Ref. Ares(2023)8830136 - 22/12/2023

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"

D6.5 (D30) - Integrated SMART BEAR platform v3

Due date of deliverable: 31/12/2023
Actual submission date: 21/12/2023

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	X
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	

D6.5 (D30) - Integrated SMART BEAR platform v3

Editors

Alberto Acebes (ATOS)

Manuel M Perez (ATOS)

Contributors

Sotiria Antaranian (ATC)

Ioannis Kouris (ICCS)

Lefteris Koumakis (STS)

Adalberto Sangalli / Gianluca Carletti (ARIA)

Jonatan Maggesi / Mattia Occhipinti (UMIL)

Reviewers

Konstantinos Ractivan (ICCS)

Eleftheria Iliadou (NKUA)

Executive Summary

The present deliverable details the functionalities, components, and tools used for building the SMART BEAR Platform in its final version, V3.

In technical terms, this version includes the complete migration to Kubernetes from the previous environment based on simple virtual machines and integrates components previously used from third-party hosting platforms (the *Android backend* and *Notifications*) and includes a new component (the *Security and Privacy Assurance Suite*). Moreover, the creation of a *staging* environment where the platform-released versions of the components are functionally tested prior to promoting them to the final and recently created Kubernetes *production* one.

Regarding the testing, validation, and support for the platform, specific activities have been carried out to ensure the transition from the Pilot of Pilots (PoP) to the large-scale pilots, and procedures implemented to provide support to the operation of the platform.

Contents

Executive Summary **3**

List of acronyms **7**

List of tables **8**

List of figures **9**

1 Introduction **10**

1.1 Overview 10

1.2 Goals of the deliverable 10

2 Changes from last version **11**

2.1 Setting up a new environment: the staging environment 11

2.2 Changes in subdomains 11

2.3 Updates in docker registry credentials 12

2.4 Integration of components previously supported by third-party services 13

2.4.1 SB Notifications component and its Mongo DB 13

2.4.2 Android App Backend component and its Mongo DB 13

2.5 Interceptor Operation in Clinical Data Repository (CDR) 13

2.6 New Security and Privacy Assurance suite (SPA) component 14

3 Release management process procedure **15**

3.1 Setting up a staging environment 15

3.2 Migration from VM-based environments 16

3.3 Versions release procedure 16

3.4 Deployment to Production environment 21

3.5 The SMART BEAR physical storage, NFS persistence 23

4 New Kubernetes cluster for PROD environment **24**

4.1 Physical resources requirements 24

4.2 Base Kubernetes installation 25

4.3 Network requirements 25

4.3.1 Public IP/domain 26

4.3.2 Firewall setup 26

4.3.3 Calico CNI plugin 26

4.4 Access to the cluster 27

4.5 Backups via VolumeSnapshots 27

4.6 Monitoring 28

5 Data migration **29**

5.1 Security Component 30

5.1.1 KeyCloak identity server 30

5.1.2 KrakenD API gateway 32

5.1.3 UserManager participants management 33

5.2 Clinical Data Repository (CDR) 35

5.2.1 Database dump, load and upgrade 35
5.2.2 Load of ATC ValuSet into the CDR 41
5.3 Decision Support System (DSS) 42
5.4 SB Notifications 43
5.5 Android App Backend 45
5.6 Big Data Analytics Engine (BDA) 46
5.7 Data migration validation 46
5.7.1 Security Component 47
5.7.2 Clinical Data Repository (CDR) 48
5.7.3 Decision Support System (DSS) 49
5.7.4 SB Notifications 49
5.7.5 Android App Backend 49
5.7.6 Big Data Analytics Engine (BDA) 49

6 Testing 50
6.1 Overall testing framework 51
6.1.1 Functional test cases 51
6.1.2 Scenarios and related test cases 51
6.1.3 Functional testing procedure 53
6.1.4 Functional testing results 54
6.2 Continuous testing during large scale pilots 56
6.3 Automatic (integration) test 57

7 Future maintenance needs 59
7.1 Resources monitoring 59
7.2 Proper backups execution 59
7.3 Kubernetes CVE 59
7.4 GitLab instance migration 60

ANNEX I. Troubleshooting 62
MariaDB can’t boot 62
MariaDB migration schema exceptions 62
Deployments wrong scaling 63
Headers too big for nginx 63
Nginx client intended to send too large body 64
Nginx failed to update lock: configmaps 64
Calico IPIP mode and performance 64
Bulk export jobs 64
DeadlineExceeded desc = context deadline exceeded 65

ANNEX II. Checklist 67

ANNEX III. Testing 68
User registration and profiling (Dashboard) 68
User registration and profiling 68
Patient Enrolment 68

Creation of a patient 69
Editing patient data (Demographics) 69
Checking exclusion criteria 70
Baseline assessment 70
Kit preparation and distribution 71
Smartwatch setup 71
Hearing aids pairing 72
SMB APP installation and devices configuration on the smartphone 72
Device management in the Dashboard 74
Monitoring 74
Heart Rate Screens 75
Blood Pressure Screen 75
Sleep Statistics Screen 75
Activity Screen 76
Weight Screen 76
Temperature Screen 77
Oximeter Screen 77
Hearing Aids Screen 77
Light Levels Screen 78
Movement Screen 78
Comfort Screen 79
Medication Screen 79
Diet Screen 80
Rehabilitation Screens 80
Measurements page (Dashboard, CCM) 81
Measurements page (Dashboard, HD operator/tech support) 81
Analytics management 81
Notebook creation 81
Showing analytics in the BDA 82
Interventions creation and delivery 82
Rule-based interventions creation from dashboard 82
Intervention creation and delivery (based on DSS rules) 83
Intervention creation and delivery (based on APP rules) 84
Interventions notification visibility 84
Synergies 85
Smart4Health synergy test 85
Holobalance synergy test 85

List of acronyms

Acronym	Description
ATC	Anatomical Therapeutic Chemical Classification System
BDA	Big Data Analytics
BGP	Border Gateway Protocol
CA	Certificate Authority
CDR	Clinical Data Repository
CI	Continuous Integration
CI/CD	Continuous Integration/ Continuous Delivery-Deployment
CCM	Clinical Case Manager
CNI	Container Network Interface
CVE	Common Vulnerabilities and Exposures
DB	Data base
DSS	Decision Support System
FHIR	Fast Health Interoperable Resource
GDS	Geriatric Depression Scale
GPL	General Public License
HA	Hearing Aid
HL7	Health Level 7
IADL	Instrumental Activities of Daily Living
IPAM	IP Address Management
JPA	Java Persistence API
MDPQ	Mobile Device Proficiency Questionnaire
MNA	Mini Nutritional Assessment
NDJSON	Newline Delimited JSON
NFS	Network File System
PoP	Pilot of Pilots
RGA	Rapid Geriatric Assessment
SANs	Subject Alternative Names
SB	SMART BEAR
SPA	Security and Privacy Assurance suite
SPAP	Security and Privacy Assurance Platform (SPAP)
SCP	Secure Copy
SFTP	Secure File Transfer Protocol
TC	Test Case
UOI	University of Ioannina
VM	Virtual Machine
WHO	World Health Organization

List of tables

Table 1: SMARTBEAR Platform environments 15
Table 2: StorageClass per environments. 23
Table 3: Kubernetes PROD cluster initial nodes setup. 24
Table 4: Storage technologies and dependencies. 29
Table 5: CDR database migration versions reference 35
Table 6: Storage technologies and dependencies. 47
Table 7: DSS DBs data validation 49
Table 8: Scenarios and related test cases for the overall testing framework 52
Table 9: Example of a test case 53
Table 10: Relationship between scenarios and platform components 53
Table 11: Functional test cases report 56
Table 12: The "core" feature set 57

List of figures

Figure 1: Software architecture of the Security and Privacy Assurance suite....14
Figure 2: VM-based environments and Kubernetes DEVELOP-STAGING and PROD clusters. 16
Figure 3: Platform component versions promoting thought environments with registry support. ...17
Figure 4: Git tag created and pointing to a specific commit. 19
Figure 5: GitLab CICD pipeline building tagged image and deploying to STAGING env. 19
Figure 6: Nexus Docker images private registry 20
Figure 7: VM-Deployment to DEVELOP and STAGING environments. 20
Figure 8: VM-Deployment to PROD environment (deploy-prod pipeline stage). 21
Figure 9: GitLab provide version to deploy on Prod environment. 22
Figure 10: GitLab allow access to project with CI_JOB_TOKEN. 22
Figure 11: NFS-filesystem environments and backups distinction. 23
Figure 12: Cluster network interfaces preview 25
Figure 13: Cluster network interfaces preview 28
Figure 14: Keycloak realm creation from previously exported one. 30
Figure 15: MariaDB upgrade check result. 38
Figure 16: Status of fly_hfj_migration table after migration process. 40
Figure 17: CDR HAPI-FHIR started with the migrated resources data. 40
Figure 18: MongoDB create new Connection. 43
Figure 19: MongoDB select to export all collections from the database. 43
Figure 20: MongoDB select JSON as export format. 44
Figure 21: MongoDB configure export. 44
Figure 22: MongoDB select to import collections. 44
Figure 23: MongoDB configure Import. 45
Figure 24: Sample data-diff tool for the validation of the ‘identities’ records. 48
Figure 25: Use of get-resources-count endpoint for counting FHIR resources 48
Figure 26: SMB Platform Assistance and Maintenance Process Framework 50
Figure 27: Tracking and managing issues in basecamp 54
Figure 28: Maintenance and testing cycle 57
Figure 29: SB-integration component 58
Figure 30: GitLab exporting “Smartbear” group and specific single project. 60

1 Introduction

This document form part of the set of deliverables describing the different versions of the SMART BEAR platform integration, with the following sequence:

- D6.7 Integrated SMART BEAR platform v0.1. *(already delivered on month 19)*
- D6.8 Report on integrated SMART BEAR platform v0.1. *(already delivered on month 19)*
- D6.1 Integrated SMART BEAR platform v1. *(already delivered on month 30)*
- D6.2 Report on integrated SMART BEAR platform v1. *(already delivered on month 30)*
- D6.3 Integrated SMART BEAR platform v2. *(already delivered on month 42)*
- D6.4 Report on integrated SMART BEAR platform v2. *(already delivered on month 42)*
- ***D6.5 Integrated SMART BEAR platform v3. (current document, to be delivered on Dec 2023)***
- *D6.6 Report on integrated SMART BEAR platform v3. (in parallel, to be delivered on Dec 2023)*

1.1 Overview

This deliverable describes the advances from the previous deliverable and the achievements for the version v3 of the SMART BEAR platform, and is composed of the following sections:

- ***2 Changes from last version***, describing significant changes from the previous version due to evolution, advancement and maintenance of the technologies used and changes in the original definition of the integration of the components.
- ***3 Release management process procedure***, about the definition of a common process for releasing different component versions and how to deploy them into a staging environment where they should intensively test, especially when it comes to new features.
- ***4 New Kubernetes cluster for PROD environment***, with the notes about the process of creation of the dedicated Kubernetes cluster for production, and differences with the previously created and dedicated for *develop* and *staging* environments.
- ***5 Data migration***, related with the Data Migration needs when dealing with different environments, as was necessary migration from the *Pilot of Pilots* (PoP) infrastructure based on Virtual Machines (VM) to the new Kubernetes cluster, emphasizing not only on how to dump the components DBs data and later import, but into the synchronization requirements between them by the use of different identifiers in a distributed way, with the technical and functional checks to perform for validating that the migration process is successful.
- ***6 Testing***, containing the testing activities implemented to ensure a safe transition from the PoP, with an environment based on simple virtual machines, to the large-scale pilots, with a structured framework based on Kubernetes.
- ***7 Future maintenance needs***, explaining the potential future needs in terms of platform maintenance, physical resources and backups monitoring, infrastructure and security updates.

1.2 Goals of the deliverable

Even during its drafting status, the present document has been useful for all the partners as an implementation guidance that has evolved along the full implementation of the SMART BEAR platform. This final version will serve as well for future reference as it defines the procedures and tasks for the platform testing, validation, and support in its use.

2 Changes from last version

Due to the evolution to this "version 3" of the platform, some changes have had to be made, due to improvements considered necessary for the development and maintenance process, as well as adaptation to the new incorporated functionalities, including a new Security and Privacy Assurance suite (SPA) component and completing, although previously introduced in the "version 2"[1] of the platform, the just partially integrated Big Data Analytics Engine (BDA) component.

2.1 Setting up a new environment: the staging environment

Initially two main environments for Smart Bear were set up: the development environment, and the production one. The former was devoted to developing and testing applications that once they are considered to work properly, they were promoted manually to the production environment. As the complexity of the platform increased due to the implementation of new functionalities and uses cases to be tested it was necessary to set up a new environment, the staging environment with the aim of further testing the functionalities too but in an environment similar to the production one.

2.2 Changes in subdomains

The addition of new execution environments to the platform, since three were planned (`develop`, `staging` and `prod`) has required changes from the initial generic preliminary naming (`k8s.`) to be more specific and descriptive for proper differentiation. This change requires some adjustments in the certificate configuration for the Kubernetes API to be used, so it would be reached via domain better than just count with the public IP. The configuration from GitLab was using the **`k8s.`** subdomain (in the CI/CD variable `K8S_SERVER`) as the endpoint to reach the Kubernetes API and needed to be adjusted to the new one.

To get rid of **`k8s.`**subdomain completely, the following changes are needed:

- The Kubernetes API server certificate *Subject Alternative Names* (SANs) needs to be updated. This cert is used to authenticate connections and encrypt traffic to/from the API server and the attempt to connect (via `kubectl` client) will fail if using a hostname or IP address not included in the certificate's list of SANs. Replacing **`k8s.`**`smartbear.eu` by **`cloud-develop.`**`smartbear.eu` could be done with the following commands within the master node of the Kubernetes cluster.

```
                                                    // Extract current configuration
$ kubectl -n kube-system get configmap kubeadm-config -o jsonpath='{.data.ClusterConfiguration}' > kubeadm.yaml

                                    // Move existing one and key, otherwise kubeadm will reject create new ones
$ mv /etc/kubernetes/pki/apiserver.{crt,key} ~

                                    // Edit the yaml file to add the proper list of Ips and domains and save
$ vi kubeadm.yaml

                                                    // Execute the cert update
$ kubeadm init phase certs apiserver --config kubeadm.yaml
```

[1] D6.3 (D28) - Integrated SMART BEAR platform v2, components: 2.5.8 Big Data Analytics Engine (BDA)

The change is verified by using the following command in the Kubernetes control plane node, decoding the certificate and showing the list of SANs included on it:

```
                                                    // Verify the subdomain is listed in the list of SANs of the certificate
$ openssl x509 -in /etc/kubernetes/pki/apiserver.crt -text
  Certificate:
    Data:
      ...
      X509v3 extensions:
          ...
          X509v3 Subject Alternative Name:
              DNS:cloud-develop-smart-bear.eu, DNS:kubernetes, DNS:kubernetes.default,
DNS:kubernetes.default.svc, DNS:kubernetes.default.svc.cluster.local, DNS:smartbear-k8s-master, IP
Address:10.96.0.1, IP Address:147.102.33.183
```

The `kube-apiserver-smartbear-k8s-master` needs to be restarted after the change, in this installation is directly a pod withing the cluster.
Finally, the `kubeadm` ConfigMap stored in the cluster should be updated for future use of kubeadm to orchestrate a cluster upgrade to consider the updated information:

```
                                                    // Update kubeadm configfile
$ kubeadm init phase upload-config kubeadm --config kubeadm.yaml

                                                    // Verify the changes were properly applied
$ kubectl -n kube-system get configmap kubeadm-config -o yaml
```

- Update of CI/CD variable `K8S_SERVER/_PROD` to point, via DNS, to the master node of the cluster.
- Notify any user of the Kubernetes API that has configured its client via a *kubeconfig* file using the subdomain to remove, to replace it by the new one in the `clusters-cluster-server` configuration.

2.3 Updates in docker registry credentials

Most of the components deployment are supported by the project's Docker private registry (Nexus[2]), where the Docker images versions are stored. This registry is accessed now by a generic user that had to be updated due to policy changes in the company hosting it (ATOS).

The credentials are stored in a secret of type `kubernetes.io/dockerconfigjson` and name `smartbear-registry-credentials`, for the `namespaces` of each environment, and although change it is something trivial, it worth to be documented for future reference. The command should be performed against the target namespace and with a docker service available.

```
                                                    // Update of smartbear-registry-credentials
$ kubectl create -n smartbear secret docker-registry smartbear-registry-credentials --docker-
server=registry.atosresearch.eu:18449 --docker-username=health --docker-password=** --docker-
email=alberto.acebes@eviden.com
```

[2] Sonatype Nexus repository, manage components, binaries and build artifacts across your entire software supply chain. (*https://www.sonatype.com/products/sonatype-nexus-repository*)

2.4 Integration of components previously supported by third-party services

This platform version 3, migrated to the Kubernetes environments, includes components and services that in previous versions were hosted in third party infrastructure and were used as external services. With the objective of centralizing resources, and better management and maintenance of the platform, the external components were migrated to the project cluster along with the rest of the components. Other tasks such as performance control, security, and backups were also centralized this way. In addition, like the rest of the components, the external components have been integrated into the CI/CD processes for control and deployment automation.

Although initially different tasks were carried out to include the *OpenHABCloud*[3] component into the cluster, with the setup of the tool, the required persistence for databases and the definition of all the deployment descriptors and CI/CD jobs, eventually the partner in charge of the component (UOI) decided to keep it in the external public *myopenhab.org* service[4].

2.4.1 SB Notifications component and its Mongo DB

Even though the *SB Notifications* component was part of the previous implementation, its DB was hosted in a public *cloud.mongodb.com*[5]. This migration consisted in defining the needed configuration, deployment descriptor files, configuring a database, transferring its data and configuring the component to be executed with the requirements in the final environment, among other tasks. The data migration process and its validation are described later in this document.

2.4.2 Android App Backend component and its Mongo DB

The *Android App Backend* was hosted in *Heroku*[6], a cloud application platform and its database (DB) was hosted in *cloud.mongodb.com* as well. The component is now moved to the instances provided within the project Kubernetes clusters. The data migration process and its validation are also described later in this document.

2.5 Interceptor Operation in Clinical Data Repository (CDR)

The migration of data collected during the PoP has revealed some extra specific needs due to the handling of large volume data. For example, the common use of resource counters provided by the CDR was not as fast as expected using the methods provided by the current HAPI-FHIR standard implementation[7] . This functionality is required to provide the summaries in the main screen of the platform dashboard, and it took longer times than expected to retrieve the data leading with time-out problems, so it was decided to create a custom *extended operation*[8] to directly attack the database with a native *count* operation.

---

[3] OpenHAB, empowering the smart home (*https://www.openhab.org*)
[4] Myopenhab.org by the openHAB Community and the openHAB Foundation.(*https://myopenhab.org*)
[5] cloud.mongodb.com , public DB service provided by the MongoDB project (*https://cloud.mongodb.com*)
[6] Heroku, a cloud platform that lets companies build, deliver, monitor and scale apps (*https://www.heroku.com*)
[7] HAPI-FHIR REST Operations: Extended Operations (*https://hapifhir.io/hapi-fhir/docs/server_plain/rest_operations_operations.html*) http://hl7.org/fhir/operations.html
[8] HL7 FHIR Extended Operations on the RESTful API (*http://hl7.org/fhir/operations.html*)

2.6 New Security and Privacy Assurance suite (SPA) component

The Security and Privacy Assurance suite (SPA) component facilitates security and privacy assessments by utilizing vulnerability analysis, penetration testing, and continuous monitoring of SMART BEAR assets. SPA is designed to conduct ongoing monitoring of several SMART BEAR platform components. It operates as a model-driven platform capable of performing hybrid security and privacy assessments, encompassing threat and vulnerability analysis, static analysis, penetration testing, and continuous runtime monitoring. SPA offers the necessary mechanisms and software components for continuous asset monitoring and the implementation of security controls to mitigate known threats and vulnerabilities. Built on the STS Assurance Platform, this platform has been tailored and integrated into the SMART BEAR platform to enable real-time, continuous assessment of the security and privacy posture of the entire deployment. Its primary purpose is to support management procedures for securing IT infrastructure, with a specific focus on the healthcare ecosystem.

To facilitate evidence collection, the platform ensures interoperability with various components and offers programmatic connectivity to key components through probes such as event captors. These probes allow the platform to gather the necessary monitoring and testing data for assessments while ensuring complete data anonymization.

The Security and Privacy Assurance Platform (SPAP), depicted in *Figure 1*: Software architecture of the Security and Privacy Assurance suite. serves as the core of this component. It is responsible for monitoring, testing, and assessing the SMART BEAR platform to ensure the framework's proper operation. The types of evidence it collects include monitoring and validation (integrity) of data and testing data. As the development progresses, additional sources may be integrated to combine runtime monitoring and dynamic runtime testing, ultimately ensuring the accurate and effective functioning of security controls.

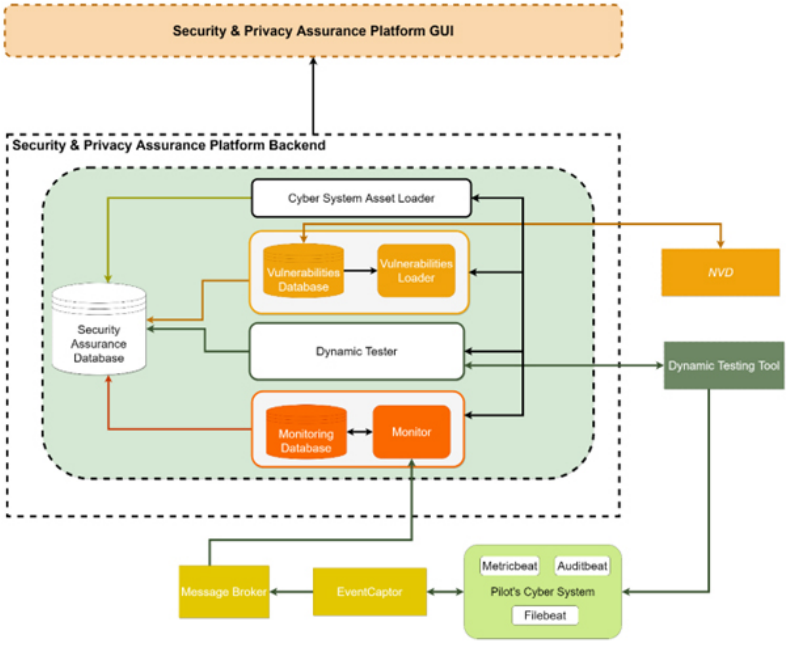

Figure 1: Software architecture of the Security and Privacy Assurance suite.

More details about the SPA suite can be found in deliverables *D5.4* and D5.6 *(Report on Continuous Security Assurance & Privacy by design - enabling mechanisms v2/v3)*.

3 Release management process procedure

Once a mayor level of development and advanced complexity has been reached in the SMART BEAR platform, a better management of the sequential versions and actual performed deployments has become necessary. Until now, and during the PoP, the responsibility and management in the integration of new functionalities, changes in them or bug fixing, was fully delegated to the component owners and so it was considered sufficient up to that point. But with the increase in complexity of the platform as a whole, it was realized the need for a formal testing phase, not just for the piloting phase but also with a view to its exploitation in a real and demanding production environment, the definition of clear procedures for its management has become an absolutely necessity, for the sake of its implementation.

The use of Kubernetes as the framework and integration of the components is supported by GitLab as a centralizing tool where the code maintenance and a series of CI/CD process (as explained in previous deliverables in this series) provides a solid foundation to manage a large part of the software development and management operations.

3.1 Setting up a staging environment

The creation of a ***develop*** **environment** on Kubernetes, based on the common use of a `smartbear-develop` namespace in the cluster, provides the first infrastructure where the most basic and preliminary integration of the different SMART BEAR platform components is put in common. The *develop* environment, although controlled by the continuous integration (CI) functionalities, is still an environment not enough stable in terms of being a target for testing functionality, since the developers of each component count on its capacity and availability to deploy changes at any time, and as soon as code changes are sent to the repository, without relying on any other partner or component operations. On the other hand, the importance of putting the software to be released through a battery of tests, before considering it suitable for being released into a production environment, makes it necessary to create an intermediate environment where specific versions of the components will be integrated in a more controlled and stable way to be tested, and this will be the named ***staging*** **environment.** In technical terms, the *staging* environment as such consists of a differentiation by a *namespace* (`smartbear-staging`) in the same physical cluster that hosts the *develop* environment. The different functionalities of Kubernetes allow us to maintain the environments isolated at the functional level despite sharing resources, in addition to the fact that their management and assignment can be done by configuration, which allows us also to optimize the necessary global resources and costs. The same will not happen with the *prod* environment, which, as a precaution in terms of security and performance, will be hosted in a fully complete and differentiated Kubernetes cluster. Following *Table 1* summarises it.

Environment	K8S Cluster	Namespace	Subdomains
DEVELOP	K8s DEVELOP/STAGING Cluster	`smartbear-develop`	`cloud-develop.smart-bear.eu`
STAGING		`smartbear-staging`	`cloud-staging.smart-bear.eu`
PROD	K8s PROD Cluster	`smartbear`	*(*)* `cloud.smart-bear.eu`
		(*) preliminarily, and before switching to the definitive domain, *cloud-**prod**.smart-bear.eu* was provided	

Table 1: SMARTBEAR Platform environments

3.2 Migration from VM-based environments

As described in previous deliverables, the urgency of bringing the platform to actual use in the PoP meant that the initial idea of integrating all the components in a Kubernetes-based environment was not possible from the beginning. Instead, a simpler integration was set up based on Virtual Machines and Docker compose. The limitations of this type of integrations were reported in previous deliverables[9] Since then, the balance has been maintained between the priority of solving the issues arising in the functionality, the implementation of new requirements and the plan to perform the effective migration to the Kubernetes cluster for setting up a CI/CD framework. Furthermore, as infrastructure resources (disk space, memory) are not unlimited, and planning for the reuse of those released has always been considered. *Figure 2* schematize this scenario.

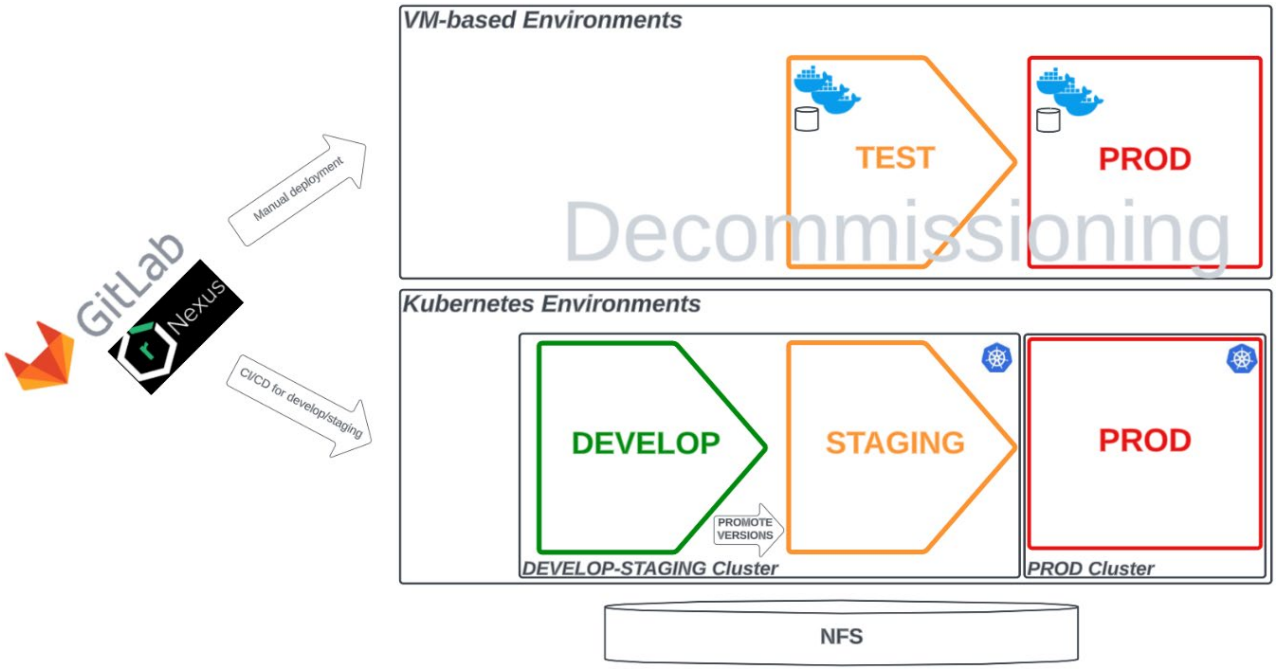

Figure 2: VM-based environments and Kubernetes DEVELOP-STAGING and PROD clusters.

3.3 Versions release procedure

The SMART BEAR platform is made up of several components and subcomponents (as a non-exhaustive reminder: the *SecurityComponent* and its submodules: *Keycloak*, *KrakenD* and *UserManager*), the *android-app-backend*, the *clinical-data-repository*, the *Decision Support System* (*DSS)*, the *Big Data Analytics* (*BDA)* and its submodules, *SB_Notifications* component, *SB_Dashboard*, and they are implemented and maintained by several project partners, without forgetting that they are developed in different technologies (java, .NET, Vue.js, etc.).

The process of how to *promote* through the environments in a controlled integration of the different developments is formalized and agreed under this proposed *release* process in a way that has been suitable to all of them, being sufficiently flexible and not too effort demanding and considering the

[9] Deliverables D6.1, D6.7 as well D6.2 and D6.8.

scope of the project. Furthermore, the possibilities offered by *GitLab* and, in turn, by Git[10], have been used to control the version of the code in the different releases.

Following the procedure is extremely important due the fact that the figure of a *release manager*[11] is not foreseen in the SMART BEAR platform, and each component owner is responsible for their own development deployments.

Here we describe the *terminology* used to distinguish the different environments in the migration from the preliminary VM-based environments to, and within the Kubernetes clusters:

- The *VM-based* environments (decommissioned) where two:
  - **TEST**, the old integration testing environment.
  - **PROD**, the production environment used during the *Pilot of Pilots* phase.

- The *Kubernetes* environments are three, as depicted in *Figure 3*:
  - **DEVELOP**, the development environment based on CI/CD functionality. *GitLab* will deploy the 'develop' branch automatically when committing changes based on the instructions described under the `.gitlab-ci.yml` file.
  - **STAGING**, the environment specifically set up for functional testing purposes. It is a pre-production one, where the creation of Git 'tags' will trigger Docker image builds with specific version tags to be deployed to this environment.
  - **PROD**, the final production environment.

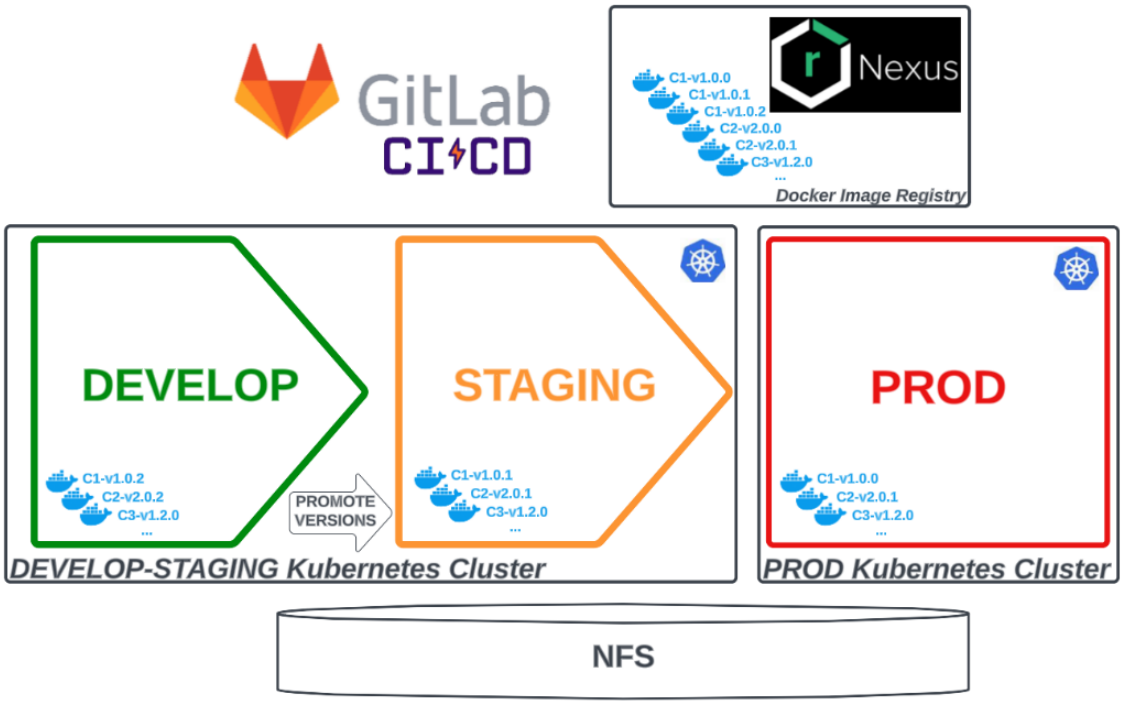

Figure 3: Platform component versions promoting thought environments with registry support.

[10] Git, a distributed version control system (*https://git-scm.com*)
[11] *Release manager*; a skillful person in charge of centralizing the release the latest version of the different components.

Each component, under the GitLab source code project `/deployment/kubernetes` folder differentiate three subfolders where reside the manifests for k8s resources or helm charts deployment configuration in any of the environments.

- **DEVELOP**: `/deployment/kubernetes/cloud-develop.smart-bear.eu`,
  - resources are created under `smartbear-develop` *namespace,*
  - persistence depends on `smartbear-nfs-csi-storageclass-develop` *storageClass*
  - docker build and push to the registry image versions tagged as `latest`,
  - deployment manifests rely on such `latest` version,
  - the `.gitlab-ci.yml` file will:
    - The job refers to `develop` Git branch and the set of develop configuration files.
    - Use the `smartbear-develop` namespace where needed.
    - Use the `environment:develop` for tagging the specific deployment job.
    - Create *secret* resources from the `PROJECT_K8S_CONFIG` CI/CD variable, where credentials or any other critical data will be stored rather than stored inside the source code.
- **STAGING**: `/deployment/kubernetes/cloud-staging.smart-bear.eu`, similar to the previous, slightly adapted:
  - resources are created under `smartbear-staging` *namespace,*
  - persistence depends on `smartbear-nfs-csi-storageclass-staging` *storageClass,*
  - docker build and push to the registry image component specific fixed tagged versions,
  - deployment manifests rely on such fixed version,
  - the `.gitlab-ci.yml` file will:
    - include a *manual* job, with `only: refs: - tags` pointing to the set of develop configuration files,
    - use the `smartbear-staging` namespace where needed,
    - use the `environment:staging` for tagging the specific deployment job, and
    - create *secret* resources from the `PROJECT_K8S_CONFIG` CI/CD variable, where credentials or any other critical data will be stored rather than stored inside the source code.
- **PROD**: `/deployment/kubernetes/cloud.smart-bear.eu`, unlike what happens in previous **DEVELOP** and **STAGING** set up, where the files are used automatically by the GitLab CI/CD functionalities, the files for **PROD** are just listed in order to be available for reference during deployments to production, via manual procedures, or the automatization described in this document *3.4 Deployment to Production environment* section.

Regarding the promotion of the different versions through the environments, when the code is ready by basic integration validation under the development environment for being promoted to staging, the recommendation is the `develop` branch to be merge into the `master/main`, and in that moment provide the changes needed for build a fixed docker version (*Figure 4*), and not use the 'latest' tag anymore. Depending on the technology will require adjusting the docker build command to use a specific version tag in the *DockerFile*, the version in a configuration file, etc.

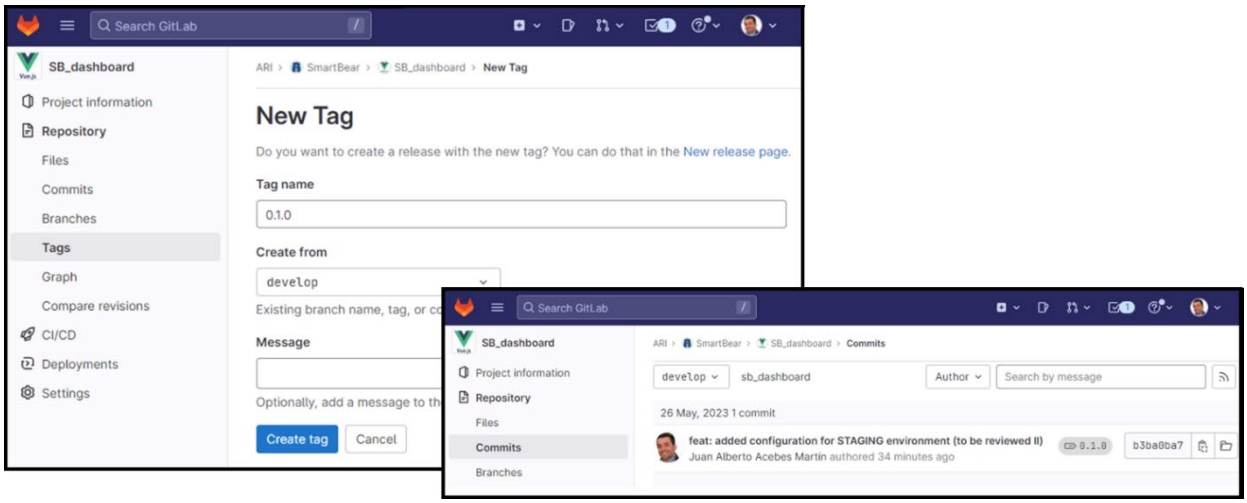

Figure 4: Git tag created and pointing to a specific commit.

The configuration will build and push to Nexus the docker image with the specific version as tag, and eventually execute the proper command for deploying it into the **STAGING** environment (*Figure 5*).

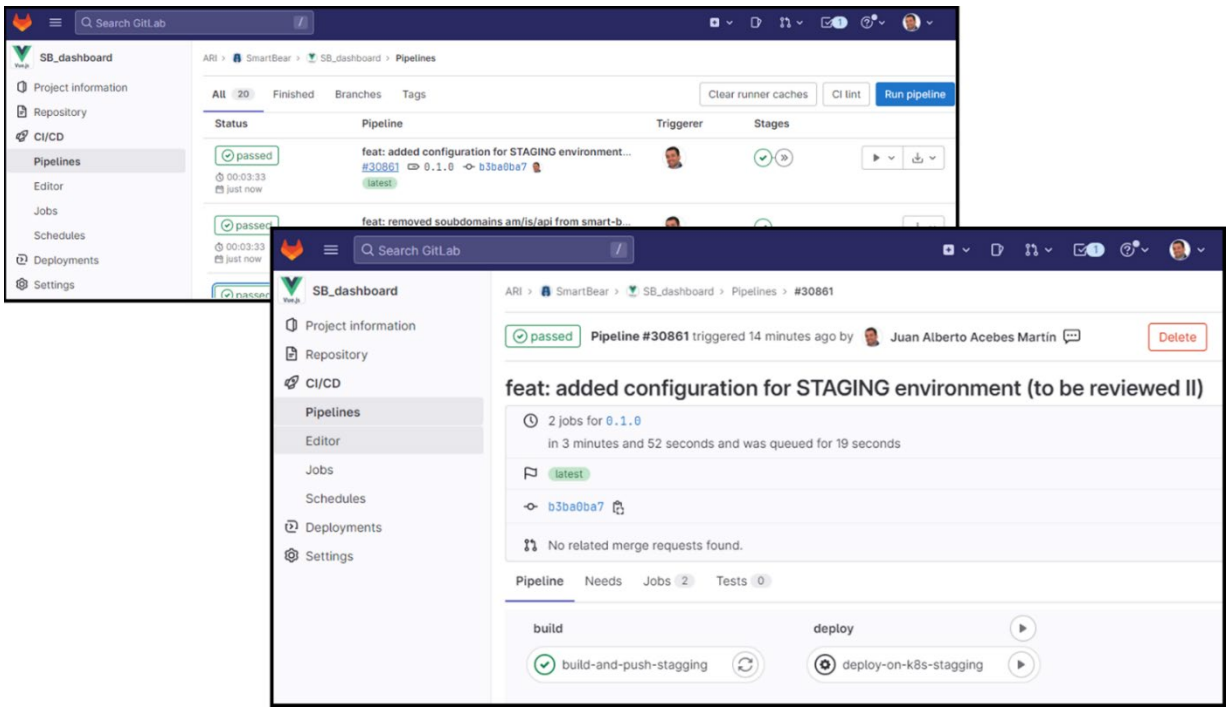

Figure 5: GitLab CICD pipeline building tagged image and deploying to STAGING env.

Also note that, from the previous step on, the docker image generated and pushed will be stored in the project docker images private registry (Nexus), and available for future promotion to **PROD** environment (*Figure 6*).

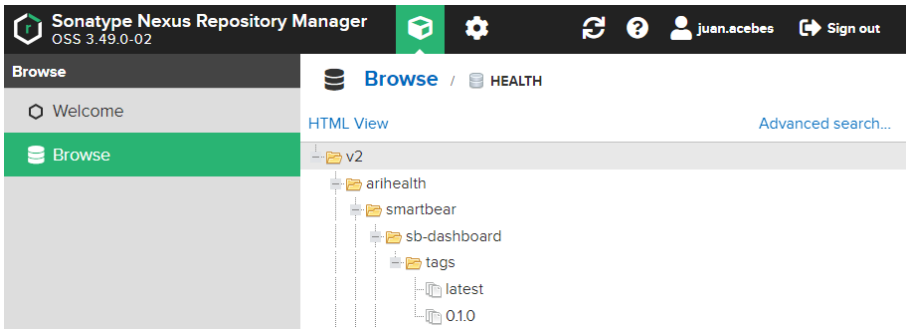

Figure 6: Nexus Docker images private registry

Schematically, the following flowchart[12] represents the CI/CD processes for the develop and staging environments (*Figure 7*).

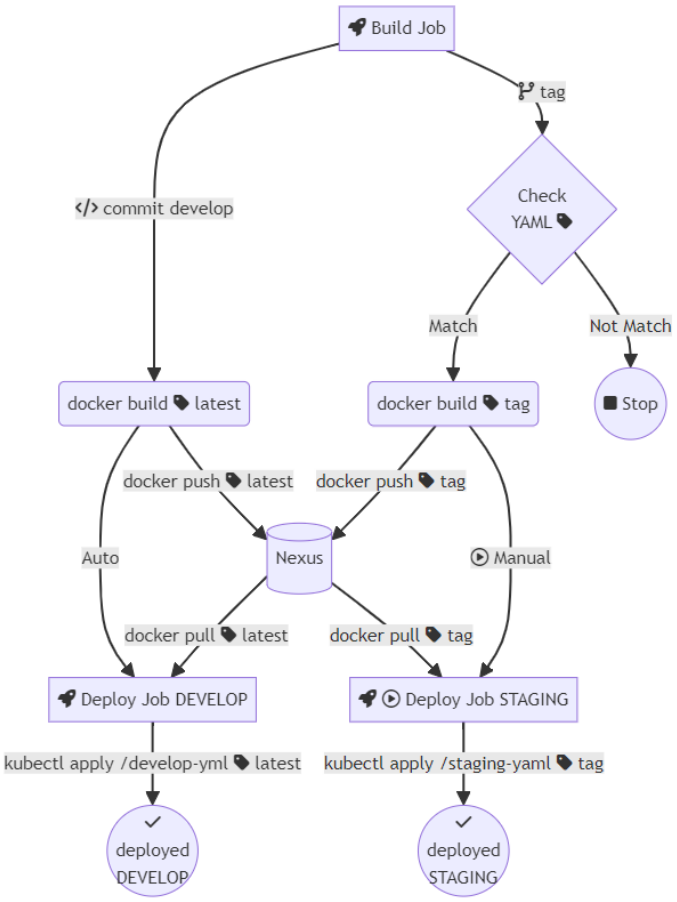

Figure 7: VM-Deployment to DEVELOP and STAGING environments.

At cluster level, configuration can rely on the `smartbear-registry-credentials` secret that contains the details for dealing the private docker registry, where the images are available.
The project partners maintain a common place, the GitLab project WIKI[13], where to keep record of the versions deployed in each environment and keep other partners informed of the current deployment status, especially for **STAGING** and **PROD**.

[12] Mermaid Diagramming and charting tool flowChart (*https://mermaid.js.org/syntax/flowchart.html?id=flowcharts-basic-syntax*)
[13] SMART BEAR GitLab WIKI, (*https://gitlab.biomed.ntua.gr/smartbear/smartbear/-/wikis/home*)

3.4 Deployment to Production environment

Although it is each component owner responsibility the way of deploying its components into the **PROD** environment, a convenient manner procedure is suggested, based on a CI/CD pipeline defined under the GitLab `/smartbear` project (*Figure 8*), centralizing the deployment of the desired versions to the **PROD** environment. The CI/CD Pipeline configuration include a block of jobs grouped under the `deploy-prod` *stage* (and `Prod` label environment) ready to be used to execute the needed commands against the **PROD** cluster. Several CI/CD variables (`$K8S_SERVER_PROD`, `$K8S_TOKEN_PROD` and `$K8S_CA_CERT_PROD`) have been configured with a proper (and limited) `sb-gitlab-cicd-sa` *ServiceAccount* in order to be able to perform the needed deployment operations. Secrets are not created automatically and should be maintained manually as a security measure.

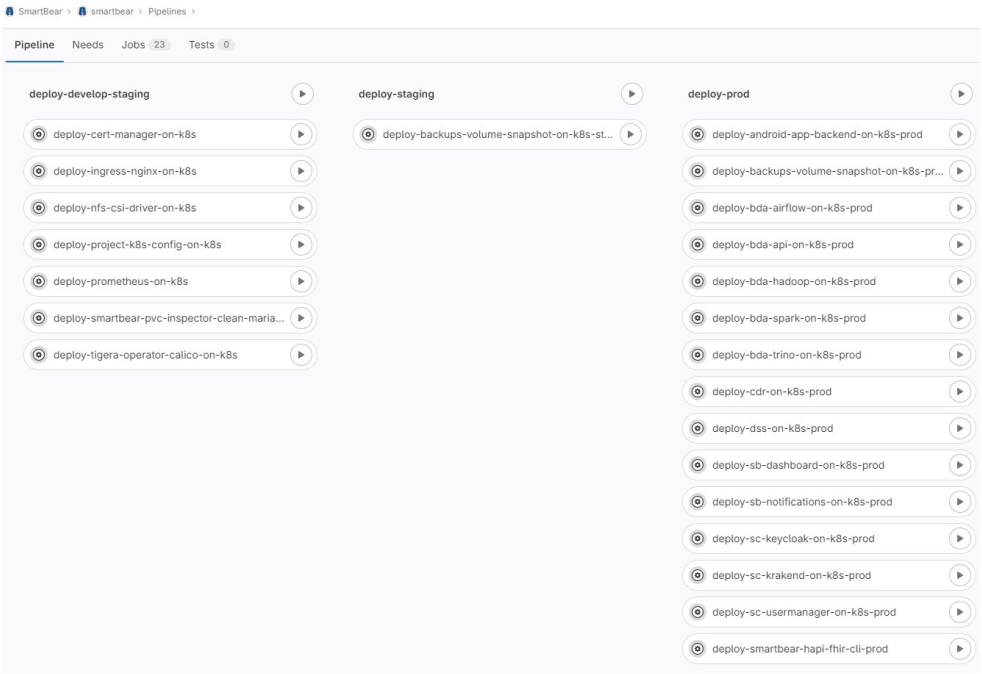

Figure 8: VM-Deployment to PROD environment (deploy-prod pipeline stage).

By default, the version of the components to deploy is the one configured under the deployment configuration under the `master`/`main` branch. If instead, a specific version defined in a TAG should be used, the following variables must be provided to the pipeline *job* (*Figure 9*).

- Security Component:
  - Keycloak: `VERSION_SC_KEYCLOAK`
  - KrakenD: `VERSION_SC_KRANKEND`
  - UserManager: `VERSION_SC_USERMANAGER`
- CDR: `VERSION_CLINICAL_DATA_REPOSITORY`
- SB_dashboard: `VERSION_SB_DASHBOARD`
- android-app-backend: `VERSION_ANDROID_APP_BACKEND`
- sb-notifications: `VERSION_SB_NOTIFICATIONS`
- DSS: `VERSION_DSS`
- BDA:
  - bda-api: `VERSION_BDA_API`
  - airflow: `VERSION_BDA_AIRFLOW`

- hadoop: `VERSION_BDA_HADOOP`
- hive-metastore: `VERSION_BDA_HIVEMETASTORE`
- spark: `VERSION_BDA_SPARK`
- trino: `VERSION_BDA_TRINO`
- zeppelin: `VERSION_BDA_ZEPPELIN`

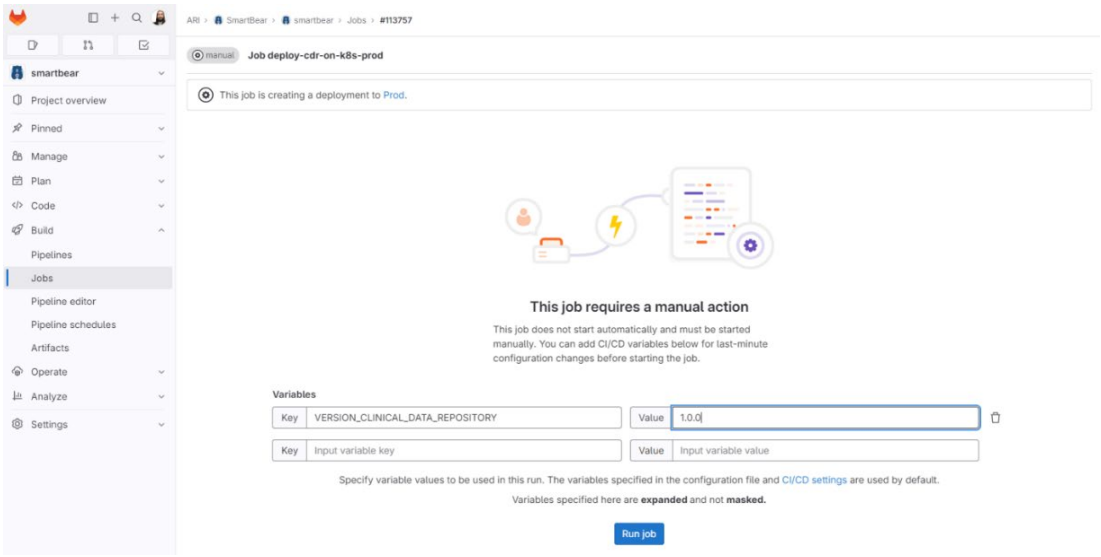

Figure 9: GitLab provide version to deploy on Prod environment.

It is worth mentioning that in order the `/smartbear` project CI/CD jobs to be able to reach other repositories files, and since the common `CI_JOB_TOKEN` cannot be used in the endpoint, and for the API to provide direct access to files (`/projects/:id/repository/files`[14]), we have to enable a `GROUP_ACCESS_TOKEN` and use `PRIVATE-TOKEN` header in the requests to such GitLab endpoint. Note also that the `GROUP_ACCESS_TOKEN` could expire and, in such case, should be re inserted in the configured global CI/CD variable (*Figure 10*).

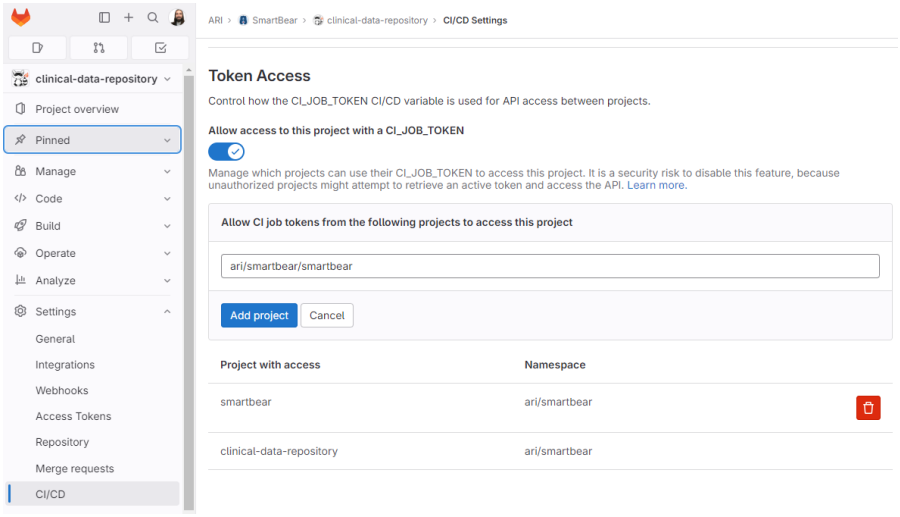

Figure 10: GitLab allow access to project with CI_JOB_TOKEN.

[14] GitLab Repository files API (*https://docs.gitlab.com/ee/api/repository_files.html*)

3.5 The SMART BEAR physical storage, NFS persistence

Although the three environments are separated into two different clusters, the NFS-based persistence service will be shared and provided by the same machine since it has been created with the basic security, storage capacity and replication feature necessary for the platform for any of its environments.

In order to maintain some order in the file system, under the `smartbear-k8s` folder of the NFS shared root path, different folders have been created and three *StorageClass* Kubernetes resources has being configured pointing to each of them, as summarized in the following *Table 2*Table 1.

Environment	Namespace	StorageClass	Path
		`smartbear-nfs-csi-storageclass` *(*)*	`/smartbear-k8s`
DEVELOP	`smartbear-develop`	`smartbear-nfs-csi-storageclass-develop`	`/smartbear-k8s/develop`
STAGING	`smartbear-staging`	`smartbear-nfs-csi-storageclass-staging`	`/smartbear-k8s/staging`
PROD	`smartbear`	`smartbear-nfs-csi-storageclass-prod`	`/smartbear-k8s/prod`
(*) For common tools storage needs, like Prometheus in charge of monitoring			

Table 2: StorageClass per environment.

In addition, a folder has been reserved for the "backups" produced by the *VolumenSnapshots* functionality (*Figure 11*), described in deliverable *D6.3 (D28) - Integrated SMART BEAR platform v2 - 2.3 Backup's strategy with VolumeSnapshot*. This *StorageClass* also have the precaution of pointing and creating the backups files on another physical machine.

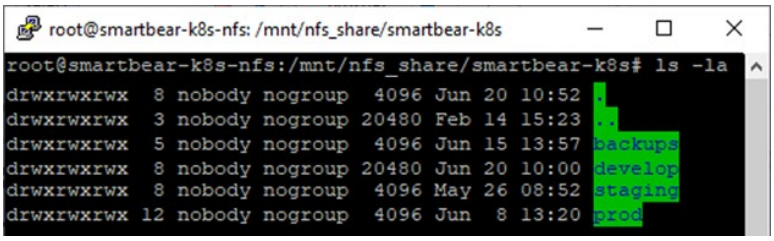

Figure 11: NFS-filesystem environments and backups distinction.

4 New Kubernetes cluster for PROD environment

As previously mentioned, the maintenance of the different Kubernetes-based environments (DEVELOP, STAGING and PROD) was physically divided into two differentiated clusters, for ensuring safety and maintenance, as an initial design decision, for the SMART BEAR platform services provision. Since the installation of the first cluster (dedicated for DEVELOP and STAGING environments), some trends have changed in terms of advances in the required modules and their way of being installed and managed.

In the previous deliverable *D6.1 Integrated SMART BEAR platform v1*, its chapter *3.2 Installation of Kubernetes cluster*, dealt with its base installation: networking, persistence based on NFS, DNS resolution, cluster monitoring dealing with certificates, etc.

Afterwards, the deliverable *D6.3 Integrated SMART BEAR platform v2*, in its chapter *2.1 Upgrade and maintenance of Kubernetes cluster*, also had to dealt with upgrades and improvements in the infrastructure of the cluster, to keep it up to date and align with trends and needs, which ensures safety and maintenance.

The Kubernetes PROD cluster is intended to be composed by three independent VMs all of them based on Ubuntu 22.04.2.0.LTS installations, identified by the following hostnames:

- `smartbear-k8s-prod-master` `147.102.33.194` `control-plane node`
- `smartbear-k8s-prod-node-0` `147.102.33.195` `worker node`
- `smartbear-k8s-prod-node-1` `147.102.33.196` `worker node`

The firsts two were already set up and were running with the configuration detailed below in this document during the initial cluster installation. The node `smartbear-k8s-prod-node-1` was joined later to allow the testing of the components that could require balance its work in more than a single node.

4.1 Physical resources requirements

In the same way as it happened with the cluster dedicated to the DEVELOP and STAGING environments, the machines prepared as Kubernetes *master* and *worker* nodes have been set up based on *Ubuntu 22.04.3 LTS* installations, with the following initial configuration described in *Table 3*, although they are pending allocation of more resources based on the need and release of other environments. as it mentioned above. Since are virtual machines, they can be transparently scaled at any time in terms of processor, memory or storage space depending on the available and needed resources.

	Master smartbear-k8s-prod-master	Worker smartbear-k8s-prod-node-0	Worker smartbear-k8s-prod-node-1
CPU	4 vCPU	4 vCPU	4 vCPU
RAM	8GB	12GB	24GB
DISK	120GB	120GB	120GB
			Note that these values are the initial setup but could be changed anytime

Table 3: Kubernetes PROD cluster initial nodes setup.

4.2 Base Kubernetes installation

The base installation of the Kubernetes cluster is similar to the one described for the DEVELOP/STAGING cluster, with the difference of the IPs and subdomains in use, not forgetting key points how to deactivate memory swap, and the extra *containerd* configuration.

// Containerd configuration

```
kubeadm init --pod-network-cidr=192.168.0.0/16 --apiserver-advertise-address=147.102.33.194 --apiserver-
cert-extra-sans=147.102.33.194
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
kubeadm join 147.102.33.194:6443 --token **** --discovery-token-ca-cert-hash sha256:***
--apiserver-cert-extra-sans METER cloud.smart-bear.eu…!!
/etc/hosts file with a reference to my cluster endpoint name kube-master
kubectl taint node smartbear-k8s-prod-master node-role.kubernetes.io/control-plane:NoSchedule-
```

For the installed version, an extra *containerd*[15] configuration has also been necessary, or the containers restarted randomly (kill signals error code 143 sent to pods), losing network connectivity with the cluster.

// Containerd configuration

```
sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml
```

4.3 Network requirements

In order to set up a functional cluster, the following networking requirements needs to be fulfilled, all of them keeping the idea of limiting access to the maximum and prevent security issues such non-permitted accesses and flaws in the security of the platform that led to data compromise. Network interfaces were identified and considered for the following configuration needs of the different nodes of the cluster (*Figure 12*).

```
Cloud-init v. 23.1.2-0ubuntu0~22.04.1 running 'init' at Tue, 30 May 2023 09:20:08 +0000. Up 18.53 seconds.
ci-info: ++++++++++++++++++++++++++++++++++++++++++++++Net device info++++++++++++++++++++++++++++++++++++++++++++++
ci-info: +--------+------+-----------------------------------------+---------------+--------+-------------------+
ci-info: | Device |  Up  |                 Address                 |      Mask     | Scope  |     Hw-Address    |
ci-info: +--------+------+-----------------------------------------+---------------+--------+-------------------+
ci-info: |  eth0  | True |              147.102.33.194             | 255.255.255.0 | global | 00:15:5d:21:e5:05 |
ci-info: |  eth0  | True | 2001:648:2000:21:215:5dff:fe21:e505/64  |       .       | global | 00:15:5d:21:e5:05 |
ci-info: |  eth0  | True |       fe80::215:5dff:fe21:e505/64       |       .       |  link  | 00:15:5d:21:e5:05 |
ci-info: |   lo   | True |                127.0.0.1                |   255.0.0.0   |  host  |         .         |
ci-info: |   lo   | True |                 ::1/128                 |       .       |  host  |         .         |
ci-info: +--------+------+-----------------------------------------+---------------+--------+-------------------+
ci-info: ++++++++++++++++++++++++++++++Route IPv4 info++++++++++++++++++++++++++++++
ci-info: +-------+--------------+---------------+---------------+-----------+-------+
ci-info: | Route | Destination  |    Gateway    |    Genmask    | Interface | Flags |
ci-info: +-------+--------------+---------------+---------------+-----------+-------+
ci-info: |   0   |   0.0.0.0    | 147.102.33.200 |    0.0.0.0    |    eth0   |   UG  |
ci-info: |   1   | 147.102.33.0 |    0.0.0.0    | 255.255.255.0 |    eth0   |   U   |
ci-info: +-------+--------------+---------------+---------------+-----------+-------+
ci-info: +++++++++++++++++++++++++++++Route IPv6 info+++++++++++++++++++++++++++++
ci-info: +-------+---------------------+-------------------------+-----------+-------+
ci-info: | Route |     Destination     |         Gateway         | Interface | Flags |
ci-info: +-------+---------------------+-------------------------+-----------+-------+
ci-info: |   1   | 2001:648:2000:21::/64 |            ::           |    eth0   |   Ue  |
ci-info: |   2   |      fe80::/64      |            ::           |    eth0   |   U   |
ci-info: |   3   |        ::/0         | fe80::aec:f5ff:fed0:d91d |    eth0   |  UGe  |
ci-info: |   5   |        local        |            ::           |    eth0   |   U   |
ci-info: |   6   |        local        |            ::           |    eth0   |   U   |
ci-info: |   7   |      multicast      |            ::           |    eth0   |   U   |
ci-info: +-------+---------------------+-------------------------+-----------+-------+
```

Figure 12: Cluster network interfaces preview.

[15] Containerd, An industry-standard container runtime (*https://containerd.io*)

4.3.1 Public IP/domain

A public IP or subdomain redirection is necessary to reach the `smartbear-k8s-prod-master` node from public internet, and the subdomain `https://cloud.smartbear.eu` that was pointing to the previous environment, eventually will point the new PROD cluster. Also, and for testing purposes, the temporary subdomain `https://cloud-prod.smartbear.eu` were pointing to the same machine prior to the final switching of the main subdomain once the migration is concluded.

4.3.2 Firewall setup

The nodes that make up the cluster are configured in the same way the previous cluster was done, since the set of services and reserved ports required remains with the same setup needs as described in previous deliverables.

4.3.3 Calico CNI plugin

Also previously mentioned, by default Kubernetes provide some basic network connectivity, but it is common to use third party network implementations that use the Kubernetes Container Network Interface (CNI) and as a Pod Network Add-On allow the pods to communicate more effectively.

Although the lasts versions of Calico CNI plugin strongly recommend the use of a *Helm Chart*[16] to perform the installation, via the so-called *Tirega operator*, its degree of maturity or base setup assumptions does not seem to be appropriate for this installation, so a customized setup based on manifests has been carried out, as was done in the previous cluster.

Taking the same precautions that were performed on the previous occasion, with the network interface autodetection (`IP_AUTODETECTION_METHOD=interface=eth*` , check that matches with `ifconfig`) and the BGP type of network, we start by executing the following commands in each node.

// Calico CNI plugin installation in the cluster nodes

```
$ curl -L -o /opt/cni/bin/calico https://github.com/projectcalico/cni-
plugin/releases/download/v3.21.2/calico-amd64
$ chmod 755 /opt/cni/bin/calico
$ curl -L -o /opt/cni/bin/calico-ipam https://github.com/projectcalico/cni-
plugin/releases/download/v3.21.2/calico-ipam-amd64
$ chmod 755 /opt/cni/bin/calico-ipam
```

Then, from the control plane node we create a *kubeconfig* file (`cni.kubeconfig`) for the CNI plugin to access Kubernetes also from each node, based on the instructions provided by the Calico community project[17], and copying it to each of the worker nodes as `/etc/cni/net.d/calico-kubeconfig/` (with `600` permissions). After that was enough for them to connect each other properly and be shown as 'ready' in the cluster, which means that also the *ipam* (IP address management) plugin was properly configured.

// Switch the Boolean flag in SystemCgroup within config.toml

```
$ sudo sed -i 's/  SystemdCgroup = false/   SystemdCgroup = true/' /etc/containerd/config.toml
```

[16] Helm Chart, Helm is the best way to find, share, and use software built for Kubernetes. (*https://helm.sh*)
[17] Calico CNI plugin install (*https://docs.tigera.io/calico/latest/getting-started/kubernetes/hardway/install-cni-plugin*)

4.4 Access to the cluster

Unlike the Kubernetes develop/staging cluster, where in order to execute GitLab runners and other CI/CD functionality, user accounts and tokens were created with default suggestions without stressing too much the security aspect, in the case of the new one intended for the production environment, access to it will be limited to the extreme, tuning to the specific needs of administration and necessary automation. The administration user, with the maximum permissions (including `cluster-admin` role) is reserved exclusively for low-level maintenance and monitoring of the cluster by the administrators. The other necessary access provided is for the deployment of components from the functionality enabled and explained in the next section (*4.5 Backups via VolumeSnapshots*). It is worth mentioning that the latest versions of Kubernetes no longer create a token by default in their installation, as happened in the setup of the first of the clusters, and now it should be created manually.

The following commands describe the creation of the triplet `UserAccount/Roles/ClusterRoleBinding`, and a *ServiceAccount* token (stored as `Secret` with a certificate authority CA certificate) associated to the `UserAccount`. That token will be used as bearer token[18] in the header of the API client `kubectl` requests from the scripts executed from GitLab CI/CD.

```
                                                    // Creation of ServicerAccount (namespace default)
$ kubectl create serviceaccount sb-gitlab-cicd-sa

                                                    // Manually create a long-lived token for such ServiceAccount
$ kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: sb-gitlab-cicd-secret
  annotations:
    kubernetes.io/service-account.name: sb-gitlab-cicd-sa
type: kubernetes.io/service-account-token
EOF
$ kubectl describe secrets/sb-gitlab-cicd-secret

$ kubectl create secret generic sb-gitlab-cicd-sa-token-sercret --type=kubernetes.io/service-account-
token
                                                    // Creation of Role with the minimum rules and ClusterRoleBinding
$ kubectl create clusterrole sb-gitlab-cicd-clusterrole --verb=get,list,watch,create,update,patch,delete
--resource='*'
$ kubectl create clusterrolebinding sb-gitlab-cicd-sa-clusterrolebinding --clusterrole='sb-gitlab-cicd-
clusterrole' --serviceaccount='default:sb-gitlab-cicd-sa' --namespace='default'
```

4.5 Backups via VolumeSnapshots

For basic backup, and as continuation of the *VolumeSnapshot* functionality described in Deliverable D6.3, three *CronJobs* has been configurated to regenerate a series of volume snapshots with the following *cron* expressions[19]:

- `smartbear-backups-daily-cronjob`: Daily, every day at 12:00 AM (`0 0 * * *`)
- `smartbear-backups-daily-cronjob`: Weekly, at 12:00 AM, only on Monday (`0 0 * * MON`)
- `smartbear-backups-daily-cronjob`: Monthly, At 12:00 AM, on day 1 of the month (`0 0 1 * *`)

[18] Kubernetes Authenticating (*https://kubernetes.io/docs/reference/access-authn-authz/authentication/#service-account-tokens*)
[19] Cron Expression, string comprising five or six fields that represents a set of times to schedule the execution of a routine (*https://en.wikipedia.org/wiki/Cron*)

```
                                                    // First execution of smartbear-backups-daily-cronjob
********************************** SMART BEAR **********************************
*********************** smartbear-backups-daily-cronjob ************************
* Job trigered from smartbear-backups-daily-cronjob CronJob...
  - APISERVER: https://kubernetes.default.svc
  - SERVICEACCOUNT: /var/run/secrets/kubernetes.io/serviceaccount
  - NAMESPACE: smartbear
  - CACERT: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
  - TOKEN: ***
fetch http://dl-cdn.alpinelinux.org/alpine/edge/community/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.18/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.18/community/x86_64/APKINDEX.tar.gz
(1/1) Installing kubectl (1.28.1-r1)
Executing busybox-1.36.1-r2.trigger
OK: 59 MiB in 19 packages
* Removing previous DAILY Jobs...
  - Checking previous completed *smartbear-backups-daily-cronjob-* jobs...
  - No smartbear-backups-daily-cronjob-* jobs found.
* Removing previous DAILY VolumeSnapshots...
No resources found in smartbear namespace.
  - No *-daily-volumesnapshot volumesnapshots found.
* Creating new DAILY VolumeSnapshots...
volumesnapshot.snapshot.storage.k8s.io/smartbear-sc-keycloak-daily-volumesnapshot created
volumesnapshot.snapshot.storage.k8s.io/smartbear-sc-usermanager-daily-volumesnapshot created
volumesnapshot.snapshot.storage.k8s.io/smartbear-cdr-daily-volumesnapshot created
volumesnapshot.snapshot.storage.k8s.io/smartbear-android-app-backend-daily-volumesnapshot created
volumesnapshot.snapshot.storage.k8s.io/smartbear-sb-notifications-daily-volumesnapshot created
volumesnapshot.snapshot.storage.k8s.io/smartbear-dss-daily-volumesnapshot created
* CronJob smartbear-backups-daily-cronjob END.
```

4.6 Monitoring

A utility that allows monitoring basic aspects of the cluster, although they are not detailed metrics of each component, is essential to be able to deal with issues quickly and efficiently with the behaviour of the cluster, providing historical info about physical resources in use and potential peaks. Cluster monitoring is performed based on Prometheus[20] and installed via its default official Helm Chart, process already described in the staging/development cluster creation.

The *Figure 13* below shows a screenshot of *OpenLens*[21] using the monitoring tool in the cluster to provide visualization of CPU and memory consumption as well as running processes and its readiness status.

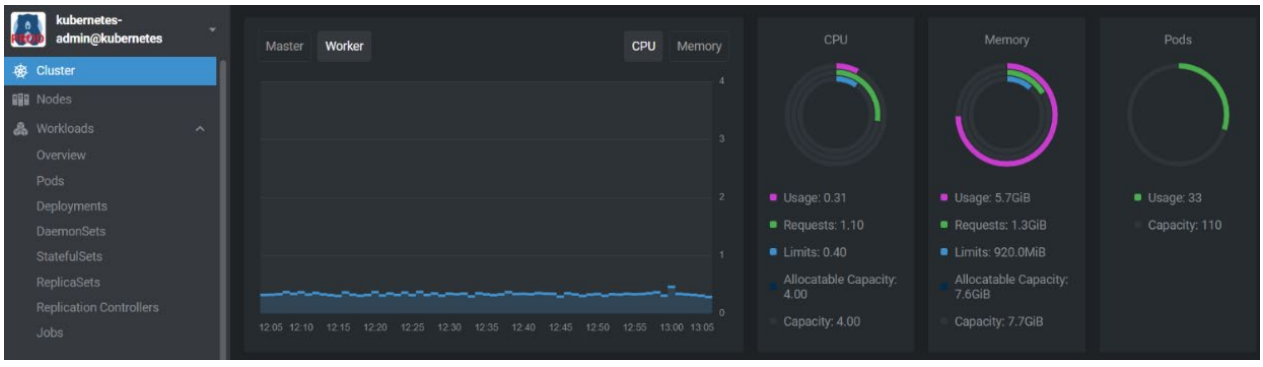

Figure 13: Cluster network interfaces preview.

[20] Prometheus, from metrics to insight (*https://prometheus.io/docs/introduction/overview*)
[21] OpenLens, Lens Desktop Core "OpenLens" (*https://github.com/lensapp/lens*)

5 Data migration

The migration of data from the preliminary VM-based environments (components running under *docker-compose*) to the final Kubernetes-based environments, requires a controlled procedure to ensure the consistency of the data not just for the component itself, but also when several depend on each other. This process was carried out in a coordinated manner, with the stop of the running platform at source to ensure a proper snapshot and avoid any corruption or misalignment in the target components, once again running.

Equally important is to formalize the process that verifies and validates that the migration was performed as expected. The migration process also refers to the change in technologies and methodologies that will be used in the destination environment, in terms of data persistence with the use of the Kubernetes *PersistentVolumeClaims* abstraction, and the backup strategies to follow up for now on.

The following *Table 4* summarizes the technologies used in the components for persistence of its data, and dependencies identified between them that should be maintained in the migration.

Partner	Component	Persistence technology	Dependency
ATOS	Clinical Data Repository (CDR)	*MariaDB* database, migration based on complete DB dump and restore, maintaining all the original status, including all keys. Target PVC: `smartbear-cdr-database-pvc`	The list of patients should match the stored in the *UserManager*.
STS	Security Component Keycloak	*PostgreSQL* database, Target PVC: `data-smartbear-sc-keycloakdb-postgresql-0`	
	Security Component Usermanager	*PostgreSQL* database, Target PVC: `data-smartbear-sc-usermanagerdb-postgresql-0`	Should match with the patients in CDR
ICCS	Decision Support System (DSS)	*MariaDB* database, migration based on complete DB dump and restore. Target PVC: `ssmartbear-dss-database-pvc`	
ATC	SB Notifications	*MongoDB* database, Target PVC: `smartbear-sb-notifications-database-pvc`	
	Android App Backend	*MongoDB* database, Target PVC: `smartbear-android-app-backend-database-pvc`	
UMIL	Big Data Analytics Engine (BDA)	*PostgreSQL (API-BDA, airflow, hive metastore), Hadoop* Target PVC: `smartbear-bda-api-database-postgresql-0` Target PVC: `data-smartbear-bda-airflow-database-postgresql-0` Target PVC: `data-smartbear-bda-metastore-database-postgresql-0` Target PVC: `smartbear-bda-hadoop-datanode-statefulset-0 / 1 / 2`	

Table 4: Storage technologies and dependencies.

5.1 Security Component

The three subcomponents that conforms the Security component, are treated individually in terms of configuration loading or migration.

5.1.1 KeyCloak identity server

Upon first run, the following configuration steps are required (for a from-scratch installation, without loading previously exported configuration).

5.1.1.1 Create a new realm

Create the SMART BEAR realm by clicking the dropdown menu on the sidebar and selecting "Create Realm". In realm name enter "`smartbear`" and click "Create" (*Figure 14*).

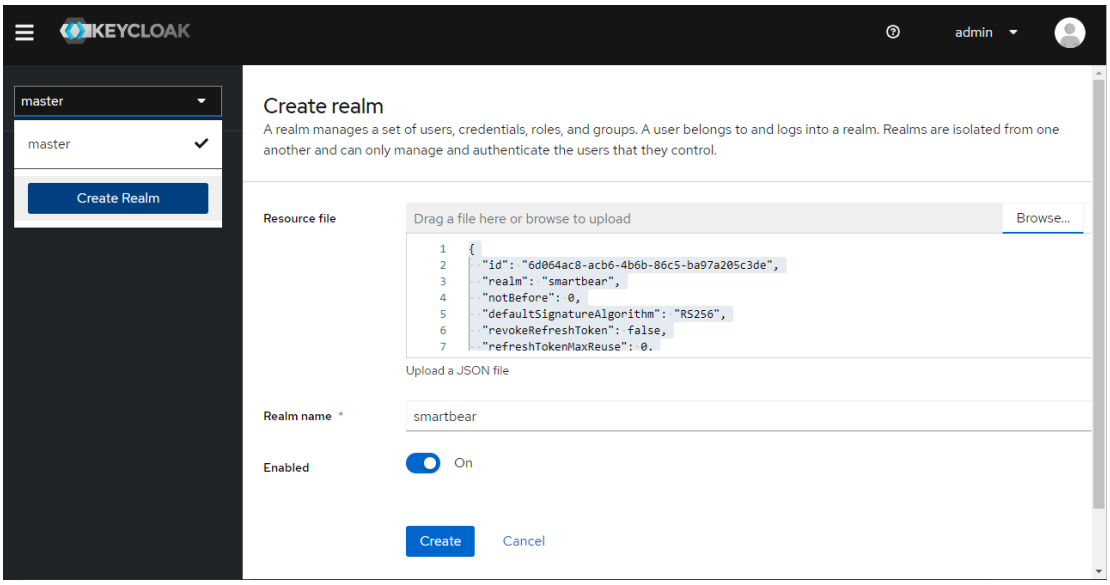

Figure 14: Keycloak realm creation from previously exported one.

5.1.1.2 Add realm roles

Click on the "Realm Roles" in the sidebar, and add the following roles[22]:

- `Auditor`: accesses the SMART BEAR platform in order to audit its behaviour, especially for monitoring the data processing procedures in terms of compliance with guidelines and regulations.
- `Caregiver`: provides care for a person who needs extra help. This could mean a family caregiver, a respite caregiver, a home caregiver, or a primary caregiver, to name but a few. In the context of elderly care, this job title typically refers to a private duty home caregiver or senior caregiver.
- `CCM`: a health care professional that works as a primary caregiver of a patient in a hospital, skilled nursing facility, clinic, or patient's home. A clinician diagnoses and treats patients. The

[22] Also defined in D2.1 *SMART BEAR Requirements*, chapter 5 *User and Platform Requirements* (page 132)

Case Manager uses the SMART BEAR platform in order to increase the level of evidence based practices and care plans.

- `DataScientist`: a professional responsible for collecting, analysing and interpreting the overall amount of data collected by the SMART BEAR Cloud. Data science plays an important role for the effective exploitation of the data collected by the SMART BEAR platform.
- `PolicyMaker`: exploits the statistical report and analytics resulting from the SMART BEAR Cloud to create policies and plans, especially for the good of a territory or community.
- `SysAdmin`: is responsible for the setup and reliable operation of the SMART BEAR platform. It should be able to ensure that the uptime, performance, resources, and security of the computers they manage meet the needs of the other users. The system administrator may require access to the system components and to the registers accounting for the behaviour of the SMART BEAR platform.
- `Tech`: a technician from the clinical site who provides assistance to the patients for the wearables and IoT devices.

5.1.1.3 Add clients

Go to "Clients" in the sidebar, click on "Create client" and add the following clients:

- Dashboard:
  - Name: `SMART-BEAR Dashboard`
  - Click on "Next". Disable "Direct Access Grants" and keep only "Standard flow" enabled.
  - Go back to the client list, click on the newly created client and fill in the following:
  - Valid Redirect URIs: http://localhost:8080/, /
  - Valid post logout redirect URIs: +
  - Web origins: +
  - Save.
- UserManager
  - Client ID: `usermanager`
  - Click on "Next". Enable "Client authentication". Disable "Standard flow" and "Direct Access Grants". Enable "Service account roles".
  - Save.
  - Go to "Users" in the sidebar. Select "service-account-usermanager". Go to the Role Mapping tab. Click on "Assign Role" and add "realm-management view-users".
  - Save.
- HoloBalance
  - Client ID: `holobalance`
  - Click on "Next". Disable "Direct Access Grants" and "Direct Access Grants". Enable "Service account roles".
  - Save.
- Mobile
  - Client ID: `mobile`
  - Click on "Next". Disable "Direct Access Grants" and "Direct Access Grants". Enable "Service account roles".
  - Save.
- Smart4Health

  - Client ID: `smart4health`
  - Click on "Next". Disable "Direct Access Grants" and "Direct Access Grants". Enable "Service account roles".
  - Save.
- Smart4HealthBox
  - Client ID: `smart4healthbox`
  - Click on "Next". Disable "Direct Access Grants" and "Direct Access Grants". Enable "Service account roles".
  - Save.

5.1.1.4 SmartBear theme

Go to "Realm Settings" in the sidebar, click on the "Themes" tab. From the dropdown, select "smartbear" for the "Login theme" and "Email theme".

5.1.1.5 Configure email

Go to "Realm Settings". Click on the "Email" tab. In the "From" field, enter "no-reply@smart-bear.eu".

Under Connection & Authentication, enter the following:

- Host: `mail.smart-bear.eu`
- Port: `587`
- Authentication: `Enabled`
- Username: `no-reply@smart-bear.eu`
- Password: `***TBD***`

To test the connection, you need to have added an e-mail to the admin account. To do this, go back to the "master" realm and edit the "admin" user.

5.1.1.6 Configure Token Claims

- Go to "Client scopes" in the sidebar.
- Select "Create client scope". Enter "organisation" in the name and click on "Save".
- Select the new client scope, go to the "Mappers" tab. Click "Configure new mapper".
- Select "User attribute". Enter "organisation" in the "name", "user attribute", and "token claim name" fields. Click "Save".

5.1.2 KrakenD API gateway

We use KrakenD's flexible configuration, to configure the gateway. The main configuration file is `krakend.tmpl` which is a Go template. The config directory contains additional configuration files used by the template, including:

- The templates directory which contains templates for the endpoints, with one template created for each component.
- The partials directory which contains the configuration for Keycloak, directly injected into the endpoint templates.

- The settings directory which contains variables used to configure the component addresses in the form of <host>:<port>.

KrakenD plugins are written in Go and loaded at runtime. In this repository, plugins are located in the plugins directory and compiled according to the *Dockerfile*. The following plugins are included:

- `bda-pseudonymization`: performs pseudonymization on request and response data of the BDA component.
- `sb-data-repo-pseudonymization`: performs pseudonymization on request and response data of the SB Data Repo component. This is the largest plugin, as it handles complex FHIR resources, including recursive structures.
- `mobile-notification`: sends an e-mail notification to a participant's clinical case manager.
- `http-client`: operates just before the request is sent to the backend and just after the response is received. It is used to store the X-Synergy header added to the request by a previous plugin, and to add it to the response to be used by a subsequent response plugin. Otherwise, the header would be lost.

5.1.3 UserManager participants management

User management is handled by Keycloak. APIs for the basic operations of user management (add/edit/delete users for the `smartbear` realm) have been created and can be used in the future to create a UI within the dashboard for the user management.

To manage users from the Keycloak, one must have setup a realm and select the realm from the main menu. Then the following actions can be performed.

5.1.3.1 Add users

- Go to "Users" in the sidebar. Click on "Add user".
- Enter user details (username, email and name)
- Click on "Create".
- Go to the newly created user.
- Click on "Attributes" and enter "organisation" in the key, and the abbreviation of the user's organisation in the value. Click "Save".
- Go to "Role Mapping". Click "Assign Role" and enter the user's role (make sure that “filter by realm roles is selected).
- Go to "Credentials". Select "Credential Reset" and select "Reset password". An email will be sent to the user with instructions on how to set their password.

5.1.3.2 Edit users

- Go to "Users" in the sidebar. Click on the user.
- Click on "Attributes" and update "organisation" in the key, and the abbreviation of the user's organisation in the value. Click "Save".
- Go to "Role Mapping". Click "Assign Role" and update the user's role (make sure that “filter by realm roles is selected).

- Go to "Credentials". Select "Credential Reset" and select "Reset password". Optionally, if you click the ‘send email’ an email will be sent to the user with instructions on how to set their password.

5.1.3.3 Disable account

- Go to "Users" in the sidebar. Click on the user.
- Go to "Credentials". Select "Credential Reset" and select "Delete account".

5.2 Clinical Data Repository (CDR)

In this component, the data storage requirements are covered by a *MariaDB* relational database engine, which derived from *MySQL* (but licensed under the GPL) and maintains MySQL commands compatibility.

The first installation of HAPI-FHIR for the SMART BEAR project almost three years ago was based on version 5.0.0[23], supported by a *MariaDB* 10.5 (and *mariadb-jdbc-client 2.7.2*) deployed as container based on its official *Docker* image as described in *Table 5*. When we migrated to the new K8s-based infrastructure, we took advantage to update to a newer version that provides improved functionalities and bug fixing, namely the HAPI-FHIR 6.6.0[24]. Also, the MariaDB version was upgraded to a newer 10.6.4[25] version, aligning it with other components that are using also MariaDB, like the *Decision Support System* (DSS) so that only one Docker image is required, and this way minimize as much as possible the number of unnecessary third-party dependencies, and also its configuration unified.

	VM-based Environment *(docker-compose)*	Kubernetes cluster
MariaDB	`mariadb:10.5`	`mariadb:10.6.4`

CDR	`HAPI-FHIR 5.0.0`
	`mariadb-jdbc-java-client 2.7.2`
	`Hibernate-5.4.14`
		`HAPI-FHIR 6.6.0`
		`mariadb-jdbc-java-client 3.1.4`
		`Hibernate-5.6.12`

Table 5: CDR database migration versions reference.

However, this migration involves certain challenges. The HL7 FHIR standard implementation provided by the HAPI-FHIR project, alongside the module that provides the *Java Persistence API (*JPA) functionalities for the abstraction of the data layer in the Java implementation, is subject to continuous improvement. Therefore, we had to implement adaptations in the structure of the database schema that supports the application, and it needs to be upgraded. For this process there is a migration utility, but which in turn only supports an old MariaDB Hibernate dialect (v10.1), since the project is not supporting MySQL based Data Base (DB) anymore. In the practice, the migration had to maintain in the *SpringBoot* implementation configuration the old `MariaDBDialect` dialect but not the newer `MariaDB106Dialect` that is also due to the mayor changes in internal DB specifications, like the new generation of `SEQUENCES` that did not exist in the DB used in the previous PoP version.

5.2.1 Database dump, load and upgrade

The FHIR implementation of the CDR (HAPI-FHIR) manages, wrapping via a JPA[26] layer, the DB that provides the support for the data storage. Even though the CDR also provides data import/export functionality tools, since this is a full migration process, the data movement was better performed at the lowest level possible, with a “full dump” and restoring the whole DB content in the target installation.

[23] HAPI-FHIR v5.5.0 (*https://github.com/hapifhir/hapi-fhir/tree/v5.5.0*)
[24] HAPI FHIR v6.0.0 release (*https://github.com/hapifhir/hapi-fhir/releases/tag/v6.6.0*)
[25] MariaDB 10.6.4 (*https://mariadb.com/kb/en/mariadb-1064-release-notes*)
[26] Java Persistence API (*https://es.wikipedia.org/wiki/Java_Persistence_API*)

MariaDB (and its precedent MySQL) provides a set of generic tools (`mysqldump` or `mariadb-dump` command line[27]) for dumping and restoring data, usually for backup purposes, that can be equally used in this migration process. To migrate the data, the component should be switched off, so the database use is paused to prevent any other parallel operation that causes an inconsistent state in the target deployment, and the following command could be executed to generate a complete database dump file. Note that some option params ensures the later proper creation of the DB, like the selection of the character set, or complete drop of existing creation in case it already existed.

```
                                                    // going into the container if port is not exposed
$ docker exec -it mariadb bash
                                    // execute the command to dump the DB, locking tables is important if still app is up
$ root@mariadb:/# mariadb-dump --user="root" --password="******" --add-drop-database --lock-all-tables --
default-character-set=utf8mb4 --replace --databases smartbear_hapi > /tmp/smartbear_hapi-dump.sql
&& /tmp/gzip smartbear_hapi-dump.sql                    // optional to compress and transfer the file faster
$ root@62fbafcec47f:/# exit
                                                                    // copy file from the container to the host
$ docker cp mariadb:/tmp/smartbear_hapi-dump.sql.gz /home/atos/smartbear/sb-data-repository/dump
```

Note that the parameters should match the original setup and other notes:

- The database name is specified in the environmental variable `MYSQL_DATABASE` (`smartbear_hapi`, by default) in the `smartbear-cdr-database-config` Kubernetes *ConfigMap*.
- The database root password is specified in the environmental variable `SPRING_DATASOURCE_PASSWORD` in the `smartbear-cdr-database-secret` Kubernetes *Secret*.
- Notice that the export stablishes the name of the database, and force the command `USE` later at the beginning of the generated script.
- Since database users/passwords/privileges are kept in the `mysql` database, we keep in mind that such creation will be necessary in later steps, not having used `--all-databases`.
- The original use of the database created tables that newer driver version could try to generate as `sequence` objects if using by mistake newer drivers in later upgrade processes.
- If an abnormal size is suspected, the use of FHIR objects of type `Binary` can be checked and cleaned up in advance, since they are used in an auxiliary way for the generation of exports and could have not been cleaned automatically.

The previous dumped content should reach the target DB engine installation, running in a pod within the new Kubernetes cluster. The file could be big enough to be a process to last a little time, both in transfer and in its processing, and there are several options for the process, depending on the available commands, network speed and further wanted monitoring needs.

The HAPI-FHIR project provides a mean to track the changes of the migration, by creating a table named `fly_hfj_migration` and through a specific *hapi-fhir-sql-migrate*[28] module that could be executed via the 'command line tool' (`hapi-fhir-cli-app`), which executes for the version compiled, the necessary changes to adapt the database schema for the final release to be deployed. Moreover, the process records in such table, each change description and in which version was performed and effectively executed the change.

[27] msqldump, A Database Backup Program (*https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html*)
[28] hapi-fhir-sql-migrate. (*https://github.com/hapifhir/hapi-fhir/tree/master/hapi-fhir-sql-migrate*)

The following steps are a direct process based on low-level commands, although there would be the possibility of using graphical interfaces and remote connections for the transfer of files and execution of the operations (e.g.: *sftp* client like FileZilla[29], *PortForwarding* and HeidiSQL, etc).

1. The data containing the dump could be transferred directly from the source machine to the Kubernetes cluster master node, via secure copy command `scp`.

```
                                                    // Copy from the source VM to the Kubernetes master node
$ atos@smartbear-vm:/home/atos/smartbear/atos/sb-data-repository$ scp smartbear_hapi-dump.sql.gz
atos@147.102.33.194:/home/atos/
The authenticity of host '147.102.33.194 (147.102.33.194)' can't be established.
ECDSA key fingerprint is SHA256:CH+PieEGvfbjSDAHc9X8Fzo8rfQ7SM4x9Sv68LQA+Gg.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '147.102.33.194' (ECDSA) to the list of known hosts.
atos@147.102.33.194's password: *****
smartbear_hapi-dump.sql.gz           100%    0
```

2. Once having the file by hand in a node in the cluster, is in turn copied to the pod where the database engine container is running and, accessing to its own console[30], execute the proper loading command:

```
                                                    // kubectl cp command to copy a file from host to a container
$ kubectl -n smartbear cp smartbear_hapi-dump.sql smartbear-cdr-database-deployment-
????????:/tmp/smartbear_hapi-dump.sql
$ kubectl exec --stdin --tty smartbear-cdr-database-deployment-???????? -n smartbear -- /bin/bash
$ nobody@smartbear-cdr-database-deployment-????????:/$ls -la /tmp
-rw-r--r-- 1 nobody nogroup 987654321 Jun 15 15:00 smartbear_hapi-dump.sql.gz
```

3. After transfer the file and having it available in the target pod running the MariaDB engine, to recover/import it the steps are:
    a. Stop the CDR instance in case already running.

```
                                                    // scale to 0 the CDR deployment to stop the HAPI-FHIR instance
$ kubectl scale deploy smartbear-cdr-deployment -n smartbear --replicas=0
```

    b. Import the complete database from the backup using the `root` user.

```
                                                    // MariaDB command to import DB
$ mariadb --user="root" --password="*****" < "./smartbear_hapi-dump.sql"

                                                    // Note the script start creating the DB
/*!40000 DROP DATABASE IF EXISTS `smartbear_hapi`*/;
CREATE DATABASE /*!32312 IF NOT EXISTS*/ `smartbear_hapi` /*!40100 DEFAULT CHARACTER SET
utf8mb4 */;
USE `smartbear_hapi`;
```

    At the time of the migration, the database had an approximate size of 10GB, which lengthened the loading process so it could be possible to verify its progress in parallel checking the size increase.

```
                                                    // Monitor the loading by checking increasing size
$ mariadb --user=root --password=******
```

[29] FileZilla, Free FTP solution Client for FTP, FTPS and SFTP distributed under the GNU License (*https://filezilla-project.org*)
[30] Kubernetes, Get a Shell to a Running Container (*https://kubernetes.io/docs/tasks/debug/debug-application/get-shell-running-containe*)

```
MariaDB [(none)]> SELECT table_schema 'smartbear_hapi', SUM(data_length + index_length)
'Size in Bytes', ROUND(SUM(data_length + index_length) / 1024 / 1024, 2) 'Size in MiB' FROM
information_schema.tables WHERE table_schema = 'smartbear_hapi' GROUP BY table_schema;
```

c. Create a new `smartbear_hapi` user and grant privileges, with the pair *user-password* defined in the following variables to match the ones the CDR will use:
   - variable `SPRING_DATASOURCE_USERNAME` (*smartbear_hapi*, by default) in the `smartbear-cdr-database-config` Kubernetes *ConfigMap*.
   - variable `SPRING_DATASOURCE_PASSWORD` in the `smartbear-cdr-database-secret` Kubernetes *Secret*.

```
                                                    // MariaDB users and privileges grants creation

$ mariadb --user=root --password=******

MariaDB [(none)]> CREATE USER IF NOT EXISTS 'smartbear_hapi'@'localhost' IDENTIFIED BY '**';
MariaDB [(none)]> GRANT USAGE ON smartbear_hapi.* TO 'smartbear_hapi'@'localhost';
MariaDB [(none)]> GRANT EXECUTE, SELECT, SHOW VIEW, ALTER, ALTER ROUTINE, CREATE, CREATE
ROUTINE, CREATE-TEMPORARY TABLES, CREATE VIEW, DELETE, DROP, EVENT, INDEX, INSERT,
REFERENCES, TRIGGER, UPDATE, LOCK TABLES ON `smartbear\_hapi`.* TO
'smartbear_hapi'@'localhost' WITH GRANT OPTION;
MariaDB [(none)]> FLUSH PRIVILEGES;

MariaDB [(none)]> SHOW GRANTS FOR 'resample_hapi'@'localhost';
```

A file containing the previous *sql* commands could be created and copied for direct execution or typed one by one in the console.

```
                                                    // MariaDB users and privileges grants creation

$ kubectl -n smartbear cp UserPrivilegesCreation.sql smartbear-cdr-database-deployment-
7d754767c7-bkxs7:/tmp/UserPrivilegesCreation.sql
```

4. Regarding to the potential changes between MariaDB versions, the `mariadb-upgrade`[31] tool has been used to check if any update applies to the existing tables, without any remarkable issue. The official Docker image provides the `MARIADB_AUTO_UPGRADE` and `MARIADB_DISABLE_UPGRADE_BACKUP` environmental variables[32], to have the entry point check whether mysql_upgrade/mariadb-upgrade needs to run, so they are set up in the deployment descriptor files (*Figure 15*).

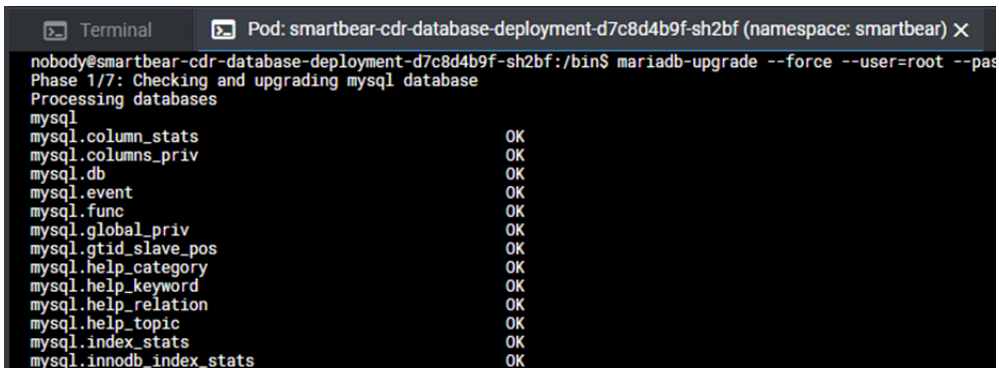

Figure 15: MariaDB upgrade check result.

[31] MariaDB upgrade. (*https://mariadb.com/kb/en/mariadb-upgrade*)
[32] MariaDB Docker image environmental variables. (*https://mariadb.com/kb/en/mariadb-server-docker-official-image-environment-variables*)

5. At this point we are also ready to apply the potential schema upgrades for the version update of the HAPI-FHIR distribution, with the help of the provided tool. A minor adjustment needs to be done in the DB since depending on the target default MariaDB setup and the following exception could occur when MariaDB is not accepting by default double quotes in the names of tables and columns in the migration commands.

```
// Exception when using double quotes in tables/column names in the hapi-fhir-cli-app migrate-database tool

org.springframework.jdbc.BadSqlGrammarException: StatementCallback; bad SQL grammar [CREATE TABLE
"FLY_HFJ_MIGRATION" ("installed_rank" INTEGER NOT NULL,"version" VARCHAR(50),"description"
VARCHAR(200) NOT NULL,"type" VARCHAR(20) NOT NULL,"script" VARCHAR(1000) NOT NULL,"checksum"
INTEGER,"installed_by" VARCHAR(100) NOT NULL,"installed_on" DATE NOT NULL,"execution_time" INTEGER
NOT NULL,"success" bit NOT NULL)]; nested exception is java.sql.SQLSyntaxErrorException: (conn=60)
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server
version for the right syntax to use near '"FLY_HFJ_MIGRATION" ("installed_rank" INTEGER NOT
NULL,"version" VARCHAR(50),...' at line 1
```

MariaDB 10.6, supports different modes and is configured based on the system variables `[GLOBAL].SQL_MODE`[33], being a string with different options[34] separated by commas and without spaces. Below is shown the default content, and how `ANSI_QUOTES` is included to support the format the migration tool uses.

```
                                                                    // Set the SQL_MODE for compatibility purpose
#mariadb> SELECT @@SQL_MODE, @@GLOBAL.SQL_MODE;
STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
#mariadb> SET GLOBAL
SQL_MODE='STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION,ANSI_QUOTES';
#mariadb> SET
SQL_MODE='STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION,ANSI_QUOTES';
```

After the adjustment, we need an environment able to run the tool, which is based on Java, preferably the closest physically to the DB location for quicker process. Although one approach is to have installed Java in the master node so we can execute directly from there, a utility and temporally pod can also be created, and this process automatized.

```
        // Copy hapi-fhir-cli.jar and DB dump from the admin client to the current Kubernetes HAPI-FHIR POD
$ kubectl -n smartbear cp hapi-fhir-cli.jar smartbear-cdr-deployment-????:/tmp/hapi-fhir-cli.jar -n
smartbear
$ kubectl -n smartbear cp smartbear_hapi-dump.sql smartbear-cdr-database-deployment-????:/tmp/
smartbear_hapi-dump.sql
                                                                    // Get a console into the pod

$ kubectl -n smartbear exec --stdin --tty smartbear-cdr-deployment-???????? -n smartbear --
/bin/bash
                                  // And finally execute the tool with the proper parameters for migrate DB schema
$ /hapi-fhir-cli/ >java -jar hapi-fhir-cli.jar help migrate-database

Usage:
  hapi-fhir-cli migrate-database [options]
This command migrates a HAPI FHIR JPA database to the current version
Options:
  -d,--driver <Driver>          The database driver to use (Options are H2_EMBEDDED,DERBY_EMBEDDED,
                                MARIADB_10_1, MYSQL_5_7, POSTGRES_9_4, ORACLE_12C, MSSQL_2012,
                                COCKROACHDB_21_1)
     --debug                    Enable debug mode
```

[33] MariaDB SQL_MODE configuration (*https://mariadb.com/kb/en/sql-mode/#ansi_quotes*)
[34] MariaDB SQL_MODES (*https://mariadb.com/kb/en/sql-mode/#setting-sql_modes*)

```
   -n,--username <Username>       The JDBC database username
      --no-column-shrink          If this flag is set, the system will not attempt to reduce the
                                  length of columns. This is useful in environments with a lot of
                                  existing data, where shrinking a column can take a very long time.
   -p,--password <Password>       The JDBC database password
   -r,--dry-run                   Log the SQL statements that would be executed but to not actually
                                  make any changes
      --skip-versions <Versions>  A comma separated list of schema versions to skip
   -u,--url <URL>                 The JDBC database URL
   -x,--flags <Flags>             A comma-separated list of any specific migration flags (these flags
                                  are version specific, see migrator documentation for details)


$ java -jar hapi-fhir-cli.jar migrate-database -d MARIADB_10_1 -u
"jdbc:mariadb://localhost:3306/smartbear_hapi?useSSL=false" -n "smartbear_hapi" -p "******" -r
```

The migration takes long time especially when reach the changes on version 5.3.0 due to changes in columns datatypes in big tables and once finished the migration, the table `fly_hfj_migration` will contain the scripts performed and its success status (*Figure 16*).

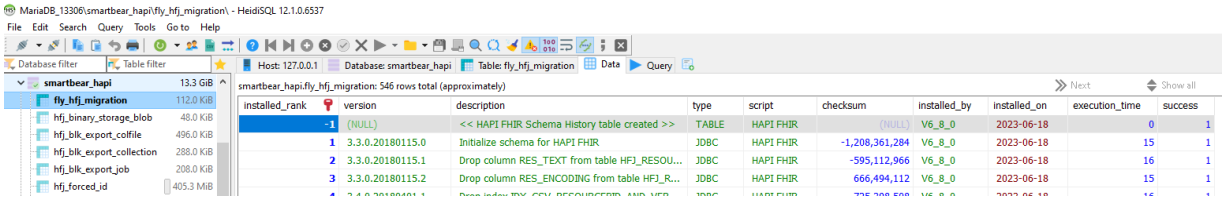

Figure 16: Status of fly_hfj_migration table after migration process.

6. At this point we can start the CDR again, undoing the scaling to zero executed in step 3.a, with the value back to 1, and the UI should show the resources properly both at the level of counters and the content of specific resources (*Figure 17*), which in turn can be validated as proposed in section *5.7.2 Clinical Data Repository (CDR)*.

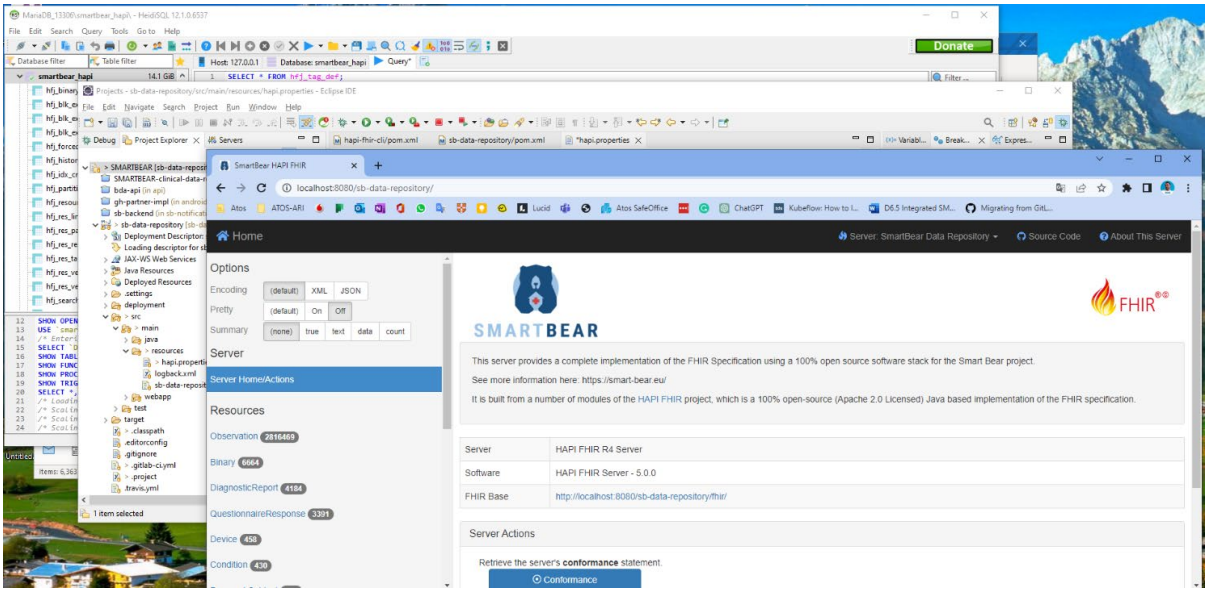

Figure 17: CDR HAPI-FHIR started with the migrated resources data.

5.2.2 Load of ATC ValuSet into the CDR

The empty CDR requires the load of a pre-defined *ValueSet* from the Anatomical Therapeutic Chemical Classification System (ATC)[35], which is a drug classification system that classifies the active ingredients of drugs. This process is performed just once by the loading of a *ValueSet* that, in turn, points to another public *ValueSet*, where the complete list of codes is maintained.

The project HL7 FHIR Implementation Guide[36] defines the value set as an expansion from the maintained by the World Health Organization (WHO) based on the version published in January 2020.

A copy is maintained in the CDR code repository[37], and the process should be performed just once when the DB is initialized from scratch, such in the case of new develop/staging environments (for the PoP was already in the past) and can be done directly by (a) executing the following commands from the current CDR pod deployed.

```
                                                    // load of the Value set directly from the pod console
$ kubectl cp ValueSet-ATCVS.json smartbear-cdr-deployment-????????:/tmp/ValueSet-ATCVS.json -n smartbear
$ kubectl exec --stdin --tty smartbear-cdr-deployment-???????? -n smartbear -- /bin/sh
$ nobody@smartbear-cdr-deployment-????????:/$ cd /tmp
$ nobody@smartbear-cdr-deployment-????????:/$ curl -X POST http://localhost:8080/clinical-data-
repository/fhir/ValueSet -H 'Content-type:application/json' -d @ValueSet-ATCVS.json
```

or (b), via the *PortForwarding* functionality, using an available free localhost port (58080 in the following example).

```
                                    // kubectl port-forward service from Kubernetes cluster to the 58080 port at localhost
$ kubectl port-forward service/smartbear-cdr-svc 58080:8080 -n smartbear
Forwarding from 127.0.0.1:58080 -> 8080
Forwarding from [::1]:58080 -> 8080
Handling connection for 58080
                                                    // from another console, while the port-forward is in place
$ curl -X POST http://localhost:58080/clinical-data-repository/fhir/ValueSet -H 'Content-
type:application/json' -d @ValueSet-ATCVS.json
```

[35] ATC, Anatomical Therapeutic Chemical Classification System (*http://www.whocc.no/atc*)
[36] SmartBear Implementation Guide ATC ValueSet (*https://smart-bear.eu/fhir/ValueSet-ATCVS.html*)
[37] ATC ValueSet (*/clinical-data-repository/src/main/resources/ValueSet-ATCVS.json*)

5.3 Decision Support System (DSS)

The decision Support System (DSS) relies on a relational DB also known as the “non-FHIR database”, which needs to be migrated, with the required following steps. The non-FHIR database has been developed using MariaDB, a community-developed relational database management system, free and open-source under the GNU General Public License.

- The only required tool to export the data is `mysqldump`, and there is no need to create the database tables in advance since the tables and the seed data are created on the first run of the application.
- The tables that contain seed data are:
  - `Interventions`: contains the intervention codes for the notifications
  - `MeasurementFilter`: contains the allowed data to be stored in measurements table

When relocating the database, the easiest way to migrate the database and keep it transparent to the non-FHIR backend and the DSS, is by creating a *mysql dump* and then simply running the database import in the new database, before starting the non-FHIR backend app. This should be enough unless the target version has evolved in some way where there are some compatibility issues and some transformation is necessary, which has not happened between MySQL versions used at the moment.

You could use the following snippet to extract the database dump:

// mysqldump command to backup DB

```
$ mysqldump --protocol=tcp --default-character-set=utf8 --single-transaction=TRUE --routines --events --
skip-add-locks --skip-lock-tables --skip-add-drop-table --skip-create-options --result-file="LOCATION-OF-
THE-FILE.sql" --all-databases
```

After the above step, the app should start as normal.

Any new database migrations will be handled automatically via the DSS codebase.

5.4 SB Notifications

In this component the storage requirements are covered by a *MongoDB* non-relational document database with server version 6.0.4[38].

Studio 3T for MongoDB[39] is a very useful tool for the migration of the MongoDB database required for this component.

Using the *Studio 3T* desktop app, a connection can be created with the MongoDB database from where the data will be migrated (Figure 18). The MongoDB database can be exported at a chosen local directory in JSON format as shown in the following screenshots. After exporting all the collections, a new connection is created with the destination database. There, the exported collections are imported successfully. Here are the steps in more detail:

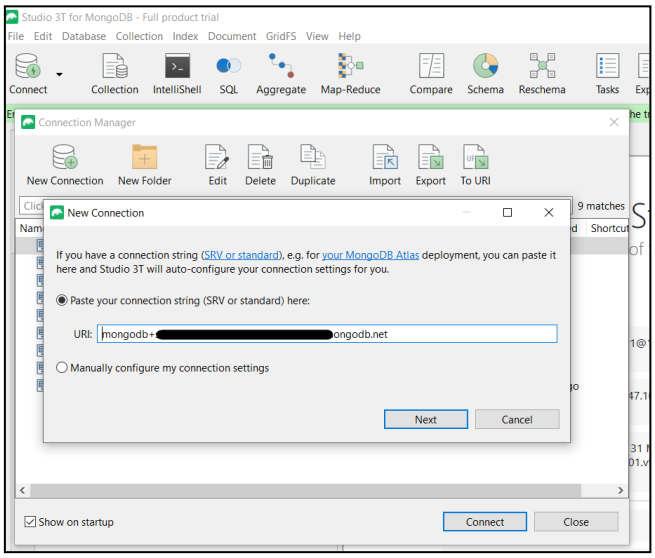

Figure 18: MongoDB create new Connection.

1. Create a connection with the MongoDB from where the data will be exported. After that, select the option to "Export…" the database, as shown in the following *Figure 19*.

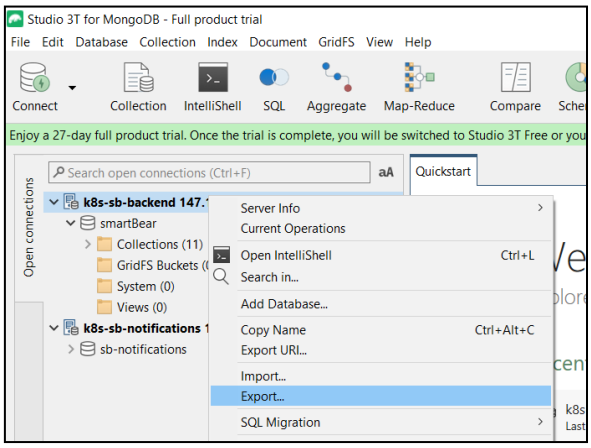

Figure 19: MongoDB select to export all collections from the database.

[38] Release Notes for MongoDB 6.0 (*https://www.mongodb.com/docs/manual/release-notes/6.0/#6.0.4---jan-26--2023*)
[39] Studio 3T for MongoDB, the professional GUI for MongoDB (*https://studio3t.com*)

2. Then choose the format for the export file, where the default JSON is enough (*Figure 20*).

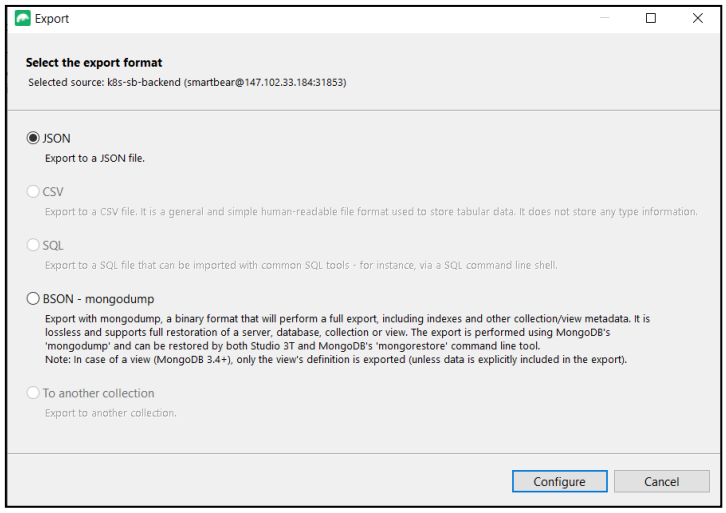

Figure 20: MongoDB select JSON as export format.

3. And select the file target and other options the export (*Figure 21*).

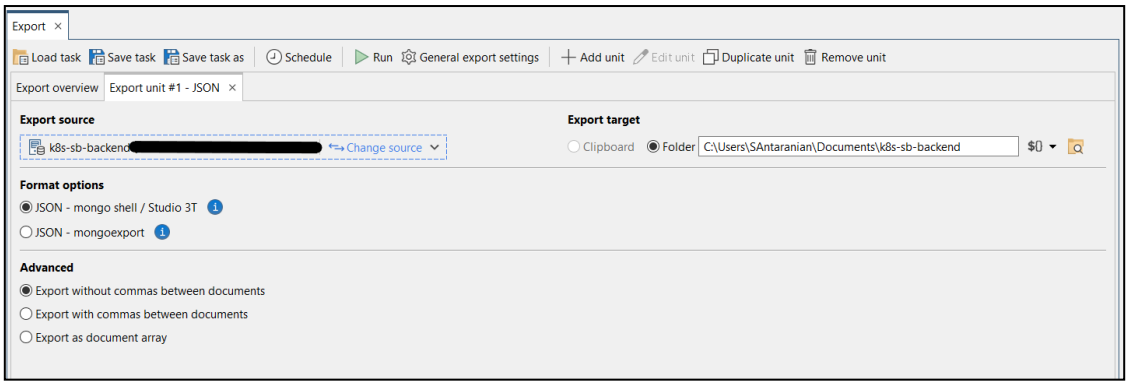

Figure 21: MongoDB configure export.

4. For the import phase, the parallel option could be selected (*Figure 22*).

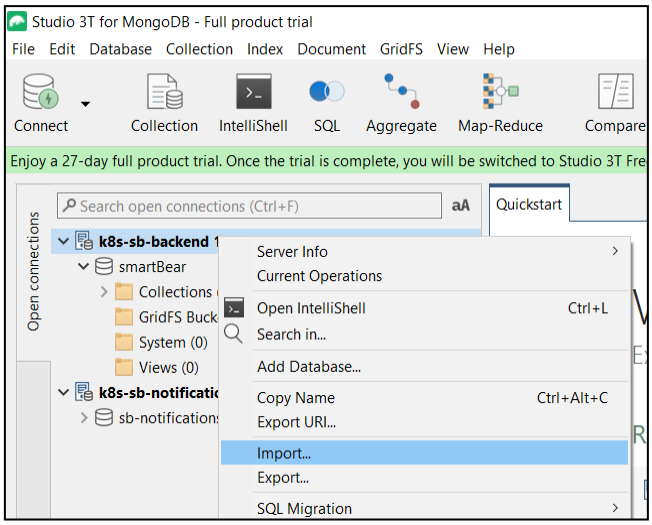

Figure 22: MongoDB select to import collections.

5. And again, the type of format should be specified (*Figure 23*).

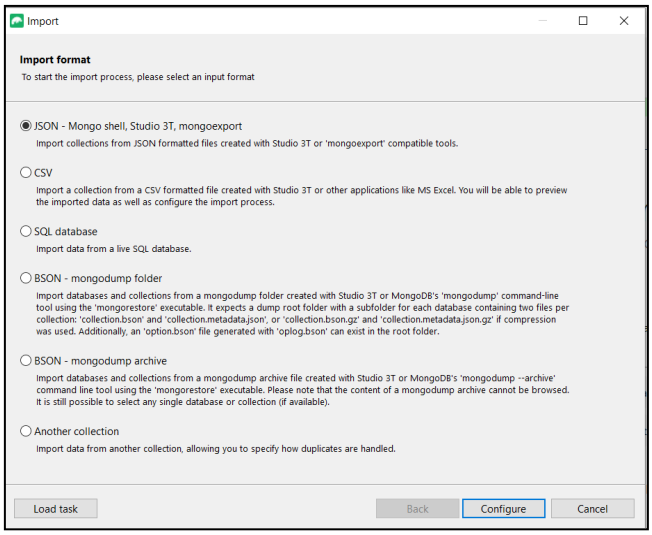

Figure 23: MongoDB configure Import.

In order to temporally access to the DB engine in the Kubernetes cluster, not exposed to the outside, the *PortForwarding*[40], functionality could be used, opening a bridge between the KubernetesAPI client and the target service, for direct connexion and data transfer. Once the migration or monitorization is performed, the specific *PortForwarding* should be closed and removed.

5.5 Android App Backend

This component, in a similar way as the previous *SB Notifications* component, uses a *MongoDB* non-relational document database in its version 6.0.4.
Studio 3T for MongoDB can again be used for the migration of the database following the same exact steps. While the component is deactivated for the migration process, internal and external components interacting with this component have the appropriate mechanisms in place in order to retry communicating with the component, when it is not responding.

[40] Kubernetes, Use Port Forwarding to Access Applications in a Cluster (*https://kubernetes.io/docs/tasks/access-application-cluster/port-forward-access-application-cluster*)

5.6 Big Data Analytics Engine (BDA)

This component does not need a data migration process since the data could be regenerated from scratch with the FHIR and non-FHIR data found in the new production environment, so the periodic executions of internal processes will be able to bring the database to the desired and expected final state.

The BDA is made up of several components:

- HDFS
- BDA-API
- Zeppelin
- HDFS
- Trino
- Airflow
- Hive Metastore

Only some components rely on DDs, like Hive Metastore, Airflow and the BDA-API but in all of the cases we choose to not migrate those old databases because the mechanism already in place does an extraction of the data from the FHIR and non-FHIR repository. Those mechanism restores the previous state without requiring dumping and reload the old ones.

Basically, the mechanism dump the data from FHIR and non-FHIR repo and move the data to the HDFS as Parquet files, the job runs continuously, so after the restoration of the other components the data will be extracted, and the analytics could run like in the previous environment.

5.7 Data migration validation

As a summary, the following components are the ones that needs to migrate its data from the previous VM-based production environment used in the Pilot of Pilots, to the new Kubernetes production cluster. The validation process is detailed below, in case its criticality and complexity truly require it, and is omitted for trivial cases. *Table 6* summarises the needs.

Partner	Component	
STS	`Security Component KeyCloak`	After the loading of the previously exported configuration, check the existence of the configuration files.
	`Security Component UserManager`	Number of users maintained in the tables comparation. Users 108. Caregivers: 3, CCM: 303, Contact phones: 142, Device metrics: 67740, Devices: 76, GDPR request: 0, GRPR request files: 0, Notification: 316, identities.
ATOS	`Clinical Data Repository (CDR)`	Process described in *5.7.2 Clinical Data Repository (CDR).* HAPI-FHIR can load properly. UI-via port forwarding could be visualized for counters and basic querying. Match of counters for the resources used
ICCS	`Decision Support System (DSS)`	Entries comparation.
ATC	`SB Notifications`	Just comparing sizes and a random sample check.

	Android App Backend	Just comparing sizes and a random sample check.
UMIL	Big Data Analytics Engine (BDA)	No real data migration need.

Table 6: Storage technologies and dependencies.

5.7.1 Security Component

The *Security Component* is made up of three submodules, of which two are those that have required data migration since the third, the *KrakenD API Gateway*, do not require any persistence and its configuration is already included in the running tool. In the case of KeyCloak, exporting the `smartbear` realm configuration in the source instance and reimporting it in the final production instance only requires manual comparison of the platform users, roles assigned and the re-generation of keys. Its basic operation is verified by attempting to authenticate the different clients. Regarding the UserManager subcomponent, caution is taken to coordinate the validation since the set must also match the list of participants contained in the CDR.

The data migration of user management coincided with the transition from the WSO2 to Keycloak. We had to migrate all the tables and entries into a new database with the appropriate mapping and we had to manage the following challenges:

- Correctly transfer the schema.
- Correctly transfer the entries.
- WSO2 has his own tables, and *UserManager* was using some attributes like the User Id. Keycloak uses other values for those attributes, so a translation/mapping had to be made.

For the migration of the database, the tool *datagrip* (from *Jetbrains*[41]) was used to perform those operations. For verification, the tool data-diff was used (*Figure 24*). The performed tests verify the correct transfer of the data. Specifically, the transferred records per DB table are:

- Users: 102 patients and 6 clinicians. The ids transformed from WSO2 UUIDs to Keycloak UUIDs. Comparing the data and the mapped UUIDs there are no differences during the migration (data-diff tools was used).
- Caregivers: 3 (data-diff result 0).
- CCM: 303 (data-diff result 0).
- Contact phones: 142 (data-diff result 0).
- Device metrics: 67740 (data-diff result 0).
- Devices: 76 (data-diff result 0).
- GDPR request: 0 (data-diff result 0).
- GRPR request files: 0 (data-diff result 0).
- Notification: 316 (data-diff result 0).
- Identities: 77 (data-diff result 0).

[41] JetBrains Developer tools (*https://www.jetbrains.com*)

```
16:35:50 INFO     Diffing using columns: key=('id',) update=None extra=('deleted_at', 'status', 'scope', 'updated_at', 'created_at',
                  'participant_id', 'identity_id').
         INFO     Using algorithm 'hashdiff'.
         WARNING  [PostgreSQL] Column 'identity_id' of type 'Text()' has no compatibility handling. If encoding/formatting differs
                  between databases, it may result in false positives.
         WARNING  [PostgreSQL] Column 'identity_id' of type 'Text()' has no compatibility handling. If encoding/formatting differs
                  between databases, it may result in false positives.
         INFO     Diffing segments at key-range: (1)..(77). size: table1 <= 76, table2 <= 76
16:35:51 INFO     Diff found 0 different rows.
         INFO     Duration: 1.23 seconds.
```

Figure 24: Sample data-diff tool for the validation of the 'identities' records.

More details about the transition from WSO2 to Keycloak and the data migration can be found in the deliverable *D5.6 Report on Continuous Security Assurance & Privacy by design - enabling mechanisms v3*.

5.7.2 Clinical Data Repository (CDR)

The validation of the CDR data migration is mainly based in counting the resources existing in the database, since them are handled by the standard implementation and in a certain way their integrity is guaranteed in terms of mandatory content and dependency relationships between them. The following counters are taken with the help of the endpoint `/fhir/$get-resource-counts` from the source environment and the target one, directly or via *PortForwarding*, and the results compared should directly match without any exception (*Figure 25*).

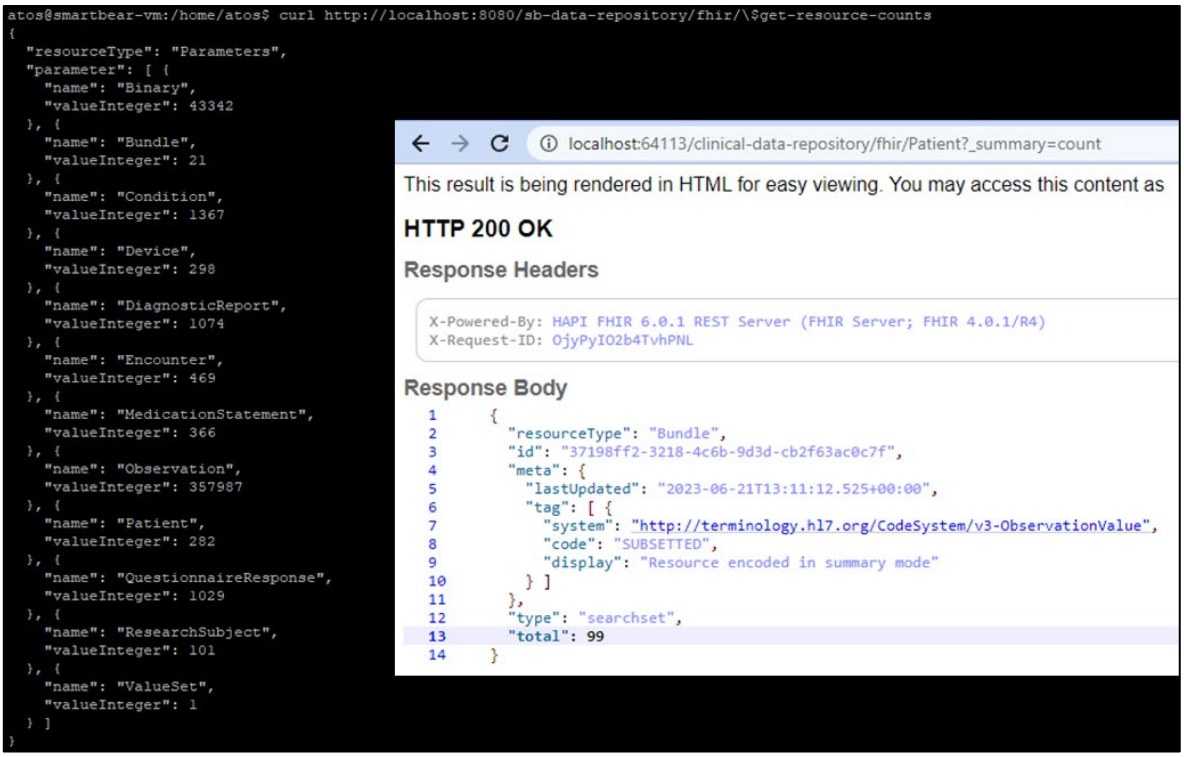

Figure 25: Use of get-resources-count endpoint for counting FHIR resources

If a specific type of resource needs to be compared, the specific endpoint pointing to it could be used.

```
                                                    // Endpoints to count FHIR resources
http://localhost:<portForwarding>/clinical-data-repository/fhir/Bundle?_summary=count
http://localhost:<portForwarding>/clinical-data-repository/fhir/CodeSystem?_summary=count
http://localhost:<portForwarding>/clinical-data-repository/fhir/Condition?_summary=count
http://localhost:<portForwarding>/clinical-data-repository/fhir/DiagnosticReport?_summary=count
http://localhost:<portForwarding>/clinical-data-repository/fhir/Encounter?_summary=count
```

```
http://localhost:<portForwarding>/clinical-data-repository/fhir/FamilyMemberHistory?_summary=count
http://localhost:<portForwarding>/clinical-data-repository/fhir/MedicationStatement?_summary=count
http://localhost:<portForwarding>/clinical-data-repository/fhir/Observation?_summary=count
http://localhost:<portForwarding>/clinical-data-repository/fhir/Patient?_summary=count
http://localhost:<portForwarding>/clinical-data-repository/fhir/Questionnaire?_summary=count
http://localhost:<portForwarding>/clinical-data-repository/fhir/QuestionnaireResponse?_summary=count
http://localhost:<portForwarding>/clinical-data-repository/fhir/ResearchSubject?_summary=count
http://localhost:<portForwarding>/clinical-data-repository/fhir/SearchParameter?_summary=count
http://localhost:<portForwarding>/clinical-data-repository/fhir/Subscription?_summary=count
http://localhost:<portForwarding>/clinical-data-repository/fhir/StructureDefinition?_summary=count
http://localhost:<portForwarding>/clinical-data-repository/fhir/ValueSet?_summary=count
http://localhost:<portForwarding>/clinical-data-repository/fhir/Device?_summary=count
```

5.7.3 Decision Support System (DSS)

The verification of the tables with data that were migrated demonstrated that all the entries were move to the Kubernetes instance of the database.

Table Name	Entries before migration	Entries after migration
Measurement	585698	585698
MeasurementFilter	60	60
Gamification	289	289
Intervention	35	35
Notification	18386	18386
Table Name	Entries before migration	Entries after migration

Table 7: DSS DBs data validation

5.7.4 SB Notifications

This data migration is done as a complete block with no other dependencies, so just comparing the sizes of the collections in the initial DB with the migrated DB and checking a few records to make sure everything was as expected, was considered enough for validation this data migration.

5.7.5 Android App Backend

Same situation as the previous case with the *SB Notifications* component data migration.

5.7.6 Big Data Analytics Engine (BDA)

This component does not need any extra validation since the data is not actually migrated but regenerated with the new FHIR and non-FHIR data found in the environment.

6 Testing

Main Objective in the Task 6.3 timeframe was ensuring safe transition from the PoP, with an environment based on simple virtual machines, to the large-scale pilots, with a structured framework based on Kubernetes.

A Platform lifecycle structured approach was set already from the beginning of the project (*cfr. D6.1 - Integrated SMAR BEAR platform v1, Chapter 5 Assistance and testing*) containing CI/CD processes, feedback collection and management, testing and assistance procedures (*Figure 26*).

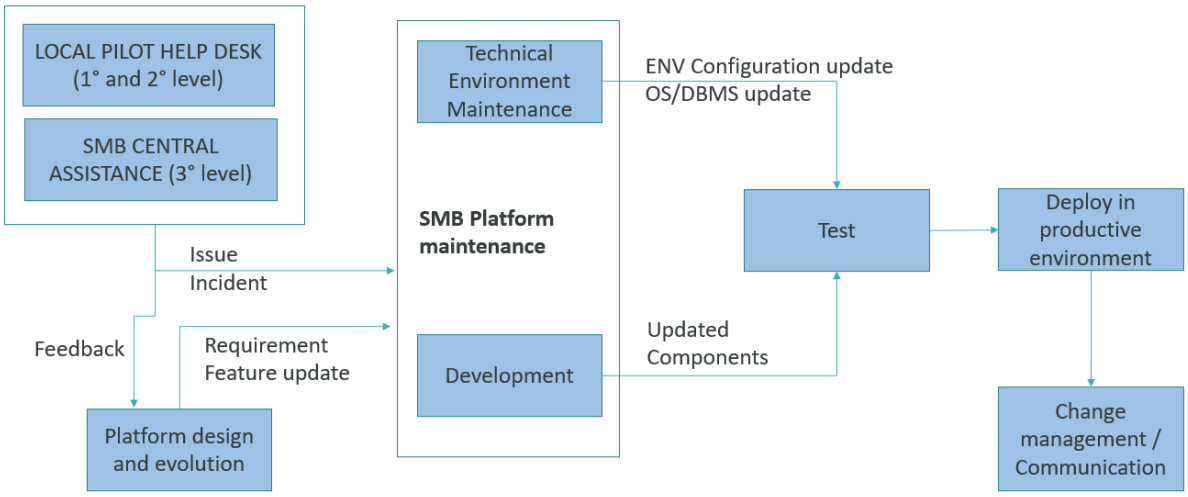

Figure 26: SMB Platform Assistance and Maintenance Process Framework

An important step made for the implementation of the designed processes, was the **setup of the Staging Environment**. A dedicated environment for testing (and training too), where data setup is accurately made for testing purpose. The configuration is intended to completely match with productive environment.

Components version update and Environment setup are automated by the GitLab CI/CD tools (*3.1 Setting up a staging environment*).

This clean environment allows to perform integration testing and functionality testing with effectiveness, and to execute automatic test cases.

The staging environment has led to the realization of two main assets, the first required to validate the migration to the K8s framework from a functional point of view, and the second one to ensure a safe and robust platform lifecycle.

The first asset is an **overall testing framework**, performed once, after the migration of the technical components and the data from PoP to the K8s new production environment, ensuring the continuity of the SMART BEAR service. The framework is also intended to be applied in case of future major releases of the platform.

The second asset is a **model** that is **continuously applied** once the migration is successfully over and represents the tools and the practices that support the **normal lifecycle of the SMB platform** in the K8s environment.

6.1 Overall testing framework

This framework is made of a set of **functional test cases**, and a **data migration validation check list**. The data migration validation check list is reported in chapter 5. Functional test cases are reported in the following sections.

6.1.1 Functional test cases

To support functional testing of the SMART BEAR platform and the mobile APP, the test cases have been grouped in a set of scenarios representing the main users' interaction paths in the SMART BEAR processes.

6.1.2 Scenarios and related test cases

The considered scenarios aim to identify, following the process that goes along the enrolment and the monitoring of a patient, what are considered as the main functionalities of the platform, empowering study participants, case managers and technicians for an extensive operability.

The following *Table 8* shows how scenarios and the relevant test cases were organized.

SCENARIO	TESTCASE ID	TESTCASE DESCRIPTION
User registration and profiling (Dashboard)	SMBS1_TC1	User registration and profiling
Patient Enrolment: Patient creation, editing patient data and questionnaires, testing exclusion criteria	SMBS2_TC1	Creation of a patient
	SMBS2_TC2	Editing patient data (demographics)
	SMBS2_TC3	Checking exclusion criteria
	SMBS2_TC4	Baseline assessment: Editing questionnaires
Kit preparation and distribution: devices pairing with APP, measurements test	SMBS3_TC1	Smartwatch setup
	SMBS3_TC2	Hearing aids pairing with Smartphone
	SMBS3_TC3	SMB APP installation and devices configuration on the smartphone
	SMBS3_TC4	Device management in the Dashboard
Monitoring: APP end user functionalities, 6 months visit	SMBS4_TC1	Heart rate screens
	SMBS4_TC2	Blood Pressure Screen
	SMBS4_TC3	Sleep Statistics Screen
	SMBS4_TC4	Daily Steps – Activity
	SMBS4_TC5	Weight screen
	SMBS4_TC6	Temperature screen
	SMBS4_TC7	Oximeter screen
	SMBS4_TC8	Hearing Aids screen

	SMBS4_TC9	Light Levels Screen
	SMBS4_TC10	Movement Screen
	SMBS4_TC11	Comfort Screen
	SMBS4_TC12	Medication
	SMBS4_TC13	Diet screen on the APP
	SMBS4_TC14	Rehabilitation screens
	SMBS4_TC15	Measurements page (Dashboard, CCM)
	SMBS4_TC16	Measurements page (Dashboard, HD operator/tech support)
Analytics management: testing BDA functionalities on Dashboard	SMBS5_TC1	Showing analytics in the BDA
Intervention creation and delivery: from a measurement to an intervention	SMBS6_TC1	Rule-based interventions creation for Cardiovascular diseases
	SMBS6_TC2	Intervention creation and delivery (based on DSS rules)
	SMBS6_TC3	Intervention creation and delivery (based on APP rules)
	SMBS6_TC4	Interventions notification filter
Synergies	SMBS7_TC1	Smart4Health synergy test
	SMBS7_TC2	*Holobalance* synergy test

Table 8: Scenarios and related test cases for the overall testing framework

Test cases specify how tests are performed, what functionalities are covered and what results are expected. They are characterised by an identifier, a description, the necessary preconditions to be satisfied for a valid test, a clear description of the procedure to be applied and the expected results.

Below *Table 9* is an example of a compiled test case:

Id	SMBS2_TC1
Description	Creation of a patient
Precondition	CCM “test_clinician” profile already registered

Use case/functionality covered	The CCM “test_clinician” log in and creates a patient.
	• Login in the Dashboard as CCM
	• Click on the Patients button
	• Click on Creating a new Patient button
	• Edit
	  • Pilot = Test
	  • Organization = Test organization
	  • SB Email address = sb-07-001@gmail.com
	  • Date of participant consent = day before the test

	• Phone number = +393331234567 • Username of CCM = test_clinician • Email of External CCM Click on the CREATE PATIENT Check OVERVIEW page
Expected result	• Patient created. Patient ID = 2 (to be confirmed) • Data in Overview page correct

Table 9: Example of a test case

Each scenario has also been linked to the technology components required to run the associated test cases *(Table 10*).

	Devices	APP (back end, SB notification)	FHIR DB, NON FHIR DB, clinical data repository	Dashboard	BDA	DSS	Security Component (user manager, keycloak)
User registration and profiling for the Dashboard			yes	yes			yes
Patient Enrolment			yes	yes			yes
Kit preparation and distribution	yes	yes					yes
Monitoring	yes	yes	yes	yes	yes		yes
Analytics management			yes	yes	yes		yes
Intervention creation and delivery	yes	yes	yes	yes		yes	yes
Synergies			yes	yes			yes

Table 10: Relationship between scenarios and platform components

All the functional test cases are detailed in the ***ANNEX III. Testing***.

6.1.3 Functional testing procedure

Before the test cases could be executed, it was necessary to ensure that some pre-requirements (additional to the ones specified in the test cases) were in place:

- an active and accessible staging environment with all the necessary components installed and aligned in terms of version, in order to ensure correct interaction.
- the presence in the staging environment databases of the preliminary data and the configuration of the needed user roles (for example in relation to devices, the use of the correct account for synergy testing, etc).
- APP APK configured to point to the staging environment

All the technical issues raised during functional testing were registered on basecamp, on a dedicated card table.

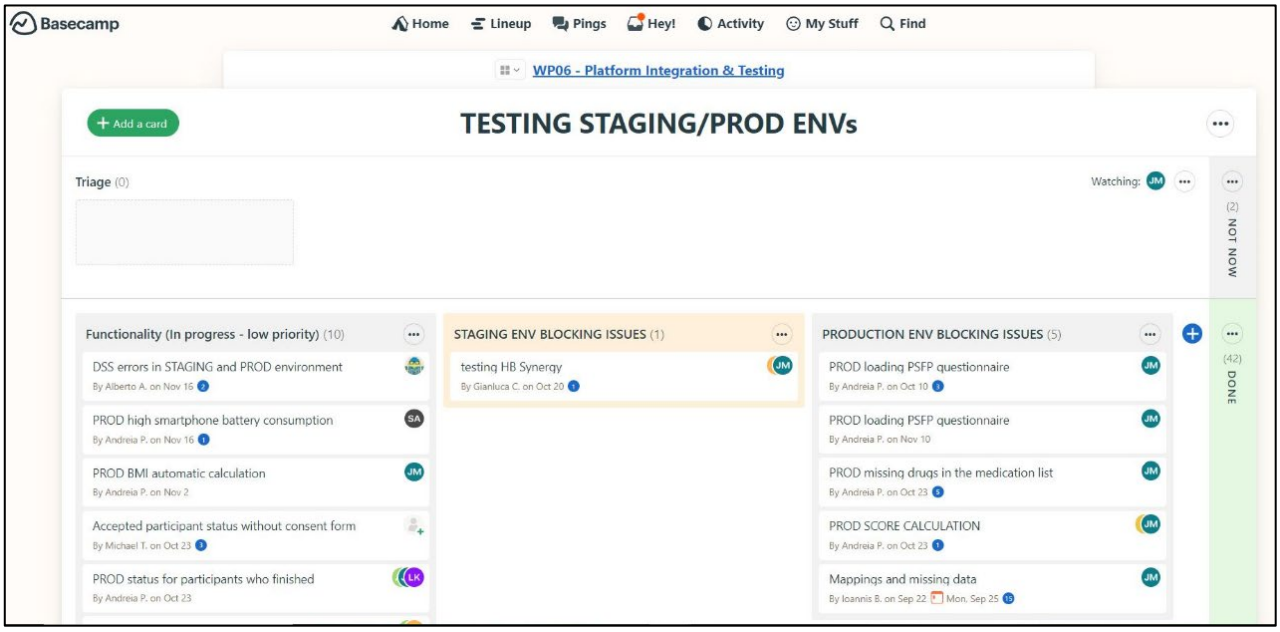

Figure 27: Tracking and managing issues in basecamp

The card table consents the users and the operators to create new cards, indicating a in a title the summary of the issue, specifying the environment, and providing a description. The assignment of the card is up to the WP6 team members, so as the placement of the card in the dedicated swim lanes (evolutions, staging env, production env). A card can be commented by all the actors and, once the input is accomplished and the issue solved, can be stored as done.

6.1.4 Functional testing results

The application of test cases has run from June to October 2023 and allowed the release of the platform in K8s to be fine-tuned. All the results were registered in an appropriate table, referring to the test case ID, the test executer, the date, the result and some additional notes.

The following table reports the results of the application of the test cases described in the present chapter. Tough for each test session, all test cases are performed, the mentioned date is the first date where a test case passed successfully.

Some test cases are not executed yet, due, for example, to the unavailability of specific devices. However, this did not affect the overall functioning of the new version of the platform (*Table 11*).

TEST CASE ID	TEST CASE description	Tester	Date of execution	Result	Notes
SMBS1_TC1	User registration and profiling	STS	-	passed	Administrator user needed.
SMBS2_TC1	Creation of a patient	ARIA	30/6/2023	passed	
SMBS2_TC2	Editing patient data (demographics)	ARIA	10/7/2023	passed	
SMBS2_TC3	Checking exclusion criteria	ARIA	10/7/2023	passed	
SMBS2_TC4	Baseline assessment: Editing questionnaires	ARIA	10/7/2023	passed	
SMBS3_TC1	Smartwatch setup	ARIA	10/7/2023	passed	
SMBS3_TC2	Hearing aids pairing with Smartphone			-	No HAs available for testing in ARIA.
SMBS3_TC3	SMB APP installation and devices configuration on the smartphone	ARIA	10/7/2023	passed	Scale not paired.
SMBS3_TC4	Device management in the Dashboard	ARIA	10/7/2023	passed	
SMBS4_TC1	Heart rate screens	ARIA	10/7/2023	passed	
SMBS4_TC2	Blood Pressure Screen	ARIA	7/8/2023	passed	No BP device available for testing in ARIA. Measurements simulated by APP owner.
SMBS4_TC3	Sleep Statistics Screen	ARIA	10/7/2023	passed	
SMBS4_TC4	Daily Steps – Activity	ARIA	10/7/2023	passed	
SMBS4_TC5	Weight screen	ARIA	10/7/2023	not passed	Issues in pairing multiple iHealth devices.
SMBS4_TC6	Temperature screen	ARIA	10/7/2023	passed	Sometimes measurements are not received by the APP (issues in pairing multiple iHealth devices).
SMBS4_TC7	Oximeter screen	ARIA	10/7/2023	passed	
SMBS4_TC8	Hearing Aids screen			-	No HAs available for testing in ARIA.
SMBS4_TC9	Light Levels Screen			in progress	Fake account available for testing.
SMBS4_TC10	Movement Screen			in progress	Fake account available for testing.
SMBS4_TC11	Comfort Screen			in progress	Fake account available for testing.
SMBS4_TC12	Medication	ARIA	13/10/2023	passed	

TEST CASE ID	TEST CASE description	Tester	Date of execution	Result	Notes
SMBS4_TC13	Diet screen on the APP	ARIA	13/10/2023	passed	
SMBS4_TC14	Rehabilitation screens				
SMBS4_TC15	Measurements page (Dashboard, CCM)				DBA engine not available yet in staging ENV.
SMBS4_TC16	Measurements page (Dashboard, HD operator/tech support)	ARIA	10/7/2023	not passed	All measurements collapsed to smartphone device. Fix currently under development.
SMBS5_TC1	Showing analytics in the BDA				DBA not available yet in staging ENV.
SMBS6_TC1	Rule-based interventions creation for Cardiovascular diseases	ARIA	24/7/2023	passed	
SMBS6_TC2	Intervention creation and delivery (based on DSS rules)	ARIA	04/08/2023	passed	
SMBS6_TC3	Intervention creation and delivery (based on APP rules)	ARIA	13/10/2023	passed	
SMBS6_TC4	Interventions notification filter				
SMBS7_TC1	Smart4Health synergy test				Testcase to be designed in detail.
SMBS7_TC2	Holobalance synergy test				Testcase to be designed in detail.

Table 11: Functional test cases report

6.2 Continuous testing during large scale pilots

During large-scale pilot projects, frequent component upgrades must be supported. Continuous improvement of software components involves an approach of continuous testing according to TRL9 requirements. The following image shows the maintenance and testing cycle of the platform during its ordinary lifetime (*Figure 28*).

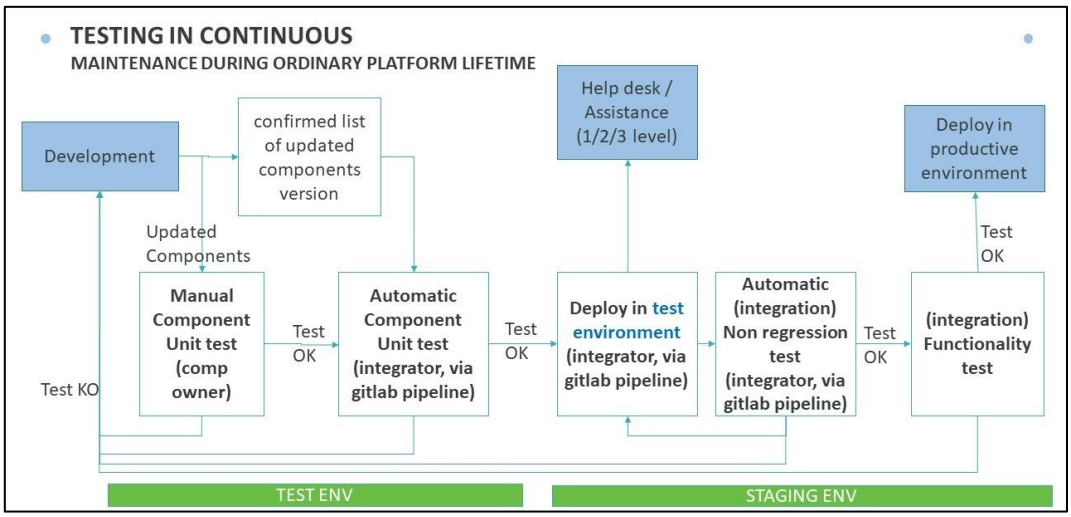

Figure 28: Maintenance and testing cycle

6.3 Automatic (integration) test

The non-regression (integration) testing phase is set up to use an automated approach. Tests are performed before each release of a new component, in addition to manual testing of use cases dedicated to the specific topic of the release (fix, new functionality). Integration tests are intended to verify a subset of functionalities considered vital or "blocking" for the proper functioning of the platform. The "core" feature set is identified as follows (*Table 12*):

Use case	Components involved	To date state of play
Patient's medical history data retrieve	Dashboard, FHIR DB, Non FHIR DB, Security component	Completed
Patient's medical history update (or creation)	Dashboard, FHIR DB, Non FHIR DB, Security component	On going
Transfer of a measurement from device to clinical repositories	APP, Dashboard, clinical data repositories, FHIR DB, Non FHIR DB, Security component	To be started
Intervention creation	APP, Dashboard, clinical data repositories, FHIR DB, Non FHIR DB, Security component, DSS	To be started

Table 12: The "core" feature set

A new dedicated component has been introduced to run automated tests in the staging environment. Following the 'test in continuous' approach, integration testing has to be performed successfully in the staging environment before a new platform version is released in the production environment. The Selenium framework is used to simulate user interaction via the dashboard. Tests are triggered by the GitLab CI/CD pipeline.

To support automated testing, a dedicated Selenium Grid service was deployed in the staging environment, initially consisting of the Selenium Hub and a Chrome node (*Figure 29*).

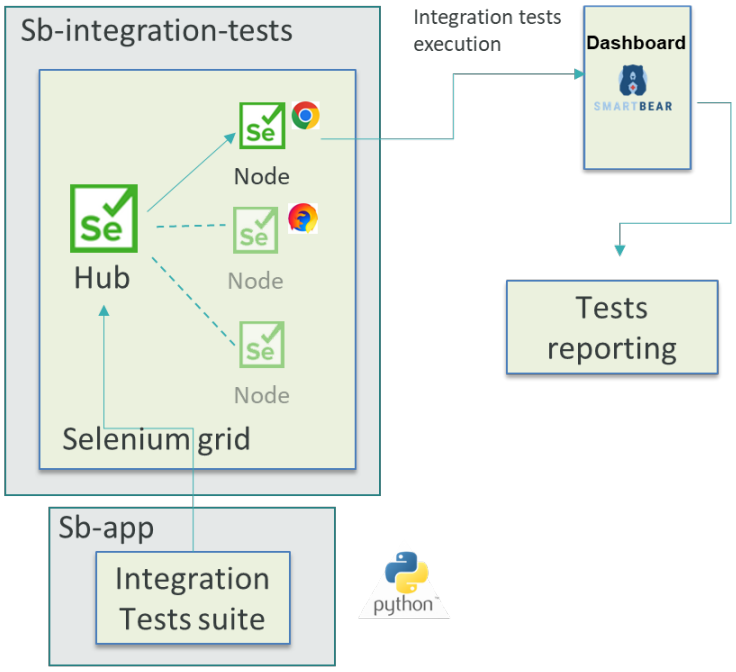

Figure 29: SB-integration component

Sb-app is a container specifically dedicated to hosting the test suite procedures.

7 Future maintenance needs

The installation and setup process of the Kubernetes PROD cluster is just an initial phase of the provision of the needed infrastructure and the successful deployment of the platform is not the end of its maintenance. In addition, if any security problem is detected in the components that make up the infrastructure, it must be audited and, if necessary, its components updated, on a regular basis, to prevent major problems.

7.1 Resources monitoring

In the following weeks to the switch of the production environments and more exhaustive use of the platform is made in the Kubernetes cluster, it must be monitored to verify that the resources estimations have been correct, and that the metrics obtained by the monitoring systems remain within the expected figures.

As introduced in section *4.6 Monitoring*, the tool *Prometheus* was installed and its service provide in its own UI (that could be accessed via *Port Forwarding* since is not exposed) or via other UIs and IDEs as *OpenLens* where CPU use, memory allocated by the components and storage disk current use and trend are visualized. No specific alerts have not been defined or configured as they are not considered necessary since regular use is expected and there is no need for any special scaling.

7.2 Proper backups execution

The Kubernetes setup of the platform ensures that the pod will recover in case they fail, or the environment is restarted. Regarding the backup strategy used for the database which contain the data, there is a twofold strategy applied. A number of scheduled jobs are executed to back up the data to another VM which is sharing a Network File Storage installation. In addition to that, the whole VM of the master and nodes of the Kubernetes cluster is snapshotted every day and stored to an external to the server location, to ensure the redundancy of the data. The VMs are deployed on a Windows Server using Hyper-V which implements isolation of virtual machines in terms of a partition, supported by the hypervisor, in which each guest operating system executes.

7.3 Kubernetes CVE

A CVE (Common Vulnerabilities and Exposures) is a system that provides a reference method for publicly known information-security vulnerabilities and exposures[42]. Kubernetes is a platform in continuous development, and it is not out of the dangers posed by security vulnerabilities in any software system, and in order to warn an alert of any risk, an official CVE feed is maintained by the community[43]. In case a critical risk is published, and a path is generated to mitigate vulnerability, the cluster should be upgraded according.

[42] CVE, Common Vulnerabilities and Exposures (*https://en.wikipedia.org/wiki/Common_Vulnerabilities_and_Exposures*)
[43] Kubernetes Official CVE Feed *(https://kubernetes.io/docs/reference/issues-security/official-cve-feed*)

7.4 GitLab instance migration

During the first 52 months of the project, ATOS is the partner providing the centralized repository for code and Atos also manages such infrastructure, being this system also in charge of centralizing the CI/CD tasks and jobs for deploying the in a suitable automatized way to the different environments of the SMART BEAR platform. The consortium agreed on transferring the GitLab repo to ICCS, who will be the partner in charge of managing this infrastructure. Now the code and images are hosted on ICCS GitLab instance located at `https://gitlab.biomed.ntua.gr/smartbear`.

Direct migration was not supported, so each of the repositories were exported from ATOS GitLab instance and then was imported in ICCS GitLab instance, since complete group export do not include the whole set of projects.

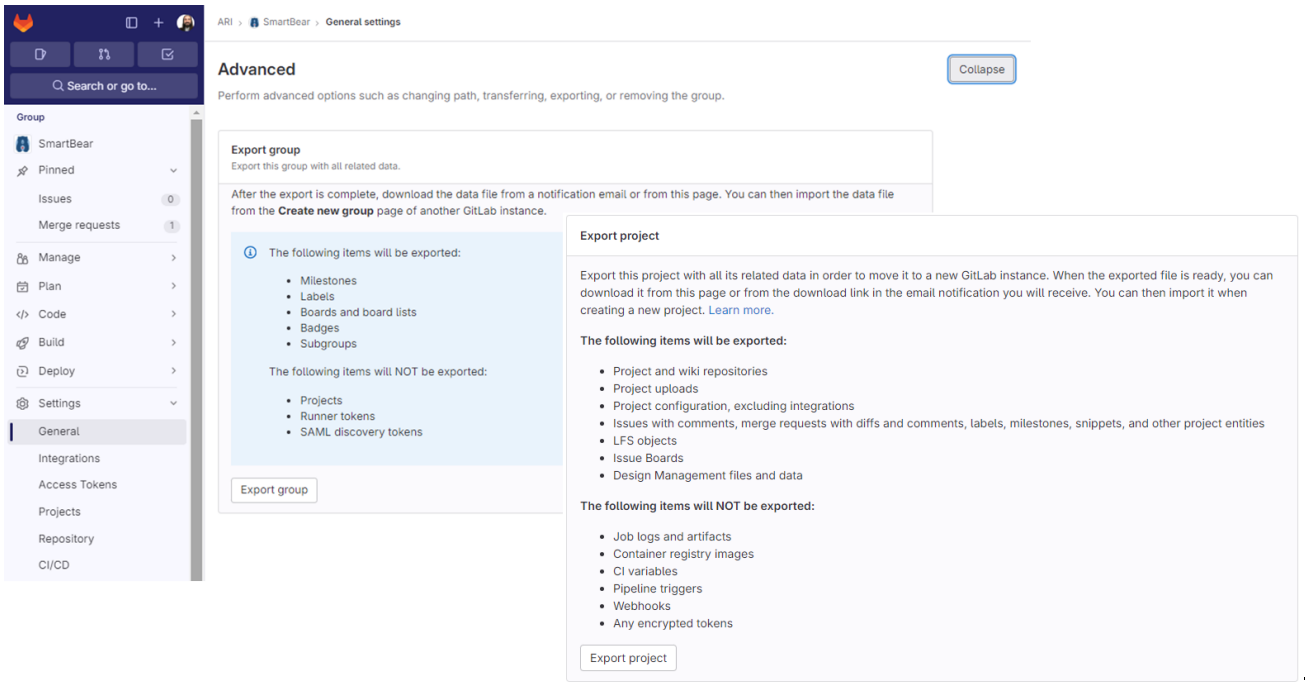

Figure 30: GitLab exporting "Smartbear" group and specific single project.

The respective groups and users of the new GitLab instance were created to allow the access of the developer to the new repositories.

The necessary environment variables were copied and adjusted for the new instance as they are excluded in the export. A major change was the transition of the image repository of the components from, Nexus used by ATOS to GitLab internal image repository. That simplifies the architecture of the CI/CD but it also required a series of steps and actions to be modified to make the pipelines compatible with the new flows. Such adjustments were the creation of new runners that monitor changes on the code and trigger the deployment jobs to the cluster along with project variables contained in the configuration files of each module.

The following list is summarizing the data migrated to the new instance, based on the recommendations of an GitLab article[44]:

[44] GitHub to GitLab migration the easy way, *(https://about.gitlab.com/blog/2023/07/11/github-to-gitlab-migration-made-easy/)*

- Repository description
- Git repository data
- Branch protection rules
- Collaborators (members)
- Pull requests
- Wiki pages
- Labels
- Release notes content
- Release notes attachments
- Comment attachments
- Pull request description attachments
- Pull request review comments
- Regular issue and pull request comments
- Git Large File Storage (LFS) objects
- Pull request reviews

After the migration to the new GitLab instance, the component owners performed a number of tests to verify that the functionality of the CI/CD environment remained the same.

ANNEX I. Troubleshooting

As an aid to problems experienced during the implementation of the platform, some of the problematic circumstances that occurred, and their solution are detailed below.

MariaDB can't boot

Sometimes, if the *pod* running the MariaDB server stops abruptly, due to unexpected crash or forced hardware shutdown, some files on the persistence will remain in a status the server consider not secure, and aborts the boot with messages of the following type:

// MariaDB not able to boot

```
[Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.6.4+maria~focal started.
[Note] mysqld (server 10.6.4-MariaDB-1:10.6.4+maria~focal) starting as process 1 ...
[Note] InnoDB: Compressed tables use zlib 1.2.11
[Note] InnoDB: Number of pools: 1
[Note] InnoDB: Using crc32 + pclmulqdq instructions
[Note] mysqld: O_TMPFILE is not supported on /tmp (disabling future attempts)
[Note] InnoDB: Using Linux native AIO
[Note] InnoDB: Initializing buffer pool, total size = 134217728, chunk size = 134217728
[Note] InnoDB: Completed initialization of buffer pool
[Note] InnoDB: Starting crash recovery from checkpoint LSN=4475411201,4475411201
[ERROR] InnoDB: Missing FILE_CHECKPOINT at ***** between the checkpoint 4475411201 and the end *****.
[ERROR] InnoDB: Plugin initialization aborted with error Generic error
[Note] InnoDB: Starting shutdown...
[ERROR] Plugin 'InnoDB' init function returned error.
[ERROR] Plugin 'InnoDB' registration as a STORAGE ENGINE failed.
[Note] Plugin 'FEEDBACK' is disabled.
[ERROR] Unknown/unsupported storage engine: InnoDB
[ERROR] Aborting
```

Generally, it does not have more consequences than the warning, and open transactions that were pending would not have been executed. In order to restore the DB engine operation, log files (`*aria_log*` and `*ib_logfile*`) should be removed, which can be done by manually accessing the file system mounted as a volume, or through the utility provided in the project repository under project `/smartbear/pvc-inspector/smartbear-pvc-inspector-cdr-database-pvc`, with a CI/CD job `deploy-smartbear-pvc-inspector-clean-mariadb` to whom, informing him of the parameters `NAMESPACE` and `PVC_INSPECTOR_MARIADB_PVC_NAME` to automatize the process. After that, the server should boot normally.

MariaDB migration schema exceptions

Using Hibernate MariaDB dialects after 10.3, creates non expected `Sequence` resources instead of `base tables` types not supported by the upgrading tool.

// HAPI-FHIR schema exception

```
[main] INFO  c.u.f.j.m.HapiMigrator Executing 6.0.0.20220207.8 Add IDX_SP_DATE_ORD_HASH_HIGH_V2 index to
table HFJ_SPIDX_DATE
2023-08-18 14:38:07.19 [main] INFO  c.u.f.j.m.t.AddIndexTask 6.0.0.20220207.8: Going to add a NON-UNIQUE
index named IDX_SP_DATE_ORD_HASH_HIGH_V2 on table HFJ_SPIDX_DATE for columns [HASH_IDENTITY,
SP_VALUE_HIGH_DATE_O
RDINAL, RES_ID, PARTITION_ID]
2023-08-18 14:42:02.60 [main] WARN  o.m.j.m.s.ErrorPacket Error: 1061-42000: Duplicate key name
'IDX_SP_DATE_ORD_HASH_HIGH_V2'
2023-08-18 14:42:02.88 [main] ERROR c.u.f.j.m.HapiMigrator Migration failed
```

```
ca.uhn.fhir.jpa.migrate.HapiMigrationException: HAPI-0047: Failure executing task "Add
IDX_SP_DATE_ORD_HASH_HIGH_V2 index to table HFJ_SPIDX_DATE", aborting! Cause:
ca.uhn.fhir.jpa.migrate.HapiMigrationException: HAPI-
0061: Failed during task 6.0.0.20220207.8: org.springframework.jdbc.BadSqlGrammarException:
PreparedStatementCallback; bad SQL grammar [create index IDX_SP_DATE_ORD_HASH_HIGH_V2 on
HFJ_SPIDX_DATE(HASH_IDENTITY, SP_VALU
E_HIGH_DATE_ORDINAL, RES_ID, PARTITION_ID)]; nested exception is java.sql.SQLSyntaxErrorException:
(conn=25) Duplicate key name 'IDX_SP_DATE_ORD_HASH_HIGH_V2'
```

Deployments wrong scaling

Kubernetes *Deployments* are supported by *ReplicaSets*, which are objects that, in case of lose the relation with the parent *Deployment* could, in an independent way, trying to keep its own replica pod set alive. This can cause unexpected situation, with a number of pods trying to access to the same resource, like a DB engine that only supports a single client, not letting the last and only expected instance to boot properly, so the obsolete *ReplicaSets* should be removed to return to the expected situation.

Headers too big for nginx

The configuration by default of nginx ingress controller performed via Helm Chart, could not be enough for some use of headers, for example if including big cookies on them, and the requests fail with the error log:

```
upstream sent too big header while reading response header from upstream,
```

In the `values.yaml` file could be included the ingress-nginx needed configuration, that will be stored in the *ConfigMap* the pod uses when starts.

Add in the nginx installation.

```
config: # SMART BEAR
    client-header-buffer-size: "16k"
    large-client-header-buffers: "4 16k"
```

Do not forget that requests access the cluster through a reverse proxy (HAProxy), which by default sets the following buffer sizes:

```
tune.bufsize=16384
tune.maxrewrite=8192 # Auto to half of bufsize
```

In case it needs to be increased, in the master node, global section of the `/etc/haproxy/haproxy.conf` file, the previous global parameters could be adjusted following the recommendations[45]. Still note that the servers http header size by default in most of them are under such figure, and all the chain of proxies should be coherent in this adjustment.

If an HTTP request is larger than (tune.bufsize - tune.maxrewrite), haproxy will return HTTP 400 (Bad Request) error. Similarly, if an HTTP response is larger than this size, haproxy will return HTTP 502 (Bad Gateway)[46]. Logs could be checked via:

```
journalctl -u haproxy.service
```

[45] HAProxy configuration, *(https://cbonte.github.io/haproxy-dconv/2.3/configuration.html#3.2-tune.maxrewrite)*
[46] Nginx Custom max body size annotation (*https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/#custom-max-body-size)*

Nginx client intended to send too large body

By default, nginx ingress configuration of maximum body payload size in the requests is set to just 1Mb[47], and this could be too small for certain data transfer. Increased by adding the `client_max_body_size` annotation to the Ingress controller config map, setting and extended size:

```
data:
...
  allow-snippet-annotations: 'true'
  client-header-buffer-size: 16k
  large-client-header-buffers: 4 16k
  client_max_body_size: 8m
```

Nginx failed to update lock: configmaps

The default installation of nginx ingress controller performed via Helm Chart, could throw the following error traces in the pod log:

```
Failed to update lock: configmaps "ingress-controller-leader" is forbidden: User
"system:serviceaccount:kube-system:ingress-nginx-1688128989" cannot update resource
"configmaps" in API group "" in the namespace "kube-system"
```

Can be solved by providing the cluster role the proper permission:

```
- apiGroups:
  - ''
verbs:
  - update
  resources:
  - configmaps
```

Calico IPIP mode and performance

By default, Calico network plugin is installed to allow overlay networking across multiple VPCs/subnets although this is not the way it provides the fest performance, to selectively encapsulate only the traffic that is routed between the VPCs/subnets and run without encapsulation within each VPC/subnet. Running the entire Calico network with encapsulation as an overlay network is a quick way to get started without setting up BGP peering or other routing information in your underlying network. To disable it, in the IP Pool configuration we can disable it by switching:

```
ipipMode: Always
```

to

```
ipipMode: Off
```

Bulk export jobs

The CDR component provides the HL7 FHIR *bulk data export* functionality defined in the corresponding Implementation Guide[48] to retrieve massive data and intended to be used by developers of backend services (clients) and FHIR Resource Servers, defining APIs through which a client may request an export from a server, check the progress in the generation of the requested

[47] HAProxy configuration, *(https://cbonte.github.io/haproxy-dconv/2.3/configuration.html#3.2-tune.bufsize)*
[48] HL7 FHIR bulk data export (*https://hl7.org/fhir/uv/bulkdata/export.html)*

files, and eventually, the link to download such data. Components of the SMART BEAR platform could use it, like the BDA, but its use should be based on a proper use and limited by some restrictions to so as not to undermine performance.

This operation could be very demanding in terms of computing, time and storage, since it creates binary resources to contain the files to be later downloaded and, in case of not using properly the parameter "_since" (to use it in incremental way) a near future blocking process.

The HAPI-FHIR implementation in versions prior to 6.6.0 based the jobs maintenance in three tables in the DB (`hfj_blk_export_job`, `hfj_blk_export_collection` and `hfj_blk_export_colfile`), with the creation time, expiration (by default, after 2h), job_id, request that originates the job and status, but last versions moved to "batch2" strategy, using an alternative `bt2_job_instance` table, keeping obsolete the old table. Furthermore, no endpoint to delete pending jobs is implemented, so a drastic way of get rid of pending jobs (since they will be resumed even after rebooting the system as long as they remain pending) is to modify the status directly in the table, based on the following `ca.uhn.fhir.batch2.model.StatusEnum` definition:

- `QUEUED`: Task is waiting to execute and should begin with no intervention required,
- `IN_PROGRESS`: Task is current executing,
- `FINALIZE`: For reduction steps,
- `COMPLETED`: Task completed successfully,
- `FAILED`: Task has failed and is known to be unrecoverable. There is no reason to believe that retrying will result in a different outcome,
- `CANCELLED`: Task has been cancelled by the user.

DeadlineExceeded desc = context deadline exceeded

Sometimes, the default timeout dealing with the NFS could be not enough[49] and needs to be extended, but the Helm Chart that install the NFS-CSI driver do not allow to provide arguments neither to the *provisioner* nor the *snapshotter*. But after the installation, in the `csi-nfs-controller` Deployment descriptor, more arguments could be provided manually, like extending to 120 seconds the timeout: permission:

```
containers:
  - name: csi-provisioner
    image: registry.k8s.io/sig-storage/csi-provisioner:v3.5.0
    args:
      - '-v=2'
      - '--csi-address=$(ADDRESS)'
      - '--leader-election'
      - '--leader-election-namespace=kube-system'
      - '--extra-create-metadata=true'
      - '--timeout=120s'[50] # 1 minute is used by default
      - --retry-interval-start=1s # Default value is 1 second
      - --retry-interval-max:300s # Default value is 5 minutes
  - name: csi-snapshotter
    image: registry.k8s.io/sig-storage/csi-snapshotter:v6.2.2
    args:
      - '--v=2'
```

[49] CSI timeout handling, *(https://github.com/kubernetes-csi/external-provisioner#csi-error-and-timeout-handling)*
[50] CSI-Provisioner arguments, *(https://github.com/kubernetes-csi/external-provisioner#recommended-optional-arguments)*

```
        - '--csi-address=$(ADDRESS)'
        - '--leader-election-namespace=kube-system'
        - '--leader-election'
        - '--timeout=120s'[51] # 1 minute is used by default
```

After the edition, restarting the deployment, we should verify in the pod logs that the config was taken properly:

// NFS CSI app log showing configuration set

```
1 main.go:104] Version: v6.2.2
I0929 07:19:04.315024       1 main.go:184] CSI driver name: "nfs.csi.k8s.io"
I0929 07:19:04.315067       1 common.go:111] Probing CSI driver for readiness
I0929 07:19:04.316719       1 main.go:222] Start NewCSISnapshotSideCarController with snapshotter
[nfs.csi.k8s.io] kubeconfig [] csiTimeout [2m0s] csiAddress [/csi/csi.sock] resyncPeriod [15m0s]
snapshotNamePrefix [snapshot] snapshotNameUUIDLength [824637349968]
I0929 07:19:04.319040       1 leaderelection.go:248] attempting to acquire leader lease kube-
system/external-snapshotter-leader-nfs-csi-k8s-io...
I0929 07:19:04.336902       1 leaderelection.go:258] successfully acquired lease kube-system/external-
snapshotter-leader-nfs-csi-k8s-io
I0929 07:19:04.337009       1 leader_election.go:178] became leader, starting
```

[51] CSI-Snapshotter arguments, *(https://github.com/kubernetes-csi/external-snapshotter#important-optional-arguments-that-are-highly-recommended-to-be-used)*

ANNEX II. Checklist

To align the different components integration level and actions performed, the following checklist was produced and shared during the migration, focusing on the basic alignment in configuration and deployment processes pathway through the different environments.

Note that this excludes the test of specific functionality.

☐ Source code is maintained updated in the centralized project GitLab repository.

☐ Source code do not contain hardcoded configuration that potentially need to be adjusted outside the folder /deployment.

☐ Project README.md contains the basic instructions on requirements for extra needed configuration, like expected environmental variables, tokens/passwords, etc.

☐ Source code do not contain hardcoded any sensitive detail (tokens or passwords), this should be maintained in independent infrastructure objects, such as secrets in Kubernetes.

☐ Source code do not print in logs any sensitive data and differentiates the trace level between development and production environments, in the case that it may be decisive for its understanding or the performance of the component.

☐ Git branches are maintained aligned with agreed procedures, this is "*develop*" branch for deploying to develop environment and tagging master/main branch commits for consecutive staging/prod environments.

☐ CI/CD deployment jobs for deploying into develop/staging environments have been verified.

☐ CI/CD deployment jobs for deploying into prod environment have been verified.

☐ Docker images are tagged according to a clear pattern aligned with git tagging and Kubernetes deployment descriptor files, following the recommendations in the Wiki.

☐ Validate access to the Kubernetes clusters (develop/staging and prod) and basic operation with the deployments: restart pods, access pod logs, validate *ConfigSet* and Secret objects, etc.

☐ Basic recovery processes for common issues have been documented (e.g.: boot stopped due to DB engines protection mode, or dependencies on third-party services not available at the moment).

☐ Basic log trace review through the component functionalities, cleaning any log classified as ERROR if they are actually not. This includes the passing through the gatekeeper API when applicable.

☐ Confirm that component owners are aware of the operation of the backup mechanisms, scope and use of it (recover from it) in case of need (*VolumeSnapshots* or whole NFS filesystem copy).

☐ Both WP6 cards boards, "SMART BEAR Platform I&T" and "STAGING ENV TESTING" are reviewed, providing feedback on issues not resolved and categorizing them by priority.

ANNEX III. Testing

Functional test cases introduced in chapter *6 Testing* are detailed in the present annex, following the scenarios order.

User registration and profiling (Dashboard)

User registration and profiling test is applied to Clinical Case Manager (CCM) and technical operator profiles.

User registration and profiling

Id	SMBS1_TC1
Description	User registration and profiling.
Precondition	-
Use case/functionality covered	New users on the dashboard are created for the following profiles: • Clinical Case Manager (CCM). • Operator profile. Actions (to be performed for each profile): • Click on the Create Account button in the Login form. • Fill up username field. • Fill up First Name, Last Name, Email, Password and organization (choose UMIL), accept privacy consent, press register button. • The administrator accepts the new user and profiles it. • Confirm the user registration using the email received on the submitted email address. • Login. Check for available views and information access according to the profile.
Expected result	Case Manager user and Operator user created, profiled and login enabled.

Patient Enrolment

The present scenario simulates the first visit for a study participant, and consists in the following test cases:

- Creating a new patient (study participant).
- Editing patient data (demographics).
- Checking exclusion criteria.
- Baseline assessment.

Creation of a patient

Id	SMBS2_TC1
Description	Creation of a patient.
Precondition	CCM "test_clinician" profile already registered.

Use case/functionality covered	The CCM "test_clinician" log in and creates a patient.
	• Login in the Dashboard as CCM.
	• Click on the Patients button.
	• Click on Creating a new Patient button.
	• Edit:
	  • Pilot = Test.
	  • Organization = Test organization.
	  • SB Email address = sb-07-001@gmail.com.
	  • Date of participant consent = day before the test.
	  • Phone number = +393331234567.
	  • Username of CCM = test_clinician.
	  • Email of External CCM.
	Click on the CREATE PATIENT.
	Check OVERVIEW page.

Expected result	Patient created. Patient ID = 2 (to be confirmed). Data in Overview page correct.

Editing patient data (Demographics)

Id	SMBS2_TC2
Description	Editing patient data (demographics).
Precondition	CCM "test_clinician" profile already registered. Patient ID 2 profile already created.

Use case/functionality covered	During the first visit for a study participant, CCM edits a patient's demographics.
	• Login in the Dashboard as CCM test_clinician.
	• Selected of Patient ID 2 (edit).
	• Fill the following fields in the overview section:
	  • Age = 1945 august 22nd.
	  • Gender = female.
	  • Type of accommodation = Housed.
	• Fill the following fields on the demographic section:
	  • Biological Gender = Male.
	  • Education level = Elementary School.
	  • Living situation = Lives in apartment with elevator access.
	  • Source of referral = social media ads.
	  • Ethnicity = Caucasian.
	  • Lives with = Alone.

	• Type of accommodation = Housed. • Set the patient as ACCEPTED (patient page).
Expected result	Demographics data added to the repository. Patient ID 2 set as active.

Checking exclusion criteria

Id	SMBS2_TC3
Description	Checking exclusion criteria.
Precondition	CCM profile and Patient profile already registered. Patient data fulfilled (demographics). Patient set as ACCEPTED (patient page).
Use case/functionality covered	During the first visit for a study participant, CCM edits a patient's demographics. • Login in the Dashboard as CCM test_clinician. • Select Patient ID 2 (edit). • Select Demographic tab. • Add a new assessment. • Select patient history tab. • Tick History of substance abuse. • Check if warning popup appears (the patient should be excluded from the project) and select “do not exclude”. • Untick History of substance abuse and tick History of brain injury. • Check if warning popup appears (the patient should be excluded from the project) and select “do not exclude”. • Untick History of brain injury. • Edit questionnaire MoCA (at least 26 out of 30 points to be scored). • Edit questionnaire Dexterity (with positive outcome). • Check if warning popup appears (the patient should be excluded from the project) and select “do not exclude”.
Expected result	Warning correctly appears.

Baseline assessment

Id	SMBS2_TC4
Description	Baseline assessment: Editing questionnaires.
Precondition	CCM profile and Patient profile already registered.
Use case/functionality covered	During the first visit for a study participant, CCM compiles the questionnaires. • Login in the Dashboard as CCM test_clinician. • Select Patient ID 2 (edit). • Select the Demographic tab.

	• Add a new assessment. • Select patient history tab. • Tick all options but History of substance abuse and History of brain injury. • Fill the “life habits” section. • Fill the “Physical Examinations” section. • Register a new item in the “Medications” section. • Register a new item in the “Diet Supplements” section. • Edit questionnaire Patient Specific Functional Scale (PSFP, only in Smart4Health). • Edit questionnaire EQ-5D-5L. • Edit questionnaire Mini Nutritional Assessment (MNA). • Edit questionnaire Geriatric Depression Scale (GDS). • Edit questionnaire Instrumental Activities of Daily Living (IADL). • Edit questionnaire Rapid Geriatric Assessment (RGA). • Edit questionnaire Godin Leisure Time Exercise. • Edit questionnaire Numeric Pain Rating Scale (only in Smart4Health). • Edit questionnaire Global Perceived Effect (only in Smart4Health). • Edit questionnaire Mobile Device Proficiency Questionnaire (MDPQ). • Select the Hearing Loss TAB and fulfil all attributes. • Select the Balance Disorders TAB and fulfil all attributes. • Select the Cardiovascular disease TAB and fulfil all attributes. • Select the Cognitive Disorders TAB and fulfil all attributes. • Select the Frailty TAB and fulfil all attributes.
Expected result	Baseline assessment completed.

Kit preparation and distribution

The present scenario simulates the preparation of a devices kit and its delivery to a study participant, and consists in a set of test cases; one for each of the following devices:

- SB@SmartWatch (Garmin).
- SB@BloodPressure (OMRON).
- SB@Weight (iHealth).
- SB@Temperature (iHealth).
- SB@Oximeter (iHealth).
- Hearing aids.
- OpenHAB Devices.

For each device the following steps are tested:

- Devices pairing with APP.
- Checking correct allocation of the device on patient file on the dashboard.

Smartwatch setup

Id	**SMBS3_TC1**

Description	Smartwatch setup.
Precondition	Tested Phones: • Samsung Galaxy A31 • Samsung Galaxy S10 Android version 11 or above. One UI version 3.1. Gmail account dedicated to the patient (name convention smartBear[x]@gmail.com). Wi-Fi, Bluetooth and Location of the mobile devices are always ON.
Use case/functionality covered	Garmin Connect application installation and setup. • Garmin account creation (using patient dedicated gmail address). • Garmin APP installation on the phone. • Disable the battery optimization option. • Select the appropriate location option (all the time). Smartwatch pairing.
Expected result	Smartwatch paired and present in the Garmin APP.

Hearing aids pairing

Id	SMBS3_TC2
Description	Hearing aids pairing with Smartphone.
Precondition	Tested Phones: • Samsung Galaxy A31. • Samsung Galaxy S10. Android version 11 or above. One UI version 3.1. Wi-Fi, Bluetooth and Location of the mobile devices are always ON. Hearing aids (HAs) already fitted by an audiologist.
Use case/functionality covered	HAs Pairing via Bluetooth configuration app.
Expected result	HAs paired.

SMB APP installation and devices configuration on the smartphone

Id	SMBS3_TC3
Description	SMB APP installation and devices configuration on the smartphone.
Precondition	Tested Phones:

	• Samsung Galaxy A31. • Samsung Galaxy S10. Android version 11 or above. One UI version 3.1. Gmail account dedicated to the patient (name convention smartBear[x]@gmail.com) Wi-Fi, Bluetooth and Location of the mobile devices are always ON. Garmin account created and Garmin smartwatch paired. Withings account created. iHealth thermometer. iHealth oximeter. iHealth Scale: Make sure the participant has used the scale once before entering the "WEIGHT" app.
Use case/functionality covered	SMB installation • SMB APP installation through the APK pointing the staging ENV. • Disable the battery optimization option. • Select the appropriate location option (all the time). • Select "allow" for "Nearby devices permission" (Android 12+). • Select "allow" for "notifications". SMB configuration and device pairing: • Open SMB APP (username=smartbear and password= smartbearDevice_1). • Go to settings. • Give the APP permission to taking pictures and record videos. • Connect Withings account. • Connect Garmin account. • Open Hab. Switch on the "Patient has OpenHAB account" and fill in the corresponding OpenHAB cloud username and OpenHAB cloud password (If patient is provided with a HomeHAB device). • HAs. Switch on the "Patient has hearing aids" (if patient has HAs). • iHealth oximeter setup. Scan the barcode/QR code by tapping on the corresponding button. • iHealth scale setup. Scan the barcode/QR code by tapping on the corresponding button. • iHealth Thermometer: just turn on the device and press the button on the app "SCAN THERMOMETER MAC". • Insert Patient ID (ID 1 for testing purposes). • Insert the IMEI of the smartphone. • Press set fields. • Agree on privacy policy.
Expected result	APP installed and all devices configured.

Device management in the Dashboard

Id	SMBS3_TC4
Description	Device management in the Dashboard.
Precondition	Smartwatch setup completed (SMBS3_TC1). HAs pairing completed (SMBS3_TC2). SMB APP installation and device configuration completed (SMBS3_TC3). CCM "test_clinician" profile already registered. Patient ID 1 profile already created.

Use case/functionality covered	During the first visit for a study participant, CCM edits a patient's demographics.
	• Login in the Dashboard as CCM test_clinician.
	• Selected of Patient ID 1 (edit)
	• Select Device management section.
	  • Create a new device: category Smartphone, ID = IMEI.
	  • Create a new device: category Smartwatch, ID = device code, status active.
	  • Create a new device: category Oximeter, ID = device code, status active.
	  • Create a new device: category Thermometer, ID = device code, status active.

Expected result	Devices correctly inserted.

Monitoring

This scenario simulates study participants under the monitoring phase.

It consists in the following test cases, related to APP functionalities:

- Heart Rate Screens.
- Blood Pressure Screen.
- Sleep Statistics Screen.
- Activity Screen.
- Weight Screen.
- Temperature Screen.
- Oximeter Screen.
- Hearing Aids Screen.
- Light Levels Screen.
- Movement Screen.
- Comfort Screen.
- Medication Screen.
- Diet Screen.
- Rehabilitation Screens.

- Incoming Notifications.

Heart Rate Screens

Id	SMBS4_TC1
Description	Heart rate screens.
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created. Heart rate measurements continuously taken over the last 24h.
Use case/functionality covered	Navigate from Homepage to Heart (red frame 1a) or Balance (red frame 1b) and then to Heart Rate (red frame 2). Check latest measurement. Check history.
Expected result	Latest Measurement available and fitting with Garmin APP. Measurements from history available and fitting with Garmin APP.

Blood Pressure Screen

Id	SMBS4_TC2
Description	Blood Pressure Screen.
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created. Blood pressure measurements taken [..].
Use case/functionality covered	Navigate from Homepage to Heart (red frame 1a) or Balance (red frame 1b) and then to Blood Pressure (red frame 2). Check latest measurement. Check history.
Expected result	Latest Measurement available. Measurements from history available.

Sleep Statistics Screen

Id	SMBS4_TC3
Description	Sleep Statistics Screen.
Precondition	Smartwatch setup completed (SMBS3_TC1).

	SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created. Activity (sleep) measurements continuously taken over the last 24h.
Use case/functionality covered	Navigate from the Homepage to Mood (red frame 1) and then to Sleep Statistics (red frame 2). Check statistics about the sleep related to the night before.
Expected result	Last night statistics available.

Activity Screen

Id	SMBS4_TC4
Description	Daily Steps – Activity.
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created. Activity (step, distance) measurements continuously taken over the last 24h.
Use case/functionality covered	Navigate from Homepage to Heart (red frame 1a) or Mood (red frame 1b) or Balance (red frame 1c) and then to Activity (red frame 2). Check latest measurement. Check history.
Expected result	Latest Measurement available. Measurements from history available.

Weight Screen

Id	SMBS4_TC5
Description	Weight screen.
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created. At least 2 weight measurements taken over the last 48h.
Use case/functionality covered	Navigate from Home page to Heart (red frame 1a) or Balance (red frame 1b) and then to Weight (red frame 2). Check latest measurement. Check history.
Expected result	Latest Measurement available.

	Measurements from history available.

Temperature Screen

Id	SMBS4_TC6
Description	Temperature screen.
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created. At least 2 temperature measurements taken over the last 48h.
Use case/functionality covered	Navigate from Homepage to Heart (red frame 1) and then to Temperature (red frame 2). Check latest measurement. Check history.
Expected result	Latest Measurement available. Measurements from history available.

Oximeter Screen

Id	SMBS4_TC7
Description	Oximeter screen.
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created. At least 2 oximeter measurements taken over the last 48h.
Use case/functionality covered	Navigate from the Homepage to Heart (red frame 1) and then Oximeter (red frame 2). Check latest measurement. Check history.
Expected result	Latest Measurement available. Measurements from history available.

Hearing Aids Screen

Id	SMBS4_TC8
Description	Hearing Aids screen.
Precondition	Smartwatch setup completed (SMBS3_TC1).

	HAs pairing completed (SMBS3_TC2). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created. At least 2 HAs measurements taken over the last 48h.
Use case/functionality covered	Navigate to "HEARING" (red frame 1) and then to "HA USAGE" (red frame 2) and see the current program and volume of your hearing aids. Change the program by tapping on "P1". Change the volume by dragging the slider to value 3. Check if HAs are responding. Then roll back. Check latest measurement Check history
Expected result	HAs responded regularly. Latest Measurement available. Measurements from history available.

Light Levels Screen

Id	SMBS4_TC9
Description	Light Levels Screen.
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created. At least 2 Light measurements taken over the last 48h.
Use case/functionality covered	Navigate from Home page to Mood (red frame 1a) or Memory (red frame 1b) and then to Light Levels (red frame 2). Update the Brightness to the higher level by dragging the scroll bar. Tap the "UPDATE" button. Check the light behaviour.
Expected result	Brightness set to its maximum.

Movement Screen

Id	SMBS4_TC10
Description	Movement Screen.
Precondition	Smartwatch setup completed (SMBS3_TC1).

	SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created. At least 2 movement measurements taken over the last 48h.
Use case/functionality covered	Movement: Navigate from Home to Mood (red frame 1) and then to Movement (red frame 2). Check last movement time.
Expected result	Movement time correctly reported.

Comfort Screen

Id	SMBS4_TC11
Description	Comfort Screen.
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created. At least 2 comfort measurements taken over the last 48h.
Use case/functionality covered	Navigate from Home page to Memory (red frame 1) and then to Comfort (red frame 2). Check humidity and temperature.
Expected result	humidity and temperature correctly reported.

Medication Screen

Id	SMBS4_TC12
Description	Medication.
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created. Medication of “sodium fluoride” correctly set up in the dashboard for patient 2, with frequency “twice a day”.
Use case/functionality covered	Click on the medication button. Look for “sodium fluoride”. Check the progress in the day is zero. Add one dosage. Check the progress in the day is 50%. Check notifications (30 min before medication time, 30 min after medication time).

Expected result	initial progress = 0, updated progress = 50%. notification correctly working.

Diet Screen

Id	SMBS4_TC13
Description	Diet screen on the APP.
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created.
Use case/functionality covered	Press the Diet button on the APP. State the level of physical activity: medium. Fill the questionnaire with eating habits.
Expected result	Eating habits correctly inserted.

Rehabilitation Screens

Id	SMBS4_TC14
Description	Rehabilitation screens.
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID=2.
Use case/functionality covered	Open SMB APP. • Scenario 1 (Exercise not allowed): Answer “YES” to any of the first 3 questions or state a high level of unsteadiness (8-10) Check if a dialog appears with “exercise not allowed”. • Scenario 2 (Exercise with caution): Answer “NO” to all of the first 3 questions, but state a medium level of unsteadiness (6-7) Check if a “warning message” appears before moving forward with the exercises. • Scenario 3 (move to next level): Answer “NO” to all of the first 3 questions and state a low level of unsteadiness (0-2). Answer yes to the ability to perform exercises. Check if end user is set to the higher level. Check all the notification categories available.
Expected result	See above.

Measurements page (Dashboard, CCM)

Id	SMBS4_TC15
Description	Measurements page (Dashboard).
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID=2. Heart rate measurements already taken and tested (SMBS4_TC1).
Use case/functionality covered	Patient ID 2 is under monitoring. • Login in the Dashboard as CCM test_clinician. • Select Patient ID 2 (edit). • Select in the overview, select clinical data TAB. • Check heart rate measurements.
Expected result	Measurements correctly reported in the dashboard.

Measurements page (Dashboard, HD operator/tech support)

Id	SMBS4_TC16
Description	Measurements page (Dashboard, HD operator/tech support).
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID=2. Heart rate measurements already taken and tested (SMBS4_TC1).
Use case/functionality covered	Patient ID 2 is under monitoring • Login in the Dashboard as CCM test_clinician. • Select TAB tech support. • Select Patient ID 2 (edit). • Select smart watch device. • Check heart rate measurements.
Expected result	Measurements correctly reported in the dashboard.

Analytics management

This scenario is aimed to test analytics functionalities provided by BDA.

On the dashboard, the Notebooks allows to create analytics by analysing aggregated input data from all the patients or a defined group of them, then input data can be updated anytime.

Notebook creation

Id	SMBS5_TC1

Description	Notebook creation on Dashboard.
Precondition	CCM profile and Patient profile already registered. Patient data fulfilled (demographics). Patient set as ACCEPTED (patient page).
Use case/functionality covered	• Login in the Dashboard as CCM test_clinician. • Select Patient ID 1 (edit). • Select Notebook page. • Create a new notebook, inserting name and description.
Expected result	Notebook created.

Showing analytics in the BDA

Id	SMBS5_TC2
Description	Showing analytics in the BDA.
Precondition	SMBS5_TC1.
Use case/functionality covered	A user profiled as CCM connects to the dashboard, and reach the notebook functionality: • an analytic is selected from the catalogue. • the selected analytic is run. Results are showed.
Expected result	Analytic results showed.

Interventions creation and delivery

This scenario simulates the generation and distribution of an intervention for a specific user starting from a measurement.

Rule-based interventions creation from dashboard

Id	SMBS6_TC1
Description	Rule-based interventions creation for Cardiovascular diseases.
Precondition	CCM test_clinician existing and profiled. Patient ID = 2 existing and profiled.
Use case/functionality covered	• Login with CCM test_clinician. • Select patient ID 2. • Click on the +INTERVENTION button in the Interventions tab. • Confirm predefined values for Extreme Low Value, Extreme High Value, Optimal Low Value, Optimal High Value, Threshold value and type the value that triggers the Intervention under Systolic Blood Pressure. • Follow the previous step for Diastolic Blood Pressure values.

	Save the data.
Expected result	Intervention created.

Intervention creation and delivery (based on DSS rules)

Id	SMBS6_TC2
Description	Generation of an intervention for a specific user and his. CCM, based on generated data and pre-set DSS criteria.
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created. Patient assessment reports CVD (CVD tab is ON), Hypertension and Heart Failure with reduced Ejection Fraction subjects (scompenso cardiaco rEF). Blood pressure device paired.
Use case/functionality covered	The DSS triggers the intervention based on the following standard criteria. Scenario **Heart Failure with reduced Ejection Fraction subjects.** **HF5. Extreme BP values (SBP, DBP too high, SBP or too low)** Patient ID 2 generates the following measurements • SystolicBP = 200mmHg • DiastolicBP = 130mmHg DSS generates a request to repeat the measurement (within 10 minutes). Within 5 minutes, Patient ID 2 generates again the following measurements: • SystolicBP = 200mmHg • DiastolicBP = 130mmHg DSS generates a RED ALERT. The platform advises the user to seek for medical review. The user has to acknowledge the alert to set it off. From the APP: • The “Manual Sync” button (blue frame) syncs with all the devices. • Tap on the bell icon or “Notifications” in the 3 dots menu (red frames) to see your notifications. From the Dashboard • Login in the Dashboard as CCM test_clinician • Select Patient ID 2 (edit) Select Inbox tab to see your notifications.
Expected result	Request for repetition of a measurement, and RED Alerts received by the APP and by the dashboard.

Similar test case can be activated for **Heart Failure with preserved Ejection Fraction subjects (scompenso cardiaco pEF): HpF5. Extreme BP values (too high, or too low)**

Same measurements are needed: SystolicBP = 200mmHg, DiastolicBP = 130mmHg

Intervention creation and delivery (based on APP rules)

Id	SMBS6_TC3
Description	Intervention creation and delivery (based on APP rules).
Precondition	Smartwatch setup completed (SMBS3_TC1). HAs pairing completed (SMBS3_TC2). SMB APP installation and device configuration completed (SMBS3_TC3) Patient ID=2. Precondition for the generation of a notification = inactivity.
Use case/functionality covered	Open SMB APP after 10 days of inactivity. Tap on the bell icon or “Notifications” in the 3 dots menu (red frames) to see notifications.
Expected result	Inactivity alerts shown.

Interventions notification visibility

Id	SMBS6_TC4
Description	Interventions notification visibility. Pilots can configure whether an intervention should be notified or not.
Precondition	Smartwatch setup completed (SMBS3_TC1). HAs pairing completed (SMBS3_TC2). SMB APP installation and device configuration completed (SMBS3_TC3) Patient ID=2. Test case SMBS6_TC2 passed.
Use case/functionality covered	• Set the configuration for the testing pilot: notifications are allowed for CVD. • Run SMBS6_TC2. • Check if notifications are visible. • Set the configuration for the testing pilot: notifications are NOT allowed for CVD. • Run SMBS6_TC2 Check if notifications are filtered.
Expected result	First round: notification is shown. Second round: notification is not shown.

Synergies

This scenario is aimed to test interoperability with Smart4Health and *Holobalance* projects.

Smart4Health synergy test

Id	SMBS7_TC1
Description	*Smart4Health* synergy test.
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created SMBS2_TC1, SMBS2_TC2, SMBS2_TC3, SMBS2_TC4.
Use case/functionality covered	• A user profiled as CCM logins on the dashboard. • A study participant for both SMB and S4H project is chosen. • In the ID management tab, the Smart4Health ID is set. • Training with MedX machine is performed. • On the Clinical Data tab, data collected from S4H is visualized.
Expected result	Visualization of data collected from S4H.

Holobalance synergy test

Id	SMBS7_TC2
Description	*Holobalance* synergy test.
Precondition	Smartwatch setup completed (SMBS3_TC1). SMB APP installation and device configuration completed (SMBS3_TC3). Patient ID 2 profile already created SMBS2_TC1, SMBS2_TC2, SMBS2_TC3, SMBS2_TC4.
Use case/functionality covered	• A user profiled as CCM logins on the dashboard. • A study participant for both SMB and Holobalance project is chosen. • In the ID management tab, the Holobalance ID is set. • Training with Holobox machine is performed. • On the Clinical Data tab, data collected from Holobalance is visualized.
Expected result	Visualization of data collected from Holobalance.