EUROPEAN COMMISSION

pulse

Platform for European Medical Support During Major Emergencies

D6.1 Integration Infrastructure Description

skytek · ESS Centre for European Security Strategies · ONEST SOLUTIONS · TRILATERAL RESEARCH · UNIVERSITÀ CATTOLICA del Sacro Cuore · Selex ES A Finmeccanica Company · IAEMO

PULSE

Platform for European Medical Support during major emergencies

WP6 Integration

Deliverable D6.1 – Integration infrastructure description

29/02/2016

Contractual Delivery Date:	Actual Delivery Date:	Deliverable Type*-Security**:
29/02/2016	18/04/2016	R - PU
607799	PULSE	Platform for European Medical Support during major emergencies

*Type: P: Prototype; R: Report; D: Demonstrator; O: Other.

**Security Class: PU: Public; PP: Restricted to other programme participants (including the Commission); RE: Restricted to a group defined by the consortium (including the Commission); CO: Confidential, only for members of the consortium (including the Commission).

Responsible:	Organisation:	Contributing WP:
Francesco Malmignati	SES	WP6

Authors (organisation)
Francesco Malmignati (SES), Massimiliano Taglieri (SES), Karl Chadwick (SKY)

Abstract:
This document aims at describing the technological solutions that have been adopted to develop the PULSE integration and deployment infrastructure. It then gives insights on how all the single modules are connected with each other and provides a description of the different tools' APIs invocations.

Keywords:
Integration, infrastructure, containers, microservices, SOA, service, Docker, deploy

Revisions

Revision	Date	Description	Author (Organisation)
0.1	18/02/16	TOC proposal	Francesco Malmignati (SES)
0.2	29/02/16	Contribution in paragraphs 4.1, 4.2, 4.3, 4.4 and chapter 5	Francesco Malmignati (SES), Massimiliano Taglieri (SES)
0.3	30/02/16	Integration of SKYTEK contribution	Francesco Malmignati (SES), Karl Chadwick (SKY)
0.4	11/03/16	Contribution to paragraph 5.2 and editing of the document	Francesco Malmignati (SES), Massimiliano Taglieri (SES)
0.5	15/03/16	Internal review and final editing of the document	Francesco Malmignati (SES), Massimiliano Taglieri (SES), Paul Kiernan (SKY)
1.0	15/03/16	Final version of the document	
2.0	13/04/16	Addressing EU external reviewer's remarks regarding the integration with external system and standards. Two new sections (4.11 and 4.12) have been added	Francesco Malmignati (SES)
2.1	18/04/16	Internal review and final editing of the document.	Francesco Malmignati (SES) Paul Kiernan (SKY)

Table of contents

List of figures

1 List of acronyms

Acronym
DSVT
DoW
EDXL
ENSIR
GUI
HTTP
IAT
JSON
LT
PCET
MPORG
REST
SA
SARS
SGCT
SIR
SOA
SOAP
TT

2 Executive Summary

As stated in the DoW, one of the objective of WP6 is to *"Design, develop and test the integration infrastructure which will allow to easily integrate the different tools developed in WP3-4".* This document aims at satisfying this requirement and then to describe the technological solutions that have been adopted to develop the aforementioned PULSE integration infrastructure.

The first part of this report is a general re-introduction to the PULSE architecture (already defined and described in WP4) and a detailed description of the tools' integration. This has been accomplished by giving insights on how all the single modules are connected with each other and by providing a description of the different tools' APIs invocations.

The second part of the report consists instead in a more detailed summarization of the technical solutions that have been put in place to efficiently create the deployment infrastructure. After the analysis of four different available integration patterns, the report provides a description of the adopted solution (specifically, the Microservices approach) and the relative deployment framework (Docker) that has been used to deploy the platform and to provide a common point of access to possible external WP7 stakeholders.

3 Introduction

3.1 Scope of the Document

This document aims at summarizing the technological solutions that have been adopted for the definition and development of the PULSE integration infrastructure. Moreover, it plans to outline the methodology used for integration of all the heterogeneous tools' functionalities by describing the data structures and the APIs provided by the different PULSE tools.

3.2 Structure of the Document

This document is structured into the following sections.

- High level description of the PULSE architecture;
- Description of the different logical integrations occurring between the PULSE tools;
- Description of the PULSE communication layer;
- Description of the PULSE Integration infrastructure;
- Description of the deployment infrastructure.

3.3 Relations with other Deliverables

The work presented in this deliverable is related to all the WP4 deliverables containing the reports on the PULSE tools development:

- D4.1 – Decision support validation tool [2]
- D4.2 – IAT Tool [3]
- D4.3 – Logistic tool [4]
- D4.4 – Surge Capacity Generation Tool [5]
- D4.5 – Training tools [6]
- D4.6 – Post Crisis Evaluation Tool [7]
- D4.7 – Event evolution model for biological events.[8]

4 Architecture

4.1 High level description (SES)

D4.1 [2] already provides an high level description of the PULSE platform. We decided to propose it again in order to facilitate the reading of the present deliverable.

The architecture of the PULSE platform is composed of several software modules distributed on a service-based architecture.

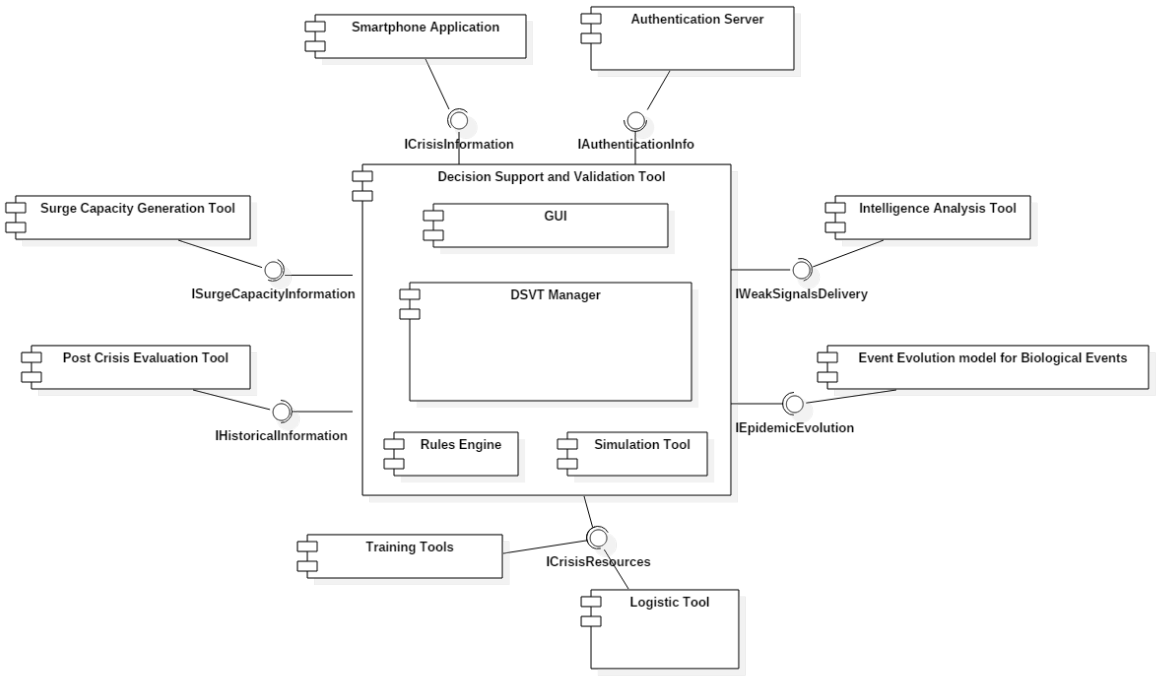

Figure 1: PULSE Architecture – Component Diagram

As shown in Figure 1 the core of the architecture is represented by the *Decision Support and Validation tool* (DSVT) that acts as the front-end interface as well as the communication backbone of the platform. All the other PULSE tools can (1) exploit the interfaces provided by the DSVT or (2) provide functionalities to the DSVT itself.

The PULSE platform is specifically composed of the following tools:

- **Decision Support and Validation Tool (DSVT)**: it is front-end interface as well as the communication backbone of the platform..
- **Intelligence Analysis Tool (IAT)**: it focuses on weak signal detection to alert decision makers to the occurrence of an unusual biological event.
- **Logistic Tool (LT)**: it is used to assess the required stockpiles of any necessary equipment, medications and vaccinations..
- **Surge Capacity Generation Tool (SCGT)**: it provides support for the creation of surge capacity in the event of a major health crisis.
- **Training Tools (TT)**: these tools include a MPORG training platform for personnel involved in crisis management and a training learning management

system (LMS) tailored for the emergency and health services.

- **Post Crisis Evaluation Tool (PCET)**: this tool is in charge of storing and classifying all the resources (including geospatial and time references), events and decisions that have been taken during the crisis. It allows then the creation of an historical crisis report and the definition of lessons learnt.
- **Event evolution model for Biological Events (ENSIR)**: this tool is the implementation of a mathematical model of epidemics evolution.
- **Smartphone application (SA)**: the Android application can be used to access the PULSE platform.

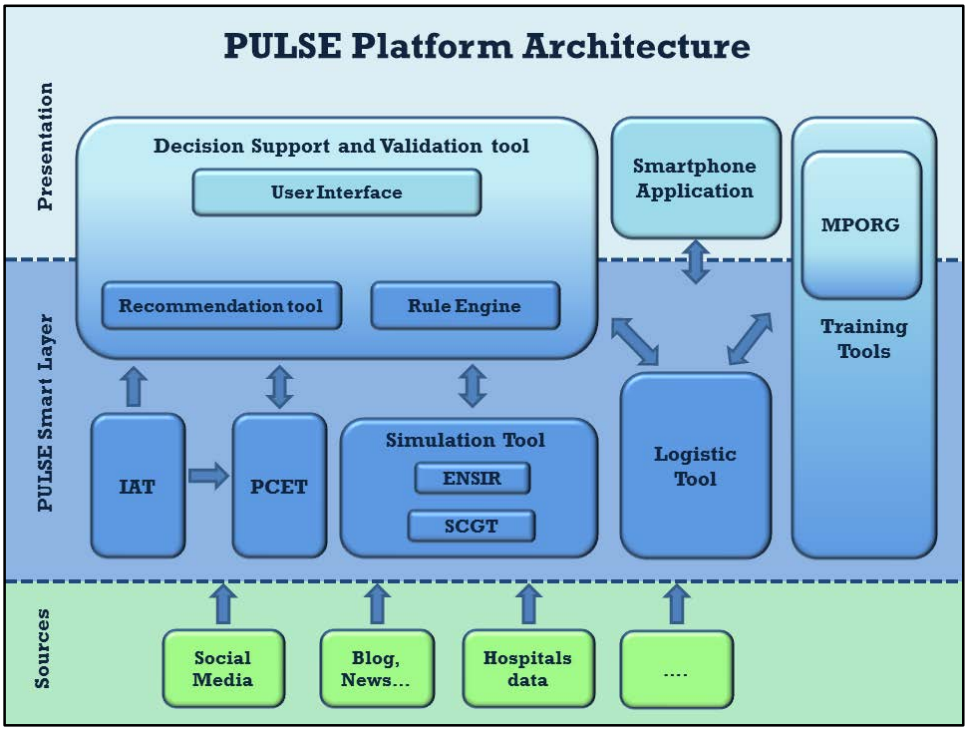

Figure 2: High level architecture

Figure 2 shows the architecture from a multi-layers perspective. The tools have been grouped in three different layers:

- The **Presentation Layer**: composed of the *User Interface module* (part of the *DSVT*), the *Smartphone Application* and the *MPORG*'s GUI, represents the graphical user interface of the PULSE platform. It gives the opportunity, to the different consumers to exploit the features provided by the platform.
- The **PULSE Smart Layer**: it is the core of the PULSE platform. It is composed of all those tools that are able to analyse, store and elaborate the pieces of information coming from the Sources Layer and to provide enriched crisis management functionalities to the upper Presentation level.
- The **Sources Layer**: it is the bottom layer of the platform. It includes all the external services, data sources providing the medical and environmental information.

4.2 DSVT – Logistic tool

The main objective of the PULSE platform is to develop an operational framework that allows the platform's stakeholders to have access to timely key data, planning and decisions that efficiently help them to manage a major healthcare crisis. To achieve this the DSVT (more info in D4.1 [2]), which provides the front-end interface of the platform, invokes the Logistic tool (more info in D4.3 [4]), which is in charge of managing all the data regarding events, such as the Ambulances, First responders, Wounded, Hospitals in an incident-like scenario and the probable and confirmed cases and weak signals in a SARS-like scenario.

The Logistic tool's functionalities are made available through a standard RESTful interface so the messages exchanged are HTTP-based requests and responses. The messages' body is instead represented in JavaScript Object Notation (JSON).

The Logistic tool handles a diverse number of resources such as:

- Ambulance, AmbulanceCategory, Rescuer, Hospital, Person, Status, Resource, ResourceCategory, Triage, TriageCategory, Task, TaskCategory, Contact, ClinicalRecord, Symptom, SymptomCategory, Record, Location, Multimedia.

The Logistic tool provides a set of methods that allow to create, read, delete and update each one of above-mentioned resources and that are directly invoked by the DSVT to obtain or update the current status of the crisis.

In the following paragraph, for the sake of brevity, we describe only the possible interactions for the management of the Ambulance resource by the DSVT but please consider that the same interactions are available also for the other resources.

Five different queries are provided by the Logistic tool in order to manage the Ambulance resource:

- **Get all** Ambulances
- **Create** Ambulance
- **Get** Ambulance details
- **Update** Ambulance
- **Delete** Ambulance

The first query *"Get all Ambulances"* is invoked by the DSVT whenever an update on the status of all the ambulances is requested. The invocation can be performed by calling a HTTP request method GET at the following URL:

- http://hostname:8080/logistic/ambulances

Figure 3 shows an example of possible message body for the result of this invocation.

```
[
  {
    "id": 1,
    "name": "Ambulance - Gemelli Hospital",
    "currentLocation": {
      "latitude": 45.433,
      "longitude": 12.45
    },
    "resources": [
      {
        "id": 11,
        "resourceCategory": {
          "id": 1,
          "name": "defribillator"
        },
        "resourceCategoryId": 1
      }
    ],
    "ambulanceCategory": {
      "id": 1,
      "name": "A"
    },
    "ambulanceCategoryId": 1
  },
{
    "id": 2,
    "name": "Ambulance – San Pietro Hospital",
    "currentLocation": {
      "latitude": 44.32,
      "longitude": 11.47
    },
    "resources": [
      {
        "id": 11,
        "resourceCategory": {
          "id": 1,
          "name": "defribillator"
        },
        "resourceCategoryId": 1
      }
    ],
    "ambulanceCategory": {
      "id": 1,
      "name": "A"
    },
    "ambulanceCategoryId": 1
  }

]
```


Figure 3: Get all Ambulances Response

The second query *"Create Ambulance"* is invoked by the DSVT whenever a new Ambulance has to be added to the list of the available Ambulances. The invocation can be performed by calling a HTTP request method POST at the following URL:

- http://hostname:8080/logistic/ambulances

This request must contain a body describing the Ambulance information. Figure 4 shows a possible body message.

```
{
  "ambulanceCategoryId": 1,
  "currentLocation": {
    "latitude": 45.433,
    "longitude": 12.45
  },
  "name": "Ambulance - Gemelli Hospital",
  "resources": [
    {
      "resourceCategoryId": 1
    }
  ]
}
```

Figure 4: Create Ambulance Input

The third query *"Get Ambulance details"* is invoked by the DSVT whenever the details of a single Ambulance are requested. The invocation can be performed by calling a HTTP request method GET at the following URL:

- http://hostname:8080/logistic/ambulances/{ambulanceId}

where ambulanceId is the id of the ambulance of interest. The id can be obtained by invoking the *"Get all ambulances"* query.

Figure 5 shows an example of possible message body for the result of this invocation.

```
{
  "id": 1,
  "name": "Ambulance - Gemelli Hospital",
  "currentLocation": {
    "latitude": 45.433,
    "longitude": 12.45
  },
  "resources": [
    {
      "id": 11,
      "resourceCategory": {
        "id": 1,
        "name": "base"
      },
      "resourceCategoryId": 1
    }
  ],
  "ambulanceCategory": {
    "id": 1,
    "name": "A"
  },
  "ambulanceCategoryId": 1
}
```

Figure 5: Get Ambulance details Response

The fourth query *"Update Ambulance"* is invoked by the DSVT whenever a specific Ambulance has to be updated. The invocation can be performed by calling a HTTP request method PUT at the following URL:

- http://hostname:8080/logistic/ambulances/{ambulanceId}

where ambulanceId is the id of the ambulance of interest. As said above, the id can be obtained by invoking the *"Get all ambulances"* query.

This request must contain a body describing the new updated Ambulance information. Figure 6 shows a possible body message.

```
{
  "currentLocation": {
    "latitude": 45.233,
    "longitude": 12.65
  }
}
```

Figure 6: Update Ambulance Input

The last fifth query *"Delete Ambulance"* is invoked by the DSVT whenever an Ambulance must be removed from the list of the available Ambulances. The invocation can be performed by calling a HTTP request method DELETE at the following URL:

- http://hostname:8080/logistic/ambulances/{ambulanceId}

where ambulanceId is the id of the ambulance of interest. In this case neither an input nor a response message is necessary.

4.3 DSVT – IAT

The *Intelligent Analysis Tool* (IAT) is an architectural component that is able to systematically gathers and analyses incoming disease-related data and, according to them, it notifies the presence of possible epidemic weak signals.

The IAT interfaces with the DSVT for different reasons:

1. it receives from the DSVT electronic clinical records related to symptoms caused by SARS infection;
2. DSVT is in charge to set up the keywords needed to configure the IAT for filtering incoming tweets by selecting only those related to SARS symptoms;
3. DSVT is also in charge to set up the rules that specify how the IAT internal *CEP Engine* has to derive weak signals from the input data;
4. IAT is in charge to send a week signal to the DSVT whenever it is generated; at this point, the DSVT can show on the Graphical User Interface (GUI) the message related to that week signal.

The first three functionalities are made available by the IAT through a Web Service RESTful interface. The forth one is instead offered by the DSVT again through a Web Service Restful interface. The messages exchanged through these interfaces are HTTP-based requests and responses. They include a message body whose information content is represented in JavaScript Object Notation (JSON). Additional details on each operation are reported below.

An electronic clinical record can be sent to the IAT by calling the HTTP request method POST at the following URL:

- http://hostname:8082/crt.

Figure 7 shows an example of possible message body for this type of operation.


```
{
   "id": "12345",
   "text": "Dr. Nutritious Medical Nutrition Therapy for Hyperlipidemia Referral from: Julie Tester, RD, LD, CNSD Phone
            contact: (555) 555-1212 Height: 144 cm   Current Weight: 45 kg   Date of current weight: 02-29-2001   Admit
            Weight:  53 kg   BMI: 18 kg/m2 Diet: General Daily Calorie needs (kcals): 1500 calories, assessed as HB + 20%
            for activity. Daily Protein needs: 40 grams,  assessed as 1.0 g/kg. Pt has been on a 3-day calorie count and has
            had an average intake of 1100 calories.  She was instructed to drink 2-3 cans of liquid supplement to help
            promote weight gain.  She agrees with the plan and has my number for further assessment. May want a Resting
            Metabolic Rate as well. She takes an aspirin a day for knee pain.",
   "latitude": 42.0,
   "longitude": 12.15
}
```


Figure 7: 'Store Clinical Record' request – Example of message body

- *id* that represents the identifier of the electronic clinical record;
- *text* that includes all the medical information about the patient which the electronic clinical record is referred to;
- *latitude* which represents the latitude of the hospital where the clinical record was compiled;
- *longitude* which represents the longitude of the hospital where the clinical record was compiled.

When the DSVT needs to submit a clinical records to the IAT, it triggers this HTTP POST request with an appopriate message body by using an internally integrated RESTful client.

A keyword for configuring the selection of incoming tweets can be communicated to the IAT by calling a HTTP request method POST at the following URL:

- http://hostname:8082/cepcfg/keyword.


```
{
   "keyword":"fever"
}
```


Figure 8: 'Store Keyword' request – Example of message body

Figure 8 shows an example of possible message body for this type of operation. In this case the message body includes an unique JSON member named *keyword* that represents the keyword to be communicated to the IAT for configuration purposes. Similarly to the previous operation, the DSVT can uses its internal RESTful client to send this HTTP request to the IAT.

A rule for specifying the criteria to generate week signals can be set up in the IAT by calling a HTTP request method POST at the following URL:

- http://hostname:8082/cepcfg/rules.

Figure 9 shows an example of possible message body for this type of operation.

```
{
   "radius":100,
   "latitudine":41.8,
   "longitudine":12.48,
   "desc":"rule_1",
   "params": [
      {
         "nEvents":1,
         "timeInterval":300,
         "source":"CLINICAL_RECORD"
      },
      {
         "nEvents":1,
         "timeInterval":300,
         "source":"TWITTER"
      }
   ]
}
```

Figure 9: 'Store rule' request – Example of message body

- *radius* that represents the radius of the circular area which the rule applies to;
- *latitude* that contains the latitude of the circular area's centre which the rule applies to;
- *longitude* that contains the longitude of the circular area's centre which the rule applies to;
- *desc* which includes a textual description of the rule;
- *params* which specifies an array composed of elements. Each element consists of a set of parameters that, if considered as a whole, determine the criteria to be fulfilled to generate a week signal. These parameters are:
    - *nEvents* which specifies the number of relevant events that should occur in the circular area for generating a week signal;
    - *timeInterval* that defines the duration, in terms of seconds, of the timeslot in which *nEvents* should occur to generate a week signal;
    - *source* that specifies the typology of the source from which the aforementioned events originate (e.g., clinical record, Twitter).

Also in this case, the DSVT can uses its internal RESTful client to send this HTTP request to the IAT for configuring a rule.

A week signal, once generated, can be sent to the DSVT by calling a HTTP request method POST at the following URL:

- http://hostname:8082/week_signals

Figure 10 shows an example of possible message body for this type of operation.


```
{
	"id": 1,
	"description": "description",
	"crawler": [
		{
			"URL": "http://hostname_1",
			"text": "text"
		},
		{
			"URL": "http://hostname_2",
			"text": "text"
		}
	],
	"clinicalRecords": [
		{
			"text": "Dr. Nutritious Medical Nutrition Therapy for Hyperlipidemia Referral from: Julie Tester, RD, LD,
				CNSD Phone contact: (555) 555-1212 Height: 144 cm  Current Weight: 45 kg  Date of current
				weight: 02-29-2001  Admit Weight:  53 kg  BMI: 18 kg/m2 Diet: General Daily Calorie needs
				(kcals): 1500 calories, assessed as HB + 20% for activity. Daily Protein needs: 40
				 grams,  assessed as 1.0 g/kg. Pt has been on a 3-day calorie count and has had an average
				 intake of 1100 calories.  She was instructed to drink 2-3 cans of liquid supplement to help
				 promote weight gain.  She agrees with the plan and has my number for further assessment. May
				want a Resting Metabolic Rate as well. She takes an aspirin a day for knee pain.",
			"symptoms": [
				{
					"symptom": "fever"
				},
				{
					"symptom": "cough"
				}
			],
			"coordinates": {
				"latitude": 53.222,
				"longitude": 45.332
			}
		}
	],
	"twitter": [
		{
			"text": "I have so much cough!",
			"symptoms": [
				{
					"symptom": "cough"
				}
			],
			"coordinates": {
				"latitude": 45.222,
				"longitude": 54.332
			}
		},
		{
			"text": "Fever!",
			"symptoms": [
				{
					"symptom": "fever"
				}
			],
			"coordinates": {
				"latitude": 44.222,
				"longitude": 53.332
			}
		}
	]
}
```


Figure 10: 'Notify Week Signal' request – Example of message body

More generally, the message body consists of five main JSON members:

- *id* that represents the identifier of the weak signal;
- *description* that includes a textual description of the weak signal;
- *crawler* that specifies an array of data related to web contents which have corroborated the hypothesis of a possible SARS-like epidemic outbreak. Each element of this array consists of the following information:
    - *URL* which is the URL of a website where an information attributed to SARS-like outbreaks has been found;
    - *text* that reports the article or, in any case, a textual content related to the SARS-like event which is supposed to be relevant;
- *clinicalRecords* that contains the array of those clinical records which sustain the hypothesis of a possible SARS-like outbreak. Each element of this array includes the following information related to a single clinical record:
    - *text* that includes all the medical information about the patient which the electronic clinical record is referred to;
    - *symptoms* which is an array containing the symptoms experienced by that patient;
    - *coordinates* which includes the geospatial coordinates of the hospital where the clinical record originated;
- *twitter* that contains the array of those tweets which, all together, corroborate the hypothesis of a possible SARS-like epidemic outbreak. Each element of the array consists of the following information related to a single tweet:
    - *text* that represents the text reported in the tweet;
    - *symptoms* which is an array containing the symptoms found within the tweet (each symptom is considered as relevant only if it was specified among the aforementioned configuration keywords);
    - *coordinates* which includes the geospatial coordinates of the tweet.

Similarly to other HTTP requests here described, if the IAT needs to invoke this POST method to communicate the generation of a week signal to the DSVT, it can simply use its internal RESTful client.

The activity carried out to integrate DSVT and IAT mainly consisted (i) in adapting the DSVT RESTful client to correctly invoke the IAT operations and, vice versa, (ii) in adapting the IAT RESTful client to invoke the DSVT operation. The DSVT was also treated to correctly interpret the content of each incoming weak signal to properly report its information content to a user-friendly GUI.

4.4 DSVT – PCET

The Post Crisis Evaluation Tool (PCET) is another relevant component that interfaces with the DSVT. From the logical point of view, PCET implements two categories of functionalities: (i) those used to store information into its internal repository and (ii) those which allow to retrieve that information through the elaboration of ad hoc correlations, analytics and statistics. The first category is used by the DSVT to store historical information about an emergency (when it occurs), the second allows the DSVT to recover and analyse that information in order to show graphics, analytics and data that help understanding the evolution of the event for a post crisis evaluation.

All the mentioned functionalities are made available by PCET through a specific RESTful interface. The offered HTTP methods are POST requests that include a

message body whose information content is represented in JSON. On the other hand, the DSVT is equipped with an integrated RESTful client which is used to invoke the desired operations. Additional details on implemented POST requests and structure of possible message bodies are already reported in D4.6 [2].

The activity carried out to integrate DSVT and PCET mainly consisted in adapting the DSVT RESTful client to correctly invoke the PCET operations in order to obtain useful data for the post crisis evaluation. The DSVT was also treated to correctly interpret the content of each response message body, both in case of information storage and in case of query for information retrieval. This content, if present, is returned in JSON format and afterwards it is properly reported to the user with a user-friendly GUI.

4.5 DSVT – ENSIR

As described in D4.7 [8], the objective of the ENSIR Tool is to provide the expected evolution of the spatial distribution of an epidemic, taking into account different factors that depend on the social and logistic characteristics of the interested area. In the SARS-like scenario, a decision maker (that accessed the PULSE platform through the DSVT) could be interested in knowing the possible spread of the disease within the following days/months/years. This information can help him/her to suggest an intervention (e.g., major procurement of resources) in the hospitals located in the zones that according to the simulated scenario will probably suffer from the epidemic evolution.

The ENSIR functionalities are made available through a Web Service interface based on the SOAP protocol. The DSVT integrates then a SOAP client able to specifically invoke the methods provided by the ENSIR tool. Additional details on implemented requests and structure of possible message bodies are already reported in D4.7.

The activity carried out to integrate the DSVT and the ENSIR mainly consisted in adapting the DSVT SOAP client to correctly invoke the ENSIR Web service operations. The DSVT, once obtained the data from the ENSIR, performs an internal evaluation of the responses and directly visualizes on a map the possible spread of the disease, as described in D6.1.

4.6 DSVT – SCGT

As described in D4.4 [5], the objective of SCGT is to predict the possible evolution of some critical medical resources during a major health crisis. This functionality is exploited by the DSVT during the simulation phase that is, as said in D6.1, the description of the possible behaviour of all the actors involved in the scenario (e.g. casualties, first responders, ambulances, hospitals) according to the available information and the integrated evolution models.

The SCGT functionalities are made available through a Web Service interface based on the SOAP protocol. Similarly to the DSVT – ENSIR integration described in 4.5, the DSVT integrates a SOAP client able to specifically invoke the methods provided by the SCGT tool. Additional details on implemented requests and structure of possible message bodies are already reported in D4.4.

The activity carried out to integrate the DSVT and the SCGT mainly consisted in adapting the DSVT SOAP client to correctly invoke the SCGT Web service operations. Therefore, the DSVT elaborates the data coming from the SCGT during the simulation to efficiently predict the possible evolution of the emergency situation.

4.7 MPORG – Logistic tool

Similar to the DSVT, the MPORG provides a front end to the other Pulse services, in this case for the purposes of game-based training. After authentication, the MPORG interfaces with the other Pulse components in a read-only manner.

For game setup, the MPORG requests the logistics data with GET requests for Locations, Hospitals and Ambulances, in the same manner as described in 4.2.

To calculate the players score, the MPORG compares each decision made with the optimal decision (the Optimization functionality has been already deeply described in D4.3 [4]) at that time by sending the current hospital & patient state details to the Logistic tool and comparing that result with the state of the game after that decision has been made.

A request for a new optimal decision is sent to the Logistic tool by calling a HTTP request method POST at the following URL:

- http://hostname:8080/optimization

```
{
   "name": "Named Event",
   "patients": [ . current patient list & statuses.. ],
   "hospitals": [ . current hospital list & statuses .. ]
}
```

Figure 11: Optimization functionality invocation

Figure 11 shows an example of possible message body for this type of operation.

On completion of the game, the results are added up and displayed on screen for analysis.

```
{
   "name": "Named Event",
   "patients": [ . optimal patient list & statuses.. ],
   "hospitals": [ .. optimal hospital list & statues. ]
}
```

Figure 12: Response from optimization functionality

Figure 12 shows an example of possible message body for the result of the optimization request to the Logistic tool.

Once the game session is closed, all data retrieved from the services is discarded, and no other requests are made.

4.8 MPORG – SCGT

For generation of casualty details, the SCGT service is used using the jar library as defined in D3.1 [9] and integration as described in section 4.6 here.

During the game, as the player applies treatment to a patient subsequent requests are made to the SCGT service to update the status of the patient. An example is

described in the following java code:

```
PatientStatus  updatedStatus = BIO_SERVICE.patEvlove(patientEvolveParams);
geoPatient.setHealthStatus(updatedSatus);
```

Figure 13: SCGT Invocation

Where BIO_SERVICE is the SCGT tool that takes the current status of a patient, the time and the applied therapy as parameters and generates an appropriate new status, which is then applied to the geo-located patient in the game.

4.9 Smartphone app – Logistic tool

The Smartphone App is another front-end to the system, in a mobile/tablet form factor for the specific task of allowing responders in the field to read tasks & respond by submitting data records.

The App authenticates the users as normal, then requests the latest list of tasks from the Logistic tool with a GET request to the following URL:

- http://hostname:8080/logistic/tasks

```
"tasks": [
        {
            id: 1,
            type: "triage",
            title: "Perform Triage at site X",
            description: "<p>This is an example triage task, with a longer html description.</p>",
            timestamp: 1456145557539,
            status: "open"
        }
```

Figure 14: Retrieve Tasks list

Figure 14 shows an example of possible message body for the result of this invocation.

Similarly, to obtain the list of messages a GET message is sent to the following URL

- http://hostname:8080/logistic/messages

```
"messages": [
   {
      id: 1,
      title: "Stadium crush in Dublin at a concert in Aviva
      Stadium, multiple causalities.",
      description: "Lorem ipsum dolor sit amet, consectetur
      adipiscing elit, sed do eiusmod ... ",
      timestamp: 1456145557539
   },
   {
      id: 2,
      ...
```

Figure 15: Retrieve Messages List

Figure 15 shows instead an example of possible message body containing the list of the messages to be displayed on the Smartphone app.

An important part of the Smartphone app is to allow a user to accept a task. The app is able to do so by sending a PUT request to the following URL:

- PUT http://hostname:8080/logistic/task/{taskId}

```
{
   "status": "in_progress",
    timestamp: 1456145557539
}
```

Figure 16: A user accept a task

Similarly, when a task is complete a user can send an update that specific task specifying for example, a new task status and the actual user coordinates.

- PUT http://hostname:8080/logistic/task/{taskId}

```
{
   "status": "busy ",
   "coordinates": {
                    "latitude": 44.222,
                    "longitude": 53.332
    },
    timestamp: 1456145557539
}
```

Figure 17: Task update

For recording data, the app can send a new triage data record to the Logistic tool by sending a POST message to the following URL:

- http://hostname:8080/logistic/triage/

Figure 18 shows an example of message body containing the actual triage information of a casualty. The message can contain the barcode id taken directly from the bracelet put on the causality wrist, a photo of the injured person, a text and an audio note and the actual location of the person.

```
{
   "task_id": 1,
   "type": "triage",
   "barcode": "SOME_CODE_AS_UID_STRING",
   "risk": "high",
   "textNote": "This is some text manually typed into a  textarea.",
   "audioNote": "data:audio/mp3;base64,T2dnUwACAAA ... ",
   "photo": " ... ",
   "location": "53.338265899999996,-6.2505142",
   "timestamp": 1456145557539
}
```

Figure 18: Triage message body

4.10 Communication layer

The previous paragraphs summarize all the logical connections occurring between the PULSE tools. But from an operational point of view, all the connections are actually handled by the DSVT that acts as the backbone communication layer and can be considered the “Service Bus” used by the PULSE tools to communicate.

In the following paragraph we propose again the internal structure of the DSVT (that has been already described in D4.1) in order give a complete overview on the integration structures that have been built to develop the PULSE platform.

The DSVT Manager is able to:

- Group the tool functionality under a *single software interface*
- Group several and heterogeneous data sources and provide such sources as a single repository
- Hide the internal modularity of the platform
- Make transparent to the user the internal mapping of the tool

Because of this, the DSVT Manager provides an API that makes available the internal functionalities of the tool.

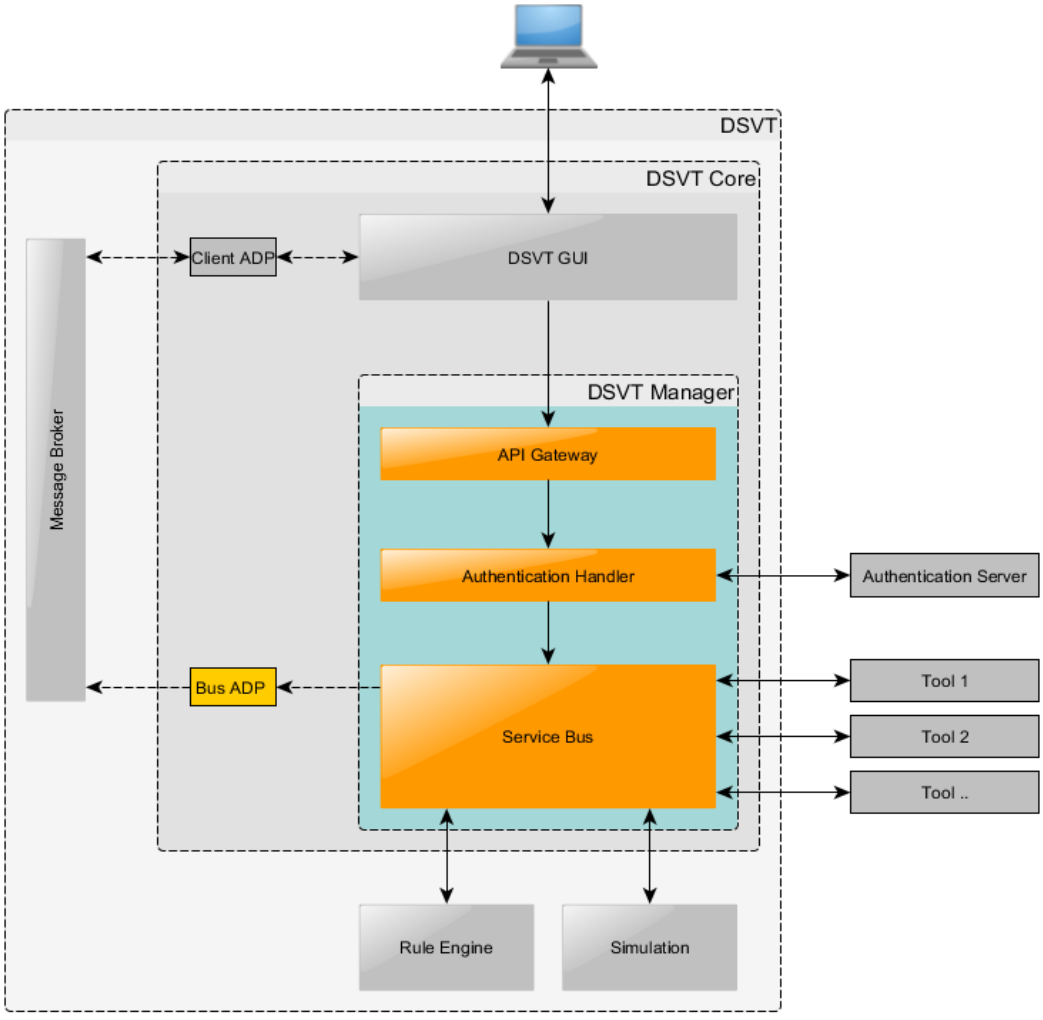

Figure 19: DSVT Manager

The block diagram represented in Figure 19 shows the functional elements, which constitute the DSVT Manager:

- **API Gateway:** provides an HTTP interface to Clients. Each request is processed here and is routed the *Authentication Handler*

- **Authentication Handler:**
  - Act as a proxy gateway handling the incoming requests;
  - Manage the User authentication against the Authentication server;
  - Manage sessions after user authentication;
  - Manages the request of access token from the Authentication Server and return to the user an access token
  - Validates the access token against the authentication server and routes the request to *Service Bus* or, if necessary, rejects the request

- **Service Bus:** it is the aggregation layer; it provides a unique software interface and manages the aggregation of the platform tools.

4.11 Integration with external systems

Considering the necessity to offer always updated information and to provide a complete and efficient support, the PULSE platform has been integrated with already existing services that are currently used by actors involved in emergency coordination activities.

Three different systems have been selected for this purpose:

- **Open Data for real-time access in the emergency department of Lazio**:
  - This dataset allows to acquire the status of the emergency department of all the hospitals located in the Lazio Region (in Italy).
    For each hospital it is possible to see the number of patients under observation, under treatment or that are waiting to be examined by a doctor. The number of patients is in turn divided among four different categories (red, yellow, green, white) depending on the priority code assigned to each individual patient.
  - The system is currently used by the Emergency coordinator operating in the Lazio Region and it is publicly accessible through the Lazio Region web site: http://www.regione.lazio.it/accessiprontosoccorso/
  - The PULSE platform integrates the functionalities provided by this dataset by invoking the API accessible at this link: http://dati.lazio.it/catalog/en/dataset/pronto-soccorso-accessi-in-tempo-reale
  - The functionality has been integrated into the DSVT GUI and allows any emergency coordinator using the PULSE platform to have a real-time access to the Lazio Hospital emergency department information. Figure 20 shows the current available information of Hospital Gemelli's emergency department.

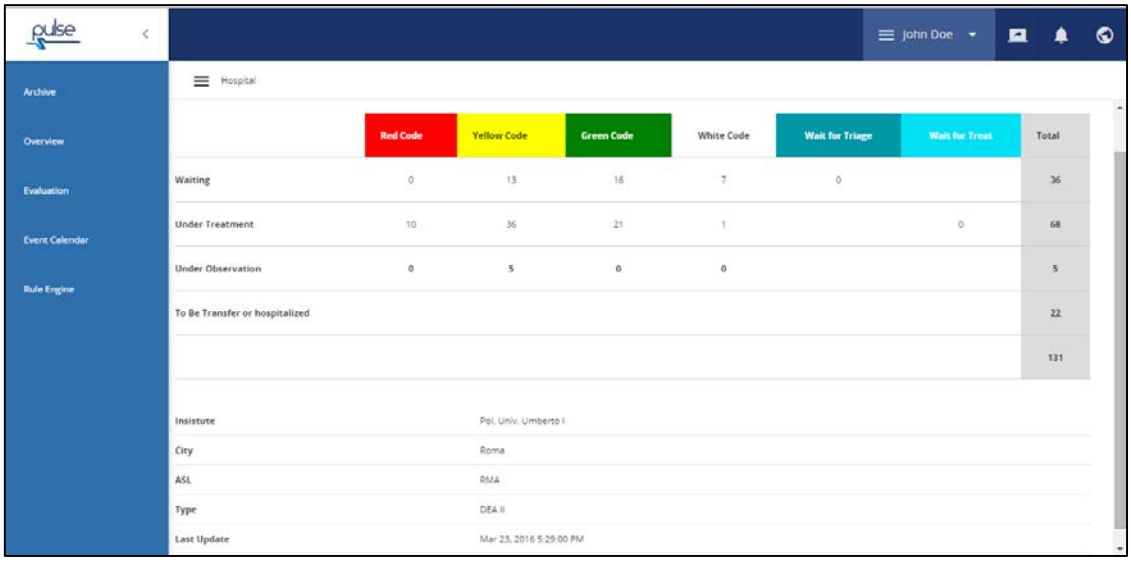

Figure 20: DSVT - Emergency Department integration

- **ProMED - the Program for Monitoring Emerging Diseases:**
  - ProMED is dedicated to the rapid dissemination of information on outbreaks of infectious diseases and acute exposures to toxins that affect human health. It is also able to provide up-to-date and reliable news about threats to human, animal, and food plant health around the world.
  - The system is accessible at the website http://promedmail.org/
  - The PULSE platform allows to access the information handled by the ProMED website directly from the DSVT GUI. As shown in Figure 21, a user can select a specific event, can see the communications related to that selected event and can visualize it on a map.

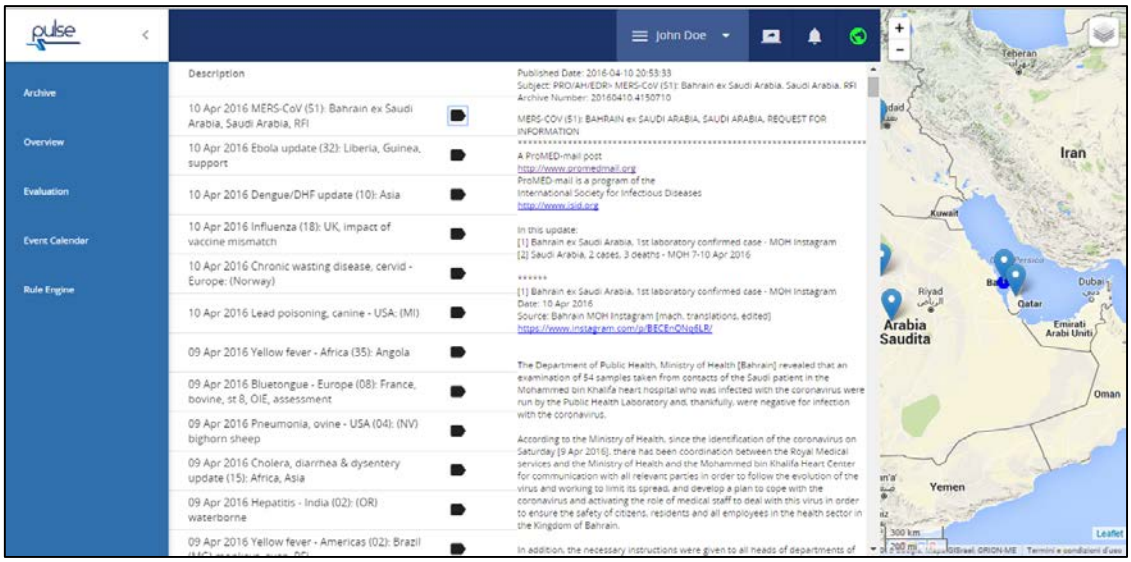

Figure 21: DSVT - ProMED Integration

- The PULSE platform provides also an alerting system that periodically checks the ProMED website and sends an alert whenever a new

ProMED communication has been added to the system. This allows the decision makers to have an always updated status of epidemics all around the world.

- **HealthMap:**
  - As stated in [14], HealthMap brings together disparate data sources, including online news aggregators, eyewitness reports, expert-curated discussions and validated official reports, to achieve a unified and comprehensive view of the current global state of infectious diseases and their effect on human and animal health. The system monitors, organizes, integrates, filters, visualizes and disseminates online information about emerging diseases.
  - The system is accessible at the website http://www.healthmap.org/en/
  - The PULSE platform integrates the search widget and the results generated by HealthMap directly into the DSVT general overview map. In this way, a decision maker can easily access to the HealthMap-related alerts by simply selecting the HealthMap layer on the DSVT GUI.

, As well as integrating data from external emergency response system, PULSE also provides an external interface to facility the integration of PULSE generated information into third party applications. Third party applications can easily integrate with the PULSE platform by simply invoking the RESTful API provided by the PULSE tools. For example, as described in 4.2, 4.7 and 4.8, the Logistic tool exposes a standard RESTful interface and provides set of methods that allow to create, read, delete and update the crisis resources. These methods, as seen above, are actually invoked by other PULSE tools (e.g., DSVT, Smartphone app and MPORG), but the same integration can be performed by other authorized external systems that can exploit the Logistic tool functionalities and then retrieve and possibly update the crisis resources handled by the PULSE platform.

4.12 Integration with existing standards

As described in WP4, all the PULSE tools are based on well-known and up-to-dated standards for the web communication such as:

- **HTTP** with the *RESTful* approach used by most of the PULSE tools
- **SOAP** used by 2 tools (SCGT and ENSIR).

As a result of this and considering the increasing acceptance of these two standards as the standards de-facto for the systems communication, the PULSE platform can be easily adaptable and integrated with a plethora of already existing systems.

This approach can be considered valid for most of the domains but in the case specific of the Health domain, some effort has been spent towards the integration with already existing standards that can facilitate the communication with a usually strict and pretty closed environment. In this specific case, the **EDXL-HAVE** standard [15], which specifies through an XML document format, the communication of the status of a hospital, its services, and its resources, has been considered for the integration within the PULSE platform.

Specifically, 3 different steps have been performed:

- The **EDXL-HAVE** XML schema (and its dependencies) [16] has been collected and analysed.
- The XML Schema has been used to generate an adapter for the marshalling (and un-marshalling) of **EDXL-HAVE** messages to Java Objects (and vice versa).
- The adapter has been integrated into the Logistic tool, which is the PULSE tool in charge of storing and managing the hospitals resources.

The EDXL-HAVE is just an example of possible standards' integration and it is plausible that more standards will be integrated during a possible future exploitation of the PULSE platform.

5 Integration infrastructure

5.1 Possible implementation strategies

In the last 20 years several integration patterns have been defined for the development and deployment of software applications.

These patterns can be summarized in 4 main approaches:

- Monolithic approach;
- Multi-tier approach;
- Service Oriented architecture;
- Microservices approach.

5.1.1 Monolithic approach

It is basically the first approach where all the application were developed and deployed as a single entity. In such approach all the functionalities provided by the software applications are piled into a single monolithic application that aggregates everything (e.g. authorization, business, data storage functionalities). This kind of approach is suitable for proof of concepts and prototypes but in production environments the deployment, scaling and upgrading of such monolithic application can become really difficult. Moreover, as a single entity, a monolith can only scale by replicating the entire application. This could become too much costly and a waste of resources as traffic and load grows.

5.1.2 Multi-tier approach

The limits of the monolithic approach have been partially addressed with the definition of the Multi-tier architecture approach. In this approach an application is logically distributed among different layers that generally consist of a data layer, a business logic layer, and a presentation layer (and for this reason it is usually called 3-tier application).

In this case, the three tiers can be upgraded or replaced independently in response to changes in requirements and, in case of an increasing load, we would be able to just scale the business logic layer. However, the same drawbacks of the monolith approach can appear and be related to just the business logic tier in case it would become too much big and then hard to scale.

5.1.3 Service oriented architecture

The Service Oriented Architecture (SOA) was designed to overcome some of the aforementioned limitations by introducing the concept of *service* which is an aggregation and grouping of similar functionalities offered by an application.

For example, developers would create a user service that handles authentication, an order service that handles billing or a notification service that handles sending emails. This approach can effectively improve the scalability of the system as each service would be smaller and so more easy to scale.

This approach brings a great improvement regarding the integration capabilities of a

system, but can lead to complex services that would start growing over time by accumulating several dozens of functionalities. For this reason SOA applications could turn as well into a mixture of a several monolithic service instances bringing along all the aforementioned drawbacks.

5.1.4 Microservices approach

The idea behind the microservices architecture (MSA) is about developing a single application as a suite of small and independent services that are running in their own process, developed and deployed independently. Instead of connecting various applications together, the microservices pattern aims to create a single, cohesive application comprised of independently developed and deployed services that each follow the single responsibility principle. By limiting the scope of what a service can do, developers can ensure they do not unintentionally end up with a large number of monoliths. It differs from the service-oriented architecture (SOA) approach since the latter aims at integrating various (business) applications whereas several microservices belong to one application only. Microservices architectures are substantially less cumbersome than traditional SOA and don't require the same level of governance and canonical data modelling to define the interface between services.

5.2 Our solution – Docker

As described in 5.1, different approaches can be followed for the implementation of a software platform. In our case, according to the PULSE platform nature and the provided requirements, the most suitable one is the Microservices approach. The PULSE platform, as described in 4.1, is composed of a constellation of tools interconnected each other. If we consider each tool as a "microservice", the similarity between the Microservices architecture approach and the PULSE platform is pretty straightforward.

In this context a new rising technology that comes in our support is Docker [10]. Docker is an open-source project that automates the deployment of applications inside software containers. A Docker software container can be seen as bundle that wraps up an application and all its dependencies (code, runtime, system tools, system libraries) in a single package, giving the possibility to completely isolate an application.

At the core of the Docker platform is the Docker Engine [11], a runtime tool that builds and runs the Docker software containers. In the PULSE context, we decided to exploit the functionalities offered by the Docker platform and to transform each tool composing the PULSE platform in a Docker container. All these "Docker-ized" tools lie then on top of the Docker Engine that handles the communication and the interconnection between the tools/containers.

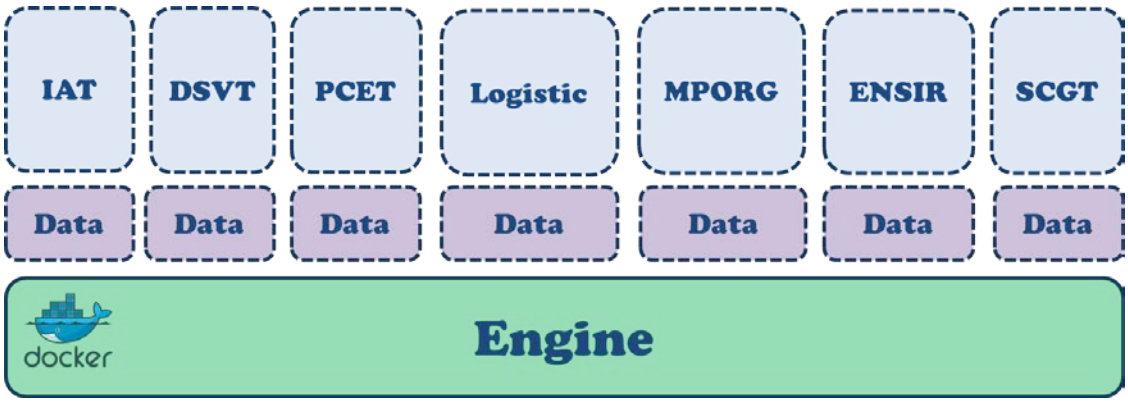

Figure 22: Pulse Docker architecture

Figure 20 shows the Pulse Docker architecture. Each dotted border light-blue component represents a single container that includes one of the PULSE tools. The second layer composed of light-violet dotter components represents instead a set of containers that manage the data of the upper components. The bottom layer is then composed of the Docker Engine that, as said above, manages the interconnections between all the Docker containers.

Docker allows to simply build containers by reading the instructions reported in particular text documents called *Dockerfiles*.

```
# Dockerfile to run the Post Crisis Evaluation Tool

FROM java:8-jre
ADD "PostCrisisEvaluationTool" "/home/Pulse/Software/PostCrisisEvaluationTool"
WORKDIR /home/Pulse/Software/PostCrisisEvaluationTool
EXPOSE 8080
CMD ["java","-jar","PostCrisisEvaluationTool.jar"]
```

Figure 23: Post Crisis Evaluation Tool – Dockerfile to build the container's image

Figure 21 shows an example of dockerfile that can be used to automatically build an image of a container including the minimal software components needed to make running the PCET. Let's go to analyse the content of this dockerfile:

- *FROM* command is used to download and install the Java Runtime Environment where the PCET is executed (the presence of a Java layer is an essential requirement since PCET is, in effect, a Java application);
- *ADD* command is used to copy all the files composing the PCET in the specified working path inside the container (i.e., */home/Pulse/Software/PostCrisisEvaluationTool*);
- *WORKDIR* command is used to promote to working directory that located in the reported path (i.e., */home/Pulse/Software/PostCrisisEvaluationTool*); this directory is then considered for the execution of the subsequent commands;
- *EXPOSE* command informs the Docker Engine that the container listens on the specified network port at runtime (i.e., 8080); this allows external PCET clients (i) to send HTTP requests to the PCET running inside the container and (ii) to get back the related HTTP responses;
- *CMD* command is used to run the PCET's executable Java file named *PostCrisisEvaluationTool.jar*.

Once the container has been built, the ulterior step is to trigger the process related to the container in background.

The dockerfile in Figure 21 per se allows to build an image used to make running the PCET related container. However, if that dockerfile is used alone, the resulting container will also include the data stored in the PCET's *Historical Information Repository*. This means that there is no separation between the application tool and its related data, since both are located into the same container. This construction has a considerable drawback: if the application tool needs to be changed with an updated release, its related data will be lost, since the container needs to be removed and substituted with another container including the newer release.

A solution to avoid this problem is to separate tools and data as shown in Figure 20. As previously stated, this kind of architectural organization is based on two containers for each tool, one for the application tool and one for its data. In such a way, should an application tool be substituted with an updated version, the only container that needs to be changed is that including the application itself, whereas the container including the data does not require any change.

Docker allows to realize such architectural organization exploiting its integrated features. Let's consider again the PCET case to help explain how it can be realised. The dockerfile reported in Figure 21 is still valid to build an image for the container related to the application tool. However a further dockerfile is needed to build an additional image for the container including the data stored in the *Historical Information Repository* (this dockerfile is omitted here as it is similar to the one reported above and it does not add further interesting notions to the discussion).

At this point, the Docker Engine is responsible for the communication between these two containers that is realized through the so called *docker-compose* [12], a tool for defining and running multi-container Docker applications.

```
pcet:
  build: ./PCET
  ports:
   - "8080:8080"
  volumes_from:
   - pcet_data
pcet_data:
  build: ./PCET_data
  volumes:
   - /home/Pulse/Software/PostCrisisEvaluationTool/data
```

Figure 24: Post Crisis Evaluation Tool – Docker-compose

The docker-compose tool analyses the instructions reported in a particular text document in order to attach the application tool container to the related data container. Figure 22 shows the docker-compose document that links the PCET's containers. Let's analyse its content:

- *pcet* refers to instructions used to run the PCET container, in particular:
  - *build* specifies the relative path where is located the folder containing the PCET Java application and the dockerfile to build an image of its container;
  - *ports* defines a mapping between the exposed container port and the corresponding port of the machine hosting the container (i.e., host:container);
  - *volumes_from* mounts the data volumes taken from another container (i.e., that defined in *pcet_data*);

- *pcet_data* refers to instructions used to run the PCET's data container, in particular:
    - *build* specifies the relative path where is located the folder containing the PCET basic data and the dockerfile to build an image of its container;
    - *volumes* specifies the container's data path.

6 Deployment

As described in 5.2, the PULSE platform follows the Microservices architecture paradigm and has been integrated on top of the Docker tools suite. Docker gives the possibility to create and deploy applications faster and easier and to deploy scalable services, securely and reliably, on a wide variety of platforms.

In our case, we decided to deploy our docker-compliant components in a Ubuntu [13] machine with Docker installed. This allows us to rapidly have the PULSE platform up and running in a really short amount of time and to easily replicate the same deployment in another Linux machine for e.g. testing purposes or privacy data issues.

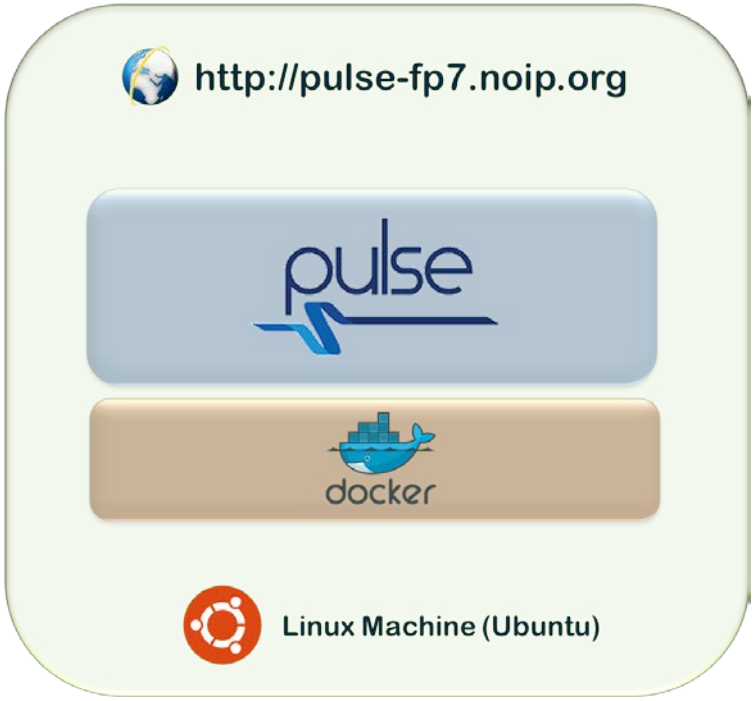

Figure 25: PULSE Platform Deployment

As shown in Figure 23, we have also defined a public address (http://pulse-fp7.noip.org) for the external provision of the PULSE platform. We plan to use the platform hosted at this address during the trials that will be performed in WP7.

References

[1] PULSE Project – Description of Work, version 1.0, October 2013.

[2] PULSE Project Deliverable – D4.1 Decision support validation tool.

[3] PULSE Project Deliverable – D4.2 IAT tool

[4] PULSE Project Deliverable – D4.3 Logistic tool

[5] PULSE Project Deliverable – D4.4 Surge capacity tool

[6] PULSE Project Deliverable – D4.5 Training tools

[7] PULSE Project Deliverable – D4.6 Post crisis evaluation tool

[8] PULSE Project Deliverable – D4.7 Event evaluation for biological event

[9] PULSE Project Deliverable – D3.1 Context models

[10] Docker – Official Web site https://www.docker.com/

[11] Docker Engine – Official Web site https://www.docker.com/products/docker-engine

[12] Docker compose documentation - https://docs.docker.com/compose/

[13] Ubuntu OS - http://www.ubuntu.com/

[14] HealthMap - http://www.healthmap.org/site/about

[15] Emergency Data Exchange Language (EDXL) Hospital Availability Exchange (HAVE) Version 1.0 - http://docs.oasis-open.org/emergency/edxl-have/v1.0/emergency_edxl_have-1.0.html

[16] EDXL-HAVE v1.0 XML Schema - http://docs.oasis-open.org/emergency/edxl-have/edxl-have.xsd