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"

D6.3 (D28) - Integrated SMART BEAR platform v2

Due date of deliverable: 28/02/2023
Actual submission date: 21/02/2023

Grant agreement number: 857172

Lead contractor: CNR

Start date of project: 01/09/2019

Duration: 60 months

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

D6.3 (D28) - Integrated SMART BEAR platform v2

Editors

Alberto Acebes (ATOS)

Manuel M Perez Perez (ATOS)

Contributors

Sotiria Antaraninan (ATC)

Konstantinos Bromis (ICCS)

Fanis Kalatzis (UOI)

Sotiria Antaraninan (ATC)

Jonatan Maggesi (UMIL)

Ioannis Vezakis (STS)

Reviewers

Ioannis Kouris (ICCS)

Claudiu Petrescu (ITSS)

The work described in this document has been conducted within the project SMART BEAR, started in September 2019. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857172

©Copyright by the SMART BEAR Consortium.

Executive Summary

This deliverable details the functionalities, components and tools used for builing the Smart Bear Platform in its version V2.

Contents

Executive Summary **3**

List of acronyms **5**

List of tables **6**

List of figures **7**

1 Introduction **8**

1.1 Overview 8

1.2 Goals of the deliverable 8

1.3 Kubernetes 9

2 Migration to Kubernetes **11**

2.1 Upgrade and maintenance of Kubernetes cluster 11

2.2 New staging environment 13

2.3 Backup's strategy with VolumeSnapshot 15

2.3.1 Install the NFS CSI Driver 16

2.3.2 Volume Snapshot CDRs 18

2.3.3 Volume Snapshot controller 19

2.3.4 Smartbear platform PV Snapshot backup 19

2.3.5 Restoring a PV from a Volume Snapshot 20

2.4 Temporally access to MongoDB 20

2.4.1 Via Port Forwarding 20

2.4.2 Temporal NodePort service 21

2.5 Components 22

2.5.1 Clinical Data Repository (CDR) 23

2.5.2 SecureManagerOne 25

2.5.3 User Manager 29

2.5.4 Decision Support System (DSS) 30

2.5.5 SB Notifications 32

2.5.6 Android App Backend 34

2.5.7 SB Dashboard 37

2.5.8 Big Data Analytics Engine (BDA) 38

2.5.9 OpenHAB Cloud 46

3 GitLab CI/CD **50**

3.1 The .gitlab-ci.yml file 50

3.2 Deployment via yaml kubernetes resources descriptors 50

3.3 Deployment via Helm Charts installers 51

4 Concluding Remarks **53**

List of acronyms

Acronym	Description
CDR	Clinical Data Repository
CRD	Kubernetes Custom Resource Definition
CSI	Container Storage Interface
DSS	Decision Support System
NFS	Network File System
PoP	Pilot Of Pilots

List of tables

Table 1: Components generating containers/pods in Kubernetes and configuration resources 22
Table 2: Configuration resources ‘smartbear-cdr-database’ component 23
Table 3: Configuration resources ‘smartbear-cdr’ component 24
Table 4: Configuration resources ‘smartbear-securemanagerone-database’ component 27
Table 5: Configuration resources ''smartbear-securemanagerone-am’ component.....27
Table 6: Configuration resources ‘smartbear-securemanagerone-is’ component 28
Table 7: Configuration resources ‘smartbear -usermanager’ component.....29
Table 8: Configuration resources ‘smartbear-dds-database’ component 30
Table 9: Configuration resources ‘dds_core’ component 31
Table 10: Configuration resources ‘non_fhir_backend/dds-api’ component 31
Table 11: Configuration resources ‘sb-notifications-database’ component 32
Table 12: Configuration resources ‘sb-notifications’ component 34
Table 13: Configuration resources ‘android-app-backend-database’ component 35
Table 14: Configuration resources ‘android-app-backend’ component 36
Table 15: Configuration resources ‘sb-dashboard’ component 38
Table 16: Configuration resources ‘bda-api’ component 40
Table 17: Configuration resources ‘bda-airflow’ component.....41
Table 18: Configuration resources ‘bda-hadoop’ component 41
Table 19: Configuration resources ‘bda-hive-metastore’ component 42
Table 20: Configuration resources bda-spark component 43
Table 21: Configuration resources ‘bda-trino’ component 45
Table 22: Configuration resources ‘bda-zeppelin’ component 46
Table 23: Configuration resources ‘openhabcloud-database’ component.....48
Table 24: Configuration resources ‘openhabcloud’ component.....49

List of figures

Figure 1: Firewall status for master and worker nodes 13
Figure 2: Kubernetes SMARTBEAR Environments 14
Figure 3: Kubernetes NFS CSI Driver 16
Figure 4: MongoDB Compass client connexion via Port Forwarding 21
Figure 5: Kubenretes Cluster requests access 26
Figure 6: MongoDB initialization 33
Figure 7: OpenHAB Cloud Functional Architecture 47

1 Introduction

This document form part of the set of deliverables describing the different versions of the SMARTBEAR 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. *(this document, to be delivered on month 42)***
- D6.4 Report on integrated SMART BEAR platform v2.*(in parallel, to be delivered on month 42)*
- D6.5 Integrated SMART BEAR platform v3. *(to be delivered on month 52)*
- D6.6 Report on integrated SMART BEAR platform v3. *(to be delivered on month 52)*

1.1 Overview

The last previous deliverables submitted in month 30, described the status of the integration given the decisions taken by the consortium to deal with the urgency of carrying out the Pilot of Pilots (PoP). The consortium thorougly discussed the technical alterantives and their consequences, still postponing the original intentions to carry out the whole platform integration with an environment based on Kubernetes. Therefore, as it was previously reported, the integration was based on Docker compose and Virtual Machine although certain Kubernetes implementations were advanced, were not completed at that time. Therefore, a simplified integration of the components as a more faster but less efficient alternative was carried out. For this first version of the integration we prioritized the implementation of those functionalities agreed with the reviewers for their testing, rather than integration in a more focused on deployment in an actual production environment.

This document describes the advances and details each component requirements and setup preparation for such definitive migration based on Kubernetes. It is worhty to highlight that this migracion to Kubernertes also implies the definitive abandonment of the previous preliminary environments and their maintenance in order to better manage the infrastructure resources and better control of the releases to production in a more procedural way.

1.2 Goals of the deliverable

The implementation of the minimun functionalities and the pilot were both sucessful. However, from a technical point of view, the integration of the components was not carried out under the framework expected by the initial design (in a Kubernetes cluster) but under a simplification due to time restrictions, based on simplified *docker-compose* aggregations in a single-node virtual machine.

The next evolution of the platform thus consists of the effective migration of each and every one of the components to be able to be deployed in this type of more advanced environments, and so it is allowed, in this way, to run the platform in a set of node machines ready for running containerized applications.

No less important is the fact that, in this way, the management of the different environments is now simplified, even adding an intermediate one, the *staging* environment, between development and production, for specific and fixed-versioning testing purposes.

1.3 Kubernetes

Continuing with the situation exposed in the introduction of the precedent deliverable *D6.1 Integrated SMARTBEAR platform v1*, the complete migration of the platform to the Kubernetes cluster has been postponed due to the prioritization of a functional, but preliminary, integration used during the PoP, which has remained a simplified integration based on a virtual machine as hosting running independently each component container.

The initial implementation is kept and it is not abandon yet, for ensuring a smoothly transition to the Kubertnetes based final one. During this period we have progresssed to migrate almost all the componentes, and we implement new ones. Still some technical issues have impaired the complete full integration. Therefore, at the time of writing this deliverable, the migration needs some refinements in order to be considered complete as mitigating this risk we decided not to abaon the previous perliminary environment used for the PoP.

Although some have already been introduced in previous platform version deliverables, for the purpose of understanding the following sections, some Kubernetes basic concepts are listed and briefly described below.

- ***Kubenretes Node***, is the smallest unit of computing hardware (physical or virtual server) in Kubernetes. Can be either of type *master* or *worker*, and in turn can be destined to one or several roles: “controlplane” that manages the executions, “etcd” a distributed key-value store, “worker” for hosting the pure processing demands.
- ***Kubernetes cluster***, is the set of Kubernetes nodes and other infrastructure needed resources to run your containerized applications. A cluster will require, at least, one instance acting with the previous listed roles, but could easily scalate horizontally if the load and available resources requires it.

And also, the following Kubernetes basic used *resource types:*

- ***Pod***, is the smallest deployable unit of computing that could be created in Kubernetes and consist on a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers.
- ***ConfigMap***, API object used to store non-confidential data in key-value pairs that can be consumed by the *Pods* as environment variables, command-line arguments, or as configuration files in a volume, allowing to decouple environment-specific configuration from container images for easy portability, and in this way minimize the risk of this data being exposed during the workflow of creating, viewing, and editing Pods.
- ***Deployment***, is the declarative information (and updates) which describes the desired state for *Pods* and *ReplicaSets*.
- ***Service***, is an abstract way to expose an application running on a set of Pods as a network service without the needed of using other service discovery mechanism, since Kubernetes gives Pods their own IP addresses and a single DNS name for a set of Pods allowing load-balancing across them.
- ***Ingress***, is the API object that manages external access to the services in a cluster, typically HTTP, providing load balancing, SSL termination and name-based virtual hosting, and ***Ingress Controllers***, is the process running that manages the ingresses defined, to root them to the proper services, handling certificates etc.

- ***Storage Classes,*** is the definition mechanism that provides administrators the means to describe different types of storage offered in the cluster oriented to different quality-of-service levels. For example, destinations with low storage capacity but high speed and other slower ones but for massive storage better for backup policies could be defined, according to the needs of the applications, to optimize the available resources.
- ***Persistent Volume Claim (PVC)***, is basically a request for storage, of a specific kind waiting for a *Storage Class* to take care of it, so will be available for the deployments to be used to persist data.
- ***Persistent Volume (PV)***, is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes.
- ***Custom Resources,*** are extensions of the Kubernetes API that are not necessarily available in a default Kubernetes installation and represents a customization of a particular Kubernetes installation. In the SMARTBEAR cluster, the network plugin or the certificates manager install their own *Custom Resource Definitions* (CRD).

2 Migration to Kubernetes

As previously introduced, the use of the Kubernetes framework in containerized and microservices-oriented applications provides great advantages for the continuous integration and management of the SMARTBEAR platform, which are formed based on the union of different components, which are even implemented separately, isolated by different partners.

The deployment in this type of framework allows the management and scaling of the necessary resources in a simplified way, and, together with the application of CI/CD practices, allows a controlled evolution of the development and release of new platform's functionalities, or bug fixes.

2.1 Upgrade and maintenance of Kubernetes cluster

The version of the Kubernetes cluster installation has been also upgraded[1] recently to the latest released versions, in order to keep it as secured as possible as it includes the latest functionalities and optimizations.

Meanwhile the migration of all and each of the components has been implemented, the nodes on which the cluster is based have been kept up to date and their stability monitored.

The framework includes commands[2] to facilitate such upgrades, like the following

```
// Kubernetes upgrade plan summary
$ kubeadm upgrade plan
```

which gets as result the summary of versions capable to be upgraded and following the instructions

```
// Unlock, install last version and lock again, and perform upgrade
$ apt-mark unhold kubeadm kubectl && apt-get update && apt-get install -y kubeadm=1.26.0-00
kubectl=1.26.0-00 && apt-mark hold kubeadm kubectl
$ kubeadm upgrade apply v1.26.0
```

and after the upgrade, then we can go to each worker node to upgrade remaining framework versions

```
// Unlock kubectl, install last version and lock again
$ apt-mark unhold kubelet && apt-get update && apt-get install -y kubelet=1.26.0-00 && apt-mark hold
kubelet
```

although a satisfactory message is displayed as a result, the truth is that after the process *kubelet* was not able to restart complaining with a message "*CRI v1 runtime API is not implemented*", so is needed *containerd*[3] 1.6+ to be updated, and it was not trivial because the package was not found in the update commands

```
// Install containerd 1.6+ and restart services
$ sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
echo \
```

[1] Kubenertes release v1.26 (*https://kubernetes.io/blog/2022/12/09/kubernetes-v1-26-release*)
[2] Kubenertes upgrade util (*hhttps://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade*)
[3] containerd. An industry-standard container runtime with an emphasis on simplicity, robustness and portability https://containerd.io/upgrade util (*https://containerd.io*)

```
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install containerd.io
containerd config default | sudo tee /etc/containerd/config.toml
systemctl restart containerd
systemctl restart kubectlt
```

Keeping the cluster up to date has also required updating the versions of other basic utilities to the latest version available at the time of writing this document, such as:

- *Prometheus*, which allows the monitoring of the cluster, that it has been updated to the latest version available at the time of writing this document.
- *Cert-manager* (to v1.11.1[4]), to automate the Let's Encrypt certs with the latest bug fixings.

Outside the cluster but in relation to the node where it is hosted, the installation of the HAProxy utility has also been prepared to occasionally allow the mapping of ports that allow access to MongoDB type databases during data migration, such as detailed later in section *2.4 Temporally access to MongoDB*, and the required port (27014) open in the firewall, as listed in *Figure 1*.

[4] Cert Manager Helm from *jetstack/cert-manager* Chart *(https://cert-manager.io/docs/installation/helm/)*

```
root@smartbear-k8s-node-0:/home/acebes# ufw status
Status: active

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere
2376/tcp                   ALLOW       Anywhere
2379:2380/tcp              ALLOW       Anywhere
10250/tcp                  ALLOW       Anywhere
32000:32767/tcp            ALLOW       Anywhere
179/tcp                    ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
2376/tcp (v6)              ALLOW       Anywhere (v6)
2379:2380/tcp (v6)         ALLOW       Anywhere (v6)
10250/tcp (v6)             ALLOW       Anywhere (v6)
32000:32767/tcp (v6)       ALLOW       Anywhere (v6)
179/tcp (v6)               ALLOW       Anywhere (v6)
```

```
root@smartbear-k8s-node-1:/home/acebes# ufw status
Status: active

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere
2376/tcp                   ALLOW       Anywhere
2379:2380/tcp              ALLOW       Anywhere
10250/tcp                  ALLOW       Anywhere
32000:32767/tcp            ALLOW       Anywhere
179/tcp                    ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
2376/tcp (v6)              ALLOW       Anywhere (v6)
2379:2380/tcp (v6)         ALLOW       Anywhere (v6)
10250/tcp (v6)             ALLOW       Anywhere (v6)
32000:32767/tcp (v6)       ALLOW       Anywhere (v6)
179/tcp (v6)               ALLOW       Anywhere (v6)
```

```
root@smartbear-k8s-master:/home/acebes# ufw status
Status: active

To                         Action      From
--                         ------      ----
1194/udp                   ALLOW       Anywhere
22/tcp                     ALLOW       Anywhere
80/tcp                     ALLOW       Anywhere
443/tcp                    ALLOW       Anywhere
2376/tcp                   ALLOW       Anywhere
2379:2380/tcp              ALLOW       Anywhere
6443/tcp                   ALLOW       Anywhere
10250/tcp                  ALLOW       Anywhere
10257/tcp                  ALLOW       Anywhere
10259/tcp                  ALLOW       Anywhere
32000:32767/tcp            ALLOW       Anywhere
179/tcp                    ALLOW       Anywhere
2049/tcp                   ALLOW       Anywhere
111/tcp                    ALLOW       Anywhere
20048/tcp                  ALLOW       Anywhere
27017/tcp                  ALLOW       Anywhere
1194/udp (v6)              ALLOW       Anywhere (v6)
22/tcp (v6)                ALLOW       Anywhere (v6)
80/tcp (v6)                ALLOW       Anywhere (v6)
443/tcp (v6)               ALLOW       Anywhere (v6)
2376/tcp (v6)              ALLOW       Anywhere (v6)
2379:2380/tcp (v6)         ALLOW       Anywhere (v6)
6443/tcp (v6)              ALLOW       Anywhere (v6)
10250/tcp (v6)             ALLOW       Anywhere (v6)
10257/tcp (v6)             ALLOW       Anywhere (v6)
10259/tcp (v6)             ALLOW       Anywhere (v6)
32000:32767/tcp (v6)       ALLOW       Anywhere (v6)
179/tcp (v6)               ALLOW       Anywhere (v6)
2049/tcp (v6)              ALLOW       Anywhere (v6)
111/tcp (v6)               ALLOW       Anywhere (v6)
20048/tcp (v6)             ALLOW       Anywhere (v6)
27017/tcp (v6)             ALLOW       Anywhere (v6)
```

Figure 1: Firewall status for master and worker nodes

2.2 New staging environment

The need to create a pre-production or *staging* type environment, promoted from the development environment, where to test the functionalities in an environment isolated from the CI/CD automated deployments, was required. The functionality of the concept of *namespaces* in the scope of the Kubernetes clusters, provides a convenient mean for this purpose, taking advantage of all the configuration and resources available and isolating the deployment of the needed components sharing the cluster with the development environment.

The creation of the *staging* environment is motivated by the need to test the integration of functionalities in release candidate (RC) versions of the components, without the interference that could cause deployments to a development environment of features that misalign the set of desirable versions to test and verify, for their promotion to the production environment.

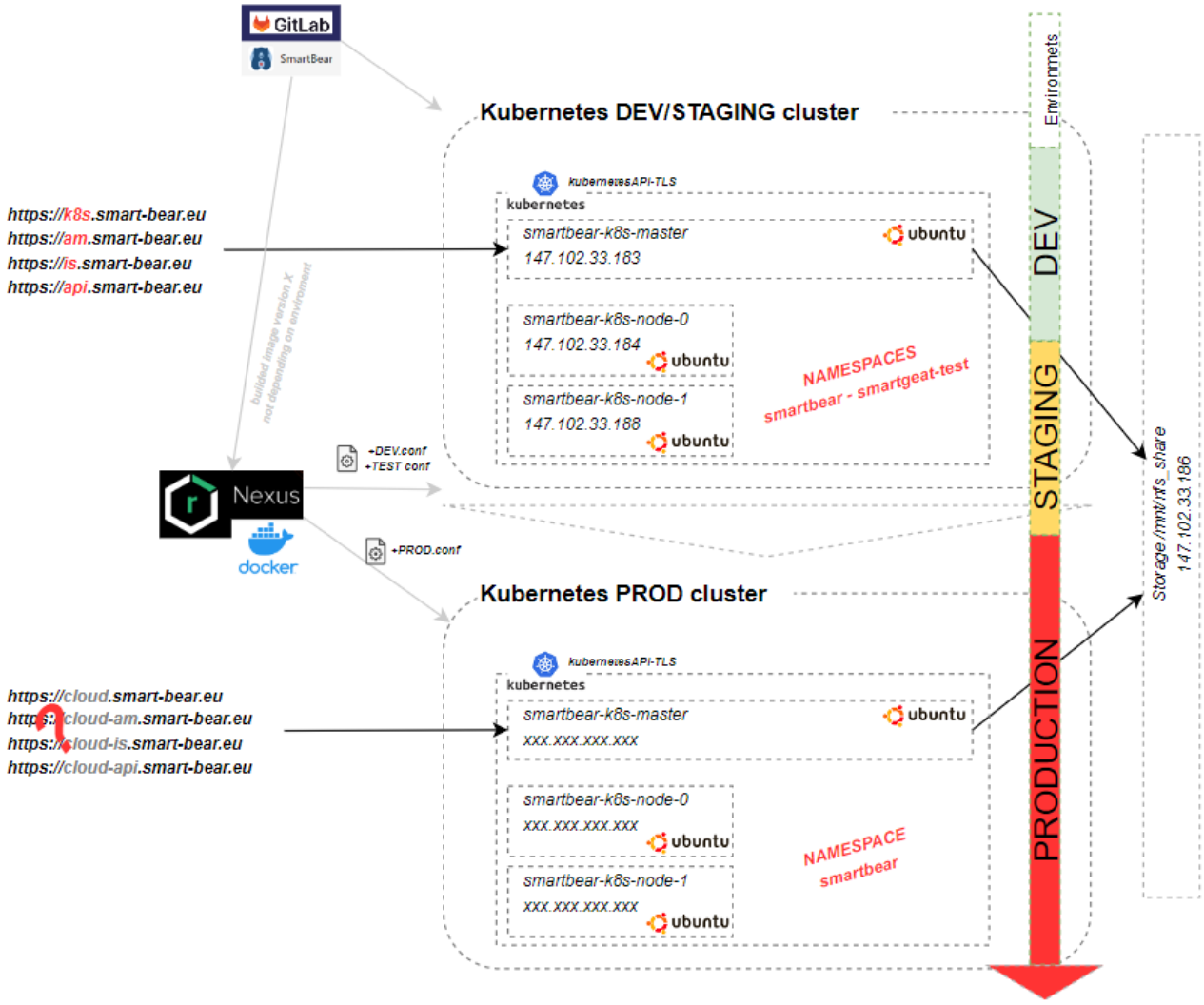

Figure 2: Kubernetes SMARTBEAR Environments

In previous stages, when defining the use of Kubernetes as integration framework, the decision of not including the production environment in the same cluster as the development (from now on also staging) was taken. This is basically to limit the risk that any incident produced in preliminary environments would affect the production environment, which must be the most stable as possible, and configured as high availability. Ingress controller deployed in master node and shared for develop and staging only, and taking care of balancing of requests depending on the subdomains.

2.3 Backup's strategy with VolumeSnapshot

The main Kubertenes strategy to provide persistence is based in the concepts of *Persistence Volume Claim* (PVC) and *Persistence Volume* (PV) as was already described in previous deliverables. This type of resources in turn are based on a configured resource of type *StorageClass*, also explained earlier, in charge of carrying out the process of saving of the data to be persisted. In the case of the SMARTBEAR platform the storage is based on remote file system directories through the Network File System (NFS) protocol with the help of a simple provider (`nfs-subdir-external-provisioner`) that was installed as provisioner[5].

The abstraction of the persistence functionality also makes it necessary, when moving to a production environment, to have a data backup strategy, so copies can be created for potential recovery in the event of an incident involving the loss of the original data. For this purpose, the concepts of *VolumeSnapshot* and *VolumeSnapshotContent*[6] are starting to be used in Kubernetes based environments, a type of resources that represents snapshots of the provisioned volumes on a configured target storage system, different from the original.

The lifecycle of a volume snapshot and volume snapshot content are based in the fact that a *VolumeSnapshotContent* are resources in the cluster and the *VolumeSnapshots* are requests for those resources. The *VolumeSnapshot* can be pre-provisioned by an administrator by carrying the details of the real volume snapshot on the storage system, which is available for use by cluster users, or could be dynamic, as we are configuring for the SMARTBEAT platform, doing it dynamically with just the reference to the *PersistentVolumeClaim*. For his part, the *VolumeSnapshotClass* specifies storage provider-specific parameters to use when taking the snapshot, for example by using simple host folders o remote directories via NFS.

An important behaviour to highlight is the Snapshot Source Protection, which ensure that an *in-use PersistentVolumeClaim* object is not removed from the system while a snapshot is being taken from it, avoiding any data loss, and is postponed until the snapshot is ready-to-use or aborted. The deletion policy of the *VolumeSnapshot* object can be configured either as Delete or Retain, so the underlying storage snapshot along with the *VolumeSnapshotContent* object will be deleted or will remain.

[5] SMARTBEAR D6.1 Integrated SMART BEAR platform v1, section *3.2.5.2.*

[6] Kubernetes VolumeSnapshot (*https://kubernetes.io/docs/concepts/storage/volume-snapshots*)

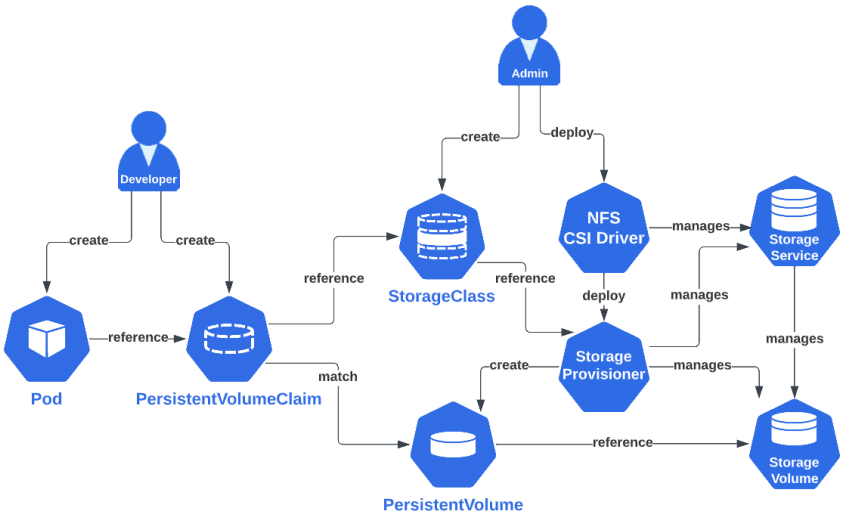

Figure 3: Kubernetes NFS CSI Driver

The described volume snapshot features of binding and mapping one-to-one the *VolumeSnapshot* object with an appropriate *VolumeSnapshotContent* and the current disk operations are implemented with the union of several characteristics that need to be clarified for its correct installation and configuration. Basically needs:

- the CSI Driver and StorageClass (along with CSI Snapshotter sidecar),
- the *VolumeSnapshot*, *VolumeSnapshotContent*, and *VolumeSnapshotClass* API objects which are custom resources (CRDs) not part of the core API, and
- the volume snapshot controller and the snapshot validation webhook.

2.3.1 Install the NFS CSI Driver

The use of this new type of resources requires a more advanced kind of drivers, based in and API called Container Storage Interface (CSI), so the original provisioner needs to be replaced by this new approach[7], and the new references will use the driver identified as `nfs.csi.k8s.io` instead of the previous provider configured. Using this driver is also to advance the support for expanding CSI volumes that, although is enabled by default, it requires this new kind of drivers to support it. The installation[8] of the CSI driver is described below, and can be done via a Helm Chart ready to use:

// NFS CSI driver installation via Helm Chart

```
$ helm repo add csi-driver-nfs https://raw.githubusercontent.com/kubernetes-csi/csi-driver-nfs/master/charts
$ helm install smartbear-csi-driver-nfs csi-driver-nfs/csi-driver-nfs --namespace kube-system --version v4.1.0 --set driver.name="nfs.csi.k8s.io" --set controller.name="csi-nfs-controller" --set feature.enableFSGroupPolicy=true

NAME: smartbear-csi-driver-nfs
LAST DEPLOYED: Wed Feb  8 18:13:03 2023
NAMESPACE: kube-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The CSI NFS Driver is getting deployed to your cluster.
```

[7] Kubernetes NFS CSI D*river* (*https://github.com/kubernetes-csi/csi-driver-nfs*)
[8] Kubernetes NFS CSI D*river* installation (*https://github.com/kubernetes-csi/csi-driver-nfs/blob/master/docs/install-csi-driver-v4.1.0.md#install-nfs-csi-driver-v410-version-on-a-kubernetes-cluster*)

```
To check CSI NFS Driver pods status, please run:
  kubectl --namespace=kube-system get pods --selector="release=smartbear-csi-driver-nfs" --watch
```

In case that Helm is not available, could be also installed via a ready to use *sh* script which aggregate the *yaml* files:

```
// Remote install of the NFS CSI driver

$ curl -skSL https://raw.githubusercontent.com/kubernetes-csi/csi-driver-nfs/v4.1.0/deploy/install-
driver.sh | bash -s v4.1.0 --
```

the installation on the execution via the `kubectl` command of the different descriptor files, for the specific version to be installed.

```
// Downloaded from https://github.com/kubernetes-csi/csi-driver-nfs/tree/v4.1.0/deploy/v4.1.0

$ kubectl apply -f $repo/rbac-csi-nfs.yaml
$ kubectl apply -f $repo/csi-nfs-driverinfo.yaml
$ kubectl apply -f $repo/csi-nfs-controller.yaml
$ kubectl apply -f $repo/csi-nfs-node.yaml
```

or as final option by te use of Helm Chart[9]

```
// Use of Helm Chart for the intallation of CSI driver with Helm

$ helm repo add csi-driver-nfs https://raw.githubusercontent.com/kubernetes-csi/csi-driver-
nfs/master/charts
$ helm install csi-driver-nfs csi-driver-nfs/csi-driver-nfs --namespace kube-system --version v4.1.0
kubectl apply -f $repo/rbac-csi-nfs.yaml
```

Once the pod is deployed, although the driver is available, still is needed the related *StorageClass* to be ready for provision a *PersistenceVolume* when required by the *PersistenceVolumeClaim* resources. For that just is needed.

```
// StorageClass based on NFS CSI driver

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: smartbear-nfs-csi-storageclass
  annotations:
    storageclass.kubernetes.io/is-default-class: 'true'
provisioner: nfs.csi.k8s.io
parameters:
  server: 147.102.33.186
  share: /mnt/nfs_share/smartbear-k8s
  subDir: /${pvc.metadata.name}
#  mountPermissions: 0 # 0 is root
reclaimPolicy: Retain
allowVolumeExpansion: true
volumeBindingMode: Immediate
mountOptions:
  - nfsvers=4.2
```

To select the proper NFS protocol version available to be mounted, the compatibility could be checked by:

```
// NFS client versions available in the system

$ nfsstat -c
$ sudo cat /proc/fs/nfsd/version
```

[9] Kubernetes NFS CSI D*river* install with Chart (*https://github.com/kubernetes-csi/csi-driver-nfs/tree/master/charts*)

The *VolumeSnapshotClass* resource is equivalent to the core native *StorageClass* resource used to provision volumes, but in this case contains the specific features needed to persist *snapshots*.

```
// smartbear-volumesnapshotclass

apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: smartbear-volumesnapshotclass
  annotations:
    snapshot.storage.kubernetes.io/is-default-class: 'true'
driver: hostpath.csi.k8s.io
deletionPolicy: Retain # or Delete
parameters:
  # describes volume snapshots belonging to the volume snapshot class,
  # or others accepted depending on the driver
...
```

The previous process has been collected for automation in the "smartbear" base project (under `/csi-driver-nfs-4.1.0` folder) of the repository, which will speed up the complete implementation of a cluster from scratch.

2.3.2 Volume Snapshot CDRs

As already mentioned, as long as the resource types are not integrated into the Kubernetes core, it will be necessary to include them as CDRs. As a precaution, we first check that still do not exist, in any API version (version 6.2 at the time of writing this document):

```
// Check CDRs do not exist already

$ kubectl api-resources | grep volumesnapshot
```

If not present we can continue with the direct installation of the CDRs based on the desired released[10] version:

```
// CDRs installation

$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-
6.2/client/config/crd/snapshot.storage.k8s.io_volumesnapshotclasses.yaml
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-
6.2/client/config/crd/snapshot.storage.k8s.io_volumesnapshotcontents.yaml
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-snapshotter/release-
6.2/client/config/crd/snapshot.storage.k8s.io_volumesnapshots.yaml
```

and when finished, check that are properly listed along the rest of available resources in the cluster:

```
// Verify CDRs installed

$ kubectl api-resources
NAME                    SHORTNAMES        APIVERSION                    NAMESPACED   KIND
...
volumesnapshotclasses   vsclass,vsclasses snapshot.storage.k8s.io/v1    false        VolumeSnapshotClass
volumesnapshotcontents  vsc,vscs           snapshot.storage.k8s.io/v1    false        VolumeSnapshotContent
volumesnapshots         vs                snapshot.storage.k8s.io/v1    true         VolumeSnapshot
...
```

The previous process has been collected for automation in the "smartbear" base project (under `/volume-snapshot` folder) of the repository, which will speed up the complete implementation of a cluster from scratch.

[10] Kubernetes VolumeSnapshot (*https://kubernetes.io/docs/concepts/storage/volume-snapshots*)

2.3.3 Volume Snapshot controller

For the *VolumeSnapshot* to trigger the creation of the *VolumeSnapshotContent* resources, a controller process, commonly called *snapshotter*[11], should be also installed or the resource creation will not trigger any process. It is the logical behaviour considering that this still functionality not part of the core of Kubernetes.

Several are the processes needed for carrying out this functionality, with the main distinction of the *snapshot-controller* and a *CSI-external-snapshotter* sidecar. The snapshot controller will be watching the Kubernetes API server for *VolumeSnapshot* and *VolumeSnapshotContent* CRD objects, and the second process only watches the Kubernetes API server for *VolumeSnapshotContent* CRD objects, triggering the creation of the snapshot on the storage system:

```
// snapshot controller installation

$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-
snapshotter/v6.2.1/deploy/kubernetes/snapshot-controller/rbac-snapshot-controller.yaml
$ kubectl apply -f https://raw.githubusercontent.com/kubernetes-csi/external-
snapshotter/v6.2.1/deploy/kubernetes/snapshot-controller/setup-snapshot-controller.yaml
```

After the installation, a pod will be listening and ready for dispatching the volume snapshot requests.

The previous process has been collected for automation in the "smartbear" base project (under `/volume-snapshot` folder) of the repository, which will speed up the complete implementation of a cluster from scratch.

2.3.4 Smartbear platform PV Snapshot backup

The SMARTBEAR platform includes some critical components that has PVC and requires backing of their volumes. The remaining PVs are supposed to be of the kind that its data could be regenerated by the components without the need of backups.

As an example, the *VolumeSnapshot* for the PVC requested by the **`smartbear-cdr-deployment`**, which name is **`smartbear-cdr-deployment-pvc`**, could be defined like follows:

```
// smartbear-cdr-database-volumesnapshotclass

apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: smartbear-cdr-database-pv-snapshot
  namespace: smartbear
spec:
#  volumeSnapshotClassName: smartbear-volumesnapshotclass
  source:
    persistentVolumeClaimName: smartbear-cdr-database-pvc
```

The previous process has been collected for automation in the "smartbear" base project (under `/volume-snapshot` folder) of the repository, which will speed up the complete implementation of a cluster from scratch.

[11] Kubernetes snapshotter (*https://kubernetes-csi.github.io/docs/snapshot-controller.html / https://github.com/kubernetes-csi/external-snapshotter*)

2.3.5 Restoring a PV from a Volume Snapshot

In the case of need for restoring and provisioning Volumes from Snapshots, a new volume, pre-populated with data from a snapshot, can be done by using the *dataSource* field in the PersistentVolumeClaim object[12].

```
// Example of restoring PVC from a VolumeSnapshot

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: smartbear-cdr-database-pvc
  namespace: smartbear
spec:
  accessModes:
  - ReadWriteOnce
  dataSource:
    name: smartbear-cdr-database-pv-snapshot
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
  resources:
    requests:
      storage: 10Gi
```

2.4 Temporally access to MongoDB

The migration to the Kubernetes cluster require, for some components, to transfer also the data collected in the MongoDBs either in *development* and, in the near future, the *production* environments. To facilitate this process but also in case of need to inspect the data contained in the DBs, or debug the interaction with it, the service could be exposed to be accesible from outside the cluster and managed with any kind of UI tool, like *MongoDBCompass*[13].

2.4.1 Via Port Forwarding

This method allows an Kubernetes API client to connect to the service via a *proxy* connexion between the service internal port (*Pod Port*) within the cluster, and a random port mapped in the client (*Local Port*).

```
// Creating a Port Forwarding

# kubectl port-forward service/<service-name> [<locahost-port>]:<pod-port> -n namespace &

$ kubectl port-forward service/smartbear-sb-notifications-database-svc 50836:27017 -n smartbear &
```

If the command is accepted, a random port (e.g.:`50836`) is provided and then the client could connect using it in the connection string URI. The following *Figure 4* shows the connexion configuration, via localhost and the port mapped.

[12] Kubernetes VolumeSnapshot CDRs (*https://github.com/kubernetes-csi/external-snapshotter/tree/release-6.2/client/config/crd*)
[13] Compass. The GUI for MongoDB (*https://www.mongodb.com/products/compass*)

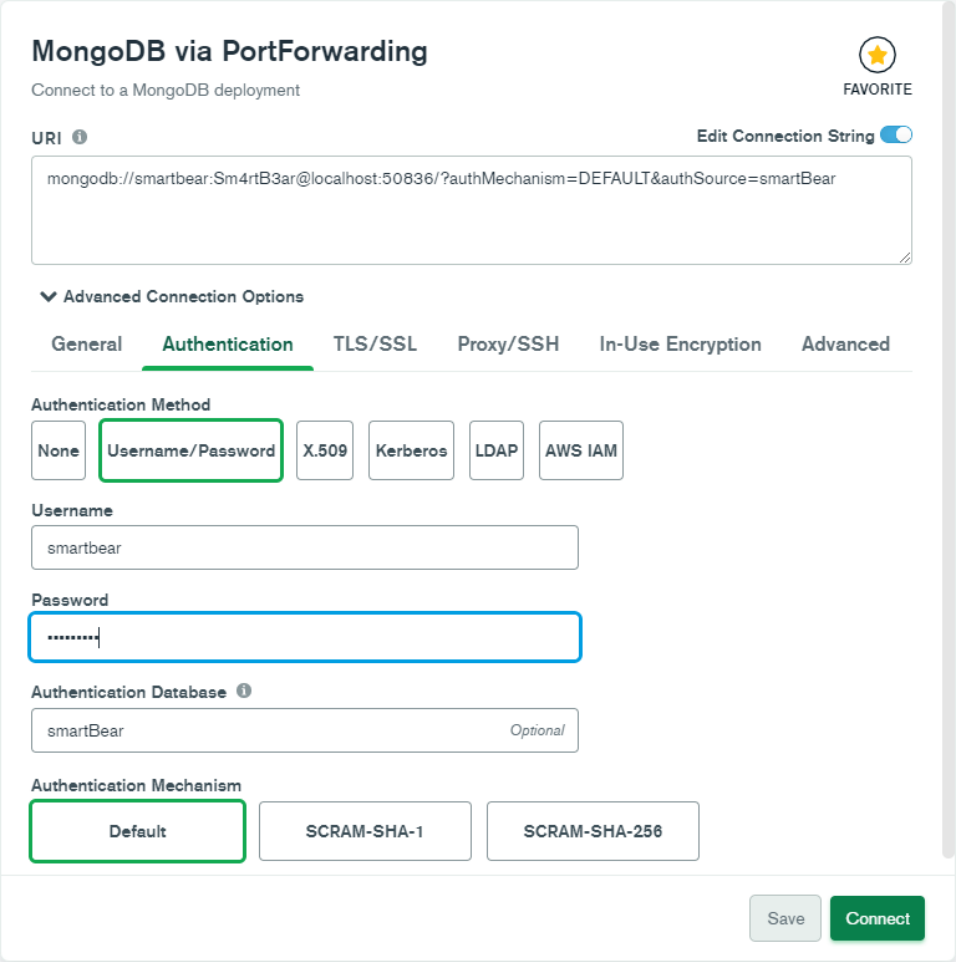

Figure 4: MongoDB Compass client connexion via Port Forwarding

2.4.2 Temporal NodePort service

As alternative to the previous method, it is possible to provide a temporal access by using Kubernetes services of type *NodePort,* which allows to fix a specific port number to the service and, with a combination of firewall and proxy configurations, provide a complete routing from internet to the original service deployed in the cluster.

The file containing the definition for the services is located in the project GitLab repository under the root project `smartbear/mongodb-nodeport-service`, and its use could be applied and removed in the specific moments when needed.

Alongside the creation of the services that exposes the ports, firewall should allow incoming connexion from that ports and HAProxy whould configure the mapping between the external port published and the Kubernetes cluster nodePort specified in the file. With this configuration in place, the MongoDB service could be accessed via the traditional connexion urls, e.g.:

```
// MongoDB connection string to temporally exposed service
mongodb://smartbear:Sm4rtB3ar@k8s.smart-bear.eu:27017/smartBear?retryWrites=true
```

The previous process has been collected for automation in the "smartbear" base project (under `/mongodb-nodeport-service` folder) of the repository, which will speed up the complete implementation of a cluster from scratch.

2.5 Components

The platform deployed in the Kubernetes cluster for this V2 consists of the following components, where some of them have been added since previous integrations, and listed with the expected configuration resources related and described below in the *Table 1*.

Partner	Component	Subcomponent	ConfigMap	Secret	PVC	Helm Chart	Vol. Snapshot
ATOS	Clinical Data Repository (CDR)	smartbear-cdr-database	✓	✓	✓	✗	✓
		smartbear-cdr	✓	✓	✗	✗	✗
ICCS	Decision Support System (DSS)	non_fhir_database	✓	✓	✓	✗	✓
		non_fhir_backend	✓	✓	✗	✗	✗
		dss_core	✓	✓	✗	✗	✗
ATC	SB Notifications	sb-notifications-database	✓	✓	✓	✗	✓
		sb-notifications	✓	✓	✗	✗	✗
	Android App Backend	android-app-backend-database	✓	✓	✓	✗	✓
		android-app-backend	✓	✓	✗	✗	✗
STS	Secure Manager One	smartbear-securemanagerone-database	✓	✓ (1)	✓	✗	✓
		smartbear-securemanagerone-is	✓		✗	✗	✗
		smartbear-securemanagerone-am	✓		✗	✗	✗
	Usermanager	usermanager	✗		✗	✗	✗
UMIL	Dashboard	sb_dashboard	✓	✓	✗	✗	✗
	Big Data Analytics Engine (BDA)	bda-airflow	✓	✓ (1)	✓	✓	✗
		bda-hive-metastore	✓		✓	✗	✗
		bda-hadoop	✓		✓	✗	✗
		bda-spark	✓		✓	✓	✗
		bda-trino	✓		✓	✗	✗
		bda-zeppelin	✓		✓	✗	✗
		bda-api	✓		✗	✗	✗
UOI	OpenHAB Cloud	smartbear-openhab-cloud-database	✓	✓	✓	✗	✓
		smartbear-openhab-cloud	✓	✓	✗	✗	✗

(1) All the subcomponets share the secret resource

Table 1: Components generating containers/pods in Kubernetes and configuration resources

2.5.1 Clinical Data Repository (CDR)

Managed by ATOS, the **CDR** is the component in charge of storing the patient clinical data based in the HL7 FHIR standard and its deployment requires a relational database that supports the data storage (`smartbear-cdr-database`), and the application providing the REST API implementation of the standard (`smartbear-cdr`).

The files supporting the Kubernetes deployment are (`\deployment\kubernetes`):

- `01-smartbear-cdr-database-config.yaml` — *ConfigMap*
- `02-smartbear-cdr-database-deployment.yaml` — *PVC*, *Deployment* and *Service*
- `03-smartbear-cdr-config.yaml` — *ConfigMap*
- `04-smartbear-cdr-deployment.yaml` — *Deployment* and *Service*

2.5.1.1 smartbear-cdr-database

The CDR database is based in a MariaDB[14] relational DB engine and running in an independent container. Its deployment is based in the public Docker image (tagged as `mariadb:10.6.4`.), so there is no need of any building phase for this subcomponent.

Its requirements of configuration are just a *configMap* (`smartbear-cdr-database-config`) containing basic setup for the DB, and a *secret* (`smartbear-cdr-database-secret`) containing the passwords for DB admin and smart-bear DB schema.

Table 2 summarizes the configuration provided.

	Resource	Variable	Purpose
Secret	smartbear-cdr-database-secret	MYSQL_DATABASE	DB schema name (default: `smartbear_hapi`)
		SPRING_DATASOURCE_USERNAME	DB smartbear user (default: `smartbear_hapi`)
Config Map	smartbear-cdr-database-config	MYSQL_ROOT_PASSWORD	DB root user password
		SPRING_DATASOURCE_PASSWORD	DB smartbear user password

Table 2: Configuration resources 'smartbear-cdr-database' component

2.5.1.2 smartbear-cdr

The CDR is a java application based on the open-source reference implementation of the HL7 FHIR standard named HAPI-FHIR[15]. The application is slightly customized, so it is required its construction from the source code hosted in the code repository, and then the generation of the docker image to host it and make it available for the deployment in the Kubernetes cluster.

The requirements of configuration are a *secret* (`smartbear-cdr-secret`) containing the passwords for connecting to the DB smart-bear schema, and a *configMap* (`smartbear-cdr-config`) containing basic setup for the app to boot up.

Table 3 summarizes the configuration provided.

[14] HAPI-FHIR. Complete implementation of HL7 FHIR standard for healthcare interoperability in Java (*https://hapifhir.io*)
[15] MariaDB Server, the open-source relational database (*https://mariadb.org*)

	Resource	Variable	Purpose
Secret	smartbear-cdr-database-secret	*Uses the smartbear-cdr-database already and previously defined secret*	
ConfigMap	smartbear-cdr-config	HAPI_FHIR_SERVER_ADDRESS	Endpoint of the HAPI FHIR server
		SERVER_SERVLET_CONTEXT_PATH	Optional context path (default: o /clinical-data-repository)
		SPRING_DATASOURCE_DRIVERCLASSNAME	DB smartbear user (default: org.mariadb.jdbc.Driver)
		SPRING_DATASOURCE_URL	jdbc:mariadb URI to the DB engine
		SPRING_JPA_PROPERTIES_HIBERNATE_DIALECT	Hibernate dialecr (default: org.hibernate.dialect.MariaDB103Dialect

Table 3: Configuration resources 'smartbear-cdr' component

2.5.2 SecureManagerOne

Managed by STS, the component **SecurityComponent** is the one in charge of the security of the platform: authentication and authorization, protection of access to the different exposed APIs and user (participants) management. To serve these purposes, the component is made up of three different utilities:

- WSO2 Identity Server[16], which provides modern identity and access management,
- WSO2 API Manager[17], which protects the different APIs exposed by the platform (clinical data repository, BDA, DSS, UserManager and Mobile). The component provides an administration UI and a publisher mechanism to evolve versions in a controlled manner.
- UserManager, actually a "participants" management, a component that uses the Security Component's database to store basic data regarding the patients, their assigned clinicians and caregivers, their SMARTBEAR devices and any other external identities (e.g. their corresponding ID in Holobalance). This data is stored in parallel to the information contained in the FHIR and non-FHIR databases.

In addition, the SecureManagerOne is responsible for the pseudoanonymization of all requests that it handles. As the SMARTBEAR's API Gateway, it is always looking for patient IDs in the requests' or responses' payloads. For example, end-users on the dashboard can see a patient id which we refer to as pseudo-id 1. In the backend services however, a different id – pseudo-id 2 – is used to refer to the same patient. To ensure a seamless communication, the Security Component finds the pseudo-id 1 in the request's body, queries, or parameters, and substitutes it with pseudo-id 2 before forwarding it to the backend, and vice-versa. The process is the same with other ids as well, such as a patient's external id assigned for another project (smart4health, holobalance, etc).

The files supporting the Kubernetes deployment are (`\deployment\kubernetes`):

- `01-smartbear-securemanagerone-database-config.yaml` *ConfigMap*
- `02-smartbear-securemanagerone-database-pvc.yaml` *PVC*
- `03-smartbear-securemanagerone-database-deployment.yaml` *Deployment*
- `04-smartbear-securemanagerone-database-svc.yaml` *Service*
- `05-smartbear-securemanagerone-is-config.yaml` *ConfigMap*
- `06-smartbear-securemanagerone-is-deployment.yaml` *Deployment*
- `07-smartbear-securemanagerone-is-svc.yaml` *SVC*
- `08-smartbear-securemanagerone-am-config.yaml` *ConfigMap*
- `09-smartbear-securemanagerone-am-deployment.yaml` *Deployment*
- `10-smartbear-securemanagerone-am-svc.yaml` *Service*
- `11-smartbear-usermanager-config.yaml` *ConfigMap*
- `12-smartbear-usermanager-deployment.yaml` *Deployment*
- `13-smartbear-usermanager-svc.yaml` *SVC*

[16] WOS2 Identity Server (*https://wso2.com/identity-server*)
[17] WOS2 API Manager (*https://wso2.com/api-manager*)

*Figure 5 sh*ows graphically how the access to the cluster from the internet occurs and how it is routed by the API manager.

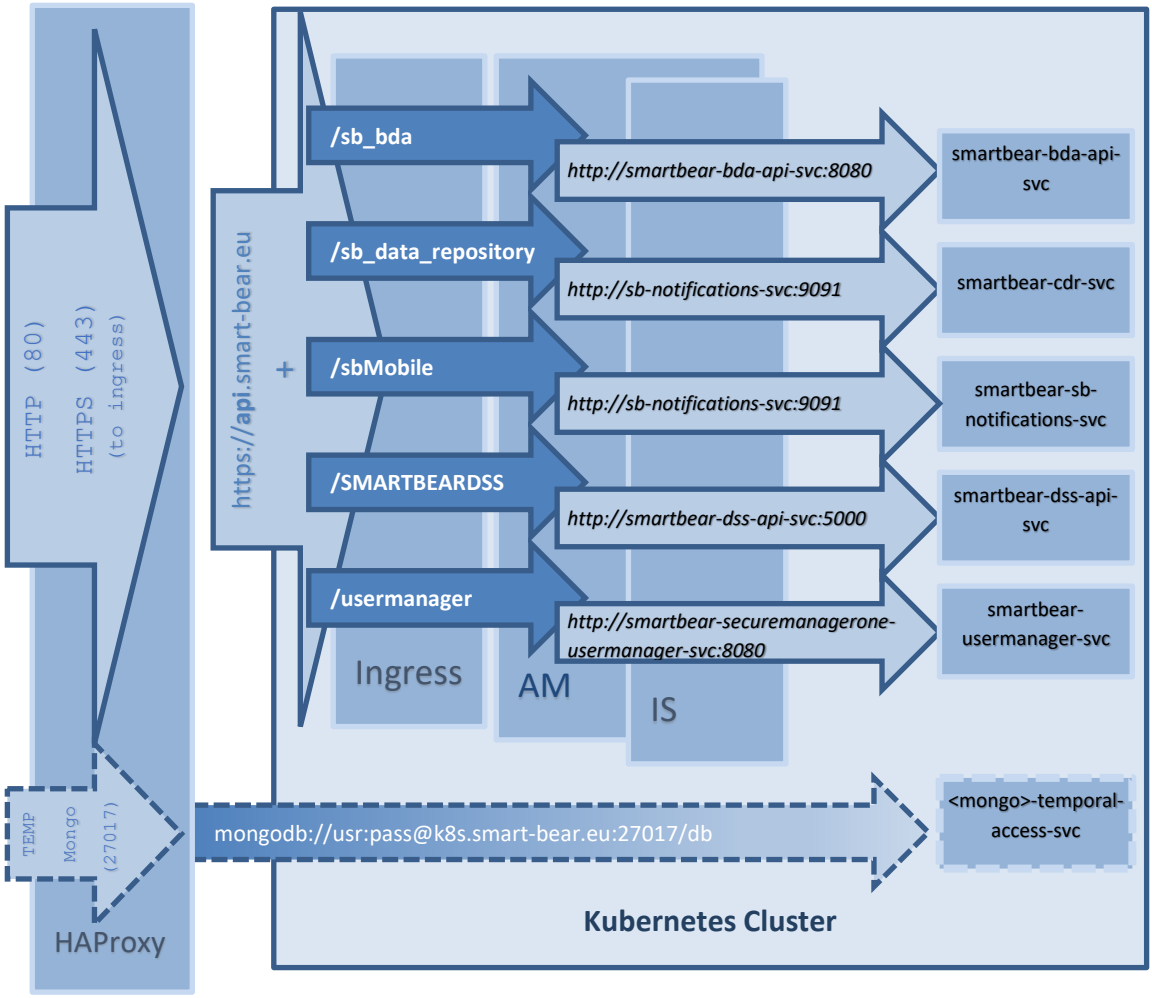

Figure 5: Kubenretes Cluster requests access

2.5.2.1 smartbear-securemanagerone-database

The smartbear securemanagerone database is a component central to the Security Component as it is used by all the other three, i.e. WSO2 Identity Server, WSO2 Application Manager, and UserManager. Without it, these components would not be functional.

Table 4 summarizes the configuration provided.

	Component	Variable	Purpose
Secret	smartbear-securemanagerone-secret	POSTGRES_USER	The to-be-created admin account's username
		POSTGRES_PASSWORD	The to-be-created admin account's password
ConfigMap	smartbear-securemanagerone-database-config	docker_postgres_init.sql	Required by WSO2 to create the database schema upon first start
		postgresql_apim.sql	
		postgresql_shared.sql	

Table 4: Configuration resources 'smartbear-securemanagerone-database' component

2.5.2.2 API Manager (WSO2 AM)

The API Manager (AM) is the component acting as an API Gateway. All requests pass through this component first, where the user's JWT token's validity is checked against the Identity Server, and if valid, the request is pseudoanonymized and forwarded to the appropriate component. It is thus responsible for the API calls routing, pseudoanonymization, and authorization.

Table 5 summarizes the configuration provided.

	Component	Variable	Purpose
Secret	smartbear-securemanagerone-secret	AM_PLATFORM_URL	The URL at which the Application Manager will be available. For example, api.smart-bear.eu
		DB_USERNAME	Credentials to be used to establish a connection to the security component's database.
		DB_PASSWORD	
ConfigMap	smartbear-securemanagerone-am-config	deployment.toml	The Application Manager's deployment configuration, holding all required parameters.

Table 5: Configuration resources ''smartbear-securemanagerone-am' component

2.5.2.3 Identity Server (WSO2 IS)

The Identity Server (IS) is the component is in charge of the authentication. It provides users that are successful in logging in with a JWT token, and also exposes the public key required to check the token's validity.

Table 6 summarizes the configuration provided.

	Component	Variable	Purpose
Secret	smartbear securemanagerone-secret	IS_PLATFORM_URL	The address at which the identity server will be available. This is also necessary to setup the dashboard' redirect to the login page.
		DB_USERNAME	Credentials required to access the database. This is necessary to access and store user details, such as usernames and passwords.
		DB_PASSWORD	
ConfigMap	smartbear-securemanagerone is-config	deployment.toml	The Identity Server's deployment configuration, holding all required parameters.

Table 6: Configuration resources 'smartbear-securemanagerone-is' component

Following technical problems that arose during the production run of WSO2 on the Pilot of Pilots, the two WSO2 components are subject to be replaced with Keycloak for authentication (in place of the Identity Server), and KrakenD (in place of the API Manager), in the near future. In that case, functionality will remain identical.

2.5.3 User Manager

Managed by STS, the **UserManager** component is the application in charge of storing patient details, all the IDs associated with them, and their relations to other clinicians, caregivers, and devices. It depends upon the WSO2 security component and its database as it needs information about created users to assign patients to them. In addition, it is a very important component for the pseudoanonymization functionality, as it is in charge of creating and storing all of the patients' pseudo-ids (including pseudo-id 1, pseudo-id 2, and any external project ids). Written in GoLang, it does not depend on external .sql files, as it generates the appropriate schema at the target database by itself. This is in contrast to WSO2, where the .sql files are provided in the database's config.

The user manager has dependencies with:

- Security Component IS, for user authentication purposes.
- CDR for creating the new patients

The files supporting the Kubernetes deployment are (`\deployment\kubernetes`):

- `01-smartbear-usermanager-config.yaml` *ConfigMap*
- `02-smartbear-usermanager-deployment.yaml` *Deployment, Service* and *Ingress*

Table 7 summarizes the configuration provided.

	Component	Variable	Purpose
Secret	smartbear-securemanagerone-secret	PLATFORM_URL	The dashboard URL
		DB_USERNAME	Credentials to connect to the security component's database
		DB_PASSWORD	
		DB_HOST	Used to construct the database's address, e.g. securitydb:5432
		DB_PORT	
		SB_DATA_REPOSITORY_SVC	The service name of the FHIR repository. This is necessary during creation of a new patient, as the same patient must be created in the FHIR repository as well.
ConfigMap	smartbear-usermanager-config	N/A	

Table 7: Configuration resources 'smartbear -usermanager' component

2.5.4 Decision Support System (DSS)

Managed by ICCS, the **DSS** component maintains the containers characteristics described in the previous version with the enhancement of adding a dependency control with its own database needs, by adding `initContainers` wating for the `smartbear-dss-database` service to be ready. Its three subcomponents (`non_fhir_database`, `dss_core` and `non_fhir_backend/api-dss`) configuration needs are the following.

The files supporting the Kubernetes deployment are (`\deployment\kubernetes`):

- `01-smartbear-dss-database-config.yaml` — *ConfigMap*
- `02-smartbear-dss-database-deployment.yaml` — *PVC*, *Deployment* and *Service*
- `03-smartbear-dss-config.yaml` — *ConfigMap*
- `04-smartbear-dss-deployment.yaml` — *Deployment* and *Service*
- `05-smartbear-dss-api-config.yaml` — *ConfigMap*
- `06-smartbear-dss-api-deployment.yaml` — *Deployment* and *Service*

2.5.4.1 non_fhir_database

In a version alignment with the CDR DB needs, the Non-FHIR DB is also based in a MariaDB based in the public Docker image (tagged as `mariadb:10.6.4`.).

The Non-FHIR DB has dependency with the dss_core, as it is used to log the actions that generate the notifications.

The basic configuration to deploy it is listed in the following detail, with a *secret* (`smartbear-dss-database-secret`) containing the passwords for DB admin and smart-bear DB schema, and a *configMap* (`smartbear-dss-database-config`) containing basic setup for the DB. *Table 8* summarizes the configuration provided.

	Resource	Variable	Purpose
Secret	smartbear-dss-database-secret	MYSQL_DATABASE	DB schema name (e.g: Sm4rtbear)
		SPRING_DATASOURCE_USERNAME	DB smartbear user (e.g.:: Sm4rtbear)
Config Map	smartbear-dss-database-config	MYSQL_ROOT_PASSWORD	DB root user password
		SPRING_DATASOURCE_PASSWORD	DB smartbear user password

Table 8: Configuration resources 'smartbear-dds-database' component

2.5.4.2 dss_core

This module has dependency with the non_fhir_database and smartbear-cdr as both components contain the data that trigger the generation of the notifications.

The basic configuration to deploy it is listed in the following detail, with a *secret* (`smartbear-dss-secret`) containing the connection strings, because they contain passwords, and API keys, and a *configMap* (`smartbear-dss-config`) containing basic setup for application. *Table 9* summarizes the configuration provided.

	Resource	Variable	Purpose
Secret	smartbear-dss-secret	CONNECTION_STRINGS_MYSQL	Server=smartbear-dss-database-svc;Port=3306;Database=smartbear;Uid=***;Pwd=***
		DSS_SECURITY_API_KEY	API Key to cross the API Manager
ConfigMap	smartbear-dss-config	URLS	http://smartbear-dss-api-svc:5000
		DEBUG	true
		ENVIRONMENT	Development
		DSS_SECURITY_URL	https://k8s.smart-bear.eu:8245/sbMobile/1.0/notification
		DSS_FHIR_DATABASE_URL	http://smartbear-cdr-svc:8080/clinical-data-repository/fhir

Table 9: Configuration resources 'dds_core' component

2.5.4.3 non_fhir_backend

This project, also known as DSS-API or api-dss is on charge of the functionalities related to the functionalities of supporting the... generation of the personalized notifications for the clinicians and the patients. The module has dependency with non_fhir_database, where the data of the non_fhir_backend are stored.

The basic configuration to deploy it is listed in the following detail, with a *secret* (`smartbear-dss-api-secret`) containing the connection strings, because they contain passwords and a *configMap* (`smartbear-dss-api-config`) containing basic setup for application, like the url of the service and flags for environment and debug status. *Table 10* summarizes the configuration provided.

	Resource	Variable	Purpose
Secret	smartbear-dss-api-secret	CONNECTION_STRINGS_MYSQL	Server=smartbear-dss-database-svc;Port=3306;Database=smartbear;Uid=***;Pwd=***
ConfigMap	smartbear-dss-api-config	ENVIRONMENT	Development
		URLS	http://smartbear-dss-svc:5000
		DEBUG	true

Table 10: Configuration resources 'non_fhir_backend/dds-api' component

2.5.5 SB Notifications

Managed by ATC, the **sb-notifications** is the app in charge of delivering notifications and alerts to the SB@App. It is based on Firebase Cloud Messaging (FCM) , which is a cross-platform messaging solution that allows to reliably send messages.

This component is triggered by the DSS when there is a need to notify the users via the SB@App.

The files supporting the Kubernetes deployment are (`\deployment\kubernetes`):

- `01-smartbear-sb-notification-database-config.yaml` — *ConfigMap*
- `02-smartbear-sb-notification-database-deployment.yaml` — *PVC*, *Deployment* and *Service*
- `03-smartbear-sb-notification-config.yaml` — *ConfigMap*
- `04-smartbear-sb-notification-deployment.yaml` — *Deployment* and *Service*

2.5.5.1 sb-notifications-database

The *sb-notifications-database* is a *MongoDB* service configured for the component.

The basic configuration to deploy it is listed in the following detail, with a *secret* (`smartbear-sb-notifications-database-secret`) containing the passwords for the MongoDB admin and custom user, and a *configMap* (`smartbear-sb-notifications-database-config`) containing basic setup for the DB and a file to initialize it. Table 3*Table 11* summarizes the configuration provided.

	Resource	Variable	Purpose
Secret	smartbear-sb-notifications-database-secret	MONGO_INITDB_ROOT_PASSWORD	Password of the root/admin user of the MongoDB
		MONGO_INITDB_PASSWORD	Variable for creating a new user in the project database (will be injected in the initialization script)
ConfigMap	smartbear-sb-notifications-database-config	MONGO_INITDB_ROOT_USERNAME	root/admin user of the MongoDB
		MONGO_INITDB_DATABASE	The Database to be created with customized name for the project
		MONGO_INITDB_USERNAME	Variable for creating a new user in the project database (will be injected in the initialization script)
		MONGO_ENTRYPOINT_JS	Is an embedded file to be executed on the first boot of the DB engine, creating an extra user for the project database.

Table 11: Configuration resources 'sb-notifications-database' component

For the MongoDB to be initialized with a custom user/password and a DB specific name, a process of "Initializing a fresh instance" will be performed. When a container is started for the first time mongosh (for mversions 6+) will execute in alphabetical order the files with extensions .sh and .js that are found mounted in the path `/docker-entrypoint-initdb.d`., using the database specified by the `MONGO_INITDB_DATABASE` variable, if it is present, or `test` otherwise.

```
                                                                    // Initializing a fresh MongoDB instance
...
  MONGO_ENTRYPOINT_JS: |
    var MONGO_INITDB_ROOT_USERNAME= process.env["MONGO_INITDB_ROOT_USERNAME"];
    var MONGO_INITDB_ROOT_PASSWORD= process.env["MONGO_INITDB_ROOT_PASSWORD"];
    var MONGO_INITDB_DATABASE= process.env["MONGO_INITDB_DATABASE"];
    var MONGO_INITDB_USERNAME= process.env["MONGO_INITDB_USERNAME"];
    var MONGO_INITDB_PASSWORD= process.env["MONGO_INITDB_PASSWORD"];
    let res = [db.createUser({
        user: MONGO_INITDB_USERNAME,
        pwd: MONGO_INITDB_PASSWORD,
        roles: [{role: "dbOwner", db: MONGO_INITDB_DATABASE}]
     })
    ];
    print('# Creation of user \'' + MONGO_INITDB_USERNAME + '\' in DB \'' + MONGO_INITDB_DATABASE + '\':
' +  JSON.stringify(res));
```

As a result of the first boot of the MongoDB, a trace like the following (*Figure 6*) will confirm the proper initialization.

```
######################################## SMARTBEAR START ########################################
## MONGO_INITDB_ROOT_USERNAME: admin
## MONGO_INITDB_ROOT_PASSWORD: [redacted]
## MONGO_INITDB_DATABASE: smartBear
## MONGO_INITDB_USERNAME: smartbear
## MONGO_INITDB_PASSWORD: [redacted]
# Creation of user 'smartbear' in DB 'smartBear': [{"ok":1}]
######################################## SMARTBEAR END ##########################################
```

Figure 6: MongoDB initialization

2.5.5.2 sb-notifications

The sb-notifications is the app in charge of delivering notifications and alerts to the SB@App. It is based on Firebase Cloud Messaging (FCM) , which is a cross-platform messaging solution that allows to reliably send messages.

This component is triggered by the DSS when there is a need to notify the users via the SB@App.

The basic configuration to deploy it is listed in the following detail, with a *secret* (`smartbear-sb-notifications-secret`) containing the passwords for the MongoDB admin and custom user, and a *configMap* (`smartbear-sb-notifications-config`) containing basic setup for the application and details for the connection with the DB. *Table 12* summarizes the configuration provided.

	Resource	Variable	Purpose
Secret	smartbear-sb-notifications-secret	MONGO_INITDB_PASSWORD	MongoDB user password
ConfigMap	smartbear-sb-notifications-config	SERVER_PORT	port with which the application is accessed
		MONGO_INITDB_USERNAME	MongoDB user username
		SERVER_SERVLET_CONTEXT	name with which the application is accessed
		SPRINGDOC_SWAGGER	Swagger URL
		GOOGLE_APPLICATION_CREDENTIALS	the location of the credential JSON file

Table 12: Configuration resources 'sb-notifications' component

2.5.6 Android App Backend

Managed by ATC, the **android-app-backend** is the app in charge of receiving the GARMIN data from the GARMIN cloud and requesting Withings token and data for the users. This app also provides the authentication for the user's GARMIN and Withings accounts.

The SB@App makes requests to this app in order to gather the user's data from the GARMIN and Withings devices.

The files supporting the Kubernetes deployment are (`\deployment\kubernetes`):

- `01-smartbear-android-app-backend-database-config.yaml` *ConfigMap*
- `02-smartbear-android-app-backend-database-deployment.yaml` *PVC*, *Deployment* and *Service*
- `03-smartbear-android-app-backend-config.yaml` *ConfigMap*
- `04-smartbear- ndroid-app-backend-deployment.yaml` *Deployment* and *Service*

2.5.6.1 android-app-backend-database

The *android-app-backend-database* is a *MongoDB* service configured for supporting the persistence needs of the Android App Backend component. Mainly, here are stored the data the app receives from the GARMIN cloud.

The basic configuration to deploy it is listed in the following detail, with a *secret* (`smartbear-android-app-backend-database-secret`) containing the passwords for the MongoDB admin and custom user, and a *configMap* (`smartbear-android-app-backend-database-config`) containing basic setup for the DB and a file to initialize it.

Table 13 summarizes the configuration provided.

	Resource	Variable	Purpose
Secret	smartbear-android-app-backend-database-secret	MONGO_INITDB_ROOT_PASSWORD	Password of the root/admin user of the MongoDB
		MONGO_INITDB_PASSWORD	Variable for creating a new user in the project database (will be injected in the initialization script)
ConfigMap	smartbear-android-app-backend-database-config	MONGO_INITDB_ROOT_USERNAME	root/admin user of the MongoDB
		MONGO_INITDB_DATABASE	The Database to be created with customized name for the project
		MONGO_INITDB_USERNAME	Variable for creating a new user in the project database (will be injected in the initialization script)
		MONGO_ENTRYPOINT_JS	Is an embedded file to be executed on the first boot of the DB engine, creating an extra user for the project database.

Table 13: Configuration resources 'android-app-backend-database' component

In the similar scenario to the component `sb-notifications`, the MongoDB will be initialized by a `smartbear.js` script mounted in the path `/docker-entrypoint-initdb.d`. that automatize the DB initialization with the creation of a named `MONGO_INITDB_DATABASE` and an `MONGO_INITDB_USERNAME` user.

2.5.6.2 android-app-backend

The *android-app-backend* is the main app serving the functionality of the Android App Backend component receiving via the GARMIN cloud the users' data from the GARMIN devices, stores them to the *android-app-backend-database* and handles the request of Withings data for the SB@App.

Table 14 summarizes the configuration provided.

	Resource	Variable	Purpose
Secret	smartbear-android-app-backend-secret	MONGO_INITDB_ROOT_PASSWORD	Password of the MongoDB root user.
		MONGO_INITDB_PASSWORD	User in the project database (will be injected in the initialization script)
ConfigMap	smartbear-android-app-backend-config	LOGGING_PATH	Logging path
		WITHINGS_CLIENT_ID	Withings client ID
		OAUTH_CONSUMERKEY	GARMIN consumer key
		EVENTPUSH_CONSUMERKEY	GARMIN consumer key
		LEGACYPUSH_CONSUMERKEY	GARMIN consumer key
		PARTNER_USER_0_USER_NAME	Default GARMIN username
		PARTNER_USER_0_USER_ID	Default GARMIN user ID
		MANAGEMENT_CONTEXT_PATH	Actuator management path
		MANAGEMENT_SECURITY_ENABLED	Actuator monitoring configuration
		ENDPOINTS_HEALTH_ID	Actuator monitoring configuration
		ENDPOINTS_HEALTH_SENSITIVE	Actuator monitoring configuration
		ENDPOINTS_HEALTH_ENABLED	Actuator monitoring configuration
		ENDPOINTS_INFO_ID	Actuator monitoring configuration
		ENDPOINTS_INFO_SENSITIVE	Actuator monitoring configuration
		ENDPOINTS_INFO_ENABLED	Actuator monitoring configuration
		INFO_APP_NAME	App name
		ENDPOINTS_ACTUATOR_ENABLED	Actuator monitoring configuration
		SPRING_APPLICATION_ADMIN_ENABLED	Whether to enable admin features
		ENDPOINTS_HYPERMEDIA_ENABLED	Provide a hypermedia-based “discovery page” for the endpoints.
		MANAGEMENT_INFO_GIT_MODE	Display properties from git.properties.
		SPRING_DATA_JPA_REPOSITORIES_ENABLED	Whether to enable JPA repositories
		REQUESTTOKEN_URL	GARMIN request token URL
		OAUTHCONFIRM_URL	OAUTH GARMIN URL
		OAUTHACCESSTOKEN_URL	GARMIN access token URL
		CALLBACKURL_URL	Withings callback URL

Table 14: Configuration resources ‘android-app-backend’ component

2.5.7 SB Dashboard

Managed by UMIL, the SMARTBEAR **Dashboard** component is the frontend application, based on the Vue.js[18] framework, which is an open-source *model-view-viewmodel* front end JavaScript framework for building user interfaces and single-page applications. The dashboard has dependencies with:

- Security Component IS, for user authentication purposes.
- Security Component UseManager for listing the registered subjects/patients.
- CDR for requesting clinical patient data.

The files supporting the Kubernetes deployment are (`\deployment\kubernetes`):

- `01-smartbear-sb-dashboard-config.yaml` *ConfigMap*
- `02-smartbear-sb-dashboard-deployment.yaml` *Deployment, Service* and *Ingress*

Table 15 summarizes the configuration provided.

[18] Vue.js framework (*https://vuejs.org*)

	Component	Variable	Purpose
Secret	smartbear-sb-dashboard-secret	VUE_APP_REGISTRATION_TOKEN	OAuth2 config parameters
		VUE_APP_LOGIN_TOKEN	OAuth2 config parameters
		VUE_APP_AUTHENTICATION_CLIENT_ID	OAuth2 config parameters
		VUE_APP_AUTHENTICATION_CLIENT_SECRET	OAuth2 config parameters
ConfigMap	smartbear-sb-dashboard-config-config	VUE_APP_BASE_URL	(e.g.: https://am.smart-bear.eu/api)
		VUE_APP_SECURITY_COMPONENT_BASE_URL	UserManager (e.g.: /usermanager/1.0)
		VUE_APP_DATA_REPOSITORY_BASE_URL	CDR url (e.g.: /sb_data_repository/1)
		VUE_APP_DSS_BASE_URL	DSS url (e.g.: /SMARTBEARDSSAPI/v1/api)
		VUE_APP_SB_BDA_URL	BDA url (e.g.: /sb_bda/v1)
		VUE_APP_REGISTRATION_URL	(e.g.: https://is.smart-bear.eu)
		VUE_APP_LOGIN_URL	(e.g.: https://am.smart-bear.eu/api)
		VUE_APP_I18N_LOCALE	Locale for the framework (e.g.: en)
		VUE_APP_I18N_FALLBACK_LOCALE	Locale for the framework (e.g.: en)
		VUE_APP_AUTHENTICATION_ENDPOINT	Auth endpoint (e.g.: https://is.smart-bear.eu/oauth2/authorize)
		VUE_APP_AUTHENTICATION_TOKEN	Auth token (e.g.: https://is.smart-bear.eu/oauth2/token)
		VUE_APP_AUTHENTICATION_LOGOUT	Auth logout (e.g.: https://is.smart-bear.eu/oidc/logout)
		VUE_APP_CALLBACK_URL	Auth callback (e.g.: https://k8s.smart-bear.eu/callback)
		VUE_APP_CALLBACK_URL_LOGOUT	Auth callback logout (e.g.: https://k8s.smart-bear.eu/logout)

Table 15: Configuration resources 'sb-dashboard' component

2.5.8 Big Data Analytics Engine (BDA)

Managed by UMIL, the **BDA** is a set of components that manage/organize/launch data analytics within the platform. This can be done manually via the dashboard in the case of pre-compiled analytics that require some configuration parameters, which is scheduled by Airflow to calculate recurring analytics, export and clean data.

The component is divided into several submodules each with a specific function and configuration, in this way the deployment and version advancement of one of the components is easier for the deployment part on Kubernetes.

The files supporting the Kubernetes deployment are (\deployment\kubernetes):

- bda-api/ 01-smartbear-bda-api-config.yaml *ConfigMap*
- bda-api/ 02-smartbear-bda-api-deployment.yaml *Deployment* and *Service*

- `airflow/` `01-smartbear-bda-airflow-config.yaml` *ConfigMap*
- `airflow/` `values.yaml` *Helm Chart values*
- `hadoop/` `01-smartbear-bda-hadoop-config.yaml` *ConfigMap*
- `hadoop/` `02-smartbear-bda-hadoop-deployment.yaml` *Deployment* and *Service*
- `hive-metastore/` `01-smartbear-bda-hive-metastore-config.yaml` *ConfigMap*
- `hive-metastore/` `02-smartbear-bda-hive-metastore-deployment.yaml` *Deployment* and *Service*
- `spark/` `01-smartbear-bda-spark-config.yaml` *ConfigMap*
- `spark/` `02-smartbear-bda-spark-deployment.yaml` *Deployment* and *Service*
- `trino/` `01-smartbear-bda-trino-config.yaml` *ConfigMap*
- `trino/` `02-smartbear-bda-trino-deployment.yaml` *Deployment* and *Service*
- `zeppelin/` `01-smartbear-bda-zeppelin-config.yaml` *ConfigMap*
- `zeppelin/` `02-smartbear-bda-zeppelin-deployment.yaml` *Deployment* and *Service*

2.5.8.1 BDA-API

The BDA API component includes the API and a *PostgreSQL* database engine to support its persistence needs. The DB request 500Mi of disk space via a PVC (`bda-postgresql-pvc`) and the configuration is loaded from the following resources. This component is the one that manages the execution of custom analytics and it will be used to fetch results of the analytics back from the dashboard.

Table 16 summarizes the configuration provided.

	Resource		Variable	Purpose
ConfigMap	smartbear-bda-configmap	postgresql	POSTGRES_USERNAME_BDA(1)	DB username (e.g.: bda_admin)
			POSTGRES_PASSWORD_BDA(2)	DB password
			POSTGRES_DB_NAME_BDA(3)	DB username (e.g.: bda)
		api	KUBERNETES_NAMESPACE	Loaded from cluster info (metadata.namespace)
			POSTGRES_URL_BDA	URL (metadata)
			POSTGRES_USERNAME_BDA(1)	DB username (bda_admin)
			POSTGRES_PASSWORD_BDA(2)	DB password
			POSTGRES_DB_NAME_BDA(3)	DB username (bda)
			AIRFLOW_API_URL	URL (airflow-smartbear-airflow:8080/api/v1)
			AIRFLOW_TOKEN	Token (airflow:airflow)
Secret	smartbear-bda-secret	postgresql	POSTGRES_USERNAME_BDA(1)	DB username (bda_admin)
			POSTGRES_PASSWORD_BDA(2)	DB password
			POSTGRES_DB_NAME_BDA(3)	DB username (bda)
		api	KUBERNETES_NAMESPACE	Loaded from cluster info (metadata.namespace)
			POSTGRES_URL_BDA	URL (metadata)
			POSTGRES_USERNAME_BDA(1)	DB username (bda_admin)
			POSTGRES_PASSWORD_BDA(2)	DB password (S3s4rl4b)
			POSTGRES_DB_NAME_BDA(3)	DB username (bda)
			AIRFLOW_API_URL	URL (airflow-smartbear-airflow:8080/api/v1)
			AIRFLOW_TOKEN	Token (airflow:airflow)
(1) (2) (3) *Same origin environmental variables shared by several components' deployment configuration*				

Table 16: Configuration resources 'bda-api' component

2.5.8.2 Airflow

Apache Airflow[19] is an open-source workflow management platform for data engineering pipelines. It started at Airbnb in October 2014 as a solution to manage the company's increasingly complex workflows. Creating Airflow allowed Airbnb to programmatically author and schedule their workflows and monitor them via the built-in Airflow user interface.

The component will be used to schedule the executions of the analytics and to monitor the various execution. It is used also to trigger the execution of custom analytics via parameters passed to the API, to manage this things the BDA-API will be the component that control the access to this component and add some kind of intelligence to him.

Table 17 summarizes the configuration provided.

[19] Apache Airflow (*https://airflow.apache.org/*)

	Resource	Variable	Purpose
Secret	smartbear-bda-secret	*Uses the smartbear-bda-secret already and previously defined secret*	
ConfigMap	smartbear-bda-airflow-config	AIRFLOW_API_URL	Airflow API (e.g.: smartbear-bda-airflow-svc:8080/api/v1)
		AIRFLOW_DATABASE_HOST	DB host (e.g.: asmartbear-bda-airflow-database-svc)

Table 17: Configuration resources 'bda-airflow' component

2.5.8.3 Hadoop

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers. It is designed to scale up from single servers to thousands of machines. We use HDFS the Hadoop Distributed File System as a primary storage to handle data exported from the FHIR repository and other various sources in order to compute analytics and store efficiently the data using Delta Tables.

HDFS can:

- *Manage Large Dataset efficiently*
- *Detecting faults*

Table 18 summarizes the configuration provided.

	Resource	Variable	Purpose
Secret	smartbear-bda-secret	*Uses the smartbear-bda-secret already and previously defined secret*	
ConfigMap	smartbear-hadoop-config	CLUSTER_NAME	Cluster name (e.g.: smartbear-hdfs)
		CORE_CONF_fs_defaultFS	(e.g.: hdfs://smartbear-namenode-svc:9000)
		CORE_CONF_hadoop_http_staticuser_user	(e.g.: root)
		CORE_CONF_hadoop_proxyuser_hue_hosts	(e.g.: *)
		CORE_CONF_hadoop_proxyuser_hue_groups	(e.g.: *)
		CORE_CONF_io_compression_codecs	(e.g.: org.apache.hadoop.io.compress.SnappyCodec)
		HDFS_CONF_dfs_webhdfs_enabled	(e.g.: true)
		HDFS_CONF_dfs_permissions_enabled	(e.g.: false)
		HDFS_CONF_dfs_namenode_datanode_registration_ip___hostname___check	(e.g.: false)

Table 18: Configuration resources 'bda-hadoop' component

2.5.8.4 Hive-Metastore

Apache Hive[20] is a distributed, fault-tolerant data warehouse system that enables analytics at a massive scale and facilitates reading, writing, and managing petabytes of data residing in distributed storage using SQL. This component doesn't start a full Hive server, but only the *Hive Metastore(HMS)* since this component is required by the Trino to access Delta Tables and it will be used also by the Spark component in order to access the tables using Spark-SQL. The HMS is a metadata repository contained in a relational database that could be accessed by the other clients such Spark, Trino and so on.

Table 19 summarizes the configuration provided.

	Resource	Variable	Purpose
Secret	smartbear-bda-secret	*Uses the smartbear-bda-secret already and previously defined secret*	
ConfigMap	hive-metastore-config	HIVE_SITE_CONF_javax_jdo_option_ConnectionURL	(e.g.: jdbc:postgresql://smartbear-metastore-database-svc.svc.cluster.local/metastore)
		HIVE_SITE_CONF_javax_jdo_option_ConnectionDriverName	(e.g.: org.postgresql.Driver)
		HIVE_SITE_CONF_javax_jdo_option_ConnectionUserName	(e.g.: hive)
		HIVE_SITE_CONF_javax_jdo_option_ConnectionPassword	(e.g.: ***)
		HIVE_SITE_CONF_datanucleus_autoCreateSchema	Autocreate schema (e.g.: false)
		HIVE_SITE_CONF_hive_metastore_uris	Hive metastore uri (e.g.: thrift://hive-metastore-svc.svc.cluster.local:9083)
		HDFS_CONF_dfs_namenode_datanode_registration_ip___hostname___check	(e.g.: false)
		CORE_CONF_fs_defaultFS	Default FS (e.g.: hdfs://smartbear-namenode-svc.smartbear.svc.cluster.local:9000)
		CORE_CONF_hadoop_http_staticuser_user	(e.g.: root)
		CORE_CONF_hadoop_proxyuser_hue_hosts	(e.g.: *)
		CORE_CONF_hadoop_proxyuser_hue_groups	(e.g.: *)
		HDFS_CONF_dfs_webhdfs_enabled	(e.g.: true)
		HDFS_CONF_dfs_permissions_enabled	(e.g.: false)

Table 19: Configuration resources 'bda-hive-metastore' component

[20] Apache Hive (*https://airflow.apache.org/*)

2.5.8.5 Spark

Apache Spark[21] is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters,iwht the following key features:

- *Batch/streaming data*: Unify the processing of your data in batches and real-time streaming, using your preferred language: Python, SQL, Scala, Java or R.
- *SQL analytics*: Execute fast, distributed ANSI SQL queries for dashboarding and ad-hoc reporting. Runs faster than most data warehouses.
- *Data science at scale*: Perform Exploratory Data Analysis (EDA) on petabyte-scale data without having to resort to downsampling
- *Machine Learning*: Train machine learning algorithms on a laptop and use the same code to scale to fault-tolerant clusters of thousands of machines.

The spark components deployed as part of the SMARTBEAR platform are based on the image provided by Bitnami that also provides an Helm Chart in order to deploy it in Kubernetes. The version used by this repo is the 3.2.0 with hadoop 2.7.4 support. The official docs seems to use the spark base image with some other scripts in order to use Kubernetes as a scheduler, the Bitnami seems to ignore that conf and use a standard Master/Executor deploy. TODO: i think that we can try to launch it from AIRFLOW using Kubernetes as scheduler, but i didn't tried yet.

The Helm Chart configuration provides the credentials for the private image registry, via the configuration of a global.imagePullSecrets with smartbear-registry-credentials.

Table 20 summarizes the configuration provided.

	Resource	Variable	Purpose
Secret	smartbear-bda-secret	*Uses the smartbear-bda-secret already and previously defined secret*	
Config Map	smartbear-bda-spark-config	*Ready for future expected configuration needs*	

Table 20: Configuration resources bda-spark component

Note the use in the values.yaml of fullnameOverride: "smartbear-spark" and the namespaceOverride: "smartbear".

2.5.8.6 Trino

Trino[22] is a fast distributed SQL query engine for big data analytics.

We use it because through this component it is easier to interact and launch queries on the data present in the platform (HDFS) and it also allows you to launch queries on different data sources. Federated queries in Trino can access your object storage, your main relational databases, all in the same query. It is faster than Hive to run queries and could substitute it to launch analytics and to performs ETL tasks that are a common to be long running batch jobs. Using Trino we could simplify the process and be faster.

[21] Apache Spark (*https://spark.apache.org/*)
[22] Trino, a query engine that runs at ludicrous speed (*https://trino.io/*)

Table 21 summarizes the configuration provided.

	Resource	Variable	Purpose
Secret	smartbear-bda-secret	*Uses the smartbear--bda-secret already and previously defined secret*	
ConfigMap	smartbear-trino-catalog-configmap	tpch.properties: | connector.name=tpch tpch.splits-per-node=4	Trino Catalog TPCH properties
		tpcds.properties: | connector.name=tpcds tpcds.splits-per-node=4	Trino Catalog TPCDS properties
		hive.properties: | connector.name=hive-hadoop2 hive.metastore.uri=thrift://smartbear-metastore-database-svc:9083 hive.non-managed-table-writes-enabled=true hive.storage-format=Parquet hive.allow-drop-table= true hive.allow-rename-table=true hive.allow-add-column=true hive.allow-drop-column=true hive.allow-rename-column=true	Trino Catalog Hive properties
	smartbear-trino-coordinator-configmap	node.properties: | node.environment=production node.data-dir=/data/trino plugin.dir=/usr/lib/trino/plugin	Trino Coordinator Node properties
		jvm.config: | -server -Xmx8G -XX:+UseG1GC -XX:G1HeapRegionSize=32M -XX:+UseGCOverheadLimit -XX:+ExplicitGCInvokesConcurrent -XX:+HeapDumpOnOutOfMemoryError -XX:+ExitOnOutOfMemoryError -Djdk.attach.allowAttachSelf=true -XX:-UseBiasedLocking -XX:ReservedCodeCacheSize=512M -XX:PerMethodRecompilationCutoff=10000 -XX:PerBytecodeRecompilationCutoff=10000 -Djdk.nio.maxCachedBufferSize=2000000	Java VM configuration
		config.properties: | coordinator=true node-scheduler.include-coordinator=true http-server.http.port=8080 query.max-memory=4GB query.max-memory-per-node=1GB memory.heap-headroom-per-node=1GB discovery-server.enabled=true discovery.uri=http://smartbear-trino-svc:8080	Trino Coordinator Config
		log.properties: | io.trino=INFO	Trino Coordinator Logger config
	smartbear-trino-worker-configmap	node.properties: | node.environment=production node.data-dir=/data/trino plugin.dir=/usr/lib/trino/plugin	Trino Worker Node properties
		jvm.config: | -server -Xmx8G -XX:+UseG1GC -XX:G1HeapRegionSize=32M -XX:+UseGCOverheadLimit -XX:+ExplicitGCInvokesConcurrent -XX:+HeapDumpOnOutOfMemoryError -XX:+ExitOnOutOfMemoryError -Djdk.attach.allowAttachSelf=true	Trino Worker Java VM configuration

		-XX:-UseBiasedLocking -XX:ReservedCodeCacheSize=512M -XX:PerMethodRecompilationCutoff=10000 -XX:PerBytecodeRecompilationCutoff=10000 -Djdk.nio.maxCachedBufferSize=2000000	
		config.properties: | coordinator=false http-server.http.port=8080 query.max-memory=4GB query.max-memory-per-node=1GB memory.heap-headroom-per-node=1GB discovery.uri=http://smartbear-trino-svc:8080	Trino Worker Config
		log.properties: | io.trino=INFO	Trino Worker Logger config

Table 21: Configuration resources 'bda-trino' component

2.5.8.7 Zeppelin

Apache Zeppelin[23] is a web-based notebook that enables data-driven, interactive data analytics and collaborative documents with SQL, Scala, Python, R and more.

Zeppelin will be used by the Data Scientist(or people that can code) to sketch analytics and to share results to the other people of the same organization. This analytic could be then a starting base to be implemented and added to the catalogue of the available analytics to be launched with the proper parameter from the UI.

There are 5 different images that are built during the process:

- zeppelin-distribution: this one is used by the others as a base(not launched by anyone)
- zeppelin-server: this is the one that launches the server image and is the main point of control
- zeppelin-interpreter: this is the one launched to execute some task since it contains all the dependencies required(like for python or R or JDBC ...)
- zeppelin-spark: this is the one used to execute Spark tasks by Zeppelin
- zeppelin-shiro-conf: it's a busybox image used only for the initContainer phase to build the file needed to add authentication for the user

Table 22 summarizes the configuration provided.

	Resource	Variable	Purpose
Secret	smartbear-bda-secret	*Uses the smartbear--bda-secret already and previously defined secret*	
ConfigMap	zeppelin-config	SERVICE_DOMAIN	Zeppelij service domain (e.g.: local.zeppelin-project.org:8080)
		ZEPPELIN_K8S_INTERPRETER_IMAGEPULLSECRETS	Credentials for the private repository where Zeppelin Interpreter Docker image are stored (e.g.: smartbear-registry-credentials)
		ZEPPELIN_K8S_SPARK_CONTAINER_IMAGE	Zeppelin Spark Docker image (e.g.:

[23] Apache Zeppelin (*https://zeppelin.apache.org/*)

			registry.atosresearch.eu:18449/arihealth/smartbear/zeppelin-spark:0.10.1)
		ZEPPELIN_K8S_CONTAINER_IMAGE	Zeppelin Interpreter Docker image (e.g.: registry.atosresearch.eu:18449/arihealth/smartbear/zeppelin-interpreter:0.10.1)
		ZEPPELIN_HOME	Zeppelin home URI (e.g.: /opt/zeppelin)
		ZEPPELIN_SERVER_RPC_PORTRANGE	RPC port range (e.g.: 12320:12320)
		SPARK_MASTER	Is the default value of 'master' property for spark interpreter. (e.g.: k8s://https://kubernetes.default.svc)
		SPARK_HOME	Spark home URI (e.g.: /spark)
		init.db	DB initialization
		nginx.conf	NGINX configuration
		interpreter-spec.yaml	Interpreter pod configuration

Table 22: Configuration resources 'bda-zeppelin' component

2.5.9 OpenHAB Cloud

Managed by UOI, rhis component is an **openHAB** Cloud[24] instance (v1.0.17[25]) installed in the Kubernetes cluster, substituting *myopenHAB.org* which is a public instance of the openHAB Cloud service hosted by the openHAB Foundation[26]. The installation of the component in the SMARTBEAR Kubernetes cluster requires:

- The **openhab-cloud** application itself, based on node.js and express.js technologies, with the following docker image published available: openhab/openhabcloud-app:latest.
- A **MongoDB** database with a version4+ (v6.0.4 aligning to the docker image required for other platform components),
- A **Redis** session manager, which is an in-memory data store, with the following docker image published available: bitnami/redis:latest.

[24] openHAB Cloud (*https://github.com/openhab/openhab-cloud*)
[25] openHAB Cloud Releases (*https://github.com/openhab/openhab-cloud/releases*)
[26] OpenHAB Fundation (*https://www.openhabfoundation.org*)

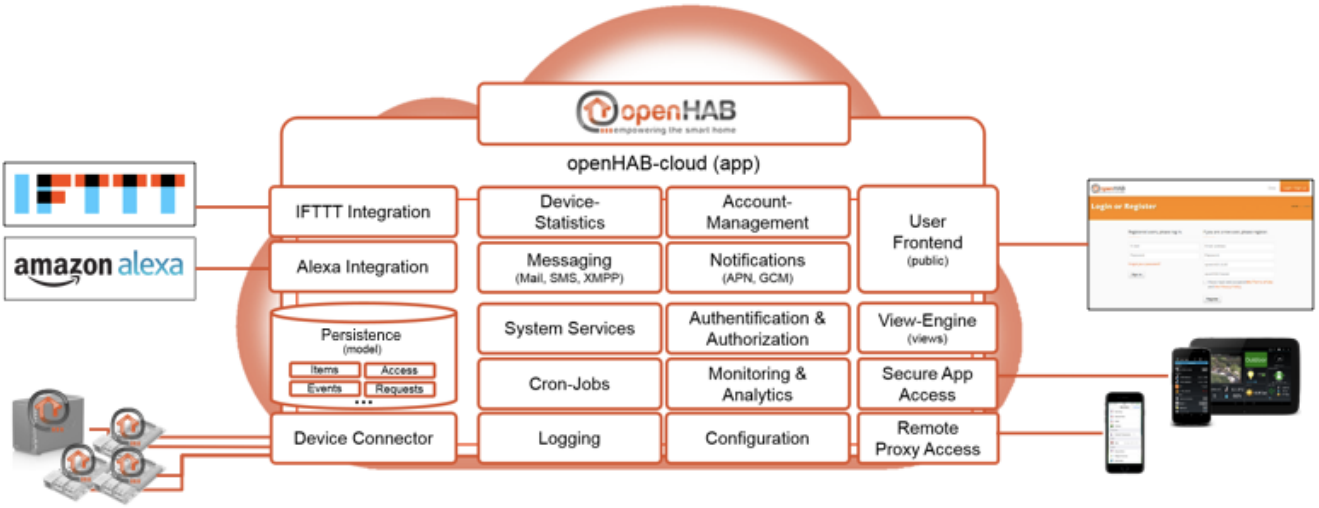

Figure 7: OpenHAB Cloud Functional Architecture[27]

The process of configuration requires, on the one hand provide the DBs required by the component, a MongoDB and REDIS instances, and in the other hand the configuration via a template system of the cloud instance.

The files supporting the Kubernetes deployment are (`\deployment\kubernetes`):

- `01-smartbear-openhabcloud-config.yaml` — *ConfigMap*
- `02-smartbear-openhabcloud-deployment.yaml` — *PVC*, *Deployment* and *Service*
- `03-smartbear-openhabcloud-config.yaml` — *ConfigMap*
- `04-smartbear-openhabcloud-deployment.yaml` — *Deployment*, *Service* and *Ingress*

2.5.9.1 openhabcloud-database

The openhabcloud-database component is in charge of running two different DB systems, a MongoDB and also a REDIS. The only set up needed is, for the MongoDB to initialize a DB with a specific `MONGO_INITDB_DATABASE` name, since for now it was decided that authentication is not necessary. Setting it would require the initialization of the DB with a specific username and password on the created database, and its setting in the configuration of the main instance of the openhabcloud. Regarding for REDIS, just providing a `REDIS_PASSWORD` is enough and this value will be the one used later in the main component.

Table 23 summarizes the configuration provided.

[27] openHAB Cloud (*https://github.com/openhab/openhab-cloud/blob/main/docs/FunctionalArchitecture_openHAB-cloud_v1.0.png*)

	Resource	Variable	Purpose
Secret	smartbear-openhubcloud-database-secret	MONGO_INITDB_ROOT_PASSWORD	MongoDB root user password
		MONGO_INITDB_PASSWORD	Variable for creating a new user in the project database (will be injected in the initialization script) as custom smartbear user password.
		REDIS_PASSWORD	The REDIS password, that will be needed fro the main openhabcloud component
ConfigMap	smartbear-openhubcloud-database-config	MONGO_INITDB_ROOT_USERNAME	MongoDB Root username (e.g.: admin)
		MONGO_INITDB_USERNAME	Custom smartbear username
		MONGO_INITDB_DATABASE	MongoDB initial and default database name (default: openhab)

Table 23: Configuration resources 'openhabcloud-database' component

2.5.9.2 openhabcloud

The openhabcloud is the instance of OpenHAB Cloud deployed in the cluster and customized for the SMARTBEAR project. This integration ensures the high-availalbility and secure manageability of the homeHAB installations.

The configuration of the tool, based on a simple template system, has had to be expanded to allow the template to be fulilled with the configuration details stored in the Kuberentes resources of type configmap and secret (and defined in yaml manifest files), aligning to the project configuration convention for the SMARTBEAR integration in the cluster. The process requires also to point to the databases created for supporting the persistence of this component as described in previous section.

Embedded in the configmap a file (named `config.json.template`) will be copied within the container. This file is drafted with a number of placeholders that will complete the needed configuration when the application boots, replacing placeholders by environmental values and overwrite the file expected for booting the application, gives us a chance to customize configs through runtime environment.

The requirements of configuration are a *secret* (`smartbear-openhabcloud-secret`) containing the passwords for connecting to the DB smart-bear schema, and a *configMap* (`smartbear-cdr-config`) containing the basic setup for the app to boot up, which, as mentioned, will be used to complete the template file.

Table 24 summarizes the configuration provided.

	Resource	Variable	Purpose
Secret	smartbear-openhabcloud-secret	MONGO_INITDB_PASSWORD	Password for the custom smartbear user in MongoDB (e.g.: Sm4rtB3ar)
		REDIS_PASSWORD	To inject into the template, the password used to initialize Redis (e.g.: ****)
		MAILER_PASSWORD	To inject into the template, the password used to authenticate in the Mailer service
ConfigMap	smartbear-openhabcloud-config	EXPRESS_KEY	Variable to inject into the template
		DOMAIN_NAME	Variable to inject into the template
		MONGO_DATABASE	Variable to inject into the template
		MONGO_INITDB_USERNAME	The custom smartbear username (e.g.: smartbear)
		MONGODB_HOSTS	Variable to inject into the template
		REDIS_HOST	Variable to inject into the template
		REDIS_HOST_PORT	Variable to inject into the template (e.g.: 6379)
		CONFIG_JSON_FILE	Is the template that openhabcloud use to initialise the instance, ready with the needed placeholders to be replaced on the initialization of the container.

Table 24: Configuration resources 'openhabcloud' component

3 GitLab CI/CD

The SMARTBEAR platform components buiding process, docker images generation and automated deployment into the Kubernetes development environment for integration, is mainly handled by the set up of the GitLab CI/CD functionalities as repeatedly described previously.

The tool behaviour is based on the definition of CI/CD *pipelines* and the circumstances in which they are executed. To homogeneize the configuration of the GitLab CI/CD functionality, the different projects maintained in the repository, aggregates the required kubenretes descriptor files under the directory `/deploment/kubernetes/<environment_subdomain>`. This structure will make it possible to differentiate between the configuration required by the different environments, mainly due to the use of different subdomains and what this implies in terms of configuration, for example, configuring the ingress to the Kubernetes cluster.

3.1 The .gitlab-ci.yml file

To configure the functionality provided by Gitlab CI/CD, a file called .gitlab-ci.yml should be created in the root of the projects repository. This file defines:

- The specific scripts to run
- If the scripts shoud run automatically or be triggered manually.
- Configuratio from other configuration files, templates or CI/CD variables defined to include.
- Workflow dependencies, scripts that should run in sequence and those you want to run in parallel.
- Posibility of ussing caches, for exemple building maven or npm applications
- The locations were to deploy the resulting artifacts, images registries, packages repositories, even also deployment to execution cluster.

Most of the componenets forming part of the SMARTBEAR platform has .gitlab-ci.yaml files in their repositories that will facilitate the operations of building and deploying them and are configured mainly to use as refenrence a GIT branch “develop” and deploy latest/snapshot versions to de development environment.

Promotion to subsequent environments, like staging or production, use the master/main branches and manually fix the versions for generating the artifacts for such environments ina controller deleasing procedure.

3.2 Deployment via yaml kubernetes resources descriptors

The basic method to create resources in a Kubernetes, such as pods, services, and deployments, is by using the *declarative* way based on using YAML files. The command `kubectl` provides the means to interact with the Kuberentes API and supply such files.

As mentioned above, as a convention it has been established that the YAML files necessary for each project (that will use this method for deployment) must be located in a folder `/deployment/kubernetes` under the root folder of each project code repository, to allow the homogeneous maintenance of the configuration `.gitlab-ci.yaml` files that configure the Gitlab CI/CD functionalities.

For this scenarios based directly in yaml files, the `.gitlab-ci.yaml` file approach for deploy the component, and depending on the needs of the specific component build and deployemnt process, perform the following two jobs:

- ***Building*** job
  Since all components should eventually generate and send a docker image to the project private image registry, the `docker` command should be available, sometimes as a *dind*[28] (docker-in-docker) GitLab runner service.
  - Based on docker images:
    - `maven:3.6.3-jdk-11-slim`, `maven:3.8-jdk-11`, `maven:3.8.5-openjdk-11-slim`, etc. dependieng on the specific component, for building phases for *Java+Maven* projects that needs the `mvn` command for build the component, and `docker` to generate the images.
    - `docker:latest`, for building based on executing docker build over project provided building files (Dockerfiles).
  - Execute the proper buid command depending on the component technology.
  - Execute the proper `docker build` image genaration comand if needed.
  - Execute the proper image tagging with the `docker tag` command if needed.
  - Execute the `docker pull` operation to store the image in the private repository.
- ***Deployment*** job;
  - Based on the image `lachlanevenson/k8s-kubectl:latest` that provides a maintained access to the command `kubectl`, althoug considering moving to the well known `alpine`[29] docker image and to install on it the `kubectl` package from the project test repository.
  - Execute the `kubectl apply` command to the base resources, Namespace and Secrets, contained in the GitLab CI/CD Variable `$PROJECT_K8S_CONFIG`.
  - Execute the `kubectl apply` command over the numered yaml files in the stablished sequence order.
  - Optional command to `rollout restart` the deployment.

Once finished a successfly deployment, with all the resources created/updated, the way to remove the component manually in case of need, will be by executing the `kubectl delete` command with all the *yaml* files processed in the creation in the oposite order. In any case, since the scripts uses the *declarative* way (`kubectl apply`) instead of the *imperative* way (`kubectl create`), subsequent executions will update the status over the already existing resources, in such case.

3.3 Deployment via Helm Charts installers

Helm is a package manager for Kubernetes and uses a packaging format called *charts* for automating installations. A *chart* is a collection of files that describe a related set of *Kubernetes* resources and could be used for simple deployments but its pottential is when become complex, like a full app stack with servers, databases, caches, and more.

---

[28] GitLab Docker-in-Docker (*https://docs.gitlab.com/ee/ci/docker/using_docker_build.html#use-docker-in-docker*)
[29] Alpine, a minimal Docker image based on Alpine Linux with a complete package index and only 5 MB in size! (*https://index.docker.io/_/alpine*)

Some of the SMARTBEAR components, due to the complexity of its deployment and becaouse are open-source maintained by the community, comes with an installation process based in *charts*, with just the need of providing the proper versioned parameters in a `values.yaml` file, to generate and deploy the component. At the same time, this configuration files could be filled from other configuration, for example in the case of using some kind of confidencial okens or passwords, thak could be taken from *secret* resources from *Kubernetes*. In addition, this technique allows updating in a controlled manner.

For this scenarios based on charts, the `.gitlab-ci.yaml` file approach for deploy the component, and based on a docker image `alpine`, perform the following steps:

- Install via APK the packages curl, helm and ca-certificates.
- Add as repository the command `kubectl`.
- Execute the `kubectl apply` command to the base resources, Namespace and Secrets, contained in the GitLab CI/CD Variable `$PROJECT_K8S_CONFIG`.
- Add to the Helm repository the source of the chart, and update list of charts.
- Execute the command uptade, with the proper configuration for also install it for fist time if nos still installed.

After the installation, the component installed vía this method will be listed as `Helm>Releases`, grouping under this concept all the resources deployed.

4 Concluding Remarks

The process to get ready all and each SMARTBEAR platform components for being deployed into a standardized Kubernetes cluster, despite the effort required, provides the system an execution environment more maintainable, secure, scalable, and adaptable to the necessary resources.

In addition, its setup together with CI/CD functionalities provides an agile development environment in solving bugs and adopting new functionalities. Likewise, the adoption of the standardized configuration required by the Kubernetes environment would potentially allow the hosting of the platform in any provider ready for this technology.