CALL H2020-SC1-FA-DTS-2018-2020
Trusted digital solutions and Cybersecurity in Health and Care
TOPIC DT-TDS-01-2019
Smart and healthy living at home

SMART BEAR

"Smart Big Data Platform to Offer Evidence-based Personalised Support for Healthy and Independent Living at Home"

D6.7 (D73) - Integrated SMAR BEAR platform v0.1

Due date of deliverable: 31-03-2021
Actual submission date: 21-05-2021

Grant agreement number: 857172
Start date of project: 01/09/2019

Lead contractor: CNR
Duration: 60 months

Project funded by the European Commission within the EU Framework Programme for Research and Innovation HORIZON 2020	
Dissemination Level	
PU = Public, fully open, e.g. web	✓
CO = Confidential, restricted under conditions set out in Model Grant Agreement	
CI = Classified, information as referred to in Commission Decision 2001/844/EC.	
Int = Internal Working Document	

D6.7 (D73) - Integrated SMART BEAR platform v0.1

Editors

Alberto Acebes - ATOS
Luis Gato - ATOS
Adrian Arroyo - ATOS
Manuel M Perez Perez - ATOS
Elsa Prieto - ATOS

Contributors

Jonatan Maggesi - UMIL
Stefano Siccardi - UMIL
Othonas Soultatos – STS
Konstantina Koloutsou - STS
Ioannis Basdekis - STS
Ioannis Kouris - ICCS
Pablo Malmierca - ATOS

Reviewers

Stéphane Railhet - SV
Christos Kloukinas - CITY

Executive Summary

This document shows the practical execution of the integration of the SB@Cloud components in order to implement the basic functionalities for the first prototype to be used for the Pilot of Pilots (PoP).

Everything described here has a direct and complementary relationship with the documented content in deliverable D6.8 (*Report on integrated SMART BEAR platform v0.1*), and the union of both provide a complete vision, practical, and demonstrative information about the integration of the components for the prototype. For this reason, and in order not to be repetitive in term of content, frequent cross-references are made, whereas in some case and for clarity shake some tables are repeated. In that case, the reference to D6.8 is provided.

It is emphasised that the scope of this integration varies from the original definition of the process planned for later versions, since at this early stage, the whole infrastructure is still not available and therefore, a simplified integration was accomplished, sufficiently for demonstration and pilot execution.

Due to the above described, this document takes special care to not create confusion between the advanced work for the integration of the final platform (CI/CD, Kubernetes related configuration, extended security aspects) and the work done and temporally alternatives applied in this integration for the PoP.

Contents

Executive Summary 3
List of acronyms 5
List of tables 6
List of figures 7
1. Introduction 8
1.1 Overview 8
1.2 Goals of the deliverable 8
1.3 Pilot of Pilots 8
1.4 Runnable implemented components for SB@Cloud 9
2. Installation and configuration 11
2.1 Containerisation 11
2.2 Git repository (GitLab) 12
2.3 Private Registry (Nexus) 13
2.4 Virtual Machine (VM) 13
2.5 Docker 14
2.6 Docker-compose 15
2.6.1 Component sb-data-repository docker-compose yaml 16
2.6.2 Component Secure_manager_one docker-compose yaml 16
2.6.3 Component SB_dashboard docker-compose yaml 18
2.6.4 Component SB_BDA docker-compose yaml 18
2.6.5 Component DSS (includes non-FHIR-database) docker-compose yaml 19
2.7 Databases 20
2.8 Networking, domains and ports 20
2.8.1 Let's Encrypt Certificates 22
2.8.2 Reverse proxy 24
3. Next steps 25
4. Concluding Remarks 26

List of acronyms

Acronym	Description
ACME	Automatic Certificate Management Environment
API	Application programming interface
BDA	Big Data Analytics
CA	Certificate Authority
DSS	Decision Support System
GDPR	General Data Protection Regulation
PoP	Pilot of Pilots
REST API	(or RESTful API) is API that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services
SB	Smart Bear
SB@App	Smart Bear smartphone app
SB@Cloud	Smart Bear Cloud infrastructure
SB@Dashboard	Smart Bear Dashboard component
SB@HomeHub	Smart Bear Home Hub component
SB@Repository	Smart Bear Repository component
SB@SecurityComponent	Smart Bear Security component
TLS	Transport Layer Security

List of tables

Table 1: List of SB@Cloud components initially proposed containers as they are also described in D6.8........10

Table 2: List of SB@Cloud components as final containers........11

Table 3: Domains and ports accessible from internet........21

List of figures

Figure 1: Scheme of the SB PoP showing the different layers as it is also reported in D6.8............ 9
Figure 2: Containers and exposed ports 11
Figure 3: ATOS research GitLab group for SMART BEAR...12
Figure 4: SSH connexion to the VM provided for the PoP 13
Figure 5: Check docker version installed...14
Figure 6: Check docker-compose version installed 15
Figure 7: Docker-compose up and down commands example...15
Figure 8: WSO2 API Management API 21
Figure 9: WSO2 API Management UI 21
Figure 10: WSO2 Identity Server UI...22
Figure 11: SmartBear Dashboard 22
Figure 12: Reverse proxy to redirect ports and provide TLS (https) 24

1. Introduction

1.1 *Overview*

The SMART BEAR platform is formed of a complex set of components; this is th reason why the issue of integration is an important aspect to consider. Therefore, the platform integration was discussed from the beginning of the project, when it was decided to follow CI/CD methodologies to facilitate this aspect and for the integration to be carried out in the most efficient possible way.

An important issue we have faced is the fact that the demonstration of the first pilot is ahead to the complete installation and configuration of the expected infrastructure (based on a Kubernetes cluster), so for the demonstration of the prototype, it was decided to be performed by means of a simpler environment (based on a Virtual Machine VM).

Although all the objectives of the prototype demonstration are maintained, the VM-based integration does not use part of the integration work advanced with Kubernetes, and it also required specific configuration and alternative tools that were not considered in the original plans.

However, an important part for their integration (the containerisation of the components) has been carried out and also the commissioning of tools and services to allow it. All this work done will be used in the definitive platform and subsequent versions.

1.2 *Goals of the deliverable*

This deliverable focuses on the practical integration of the components in the computational environment provided for the first prototype of the SMART BEAR Platform. They are the components of the SB@Cloud defined in de architecture and developed by the partners.

Moreover, this deliverable meets the goal to clarify, in terms of integration, the current scope and deviation from the original integration methodology objective of the platform, since at this point, part of the tools are not yet implemented but have been prepared for future integrations using Kubernetes

Along with the document, there will be continuous mentions related to the work carried out for the final integration plans, and to explain the alternatives used in this integration that differs from the original ideas, and for the next integration of the platform based on Kubernetes.

1.3 *Pilot of Pilots*

The Pilot of Pilots (PoP) is the initial demonstration of the implementation and integration of the minimum functionalities of the SMART BEAR platform according to the suggestions provided during the first review of the project and will allow to get feedback from all stakeholders involved in the project.

As described in the following figure, the integration made for this demonstration gathers all the main components described in the architecture of the platform, in a way it complies with the GRPD, as well as set up the needed interfaces, setting a milestone for the next phase of improvements and functionalities addition.

In terms of integration, the work performed to establish the bases and fix the definition of the components, allowing further configuration of automations to meet the objective of implementing CI/CD mechanisms.

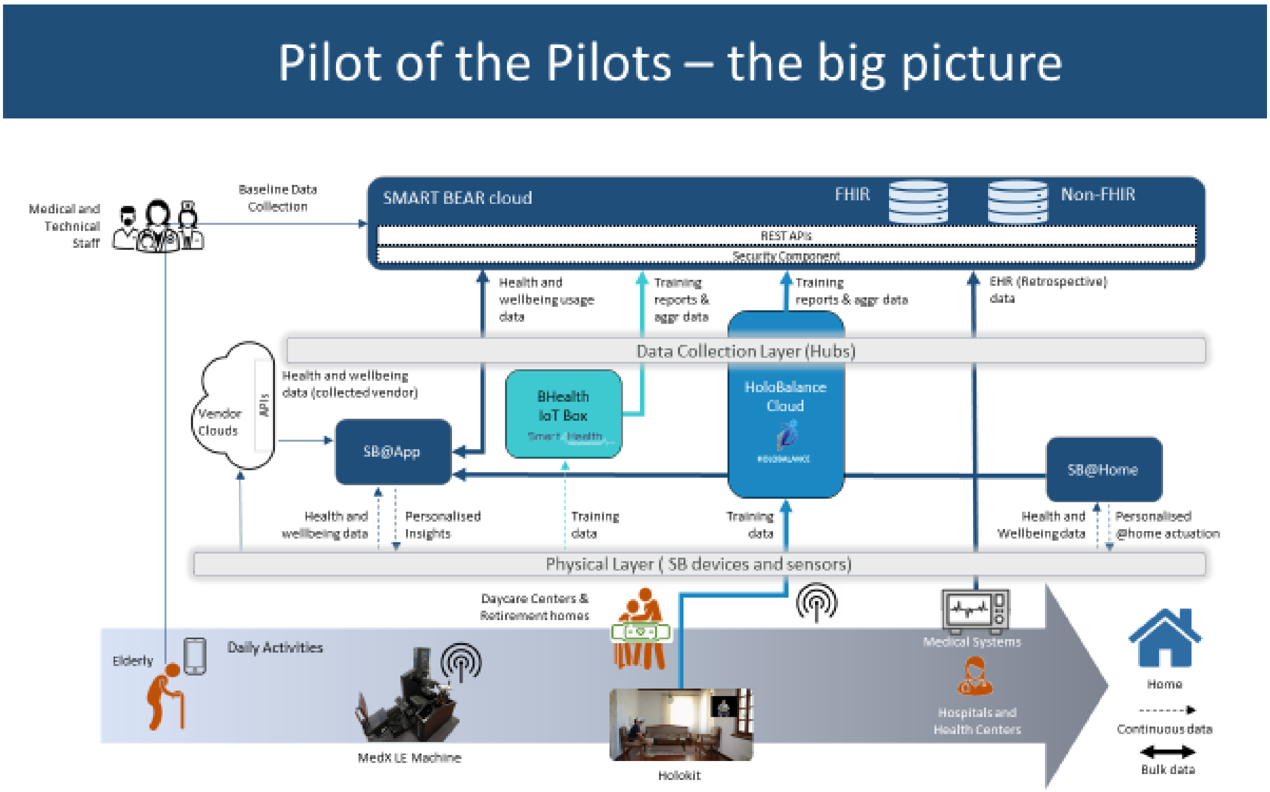

Figure 1: Scheme of the SB PoP showing the different layers as it is also reported in D6.8

1.4 *Runnable implemented components for SB@Cloud*

Extensively described in D6.8 (*Report on integrated SMART BEAR platform v0.1*), after the detail of the high-level set of the platform ecosystem components, we adjusted the functionalities and associated components to the scope of the Pilot of Pilot. Components' owners were asked to package their components as containers, and in different conversations, we defined a naming convention, and specific interfaces between them were agreed for this VM environment, as the exposed ports and service names.

The different components will be run and maintained in the VM by their owners, managed by docker-compose descriptor files.

	Component	Subcomponent	Description
1	Secure_manager_one	usermanager	User manager API under /users
		am	WSO2 API Management UI[1]
		is	WSO2 API Identity Server UI[2]
		postgres	Relational DB supporting Identity server

[1] https://wso2.com/api-manager/

[2] https://wso2.com/identity-and-access-management

2	sb-data-repository	sb-data-repository	HL7 FHIR repository based on HAPI FHIR server to manage the clinical data
		mariadb	Relational DB supporting sb-data-repository
3	SB_dashboard	sb_dashboard	SB Dashboard
4	SB_BDA	Sb_bda	Big Data Analytics
5	Non-FHIR_database	database	DSS auxiliary data base
6	DSS	api_dss	Decision Support System
		dss_core	DSS core services

Table 1: List of SB@Cloud components initially proposed containers as they are also described in D6.8.

2. Installation and configuration

2.1 *Containerisation*

Each component, after their build, tests, and packaging according to the technology used, includes the proper configuration to generate a containerised version of the service and to provide one or several *docker* images alongside a *docker-compose* yaml file as explained later.

Below is the chosen nomenclature for each container generated. Please note that a single functional component can be composed of one or several containers (subcomponents) and that functionalities themselves spread over several containers.

Component	Subcomponent	Description
sb-data-repository	sb-data-repository	HL7 FHIR repository (HAPI FHIR) to manage the clinical data
	mariadb	Relational DB supporting sb-data-repository
Secure_manager_one	usermanager	User manager API under /users
	am	WSO2 API Management UI
	is	WSO2 API Identity Server UI
	postgres	Relational DB supporting Identity server
SB_dashboard	sb_dashboard	SB Dashboard
SB_BDA	Sb_bda	Big Data Analytics
DSS	database	Non-FHIR_database
	api_dss	Decision Support System
	dss_core	Core of DSS

Table 2: List of SB@Cloud components as final containers.

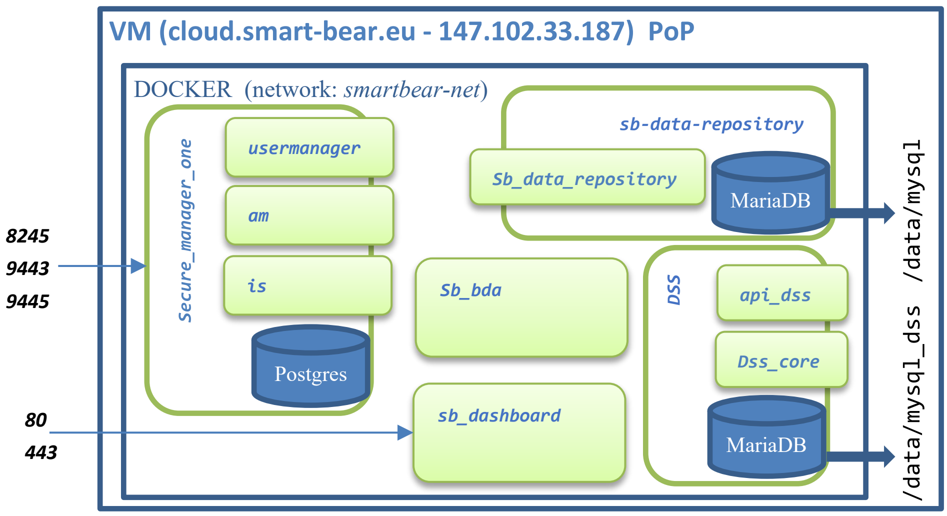

Figure 2: Containers and exposed ports

2.2 *Git repository (GitLab)*

In order to centralise the maintenance of the source code of the different components and to be able to configure the proper CI/CD pipelines, ATOS provided access to its own GitLab[3] DevOps platform. A new private group named “SmartBear”[4] under which the necessary git repositories for each component were created.

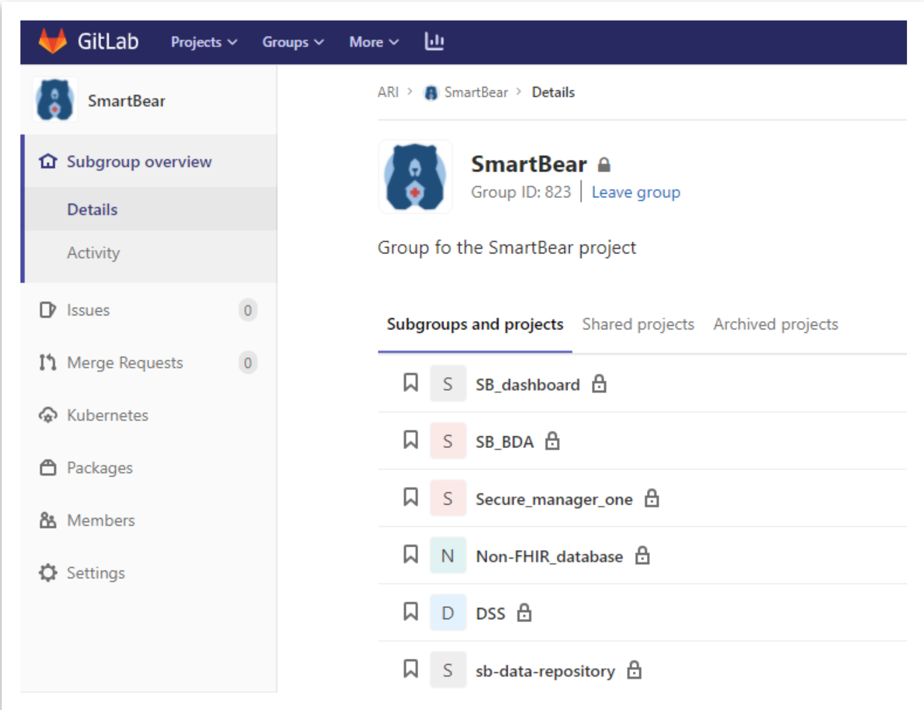

Figure 3: ATOS research GitLab group for SMART BEAR

Every owner was granted the proper permissions to commit and maintain their code.

- SB_dashboard ....Jonatan Maggesi (UMIL)
- SB_BDA..................................Stefano Siccardi (UMIL)
- Secure_manager_one ....Othonas Soultatos (STS)
- Non-FHIR_database ....Ioannis Kouris (ICCS)
- DSS...Ioannis Kouris (ICCS)
- sb-data-repository...................Pablo Malmierca (ATOS) / Alberto Acebes (ATOS)

In each repository, in addition to the component’s own code, configuration files have been required to create the docker images (Dockerfile[5]) and, for future use in Kubernetes, deployment descriptors of the component.

[3] https://about.gitlab.com

[4] https://scm.atosresearch.eu/ari/smartbear

[5] https://docs.docker.com/engine/reference/builder/

The GitLab SmartBear group also has been prepared for direct connection to the future Kubernetes cluster, by installing a runner which monitors the triggering of CI/CD actions from GitLab, and based on deployment descriptor files, that are not detailed in this document as they are not used yet.

2.3 *Private Registry (Nexus)*

In addition to the code repository (GitLab), when working in a containerisation environment a specific docker image repository is suitable to store and make available the images to any remote docker instance. It has been considered that using a public repository is not appropriate in these development phases, so a direct connection between GitLab and Nexus has been set to deposit the images generated, or to push the docker images directly from any local development environment.

In any case, and in these early stages of development, the option to generate the images manually and copy them to the VM for later loading has been the method used for most of the component owners to simplify their work for purposes of the PoP.

Since it is a private registry, docker instances that require access to it must execute a login command with a valid user prior to any pull attempt, e.g.:

```
$ docker login registry.atosresearch.eu:18449/arihealth
$ docker image pull registry.atosresearch.eu:18449/arihealth/sb-data-repository:latest
```

2.4 *Virtual Machine (VM)*

As mentioned before, for this pilot all the components of SB@Cloud will be integrated into a virtual machine to demonstrate pilot use cases.

ICCS has provided an *Ubuntu 20.04.2 LTS* based VM accessible from IP 147.102.33.187 and the domain *cloud.smart-bear.eu*.

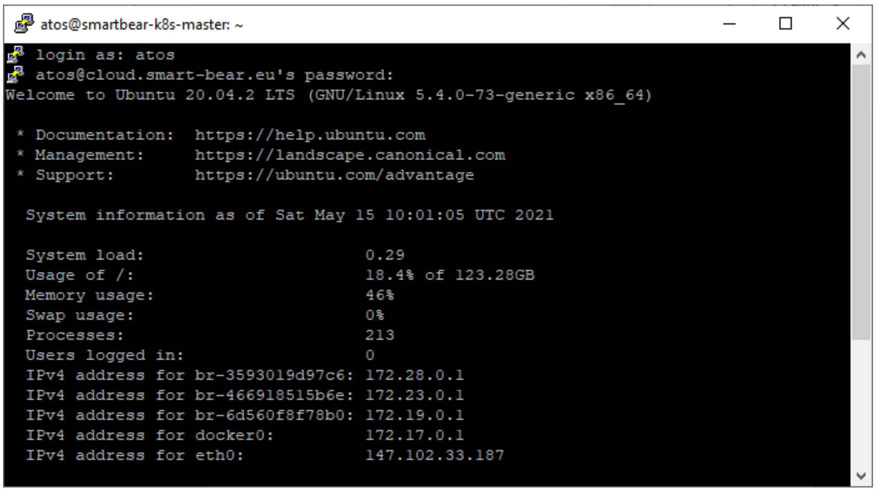

Figure 4: SSH connexion to the VM provided for the PoP

Users have been created on this machine for each component owner: *atos*, *sts*, *umil* and *beladmin* (iccs root user):

```
$ sudo useradd -s /sbin/bash -d /home/sts/ -m -G docker sts          // create sts user
$ sudo passwd sts                                                     // sets initial password for user sts
$ sudo useradd -s /sbin/bash -d /home/sts/ -m -G docker umil         // create umil user
$ sudo passwd umil                                                    // sets initial password for user umil
```

Note that if already created, users can be directly added to the docker group, otherwise the next step shows how to add them later.

2.5 *Docker*

Docker[6] is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easily share containers while you work and be sure that everyone you share with gets the same container that works in the same way.

In order to install it in the VM based on *Ubuntu,* it the root user perform the following commands

```
$ sudo apt update & sudo apt upgrade          // update the list of available packages to install
$ sudo apt install docker.io                  // install the docker command
$ sudo systemctl enable --now docker
$ docker --version                            // verify the version installed
```

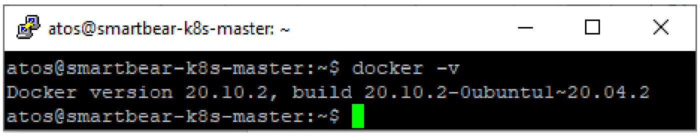

Figure 5: Check docker version installed

If the users were created previously to this installation and group *docker* was not jet created, now we can add them so they can manage docker resources.

```
$ sudo usermod -aG docker atos
$ sudo usermod -aG docker sts
$ sudo usermod -aG docker umil
```

During the development of the components, the different Dockerfile defined to build de docker images and check the creation of the containers can be tested locally, e.g.:

```
$ docker build -t sb-data-repository .          // Build the docker image
```

[6] https://docs.docker.com

2.6 *Docker-compose*

Compose[7] is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

In the same way that we have installed docker before, we also must install docker-compose utility

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.29.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose          //download the binary file
$ sudo chmod +x /usr/local/bin/docker-compose          //give execution permissions to the file
$ docker-compose -v          //test the installation
```

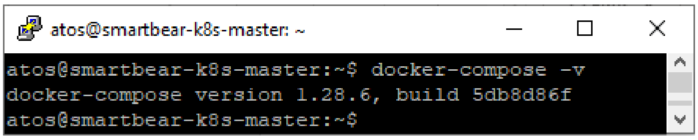

Figure 6: Check docker-compose version installed

The use of the tool is very simple once a proper yaml file has been defined, with the only precaution of considering the version of docker engine that will run it. The argument up will start the containers in the order, with the dependencies and resources defined in the file. The parameter -d is needed in case a remote logged user triggers the command to keep it as a background process. To stop all related containers, the command down can be used.

Something important to mention is that in the docker-compose files can be defined the construction of the containers (docker build command) or simply obtain of the images from the configured registry.

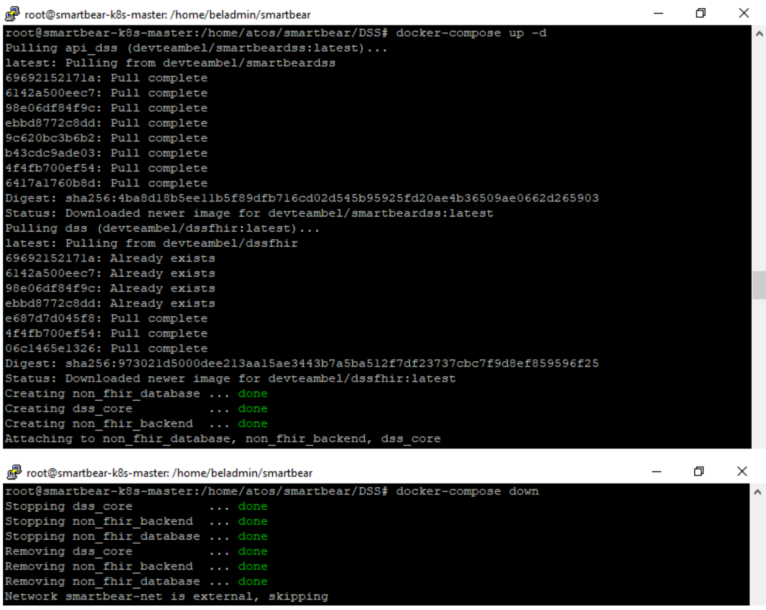

Figure 7: Docker-compose up and down commands example

[7] https://docs.docker.com/compose

2.6.1 Component sb-data-repository docker-compose yaml

The file defines two services/containers:

- `mariadb`, which starts a database engine,
- `sb-data-repository`, which is a java spring boot application containing the HAPI FHIR server.

```
version: '3.4'

services:
  mariadb:
    image: "mariadb:10.5"
    container_name: mariadb
    restart: 'always'
    volumes:
      - /data/mysql:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: #####
      MYSQL_DATABASE: smartbear_hapi
      MYSQL_USER: #####
      MYSQL_PASSWORD: #####
    networks:
      - smartbear-net

  sb-data-repository:
    image: sb-data-repository:latest
    container_name: sb-data-repository
    restart: 'always'
    ports:
      - "8080:8080"
    depends_on:
      - mariadb
    links:
      - mariadb
    networks:
      - smartbear-net

networks:
  smartbear-net:
    external: true
```

2.6.2 Component Secure_manager_one docker-compose yaml

The file defines four services/containers:

- `is`, which starts the WSO2 API Identity Server UI,
- `am`, which starts the WSO2 API Management UI,
- `postgress`, which starts the postgres DB engine,
- `usemanager`, which starts the usermanager API.

```
version: '3.4'
services:
  is:
    build: ./is/
```

```
    restart: unless-stopped
    healthcheck:
       test: ["CMD", "nc", "-z","localhost", "9443"]
       interval: 10s
      #  start_period: 180s
       retries: 20
    ports:
      - "9443:9443"
    volumes:
     - ./is/deployment.toml:/home/wso2carbon/wso2is-5.11.0/repository/conf/deployment.toml
     - type: bind
       source: ./is/newkeystore.jks
       target: /home/wso2carbon/wso2is-5.11.0/repository/resources/security/newkeystore.jks
     - type: bind
       source: ./is/wso2carbon.jks
       target: /home/wso2carbon/wso2is-5.11.0/repository/resources/security/wso2carbon.jks
    networks:
        - smartbear-net
  am:
    build: ./am/
    healthcheck:
      test: ["CMD", "nc", "-z","localhost", "9445"]
      interval: 10s
      # start_period: 180s
      retries: 20
    restart: unless-stopped
    depends_on:
       - postgres
          # condition : service_healthy
       - is
          # condition: service_healthy
    ports:
      - "9445:9445"
      - "8245:8245"
    volumes:
     - type: bind
       source: ./am/newkeystore.jks
       target: /home/wso2carbon/wso2am-3.2.0/repository/resources/security/newkeystore.jks
     - ./am/deployment.toml:/home/wso2carbon/wso2am-3.2.0/repository/conf/deployment.toml
     - ./am/libraries/SBMediator-1.0.1.jar:/home/wso2carbon/wso2am-
3.2.0/repository/components/dropins/SBMediator-1.0.1.jar
     - ./am/wso2-config-volume/:/home/wso2carbon/wso2-config-volume
    networks:
        - smartbear-net
  postgres:
        build: ./postgres/
        restart: always
        environment:
          PGDATA: /var/lib/postgresql/data/pgdata
        healthcheck:
          test: ["CMD-SHELL", "pg_isready -U regadmin"]
          interval: 10s
          timeout: 5s
          retries: 5
        ports:
            - "5432:5432"
        volumes:
          - postgresdata:/var/lib/postgresql/data
        networks:
```

```
          - smartbear-net
  usermanager:
      # build: ./usermanager
      image: user-manager
      restart: unless-stopped
      depends_on:
       - postgres
          # condition : service_healthy
      volumes:
        - ./usermanager/app:/dist/app
      #ports:
      #  - 8081:8080
      networks:
        - smartbear-net
volumes:
  postgresdata:
networks:
  smartbear-net:
    external: true
```

2.6.3 Component SB_dashboard docker-compose yaml

The file defines just one service/container:

- `sb_dashboard`, which starts the dashboard server and expose it in the port 80.

```
version: '3.4'

services:
  sb_dashboard:
    image: "sb_dashboard:latest"
    container_name: sb_dashboard
    restart: always
    expose:
      - 80
    networks:
      - smartbear-net

networks:
  smartbear-net:
    external: true
```

2.6.4 Component SB_BDA docker-compose yaml

The file defines one service/container:

- `sb_bda`, which starts the BDA application.

```
version: '3.4'

services:
  sb_bda:
    image: "sb_bda:latest"
    container_name: sb_bda
    restart: always
    ports:
      - 8001:8001
```

```
    environment:
      - CLUSTER_NAME=test
    networks:
      - smartbear-net

networks:
  smartbear-net:
    external: true
```

2.6.5 Component DSS (includes non-FHIR-database) docker-compose yaml

The file defines three services/containers:

- `database`, which starts a MariaDB database engine,
- `api_dss`, which starts the API of the DSS,
- `dss`, which starts the core application of the DSS,

```
version: '3.4'

volumes:
  datafiles:

services:
  #DataBase
  database:
    container_name: non_fhir_database
    image: mariadb
    ports:
      - "3406:3306"
    volumes:
      - /data/mysql_dss:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: '#####'
      MYSQL_DATABASE: smartbear
      MYSQL_USER: #####
      MYSQL_PASSWORD: '#####'
    networks:
     - smartbear-net

  #API DSS
  api_dss:
    container_name: non_fhir_backend
    image: devteambel/smartbeardss:latest
    depends_on:
      - database
    ports:
      - "5000:5000"
    restart: always
    environment:
      - ConnectionStrings__MySql=Server=database;Port=3306;Database=smartbear;Uid=#####;Pwd=#####
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=http://0.0.0.0:5000
    networks:
     - smartbear-net

  #DSS Core
```

```
  dss:
    container_name: dss_core
    image: devteambel/dssfhir:latest
    depends_on:
      - database
    restart: always
    environment:
      - ConnectionStrings__MySql=Server=database;Port=3306;Database=smartbear;Uid=#####;Pwd=#####
      # - Debug=true
      - ASPNETCORE_ENVIRONMENT=Development
    networks:
     - smartbear-net

networks:
    smartbear-net:
        external: true
```

2.7 *Databases*

Three of the components to integrate uses databases: *secure-manage-one* a postgres instance to store user's data, and *dss* and *sb-data-repository* respective MariaDB databases for their temporal and persistent storage.

These databases do not need a specific installation, since their engines starts as containers by their own components docker-compose files, but it is worth mentioning that persistence is achieved by mounting volumes that have their correspondence in the VM filesystem.

These definitions are also included in the docker-compose files of each component, as an example, the following section corresponding to the MariaDB configured for sb-data-repository, where the section 'volumes' map the host path with the internal container one.

```
services:
  mariadb:
    image: "mariadb:10.5"
    container_name: mariadb
    restart: 'always'
    volumes:
      - /data/mysql:/var/lib/mysql
```

2.8 *Networking, domains and ports*

To prioritise security and control access to the platform, most of the traffic accesses the platform through the *security component,* which, as a gateway, would reject any unexpected request access.

In this first version of the platform, a domain has been registered, and for the moment, all the traffic gets into the system by a single subdomain (*cloud.smart-bear.eu*) and are redirected to the different services by differentiated ports.

Component	Subcomponent	Sub-domine + port
Secure_manager_one	am	https://cloud.smart-bear.eu:8245 https://cloud.smart-bear.eu:9445
	is	https://cloud.smart-bear.eu:9443

SB_dashboard	sb_dashboard	http://cloud.smart-bear.eu:80 https://cloud.smart-bear.eu:443

Table 3: Domains and ports accessible from internet

Within the VM, when using docker compose, each set of containers by default would be part of an independent and specific virtual sub-network created at that moment. For the different components (container sets) to have access to each other, all must be configured to join the same external-network previously created.

All the docker-compose yaml files contain the following section:

```
networks:
  smartbear-net:
    external: true
```

and every interested container will join it by adding this under the specific service (container) descriptor

```
networks:
  - smartbear-net:
```

To verify all services exposed are up and running, accessing the endpoints exposed, we should get the expected responses, e.g.:

- WSO2 API Management API under: *https://cloud.smart-bear.eu:8245*

Figure 8: WSO2 API Management API

- WSO2 API Management UI under *https://cloud.smart-bear.eu:9445*

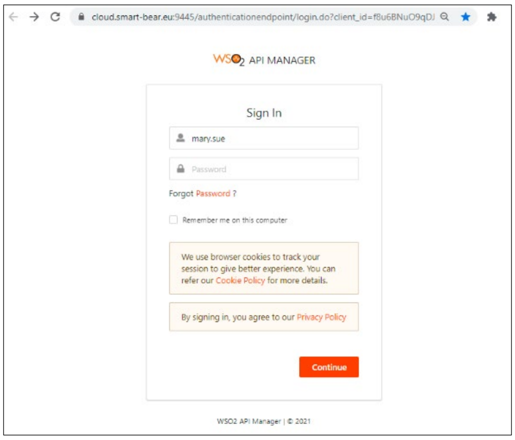

Figure 9: WSO2 API Management UI

- WSO2 Identity Server under: *https://cloud.smart-bear.eu:9443*

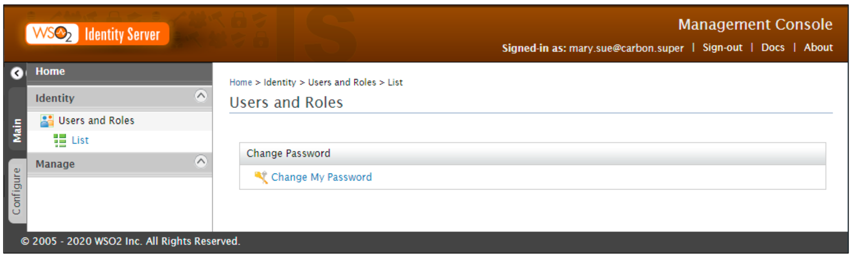

Figure 10: WSO2 Identity Server UI

- Dashboard under https://cloud.smart-bear.eu (ports http 80 or https 443)

Figure 11: SmartBear Dashboard

2.8.1 Let's Encrypt Certificates

Let's Encrypt is a non-profit Certificate Authority (CA) which provides Transport Layer Security (TLS) certificates for websites. These certificates have a short ninety-day lifetime, but its renewal can be automated through an Automatic Certificate Management Environment (ACME) client, like Certbot[8], a free, open source software tool for automatically using the Let's Encrypt certificates on manually-administrated websites to enable HTTPS.

The Cerbot website provides the step-by-step instructions and options to get the proper certificates for the installed environment. In our case the following commands for Ubuntu[9] are the needed once logged as root into the VM :

1. SSH into the server: SSH into the server running your HTTP website as a user with sudo privileges.

```
$ ssh cloud.smart-bear.eu
```

2. Installation of snapd since the instructions are based on this package manager. The VM provided, Ubuntu 20.04 LTS (Focal Fossa), has it pre-installed, so just ensure is available.

[8] https://certbot.eff.org

[9] https://certbot.eff.org/lets-encrypt/ubuntufocal-other

```
$ sudo snap install core; sudo snap refresh core
```

3. Remove certbot-auto and any Certbot OS packages in case were installed using another OS package manager like apt, dnf, or yum, to ensure that when you run the command certbot the snap is used rather than the installation from other OS package manager.
4. Install Certbot

```
$ sudo snap install --classic certbot
```

5. Prepare the Certbot command:

```
$ sudo ln -s /snap/bin/certbot /usr/bin/certbot
```

6. In case a web server is running on this machine (port 80 in use), stop the process to be able to run this command to get a certificate. Certbot will temporarily spin up a webserver on your machine. In case it cannot be stopped, arg –webroot can be used

```
$ sudo certbot certonly –standalone
```

7. Install your new certificate in the configuration file for your web server, or reverse proxy configuration in the case of SMART BEAR platform.
8. Test automatic renewal: The Certbot packages on your system come with a "*cron job*" or system timer that will renew your certificates automatically before they expire. You will not need to run Certbot again unless you change your configuration. You can test automatic renewal for your certificates by running this command:

```
$ sudo certbot renew --dry-run
```

The automatic renewal proposal installed by the package manager can be overwritten by a hand-made one, using a crontab task:

```
$ sudo crontab -e                                    //edit a crontab task
59 23 * * SUN /usr/bin/certbot renew –quiet          //check certificates every Sunday at 23:59
$ sudo crontab -l                                    //just to check if the scheduled job is listed
```

To confirm that the site is set up properly, visit https://yourwebsite.com/ in your browser and look for the lock icon in the URL bar.

To also get a *.p12 type of certificate, requested during developing for one of the components, the following command can be executed:

```
$ sudo openssl pkcs12 -export -in /etc/letsencrypt/live/cloud.smart-bear.eu/fullchain.pem -inkey
/etc/letsencrypt/live/cloud.smart-bear.eu/privkey.pem -out /home/sts/test.p12 -password pass: #####n
```

2.8.2 Reverse proxy

With two purposes, first route ports and subdomains to the proper container, and also to provide the Transport Layer Security and enable the protocol HTTPS, nginx[10] is configurated as reverse proxy in the VM.

This is a simplified configuration since the integration of nginx within a Kubernetes cluster is more complex (based on the concept of ingress controller) and do not apply to this integration in VM. Here we focus solely on protecting the dashboard with SSL, after configuring obtaining certificates from Let's Encrypt[11].

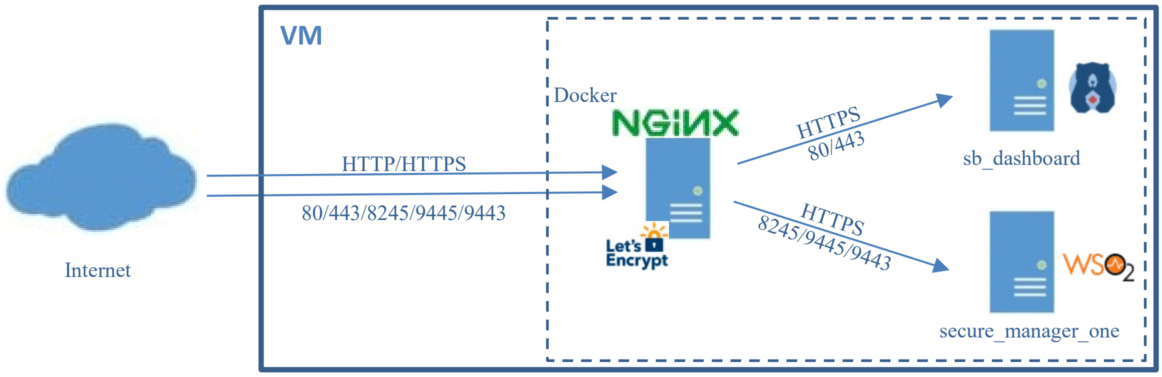

Figure 12: Reverse proxy to redirect ports and provide TLS (https)

The following configuration boot up a nginx reverse proxy configured to protect the Dashboard (ports 80 and 447) since the rest of exposed ports are secured by the own component in this prototype. In the future next Kubernetes integrated solution, all the exposed ports will be redirected and secured by the reverse proxy.

```
version: '3.4'                                   //This configuration protects the dashboard component

services:
  nginx:
    image: 'nginx:mainline-alpine'
    container_name: nginx
    restart: unless-stopped
    ports:
      - '80:80'
      - '443:443'
    volumes:
      - '/etc/nginx/conf.d:/etc/nginx/conf.d'
      - '/etc/letsencrypt:/etc/letsencrypt'
      - '/var/lib/letsencrypt:/var/lib/letsencrypt'
    networks:
      - smartbear-net

networks:
  smartbear-net:
    external: true
```

[10] https://www.nginx.com

[11] https://letsencrypt.org

3. Next steps

Following this PoP prototype integration for demo purposes, we will immediately continue working on integrating the definitive infrastructure for the SMART BEAR platform based in the Kubernetes cluster, adding the remaining CI/CD procedures and optimising the aspects that, due to lack of time prioritising the pilot, have not been carried out.

- Complete the centralisation of source code by asking the owners who are not yet maintaining their code in the common GitLab repository provided by ATOS due to practical reasons and lack of time.

- Activate CI triggers to build and deploy components automatically upon code changes and based on minimal tests for acceptance.

- Homogenising the nomenclatures of components that are not following naming rules due to rush in implementation to provide a homogeneous view of the entire platform.

- Clarify some interchangeable terms used in different components to not create confusion in the integration processes.

- Improve the data storage system in the Kubernetes cluster to ensure its maintenance, backup and privacy requirements.

- Once fully implemented, involve the different component maintainers in the knowledge and correct use of the integration processes of CI/CD.

4. Concluding Remarks

Although in the first planning it was thought that the final infrastructure of the platform would be ready for any integration including the first prototype, the reality was that other aspects of the project implementation have been prioritised and once the first demonstration needs has been reached, part of the integration had to be done manually.

In any case, most of the work advanced for this integration will be reused in the one for the Kubernetes cluster based infrastructure, which is being prepared already in parallel to the virtual machine that supports this pilot.

The sharing of the details of the components in this first integration has required the agreement and clarification of the interactions between components by each collaborator, eliminating some basic pending doubts, and marks a milestone not only in terms of integration, but also in the general implementation and improvement of the SMART BEAR platform as a whole.