Grant Agreement No.: 761488

D2.1: CPN Reference Architecture

Content Personalisation Network. Towards an improved personal news offer, enabling economic impact for large and small news publishers.

Projectcpn.eu

The CPN reference architecture is the first deliverable of WP2, related to the CPN Platform.

Starting from the objectives of the CPN project, this document gathers the feedback from WP1 in terms of vision, scenarios and use cases and defines the CPN reference architecture. In the overall project course this will be followed by the implementation of the platform and integration of technology bricks and smart components (in WP3).

The first chapter highlights the general goals of the project and in particular those relevant for the development of the platform. It focuses also on the purpose of the whole deliverable, the role and impact of the platform within the project and in the news publication and content personalization markets, at European level.

Chapter two focusses on the approach used to define the architecture. The process started from initial requirements elicited in WP1 and relies on the identification of the main actors to whom the platform is addressed: end-users and media professionals. Based on these requirements, the main needs that the platform must satisfy were identified. Different possible software architecture designs were analyzed based on a state of the art analysis of service oriented architectures and microservices architectures. The microservices architecture proved to be the best choice. With this kind of architecture, the services can operate and be deployed independently from other services. This makes it easier to integrate services leveraging on existing technology bricks and smart components, to deploy new versions of services frequently and to scale up a service independently.

Chapter three focusses on the platform implementation details. The design specifications, the patterns to be adopted and the fundamental components that will constitute the basis of the CPN platform are described.

Chapter four introduces the guidelines for the development, communication and deployment of the individual components of the platform. The CPN platform will have "core" services that will be defined in subsequent WPs (esp. WP3). These services will be integrated with each other to offer a series of features that accomplish the user requirements. Finally, as the CPN platform must be flexible and extensible, open to future integration of new services, rules have been defined for the standardization of services and for interoperability.

Co-funded by the Horizon 2020
Framework Programme of the European Union

Work package	WP 2
Task	2.1
Due date	31/01/2018
Submission date	31/01/2018
Deliverable lead	ENG
Version	1.0
Authors	Ferdinando Bosco (ENG) Vincenzo Croce (ENG)
Reviewers	Fulvio D'Antonio (LiveTech) Petru Buzulan (LiveTech)
Keywords	CPN Reference Architecture - Content Personalisation - Microservices - Open Virtual Platform

Co-funded by the Horizon 2020
Framework Programme of the European Union

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	13/12/2017	1st version of the deliverable with table of contents	Ferdinando Bosco (ENG) Vincenzo Croce (ENG)
V0.2	20/12/2017	draft version of deliverable for contributions by partners	Ferdinando Bosco (ENG) Vincenzo Croce (ENG)
V0.3	16/01/2018	version of deliverable with contributions from ATC and integration of architectural description	Ferdinando Bosco (ENG) Vincenzo Croce (ENG) Marina Klitsi (ATC) Nikos Sarris (ATC) Stratos Tzoannos (ATC)
V0.4	23/01/2018	version of deliverable with LiveTech, DCat and Imec contributions, ready for internal review	Ferdinando Bosco (ENG) Vincenzo Croce (ENG) Marina Klitsi (ATC) Nikos Sarris (ATC) Stratos Tzoannos (ATC) Michele Nati (DCat) Maria Prokopi (DCat) Matthias Strobbe (Imec) Fulvio D'antonio (LiveTech) Petru Buzulan (LiveTech)
v1.0	29/01/2018	final version of deliverable	

Co-funded by the Horizon 2020
Framework Programme of the European Union

DISCLAIMER

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 761488.

Project co-funded by the European Commission in the H2020 Programme		
Nature of the deliverable:	**Report**	
Dissemination Level		
PU	Public, fully open, e.g. web	X
CL	Classified, information as referred to in Commission Decision 2001/844/EC	
CO	Confidential to CPN project and Commission Services	

Co-funded by the Horizon 2020 Framework Programme of the European Union

EXECUTIVE SUMMARY

The main goal of CPN project is to create innovation in the way content creators can structure content production, distribution and in-depth interaction with audiences.

To do this CPN will realize an innovative Open Virtual Platform.

The microservice architecture offers a series of benefits well suited for the CPN platform requirements. A process of selection of the most proper model involved a comparison between Service Oriented Architecture and the microservices models. The analysis brought to the choice to implement a reference architecture referring to microservices model for many reasons including the possibility to have an easy and robust development and integration process for the components realized by different partners. Moreover the specific implementation was aimed at obtain the independence between applications, infrastructure, deployment and operation environments; this constitutes an easy environment for the development of solutions exploiting CPN.

This Reference Architecture document contains the guidelines and the template of the architecture to be used in CPN to develop this Open Virtual Platform.

Starting from the goals and requirements of the CPN project, the document describes the design and implementation choices to obtain a flexible and extensible architecture and the guidelines for implementation, deployment and communication among components.

Furthermore interoperability rules have been defined for the integration of external components and the creation of new features.

Co-funded by the Horizon 2020
Framework Programme of the European Union

TABLE OF CONTENTS

Co-funded by the Horizon 2020
Framework Programme of the European Union

LIST OF FIGURES

Co-funded by the Horizon 2020
Framework Programme of the European Union

LIST OF TABLES

Co-funded by the Horizon 2020
Framework Programme of the European Union

ABBREVIATIONS

API	Application Programming Interface
AMQP	Advanced Message Queuing Protocol
ATC	Athens Technology
DBs	Databases
CPN	Content Personalisation Network
DCat	Digital Catapult
DIAS	Dias Media Group
DW	Deutsche Welle
e.g.	Example given
ENG	Engineering Ingegneria Informatica
ESB	Enterprise Service Bus
esp.	Especially
etc.	Etcetera
FOMO	Fear of missing out
i.e.	That is
ICT	Information and Communications Technology
imec	Interuniversity MicroElectronics Center
IT	Information Technology
JMS	Java Message Service
JSON	JavaScript Object Notation
Mb	Megabyte
MSA	MicroServices Architecture
REST	Representational State Transfer
SOA	Service-oriented Architecture
VRT	Vlaamse Radio-en Televisieomroeporganisatie
Vs.	Versus

Co-funded by the Horizon 2020
Framework Programme of the European Union

WAN-IFRA	World Association of Newspapers and News Publishers
XML	eXtensible Markup Language

Co-funded by the Horizon 2020
Framework Programme of the European Union

1 INTRODUCTION

The challenge of the CPN project is to design and develop a method to connect millions of users to millions of content items in an advanced, personalized and innovative way while preserving the European media diversity.[1]

In this chapter we will analyze the goals of the project, the scope of this document and the impacts that it could have in the European context.

1.1 GOALS[2]

CPN tackles the challenge of developing a new approach for the personalisation of digital content, allowing both large and small media companies to benefit from being able to better target content to media consumers.

From the viewpoint of the media consumer, the challenge is to enable a better delivery of news, insights and information in the right format at the right time.In short, the core of CPN is to create innovation in the way content creators can structure content production, distribution and in-depth interaction with audiences.

- ➔ offer media professionals faster and more targeted cross-channel news & information distribution solutions as well as novel personalisation services, fully compliant with already existing content and operational infrastructure
- ➔ enable the creation of end user applications that offer users a more attractive and engaging news & information experience.

CPN will realize this by the creation of an innovative virtual platform.

The platform is not intended to be a physical system which media companies should install, instead, the CPN platform will be a virtual platform, together with a reference architecture, which can be adapted to the needs of the media company by connecting different services together.

Every media company, whether large or small, will maintain their own platforms and systems which can be extended with the CPN services and modules by interfacing with the virtual CPN platform.

As a result, this virtual platform is not conceived to function on its own. Different innovative services combining content and personal data need to be integrated with it in order to implement successful content personalisation strategies.

Furthermore, the platform is foreseen to be an open platform in the sense that it will allow external ICT providers to integrate additional services onto the platform.

[1] as stated in the CPN Grant Agreement (Part B p.4)
[2] as stated in the CPN Grant Agreement (Part B p.5)

Co-funded by the Horizon 2020
Framework Programme of the European Union

1.2 Objective

The primary objective of this document is to explain the CPN reference architecture, i.e. a template for specifying concrete system architectures. This architecture is designed to support a wide range of deployment scenarios and use cases and to fit the requirements of large and small media companies.

It does not just define a single architecture implementation, but defines a template for designing a flexible platform that offers different services for each stakeholder.

Moreover, this reference architecture allows reuse of standard IT systems and provides extensibility and openness in adopting future developments and technologies.

1.3 Impacts

Through an open virtual platform CPN will offer a series of powerful and innovative services divided in three main categories:

- User Services (to gain actionable insights into the audience's consumption patterns)
- Content Services (e.g., to analyze content items and extract higher level concepts, categories, viewpoints, etc.)
- Mapping Services (e.g., to enable personalised, contextualized recommendations)

Through these services a fully personalised and interactive user experience is guaranteed, offering users novel, personalised news feeds. By developing these technologies, CPN will have a major impact in terms of offering qualitative personalised news feeds on a European-scale level.[3]

[3] as stated in the CPN Grant Agreement (Part B p.28)

2 ARCHITECTURE SELECTION

This chapter summarizes the system and user requirements that are used as the base for the CPN Architecture and the process we adopted to select the architecture model.

2.1 KEY DRIVERS

The CPN Architecture has as concrete goal: the realization of an innovative virtual platform, flexible and extensible to future improvements.

The key drivers/requirements of CPN architecture are:

- ➔ creation of new features starting from already existing components
- ➔ the features offered by the platform must be usable by different client applications (web, mobile, smart tv, etc.)
- ➔ the platform must define some rules and interfaces for the integration of new components and the extension of the platform itself with new features

These requirements are clear and well defined within the project, while more detailed use cases, scenarios and functional requirements are being developed and will be available later in the project.

2.2 USER REQUIREMENTS & TECHNOLOGY BRICKS

At the time of writing of this document, the list of use cases is not fully defined yet. Therefore we used a more generic approach to design the reference architecture, exploiting the available information from the general user stories defined in the CPN Grant Agreement and from the inputs obtained from the requirements elicitation performed in WP1.

The features offered by the CPN platform are aimed at two types of users: end-users and content producers.

For both types of users, requirements collection activities are in progress, which will serve the identification of use cases and scenarios.

The approach used to define the architecture of the CPN platform allows us to obtain a result even exploiting use cases that are not fully defined yet. We started from the goals of the project, already available use cases and from the list of already existing, or at least defined, modules, called technology bricks, offered by the various partners (in WP3). These modules form the baseline for the definition of the "atomic" services that will realize the basic features offered by the platform.

Following the verification of the requirements, these services will eventually be combined with new ones, to offer additional and increasingly innovative features.

For a good understanding, in the following section, requirements for both identified platform actors and the list of technology bricks from which we will start to create the CPN platform, are listed.

Co-funded by the Horizon 2020
Framework Programme of the European Union

2.2.1 User Requirements

This section provides a list of possible requirements that the platform should meet taking into account the project objectives, some examples of user stories described in the CPN Grant Agreement and the first results extrapolated from the requirements analysis performed in WP1.

Starting from the identification of the actors, we divide the requirements into three categories:

User profile

Requirements related to the enrichment of the user's profile to allow a more effective personalization of the content.

For example:

- ➔ Collect user interests
- ➔ Propose personalized contents
- ➔ Analyze user behavior in different contexts (e.g. at home, at work, commuting, on holiday, etc.)
- ➔ ...

Application Features

Requirements for innovative services that do not directly affect the user profile, but contribute to improving the user experience of applications related to the CPN platform.

For example:

- ➔ Bursting ‘Filter Bubbles’
- ➔ Avoiding Fear of missing out (FOMO)
- ➔ Transparency of how user profile data is used for personalization
- ➔ ...

Production Side

Requirements related to the features offered to professionals (media, journalists, content creators), both in terms of content creation, analysis and monitoring.

For example:

- ➔ Easier content creation, management of responsibilities and accounting of revenues
- ➔ Powerful distribution of contents
- ➔ Access to relevant user data, insights and analytics
- ➔ ...

Co-funded by the Horizon 2020
Framework Programme of the European Union

2.2.2 Technology Bricks

After an initial analysis, the technology bricks that will probably be part of the basic version of the platform have been identified (the final version will be released with the deliverable of the technical requirements).

As output of WP3[4], this table summarizes the functionality of these bricks and indicates the partner that will develop them.

Table 1: Technology Bricks

Name	Owner	Description
	WEB APP FOR PRODUCERS	
Storyfication - Cute for LArge Event	ENG	Cute4LE (http://cute.eng.it/about) is a content curation platform for Marketing support using a storytelling approach. It is an evolution of the Cute tool and specialized for Large Events management. The platform allows to create stories reusing and embedding content, including user generated content harvested from web and social networks. User engagement mechanisms, livestream & Territory monitoring, influencers & trending topics management and analytics processes are blended together with the aim to exploit social network dynamics and monitor the activities related to specific events. The mechanisms cover different phases from pre-event to post-event. The Cute4LE modules and technologies will be used in the realization of the CPN platform to support media professionals to collect and use existing (user generated) content for the storytelling to address audiences with effective gamification processes.
Reward Framework	DCat	The Reward Framework provides a tool for content producers to form ad hoc teams needed for the creation of a given item of media content, e.g., requiring multidisciplinary expertise, including video production and editing capabilities, text writing skills, photography experience, etc. A number of pre-defined contracts are available for selection and agreement among team members and content distributors, while other tools allow digitisation of different team members' contributions, register them as digital assets and share rewards following their use by the content distributors. Smart contract and distributed ledger technologies are used to independently enforce such contracts and transparently and automatically account for generated revenues.
	WEB/MOBILE APP FOR READERS	
Collaboration Platform	ATC	Truly Media is a collaboration platform that helps users gather, organise and verify content. It has been jointly

[4] Technology Bricks description-v3

Co-funded by the Horizon 2020 Framework Programme of the European Union

		developed by ATC and DW and is available as a commercial service. Technologies from Truly Media can be reused to setup a collaboration platform in CPN that will allows professional users curate (gather and annotate) content.
USER PROFILE		
User modelling	LiveTech	A tool that is capable of creating user profiles at different levels of granularity; part of the profile is based on the clickstreams patterns of the user (obtained by using the context training methodology previously mentioned); other important dimensions of the user’s profile will be constituted by an unsupervised topic extraction starting from the user’s consumed content (news, articles read). Moreover, by means of a social login, can be harvested from several social networks basic profile information, summary statistics regarding their activity on the network, the number of posts made, likes, comments, groups they are member of etc.
Assessment of content trustworthiness	ATC	TruthNest (www.TruthNest.com) is an analytics tool that aids professionals, as journalists or business analysts, in exploring information found in Social Media and assessing its trustworthiness. Modules and technologies will be adapted from TruthNest to aid in the analysis of content.
Personal Data Receipt	DCat	Personal Data Receipts are a tool compliant with GDPR Articles on Information Notice and aiming to simplify users’ understanding of privacy policies, while providing them with a human-readable record on what personal data are collected, the purpose of use they have consented to, and for how long they will be stored. PDRs are an instrument to allow users to ask for data removal or for executing other digital rights. Integration with blockchain is leveraged to provide a non-repudiable receipt record, useful for future verification that personal data are used according to the user’s wishes. This tool will be tested during the project to gather user feedback for further refinement and with adopters (e.g., content distributors) in order to derive recommendation for standardized PDRs.
PRODUCERS’ CONTENT ANALYSIS & RETRIEVAL		
Hyper-parametric optimization of recommender engines	LiveTech	The current state of the art techniques for recommending items are based on two main areas: content based (that relies on good semantic modelling/feature extraction and selection on the items to be recommended) and collaborative filtering techniques (that are essentially domain-independent and take into account network metrics based on emerging similarity graphs of users and items). This module uses a hybrid approach that uses variable proportions of both techniques for each user learning (using Machine Learning techniques) from explicit and implicit feedback given by the users themselves: clicks, ratings, sharings, etc. The module is customizable for including content-delivery strategies’ optimization:

Co-funded by the Horizon 2020
Framework Programme of the European Union

		multichannel and date/time optimization (predicting the probability of interests at a given time on a given channel) and includes mechanisms for fostering "serendipitous" discoveries.
Semantic lifting of plain-text content & knowledge graph build-up	Imec	Extensive experience in lifting raw data to a more interoperable, semantically queryable form. On the one hand, this can be in the form of converting semi-structured data (CSVs, proprietary formats, …) using our mapping toolchain (rml.io). On the other hand, it can be in the form of extracting unambiguous entities from raw text, and creating a queryable knowledge graph to make the content more discoverable (e.g., http://uvdt.test.iminds.be/storyblink/). Despite the significant number of existing tools, generating Linked Data by incorporating heterogeneous data from multiple sources and different formats remains complicated, let alone generating their metadata. A sustainable semantic-driven approach, based on the RML toolchain (RML Mapper, RML Workbench, RML Editor and RML Validator), can address the shortcomings of current Linked Data generation tools and enables data owners to generate high quality Linked Data by themselves. We facilitate and automate the generation of high quality Linked Data with accurate, consistent and complete metadata, offering a granular, sustainable and generic solution that shortens the Linked Data generation workflow, and achieves higher integrity within Linked Data. The required input for building this system is -a large corpus of data sources whose semantic annotations are provided by the users relying on the aforementioned technologies. -a large training set (size in order of at least 100 data sources) with similar content to train the automated rules generation to automate the Linked Data generation. -a large corpus of data sources containing information on the entities.
Generic Training model for Relation Extraction	Imec	Extensive experience with the design and efficient training of systems to detect relations between entities from plain text (e.g., is-CEO-of (Steve Jobs, Apple)). KBP systems extract information on specific persons (e.g., birth dates, spouses,...) or organisations, (e.g., founders, member organisations,...) from text collections containing hundreds of thousands unstructured news articles. Text is processed by multiple modules from the KBP system before the missing fields are extracted from the text. These include an information retrieval component (IR), an entity linking system component (EL), a named entity recognition system (NER) and a relation extraction module (RE). The IDLab KBP system focuses on improved relation extraction between entities. Our current setup is targeted at slot filling: a trained system can take as input a relationship tuple with a blank entry to fill (e.g., subject="Trump", predicate="president_of", object=?)

Co-funded by the Horizon 2020 Framework Programme of the European Union

		and predict the most likely filler (e.g., object="United States"). The required inputs for the system to make it operational for other relations are: set of predefined relations (e.g., our current system is designed for those defined in http://surdeanu.info/kbp2014/TAC_KBP_2014_Slot_Descriptions.pdf) training data: sentences with annotated entities (e.g., persons, organizations) and the relation(s) expressed between them within that sentence a large corpus of documents (typically news articles or wikipedia data) containing information on the entities and relations of interest.
Term and OntoExtractor	LiveTech	Domain independent terminological, taxonomical and ontological extraction from unstructured sources based on metrics and strategies based on statistical, linguistics and extra-linguistic features (e.g. text tagging, position, etc.). The methodology and tool have been used in several academic (European and National projects) and Industrial applications. This can be used for the unsupervised topic extraction.
Stance detection	Imec	Classifying the stance of the view expressed in an article on a certain topic (e.g., political view, left- vs right-wing; or positive/neutral/negative in a more basic setting). This can be used for the unsupervised perspective extraction. We have neural network based setups for different problems with the same general goal of sequence classification. A baseline system for stance detection can therefore be set up reasonably quickly (1 to a few weeks). The required input for building this system isa large training set (size in order of 100k) of sequences (sentences, paragraphs) with for each sequence an indication of (1) the type of view, i.e., possible stances (e.g., political sides), and (2) the stance expressed in the sequence). Existing datasets focus on narrow cases (e.g., Trump vs Clinton in tweets on US elections)
Sentiment and Polarity Miner	LiveTech	Multiplatform, micro-service based trainable classification system for performing: polarity detection, Complex opinion and sentiment extraction (via unsupervised techniques), Spam, hate-speech and flames detection.
Context-dependent recommendation	Imec	Refining and personalising recommender systems, targeted to adjust the selection of recommended items to the location and context (e.g., at home, at work, commuting...) of the user as well as taking into account users' willingness to consume content given context. We have a series of implemented models in place that take content, users, and arbitrary context features, and predict the most interesting items. These models were developed in an off-line recommender evaluation setup and were not designed for real-time deployment. Main difference with respect to existing recommender setups is the highly dynamic character of the content (e.g., short life time for news articles).

Co-funded by the Horizon 2020 Framework Programme of the European Union

		In order to test these models, we need user click data involving -'clicks': each with user id, timestamp, content id, and click context features (e.g., location) -ideally: 'content': corresponding content for all content id's (if not available: purely based on collaborative aspect and context) -ideally: selection of content items presented at the user at the time of the click (if not available: assuming equal access to all content, or only most recent).

In section 3.3.1 about the implementation of the architecture, an example workflow will be presented including some of these modules, to better understand how these modules relate and integrate with each other, to fulfill the requirements of the platform.

2.3 Architecture Candidates

As the CPN platform needs to offer multiple services to different client applications, as clearly explained in both the project objectives and the first phase of requirements elicitation, a service-oriented architecture needs to be defined.

There are several types of services architectures: from the generic SOA (service-oriented architecture) to its more specific MSA implementation (microservices architecture).

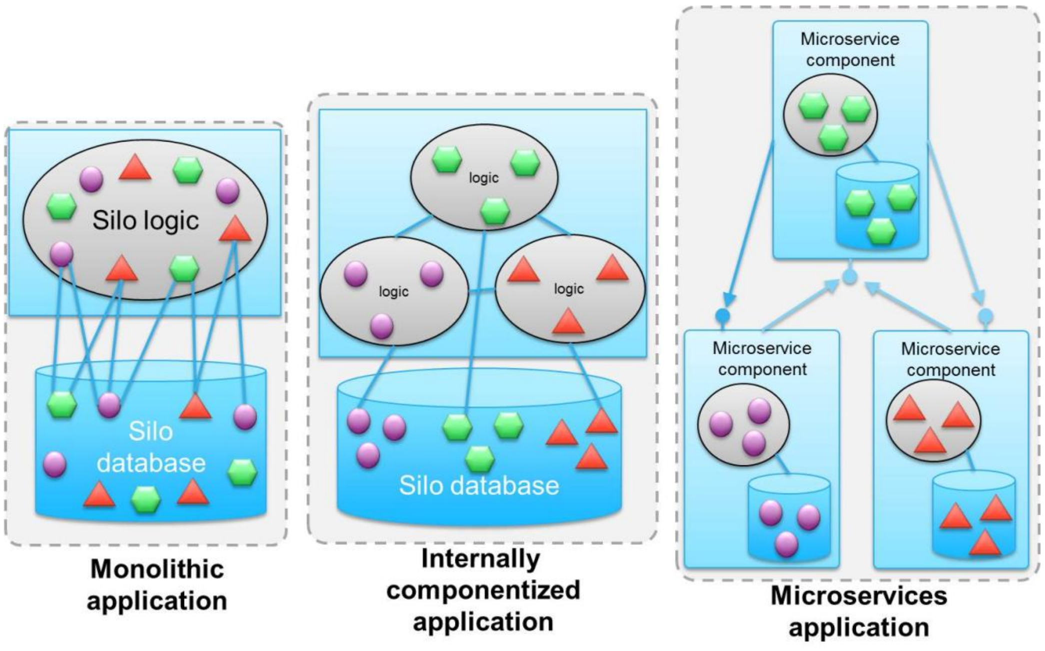

Figure 1: From monolithic applications to microservices[5]

We took these two types into consideration and analyzed what would be best choice for the CPN platform.

[5] https://www.ibm.com/developerworks/websphere/library/techarticles/1601_clark-trs/1601_clark.html

Co-funded by the Horizon 2020 Framework Programme of the European Union

SOA

A Service Oriented Architecture is a software architecture pattern, where application components provide services to other components via a communication protocol over a network. The communication can involve either simple data passing or it could involve two or more services coordinating connecting services to each other. Services (e.g., RESTful Web services) typically carry out some small functions, such as validating an order, activating an account, or providing shopping cart services.

There are 2 main roles in SOA: a service provider and a service consumer. A software agent may play both roles. The Consumer Layer is the point where consumers (human users, other services or third parties) interact with the SOA and the Provider Layer consists of all the services defined within the SOA.

MSA

Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities. The microservice architecture enables the continuous delivery/deployment of large, complex applications, composed of small, independent processes communicating with each other using language-agnostic APIs. It also enables an organization to evolve its technology stack.[6]

In a microservice architecture a service should be **independently deployable,** or be able to shut-down a service when is not required in the system without affecting any other service.

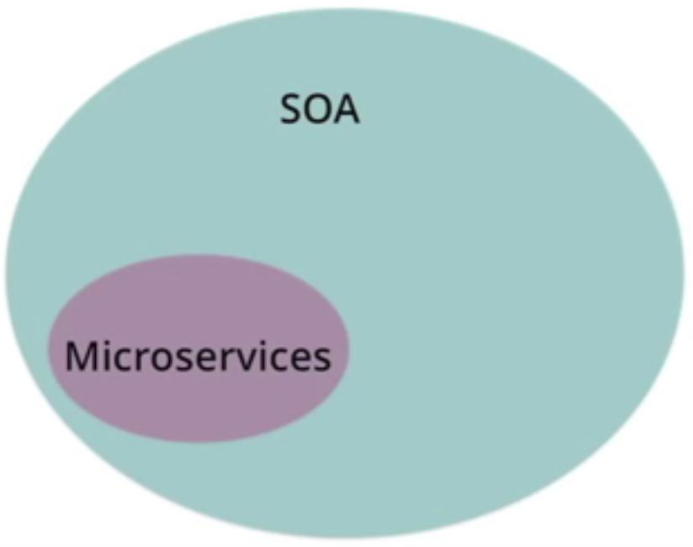

Figure 2: Microservices is an implementation of SOA

SOA vs MSA

Both architectures have similar pros and cons and some differences. In both architectures, each service - unlike a monolithic architecture - has a certain responsibility. Thus, services can be developed in various technology stacks, which brings technology diversity into the development team. The development of services can be organized within multiple teams, however, each team needs to know

[6] http://microservices.io/index.html

Co-funded by the Horizon 2020
Framework Programme of the European Union

about the common communication mechanism for inter-service communication within the overall SOA.

While iIn microservices the, services can operate and be deployed completely independent of other services, it's not always which is often not possible in other SOAs. This makes it easier to deploy new versions of services frequently or scale a service independently.In both architectures, developers must deal with the complexity and distributed nature of the architecture. Developers must implement the inter-service communication mechanism between the different microservices.

In typical SOAs, services share the same data storage while each service can have an independent data storage in microservices. Last but not least, the main difference between typical SOAs and microservices relates to size and scope. A typical microservice is significantly smaller than a regular SOA and is mainly a small(er) independently deployable service. On the other hand, a classic SOA can be either a deployment monolith or it can be comprised of multiple microservices.

2.3.1 Selected Architecture

"Microservices are the kind of SOA we have been talking about for the last decade. Microservices must be independently deployable, whereas SOA services are often implemented in deployment monoliths. Classic SOA is more platform driven, so microservices offer more choices in all dimensions."-

Torsten Winterberg Oracle ACE Director

Based on the available requirements the MSA architecture proved to be the best choice for the development of the CPN virtual platform.

An MSA architecture offers us a series of benefits:

➔ The microservices can be **independently developed and deployed**, this satisfies the need to have components developed by different partners, which offer the different functionalities for the platform

➔ Microservices are designed to offer a range of "micro" capabilities that aggregate together to **cover a variety of platform use scenarios**

➔ The microservices **do not bind the platform to the use of a given technology stack**

➔ The MSA lends itself well to both the creation of new services and the implementation of new processes, even after the deployment of the architecture, this meets the criteria of **interoperability, flexibility** and **openness** of the platform.

Co-funded by the Horizon 2020 Framework Programme of the European Union

3 REFERENCE ARCHITECTURE

In this chapter we describe the chosen architecture, the design choices and some views that demonstrate the overall vision of the architecture.

There are several models for implementing an MSA. During the design phase, we evaluated some of these patterns and made some architectural choices to achieve the project goals.

In particular, we focused on the possible methods of communication between different microservices (Inter-service / process Communication) and data management.

3.1 DESIGN

The following patterns have been identified for an effective implementation of the architecture that meets the objectives and pre-established requirements:

API Gateway Pattern[7]

In a MSA each service offers a list of APIs. The granularity of these APIs is often different than what a client needs. Microservices typically provide fine-grained APIs, which means that clients need to interact with multiple services.

These services often use a diverse set of protocols, some of which might not be web friendly.

Furthermore, in CPN, we have different kinds of client applications (web, mobile, etc.) that probably need different kinds of data.

These issues can be solved by implementing an **API gateway** that is the single entry point for all clients.

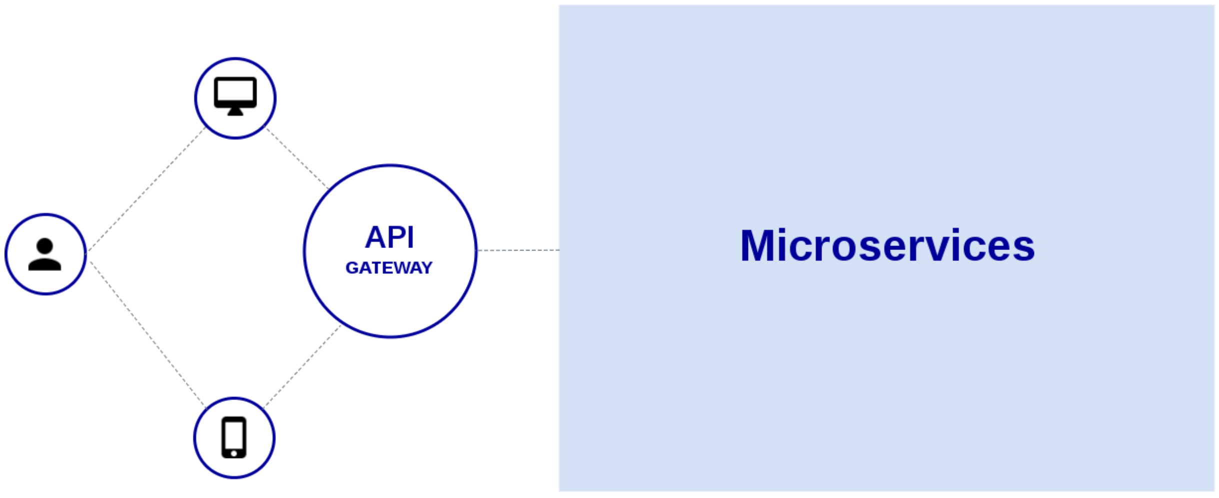

Figure 3: The API Gateway Pattern

[7] http://microservices.io/patterns/apigateway.html

Co-funded by the Horizon 2020 Framework Programme of the European Union

We will discuss this API gateway component in more detail, in section 3.3 describing the implementation of platform components.

Messaging Pattern[8]

In a MSA, services must often collaborate to handle requests from client applications and to offer features. These services must use an inter-process communication protocol, like asynchronous messaging, and a platform that allows to exchange messages is Apache Kafka.[9].

This pattern has the following benefits for our architecture:

- Greater flexibility for the platform, increasing the decoupling of services
- Improved availability since the message broker buffers messages until the consumer is able to process them

The main component of this pattern is the Message Broker. We will discuss this in more detail in section 3.3.

Database Per Service Pattern[10]

Typically, in a MSA the services have different data storage requirements. For some services, a relational database is the best choice. Other services might need a NoSQL database such as MongoDB, which is good at storing complex, unstructured data, or Neo4J, which is designed to efficiently store and query graph data.

A good approach for this need is to use the Database Per Service Pattern, keeping the microservice's persistent data private to that service and only accessible via its API.

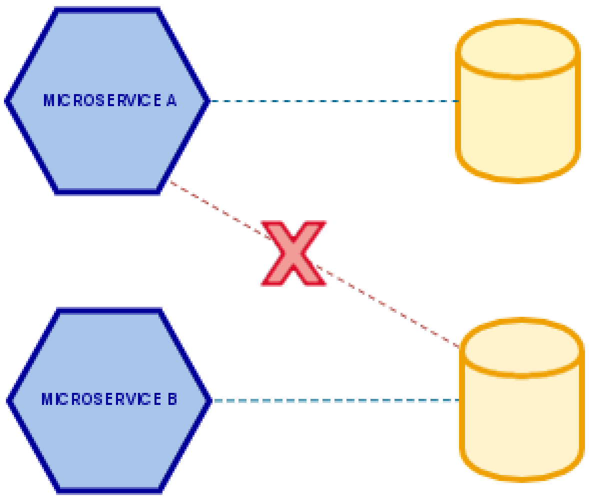

Figure 4: Database per Service Pattern

[8] http://microservices.io/patterns/communication-style/messaging.html
[9] http://kafka.apache.org/

[10] http://microservices.io/patterns/data/database-per-service.html

Co-funded by the Horizon 2020
Framework Programme of the European Union

Using a database per service has the following benefits:

- ➔ Helps to ensure that the services are loosely coupled. Changes to one service's database does not impact any other services.
- ➔ Each service can use the type of database that is best suited for its needs

Unfortunately, there is also a drawback: it is difficult to implement transactions that require multiple services. To solve this problem, the best solution is to use the Saga Pattern[11], in which a service publishes an event when updating data, and other services subscribe to these events and update their data in response.

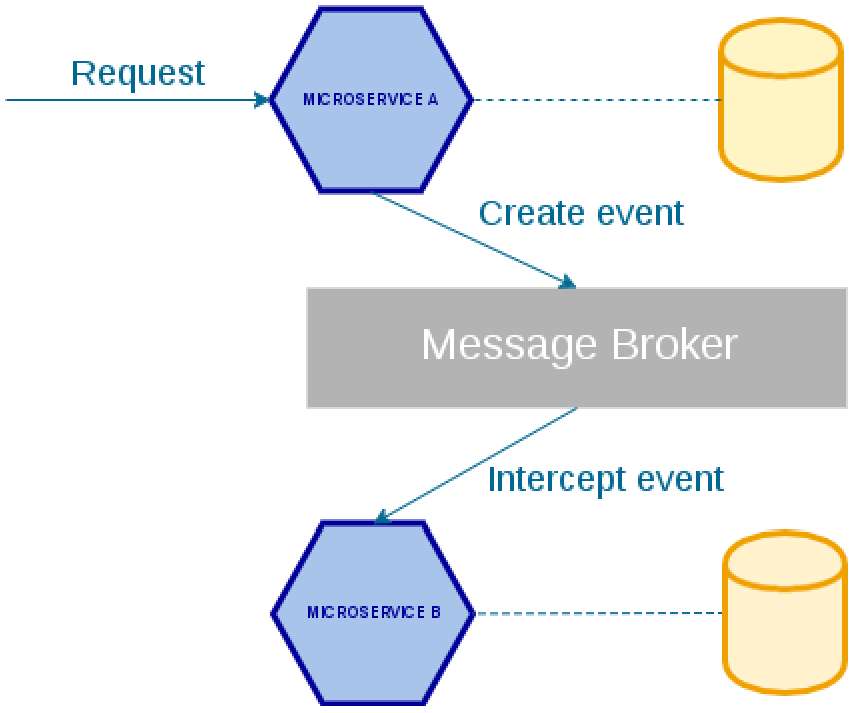

Figure 5: Event-driven data management

3.2 Logical View

As previously discussed at a logical level, the platform will offer different types of services. These services can be conceptually divided into three layers: content layer, mapping layer and user layer.[12]

Content Layer

[11] http://microservices.io/patterns/data/saga.html

[12] as stated in the CPN Grant Agreement (Part B p.13-14)

Co-funded by the Horizon 2020 Framework Programme of the European Union

The Content Layer will focus on the extraction of relevant information from the different content sources or content providers.

This layer is composed of two types of services: (1) content procurement (for structured and unstructured heterogeneous data streams gathering) and (2) knowledge extraction (for user personalisation such as relation identification and clustering)

Mapping Layer

The goal of this layer is to provide services that map content onto targeted users. The services in this layer will make specific content available to the users through personalisation and contextualization.

Further, this layer also includes permission and contract aspects, to address legal and ethical issues, as well as to preserve privacy criteria and accounting of generated revenue to content producers.

User Layer

This layer will offer specific services to deal with the user data itself. It will contain modules to create user profiles (containing preferences, socio-demographic information, history, etc.), and services to appropriately handle user context and guarantee transparency and control over user personal data.

The combination of these three types of services, satisfy all currently defined features and requirements, and these services can also be used by new components to create additional features.

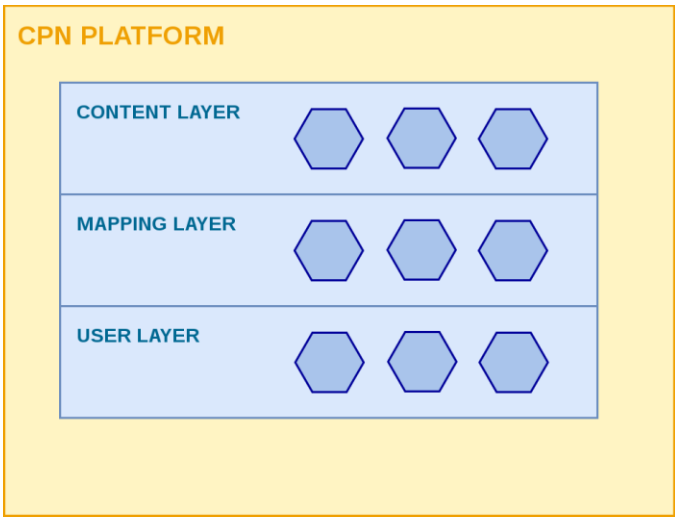

Figure 6:CPN Platform - Three logical layers

3.3 Implementation View

Another important view of the architecture, in which the choices made in the design phase are highlighted, is the implementation view.

The proposed MSA, designed according to state of the art patterns, includes the following components

Co-funded by the Horizon 2020
Framework Programme of the European Union

that are necessary for the realization of the platform: API Gateway, Orchestrator and Message Broker.

API Gateway

The API Gateway component is the entry point for every new request that's being launched by the client applications on our platform.

The API gateway exposes different APIs optimized for each client application. This central component, shared by the whole platform, allows to centralize some middleware functionalities, i.e.:

- ➔ Authentication
- ➔ Logging
- ➔ Caching
- ➔ Security
- ➔ Load Balancing

Orchestrator

Orchestration is the traditional way of handling interactions between different services in a service oriented architecture. With orchestration, there is typically one controller that acts as the "orchestrator" of the overall service interactions.

This component provides a good way for controlling the flow of the application in case of synchronous processing.

In a MSA that uses the API Gateway pattern there are two ways to implement an orchestrator: at the Microservice Layer or at the Gateway Layer.

We chose to implement the orchestrator as a microservice to get the following benefits:

- ➔ unbundling between gateway and orchestration functionalities
- ➔ no violation of the single responsibility principles
- ➔ more flexibility to implement new processes and scaling APIs

Message Broker

A message broker (or queue manager) is a software where queues can be defined, applications may connect to the queue and transfer a message onto it.

Co-funded by the Horizon 2020
Framework Programme of the European Union

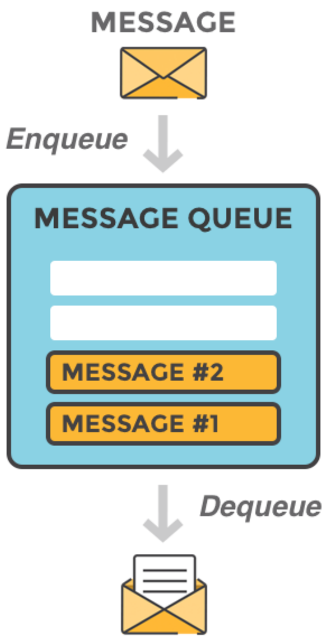

Figure 7: Message Broker Metaphor

A message can include any kind of information. For example, it could include information about a process/task that should start on another application (that could be on another server), or it could be just a simple text message. The queue-manager software stores the messages until a receiving application connects and takes a message off the queue. The receiving application then processes the message in an appropriate manner.

A message broker can act as a middleware for various services (e.g. a web application, as in this example). They can be used to reduce loads and delivery times by web application servers since tasks, which would normally take quite a bit of time to process, can be delegated to a third party whose only job is to perform them.

Message queueing allows web servers to respond to requests quickly instead of being forced to perform resource-heavy procedures on the spot. Message queueing is also good when you want to distribute a message to multiple recipients for consumption or for balancing loads between workers.

Others

Some of the services will make use of a blockchain infrastructure for auditing and accounting functionalities. An additional micro-service exposing CRUD functionalities to create transactions on the given blockchain and related APIs will be provided for vertical access to the others services.

Co-funded by the Horizon 2020
Framework Programme of the European Union

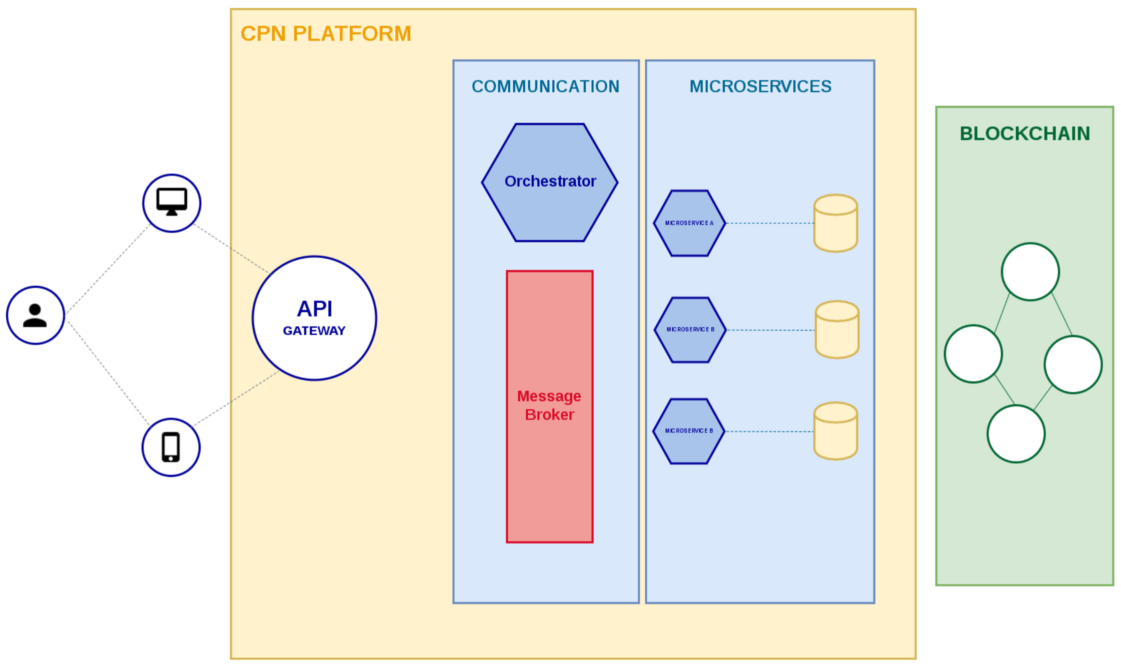

Figure 8: CPN Platform - Implementation view

3.3.1 Technology Bricks Workflow

After the introduction of the components to be realized inside the platform, this section provides an overview on how the modules developed by the partners (technology bricks – WP3) will work together within the platform to offer complex features.

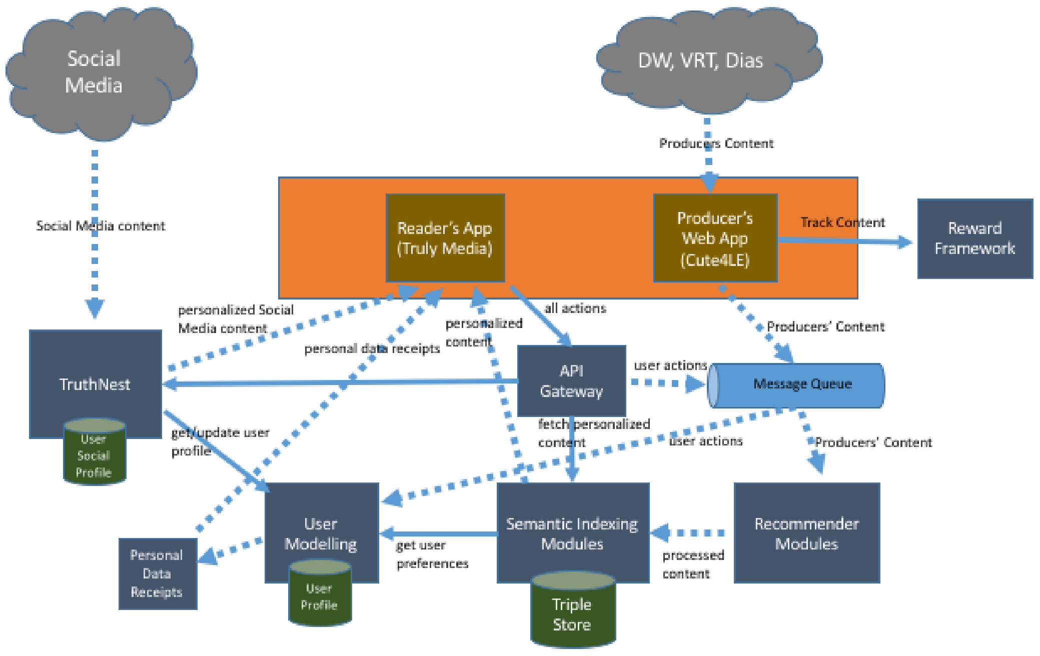

Figure 9: Technology Bricks Workflow

Co-funded by the Horizon 2020
Framework Programme of the European Union

The diagram above describes the core module groups and interactions of these technology bricks within the CPN platform. In the diagram, we are referring to families of modules. Thus, for the sake of simplicity we are not including every individual module that will be developed. This will be covered during the further design of the architecture and the full definition of components.

All calls to the CPN platform will be directed to the API gateway. The API gateway is responsible for the delegation of these calls to the related modules, based on the defined workflow. A message queue will implement a pub/sub protocol to enable modules to fetch and analyse content.

For convenience, the synchronous communication between the modules has been represented directly, although in reality, as described in section 3.4, this kind of communication is implemented through an orchestrator.

One of the goals is to avoid centralized DBs. Therefore, all modules follow a Microservices approach and store data internally in their own repository of choice.

Web App for Producers

Producers will use the Cute4LE module for pushing content to the platform. The content will be published to a message queue. Before pushing content, producers will also access the Reward Framework in order specify the contracts (e.g. cost per access) that apply to a given content and that producers and distributors have to comply. The web application and the recommender module will also interact with the Reward Framework to provide information on how contents are distributed. The information, published on a distributed ledger for transparent accountability, will allow the Reward Framework to rewarded producers according to the selected contract.

Web/Mobile App for Readers

Readers will use a customized version of the Truly Media web application to retrieve personalized content from Social Media and Producers. The web application will communicate through the API gateway with the core modules of the CPN platform. There will also be a mobile version of the client application based on Android.

User Profile

The User Modelling module will listen (through the message queue) to all user actions to modify the user profile accordingly. It can also retrieve input from the Truthnest module to take user activity/preferences in Twitter into account. Finally, the User Modelling module will also collaborate with the Personal Data Receipts (PDRs) module to generate a human-readable receipt explaining how users data are utilized by CPN and what actions are available to the users to modify their personal profiles. New PDRs are issued any time a change in the user data is provided.

Producers' Content Analysis and Retrieval

The analysis modules (Recommender, Semantic Indexing) will retrieve content, process, index, and store this content in a repository in order to fetch the appropriate content when a personalized query is performed. To retrieve the most suited content when such a personalized query is launched, these modules will combine info from the relevant 'User Profile' stored in the User Modelling module with the metadata added to the content items during the processing and indexing step.

Social Media Content Analysis and Retrieval

The Truthnest module will be responsible for detecting trending news on Twitter and other Social Media and (based on the User Profile information) forward personalized content to the user.

Co-funded by the Horizon 2020
Framework Programme of the European Union

3.4 PROCESS VIEW

Another useful view to describe the architecture is a process view, which describes the communication between the various components within the platform.

In this view the actual implementation of the platform is described and the two main types of communication within the platform are highlighted: synchronous communication through REST and asynchronous communication through Messaging.

The MSA design that we have chosen uses (as outlined above) two components that deal with communication within the platform: the orchestrator for synchronous communication and the message broker for asynchronous communication.

The choice of a hybrid solution, in which both types of communication coexist, satisfies all the needs and requirements identified so far.

Let's now analyze the benefits of this choice, describing in detail the two types of communication and how processes communicate with each other within the architecture:

Synchronous Communication - Orchestration

Orchestration is the traditional way of handling interactions between different services in Service-Oriented Architecture (SOA). With orchestration, there is typically one component that acts as the "orchestrator" of the overall service interactions. This typically follows a request/response type pattern.

In the CPN platform the orchestrator will be implemented as a microservice that will be responsible for managing the API Orchestration, decoupling this functionality from the API Gateway.

API orchestration services are a special type of light services that take an API call, split it in into multiple API calls, aggregate the result of each of these calls and return back the aggregation as the result of the original request.

API orchestration allows to simplify the client API, improves performance and security.

There are different ways to implement an orchestrator, e.g. Netflix Conductor[13] use task and workflow definitions to manage the orchestration.

[13] https://medium.com/netflix-techblog/netflix-conductor-a-microservices-orchestrator-2e8d4771bf40

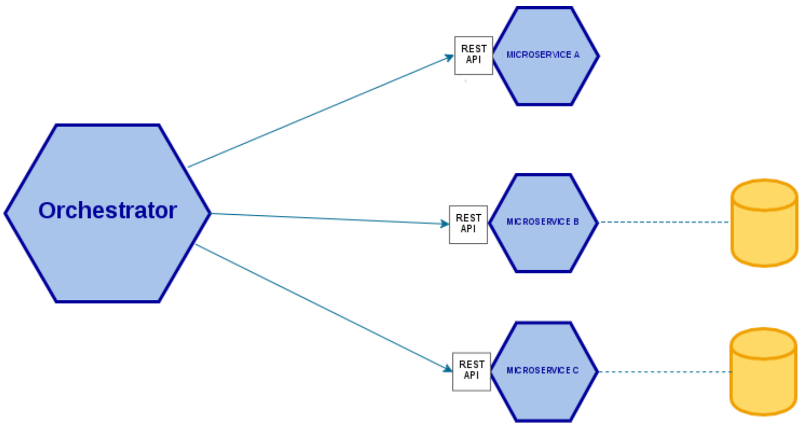

Figure 10: Synchronous Communication - Orchestration

Asynchronous Communication - Messaging

While some of the microservices are conceived to be consumed in synchronous use cases, e.g., deliver data to final consumers, real-time recommendations and so on, some others are essentially thought to be processing components that will operate in background executing tasks such as gathering information, semantically enriching the existing data, building Machine Learning models, analyze trends and finally storing the processed data to specific DBs for subsequent retrieval. These background processing components can be continuously running or can be triggered by some external events; for example, if a user clicks on one of the recommended items this could be a signal of implicit interest about the topics expressed by the news article, so the micro-service that is in charge of User Profiling would add these topics to the user list and the News Recommendation micro-service will produce an updated list of recommendations. This scenario could be implemented in a synchronous/orchestrated fashion by constructing explicit chains of inter-service calls but there are two main problems with this approach:

1. Adding other processing actions in response to the same event is obtained by modifying the orchestrator chain of calls (e.g. "*when the user clicks on an item update his profile AND search on Twitter for trending news related to that topic"*) leading to an increase in service coupling and generally complex and less maintainable orchestrators.

2. The processing time of the components in this chain can be not suitable to synchronous calls whereas some processing components can require minutes, hours or even days.

The messaging pattern is a valid option to solve these kind of problems: a message containing an information related to an event is posted to a message broker (such JMS queues, AMQP, Kafka) that are components specifically thought for high throughput, message persistence and recover from crashes. The message is then delivered to processing components ("consumers") according to variations of the following two mechanisms:

- Message queues: each message is delivered exactly once to the first consumer requesting it. This means each message is PROCESSED at most once.

- Topics: each message is posted to a topic and delivered to those consumers that explicitly subscribed to such topic (pub/sub mechanism)

The first mechanism is mainly used to speed up the processing of messages in parallel while the second one is used when we want to execute different operations associated to the same event.

So, by the use of message brokers and the messaging pattern, we are able to overcome the problems of

Co-funded by the Horizon 2020
Framework Programme of the European Union

synchronous communication in the above described scenario: we avoid to construct explicit chains service calls through the use of pub/sub mechanisms and to let the components to retrieve messages from the queue and process things at their own pace without blocking the other services.

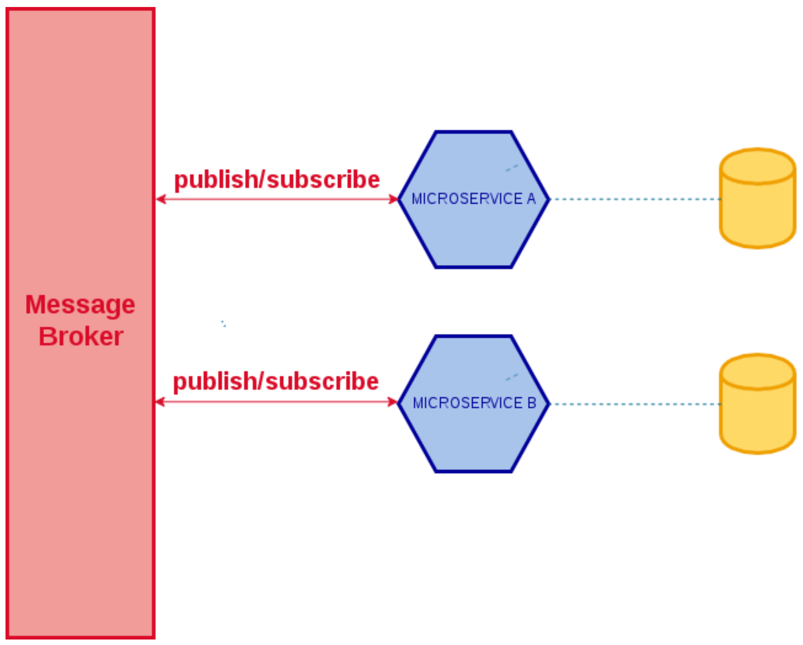

Figure 11: Asynchronous Communication - Message Broker

The following figure shows a possible implementation of a hybrid architecture in which both communication methods coexist and microservices can use one or both:

Co-funded by the Horizon 2020 Framework Programme of the European Union

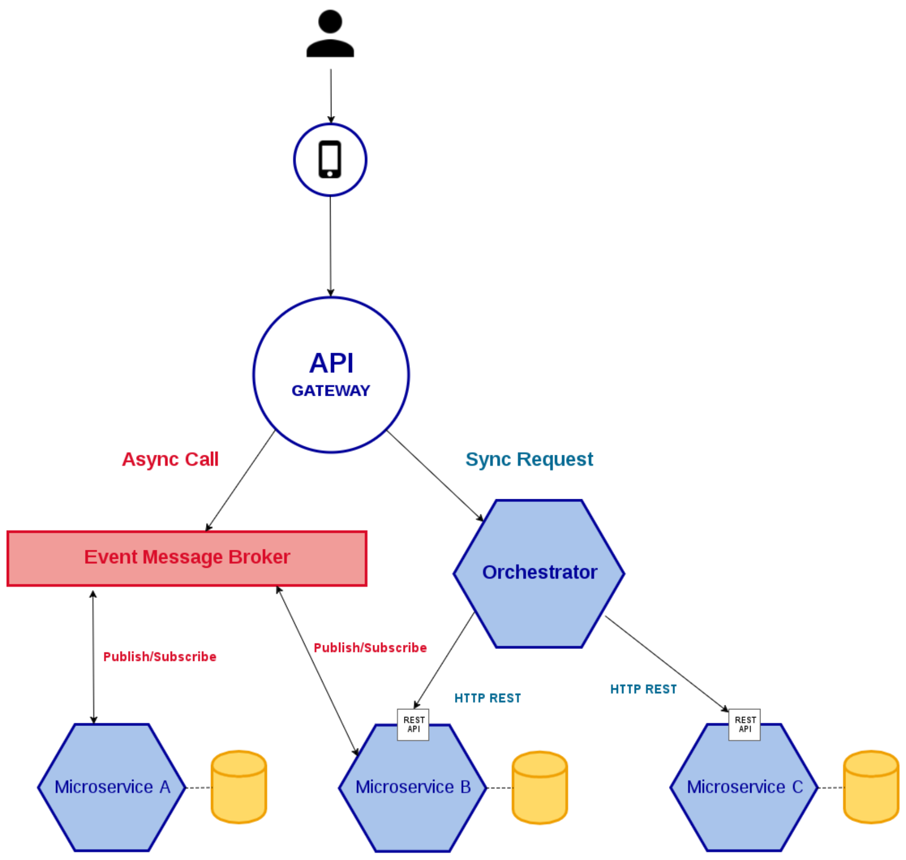

Figure 12: CPN Platform - Communication Process

Co-funded by the Horizon 2020
Framework Programme of the European Union

4 ARCHITECTURE IMPLEMENTATION

In this chapter, we will describe in detail how the different components inside the CPN platform are developed, build and integrated and specify the mechanisms of deployment and continuous integration. We will also define the rules and methods for integrating new components and creating new features.

4.1 COMPONENT REQUIREMENTS AND RECOMMENDATIONS

Below there are some guidelines for designing microservices, which are important for the implementation of the overall CPN platform:

- ➔ Single Responsibility Principle (SRP): Having a limited and a focused business scope for a microservice helps us to meet the agility in development and delivery of services.
- ➔ Make sure the microservices design ensures the agile/independent development and deployment of the service.
- ➔ A given microservice should have a limited set of operations/functionalities and simple message format.

As highlighted in previous chapters, a good choice of architectural design includes that **each** microservice manages its own database.

Furthermore, each microservice must be autonomous and be accessible for communication with other microservices through REST interfaces or through Messaging.

We will provide more detailed information concerning the communication in section 4.3 on interoperability.

4.2 INSTALLATION, DEPLOYMENT AND COORDINATION

In this section we will address the topic of platform installation and show the advantages of using containers for microservices deployment.

Furthermore, a possible tool for container orchestration will be analyzed, providing some useful indications to solve the problems of coordination and automation.

4.2.1 Docker containers

In a microservices architecture, the deployment of the microservices is of critical importance and has the following key requirements:

- ➔ Ability to deploy/undeploy microservices independent from other microservices
- ➔ Ability to scale on microservices level (a given service may get more traffic than other services)
- ➔ Fast and easy building and deployment of microservices
- ➔ Failure in one microservice must not affect any of the other services

Co-funded by the Horizon 2020 Framework Programme of the European Union

Containerization

This term refers to the use of containers. A container is a virtualized server at the operating system level, for which the virtual instance only concerns the user space, i.e. the application execution environment. Therefore, we do not virtualize the processor, storage, network connections, etc. of the physical server, which remain shared between the running containers. This approach means that containers are "lighter" than virtual machines, require fewer resources and can be activated very rapidly and therefore can respond to situations with sudden loads and peaks.

Docker

Docker is a world-leading CaaS (Container-as-a-Service platform). It is fully open source, under license Apache 2.0 and is the most dominant tool in the container ecosystem at the moment, used in production stage by companies like eBay, Uber and PayPal.[14]

The container is a way to package software along with required binaries and settings, isolated on a shared operating system. [15]

Docker makes it easier to create and manage a microservices application than the old SOA paradigm allowed. Once your services have been Dockerized in containers, you can deploy those containers to any server with Docker installed and combine them to compose complex applications. You can move them around between hosts for portability. And you can use container orchestration tools for automatic provisioning.

The key steps involved are the following: [16]

- ➔ Package the microservice as a (Docker) container image
- ➔ Deploy each service instance as a container
- ➔ Scaling is realized by changing the number of container instances
- ➔ Building, deploying, and starting a microservice is much faster using Docker containers than by using virtual machines.

[14] https://www.docker.com/
[15] https://dzone.com/articles/microservices-with-docker
[16] https://dzone.com/articles/microservices-in-practice-1

Co-funded by the Horizon 2020
Framework Programme of the European Union

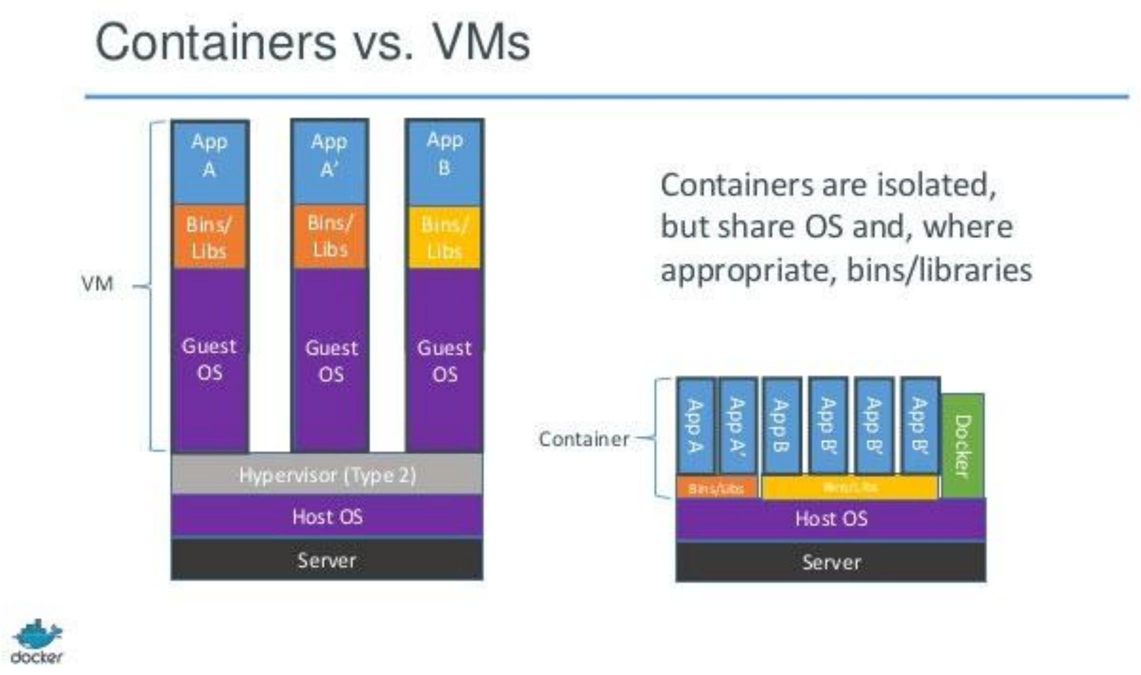

Figure 13: Microservices deployment – VMs vs Containers

The comparative table below shows the benefits of choosing containers over VMs:

Table 2: Comparative table - VMs vs Containers

	Microservice on VMs	**Microservice on Containers**
Scalability	Multiple microservices will run on the same Linux instance. To scale any of the services, we need to scale the whole VM instance, adding unwanted scaling of other services.	Individual applications can be scaled independently, without affecting other services.
Resource utilization	We cannot distribute the services based on resource utilization: CPU utilization, memory utilization, etc.	Container orchestration takes care of better utilization of resources and places the services based on the resources available on the host.
Faster deployment	Spinning up new VMs is slow, as it requires a system boot. If we have to scale a single service, we have to launch new VMs, which need some time to reboot.	Containers, by contrast, are much smaller because they do not require the operating system spin-up time associated with a virtual machine. Containers are more efficient at initialization. Overall, containers start in seconds. That's much faster than VMs.
Cost Optimization	Running the services on VMs requires more hardware resources and thus more costs compared to container systems.	Running microservices on containers helps us to see the true advantage of microservices with lower costs.

Co-funded by the Horizon 2020 Framework Programme of the European Union

Portability	Deploying services on new VMs means first ensuring the required software is installed on the machine.	Dockerized microservices can run in containers. You can deploy those containers to any server with Docker installed.

4.2.2 Container Orchestration

Container Orchestration refers to the automated arrangement, coordination, and management of software containers.

Kubernetes

Kubernetes is a container orchestrator tool, fully open source, under Apache 2.0 license, with a large user and community base. [17]

Kubernetes extends Docker's capabilities by allowing to manage a cluster of Linux containers as a single system, managing and running Docker containers across multiple hosts, offering co-location of containers, service discovery, and replication control. As you can see, most of these features are essential in our microservices context too. Hence using Kubernetes (on top of Docker) for microservices deployment has become an extremely powerful approach, especially for large scale microservices deployments.

Kubernetes manages secrets for containers, performs load balancing between instances in a group, scales up/down instances as directed, and provides rolling updates to services so that the service does not go down while being updated.

There are three key concepts in Kubernetes:[18]

- ➔ **Pods**: are the smallest deployable units that can be created, scheduled, and managed. It's a logical collection of containers that belong to an application.
- ➔ **Master**: is the central control point that provides a unified view of the cluster. There is a single master node that controls multiple minions.
- ➔ **Minion**: is a worker node that runs tasks as delegated by the master. Minions can run one or more pods. It provides an application-specific "virtual host" in a containerized environment.

[17] https://kubernetes.io/case-studies/
[18] http://blog.arungupta.me/key-concepts-kubernetes/

Co-funded by the Horizon 2020
Framework Programme of the European Union

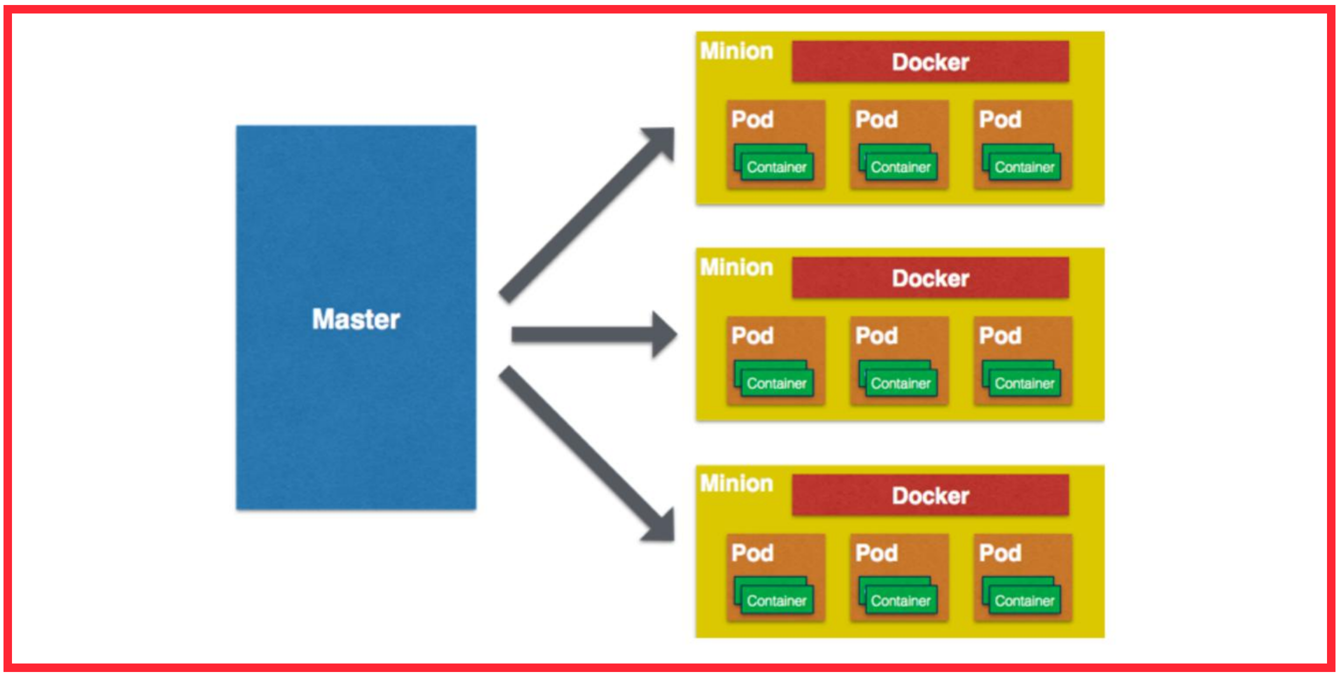

Figure 14: Kubernetes key components

4.3 Integration with External Components

As mentioned, one of the goals of the CPN platform is to allow the integration of new components and the creation of new features by external stakeholders.

To facilitate this, it is important to provide a good view on the functionality and APIs offered by the individual services of the platform, but also on the functionalities offered by the platforms as a whole, aggregating multiple services.

It is critical to have clear documentation for each service. Ideally, documentation for all services should be shared within the platform.

Both the REST APIs exposed by the microservices and the events that can be subscribed to through the message broker, should be documented.

If a service changes its endpoints, this must be tracked and made available by the platform.

Open API Documentation

APIs have become the most common way for various services to communicate with each other. In order to better expose services via APIs, a common interface for these APIs needs to be present to tell exactly what each service is supposed to do. This interface is a contract that defines the SLA between the client and the services. The OpenAPI (Swagger[19]) Specification has emerged as the standard format for defining this contract, in a way that's both human and machine readable, making it easier for services to effectively communicate with and orchestrate the entire application.

All the information about the OpenAPI Specification can be found on:

https://github.com/OAI/OpenAPI-Specification

[19] https://swagger.io/swagger-ui/

Co-funded by the Horizon 2020
Framework Programme of the European Union

Message Broker Event Documentation

Message exchanged through a message broker are usually constrained with respect to the size in bytes (e.g. around 15Mb is the limit for Kafka messages) in order not to affect the overall performance of the messaging system. However, no assumption is made on the structure of body of the message itself that could be a sequence of bytes, a string, a number, etc. A common pattern is to send the text of the message using a structured and easily-parsable format such as XML or JSON. Each consumer can therefore receive the message, extract and parse the content of the body and access/process the information of interest.

In CPN we define the very basic structure of the message to be exchanged through message brokers as the following:

Table 3: Message basic structure

Field	Description
event_type	A name describing the event that is happened. E.g. “USER_CLICK”, “NEW_TOPIC”, “TWITTER_FEED_UPDATE”,...
date	A timestamp for the event
origin	To distinguish if the message has been originated directly from the final user or from another microservice: {“USER”, “MICROSERVICE”}

This structure will be represented using JSON notation and it is assumed to be flexible, i.e., while all messages share at least the set of the above described attributes, most of them will have additional ones depending on the “event_type” value.

Example:

We describe again the scenario of page …: a user clicks on an article, the click is captured, the topics of the article are extracted and added to the list of interests of the user in his user profile and, due to these additional interests, new recommendations are computed for the user.

In terms of messages exchange this process could look as this:

First a message like this is posted on the broker

```
{
        “event_type”: “USER_CLICKS”,
        “User_ID”: 12345
        “Item_ID”: 6789
        “date”:….,
        “origin”: “USER”
}
```

the User-Profiler service, who had previously subscribed to this kind of event, receive this message, extract the “Item_ID”, retrieves the item, extract the topics and add them to the list of topics of the user identified by “User_ID”. After it has finished it posts a new message on the broker:

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
{
        "event_type": "USER_PROFILE_UPDATED",
        "User_ID": 12345
        "date":….,
        "origin": "MICROSERVICE"
}
```

this new message will be received by the Recommender service that. knowing that something has changed on the profile of a user, will start to compute the recommendations according to the new available information.

Co-funded by the Horizon 2020
Framework Programme of the European Union

5 CONCLUSIONS

The reference architecture document illustrates the goals and the scope of the CPN platform, i.e., to offer a series of innovative services for the personalization of content to end-users and professional users.

To reach these goals, it is necessary to implement a virtual, open and flexible platform. Various architectural implementations were analyzed to allow the integration of existing modules but also the possibility of creating new services.

The microservice architecture was chosen because it offers a series of benefits well suited for the CPN platform requirements.

This document describes the benefits and drawbacks of the chosen architecture, the design phase and some views representing a possible implementation of the architecture itself, demonstrating that, following some microservices paradigms, it is possible to create a platform that satisfies all the needs.

Starting from already existing modules developed by the partners, microservices will be identified following specific guidelines, those will consitute the core of the platform.

Furthermore, some basic rules have been defined for the interoperability and future integration of new components.

Co-funded by the Horizon 2020
Framework Programme of the European Union

References

[1] https://www.ibm.com/developerworks/websphere/library/techarticles/1601_clark-trs/1601_clark.html

[2] http://microservices.io/index.html

[3] http://microservices.io/patterns/apigateway.html

[4] http://microservices.io/patterns/communication-style/messaging.html

[5] http://kafka.apache.org/

[6] http://microservices.io/patterns/data/database-per-service.html

[7] https://medium.com/netflix-techblog/netflix-conductor-a-microservices-orchestrator-2e8d4771bf40

[8] https://www.docker.com/

[9] https://dzone.com/articles/microservices-with-docker

[10] https://dzone.com/articles/microservices-in-practice-1

[11] http://microservices.io/patterns/data/saga.html

[12] https://kubernetes.io/case-studies/

[13] http://blog.arungupta.me/key-concepts-kubernetes/

[14] https://swagger.io/swagger-ui/

Co-funded by the Horizon 2020
Framework Programme of the European Union