



DATA-PROTECTION TOOLKIT REDUCING RISKS IN HOSPITALS AND  
CARE CENTERS

Project N° 826284

---

**ProTego**

**D6.3 Final prototype: Architecture, requirements, and  
integrated toolkit**

---

Responsible: ICE

Contributors: Inetum, IT Innovation, IMEC, IBM, KU Leuven, UAH

Document Reference: D6.3

Dissemination Level: Public

Version: v1

Date: 31/07/2021

## Executive Summary

This document describes the platform architecture, component resources and the requirements as well as the integration work and the Integration Toolkit. ProTego is an integrated toolkit which consists of six tools: System Security Modeller, Security Information and Event Management, Continuous Authentication, Data Gateway, Access Control Framework, and Network Slicing. These tools are integrated together using the Integration Toolkit which allows for an updatable, continuously integrated platform to be deployed in each hospital. These are mostly deployed as Docker containers on a ProTego platform but certain aspects will be deployed on mobile devices, or on public or private clouds. The Platform Architecture in two levels is first described. Then, a detailed description of the Integration Toolkit is presented, which is the main focus of WP6 work. The Requirements include several points and how they relate to the project. Integration requirements and resource estimates are also provided for all components. We discuss the interoperability including the ProTego components, the Integration Toolkit and the status of the WP4 and WP5 final integration. Finally, we describe how recommendations from the previous WP6 deliverable D6.2 have been addressed and set out some conclusions.

## Contributors Table

| DOCUMENT SECTION                     | AUTHOR(S)  | REVIEWER(S)   |
|--------------------------------------|--|---|
| <b>I Introduction</b>                | Philip Usher, Noel Tomas (ICE)   | Antonio Jesús Gamito González, Luis Carrascal (Inetum), Colin Upstill (ICE)                   |
| <b>II Platform Architecture</b>      | Philip Usher, Noel Tomas, Arturo Jimenez Arriaga (ICE)   | Antonio Jesús Gamito González, Fernando Rendon, Luis Carrascal (Inetum) , Colin Upstill (ICE) |
| <b>III Integration Toolkit</b>       | Noel Tomas, Arturo Jimenez Arriaga (ICE)   | Antonio Jesús Gamito González, Fernando Rendon, Luis Carrascal (Inetum), Colin Upstill (ICE)  |
| <b>IV Component Resources</b>        | Luis Carrascal (Inetum)<br>Eliot Salant (IBM)<br>Philip Usher, Arturo Jimenez Arriaga, Noel Tomas (ICE)<br>Kostas Kouvaris (IT Innovation)<br>Esteban Muncio (IMEC)<br>Farhad Aghili, Dave Singelee (KU Leuven)<br>Cilleruelo Rodríguez Carlos (UAH) | Antonio Jesús Gamito González, Fernando Rendon (Inetum), Colin Upstill (ICE)                  |
| <b>V Requirements</b>                | Luis Carrascal (Inetum)<br>Eliot Salant (IBM)<br>Philip Usher, Arturo Jimenez Arriaga, Noel Tomas (ICE)<br>Kostas Kouvaris (IT Innovation)<br>Esteban Muncio (IMEC)<br>Farhad Aghili, Dave Singelee (KU Leuven)<br>Cilleruelo Rodríguez Carlos (UAH) | Antonio Jesús Gamito González, Fernando Rendon (Inetum), Colin Upstill (ICE)                  |
| <b>VI Interoperability</b>           | Philip Usher, Noel Tomas (ICE)<br>Kostas Kouvaris (IT Innovation)<br>Eliot Salant (IBM)  | Antonio Jesús Gamito González, Fernando Rendon, Luis Carrascal (Inetum), Colin Upstill (ICE)  |
| <b>VII Recommendations from D6.2</b> | Noel Tomas (ICE)   | Antonio Jesús Gamito González, Fernando Rendon, Luis Carrascal (Inetum), Colin Upstill (ICE)  |

|                         |  |  |
|-------------------------|--|--|
| <b>VIII Conclusions</b> | Philip Usher (ICE)<br>Noel Tomas (ICE) | Antonio Jesús Gamito<br>González, Fernando<br>Rendon, Luis Carrascal<br>(Inetum), Colin Upstill<br>(ICE) |
|-------------------------|--|--|

# Table of Contents

|  |           |
|--|-----------|
| <b>I. INTRODUCTION .....</b>   | <b>10</b> |
| <b>II. PLATFORM ARCHITECTURE .....</b>                                       | <b>11</b> |
| II.1. OVERVIEW ARCHITECTURE .....  | 11        |
| II.2. SYSTEM ARCHITECTURE .....  | 12        |
| <b>III. INTEGRATION TOOLKIT .....</b>  | <b>14</b> |
| III.1. INTEGRATION TOOLKIT KUBERNETES ARCHITECTURE .....                     | 14        |
| III.2. DEPLOYMENT OVERVIEW .....   | 15        |
| III.2.1. Ansible Playbooks .....   | 15        |
| III.2.2. Kubernetes Application Cluster .....                                | 18        |
| III.2.3. Kubernetes Rancher Cluster .....                                    | 20        |
| III.2.4. Rancher .....   | 21        |
| III.2.5. Istio .....   | 24        |
| III.3. PROTEGO COMPONENTS .....  | 27        |
| III.3.1. Kubernetes Resources .....  | 27        |
| III.3.2. Helm and Rancher Charts .....                                       | 29        |
| III.4. PLATFORM DEPLOYMENTS .....  | 35        |
| III.5. INTEGRATION STEPS .....   | 36        |
| III.6. CONTINUOUS INTEGRATION / CONTINUOUS DEPLOYMENT (CI/CD) .....          | 37        |
| III.7. STORAGE .....   | 40        |
| III.8. NEXT STEPS .....  | 42        |
| III.8.1. Installation Steps .....  | 42        |
| III.8.2. Kubernetes Native Storage .....                                     | 42        |
| III.8.3. Cloud / Terraform .....   | 43        |
| III.8.4. Other Kubernetes DevOps Utilities .....                             | 43        |
| <b>IV. COMPONENT RESOURCES .....</b>   | <b>44</b> |
| IV.1.1. TEMPLATE: WP: Component Name (ACRONYM- Partner) .....                | 44        |
| IV.2. WP4: CYBERSECURITY RISK ASSESSMENT TOOLS .....                         | 44        |
| IV.2.1. WP4: System Security Modeller (SSM – IT Innovation) .....            | 45        |
| IV.2.2. WP4: Security Information and Event Management (SIEM – Inetum) ..... | 46        |
| IV.3. WP5: CYBERSECURITY RISK MITIGATION TOOLS .....                         | 47        |
| IV.3.1. WP5: Continuous Authentication (CA - UAH) .....                      | 47        |
| IV.3.1. WP5: Data Gateway (DG - IBM) .....                                   | 48        |
| IV.3.2. WP5: Access Control Framework (ACF – KU Leuven) .....                | 49        |
| IV.3.3. WP5: Network Slicing (NS - IMEC) .....                               | 51        |
| <b>V. REQUIREMENTS .....</b>   | <b>53</b> |
| V.1. INITIAL REQUIREMENTS .....  | 53        |
| V.2. INTEGRATION REQUIREMENTS .....  | 54        |
| V.2.1. WP4: System Security Modeller .....                                   | 54        |
| V.2.2. WP4: Security Information and Event Management .....                  | 54        |
| V.2.3. WP5: Continuous Authentication .....                                  | 54        |
| V.2.4. WP5: Data Gateway .....   | 55        |
| V.2.5. WP5: Access Control Framework .....                                   | 55        |
| V.2.6. WP5: Network Slicing .....  | 55        |
| V.2.7. WP6: Integration Toolkit .....  | 56        |
| V.3. RESOURCE USAGE .....  | 56        |
| V.3.1. WP4: System Security Modeller .....                                   | 56        |
| V.3.2. WP4: Security Information and Event Management .....                  | 57        |
| V.3.3. WP5: Continuous Authentication .....                                  | 59        |
| V.3.4. WP5: Data Gateway .....   | 59        |
| V.3.5. WP5: Access Control Framework .....                                   | 60        |
| V.3.6. WP5: Network Slicing .....  | 60        |
| V.3.7. WP6: Integration Toolkit .....  | 60        |

|   |           |
|---|-----------|
| <b>VI. INTEROPERABILITY .....</b>                               | <b>62</b> |
| VI.1. INTERCONNECTIONS.....                                     | 62        |
| <i>VI.1.1. Risk Assessment integration .....</i>                | <i>62</i> |
| <i>VI.1.2. Risk Mitigation Integration - Data Gateway.....</i>  | <i>64</i> |
| VI.2. DEPENDENCIES .....  | 65        |
| VI.3. INTERFACES.....   | 66        |
| <b>VII. RECOMMENDATIONS FROM D6.2 FOR FINAL PROTOTYPE .....</b> | <b>67</b> |
| VII.1. USE-CASES .....  | 67        |
| VII.2. ITERATIVE WORKFLOW .....                                 | 67        |
| <b>VIII. CONCLUSIONS .....</b>                                  | <b>68</b> |

## Table of Figures

|  |    |
|--|----|
| Figure II-1: The high-level overview of the ProTego Intermediate Prototype .....   | 11 |
| Figure II-2: This figure shows the current sub-components of the intermediate prototype and how they are arranged within the integration toolkit. .... | 12 |
| Figure III-1: Integration Toolkit Kubernetes Architecture.....   | 15 |
| Figure III-2 RKE Cluster Nodes displayed with Kubectl.....   | 20 |
| Figure III-3 K3s Cluster Nodes displayed with Kubectl .....  | 21 |
| Figure III-4 Rancher UI with list of registered clusters .....   | 23 |
| Figure III-5 List of ProTego components deployed in the cluster with updates available .....   | 24 |
| Figure III-6 Rancher ProTego catalog with list of ProTego components .....   | 34 |
| Figure III-7 Rancher UI view for component deployment setting the questions.yaml variables ..  | 35 |
| Figure III-8 ProTego CI/CD Pipeline.....   | 38 |
| Figure III-9 Jenkins Pipeline Status View .....  | 38 |
| Figure III-10 Jenkins Pipeline Approve Step.....   | 38 |
| Figure III-11 NFS Volume .....   | 41 |
| Figure IV-1: Key for Component Diagram .....   | 44 |
| Figure IV-2: High level architecture of the System Security Modeller tool.....   | 45 |
| Figure IV-3 High Level Architecture of the SIEM .....  | 47 |
| Figure IV-4: High level architecture of the Continuous Authentication Component. ....  | 48 |
| Figure IV-5: Component Level Diagram .....   | 49 |
| Figure IV-6: Subcomponents of the ACF and their high-level interfaces.....   | 51 |
| Figure IV-7: The component level diagram for the Network Slicing Component .....   | 52 |
| Figure VI-1 Information flow diagram of the integrated SSM-SIEM system. ....   | 63 |
| Figure VI-2 SIEM to SSM communication .....  | 64 |
| Figure VI-3 SSM to SIEM communication. ....  | 64 |

## List of Tables

|   |    |
|---|----|
| Table III-1: Stages of Integration .....                                    | 36 |
| Table III-2: Integration Steps for the Final Prototype .....                | 36 |
| Table III-3 List of Variables used at Jenkins version.properties file ..... | 39 |
| Table V-1: Summary of Initial Requirements from D6.1 and D6.2 .....         | 53 |
| Table V-2: Integration Requirements from the SSM .....                      | 54 |
| Table V-3: Integration Requirements from the SIEM .....                     | 54 |
| Table V-4: Integration Requirements from Continuous Authentication .....    | 54 |

---

|  |    |
|--|----|
| Table V-5: Integration Requirements from Data Gateway .....                        | 55 |
| Table V-6: Integration Requirements from Access Control Framework .....            | 55 |
| Table V-7: Integration Requirements from Network Slicing .....                     | 55 |
| Table V-8: Integration Requirements from the Integration Toolkit.....              | 56 |
| Table V-9: Computing Resource Estimates for SSM.....                               | 56 |
| Table V-10: Computing Resource Estimates for SSM-Adaptor .....                     | 57 |
| Table V-11: Computing Resource Estimates for SIEM.....                             | 57 |
| Table V-12: Computing Resource Estimates for ElasticSearch coordinating node ..... | 57 |
| Table V-13: Computing Resource Estimates for ElasticSearch master node .....       | 57 |
| Table V-14: Computing Resource Estimates for ElasticSearch data node .....         | 58 |
| Table V-15: Computing Resource Estimates for Logstash .....                        | 58 |
| Table V-16: Computing Resource Estimates for Kibana .....                          | 58 |
| Table V-17: Computing Resource Estimates for Wazuh Manager.....                    | 58 |
| Table V-18: Computing Resource Estimates for Kafka.....                            | 59 |
| Table V-19: Computing Resource Estimates for OpenVAS Stack .....                   | 59 |
| Table V-20: Computing Resource Estimates for Continuous Authentication .....       | 59 |
| Table V-21: Computing Resource Estimates for Data Gateway .....                    | 59 |
| Table V-22: Computing Resource Estimates for Access Control Framework .....        | 60 |
| Table V-23: Computing Resource Estimates for Network Slicing .....                 | 60 |
| Table V-24: Computing Resource Estimates for K3S Cluster.....                      | 60 |
| Table V-25: Computing Resource Estimates for RKE Cluster Master Node .....         | 60 |
| Table V-26: Computing Resource Estimates for RKE Cluster Worker Node .....         | 60 |
| Table V-27: Computing Resource Estimates for NFS Node .....                        | 61 |
| Table V-28: Computing Resource Estimates for Jenkins Master Node .....             | 61 |
| Table V-29: Computing Resource Estimates for Jenkins Worker Node.....              | 61 |
| Table VI-1: Dependencies for ProTego .....   | 65 |
| Table VI-2: Interfaces for ProTego .....   | 66 |

## Table of Acronyms and Definitions

| Acronym      | Definition   |
|--------------|--|
| <b>AC</b>    | Access Control   |
| <b>ACF</b>   | Access Control Framework                                   |
| <b>API</b>   | Application Programming Interface                          |
| <b>AWS</b>   | Amazon Web Services  |
| <b>CA</b>    | Continuous Authentication                                  |
| <b>CI/CD</b> | Continuous Integration / Continuous Delivery or Deployment |
| <b>CNCF</b>  | Cloud Native Computing Foundation                          |
| <b>DG</b>    | Data Gateway   |
| <b>DNS</b>   | Domain Name System   |
| <b>EC2</b>   | Elastic Compute Cloud                                      |
| <b>EDR</b>   | Endpoint Detection Response                                |
| <b>EKS</b>   | Elastic Kubernetes Service                                 |
| <b>FHIR</b>  | Fast Healthcare Interoperability Resources                 |
| <b>HTTP</b>  | Hypertext Transfer Protocol                                |
| <b>IAAS</b>  | Infrastructure As A Service                                |
| <b>IAM</b>   | Identify Access Management                                 |
| <b>IP</b>    | Internet Protocol  |
| <b>IT</b>    | Integration Toolkit  |
| <b>JWT</b>   | JSON Web Token   |
| <b>K3s</b>   | Rancher lightweight Kubernetes                             |
| <b>KEK</b>   | Key Encryption Keys  |
| <b>KMS</b>   | Key Management Service                                     |
| <b>MB</b>    | Message Bus  |
| <b>NS</b>    | Network Slicing  |



|             |   |
|-------------|---|
| <b>PAAS</b> | Platform As A Service                     |
| <b>RBAC</b> | Role Based Access Control                 |
| <b>REST</b> | Representational State Transfer           |
| <b>RKE</b>  | Rancher Kubernetes Engine                 |
| <b>SAAS</b> | Software As A Service                     |
| <b>SIEM</b> | Security Information and Event Management |
| <b>SQL</b>  | Structured Query Language                 |
| <b>SSH</b>  | Secure Shell                              |
| <b>SSM</b>  | System Security Modeller                  |
| <b>TBD</b>  | To Be Determined                          |
| <b>UI</b>   | User Interface                            |
| <b>VA</b>   | Vulnerability Assessment                  |
| <b>YAML</b> | YAML Ain't Markup Language                |

## I. INTRODUCTION

ProTego is a toolkit and guidelines to address issues around sensitive personal data in health care services. Health care services provide a rich target for malicious actors and the sensitivity and privacy cost of this type of sensitive and personal data is paramount. As well as malicious intent accidental or poor judgement in the design of the system can leave data open. ProTego aims for a toolkit and guidelines that help to manage these risks.

ProTego focuses around risk assessment and risk mitigation. Risk assessment tools, System Security Modeller, and Security Information and Event Management system will model and measure risk of the deployed application to show the risk and mitigations actions as well as allowing updates to the risk model based on data. The risk mitigation includes the component of Data Gateway, Access Control Framework, Continuous Authentication and Network Slicing. These tools look at securing the data at rest and in motion. Finally, there is an Integration Toolkit which focuses on bringing together these tools into an integrated platform for testing with the use-cases.

This document is mainly an update from previous deliverable D6.2. with the following key differences:

- Figures in Section II Platform Architecture have been updated to reflect the latest status.
  - SSM deployed inside the cluster.
  - On-Premise/On-Cloud share the same Architecture as per the case studies deployments.
- New Section III has been added with a detailed description of the Integration Toolkit, which is the main focus of WP6 work.
- Sections IV and V have been updated to reflect latest status of WP4 and WP5.
- New section on Continuous Integration is now included in Section III and previous CI/CD section has been removed.
- The recommendations section is now changed to describe how these recommendations from previous deliverable have been addressed.
- Conclusions have been updated.

## II. PLATFORM ARCHITECTURE

The architecture is described in terms of both a high-level view at the component level and a lower level view containing sub-components. The architecture diagrams show the nature of the calls between components using a Rest API, a publish/subscribe message bus or a direct call between sub-components. The architecture shows the ProTego deployment which is the same for on-premise and on-cloud deployments, the only difference being where the Integration Toolkit is installed. In some on-cloud deployments, managed Kubernetes could be used. All ProTego components are deployed in the ProTego Kubernetes cluster. The applications that integrate with ProTego, i.e., Foodcoach and PocketEHR, are deployed outside the ProTego Kubernetes cluster.

### II.1. Overview Architecture

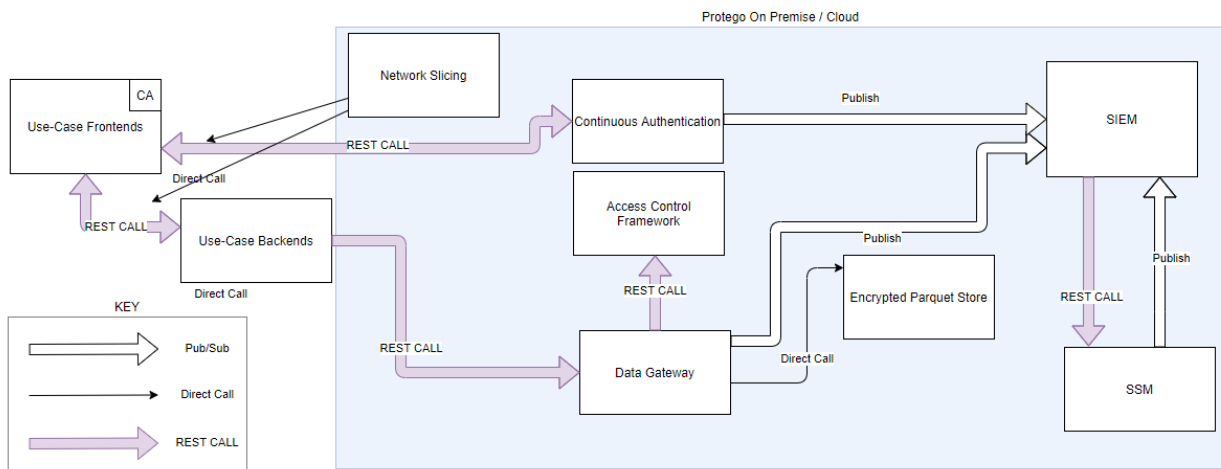


Figure II-1: The high-level overview of the ProTego Intermediate Prototype

The ProTego Final prototype (Figure II-1) consists of six main components from WP4 and WP5 integrated together using the Integration Toolkit from WP6. The Network Slicing adds a layer of complexity as it integrates at a lower level and thus provides its functionality to the connections between the components.

The Final prototype Architecture includes the plans for the Use-Cases and with implementations on-premise and on-cloud.

The ProTego components are as follows:

1. WP4: System Security modeller (SSM)
2. WP4: Security Information and Event Management (SIEM)
3. WP5: Continuous Authentication (CA)
4. WP5: Data Gateway (DG)
5. WP5: Access Control Framework (ACF)
6. WP5: Network Slicing (NS)
7. WP6: Integration Toolkit (IT)

## II.2. System Architecture

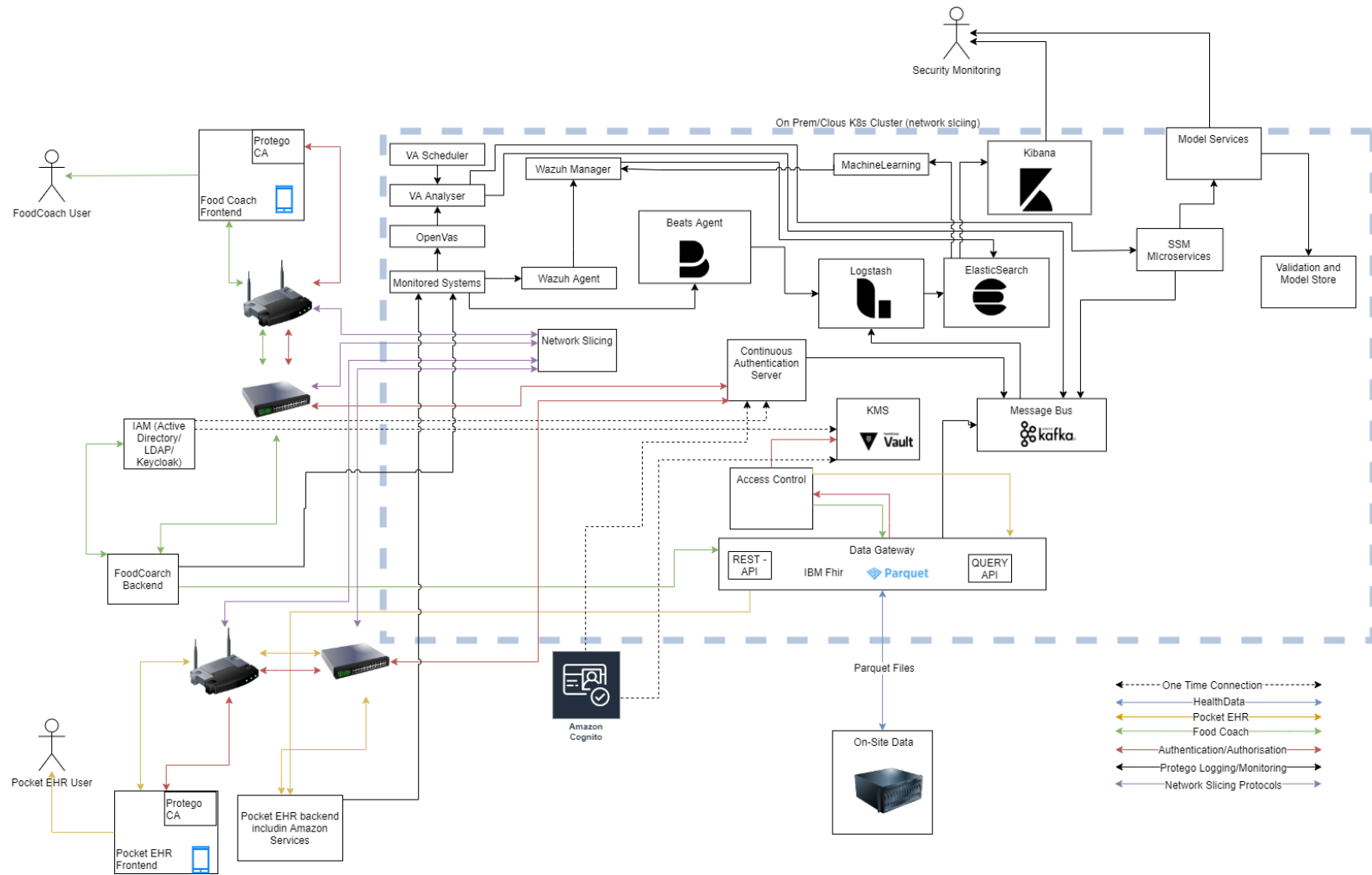


Figure II-2: This figure shows the current sub-components of the intermediate prototype and how they are arranged within the integration toolkit.

This systems level architecture (Figure II-2) shows the component broken down into their sub-component parts, and shows the nature of their connection regarding some of the key types of information transmitted through the system.

Key types of information:

- **Network Manipulation:** Specifically, for the network slicing. Requires control of specific hardware deployed in the hospital infrastructure. These control lines allow the manipulation of the network traffic directly.
- **Health Data:** Represents backend health data as treated by the ProTego components, particularly the secure transfer and data access from Parquet files, Parquet being the file format of choice for the DG.
- **Pocket EHR:** This represents the flow of data from the Pocket EHR use-case front-end to the back-end and then to the Data Gateway for secure storage and/or retrieval.
- **FoodCoach:** This represents the flow of data from the FoodCoach use-case front-end to the back-end and then to the Data Gateway for secure storage and/or retrieval.
- **Authentication/Authorisation:** Represents how user will be authorised through the system.
- **Security Logging/Monitoring:** Is the mechanism for monitoring, logging, and sending alerts to be analysed by SIEM.

The Platform is brought together with the Integration toolkit (WP6) which provides a packaging, provisioning and orchestration toolkit. More details can be found in Section III.

## III. INTEGRATION TOOLKIT

The Integration Toolkit is the key infrastructure to allow deployment and integration of the ProTego components. It is based mainly on Kubernetes and Docker technologies and provides the platform on which components are deployed and integrated. It supports the definition and use of standards for deploying and integrating components. It also provides a CI/CD pipeline supporting continuous integration/deployment of updated versions of ProTego components in a central environment, and also the publishing of these updated versions for the use-case deployments.

This Integration Toolkit was introduced in deliverables D6.1 and D6.2, but a more detailed description is provided in this deliverable, which also documents the most recent updates to the platform and describes all the work done in WP6 in this regard.

The core components of the Integration Toolkit infrastructure are:

- **Kubernetes:** leading technology in container orchestration, that manages the deployment and integration of containers. It is used as the base platform where ProTego components are deployed.
- **Docker:** leading container technology. Containers allow for applications to be run independent of the operating system environment. It allows for separate environments for each application. ProTego Toolkit components are built as Docker containers.
- **Rancher:** software that allows to deploy and manage Kubernetes clusters in a more user-friendly way both on premise and on cloud. Applications, i.e., ProTego Toolkit components, are deployed via Rancher/Helm charts to the Kubernetes clusters.
- **Helm:** the package manager for Kubernetes, applications are packaged in the form of Helm charts and deployed as a unit. By using Kubernetes yaml templates, allows multiple containers to be grouped together as well as describing the properties needed to deploy an application.
- **Istio:** service mesh technology that allows to add transparently a layer to provide the platform with enhanced connectivity, security, control and observability. It uses a sidecar container deployed along the application container to provide the service mesh features.
- **Ansible:** software for IT automation and provisioning, used to deploy the ProTego platform and its components.
- **GitLab:** a platform for software development and version control based on Git, used as Container Registry and Helm Chart catalog in ProTego.
- **Jenkins:** an automation software to build, test and deploy applications, it's the key CI/CD tool in ProTego to automatically deploy the components and provide new versions.

All these provide the base platform for the deployment and integration of the ProTego Toolkit components in an automated and repeatable way.

### III.1. Integration Toolkit Kubernetes Architecture

The Integration Toolkit is composed of two different Kubernetes clusters:

- The Rancher cluster: single node Kubernetes cluster where Rancher is deployed.
- The Application cluster: the cluster where the ProTego components are deployed. This cluster has a master and several worker nodes.

Figure III-1 shows the basic Integration Toolkit Kubernetes Architecture, with a Rancher cluster in order to manage the Application cluster and the ProTego components. Then the Application cluster, with a master node where the Kubernetes control plane runs, and some worker nodes where the ProTego components are deployed.

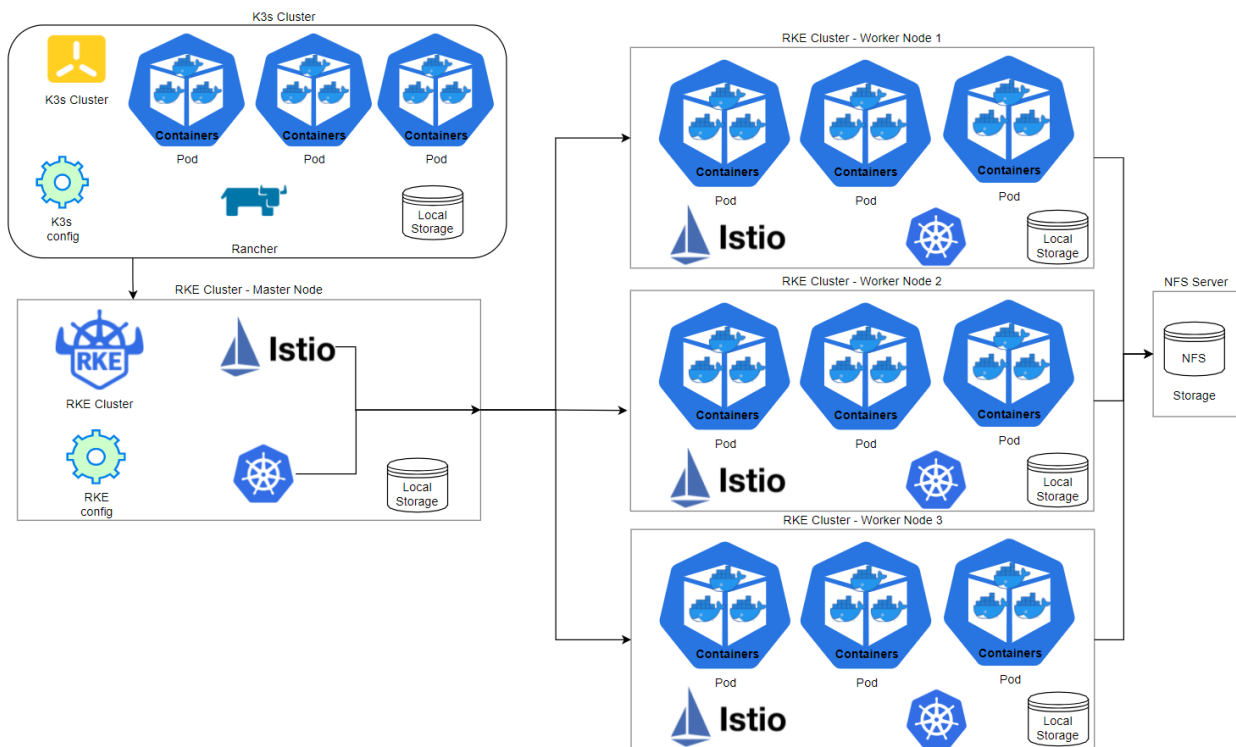


Figure III-1: Integration Toolkit Kubernetes Architecture

## III.2. Deployment Overview

In order to start using ProTego, the Integration Toolkit needs to be installed. For this installation procedure, WP6 provides several Ansible playbooks that allow for an automatic or semi-automatic deployment of the Toolkit. Detailed instructions on how to do this are also provided for the case studies. The basic steps for Integration Toolkit deployment, using the Ansible playbooks, are:

- Check/Install pre-requirements (Ansible).
- Install Kubernetes Application Cluster [RKE].
- Install Kubernetes Rancher Cluster [K3s].
- Deploy Rancher to the Rancher Cluster.
- Register the Application Cluster in Rancher.
- Deploy Istio.
- Deploy the ProTego Toolkit components.
- Deploy the CI/CD Pipeline.

In the next sections there is a brief description of each of these installation steps related to the key technologies being used and how they are being used in ProTego.

### III.2.1. Ansible Playbooks

Several ansible playbooks have been developed and provided to deploy the Integration Toolkit in any environment.

These playbooks are used to perform all the Integration Toolkit installation steps automatically or semi-automatically.

Ansible is an IT automation engine that allows for automated provisioning and configuration management and other IT tasks. It uses no agents so it is easy to deploy, and uses playbooks expressed in yaml which allows the automation tasks to be described in a declarative way.

Ansible works by connecting to the nodes, where the ProTego platform is going to be installed, and running small programs called “Ansible modules” as per Ansible playbook definitions. Ansible modules execute tasks. One or more tasks can be combined to make a play. Two or more plays can be combined to create a playbook. Ansible playbooks are lists of tasks that automatically execute against specific hosts. Groups of hosts form the Ansible inventory.

Ansible uses a single node, where it is installed. It uses ssh authorized keys to connect to the nodes where the playbooks will be run. The Ansible node can be any machine, even a user laptop, provided ssh keys are established.

Ansible uses a hosts configuration file describing how nodes are configured in groups to perform specific tasks, e.g., a user may configure a group of Kubernetes Application Cluster nodes and perform only certain tasks on such a node group. This is an example of the hosts configuration file for the Integration Toolkit installation:

```
[ansible]
ansible.protego

[rke]
master.protego
worker01.protego
worker02.protego

[rkemaster]
master.protego

[k3smaster]
rancher.protego

[rancher]
rancher.protego

[docker]
master.protego
worker01.protego
worker02.protego
rancher.protego
```

In the above example, several groups of hosts have been configured: rke, rkemaster, etc. Then these group labels are used in the playbooks to specify which tasks are run on which groups of hosts. This file would be configured according to the target ip addresses or the hostnames of the nodes of the on-premise/on-cloud deployment where the Integration Toolkit is going to be installed.



This is a sample of the Ansible playbook that performs the necessary initial steps to install the Integration Toolkit :

```
- hosts: all
  remote_user: root
  tasks:
# Create rancher group
  - name: create rancher group
    group:
      name: rancher
# Create rancher user
  - name: create rancher user
    user:
      name: rancher
      password: ""
      shell: /bin/bash
      home: /home/rancher
      group: rancher
      groups: sudo
      append: yes
      generate_ssh_key: yes
# Copy all ssh rancher public keys
  - name: fetch all id_rsa public
    fetch:
      src: /home/rancher/.ssh/id_rsa.pub
      dest: id_rsa.pub.{{ inventory_hostname }}
      flat: yes
# Copy the installation scripts
  - name: copy the installation scripts
    copy:
      src: /root/rancher_install
      dest: /home/rancher/
      owner: rancher
      group: rancher
      mode: '0644'
# Add the ssh keys to the authorized hosts
  - name: add ssh keys to the authorized hosts
    shell: cat /home/rancher/rancher_install/id_rsa.pub.{{
item }} >> /home/rancher/.ssh/authorized_keys
    args:
```

```
    executable: /bin/bash
    loop: "{{ query('inventory_hostnames','all') }}"
# Copy rancher sudoers file
- name: copy rancher sudoers file
  copy:
    src: /home/rancher/rancher_install/rancher
    dest: /etc/sudoers.d/
    mode: '0440'
    remote_src: yes
...
```

In order to run the Ansible playbook, a user needs to install Ansible. An installation script has been provided by WP6 in order to perform the Ansible installation in one node. In order to run a playbook, user needs to run the command:

```
ansible-playbook playbook-name.yaml -i hosts
```

where `playbook-name.yaml` is the playbook file name and `hosts` is the inventory file.

WP6 provides a set of Ansible playbooks for the case studies and possible future customers in order to install and deploy the Integration Toolkit. The list of provided Ansible playbooks is:

- `install-user.yaml` | `install-user_root.yaml`: this playbook creates the rancher user for the installation and distributes the ssh keys
- `install-docker.yaml`: this playbook creates the basics for the platform. Performs some pre-reqs and installs Docker.
- `install-rke.yaml`: this playbook prepares the nodes with the prerequisites and installs the Kubernetes cluster (RKE). In addition, the playbook installs and configures `kubecttl`, a software tool for Kubernetes command line management, and Helm.
- `install-certmanagerk8s.yaml`: this playbook deploys Certmanager in the Kubernetes cluster. Certmanager is a requirement for Rancher and is a software tool used to issue ssl certificates.
- `install-rancherk8s.yaml`: this playbook deploys Rancher in the Kubernetes cluster.
- `install-k3s.yaml`: this playbook deploys a K3s cluster.

Some additional scripts are provided that are run from the Ansible playbooks. In addition, detailed instructions on how to run those playbooks are also provided.

### III.2.2. Kubernetes Application Cluster

The Kubernetes Application Cluster is the Kubernetes Cluster where the other ProTego components will be deployed.

As already mentioned, Kubernetes is the leading technology in container orchestration. There are several distributions that can be used to install a Kubernetes cluster. In this case, for the Integration Toolkit at ProTego, and for the Application Cluster, the RKE distribution from Rancher has been selected.

RKE is a CNCF-certified Kubernetes distribution that runs entirely within Docker containers. That is, all the Kubernetes components, including the control plane components, which are the key ones making global decisions about the cluster, run as Docker containers. RKE is a production-grade Kubernetes distribution that removes installation complexity with Kubernetes by removing most host dependencies and presenting a stable path for deployment, upgrades and rollbacks. With RKE, the operation of Kubernetes is easily automated and entirely independent of the operating system and platform. If you can run Docker, you can deploy and run Kubernetes with RKE, building a cluster from a single command in minutes. In addition, its declarative configuration makes Kubernetes upgrades atomic and safe.

This is an extract of the Ansible playbook used to deploy the RKE cluster:

```
- hosts: rke
  remote_user: rancher
  become: yes
  become_user: root
  become_method: sudo
  tasks:
# Prepare the nodes for rke installation
- name: prepare node
  script: /home/rancher/rancher_install/prepare_node.sh
  args:
    creates: /home/rancher/rancher_install/prepare_node.lock
  when:
    - ansible_facts['distribution'] == "Ubuntu"
- hosts: rkemaster
  remote_user: rancher
  tasks:
# Copy the cluster config file to the master node
- name: copy cluster config file to master node
  copy:
    src: /home/rancher/rancher_install/cluster.yml
    dest: /home/rancher/rancher_install/cluster.yml
    mode: '0644'
# Install rke: the cluster.yaml
- name: chmod +x rke file
  file:
    path: /home/rancher/rancher_install/rke
    mode: '0755'
  tags:
    - rke
- name: install rke - run rke
  shell: |
```

```
    ./rke up
    echo "rke" > rke.lock
args:
  executable: /bin/bash
  chdir: /home/rancher/rancher_install/
  creates: /home/rancher/rancher_install/rke.lock
register: result
- name: debug rke
  debug:
    var: result
- name: create .kube directory for k8s
  file:
    path: /home/rancher/.kube
    state: directory
- name: copy config file
  copy:
    src:
/home/rancher/rancher_install/kube_config_cluster.yml
    dest: /home/rancher/.kube/config
    mode: '0644'
    remote_src: yes
...
```

Once RKE is deployed along with some Kubernetes tools like kubectl to manage the cluster, the status of the cluster can be monitored with kubectl command line utility, as shown in Figure III-2:

```
rancher@master:~$ kubectl get nodes
NAME                 STATUS    ROLES    AGE     VERSION
192.168.143.100     Ready    controlplane,etcd  362d   v1.17.6
192.168.144.100     Ready    worker    362d   v1.17.6
192.168.144.101     Ready    worker    362d   v1.17.6
rancher@master:~$
```

Figure III-2 RKE Cluster Nodes displayed with Kubectl

Kubernetes uses namespaces for application distribution and isolation. Namespaces provide a scope for names and are a way to distribute cluster resources. For ProTego, each WP4 and WP5 component has been assigned to a separate namespace, i.e., the SIEM component is deployed in the siem namespace and so on.

### III.2.3. Kubernetes Rancher Cluster

For the Kubernetes cluster where Rancher is deployed, K3s is the Kubernetes distribution selected.

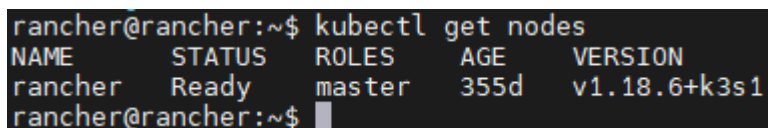
K3s is a lightweight fully-compliant Kubernetes distribution, packaged in a single binary. It uses a lightweight storage backend based on sqlite3, but other storage backends are available including etcd which is the usual Kubernetes storage. In ProTego it is deployed using sqlite3.

K3s is wrapped in simple launcher that handles a lot of the complexity of a Kubernetes installation, much like RKE but smaller and less complex. Operation of all Kubernetes control plane components is encapsulated in a single binary and process.

This is an extract of the playbook used to install K3s cluster:

```
- hosts: k3smaster
  vars:
    docker: true
    remote_user: rancher
  tasks:
# Install k3s docker runtime
  - name: install k3s docker runtime
    shell: |
      curl -sfL https://get.k3s.io | sh -s - --write-
      kubeconfig-mode 644 --node-ip {{ inventory_hostname }} --node-
      external-ip {{ inventory_hostname }} --docker
      echo "k3s" > k3s.lock
    args:
      executable: /bin/bash
      chdir: /home/rancher/rancher_install/
      creates: /home/rancher/rancher_install/k3s.lock
    register: result
    when: docker|bool
  - name: debug k3s docker runtime
    debug:
      var: result
  ...
```

Once K3s is installed along with kubectl, you can check status of the cluster like in the RKE cluster using kubectl, as shown in :



```
rancher@rancher:~$ kubectl get nodes
NAME        STATUS    ROLES    AGE   VERSION
rancher    Ready    master   355d  v1.18.6+k3s1
rancher@rancher:~$
```

Figure III-3 K3s Cluster Nodes displayed with Kubectl

### III.2.4. Rancher

Rancher is a complete platform to run Kubernetes clusters on-premises or on-cloud that fits with multi-cluster, hybrid or multi-cloud container orchestration strategy.

Rancher is an application that is deployed on a Kubernetes cluster. In ProTego, a K3s cluster is being used, and allows installation and configuration of multiple Kubernetes clusters on-premise or in the cloud. Once up and running, Rancher allows for day-2 operations management in a much more user-friendly way:

- Control access to clusters using RBAC.
- Deploy apps and multi-cluster apps from Rancher catalogs (Helm catalogs).
- Monitor workloads.
- Get notifications.
- Connect clusters to CI/CD pipelines.

In addition, Rancher also integrates some popular open-source projects like Prometheus (monitoring and alerting), Grafana (data dashboards), Fluentd (log collection) and Istio (Service Mesh). That means users are able to deploy such tools easily from the Rancher UI if necessary.

Best practices recommend deploying Rancher in a dedicated Kubernetes cluster, hence the Integration Toolkit architecture is comprised of two clusters, the Rancher cluster for the Rancher application and the Application cluster where all the ProTego components are deployed.

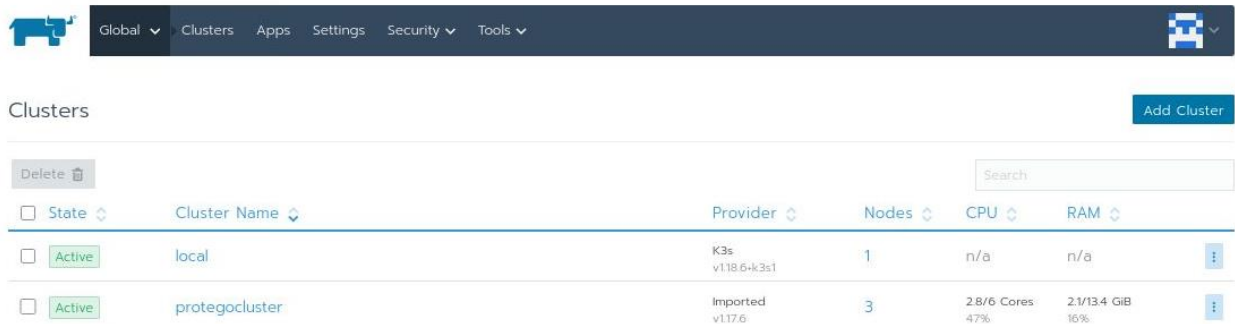
This is the Rancher installation playbook:

```
- hosts: rancher
  remote_user: rancher
  tasks:
    # Install rancher
    # helm install rancher rancher-stable/rancher --namespace
    cattle-system --set hostname=rancher.localhost
    # helm install rancher rancher-stable/rancher --namespace
    cattle-system --set hostname=rancher.kubernetes --set
    ingress.tls.source=letsEncrypt --set
    letsEncrypt.email=noel.tomas@informationcatalyst.com
    - name: install rancher
      shell: |
        helm repo add rancher-stable
        https://releases.rancher.com/server-charts/stable
        kubectl create namespace cattle-system
        helm repo update
        helm install rancher rancher-stable/rancher --
        namespace cattle-system --set hostname=rancher.kubernetes
        echo "rancher" >
        /home/rancher/rancher_install/rancher.lock
      args:
        executable: /bin/bash
        creates: /home/rancher/rancher_install/rancher.lock
        register: result
    - name: debug rancher
      debug:
```

```
var: result
```

Once Rancher is deployed in the corresponding cluster, and having the RKE Application cluster, it is only a matter of registering the Application cluster in Rancher so it can be managed from the Rancher UI. Users can perform most of the common Kubernetes administration tasks using the Rancher UI. Rancher could also be used to manage all test and production clusters in a real production scenario.

Figure III-4 represents the Rancher UI with the local cluster, the k3s Rancher cluster, and the ProTego cluster, the RKE Application cluster.



The screenshot shows the Rancher UI interface. At the top, there is a navigation bar with the Rancher logo and menu items: Global, Clusters, Apps, Settings, Security, and Tools. Below the navigation bar, the main content area is titled 'Clusters' and includes an 'Add Cluster' button. A search bar is located at the top right of the cluster list. The cluster list is a table with columns for State, Cluster Name, Provider, Nodes, CPU, and RAM. Two clusters are listed: 'local' and 'protegocluster'.

| State  | Cluster Name   | Provider            | Nodes | CPU                | RAM                 |
|--------|----------------|---------------------|-------|--------------------|---------------------|
| Active | local          | K3s<br>v1.18.6-k3s1 | 1     | n/a                | n/a                 |
| Active | protegocluster | Imported<br>v1.17.6 | 3     | 2.8/6 Cores<br>47% | 2.1/13.4 GIB<br>16% |

Figure III-4 Rancher UI with list of registered clusters

One of the main tasks that can be performed using the Rancher UI is the deployment of applications, i.e., the ProTego components, in the Kubernetes cluster. The Rancher UI and its App/Catalog view also allow users to get notifications on available upgrades of ProTego components. Then it is up to the users to decide when and how to perform the upgrades, usually deploying it first to a test cluster and, once tested, moving to the production cluster. Users are also able to review application logs and other app related tasks using this UI.

Figure III-5 represents the list of ProTego deployed components and updates available.

| Component Name      | Ports   | Status | Upgrade Available         | Active |
|---------------------|---|--------|---------------------------|--------|
| ca-jbca             | 30150/tcp, 30232/tcp  | 2      | Upgrade available (10.8)  | Active |
| elsfhirchart        | 30443/tcp   | 1      | Upgrade available (0.50)  | Active |
| kul-kmschart        |   | 1      | Up to date (0.5.9)        | Active |
| network-slicing     | 30201/tcp, 30202/tcp, 30203/tcp, 30205/tcp, 30206/tcp, 30207/tcp, 30208/tcp, 30209/tcp  | 2      | Upgrade available (10.3)  | Active |
| openvas-slave       | 30333/tcp   | 1      | Up to date (0.0.1)        | Active |
| querygwchart        | 30905/tcp   | 1      | Upgrade available (0.6.0) | Active |
| siem                | 80/http, 30017/tcp, 30144/tcp, 30244/tcp, 30344/tcp, 30444/tcp, 30514/udp, 30544/tcp, 30550/tcp, 30644/tcp, 30960/tcp, 31514/tcp, 31515/tcp | 12     | Upgrade available (10.16) | Active |
| ssm-adaptor         | 30100/tcp, 32154/tcp  | 2      | Upgrade available (0.13)  | Active |
| ssm-system-modeller | 30125/tcp, 30330/tcp  | 4      | Up to date (10.3)         | Active |

Figure III-5 List of ProTego components deployed in the cluster with updates available

Rancher has an additional grouping feature called projects where namespaces can be grouped and several policies can be applied at this level, allowing, for instance, separation of concerns for Kubernetes administrators, and ProTego application administrators if it is a requirement. In addition, Kubernetes RBAC can also be used in Rancher, so fine-grained user privileges can be set for different users, clusters or projects.

### III.2.5. Istio

Once all clusters and Rancher are set up, an Istio Service Mesh is deployed in the Application cluster.

A Service Mesh is a dedicated infrastructure layer that can be added to applications and allows transparent addition of features like observability, traffic management and security without adding them to the application code.

Istio is an open source Service Mesh that layers transparently onto existing distributed applications. Istio provides a uniform and more efficient way to secure, connect and monitor services. Istio is the path to load balancing, service-to-service authentication and monitoring, with few or no service code changes.

Istio is deployed into a Kubernetes cluster as any other application or component. You can use Rancher's out of the box integration of Istio in order to deploy Istio just by clicking a button, but in ProTego, a more fine-tuned deployment has been selected, in order to deploy the specific components required.

This is an extract of the Istio installation playbook:



```
- hosts: rkemaster
  remote_user: rancher
  tasks:
# Install istio
  - name: install istio
    shell: |
      istioctl install
      echo "istio" >
/home/rancher/rancher_install/istio.lock
    args:
      executable: /bin/bash
      creates: /home/rancher/rancher_install/istio.lock
    register: result
  - name: debug rancher
    debug:
      var: result
...
```

### III.2.5.a. How Istio works

Istio has two components: the data plane and the control plane.

The data plane is the communication between services. Istio uses an Envoy proxy deployed along with each service, as a sidecar container, that intercepts network traffic and allows a broad set of application-aware features based on the configuration set.

The control plane takes desired configuration and its view of the services, and dynamically programs the proxy servers, updating them as the rules of the environment change.

Once Istio is deployed in the Kubernetes cluster, it needs to be enabled per namespaces, just by labeling the namespace:

```
kubectl label namespace default istio-injection=enabled
```

This label configures Istio to deploy a sidecar container with every component/service deployed in this namespace, thus being able to use Istio features.

This is also an advantage since users can select which applications use Istio just by labeling or not labeling specific namespaces.

For ProTego, all component namespaces are Istio enabled in order to join the Istio Service Mesh.

### III.2.5.b. Istio and ProTego

Istio in ProTego is basically being used for Traffic Management and Security.

For traffic management, two key resources are used:

- Istio Gateway: to manage inbound and outbound traffic for the Istio Service Mesh, letting users specify which traffic is allowed to enter the mesh. Unlike other mechanisms for controlling traffic entering systems, such as the Kubernetes Ingress APIs, Istio gateways

allow the usage of Istio's traffic routing capabilities. Istio's Gateway resource allow layer 4-6 load balancing configurations such as ports to expose, TLS settings, etc. Then, instead of adding application-layer traffic routing (L7) to the same resource, an Istio Virtual Service is bound to the Gateway. This allows management of Gateway traffic like any other data plane traffic in the mesh. This is an example of an Istio Gateway resource:

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: httpbin-gateway
spec:
  selector:
    istio: ingressgateway # use Istio default gateway
  implementation
  servers:
    - port:
        number: 80
        name: http
        protocol: HTTP
      hosts:
        - "httpbin.example.com"
```

- **Virtual Service:** allows configuration of how requests are routed to a service within an Istio Service Mesh, building on the basic connectivity and discovery provided by Istio and Kubernetes. Each virtual service consists of a set of routing rules that are evaluated in order, letting Istio match each given request to the virtual service to a specific real destination within the mesh. This is an example of an Istio Virtual Service:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin
spec:
  hosts:
    - "httpbin.example.com"
  gateways:
    - httpbin-gateway
  http:
    - match:
        - uri:
            prefix: /status
      - uri:
```

```
    prefix: /delay
route:
- destination:
    port:
        number: 8000
    host: httpbin
```

In the ProTego deployment, a Gateway is deployed per component that needs to be exposed outside of the cluster. Then a Virtual Service related to that component is bound to that Gateway to direct traffic to the component. Those resources are packaged in the component Helm chart and are created when the component is deployed.

Security in Istio relates to Certificate Management, Authentication and Authorization. For ProTego the focus has been mainly on the Authentication, and more specifically on the Mutual TLS feature that is deployed out of the box with Istio.

Istio automatically configures workload sidecars to use Mutual TLS when calling other workloads in the mesh. That means that all traffic between sidecars is encrypted, thus providing an extra layer of security inside of the cluster. In addition, if any component needs to manage TLS, the passthrough method of the Gateway resource can be used, in order not to perform the SSL/TLS termination at the Gateway level.

### III.3. ProTego Components

Once the basic platform for the Integration Toolkit is installed and configured, the WP4 and WP5 components have to be deployed to the Application cluster. To this end, the components are built as Docker containers, and the corresponding Kubernetes resource yaml files have to be designed and then packaged as Helm/Rancher charts.

Basic guidelines and instructions have been provided to partners on basic Kubernetes resources and how to package them in Helm charts. Best practices have also been provided for this.

#### III.3.1. Kubernetes Resources

In Kubernetes, a workload is an application that runs on the cluster. Whether this workload is a single component or several that work together, on Kubernetes they run inside a set of Pods. Then workload resources can be used on top of Pods that will manage those Pods on user behalf. These resources configure controllers that make sure the right number of the right kind of Pods are running to match the state specified. These resources, like all Kubernetes objects, are described using a declarative API with yaml files.

For ProTego, the key resources for Kubernetes deployments are Pod, Deployment and Service, even though some components may have some specific requirements for other resources, like Stateful or Daemon sets.

##### III.3.1.a. Pod

Pods are the smallest deployable units of computing that can be created and managed in Kubernetes. A Pod is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers.

A Deployment provides declarative updates for Pods and ReplicaSets (a controller for Pods). User describes a desired state in a Deployment and the Deployment controller changes the actual state to the desired state at a controlled rate.

This is an example of a Deployment yaml file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

### III.3.1.b. Service

An abstract way to expose an application running on a set of Pods as a network service, Kubernetes gives Pods their own IP addresses, with a single DNS name for a set of Pods, and can load-balance across them.

This is an example of a Service yaml file:

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
```

```
targetPort: 9376
```

Other resources can also be used, such as ConfigMap for configuration variables, Secrets for secret management, PersistentVolumes for persistent storage, PersistentVolumeClaims to claim a PersistentVolume and use it in a Pod for data persistence, etc. In addition, Istio resources are also used for traffic ingress, routing and for security enhancement.

Once all resource yaml files are configured, they are packaged or bundled in a Helm chart, using the values.yaml file to replace template values during deployment time.

### III.3.2. Helm and Rancher Charts

Helm is the package manager for Kubernetes. Helm helps to manage Kubernetes applications using Helm charts. Helm charts help to define, install and upgrade Kubernetes application. Charts describe even the most complex apps, provide repeatability, and serve as single point of authority. In addition, charts are easy to version, share and host on public or private servers.

A helm chart is a collection of files that describe a related set of Kubernetes resources. A single chart might be used to deploy a simple app such as a single nginx pod or a more complex app such as a full web app stack with HTTP servers, databases, frontend, backend, and so on.

Charts are created as files in a particular directory tree and can be packaged into versioned archives to be deployed. The directory name is the name of the chart, without the versioning information.

A sample Helm chart structure for an app:

app/

```
Chart.yaml: a yaml file containing information about the chart
README.md (optional): a human-readable readme file
values.yaml: the default configuration values for this chart
charts/ : directory containing any charts upon which this chart depends
templates/ : directory of templates, that is the templates of the
              Kubernetes resource files that, combined with values,
              generate valid Kubernetes manifest files
```

A Helm chart uses the files in the templates/ directory as templates of Kubernetes resources for the application. Then the values file is used to render valid Kubernetes resources from the templates during deployment time.

In ProTego, Rancher charts - which are a superset of Helm charts - are used to deploy applications to the Kubernetes cluster using Rancher.

Rancher charts are very similar to Helm charts, differing only in the directory structure, which includes an app version directory charts/app/version/ ... and two additional files :

- app-readme.md: a text-based file that provides a high level overview display in the Rancher UI.
- questions.yaml: matches any values.yaml variable that needs or requires to be displayed in the Rancher UI during deployment, so that a user can set specific values for specific variables during deployment time from the Rancher UI.

This is an extract of the Helm chart for the SIEM, that shows contents of values.yaml and the kafka-deployment.yaml.

values.yaml:

```
# Secrets -----
defaultSettings:
  registrySecret: "siem-cred"
privateRegistry:
  registryUrl: registry.gitlab.com
  registryUser: "protego2020"
  registryPasswd: ""

#Image features
image:
  policy: IfNotPresent
  repository: registry.gitlab.com/protego2020/
  elastic: siem/elastic:1.0.0
  logstash: siem/logstash:1.0.2
  kibana: siem/kibana:1.0.0
  wazuh: siem/wazuh-manager:1.0.1
  kafka: siem/kafka:1.0.0
  zookeeper: siem/zookeeper:1.0.0
  gvm: siem/gvm:1.0.1
  openvas: siem/openvas:1.0.1
  postgres: siem/postgres:1.0.0
  gsa: siem/gsa:1.0.0
  analyzer: siem/analyzer:1.0.2
  scheduler: siem/scheduler:1.0.2
  busybox: siem/busybox:1.0.0

# Apps to deploy
app:
  elastic: elastic
  logstash: logstash
  kibana: kibana
  wazuh: wazuh
  kafka: kafka
  zookeeper: zookeeper
  gvm: gvm
  openvas: openvas
  postgres: postgres
```

```
gsa: gsa
analyzer: analyzer
scheduler: scheduler

replicas:
  elastic: 1
  logstash: 1
  kibana: 1
  wazuh: 1
  kafka: 1
  zookeeper: 1
  gvm: 1
  openvas: 1
  postgres: 1
  gsa: 1
  analyzer: 1
  scheduler: 1

#Ports exposed
port:
  elastic: 9200
  logstash:
    api: 9600
    filebeat: 5144
    packetbeat: 5244
    winlogbeat: 5344
    heartbeat: 5444
    metricbeat: 5544
    auditbeat: 5644
    analyzer: 9563
  kibana: 5601
  wazuh:
    api: 55000
    agents: 1514
    registration: 1515
    syslog: 514
  kafka: 9092
  zookeeper: 2181
  gvm: 9390
```

```
openvas: 51234
postgres: 5432
gsa: 9392
```

**kafka-deployment.yaml:**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: {{ .Values.app.kafka }}
  name: {{ .Values.app.kafka }}
spec:
  selector:
    matchLabels:
      app: {{ .Values.app.kafka }}
  replicas: {{ .Values.replicas.kafka }}
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: {{ .Values.app.kafka }}
    spec:
      initContainers:
        - name: wait-for-zookeeper
          image: {{ .Values.image.repository }}{{
.Values.image.busybox }}
          command: ["/bin/sh", "-c", "until nc -z -w 3 {{
.Values.app.zookeeper }} {{ .Values.port.zookeeper }}; do echo
'Waiting for Zookeeper...'; sleep 5; done"]
      containers:
        - env:
            - name: KAFKA_ADVERTISED_HOST_NAME
              value: {{ .Values.app.kafka }}
            - name: KAFKA_ADVERTISED_LISTENERS
              value: PLAINTEXT://{{ .Values.app.kafka }}:{{
.Values.port.kafka }}
            - name: KAFKA_PORT
              value: {{ .Values.port.kafka | quote }}
            - name: KAFKA_CREATE_TOPICS
```



```

    value: SSM:1:1,CA:1:1,GW:1:1,MS:1:1
  - name: KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR
    value: "1"
  - name: KAFKA_ZOOKEEPER_CONNECT
    value: {{ .Values.app.zookeeper }}:{{
.Values.port.zookeeper }}
    image: {{ .Values.image.repository }}:{{
.Values.image.kafka }}
    imagePullPolicy: {{ .Values.image.policy }}
    name: {{ .Values.app.kafka }}
    ports:
      - containerPort: 9092
nodeSelector:
  numNode: {{ .Values.selector.kafka | quote }}
restartPolicy: Always
{{- if .Values.defaultSettings.registrySecret }}
imagePullSecrets:
  - name: {{ .Values.defaultSettings.registrySecret }}
{{- end }}

```

Helm or Rancher charts in Rancher are deployed using Catalogs. A Catalog is a Git or Helm repository filled with Helm Charts ready to be deployed.

In ProTego, two main catalogs are being used. The integration catalog is used in the Integration ProTego deployment for development and testing of the different components. The production catalog is where the Helm charts get pushed once they are validated in the integration environment.

Both catalogs use GitLab as the git repository, the integration one being an on-prem deployment of GitLab, and the production one being the GitLab cloud SaaS free tier offering. This Production catalog is then registered in all the ProTego environments, i.e., OSR, MS. Once components are pushed to the GitLab cloud repository, they can be deployed or upgraded in those use-case environments.

Figure III-6 shows the ProTego catalog with the list of ProTego components:

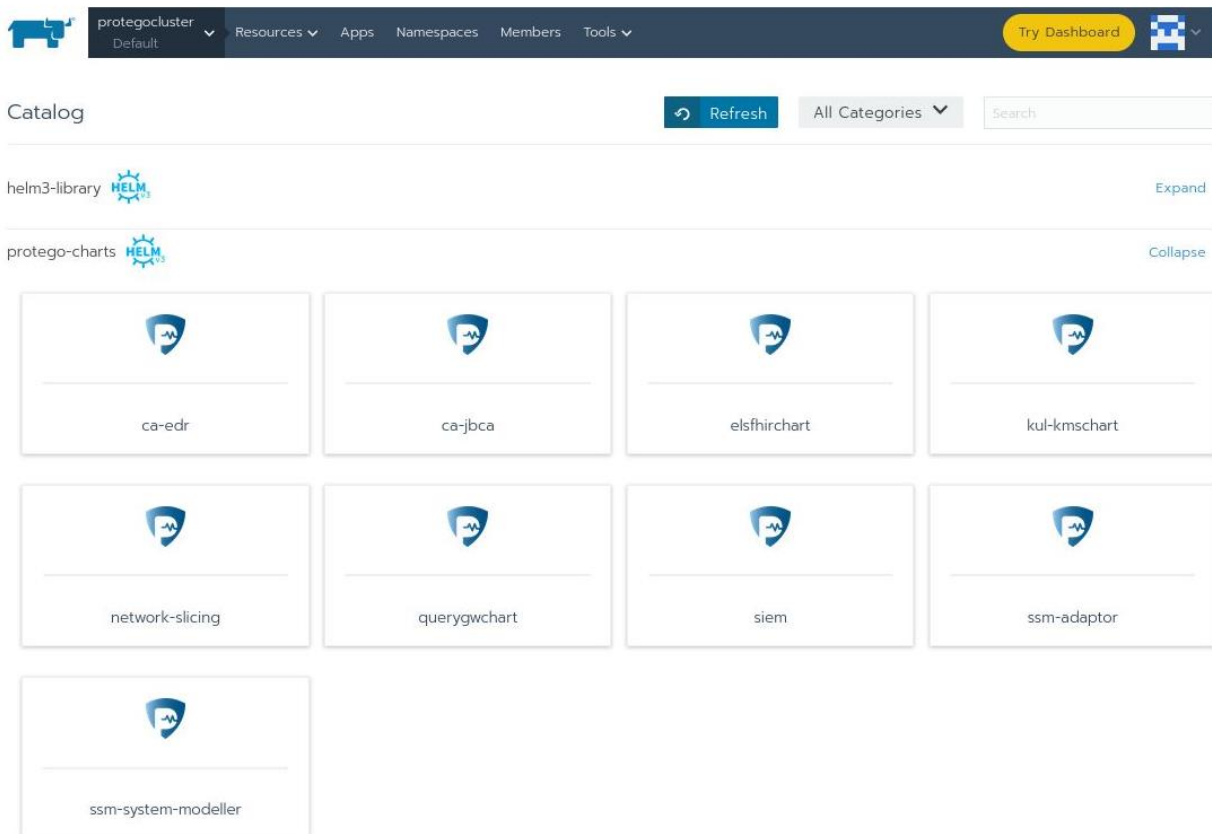


Figure III-6 Rancher ProTego catalog with list of ProTego components

The WP4 and WP5 components can be deployed using the Rancher UI one by one. That allows users to deploy the different components selectively and start integrating them in separate steps into their current software stack, and also allows customisation of specific settings for the components. That customisation can be set through the Rancher UI, using the variables from the questions.yaml file in the component chart.


Figure III-7 represents the deployment of a ProTego component from the Rancher UI setting some of the questions.yaml variables:

protegocluster  
Default

Resources Apps Namespaces Members Tools

Try Dashboard

Catalog: elsfhirchart

 Protego Data Gateway App Helm Chart

Prototype versions of a Data Gateway for both securely storing and querying FHIR data, using Parquet modular encryption.

Expand All

Detailed Descriptions  
Application information and user guide

Configuration Options  
Paste and Read actions require their respective answers to be in a yml/yaml format

Name [Add a Description](#) Template Version

elsfhirchart-hht2j 0.5.0

Select a version of the template to deploy

NAMESPACE

Namespace \* [Use an existing namespace](#)

elsfhirchart-79hgc

HELM OPTIONS

Helm Wait

True  False

Helm Timeout

300 seconds

Waits until all Kubernetes resources are ready before marking the app as active. If the timeout value is exceeded the application install will fail.

A value in seconds to wait for Kubernetes commands to complete.

[Edit as YAML](#)

Figure III-7 Rancher UI view for component deployment setting the questions.yaml variables

## III.4. Platform Deployments

One of the main tasks in WP6 has been providing support for partners in the project to :

- Design the components Helm charts and deploy them to the Integration Platform
- Integrate the WP4 and WP5 components
- Help for case studies partners to deploy the Integration Toolkit
- Help for case studies and WP4 and WP5 partners to deploy the components into the use-case platform

Regarding the platform deployment, three different environments have been provisioned with the Integration Toolkit and ProTego:

- Integration Environment: the main Integration ProTego deployment where WP4 and WP5 partners are able to deploy and test their components and integrations.
- OSR: deployment of ProTego at OSR facilities for the OSR case study, Foodcoach. This is an on-premise deployment using infrastructure provided by OSR. Deployment of the Integration Toolkit was mainly conducted by OSR Staff using Ansible playbooks and instructions provided from WP6 in the scope of WP7. Support to deploy the ProTego components from WP4 and WP5 was also provided, not only from WP6 but also from WP4 and WP5. Some troubleshooting was also performed due to some issues that arose when installing the Integration Toolkit and some components.

- Marina Salud: deployment of ProTego for the Marina Salud case study, PocketEHR. This is a fully on-cloud deployment using Amazon AWS. For this deployment, Marina Salud provisioned an EC2 instance for the Rancher cluster that was deployed using the Ansible playbooks. Then Marina Salud provisioned and registered in Rancher an EKS cluster, which is a managed Kubernetes cluster from Amazon AWS. As in the OSR deployment, support was also provided for WP4 and WP5 component installation.

The architecture in this case is exactly the same as the on-premise deployments, the main difference is that the Application cluster is a managed service from Amazon, so users do not have to bother about maintenance of that cluster. The Rancher cluster is still deployed to a virtual machine, an EC2 machine from Amazon, so that cluster is managed by the user. Deployment of Rancher and of the ProTego WP4 and WP5 components is performed in the same way as in an OnPrem deployment. This means that, once having access to Rancher, there is no difference between using an on-premise or an on-cloud deployment; the Rancher interface for Kubernetes and the Integration Toolkit is the same.

### III.5. Integration Steps

These tables show the integration status at this stage of the project.

A set of stages of the integration were defined. All integration steps have been completed.

Table III-1: Stages of Integration

| Stage                          | Step  | Status    |
|--------------------------------|---|-----------|
| Stage 1: Component Assembly    | Base Platform                                       | Completed |
|                                | Guidance Notes                                      | Completed |
|                                | Vanilla Projects                                    | Completed |
| Stage 2: Component Integration | Vanilla Project Swap outs for Partner Contributions | Completed |
|                                | Application Integrations                            | Completed |
|                                | SSM Stub APIs                                       | Completed |
|                                | Platform Migration                                  | Completed |
|                                | Integration Guidance                                | Completed |
|                                | Network Slicing Implementation                      | Completed |

Steps have been broken down further into an integration plan for the ProTego toolkit.

Table III-2: Integration Steps for the Final Prototype

| Step # | Description  | Status    |
|--------|--|-----------|
| 1      | Dockerise all components   | Completed |
| 2      | Create an orchestration file Helm Chart, or Docker compose yaml for each component                 | Completed |
| 3      | Integrate each component with its own version of the platform (using a local version with vagrant) | Completed |
| 4      | Deploy all components on a central system  | Completed |
| 5      | Integration between components within WP   | Completed |

|   |   |           |
|---|---|-----------|
| 6 | Integration between WP4 and WP5                   | Completed |
| 7 | Integration of Network Slicing                    | On-going  |
| 8 | Integration between each use-case and WP4 and WP5 | On-going  |

All steps have been completed apart from Network Slicing in the case studies. The Network Slicing requires interaction with hardware on site. It has been already integrated in the OSR case study in the scope of D7.3, and is currently being integrated in Marina Salud case study in the scope of D7.4.

### III.6. Continuous Integration / Continuous Deployment (CI/CD)

Modern development practices and DevOps are strongly based on CI/CD pipelines, in which developers are frequently merging code changes in a central repository where the building, unit tests, integration tests, delivery and deployment are automatically run.

Continuous Integration relates to the building and testing stages, where an automatic pipeline or process is triggered upon a code merge and the result is a new version of the application with the latest code changes. Continuous Delivery is an extension of the pipeline where the new version of the application is published and ready to be deployed in a specific environment.

Continuous Deployment relates to the deployment stage, that is, after the Continuous Integration and Delivery steps the application is automatically deployed to a specific environment, e.g., Production.

In ProTego we have mainly focused on the Continuous Deployment pipeline, though there has been some work on Continuous Delivery.

To run CI/CD pipelines, Jenkins along with GitLab is used: Jenkins as the main tool to design and run the CI/CD pipelines, and GitLab as the repository for Docker images and Helm charts.

The overall picture for the CI/CD pipeline in ProTego can be seen in Figure III-8.

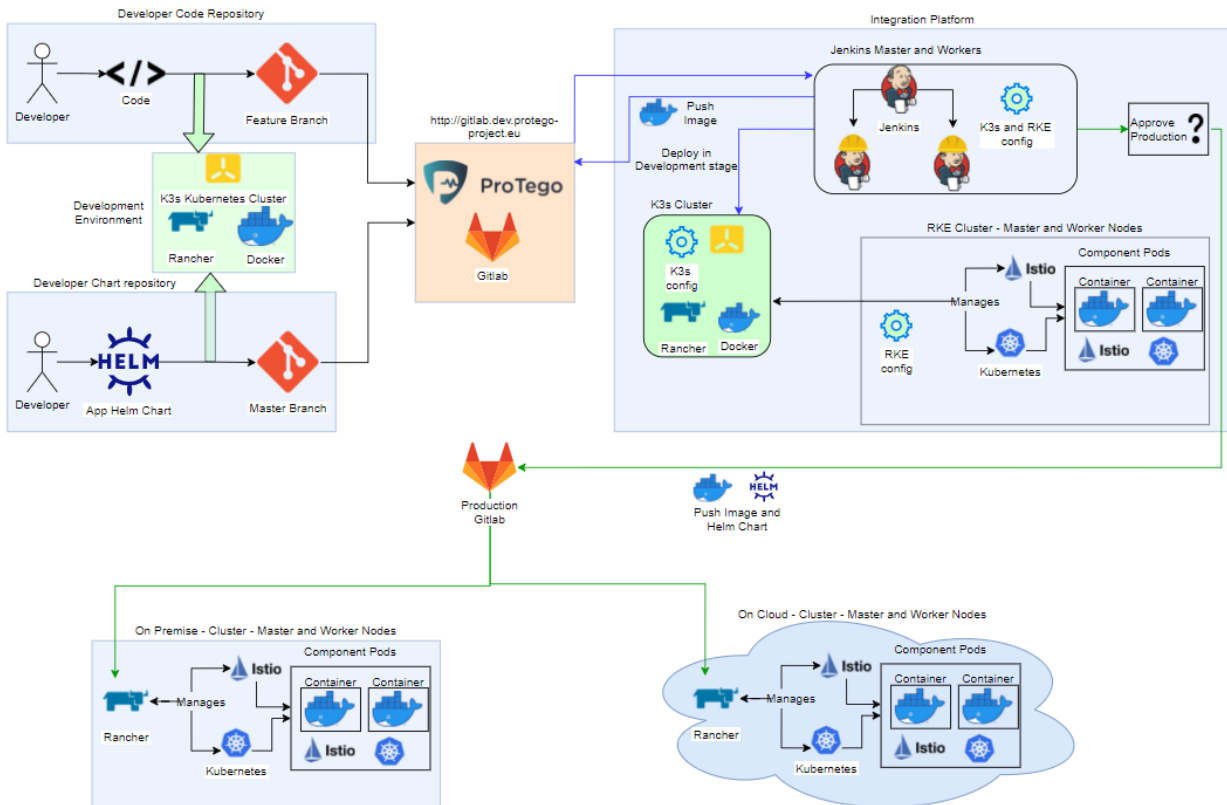


Figure III-8 ProTego CI/CD Pipeline

The CI pipeline is triggered on a code merge, using Jenkins. The component is compiled and tested if it has unit tests. Then the resulting application is built into a Docker image and pushed to the GitLab container registry in the Integration environment.

After that, the Helm chart for the component has to be updated by the component developer. The minimum update required would be updating the chart version and the new Docker image tag, when the only change is an update of the component version. Other Helm chart updates may be required at some point, when adding new Kubernetes resources for instance, but that kind of change should not be frequent, since the Helm chart design is already performed.

The update of the Helm chart triggers the CD pipeline, which deploys the component to the Integration Platform via Rancher.

Developers can inspect the pipeline status and results in the Jenkins UI as shown in Figure III-9:

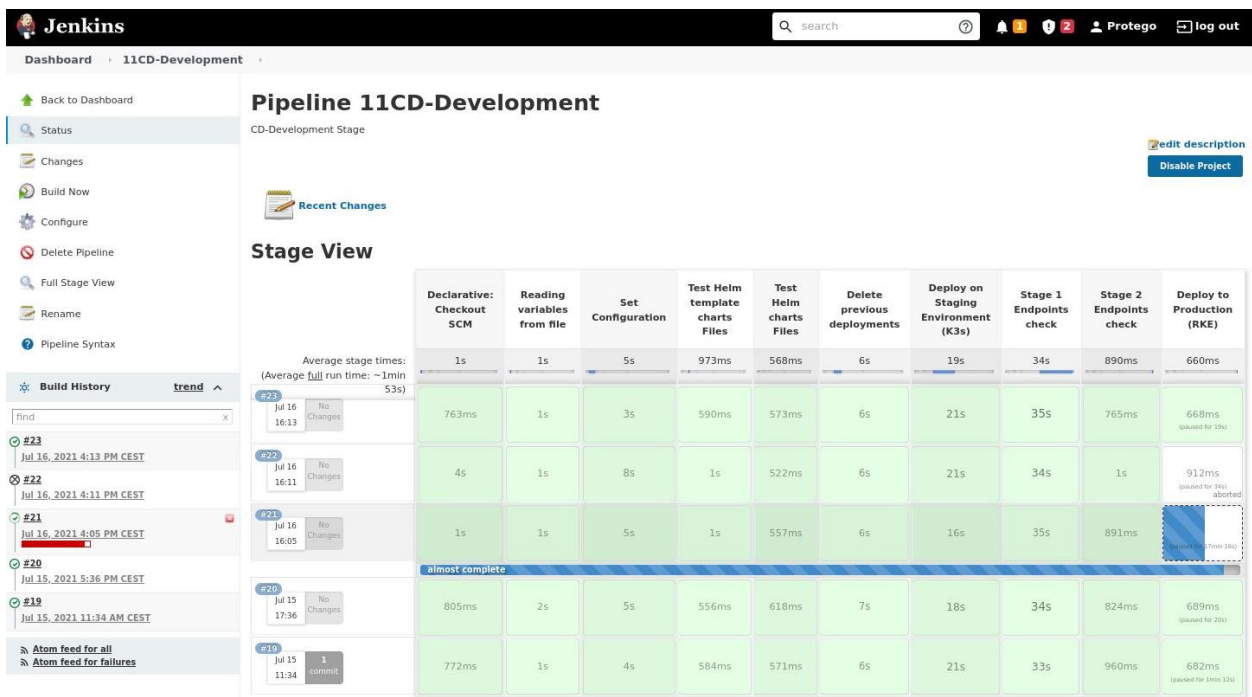


Figure III-9 Jenkins Pipeline Status View

Once the component is deployed, component developers must perform some validations and after that approve the update. This approval is performed using the Jenkins UI. The pipeline remains in pause mode until the update is approved. If the update is approved, this triggers a kind of Continuous Delivery pipeline that will publish the updated Docker image and Helm chart to the ProTego cloud GitLab repository. This repository has been previously registered in the case studies deployments, making the new version available for testing and deployment right after being published. shows a snapshot of the pipeline Approval step:

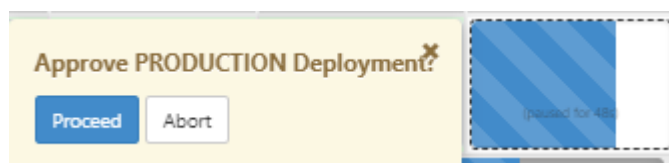


Figure III-10 Jenkins Pipeline Approve Step

Deployment of the case studies has not been implemented in the CI/CD pipelines, since the decision to update a component version should be made by those responsible for the case studies, who should make the decision as to when and where the new version is deployed. Ideally, any future ProTego customer should have two ProTego environments, Testing/Integration and Production.

Because some of the components from WP4 and WP5 are built from different open-source software and have their own code repositories, it is difficult to gather all those different repositories into one and implement a proper Continuous Integration pipeline to build the components automatically upon code changes. For ProTego, the CI pipeline is provided as an optional feature that the different components can use. This feature could be effectively used in a future exploitable ProTego toolkit, where the different owners of the components would keep developing and pushing changes to the code, and the CI pipelines would be triggered to build and test this code.

Jenkins uses a version.properties file configured in the component GitLab repository. It is used to configure a set of environment variables that will be used during the run of the pipeline. Those variables can be used in all stages of a pipeline.

Table III-3 shows a list of the different variables being used for the pipelines in ProTego:

Table III-3 List of Variables used at Jenkins version.properties file

| VARIABLE                       | Description   |
|--------------------------------|---|
| HELM_VERSION                   | Version Chart of the component  |
| IMAGE_VERSION                  | New Image Version (Tag)   |
| REGISTRY_CREDENTIAL            | Jenkins ID of Development Gitlab and Container Registry Credentials.    |
| REGISTRY_PRODUCTION_CREDENTIAL | Jenkins ID of Production Gitlab and Container Registry Credentials.     |
| REPOSITORY                     | Development Gitlab code repository                                      |
| BRANCH                         | Branch where the code is located  |
| MAVEN_POM_PATH                 | Location of files to generate the Artifacts                             |
| MAVEN_ARTIFACTS                | Location of artifacts for production                                    |
| DOCKER_FILE_PATH               | Location of the Dockerfile  |
| IMAGE_NAME                     | Docker image name for Development stage                                 |
| IMAGE_PRODUCTION_NAME          | Docker image name for Production Stage                                  |
| CONTAINER_IMAGE_REGISTRY       | Container registry for Development URL                                  |
| PRODUCTION_IMAGE_REGISTRY      | Container registry for Production URL                                   |
| APP_NAME                       | App name used for installation.   |
| CHART_NAME                     | Chart Name  |
| USERNAME                       | Username  |
| PRODUCTION_REPOSITORY          | Location of the Production Charts                                       |
| DEVELOPMENT_REPOSITORY         | Location of the Development Charts                                      |
| CLUSTER_CONFIG_PATH            | Location of config-multi containing development and production cluster. |

Some of these variables need to be modified by the component developers when updating changes.

In order to set up a pipeline, Jenkins uses the JenkinsFile, that is also stored in the component GitLab repository. The JenkinsFile contains the different stages required for a specific pipeline with the different steps in each stage. It is the single source of truth for the pipeline and contains



its definition, being the foundation of Pipeline-as-code which treats the CI/CD pipelines as part of the application, to be versioned and reviewed as code. It is a best practice to use a JenkinsFile rather than configuring the pipeline using the UI. This is an extract of the JenkinsFile that shows the stage of Deployment on Staging Environment:

```
pipeline {
  stages {
    ...
    stage ('Deploy on Staging Environment (K3s) ') {
      steps {
        script {
          try {

            withCredentials([usernamePassword(credentialsId:
            "$REGISTRY_CREDENTIAL",          usernameVariable:
            'USERNAME', passwordVariable: 'PASSWORD')]){

              sh 'echo uname=$USERNAME pwd=$PASSWORD'

              echo USERNAME

              sh 'helm upgrade --install $APP_NAME
              $CHART_NAME/' + "$HELM_VERSION" + ' --set
              privateRegistry.registryUser=$USERNAME,private
              teRegistry.registryPasswd=$PASSWORD --
              kubeconfig $CLUSTER_CONFIG_PATH'

            }
          } catch (err){
            echo err.getMessage()
          }
        }
      }
    }
    ...
  }
}
```

## III.7. Storage

Files in a container are ephemeral, which presents some problems for stateful applications when running in containers. One problem is the loss of files when the container crashes. Kubernetes restarts the container but with a clean state. Another problem occurs when sharing files between containers running together in a pod. Kubernetes volumes and persistent volumes solve such problems.

A persistent volume Kubernetes resource is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes.

A persistent volume claim is a request for storage, for a specific persistent volume or storage class, that will be mapped within a Pod to a container directory.

Typically, persistent volumes use local filesystem for data persistence, but some other issues arise in multi node clusters. If a Pod, with a local persistent volume, crashes, Kubernetes may choose to deploy it to a different node. In that case, data is not moved to the new node, thus



resulting in data loss from the application. Another issue could be when different pods need to share a persistent volume. In that case, both Pods need to be in the same node to access the same data in the volume.

In order to solve those issues, a Network File System (NFS) storage is used as the storage layer for persistent volumes. The NFS volume option allows the sharing of an existing Network Filesystem, to be mounted in the pod. If the pod is deleted or rescheduled in a different node, data is preserved. Figure III-11 shows a snapshot of an NFS volume for the DataGateway component:

```
vagrant@nfs1:/mnt/nfs_share$ sudo tree protego
protego
├── dg
│   └── elsfhirchart
│       └── els-fhir
│           └── observation_IBM_FHIR.parquet.encrypted
│               ├── part-00000-e783e868-6c8b-44d2-a7a2-18a66169a270-c000.snappy.parquet
│               ├── part-00000-fbdac718-ba8c-4989-a608-b2292dc4b30f-c000.snappy.parquet
│               └── _SUCCESS
4 directories, 3 files
```

Figure III-11 NFS Volume

Then it is just a matter of using the NFS option in the Kubernetes Persistent Volume resource as is shown in this example of a PV yaml file for the DataGateway:

```
{{- if .Values.persistence.enabled -}}
apiVersion: v1
kind: PersistentVolume
metadata:
  namespace: {{ .Release.Namespace }}
  name: {{ .Values.app.name }}-pv-volume
  labels:
    type: local
spec:
  capacity:
    storage: {{ .Values.app.volumes.pv.capacity }}
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  claimRef:
    name: {{ .Values.app.name }}-pv-claim
    namespace: {{ .Release.Namespace }}
  {{- if eq .Values.volumes.type "local" }}
  hostPath:
    path: "{{ .Values.volumes.basepath }}"
  {{- else }}
  nfs:
    server: {{ .Values.volumes.nfs_server }}
```

```
    path: "{{ .Values.app.volumes.pv.path }}"
  {{- end }}
{{- end }}
```

## III.8. Next Steps

Some key points which would improve features provided by the Integration Toolkit have been identified. This work may be performed partially in the scope of the remainder of ProTego, or in the scope of possible future projects, and includes:

- Improving the installation steps.
- Researching on Kubernetes Native Storage.
- Researching on Cloud / Terraform.
- Integrating other Kubernetes DevOps utilities.

### III.8.1. Installation Steps

The installation of the Integration Toolkit can be made more user friendly by wrapping the run of the Ansible playbooks with a backend API and a Frontend, everything bundled in a Docker container.

At the moment, as has been described, the Integration Toolkit installation is performed running the Ansible playbooks. Those playbooks are run one by one, following the installation instructions.

In order to improve this process, all playbooks and software would be packaged in a Integration Toolkit Docker Container. This image would be ready to run Ansible, so there would be no need to install any software at the installation environment other than Docker.

Backend software would be developed and packaged, providing an HTTP API to run the playbooks.

A frontend UI would be developed to provide the interface for the users to run the playbooks, similar to an installer. This UI would guide users step by step through the installation process.

Users would be able to run the Docker image on a specific node of the environment, or even from their own laptops, in order to install the Integration Toolkit.

Deployment of the WP4 and WP5 components would be also included in this installer.

### III.8.2. Kubernetes Native Storage

Besides using NFS as the storage layer for persistence in the Integration Toolkit, additional tests are being performed with Kubernetes Native Storage systems.

Such storage systems provide an abstraction layer of a distributed persistent storage to the Pods or components. They provide high availability and replication of the data, so that data is always accessible for a Pod from any node.

There are different technologies that provide such abstraction levels:

- Longhorn.
- GlusterFS.
- Rook+Ceph.
- OpenEBS.
- StorageOS.

Initially Longhorn could be tested, which was originally a project from Rancher, but is now a CNCF project.

Longhorn is a highly available persistent storage for Kubernetes, which uses the local filesystem of the nodes and provides replication and backups. It is deployed on a Kubernetes cluster as a regular application using a Helm chart. It is used by the application by selecting the specific StorageClass in a Persistent Volume Claim.

### III.8.3. Cloud / Terraform

As has been described above, the Marina Salud case study has been deployed to Amazon AWS cloud service using a mix of EC2 instances and EKS.

This opens up the possibility for the ProTego toolkit to be deployed as SaaS in the cloud, or for other On-Cloud deployments.

Terraform allows infrastructure to be expressed as code (Infrastructure as Code) in a language called HCL, similar to yaml and other declarative languages. It reads configuration files and provides an execution plan of changes, which can be reviewed and then applied and provisioned. Extensible providers allow Terraform to manage a broad range of resource, including IaaS, PaaS, SaaS and hardware services.

Terraform is used to provision infrastructure across different public clouds and services, creating reproducible infrastructure, provisioning consistent testing, staging and production environments with the same configuration.

The idea is that by using a combination of Ansible and Terraform, different configurations can be set up to deploy the Integration Toolkit and ProTego to different cloud providers, with Terraform being used to provision the infrastructure, and Ansible to provision the software, as at present.

### III.8.4. Other Kubernetes DevOps Utilities

DevOps is a trending topic nowadays, and is mostly based on using Containers and Kubernetes. The Kubernetes ecosystem and tooling is growing exponentially, providing a number of extra features and functionalities that Kubernetes may not offer out of the box.

These include:

- Extended Secret Management : Sealed-Secrets, Helm-Secrets ...
- Multi-tenancy beyond namespaces: Kiosk ...
- GitOps which is an extension of CI/CD for Kubernetes: ArgoCD, The GitOps Toolkit ...
- Resource publishing across namespaces.
- Others ...

Future work would be to research the most appropriate of these technologies and incorporate them into the Integration Toolkit.

## IV. COMPONENT RESOURCES

Each component is described in the same format for ease of understanding, with a diagram and description of the components and their internal layout. These components are based on the resources supplied in D6.1 Resources section. The template and diagram key are described below.

### IV.1.1. TEMPLATE: WP: Component Name (ACRONYM- Partner)

#### IV.1.1.a. Component Description

Describe the components functionality

This component is formed of these sub-components

1. **SubcomponentName:** Description of technologies and function
2. **SubcomponentName:** Description of technologies and function
3. **SubcomponentName:** Description of technologies and function

#### IV.1.1.b. Component Diagram

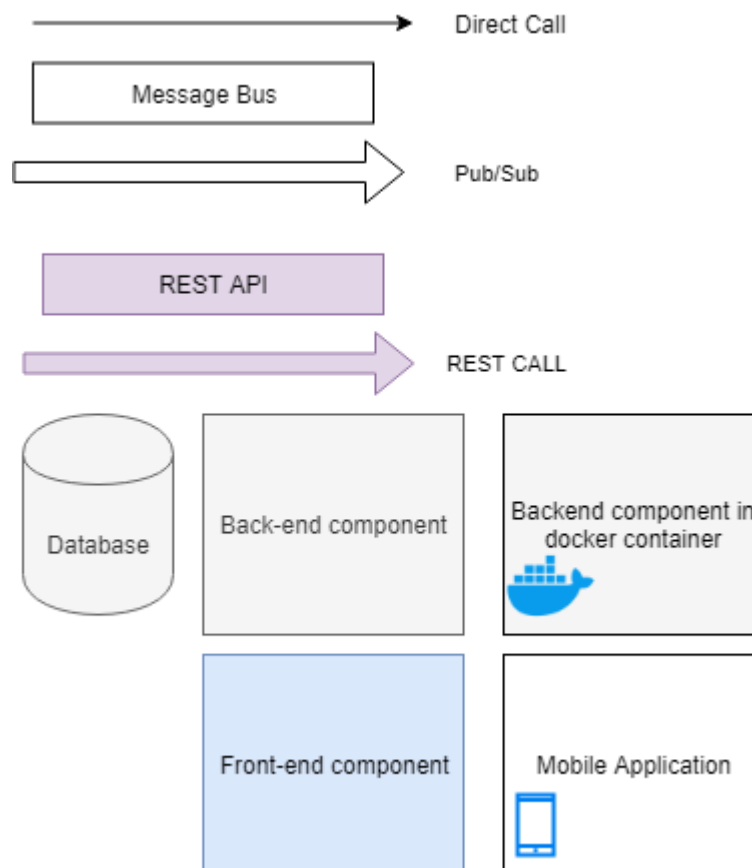


Figure IV-1: Key for Component Diagram

## IV.2. WP4: Cybersecurity risk assessment tools

WP4 focuses on risk assessment tools that will be able to monitor and assess risk of the running system and during design. This is formed of two major components: The System Security Modeller and the Security Information and Event Management. These two components allow

modelling and measurement of the system to provide an update on a risk based on the information collected from the ProTego system.

### IV.2.1. WP4: System Security Modeller (SSM – IT Innovation)

#### IV.2.1.a. Component Description

The System Security Modeller (SSM) is one of the risk assessment analysis tools in the ProTego project. SSM is a web-based graphical tool which allows a system designer or analyst to perform a design-time risk assessment of a system using the procedures defined by ISO 27005. Figure IV-2 gives an overview of the high-level architecture of the constituent services and underlying triple store. SSM is a web-based application with a user interface for a system of back-end components which is accessible via a browser application. The interface exposes selected parts of the models, such as collections of assets or threats (read only) or update methods to allow users to edit models provided by a model service. A RESTful API is defined to provide the endpoints that enable access to the services of SSM, either directly through API requests, or using the SSM UI via a browser, or some other application. The calls received through the REST layer are then sanitised and passed to the underlying model querier and model validator components. A low-level semantic store component allows access to the actual triple store, i.e., data entities composed of subject-predicate-object. The store contains one core model, and potentially multiple domain and system models. All models are stored in separate graphs, allowing for easy import and export. As part of the ProTego project an extra SSM microservice has been developed to mediate the communication between SIEM and SSM. The microservice offers a REST API for SIEM to send http requests and a pub/sub channel for sending messages to Kafka in SIEM (see D4.3 for a more detailed description of the SSM microservice and the communication between SSM and SIEM).

#### IV.2.1.b. Component Diagram

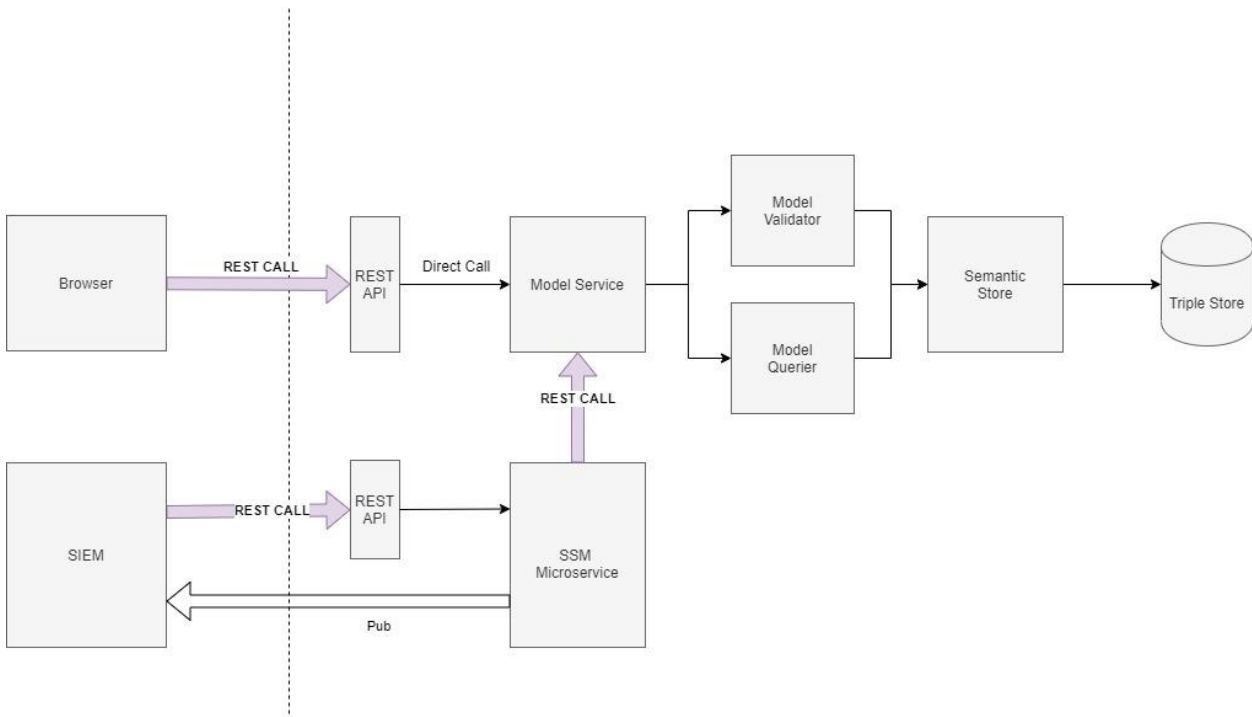


Figure IV-2: High level architecture of the System Security Modeller tool.

## IV.2.2. WP4: Security Information and Event Management (SIEM – Inetum)

### IV.2.2.a. Component Description

The SIEM is the ProTego tool that monitors security during run-time. This is accomplished by collecting information from various sources, enriching data, correlating different pieces of information, and analysing all the set.

There are diverse SIEM components, each one with a specific objective:

1. **Beats:** Agents installed on monitored systems and used for capturing data such as log events, network packets, metrics, etc. The destination is Logstash in most cases.
2. **Logstash:** Data Collection Engine. It can unify data from disparate sources and normalize data into destination (in most cases Elasticsearch).
3. **Wazuh Agent:** It is installed on monitored systems and used to collect different types of system and application data that it forwards to the Wazuh Manager.
4. **Wazuh Manager:** Correlation Engine in charge of analysing the data received from the agents. It uses decoders to identify the type of information being processed and extract relevant data elements. Next by using rules it can identify patterns in the decoded records. Data is stored into Elasticsearch indices.
5. **ElasticSearch:** Storage, Index, and Search Engine. It is the central repository where all information will be stored.
6. **Kibana:** Analytics and Visualization platform of information stored in Elasticsearch indices. It is the user interface of the SIEM.
7. **MachineLearning:** Processes that analyse network traffic to detect threats using Deep Learning techniques.
8. **VA Scheduler:** Vulnerability Assessment Scheduler. It generates the different tasks that will trigger the operations in OpenVAS.
9. **OpenVAS:** Vulnerability Scanner. It executes a continuously updated and extended feed of Network Vulnerability Tests and compiles reports based on the findings.
10. **OWASP ZAP:** Web Application Vulnerability Scanner. It searches for vulnerabilities in web applications and compiles reports based on the findings.
11. **VA Analyser:** Vulnerability Assessment Analyser. It processes the reports generated by OpenVAS, complement additional information, store it on Elasticsearch, and send it to the SSM. The SSM uses this information to recalculate risk.
12. **Kafka:** Stream-Processing Platform. Used a message bus by other components to send information to the SIEM where it will be processed by Logstash.

### IV.2.2.b. Component Diagram

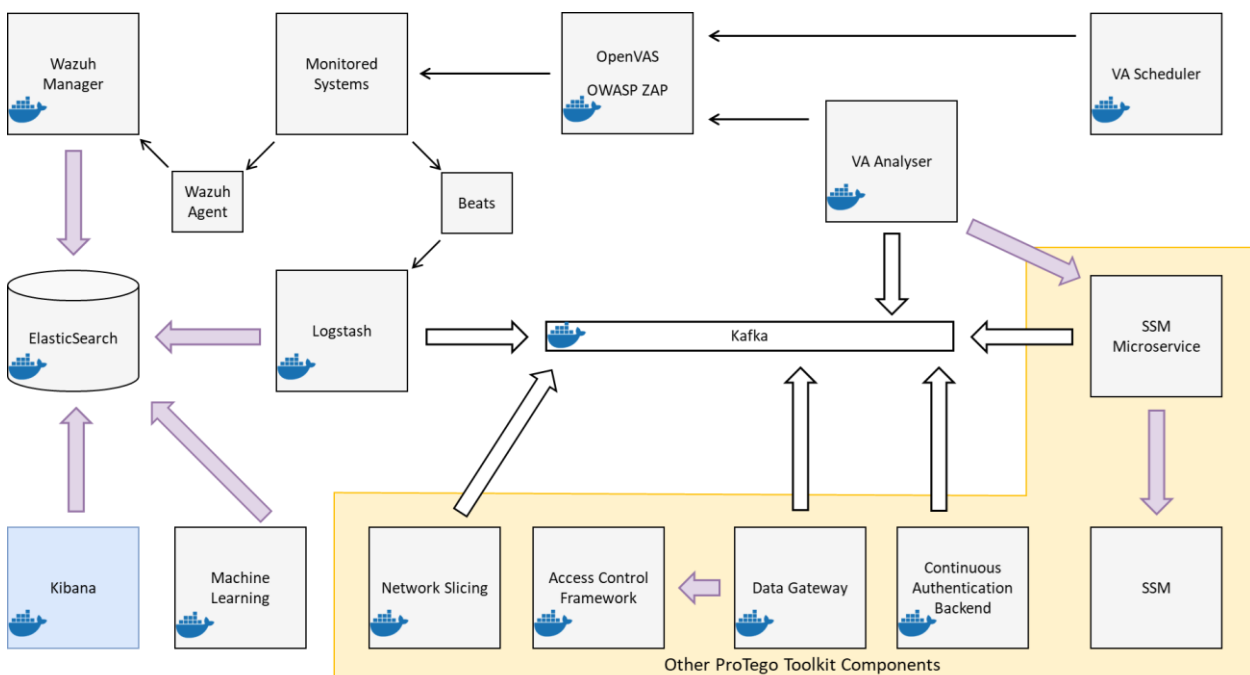


Figure IV-3 High Level Architecture of the SIEM

## IV.3. WP5: Cybersecurity Risk Mitigation Tools

WP5 focuses on the risk mitigation tools and technologies. There are four main components: Continuous Authentication, Data Gateway, Access Control Framework and Network slicing. Together they provide secure access authenticated access to medical data, and secure it at rest and in transit.

### IV.3.1. WP5: Continuous Authentication (CA - UAH)

#### IV.3.1.a. Component Description

The continuous authentication (CA) system is the main component centred in mobile device security. CA is a method of verification that continuously monitors and authenticates users based on collected data. Through this component ProTego is capable to identify the authorized users and unauthorized users of mobile devices.

The continuous authentication system is composed by the following main components:

1. **CA Agent:** This component is a mobile agent that retrieves behavioural metrics and sends them to the API. The CA agent performs as an Endpoint Detection and Response (EDR). The main task of this component is to retrieve information to continuously authenticate the user, but it also has response capabilities. For example, CA agent can raise a security alert to inform other components like the SIEM.
2. **CA API:** This component is a backend that centralizes the logs and associates each of them with the users' identity. Periodically, this API generates machine learning models based on the logs stored and evaluates the new incoming logs against these models.

These two components are responsible of the correct performing of the CA system. A typical behaviour for checking the device user authorization will be the following: the CA agent periodically asks for the trust of a user; the backend checks all the logs stored since the creation of the model and returns a value between 0 and 1 (a trustworthiness percentage).

### IV.3.1.b. Component Diagram

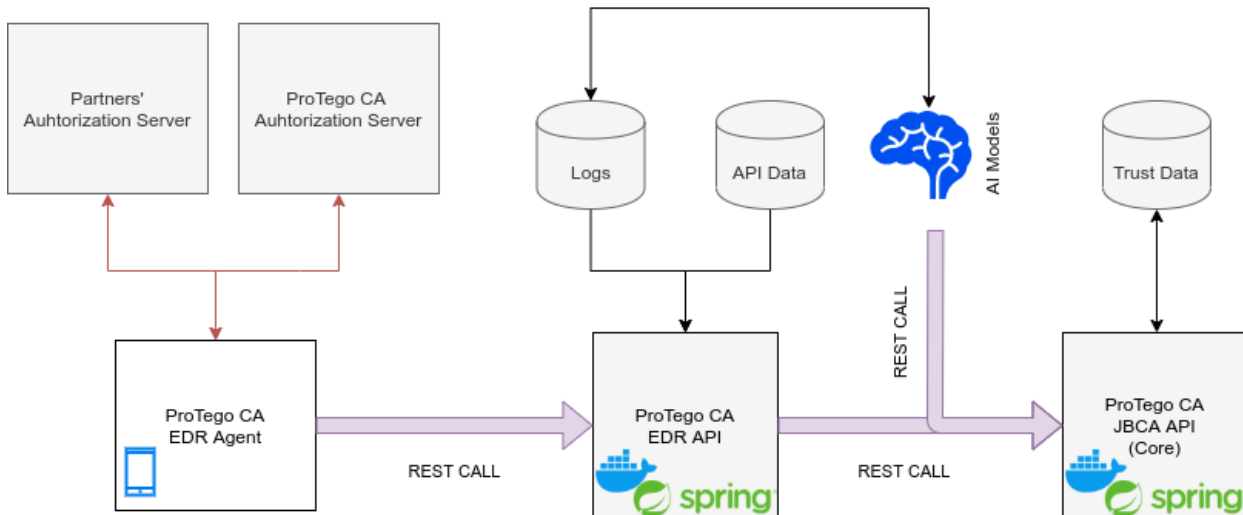


Figure IV-4: High level architecture of the Continuous Authentication Component.

### IV.3.1. WP5: Data Gateway (DG - IBM)

#### IV.3.1.a. Component description

The Data Gateway (DG) is the interface between external REST-based clients with and the ProTego backend storage of FHIR resources. The DG stores FHIR resources in encrypted Parquet format, and allows for powerful SQL queries of the stored data.

Conceptually, the Data Gateway consists of two subcomponents:

- a) A **FHIR server** which receives FHIR requests over REST and stores FHIR resources as encrypted Parquet files,
- b) **Query Gateway** which implements a REST end point to receive SQL queries and execute them against the encrypted Parquet files.

The DG communicates with the Access Control Framework to obtain the required encryption keys for authenticated users, and posts notifications to a Kafka message queue on refused access to the Key Management System or detected tampering attempts of the stored Parquet files.



### IV.3.1.b. Component Diagram

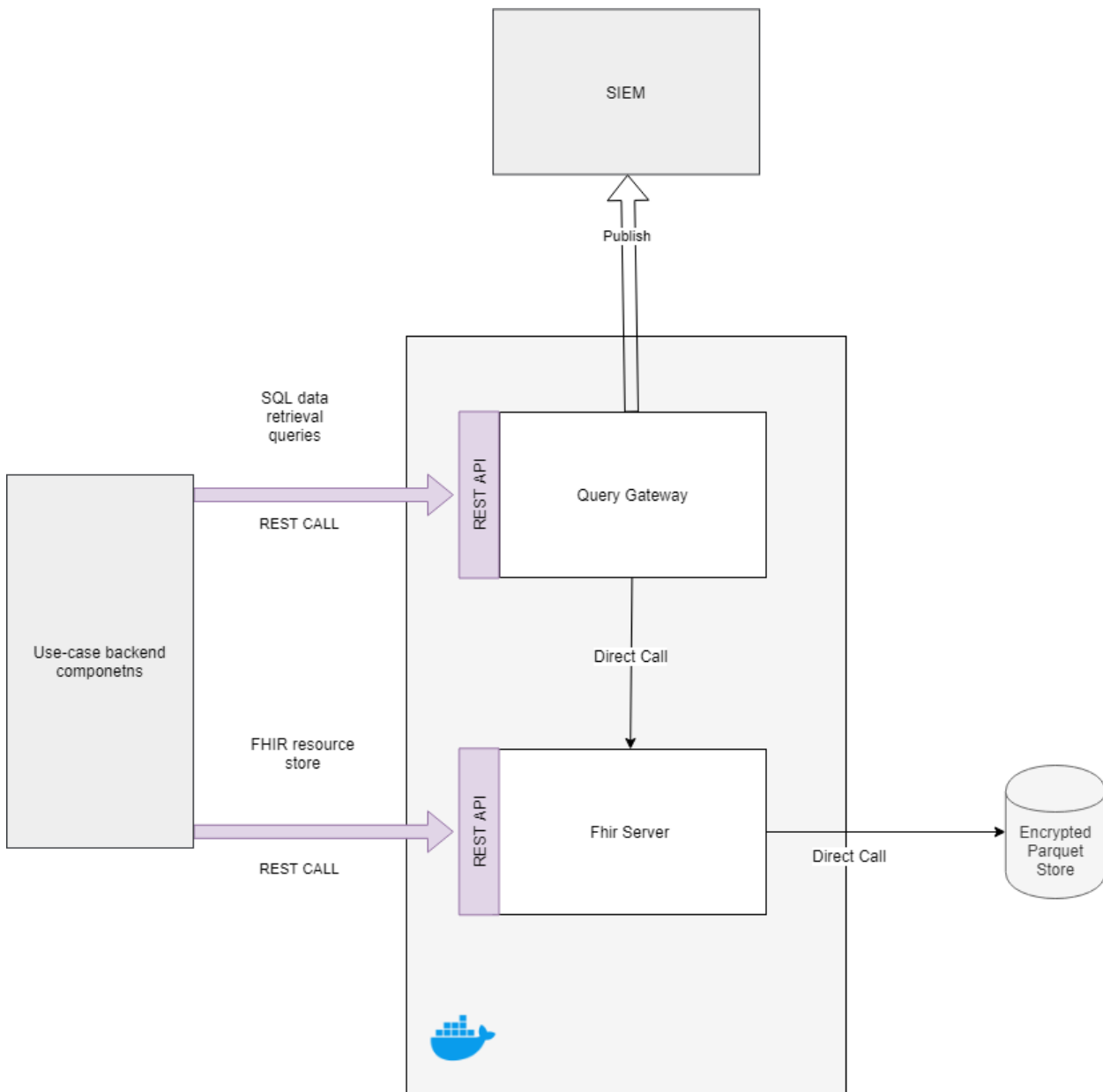


Figure IV-5: Component Level Diagram

## IV.3.2. WP5: Access Control Framework (ACF – KU Leuven)

### IV.3.2.a. Component Description

The **access control framework (ACF)** component is one of the main components in the ProTego risk mitigation system. The main application of the ACF component is to enhance the security of the Data Gateway (DG) by ensuring that only authorized users can have access to FHIR data that is stored externally. It consists of the following two sub-components:

1. **Key Management Service (KMS):** This component is used to protect the Key Encryption Keys (KEK) that are needed by the Data Gateway to access FHIR data<sup>1</sup>. The KEKs are not stored directly in the KMS, but are encrypted/decrypted with a master key that is stored

<sup>1</sup> Note that the KEK is not used directly to encrypt/decrypt FHIR data. Instead, the FHIR data is encrypted with a Data Encryption Key (DEK), which is then encrypted with a KEK.

internally in the KMS. Therefore, when access to FHIR data is needed, the DG will make encryption/decryption queries to the KMS to get access to the corresponding KEK. Within the ProTego project, the open-source solution Vault from Hashicorp is used to provide the KMS functionality.

2. **Access Control (AC):** This component is used to manage the encryption/decryption queries made to the KMS, and decide whether the KEK should be sent to the DG or not. This decision depends on whether the data access request – which triggered the DG to query for the KEK – should be granted or not. Internally, it consists of the following functions:
  - a. **Token validation:** Users and their corresponding roles and attributes are identified by a token issued by an external Identity and Access Management (IAM) system. The token validation function checks whether the token that is passed to the AC is valid token or not.
  - b. **Policy decision point:** Based on the content of the token, the auxiliary information that is stored together with the KEK and the security policies in place, the policy decision point will decide whether a particular data access request, and its corresponding query to the KMS to get a KEK, should be granted or not.
  - c. **Policy enforcement point:** This is the main interface between the KMS and the AC. It will pass encryption/decryption queries to the KMS, and will get an encrypted/decrypted KEK back from the KMS. Based on the output of the token validation and policy decision point, it then will either pass this encrypted/decrypted KEK to the DG, or give an error message.

The KMS and AC jointly realize the access control framework functionality. However, besides interacting with the DG, it is indirectly also linked to an external IAM. The latter is not one of the components developed by ProTego, but is needed to make access control decisions. The main goal of the IAM is to manage all the users in the system and their respective roles and/or attributes (e.g., being a doctor in hospital x). Users authenticate to the IAM, and receive a JSON WebToken (JWT) upon successful authentication. These tokens are digitally signed by the IAM. Two different IAMs are used in the ProTego project. The first IAM is Keycloak, which is an open-source framework. The second IAM is AWS Cognito, which is an Amazon Web Services (AWS) product.

#### IV.3.2.b. Component Diagram

The diagram of the access control framework is shown below.

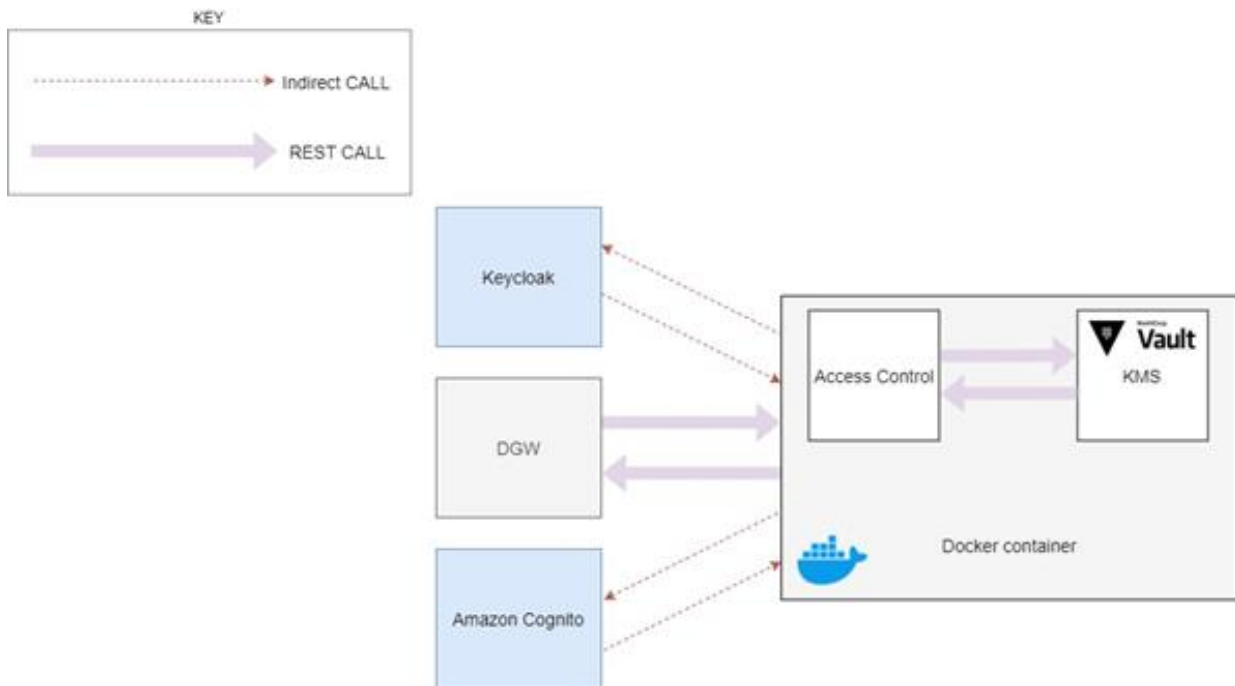


Figure IV-6: Subcomponents of the ACF and their high-level interfaces.

### IV.3.3. WP5: Network Slicing (NS - IMEC)

#### IV.3.3.a. Component Description

The network slicing control functions in ProTego are formed by 3 main components which may be virtualized:

- The **5G-EmPOWER controller** that interacts with the Wi-Fi APs or Wireless Termination Points (WTPs) through the 5G-EmPOWER protocol. It also interacts with the SDN controller (Backhaul controller) through a Pub/sub API to express the client configuration to the rest of the network.
- The **SDN controller** (backhaul controller) will configure the level-2 network through the OpenFlow protocol to enable wireless slices to extend to the wired domain. It will also tag application traffic accordingly to allow Wi-Fi slicing by the APs.
- The **Secure Interface Setup** component will provide end-to-end security by interacting with the ProTego backends.

### IV.3.3.b. Component Diagram

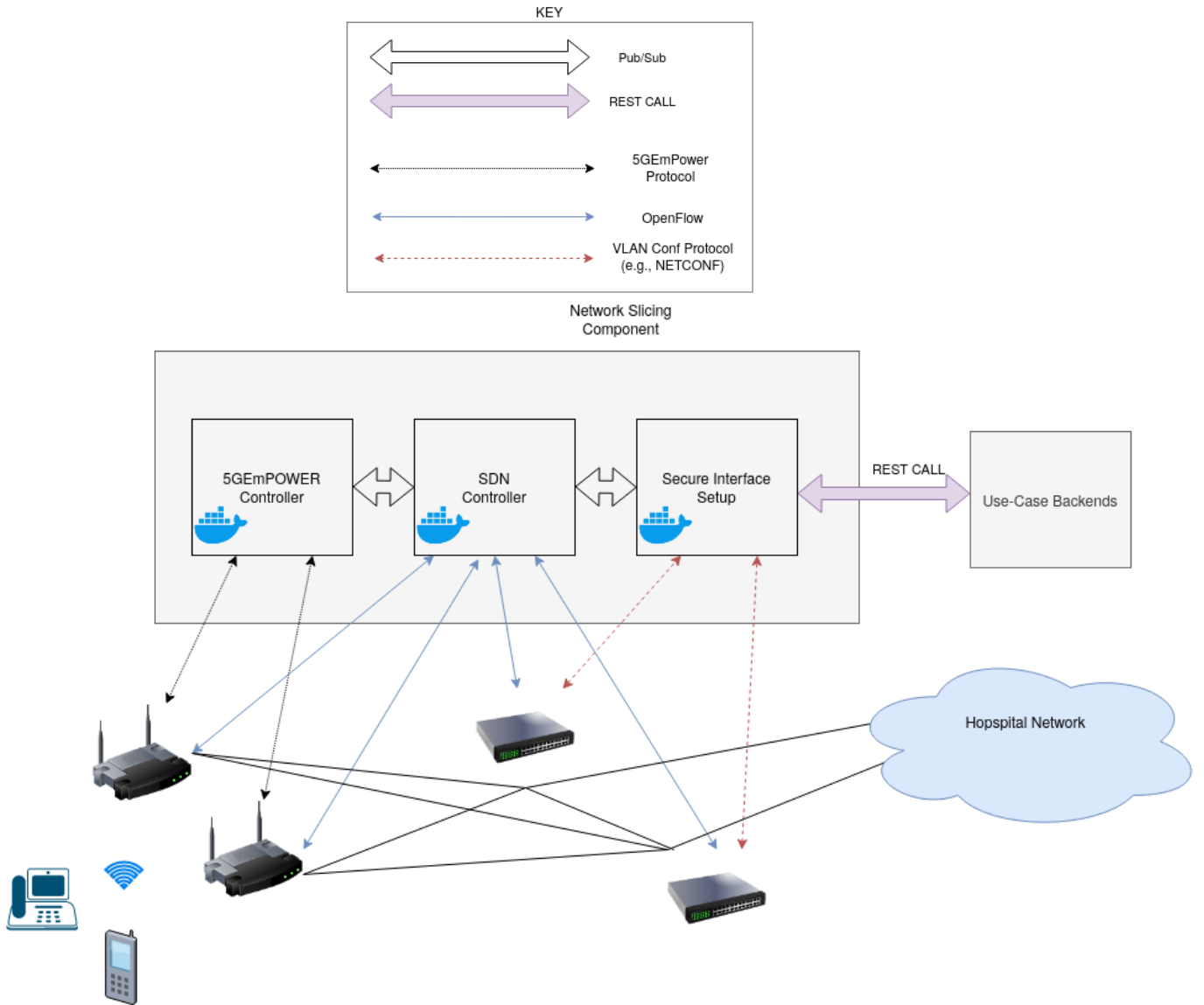


Figure IV-7: The component level diagram for the Network Slicing Component

## V. REQUIREMENTS

This section covers requirements. Initial requirements are those from deliverable D6.1, augmented with new requirements collected as part of the integration work and initial estimates of resource usage which are needed for the hardware specification.

### V.1. Initial Requirements

This section covers requirements of differing natures:

- **Use-case:** from D6.1 and D6.2, these provide the basic requirements for the use-cases; further information can be found in D2.3.
- **Project:** from D6.1 and D6.2, baseline needs, given the nature of the project consortium and goals.
- **Lifecycle:** from D6.1 and D6.2, thinking beyond the baseline needs to managing the development of the project.

Table V-1: Summary of Initial Requirements from D6.1 and D6.2

| Nature    | Requirement            | Success Criterium  |
|-----------|------------------------|--|
| Use-case  | FoodCoach              | ProTego handles FHIR data  |
| Use-case  | FoodCoach              | ProTego security fits with a web and mobile frontend architecture  |
| Use-case  | Pocket EHR             | ProTego handles bulk data storage in Parquet file  |
| Use-Case  | Pocket EHR             | ProTego needs to include on on-cloud component   |
| Project   | Modular and Extensible | ProTego provides a continuous integration toolkit around Docker containers allowing for new components to be added and other components to be updated  |
| Project   | Security               | Manages both patient data and security data  |
| Project   | Network slicing        | The network slicing means the overall technology needs to benefit from the hardware router customisation.  |
| Lifecycle | Updates                | The use of continuous integration tools helps to maintain not only the ProTego tools but also the medical application (as applicable)  |
| Lifecycle | Platform               | ProTego needs the toolkit to be provided as a platform to allow for deployment in a standardised and manageable way. And to give a baseline architecture for the ProTego component and the use-cases to run from |

## V.2. Integration Requirements

Integration requirements are specific requirements in respect of each component to make sure they fully integrate with the rest of the ProTego Toolkit.

### V.2.1. WP4: System Security Modeller

Table V-2: Integration Requirements from the SSM

| Integration Requirements and Interfaces with other components |                                     |   |                |
|---|-------------------------------------|---|----------------|
| #   | Requirement Description             | Success Criteria  | Nature         |
| 1   | Receive vulnerabilities report      | Vulnerability successful passed to the SSM API                | With WP4: SIEM |
| 2   | Publish recommendations to the SIEM | A recommendation is successfully published to the message bus | With WP4: SIEM |

### V.2.2. WP4: Security Information and Event Management

Table V-3: Integration Requirements from the SIEM

| Integration Requirements and Interfaces with other components |  |  |          |
|---|--|--|----------|
| #   | Requirement Description  | Success Criteria                             | Nature   |
| 1   | Supply of vulnerabilities report to SSM Microservice   | Successful sending vulnerability information | REST API |
| 2   | Receive recommendations from the SSM   | Successful reception of recommendations      | Kafka    |
| 3   | Receive alerts from WP5 components (Data Gateway, Access Control Framework, and Continuous Authentication Backend) | Successful reception of alerts               | Kafka    |

### V.2.3. WP5: Continuous Authentication

Table V-4: Integration Requirements from Continuous Authentication

| Integration Requirements and Interfaces with other components |   |   |                        |
|---|---|---|------------------------|
| #   | Requirement Description   | Success Criteria  | Nature                 |
| 1   | Android's EDR service should be launched by the partner's application | Partner's application starts successfully the EDR agent and can pass messages | Android implementation |

|   |  |   |                      |
|---|--|---|----------------------|
| 2 | Receiving an access token from the external IAM (Amazon Cognito or Keycloak) | A valid access token that is signed, includes the user attributes and specific pre-shared authorities (i.e., "groups", "roles", "scope", etc) | JSON Web Token (JWT) |
|---|--|---|----------------------|

## V.2.4. WP5: Data Gateway

Table V-5: Integration Requirements from Data Gateway

| Integration Requirements and Interfaces with other components |  |   |                     |
|---|--|---|---------------------|
| #   | Requirement Description                                | Success Criteria  | Nature              |
| 1   | Network connectivity between all WP 4 and 5 components | Successful execution of the retrieval and subsequent querying of a FHIR resource. | Integration Toolkit |

## V.2.5. WP5: Access Control Framework

Table V-6: Integration Requirements from Access Control Framework

| Integration Requirements and Interfaces with other components |   |   |                              |
|---|---|---|------------------------------|
| #   | Requirement Description   | Success Criteria  | Nature                       |
| 1   | The ability to interface the ACF with the Data Gateway using a REST API interface | An acceptable header and payload format with the endpoints<br>transit/encrypt/MkId or<br>transit/decrypt/MkId | Curl command                 |
| 2   | Receiving an access token from the external IAM (Amazon Cognito or Keycloak)      | A valid access token that is signed and includes the user attributes  | JSON Web Token (JWT)         |
| 3   | Keycloak configuration  | An acceptable configuration that is compatible with the ACF   | Sign-up and sign-in services |
| 4   | AWS Cognito configuration   | An acceptable configuration that is compatible with the ACF   | Sign-up and sign-in services |

## V.2.6. WP5: Network Slicing

Table V-7: Integration Requirements from Network Slicing

| Integration Requirements and Interfaces with other components |                         |                  |        |
|---|-------------------------|------------------|--------|
| #   | Requirement Description | Success Criteria | Nature |

|   |   |   |                |
|---|---|---|----------------|
| 1 | Deploy all 3 network slicing control components as described in WP5. Multiple instances may be required.                    | Successful control of network slices for different clients and traffic types. | All Components |
| 2 | Access to the ProTego API to receive application profiles and access to the ProTego authentication and encryption services. | Successful API implementation and deployment                                  | REST API       |

### V.2.7. WP6: Integration Toolkit

Table V-8: Integration Requirements from the Integration Toolkit

| Integration Requirements and Interfaces with other components |  |  |  |
|---|--|--|--|
| #   | Requirement Description  | Success Criteria                               | Nature                                 |
| 1   | The ability to update the components to maintain security                        | A successful GitLab and Jenkins CI/CD pipeline | Integration Toolkit                    |
| 2   | To deploy all WP 4 and 5 components in the same environment                      | As validated by D7.2                           | Integration Toolkit and All components |
| 3   | All components have Docker images, and orchestration file are uploaded to GitLab | Check for the files on GitLab                  | All components                         |

## V.3. Resource Usage

This section presents more accurate estimates for resource usage than in previous deliverables, after deployment and test in the case studies. They are estimates since tests under heavy load have not been performed. Resource usage is difficult to predict as for the ProTego Toolkit it depends on a complex mix of the technology, number of users, amount of data and complex paths through the system.

### V.3.1. WP4: System Security Modeller

The System Security Modeller for the ProTego testbeds incorporates two main components, the SSM itself, and the SSM-Adaptor kit.

#### System Security Modeller

The minimum requirements for the production deployment of the SSM cluster is a compute node with the following specs:

Table V-9: Computing Resource Estimates for SSM

|                                       |
|---------------------------------------|
| <b>Computation Resource Estimates</b> |
|---------------------------------------|



|            |     |
|------------|-----|
| CPU (vCPU) | 4.0 |
| RAM (GB)   | 8   |
| Disk (GB)  | 32  |

### SSM-Adaptor

The SSM-Adaptor deployment requirements are the following:

Table V-10: Computing Resource Estimates for SSM-Adaptor

| Computation Resource Estimates |     |
|--------------------------------|-----|
| CPU (vCPU)                     | 2.0 |
| RAM (GB)                       | 4   |
| Disk (GB)                      | 4   |

## V.3.2. WP4: Security Information and Event Management

Table V-11: Computing Resource Estimates for SIEM

| General Computation Resource Estimates |                            |
|--|----------------------------|
| CPU (vCPU)                             | 14.25                      |
| RAM (GB)                               | 22 Gb                      |
| Disk (GB)                              | 125 Gb ~ Huge amount of Tb |

### ElasticSearch Cluster

The minimum requirements for the production deployment of an ElasticSearch cluster are at least one coordinating node, three master nodes and at least two data nodes.

It is important to emphasize that the number of master nodes cannot be less than 3 and should always be an odd number.

Table V-12: Computing Resource Estimates for ElasticSearch coordinating node

| Computation Resource Estimates |     |
|--------------------------------|-----|
| CPU (vCPU)                     | 1.0 |
| RAM (GB)                       | 1   |
| Disk (GB)                      | 1   |

Table V-13: Computing Resource Estimates for ElasticSearch master node

| Computation Resource Estimates |      |
|--------------------------------|------|
| CPU (vCPU)                     | 0.75 |

|           |   |
|-----------|---|
| RAM (GB)  | 1 |
| Disk (GB) | 3 |

Table V-14: Computing Resource Estimates for Elasticsearch data node

| Computation Resource Estimates |   |
|--------------------------------|---|
| CPU (vCPU)                     | 1.0   |
| RAM (GB)                       | 3   |
| Disk (GB)                      | ~1000 - Space on demand, as this will depend on the conditions of data collection. But it is advisable to reserve space in the order of Tb. |

### Logstash

The requirements may change depending on the workload, as more instances of Logstash may be deployed to perform load balancing.

Table V-15: Computing Resource Estimates for Logstash

| Computation Resource Estimates |   |
|--------------------------------|---|
| CPU (vCPU)                     | 1.0   |
| RAM (GB)                       | 2   |
| Disk (GB)                      | 5. Although this space can be variable depending on the input size of the persistent queues |

### Kibana

Table V-16 Computing Resource Estimates for Kibana

| Computation Resource Estimates |  |
|--------------------------------|--|
| CPU (vCPU)                     | 1.0  |
| RAM (GB)                       | 2  |
| Disk (GB)                      | Minimal. Kibana's own information is stored in Elasticsearch in a dedicated index. |

### Wazuh Manager

Table V-17: Computing Resource Estimates for Wazuh Manager

| Computation Resource Estimates |     |
|--------------------------------|-----|
| CPU (vCPU)                     | 1.0 |
| RAM (GB)                       | 2   |
| Disk (GB)                      | 30  |

## Kafka Stack

Table V-18: Computing Resource Estimates for Kafka

| Computation Resource Estimates |     |
|--------------------------------|-----|
| CPU (vCPU)                     | 2.0 |
| RAM (GB)                       | 3   |
| Disk (GB)                      | 50  |

## OpenVAS Stack

The requirements will depend on the number of master-slave probes that are established in the infrastructure.

Table V-19: Computing Resource Estimates for OpenVAS Stack

| Computation Resource Estimates |  |
|--------------------------------|--|
| CPU (vCPU)                     | 4.0  |
| RAM (GB)                       | 3  |
| Disk (GB)                      | 30. The size on disk will depend on the size of the vulnerability database and the size of the scan reports. |

## V.3.3. WP5: Continuous Authentication

Table V-20: Computing Resource Estimates for Continuous Authentication

| Computation Resource Estimates |   |
|--------------------------------|---|
| CPU (vCPU)                     | 4   |
| RAM (GB)                       | 8.0   |
| Disk (GB)                      | 40. The size on disk will depend on the number of users, and the number of events generated by these users. |

## V.3.4. WP5: Data Gateway

Table V-21: Computing Resource Estimates for Data Gateway

| Computation Resource Estimates |  |
|--------------------------------|--|
| CPU (vCPU)                     | 2  |
| RAM (GB)                       | 8.0  |
| Disk (GB)                      | Minimal – excluding the storage space required for the FHIR resources. |

### V.3.5. WP5: Access Control Framework

Table V-22: Computing Resource Estimates for Access Control Framework

| Computation Resource Estimates        |                        |
|---------------------------------------|------------------------|
| Docker container and total file sizes | 16.4kB (virtual 337MB) |

### V.3.6. WP5: Network Slicing

Table V-23: Computing Resource Estimates for Network Slicing

| Computation Resource Estimates |         |
|--------------------------------|---------|
| CPU                            | TBD     |
| RAM (GB)                       | 8       |
| Disk (GB)                      | Minimal |

### V.3.7. WP6: Integration Toolkit

Requirements of Integration Toolkit will depend on the sum of all resources required by the other components assigned to the worker nodes of the cluster. The Integration Toolkit contains 2 clusters, one for the UI and other for the deployment of the applications. The tables below shows the suggested requirements with all components.

Table V-24: Computing Resource Estimates for K3S Cluster

| K3S (Rancher) Toolkit Computation Resource Estimates |     |
|--|-----|
| CPU (vCPU)   | 2   |
| RAM (GB)   | 4   |
| Disk (GB)  | ~30 |

Table V-25: Computing Resource Estimates for RKE Cluster Master Node

| RKE Master (Applications) Toolkit Computation Resource Estimates |     |
|--|-----|
| CPU (vCPU)   | 4   |
| RAM (GB)   | 4   |
| Disk (GB)  | ~30 |

Table V-26: Computing Resource Estimates for RKE Cluster Worker Node

| RKE Workers X2 (Applications) Toolkit Computation Resource Estimates |   |
|--|---|
| CPU (vCPU)   | 4 |
| RAM (GB)   | 4 |

|                  |     |
|------------------|-----|
| <b>Disk (GB)</b> | ~50 |
|------------------|-----|

Table V-27: Computing Resource Estimates for NFS Node

| <b>NFS (Storage) Toolkit Computation Resource Estimates</b> |      |
|---|------|
| <b>CPU (vCPU)</b>   | 2    |
| <b>RAM (GB)</b>   | 4    |
| <b>Disk (GB)</b>  | ~100 |

Table V-28: Computing Resource Estimates for Jenkins Master Node

| <b>Jenkins (Master) Toolkit Computation Resource Estimates</b> |     |
|--|-----|
| <b>CPU (vCPU)</b>  | 4   |
| <b>RAM (GB)</b>  | 4   |
| <b>Disk (GB)</b>   | ~30 |

Table V-29: Computing Resource Estimates for Jenkins Worker Node

| <b>Jenkins (Worker) Toolkit Computation Resource Estimates</b> |      |
|--|------|
| <b>CPU (vCPU)</b>  | 4    |
| <b>RAM (GB)</b>  | 4    |
| <b>Disk (GB)</b>   | ~100 |

## VI. INTEROPERABILITY

This section is concerned with the ability of ProTego resources to exchange and make use of information produced and managed by other resources. It includes two major updates to the interconnections within WP4 and WP5, as well as a section summarising the Dependencies and Interfaces from T6.1, for which no new items have been needed.

### VI.1. Interconnections

There are a significant number of disparate technologies within ProTego, and they all need to work together for ProTego to achieve its goals of securing the medical systems which it manages.

#### VI.1.1. Risk Assessment integration

The two core components of the Risk Assessment, the Security Information and Event Management (SIEM) and the System Security Modeller (SSM), need to integrate. The SSM requires the SIEM to be able to keep its security models of the system up to date and relevant. Likewise, the SIEM needs the SSM to aid identification of the rise of potential risks through the modeller. Each direction in this relationship is implemented independently and using techniques appropriate for ingestion by each technology.

To combine the capabilities from SSM with SIEM and provide an intelligent threat diagnosis at run-time, a bidirectional communication should be established between the two systems. The high-level architecture of the integrated components in WP4 is illustrated in Figure VI-I. To mediate the communication process, two components has been developed as part of the ProTego project:

- **a Run Time Microservice** which resides at SSM and is responsible for receiving messages from the SIEM system, updating system model parameters and re-calculating security risks at run-time, as well as providing recommendations for mitigation actions back to SIEM. The microservice provides a RESTful API for SIEM to communicate with; and
- **a Vulnerability Assessment (VA) component** which resides at SIEM. This component is comprised of two distinct processes; the VA Scheduler, which is responsible for periodically running vulnerability scans in the given system, monitored using OpenVAS for the system's infrastructure and OWASP ZAP for the system's web applications; and the VA Analyser, which accepts the output report from OpenVAS and ZAP, enriches it with additional data and transforms it to parse the information to the SSM microservice.

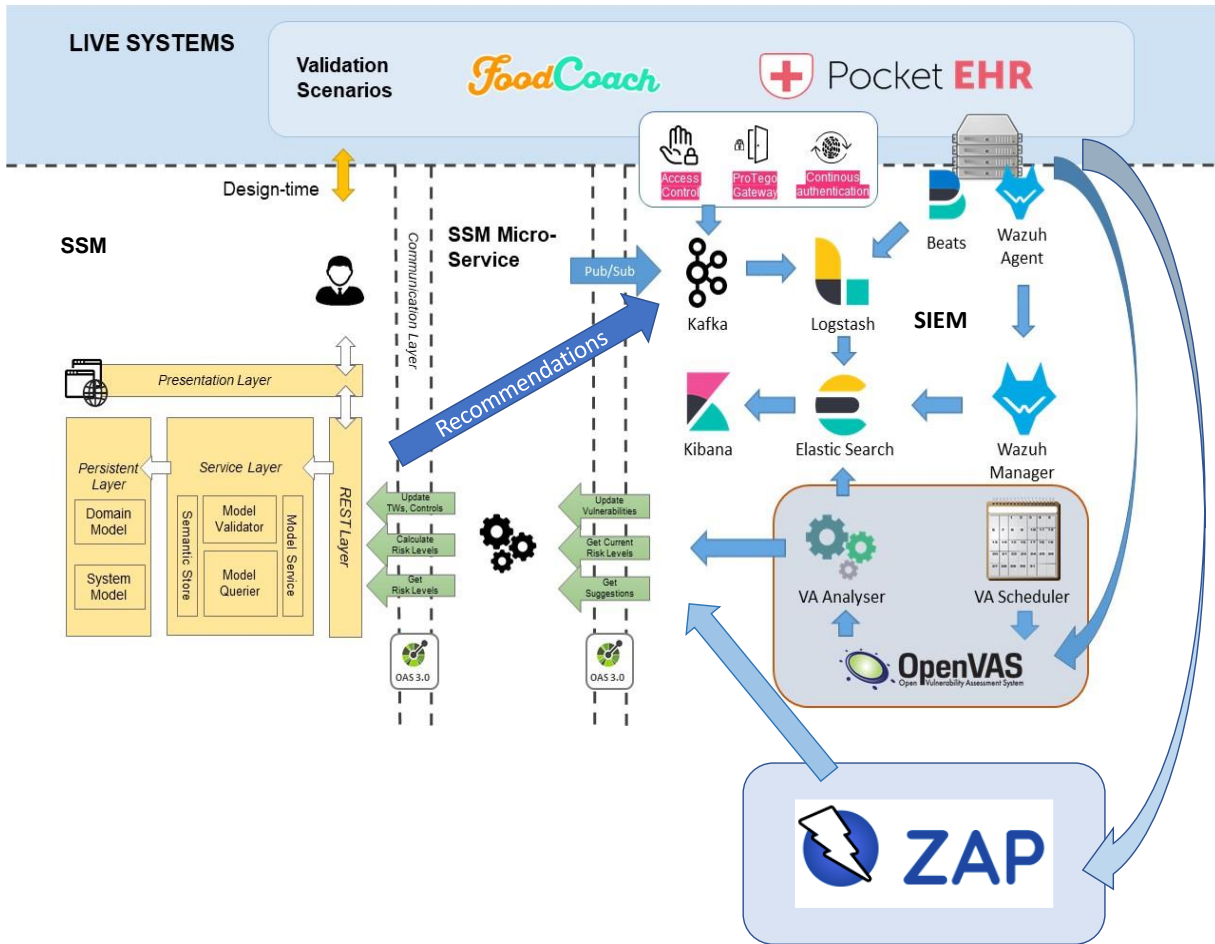


Figure VI-1 Information flow diagram of the integrated SSM-SIEM system.

Since the flow of events for the integration between the SSM and the SIEM is distinct, each direction of events flow is handled independently.

**From the SIEM to the SSM:** the SSM ingests data sent from the SIEM such that the SSM can update its models. This is supported through a REST interface described by an OpenAPI 3.0 YAML schema provided by the SSM microservice. The source of data is needed at the end of the processing, and as it is intended to include the use of ElasticSearch for the backend, ElasticSearch itself can be used as the data source for the SSM. Logstash is an open source data collection engine in ElasticSearch with real-time pipelining capabilities, and is thus an appropriate choice for pushing the data into the SSM from ElasticSearch.

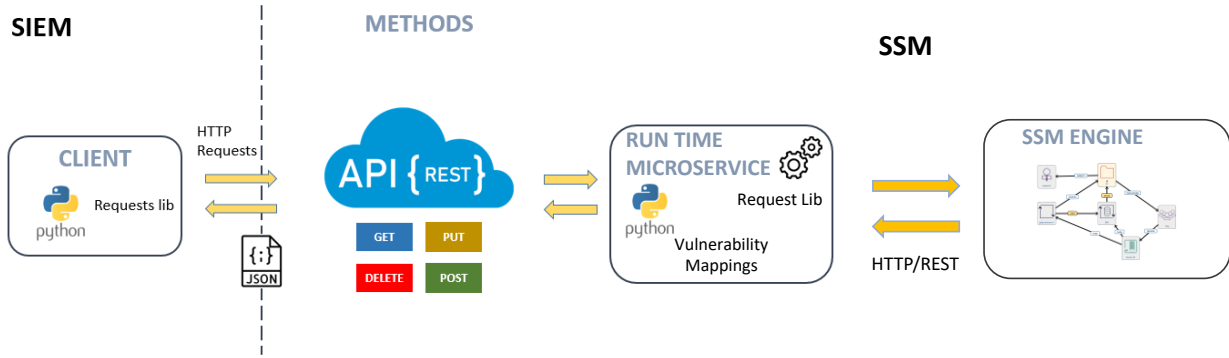


Figure VI-2 SIEM to SSM communication

**From the SSM to the SIEM:** SSM can directly inform SIEM regarding threats of high risk by message to SIEM and suggest potential controls and mitigation actions that can be followed via the SSM Run Time Microservice. The path here is different in that the SIEM is expecting a telemetry data source to put the data onto an Apache Kafka topic which resides in the SIEM system (Figure VI-III). The SSM microservice here acts as a Kafka Producer, where a Kafka client is defined (using a Kafka python library) by specifying the location of the Kafka Broker in SIEM. This data needs to be enriched with a baseline set of properties and in a specific JSON format. SIEM can consume the respective Kafka topic(s) and use the telemetry data received to display them in Kibana.

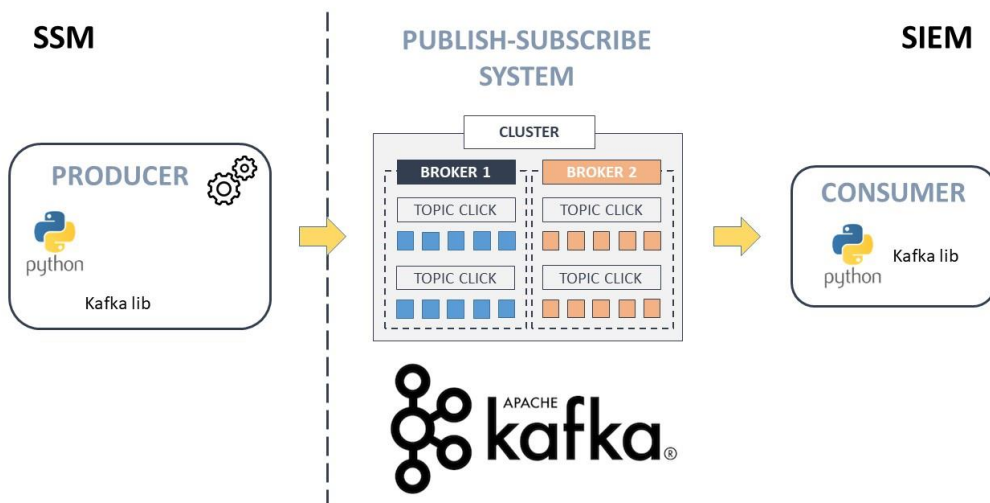


Figure VI-3 SSM to SIEM communication.

An extensive description of the components developed to mediate the communication between SSM and SIEM is provided in deliverable, D4.3.

### VI.1.2. Risk Mitigation Integration - Data Gateway

The Risk Mitigation aspect of the ProTego project needs to provide the capability for external interaction outside the bounds of the system in a secure manner. This is achieved using the Data



Gateway component with which the externally running applications securely communicate with the internal backend services such as Parquet.

The Data Gateway provides a secure interface using REST-based SQL-like query or FHIR resources for use by the medical applications so that they can query and store the data in encrypted Parquet files. The Data Gateway then utilises the Access Control Framework to identify the user and thus provide access only to authorised users. The Continuous Authentication system improves security on mobile devices and the Network Slicing improves the security at the network layer.

The WP5 integration involves initially three threads:

- **Container orchestration:** initially this includes the Data Gateway and the Access Control Framework. These have been orchestrated together using Docker and Docker Compose, and are developed within the Integration Toolkit to bring them to the ProTego platform. The backend parts of the other components will also be integrated this way.
- **Mobile Devices:** the continuous authentication components are focused on bringing increased security to mobile devices outside of the deployed toolkit.
- **Network Layer:** the network slicing aims to control the networking on the premise to increase security.

Integration is on-going, and the different components each need different methods of communication, including REST, 5G-EmPOWER, Open Flow and VLAN protocols.

## VI.2. Dependencies

The dependencies and coupling for the different aspects of the project have been kept to a minimum. Despite this, by necessity some components are bound together.

Table VI-1: Dependencies for ProTego

| Dependency                       | Status  |
|----------------------------------|---|
| Network Slicing and Provisioning | This technology is going to focus on manipulating the hardware of the hospital network infrastructure. So as planned it will be integrated into the final prototype as deployed in the use-case settings                                      |
| Data                             | The data will use FHIR protocol where applicable but additional SQL queries can the Data Gateway  |
| Access                           | Security provided by the Access Control Framework allows for secure connections with the Data Gateway integration to apply the mechanism to other components is ongoing.  |
| Communication                    | HTTPS, and REST API as well as the use of a Kafka message bus will be the main communication protocols, although additional protocols are required by the Network Slicing component to connect to the hardware routers of a hospital network. |

## VI.3. Interfaces

ProTego utilises several interfaces for communication and identification within the project.

Table VI-2: Interfaces for ProTego

| Interfaces      | Internal/External | Status  |
|-----------------|-------------------|---|
| OpenID Connect  | BOTH              | Is a secure interface that can be used for securing the connections between components  |
| REST            | BOTH              | Is the default for basic communication flow between components  |
| FHIR            | BOTH              | FHIR allows for specific data transfers and utilises the REST protocol.   |
| Kafka           | Internal Only     | Is used as a message bus to supply log messages from the component to the SIEM to monitor the security risk of the system                     |
| Network Slicing | Internal Only     | OpenFlow, NETCONF and 5G-EmPOWER protocols are used by the Network Slicing component to directly communicate with the hospital infrastructure |

## VII. RECOMMENDATIONS FROM D6.2 FOR FINAL PROTOTYPE

In D6.2 some recommendations were added. This section describes how these recommendations have been addressed for this Prototype.

### VII.1. Use-Cases

Recommendation from D6.2: Now we have an intermediate prototype of ProTego it is important to engage completely with the use-cases. This includes making sure Use-Case applications, FoodCoach, and Pocket EHR, are integrated with the ProTego toolkit.

For further details we can address D2.3 for further use-case requirements and descriptions of how each of the applications will be deployed.

How it has been addressed: The Integration Toolkit has been deployed to the case studies, at OSR and MS. In addition all ProTego components have been deployed at OSR, and is on-going for MS with some already deployed as the DataGateway and SIEM.

### VII.2. Iterative Workflow

Recommendation from D6.2: The next stage of ProTego will follow an iterative approach to finalise integration and bring the use-cases on board. Suggested goals for this iterative workflow include

- continuous Integration to manage the changes between versions;
- including example use-case applications on the central platform; and
- deploying the platform in a hospital environment.

This iterative workflow will be driven by these goals, but also as each change is managed by the continuous integration system it will allow for incremental updates to the system to deal with the challenges provided by these goals. At that point we will have the framework and components to deal with these challenges.

How it has been addressed:

- Continuous Integration has been deployed in the Integration Platform
- No use case application has been deployed in the Integration Platform since this is being tested in the case studies
- Platform has been deployed in both case studies: OSR and MS

## VIII. CONCLUSIONS

This document shows the architecture, requirements, and integration toolkit for ProTego.

The architecture is composed of several discrete components from WP4 and WP5 and the integration toolkit from WP6. Each component is contained with a Docker container allowing for easier integration.

The work done in the scope of WP6 related to the Integration Toolkit is described in detail with successful deployment of the Integration Toolkit and ProTego components in three different environments, Integration, OSR and MS.

Work in WP6 has also focus on supporting the partners in order to integrate their components and the case studies partners in order to deploy the platform.

Requirements, resources and component interoperability have been also revisited. The requirements include several aspects from D6.1 and D6.2, as well as updates based on the need for integration between components. We list the requirements and their success criteria.

This document addresses the interoperability between WPs as well as the dependencies and interfaces of the project.

Continuous Integration is an important part of the project and the successful deployment of a CI/CD pipeline in the Integration Environment has been also described.

Recommendations from D6.2 have been addressed successfully.



**ProTego**