D4.1
Design CS-AWARE common cloud-based software architecture and API specification

Grant Agreement number:	740723
Project acronym:	CS-AWARE
Project title:	A cybersecurity situational awareness and information sharing solution for local public administrations based on advanced big data analysis
Principal author:	Kim Gammelgaard, Rheasoft ApS, kim@rheasoft.dk
Co-author(s)	Peter Alfred Østergaard, Rheasoft ApS
Document version:	1.0

Table of Contents

Executive Summary

The current document is intended to give an overview of the architecture that is being used in Work Package 4 (WP4) for the common cloud-based software solution of CS-AWARE. It is not intended to specify the final product in technical detail.

The objective of WP4 is to deliver a common cloud-based solution based on the modules provided from WP3 and the CS-AWARE framework and requirements defined in WP2 [1].

Moving to the next generation infrastructure, namely cloud-based services, would enable the project to move faster with regards to development and deployment, than would be the case in traditional data centre-based setups.

In November 2018 however, at the third WP2-workshop in Rome, it became apparent, that in compliance with the IT-policies of Roma Capitale, the software had to be implemented in-house, at their local data centres, rendering our previously envisioned IT architecture, based on a number of Cloud services, unusable. Given this, we have re-thought the architecture in order to both provide a cloud service for LPA's and SME's that do not have their own data centres and LPA's who wish to deploy the system locally and have the capability to do so. [2] Therefore, we describe here, the solution that we have developed, which will run both in a local data centre environment and as a cloud-based solution.

The technical solution that we present here, is based on Docker components ready to be managed in a Kubernetes framework.

1 Introduction

The CS-AWARE project consists of a number of technical partners who each have their specialty and have their own way of dealing with development in terms of programming convention and programming language. The task of WP4 is, among others, to ensure a common software architecture and API's that will enable a final cloud deployment. As we have set out, the final deployment must be able to both be cloud based as well as data centre based.

The solution that we are developing is based on a fairly recent technology called Docker [3], which are self-contained full virtual computing instances that are able to communicate with other Docker images inside a system with little effort.

One advantage of using Docker instances is that each component can be maintained individually and updated without interfering with the common system, as long as the interface definitions are the same, so even though the partners have different approaches for the development phase, a common architecture is possible. As the components are not tied to specific hardware, it is also possible to scale the system in a fairly easy way, depending on the needs of specific customers, the level of log files and instances to be analysed, the number of cyber events to be tracked and similar highly volatile data needs.

To assemble the Docker instances, we use Docker Compose [4], a tool that enables multi-container docker applications.

When the Docker applications are running, the next step is envisioned, if required in a particular LPA is the development of a common orchestration platform, Kubernetes [5], that is able to handle management, scaling and automated deployment.

2 Overall Architecture

The latest overall architecture, depicted as an information flow model [diagram 1], shows the components as orange/yellow boxes. They are the components that must be connected in a common CS-AWARE system. The data collection is a split environment which partially resides locally in the network of the customer, while the other half will reside in the common CS-AWARE system, while the other parts are fully contained in the Docker Compose collection.

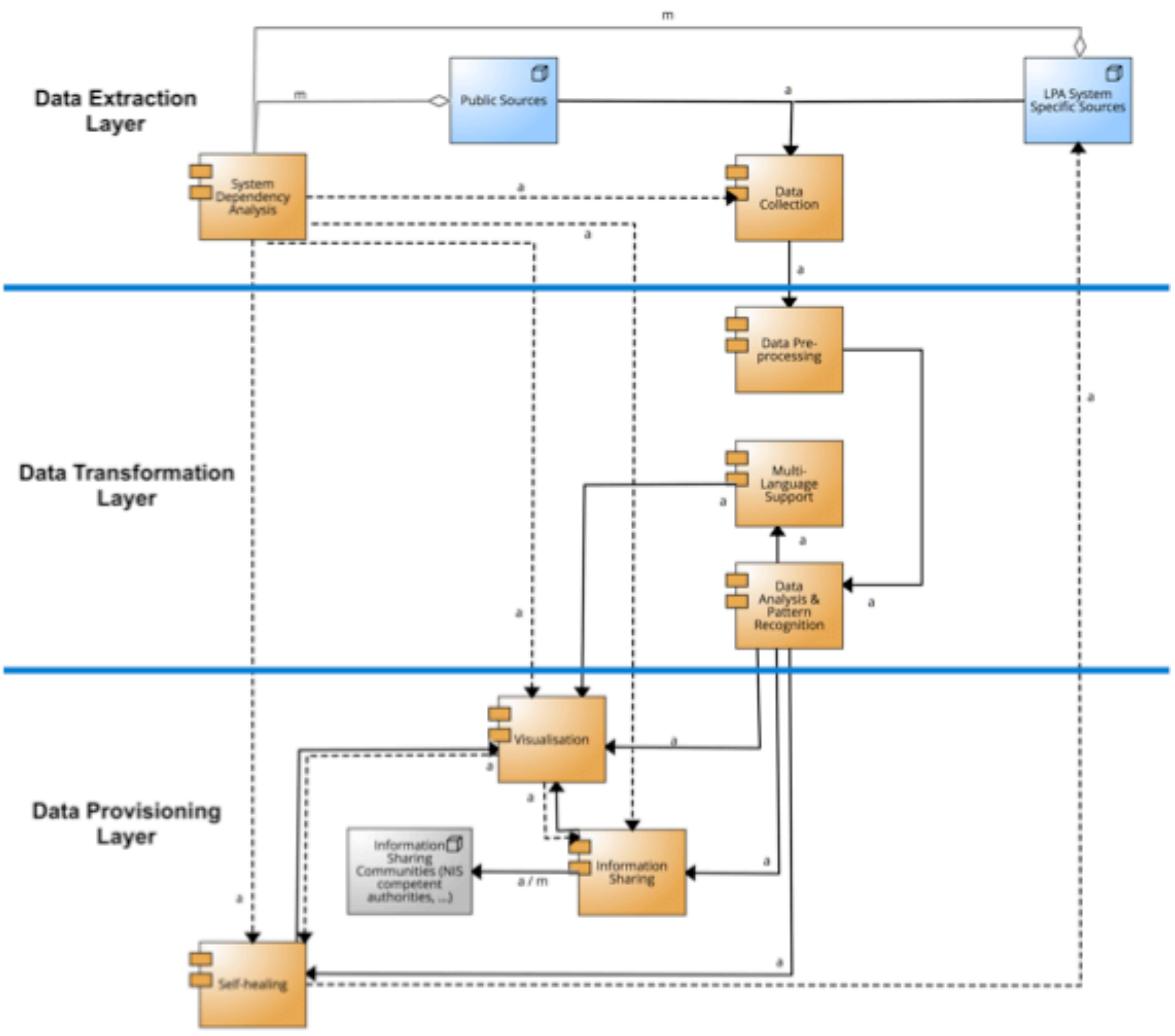
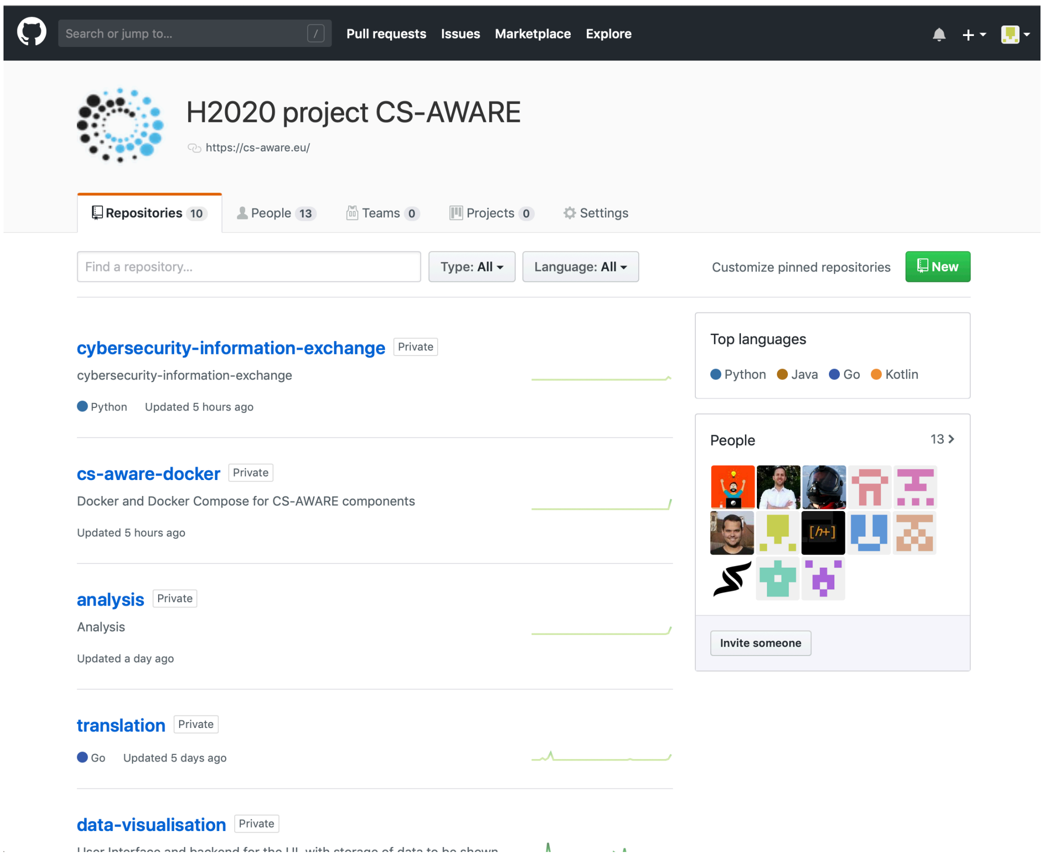


Diagram 1: current information flow model.

3 Docker Compose Content

The Modules for the Docker Compose content are first collected in a common, private CS-AWARE Github repository (see screenshot 1), from where the components are downloaded and assembled using Docker Compose. Notice that all projects are free to update their Docker module and that the current is always available.



H2020 project CS-AWARE

https://cs-aware.eu/

cybersecurity-information-exchange Private
cybersecurity-information-exchange
Python Updated 5 hours ago

cs-aware-docker Private
Docker and Docker Compose for CS-AWARE components
Updated 5 hours ago

analysis Private
Analysis
Updated a day ago

translation Private
Go Updated 5 days ago

data-visualisation Private

Top languages
● Python ● Java ● Go ● Kotlin

People 13

Screenshot 1: Repository of Docker instances to be assembled

The Docker Compose process is then defined by a YML-file which describes which docker instances to start, which dependencies they have, which port numbers that they will listen to, etc. This makes it possible to start the full application from any server, including high performance personal computers (for development), making implementation, maintenance and further troubleshooting easier than if we needed a number of server instances for the application to run.

4 Interfaces/API

The interfaces between the modules are primarily based on the STIX 2.0 standard [6]. STIX™, Structured Threat Information Expression, which is a language and serialization format used to exchange cyber threat intelligence (CTI).

STIX objects are represented in JSON and ensures that all information regarding CTI can be transferred in interfaces. As it turns out, state changes can be difficult to define inside the STIX format, and it may be too verbose for high speed/high volume purposes, so for specific tasks, simpler mechanisms are expected to be used.

The interfaces between the components inside the Docker Compose service rely on the single components and what has been defined and agreed upon in WP3. Using these interfaces, it is always questionable whether one component asks another component for input, or if one component is pushing input directly to another component. For some components, sending a request also requires an answer, for instance, if the Self-Healing component needs to know if a vulnerability needs to be resolved in a specific way, it needs to interface with the end user using the Visualization module, which is then responsible for handling the manual response from the end user and reporting back to the Self-Healing module.

We are still at the early stages of integrating the components, so the interfaces are not fully defined at this moment. For some interfaces asynchronous data transfer is a must, for other interfaces synchronous behaviour is expected. Regardless of asynchronous or synchronous the APIS is simplified by using JSON over HTTP.

Synchronous queries are expected to be HTTP GET requests returning JSON results. The query parameters could in most cases be HTTP query strings. In more complex queries a JSON input could be needed. Requests that hold new state are expected to be HTTP POST/PUT with JSON bodies representing the data input. Asynchronous requests are still standard HTTP GET/POST/PUT where the response is either queried for by the initiator or callback using a new HTTP request form the responder to the initiator.

5 Security/identity management

User identity and security will be handled in the user interface (UI), by OAUTH2/OIDC [7].
API security is minimal in the first version but will later be at network level and/or token based. Special consideration will be made where the APIs are accessible from outside CS-AWARE and also where CS-AWARE accesses the outside.

6 Next Steps

At this stage, we are in the process of deploying the Docker components containing the WP3 modules in Docker Compose and ensure that the data and control flow is possible. Only after this is in place, the components are able to interact, and cross-component development will take place.

The next step is to introduce the identity management/security level and Depending on the outcome of the cross-component development and enhancement, the needs for security/identity management, and the needs of the pilot LPA's, a decision will be taken if a second system management level, i.e. using Kubernetes, is needed and eventually to be implemented.

References

[1] D2.4 - CS-AWARE Framework
[2] D2.2- System and dependency analysis (second iteration) – Pilot scenario definition
[3] For further information on Docker see: https://www.docker.com
[4] For further information on Docker Compose see: https://docs.docker.com/compose/
[5] For further information on Kubernetes, see: https://kubernetes.io
[6] For further information on STIX see: OASIS Open. https://oasis-open.github.io/cti-documentation/
[7] For further information on OAUTH2/OIDC see https://openid.net/connect/