	April 16th 2016 at 3:33 am	May 21st 2017 at 10:27 am	M	0.00	43,295.59	0cce5d75-06cb-4aaa-af95-2d76596aced3	c67f585e-95bd-40fa-835e-f2e65a9f2431	
0274dbf5-d436-4553-80a3-1d231f89a594	Pichon	Maugis	Sainte-Radegonde	April 26th 1986	79	May 23rd 2013 at 8:02 pm	May 8th 2014 at 11:49 pm	M	30,917.37	0.00	f05e9627-2395-4f9b-8944-3791d35924e5	df46585c-2ecb-4167-bfde-a397d1d655f9	
0282400d-2bdc-452c-b1f8-49cb811c6124	Paris	Camillien	Saint-Nazaire	March 21st 1987	44	April 26th 2016 at 3:15 pm	April 4th 2017 at 10:14 pm	M	0.00	21,150.94	ee2d00ab-23c3-444e-b4bd-af479c7fee41	6f7966b1-8f51-4c50-b70b-1ecce95567fd	
02f4d2f2-661e-4bdd-b725-65844c2681ce	Dupuis	Hugues	Saint-Sauveur	September 24th 1992	33	April 9th 2016 at 7:30 pm	April 5th 2017 at 3:10 pm	M	0.00	23,906.13	402eeffe-44fe-4e6c-a58f-197fed28b2a0	7ae1316b-7f54-4542-9d91-9ca02768680e	
03d89ca7-2b78-40c4-99cf-6a6555923777	Lucas	Saganan	Saint-Cloud	January 2nd 1991	92	May 26th 2013 at 9:33 pm	April 11th 2014 at 4:16 pm	M	0.00	16,502.57	f07c8476-e6aa-4f90-ab46-8c6c342f223b	d8ead310-75e2-43f2-86b2-f8ef196bafc0	
04bb6816-8695-4fea-9909-4d35c62984b7	Arnaud	Victorin	Claix	April 22nd 1988	38	May 19th 2015 at 7:57 pm	May 14th 2016 at 3:42 pm	M	0.00	0.00	c21b2253-9b10-45be-8be5-6ce76b237e71	f667b778-2acd-46e8-8caa-b0799aa1cd86	

Showing 1 to 10 of 408 entries

Previous 1 2 3 4 5 … 41 Next

Figure 50: Big Blu Screenshot 4

Provided results will mainly have a record of value among others and in no case lead to an automatic condemnation. Big Blu will just facilitate the task of filtering and gathering data for fiscal agents in order to increase productivity.

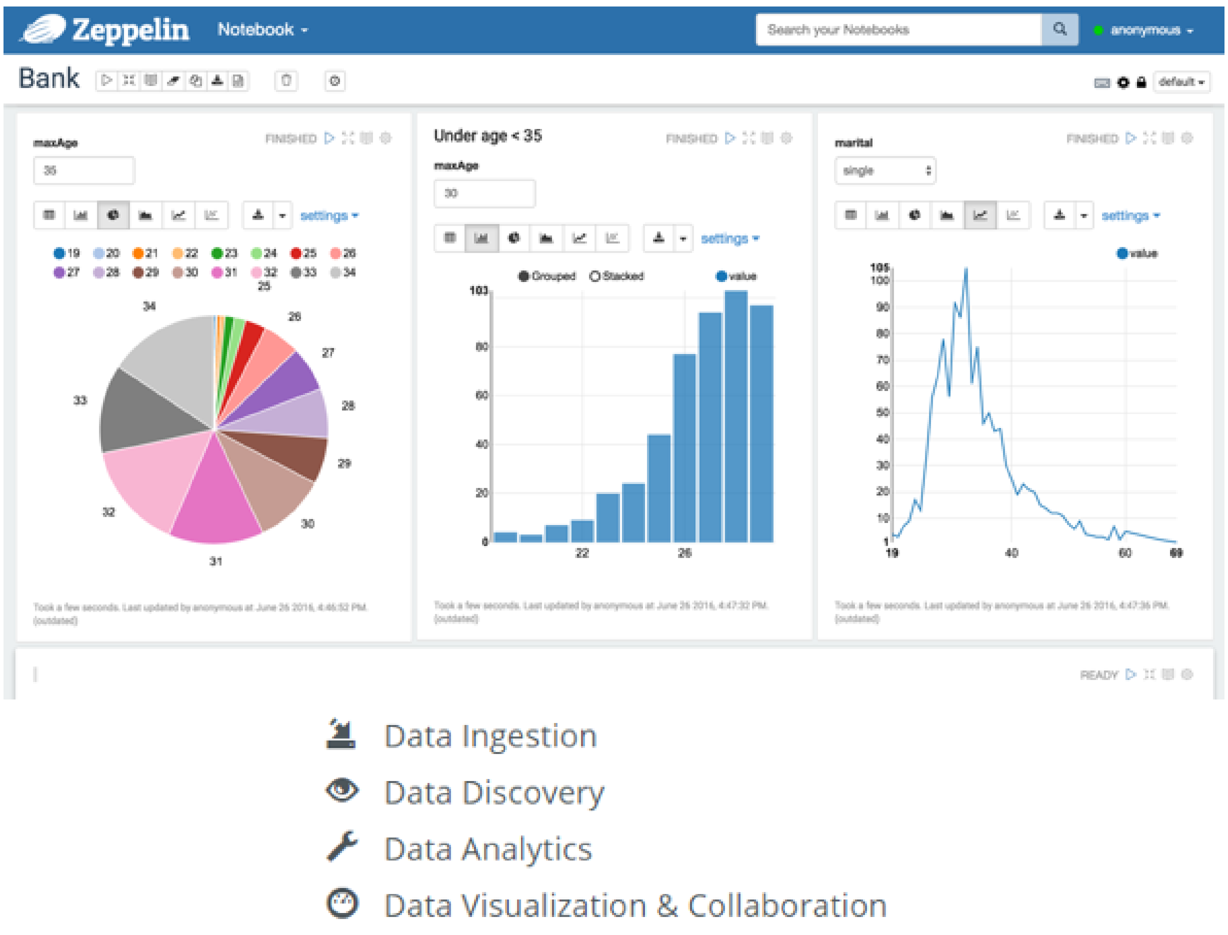


Figure 51: Apache Zeppelin

We decided to push the Fraud Management further and propose an advanced ecosystem in which advanced users can write and validate new detections without coding. For this feature, we used the open source Apache Zeppelin (https://zeppelin.apache.org/) environment where the user can query various Big Data databases for instance Cassandra and visualize the results. Moreover Machine Learning algorithms can be tested on data subsets before launching massive data processings on the whole dataset.

3.3.2 Validation & Impact analysis

An important point to underline is that Big Blu was developed from scratch during the DICE project. It means that NETF had not an existing application which will be migrated to a Big Data environment using the DICE ecosystem. The target market of NETF is emerging and to the best of our knowledge, there are no existing solutions similar to our solution. In fact, e-government applications are mainly based on legacy systems which need to be modernized before being able to make benefits of any Big Data related technologies. Actually, modernizing such software is the primary market of NETF and we are investigating to enrich our offer by proposing to our partners a direct valorization of the modernized data by using our tax fraud detection product.

The initial objectives of our product were mainly related to the adoption of the Big Data ecosystem which is completely new for us. NETF made a significant effort in going forward in the product realization and in fact we have already identified the appropriate architecture, frameworks and technologies. We also started the implementation of the prototype and are already looking for applying the results to be able to detect simple but actual frauds. That’s said, a survey recently made by Information Week shows that the main barrier SMEs are facing about using Big Data Software is that the expertise is scarce and expensive. Hadoop, MapReduce and Cassandra are not point-and-click technologies. There is quite a bit of Linux configuration, some Java coding and a set of frameworks to make smoothly work together. Unless you get hands-on experience with each of those parts in a use-case context, the climb will be steep. In fact, while working on the initial version of Big Blu, we struggled with some technical issues mainly related to specificites of the Big Data ecosystem. Thus we need concrete solutions unless we won't be able to move from an MVP to a first release. While building our solution we have faced some technical and methodological issues. We report below, a list of 4 core questions:

- How to build a reliable architecture?
- How to test different cluster configurations?
- How to accelerate my Code-Build-Deploy-Run cycle?
- How to monitor my runtime environment?

Actually some DICE tools are partially or completely tackling these issues. DICE is relieving users of a big part of this burden by proposing a set of tools in order to facilitate the adoption of Big Data technologies. In the following sections, we explain how the tools have concretly helped us to answer the questions listed above. If the reader is interested in getting more details, we recommend to see the public deliverables available online at www.dice-h2020.eu.

3.3.2.1 Simulation Tool

So to the question "How to build a reliable architecture?", the simulation tool proposes a user-friendly approach to validate an architecture at early-stage at very low cost in terms of time spent in implementation and operations. For example, given a set of requirements for a tax agency, we can identify and create optimal architecture or evaluate alternatives and measure the impact of business logic changes. For instance we have identified at which requests rate, we may face time response issues and set up 2 potential solutions which can be automatically deployed if needed. These models can also be used for documentation purpose since they are at the DPIM level which means contains all the application business logic. More detailed information about this validation can be found in **D6.3-Consolidated Implementation and evaluation.**

3.3.2.2 Enhancement Tool

The simulation tool can take as input UML models automatically adjusted by the Enhancement tools and especially DICE-FG tool which can add resource usage and breakdown information gathered directly from the production environment or the Cloud where the data-intensive software is running. In the case of NETF's application, DICE-FG considers monitoring data gathered during the execution of Spark jobs as its basis to improve the models. More detailed information about this validation can be found in **D6.3-Consolidated Implementation and evaluation.**

3.3.2.3 DICER & Delivery Tools

One of the most time consuming tasks while prototyping a Big Data application is building clusters (designing and deploying the infrastructure) since the engineer must manually install all the needed services, frameworks, etc. This task may need to be repeated several times before obtaining the most suitable cluster configuration in terms of cost, performance, etc. This is exactly the issue addressed by DICER which allows the users to create throw-away clusters for fast prototyping, or persist the ones that prove useful in the form of a model at the DDSM level. In addition, based on this model, DICER automatically generate the TOSCA blueprints to be automatically deployed. And this is exactly the role of the deployment tool which takes a TOSCA blueprint (generated by DICER or not) as an input and automatically install the required frameworks, start the services and deploy the Data Intensive Application. For this reason, this operation can be repeated at a very low cost each time the developer needs to test a new cluster configuration or a new version of the Big Data application.

NETF started working on Big Blu before the release of the DICE tools. So in order to test our application, we manually built our clusters using command lines and scripts. This activity was conducted by one engineer fully dedicated to this task. We estimate the time spent for 63 hours (approximately 9 full working days). The adoption of DICER and the deployment tool combined together allows to reach a high level of productivity. In fact, we were able to design and deploy the same cluster configuration within few hours (~6 hours compared to 9 days). The engineer who made this exercise was not expert in the manual process neither in DICER and the deployment tool.

Table 23: DICER & Deployment Tools Validation Results 1

Tested approach	Time spent (hours)
Manual	63
DICER + Deployment	6 (**Ratio = x10.5**)

Few months after the initial comparison, we did again the same installations on another cloud using a new cluster configuration. And we obtained different values mainly explained by the gained experience on the various tools. Although it is obvious that the user is able to rapidly design and deploy a cluster thanks to the DICE tools. In a concrete scenario, we will probably build our own scripts in order to move from a fully automated approach to a semi-automated one with built-in house tools. The main advantages from our point of view is the ability to create throw-away clusters for fast prototyping using various technologies and frameworks.

Table 24: DICER & Deployment Tools Validation Results 2

Tested approach	Time spent (hours)
Manual	18
DICER + Deployment	2 (**Ratio = x9**)

Recently we initiated a new project for which our Ops team insisted to use a proprietary solution. We made the same exercise and we obtained these results which can be mainly explained by the fact that this tool is not dedicated to Big Data so if the end-user needs to install Cassandra, she or he have to manage to provide all the installation scripts. More detailed information about this validation can be found in **D6.2-Initial implementation and evaluation**.

3.3.2.4 Optimization Tool

NETF collaborated with PMI to employ the Optimization Tool (D-SPACE4Cloud) to examine the impact of securing its database on the performance of Spark detection jobs. Anonymization and encryption of data were the two approaches adopted to secure the database. D-SPACE4Cloud allowed us to measure the costs for each technique. We realised that, even if from one side encryption add CPU overhead to the system, in some situations encryption resulted even in a performance improvement. D-SPACE4Cloud assisted us to get an insight into the effects of security regarding Spark jobs. D-SPACE4Cloud has been validated by the NETF case study. Three fraud indicators have been considered at different scale (10 and 30 millions of taxpayers) implementing also privacy mechanisms. Across the six cases analysed, D-SPACE4Cloud identified the correct cluster configuration (minimum number of nodes fulfilling the deadline) for five of them. In only one case, the error was of a single VM with 17% additional cost. The average percentage error cost estimate overall was 3%. More detailed information about this validation can be found in **D6.3-Consolidated Implementation and evaluation.**

3.3.2.5 Monitoring Tool

In order to validate and measure the robustness of our application, we need to supervise a plethora of metrics. Some are related to the application itself but others are related to the underlying frameworks. In order to monitor our runtime environment, we have 2 options: either install a dedicated tool per framework (if it exists) or install the DICE monitoring tool within few minutes and get all the required metrics. More than that, most of the other DICE tools offer the possibility to automatically exploit such metrics.

3.3.2.6 Fault Injection Tool

The fault injection tool (FIT) has been adopted in order to make sure that the Cassandra replication is working correctly so failed nodes can be replaced with no downtime. In fact, FIT generates faults within Virtual Machines and at the Cloud Provider Level. The purpose of the FIT is to test the resiliency of a cloud installation as an application target. With this approach, the designers can use robust testing, showing where to harden the application before it reaches a commercial environment and allows us (as application owners) to test and stress Big Blu design/deployment in the event of a cloud failure or outage. Thus, allowing for the mitigation of risk in advance of a cloud based deployment.

3.3.3 Discussion

NETF has tested and validated a part of the DICE tools. We are actually convinced of their added value mainly for early stage validation, process automation and monitoring. We think that a great work has been done not only for the implementation of the tools but also for in terms of user experience and user-friendliness. This latter was made at the level of each tool but also at the IDE level, for example in terms of scenario workflows.

4 Conclusion

The current deliverable provides an overview of the validation activities developed as part of the DICE project both on laboratory and on real life cases (use cases), to be disseminated publicly. Material from previous “confidential” deliverables has been refined and presented in a relatively simplistic manner for the average reader, while emphasis has been given to results obtained over the last 6 months of the project.

All DICE tools have been validated while KPIs have been respected and met while in some cases, these KPIs have been surpassed as obtained results proved to be much higher than the ones predicted in the original proposal.

Especially regarding the productivity of the DevOps team of a Data Intensive Application, this has been increased by far. In fact, by using DICE tools, the time needed for implementing a number of activities and deployments, has been decreased significantly, thus allowing engineers and developers to be more efficient and shift resources to parallel activities. From the managerial point of view, the DICE tools allow the team to achieve the same results in less time and with cheaper labour costs (i.e. junior engineers vs senior engineers), which is translated into significant savings for a company.

5 References

[1] The DICE Consortium. DICE Optimization Tools - Initial Version. Tech. rep. European Union's Horizon 2020 research and innovation programme, 2016.

[2] The DICE Consortium. Transformations to Analysis Models. Tech. rep. European Union's Horizon 2020 research and innovation programme, 2016.

[3] Danilo Ardagna et al. "Modeling Performance of Hadoop Applications: A Journey from Queueing Networks to Stochastic Well Formed Nets". Paper submitted for publication on the 16th International Conference on Algorithms and Architectures for Parallel Processing. 2016.

[4] Apache Hive. URL : https://hive.apache.org

[5] José Ignacio Requeno et al. Performance Analysis of Apache Storm Applications using Stochastic Petri Nets. In Proceedings 5th IEEE International Workshop on Formal Methods Integration, FMI 2017, San Diego, CA, 2017. IEEE Press.

[6] The DICE Consortium. DICE Simulation Tools - Final Version. Tech. rep. European Union's Horizon 2020 research and innovation programme, 2017.

[7] Databricks. Spark-perf. URL : https://github.com/databricks/spark-perf

[8] Flexiant Cloud Orchestator. URL : https://www.flexiant.com

[9] The DICE Consortium. DICE Verification Tool - Initial Version. Tech. rep. European Union's Horizon 2020 research and innovation programme, 2016.

[10] Marconi F, Bersani MM, Erascu M, Rossi M. Towards the formal verification of data-intensive applications through metric temporal logic. In International Conference on Formal Engineering Methods 2016 Nov 14 (pp. 193-209). Springer International Publishing.

[11] Marconi, F., Bersani, M.M. and Rossi, M., 2017, April. Formal verification of storm topologies through D-VerT. In Proceedings of the Symposium on Applied Computing (pp. 1168-1174). ACM.

[12] Marconi, F., Bersani, M.M. and Rossi, M., 2017. A model-driven approach for the formal verification of storm-based streaming applications. ACM SIGAPP Applied Computing Review. 17. 6-15. ACM.

[13] The DICE Consortium. DICE Verification Tool - Intermediate Version. Tech. rep. European Union's Horizon 2020 research and innovation programme, 2017.

[14] The DICE Consortium. DICE Verification Tool - Final Version. Tech. rep. European Union's Horizon 2020 research and innovation programme, 2017.

[15] Marconi, F., Quattrocchi, G., Baresi, L., Bersani, M.M., Rossi, M.: *On the timed analysis of big-data applications.* Accepted in Proc. of Nasa Formal Methods, 2018.