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"

D6.4 (D29) – Report on integrated SMART BEAR platform v2

Due date of deliverable: 28.02.2023

Actual submission date: 27.02.2023

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	

D6.4 (D29) – Report on integrated SMART BEAR platform v2

Editors

Manuel M Perez Perez – ATOS

Alberto Acebes – ATOS

Contributors

Daniel Calvo – ATOS

Adalberto Sangalli – ARIA

Gianluca Carletti - ARIA

Reviewers

Luigi Gallo - CNR

Ioannis Basdekis - STS

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 document describes the actions performed within the scope of the WP6 related to setting up the K8s framework and migration of the SB@Cloud components. This deliverable is closely related to deliverable D6.3 “Integrated SMART BEAR platform”, as well as the previously submitted deliverables D6.8, D6.2 (Report on integrated SMART BEAR platform) and D6.7, D6.1 (Integrated SMART BEAR platform). In this regard, this deliverable is a direct evolution of D6.2.

As also reported in D6.2, the union of all the information gathered in D6.x deliverables complements D6.4 and provides a complete vision, practical, and demonstrative information about the integration of the components for the prototype. For this reason, and in order not to be repetitive in terms of content, frequent cross-references are made, whereas in some cases and for clarity shake, some tables are repeated. In that case, the reference to the correspondent deliverable is provided.

As we reported, the integration scope evolved from the original planning to fulfil the PoP requirements. The PoP is based on a simplified integration of the platform. The provided functionalities of the MVP platform allowed the consortium to demonstrate the main functionalities to test them in clinical settings. During the PoP, the different components were evaluated, some dysfunctionalities were reported, and some bugs, as well in the code, were fixed. Therefore, during the last 12 months, the platform has been used in the PoP in Madeira. Due to the above, as already reported in D6.2, we take special care in not creating confusion between the advanced work for the integration of the final platform (CI/CD, Kubernetes-related configuration, extended security aspects) and the work done, and temporally alternatives applied in this integration for the PoP.

Contents

Executive Summary **3**

List of acronyms **5**

List of Figures **6**

1 Introduction **7**

1.1 Goals of the deliverable 7

1.2 CI/CD integration 7

1.3 Kubernetes 8

2 GitLab CI/CD **9**

2.1 GitLab CI/CD advantages 9

2.1.1 Pipelines 9

2.1.2 Variables 10

2.1.3 Environments 10

2.1.4 GitLab Runner 10

3 Kubernetes **11**

3.1 Data storage 11

3.2 Cloud oriented and load balance service – Ingress controller(nginx) 11

3.3 HAProxy 12

3.4 DNS Resolution in the cluster (CoreDNS) 12

3.5 Monitoring the cluster (Prometheus) 12

3.6 Automatic renewal of certificates in the cluster (cert-manager) 13

3.7 Kubernetes descriptor files 13

4 Security **14**

5 Assistance and Testing **15**

6 Concluding remarks **16**

List of acronyms

Acronym	Description
ACME	Automatic Certificate Management Environment
API	Application programming interface
BDA	Big Data Analytics
CA	Certificate Authority
DSS	Decision Support System
GDPR	General Data Protection Regulation
PoP	Pilot of Pilots
REST API	(Also, RESTful API) is API that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services
SB	Smart Bear
SB@App	Smart Bear smartphone app

List of Figures

Figure 1: Trello board showing testing and maintenance activities 15

1 Introduction

This document is 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. *(in parallel, to be delivered on month 42)*
- **D6.4 Report on integrated SMART BEAR platform v2. *(this document, 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 Goals of the deliverable

As described in the previous deliverables[1], the Pilot of Pilots (PoP) consisted of the initial demonstration of the implemented minimum functionalities of the SMART BEAR platform, and it was carried out successfully from the conceptual and functional point of view. The evolved versions of the platform aim at implementing full operatives prototypes ready for the large-scale pilots. Upscaling the infrastructure capabilities, as well as the maintenance process, so as to make the platform robust enough to address all pilots' requirements, are important aspects for the large-scale pilots. This deliverable provides information on the tasks performed during the past 12 months; further details can be found in the accompanying deliverable D6.3.

1.2 CI/CD integration

Continuous Integration and Continuous Deployment/Delivery (CI/CD) refers to the use of a series of practices that aims to reduce the time it takes to deliver software applications to production. Continuous Integration (CI) is the practice of synchronised code changes from multiple developers into a single codebase. Therefore, we can catch and fix errors as soon as they are reported through the SMART BEAR feedback corner. As part of T6.2, automated testing tools will also be implemented for running tests on the code every time they need to be updated. Continuous Deployment (CD) is the practice of automatically deploying code changes to the production environment as soon as they pass the automated tests. This ensures that the code changes made for improving services, refinements or resolving bugs are integrated into the software product quickly and efficiently. That is key when we address large-scale pilots in different sites, where different issues might be detected, even at the same time.

Our last activities in T6.1 are focused on migrating for a manual docker-compose integration to automate the entire development cycle (CI/CD), from the initial code development to deployment and delivery to users. This will help reduce the risk of human errors, correct bugs, improve software quality, and accelerate time-to-production more efficiently. Additionally, it can help teams to respond

[1] *D6.7 Integrated SMAR BEAR platform V1* and *D6.8 Report on integrated SMART BEAR platform V1*

more quickly to user feedback and pilots' requested changes. Therefore, the main bulk of activities during this period was orientated to set up the K8s framework, for CI/CD.

1.3 Kubernetes

As previously introduced in previous WP6 deliverables, we chose Kubernetes (K8s) as the framework for CI/CD. Kubernetes is oriented to microservice, providing substantial advantages for containerisation of the SMART BEAR applications allowing easier control of the resources and providing simplified methods for escalating them, if needed.

2 itLab CI/CD

GitLab[2] CI/CD is an integrated CI/CD deployment system built into GitLab, a web-based Git-like repository management tool. We use GitLab in SMART BEAR for easing the CI/CD, the code repository facilitates the control and traces of the different versions of the SMART BEAR components, with the additional advantage that all the components can be tested in a staging environment and easily promoted and deployed in production. GitLab CI/CD has built-in support for Docker, a popular containerisation platform, allowing developers to easily build and deploy containerised applications. GitLab CI/CD is based on a configuration file called *.gitlab-ci.yml*, which defines the jobs and stages of the CI/CD pipeline (see Section 2.1.1 below). Every job is specifically related to an individual task, as for example, building the code, executing tests, or deploying a component, and according to the provided configuration, can be run in parallel or sequentially.

2.1 GitLab CI/CD advantages

The main advantage of GitLab CI/CD is that it provides an efficient and automated way to manage the entire SMART BEAR platform development lifecycle of the project, from building and testing to deployment and monitoring. Another main advantage is the automatisation of the process of building and deployment into a development environment when modifications of the code are made, and the possibility of automating its testing. Whenever changes to the components are made to the code repository, GitLab CI/CD automatically triggers the build and test process to ensure that changes are integrated correctly, skipping the effect of the changes and notifying the circumstance otherwise. Additionally, there are facilities for the deployment of the changes to a production environment, either manually or through automated workflows.

It is important to highlight that as the SMART BEAR infrastructure is set up over three different environments, namely, the *development*, *staging* and *production*, Gitlab CI/CD offers the required multi-environment support. Therefore, it will allow us to test and deploy changes among the different environments before releasing them to production, ensuring the quality of the changes, and making them more robust, which is essential for large-scale pilots.

2.1.1 Pipelines

CI/CD pipeline, developers can catch issues and bugs earlier in the development process, reduce manual errors, and release new features and updates to production environments more quickly and reliably. In the project, some pipelines are implemented to build, test, and deploy the application components quickly and efficiently. It involves a series of steps, from code integration to testing and deployment, in order to ensure the quality and functionality of the different components. The pipelines defined includes pulling the changes from the repository automatically, compiling and building the components, and then running automated tests to validate functionalities and catch any bug or problems. Reaching the end of the pipelines successfully ensures that it is possible to release a version of the component to be promoted to the staging environment and, after passing a battery of tests in the staging environment, the components will be ready for being deployed to the production environment.

[2] GitLab (*https://about.gitlab.com/*)

2.1.2 Variables

Similar to an environment variable, but in the scope of CI/CD functionalities, these variables contain useful values that are both predefined[3] and system-maintained (current build identifier, hostname where Gitlab is installed, current user running a build, operation...) as customisable created by the user to meet specific parameterisation needs.

2.1.3 Environments

As we stated in previous deliverables, it is important in a complex project to set up differentiated environments for development and for production. In addition, in SMART BEAR, we set up a pre-production or staging environment for automatic and manual testing. Therefore, we 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. The functionality of the concept of *namespaces* in the scope of the Kubernetes clusters, provides a convenient means for this purpose, taking advantage of all the configuration and resources available and isolating the deployment of the just for testing purposes needed components versions, sharing the cluster hardware with the development environment.

2.1.4 GitLab Runner

In order to automatise the building of the platform from its individual components, the testing and the deployment of the SMART BEAR prototypes, we use the open-source tool offered by GitLab that takes care of the processing needs of these tasks, even on a remote machine, as in this case, where GitLab is located on an ATOS instance, but the runner is installed on the SMARTBEAR Kubernetes cluster. GitLab runner is an essential CI/CD tool that is perfect for working with the SMART BEAR GitLab, repository manager. The main advantage of using GitLab Runner is that it can be used to execute jobs, which are defined in a YAML file (*.gitlab-ci.yml*) and stored in a GitLab repository. These jobs will be used to build and test the components and push the generated docker images into a registry, so they are available for use from different environments. We set a predefined runner to execute the needed jobs.

The GitLab runner also allows us to automate many aspects of the implementation process, streamlining the workflows and reducing the time it takes to build the platform and test the components, which all is translated to a significant improvement in the quality of the SMART BEAR releases.

[3] GitLab (*https://docs.gitlab.com/ee/ci/variables/predefined_variables.htm/)*

3 Kubernetes

As we mentioned in the previous deliverable, the current integration was chosen as a quick alternative to the Kubernetes-based integration, for speeding up the whole process, at the cost of not having a fully operative CI/CD framework for the PoP. Now, in order to automatise the integration, the current implementation based on Virtual Machines and *Docker-Compose* is migrated to the Kubernetes framework. The main advantages are that Kubernetes facilitates the management and scalability of containerised applications. With Kubernetes, the applications are defined as a set of declarative configuration files that specify how many replicas of each container are needed, what resources are required, and how they should be networked together. There are also a number of additional features to ensure that SMART BEAR applications are highly available and resilient, such as automatic failover, rolling updates, and self-healing capabilities. It also supports a wide range of container runtimes and can run on any infrastructure, whether it be on-premises, in the cloud, or at the edge.

3.1 Data storage

The persistence of data required by the platform is still supported by an NFS service installed on a dedicated standalone machine, independent of those that form the cluster. Unlike the previous set-up in version V1 of the platform, was used a basic "NFS provisioner", this version includes a more advanced installation with an *NFS-CSI driver*, which is the technology of more recent creation and that includes more advanced security features and the possibility of using backup methods, as detailed in the deliverable D6.4.

From the point of view of the Kubernetes cluster, the operation of the functionality to provide persistence to the components is maintained in a similar way, with the use of *PersistentVolumeClaims* (PVC), which creates *PersistentVolumes* (PV) type of Kubernetes resources, but internally, the process (*StorageClass*) providing the service, use the new driver to operate the NFS, mounting the shared folder and creating the subfolder where the PV for the pod is allocated. One of the advantages of using the NFS Subdir External Provisioner is that it allows multiple pods to share the same NFS volume. This means that multiple pods can read and write to the same NFS share avoiding conflicts. It allows us to create a specific directory on the NFS server as a base for all dynamically created volumes. The directory is configured during installation. When a new PVC is created, the NFS Subdir External Provisioner creates a new subdirectory under the specified subdirectory path and mounts it to the Kubernetes pod as a persistent volume.

3.2 Cloud oriented and load balance service – Ingress controller(nginx)

In order to manage and route external traffic to services within the SMART BEAR Kubernetes cluster we use Ingress Controller (nginx). Nginx is a cloud-oriented load-balancing service that provides an HTTP/HTTPS interface to manage incoming traffic and to perform advanced routing, SSL termination, and other operations as to manage incoming traffic to Kubernetes services. Using the nginx Ingress controller, we manage the distribution of incoming traffic across multiple pods when possible, in Kubernetes, helping to balance the workload and prevent the overloading of a single pod. Additionally, the controller allows us to handle SSL/TLS encryption and decryption, allowing incoming traffic to be securely forwarded to the appropriate service. In other to allow services to be exposed on different paths., nginx offers tools such as the URL rewriting of incoming traffic, enabling services

to be exposed with custom paths and the Path-based Routing for routing traffic based on the path of the incoming request. Moreover, nginx can be used as a reverse proxy, forwarding traffic to a service running on a non-standard port. Nginx is deployed as a Kubernetes resource, rules are defined according to how incoming traffic should be routed to the SMART BEAR components and services, and it will work complementing the API manager provided by the security component.

3.3 HAProxy

Since the cluster is hosted on its own machines and not in a cloud provider, it is still necessary to use HAProxy as reverse proxy in the host machines where the nodes of the Kubernetes cluster are installed, to distribute incoming traffic.

By design of the platform, it is expected that all traffic will be done primarily through ports 80 and 443 (HTTP and HTTPS) and through the appropriate subdomains redirected to the master node of the cluster, and these are the ports forwarded directly against Kubernetes.

HAProxy configuration remains available in case of the need to expose other ports for extraordinary functions, such as monitoring or data migration. In addition, it can be used to audit the calls received by the cluster and prevent certain types of attacks.

3.4 DNS Resolution in the cluster (CoreDNS)

In the SMART BEAR Kubernetes framework, DNS resolution is a critical component that allows components and pods to communicate with each other using human-readable domain names instead of IP addresses. When a pod needs to communicate with another pod or service in the SMART BEAR cluster, it sends a request to the DNS server. The request includes the domain name of the target pod or service. The DNS server (CoreDNS) receives the request and looks up the IP address of the target pod or service. Once the IP address of the target pod or service is resolved, the DNS server returns the IP address to the requesting pod. The requesting pod can then use the IP address to communicate with the target pod or service. This method provides a flexible and extensible DNS server that can be configured, providing, therefore, custom domain name resolution within the Kubernetes cluster, as it is reported in D6.3. It can be deployed as a separate pod or as a DaemonSet to run on each node in the cluster. The resolution system also allows us to keep the services isolated between different namespaces, which is useful in the case of using them to distinguish two different environments, such as development and staging in the same cluster, and even jump between them if necessary.

By default, CoreDNS is the DNS server configured to resolve domain names for Kubernetes services and pods in the SMART BEAR cluster.

3.5 Monitoring the cluster (Prometheus)

Prometheus is a popular monitoring solution for tracking metrics and alerting on system performance. Prometheus is installed on the SMART BEAR server within the Kubernetes cluster, and it is configured to scrape metrics from the project services and components in the cluster.

Prometheus can be configured to alert when certain metrics fall outside of defined thresholds as well, and a built-in visualisation tool called Grafana can be used to create dashboards and graphs to display the cluster metrics. Therefore, Prometheus allows us to monitor the cluster and troubleshoot any performance issues that could arise during the large-scale pilot.

3.6 Automatic renewal of certificates in the cluster (cert-manager)

Cert-manager is a Kubernetes add-on that automates the management and issuance of TLS certificates. With cert-manager, we can set up a certificate authority (CA) within the Kubernetes cluster and use it to automatically issue, renew, and revoke TLS certificates for the SMART BEAR applications. We set up automatic renewal of TLS certificates for the Kubernetes applications using cert-manager and Let's Encrypt[4] as certificate provider. This ensures that SMART BEAR applications always have up-to-date certificates and reduce the risk of downtime due to expired certificates.

3.7 Kubernetes descriptor files

Kubernetes descriptor files are YAML or JSON files that define the desired state of Kubernetes objects. Kubernetes objects are the building blocks of a Kubernetes cluster, such as pods, services, deployments, and others. We use these descriptor files to manage and maintain the state of these objects. The descriptor files specify the configuration details of each object, such as the container image, number of replicas, port mappings, resource requirements, and others.

To apply a Kubernetes descriptor file, we use mainly the *kubectl apply* command, which sends the file through the Kubernetes API to the server and instructs it to create or update the specified objects. We can also use the *kubectl delete* command to delete objects specified in the descriptor file.

[4] Let's Encrypt, a nonprofit Certificate Authority providing TLS certificates (*https://letsencrypt.org)*

4 Security

The SMART BEAR platform is, at this stage, almost implemented in the development environment; prior to being deployed in production, automatic tests will be performed in the staging environment. To be used in large-scale pilots, the platform will be implemented with the highest requirements of security. As it is reported in the previous version of this deliverable, D6.2, the access routes are limited to the essential ones and will be secured.

5 Assistance and Testing

As reported in D6.2, we used Trello and Hyporello (see Figure 1) for building the SMART BEAR feedback corner. During the PoP, the consortium has used these tools to manage the issues, refinements and bugs that have arisen during the PoP. Some of them have been only managed thanks to the features offered by these tools, due to the complexity of interpreting the errors and defining responsibilities. More than 35 issues were reported and satisfactorily addressed by the SMART BEAR consortium.

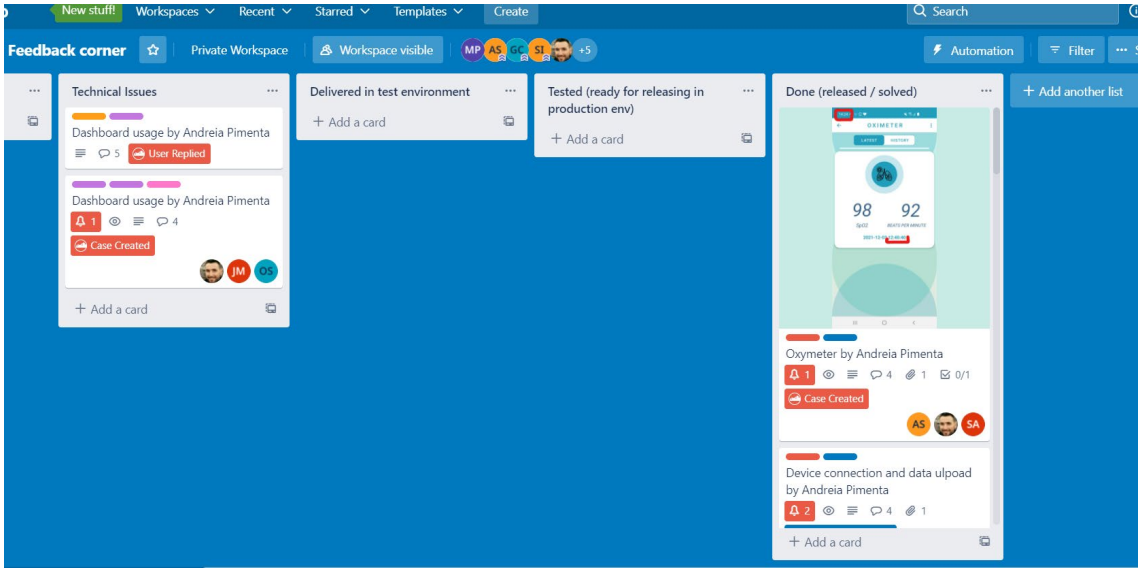

Figure 1: Trello board showing testing and maintenance activities

6 Concluding remarks

The activities performed during the past twelve months have been orientated to migrate the prototype based on Virtual Machine and docker-compose to CI/CD framework. Among the different technologies, we chose the Kubernetes framework along those technologies that better fit the requirements of the project. The current version of the SMART Platform also has taken advantage of the lesson learnt during the Pilot of the Pilots. All the technical specifications have been reported in D6.3.