



Grant Agreement No.: 761488



D2.2: CPN Open Virtual Platform v1

CPN Platform v1 - Accompanying Report

Work package	WP 2
Task	T2.3 – T2.4
Due date	30/06/2018
Submission date	29/06/2018
Deliverable lead	ENG
Version	1.0
Authors	Ferdinando Bosco (ENG), Vincenzo Croce (ENG)
Reviewers	Fulvio D'Antonio (LiveTech)
Keywords	CPN – Open Virtual Platform – Microservices – Deployment- Integration

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	31/05/2018	1 st version of the deliverable with table of contents	Ferdinando Bosco (ENG) Vincenzo Croce (ENG)
V0.2	21/06/2018	draft version of deliverable for contributions and feedback by partners	Ferdinando Bosco (ENG) Vincenzo Croce (ENG)
V0.3	27/06/2018	Version for internal review. It includes feedback and contributions from partners.	Ferdinando Bosco (ENG) Vincenzo Croce (ENG) Fulvio D'Antonio (LiveTech) Chris Develder (Imec)
V1.0	29/06/2018	Final version	Ferdinando Bosco (ENG) Vincenzo Croce (ENG) Fulvio D'Antonio (LiveTech) Chris Develder (Imec)



DISCLAIMER

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

This document reflects only the authors' views and the Commission is not responsible for any use that may be made of the information it contains.

Project co-funded by the European Commission in the H2020 Programme		
Nature of the deliverable:		DEM + R
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	



EXECUTIVE SUMMARY

The CPN project foresees three major releases of the CPN Open Virtual platform that include functionalities aligned with the requirements gathered for the three CPN pilots. Each platform release includes specific functionalities, chosen after a process of evaluation and prioritization of the user requirements.

Starting from the reference architecture document, delivered as D2.1, the first version of the open virtual platform was implemented and is reported in this document. This version of the platform consists of a series of modules named Technology Bricks and provided by CPN partners in order to satisfy the user requirements expected for the first pilot iteration.

This document is the CPN Open Virtual Platform accompanying document, all the process of installation, configuration and commissioning of the platform and related technology bricks are described, over the approach and methodology followed to implement the features necessary to satisfy the user requirements.



TABLE OF CONTENTS

1	INTRODUCTION.....	9
2	PLATFORM IMPLEMENTATION	10
2.1	Installation.....	11
2.2	Configuration.....	12
2.2.1	Docker private registry.....	12
2.2.2	CPN Catalog.....	13
2.3	Core components	15
2.3.1	API Gateway	15
2.3.2	Orchestrator	16
2.3.3	Message Broker	16
3	CPN TECHNOLOGY BRICKS	18
3.1	Deployment	19
3.2	Integration.....	20
3.2.1	Data schema.....	20
3.2.2	Documentation.....	22
3.2.3	Integration with core components.....	23
4	TEST AND COMMISSIONING	24
4.1	Unit test.....	24
4.2	Integration test.....	24
4.2.1	CONTRACT TEST	25
4.3	End-to-end test.....	25
4.4	Other tests	25
4.4.1	Scale testing.....	25
4.4.2	Resilience testing.....	26
5	CPN PLATFORM - 1ST PROTOTYPE DELIVERY	27
5.1	Approach and Methodology.....	27
5.1.1	Requirements prioritization	27
5.1.2	Mapping Requirements/Technology Bricks	28
5.1.3	Sprints	30
5.2	Results.....	30
6	CONCLUSIONS	32



LIST OF FIGURES

FIGURE 1: CPN STACKS IN RANCHER ENVIRONMENT	11
FIGURE 2: RANCHER HOST INFORMATION AND STATISTICS	12
FIGURE 3: RANCHER DEFAULT APPLICATION CATALOG.....	14
FIGURE 4: CPN APPLICATION CATALOG	14
FIGURE 5: CPN API GATEWAY GUI.....	16
FIGURE 6: EXAMPLE DOCKERFILE FOR NODE.JS APPLICATION.....	19
FIGURE 7: OUTPUT DATA SCHEMA OF A CPN TECHNOLOGY BRICK	21
FIGURE 8: OUTPUT EXAMPLE OF A CPN TECHNOLOGY BRICK.....	22
FIGURE 9: 1ST PROTOTYPE BASIC WORKFLOW	29
FIGURE 10: DEPLOYMENT OF CPN PLATFORM V1.....	31



LIST OF TABLES

TABLE 1: LIST OF CPN TECHNOLOGY BRICKS.....18

ABBREVIATIONS

API	Application Programming Interface
ATC	Athens Technology Center
CPN	Content Personalisation Network
DigiCat	Digital Catapult
e.g.	Example given
ENG	Engineering Ingegneria Informatica
etc.	Etcetera
GB	Giga Byte
GUI	Graphic User Interface
Imec	Interuniversity MicroElectronics Center
JSON	JavaScript Object Notation
JWT	JSON Web Token
RAM	Random Access Memory
REST	Representational State Transfer
TLS	Transport Layer Security
UI	User Interface
URL	Uniform Resource Locator



1 INTRODUCTION

The main goal of CPN project is to realize a virtual platform in order to make available -to media companies - innovative services and modules that combine content and personal data to offer content personalization features.

The development of this platform started with the reference architecture document, released as deliverable D2.1¹ on February 28. This document contains all the guidelines and recommendation to implement a powerful platform that satisfies all the needs of CPN project.

In this document, we report all the process of implementation and deployment of the first version of the platform and the integration of technology bricks, as reported in deliverable D3.1². This version of the platform is the first expected of a cycle of three iterations and is offering a series of starting features in order to test the platform and the related module in a pilot environment.

¹ D2.1 CPN Reference Architecture v1.0.pdf - <https://owncloud.cpn.lab.vrt.be/index.php/f/1741>

² D3.1 Initial Design & APIs of Technology Bricks _V1.0.pdf - <https://owncloud.cpn.lab.vrt.be/index.php/f/2548>



2 PLATFORM IMPLEMENTATION

Following the reference architecture document, we implemented the CPN platform as a microservices architecture based on a Container Management Platform and in particular Rancher v1.6³.

Rancher is an open source software platform that enables organizations to run and manage Docker container in production environment.

Rancher takes in raw computing resources from any public or private cloud in the form of Linux hosts. Each Linux host can be a virtual machine or physical machine. Rancher does not expect more from each host than CPU, memory, local disk storage, and network connectivity

Rancher implements a portable layer of infrastructure services designed specifically to power containerized applications. Rancher infrastructure services include networking, storage, load balancer, DNS, and security. Rancher infrastructure services are typically deployed as containers themselves, so that the same Rancher infrastructure service can run on any Linux hosts from any cloud.

Rancher includes a distribution of all popular container orchestration and scheduling frameworks today, including Docker Swarm, Kubernetes, and Mesos and, in addition to these, it supports its own container orchestration and scheduling framework called Cattle⁴.

Cattle is the orchestration framework chosen for the deployment of the first version of CPN platform. This framework organizes the modules into stack (a group of services) and services.

There are two categories of stack:

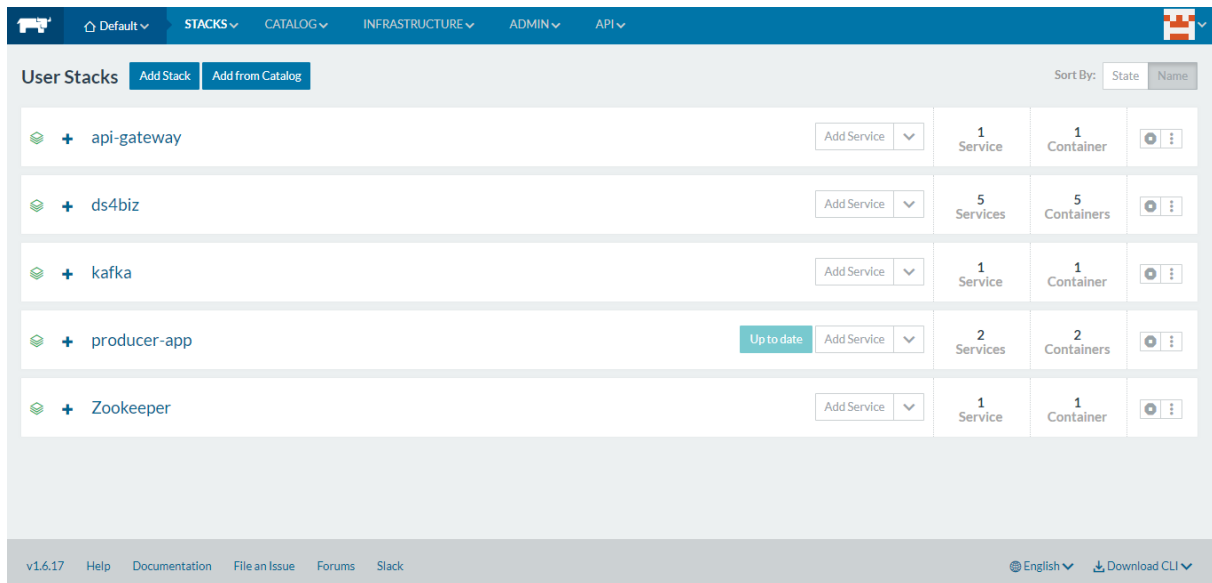
- ➔ User stacks: include all the stacks deployed by the user
- ➔ Infrastructure stacks: include all the default stacks offered by Cattle framework

The figure below represents an example list of user stacks deployed on CPN host.

³ <https://rancher.com/docs/rancher/v1.6/en/>

⁴ <https://rancher.com/docs/rancher/v1.6/en/cattle/stacks/>





Stack	Add Service	Status	Services	Containers	Actions
api-gateway	Add Service		1 Service	1 Container	⚙️ ⋮
ds4biz	Add Service		5 Services	5 Containers	⚙️ ⋮
kafka	Add Service		1 Service	1 Container	⚙️ ⋮
producer-app	Up to date Add Service		2 Services	2 Containers	⚙️ ⋮
Zookeeper	Add Service		1 Service	1 Container	⚙️ ⋮

Figure 1: CPN stacks in Rancher environment

2.1 INSTALLATION

In the following paragraph are described all the installation steps of the container management platform.

The first step is to provide a Linux host with kernel version 3.10+ and at least 1GB of RAM.

The second step is to install a supported Docker version⁵. To install Docker follow the official guidelines.⁶

In our configuration, we provide a virtual host with CentoOS 7 (3.10.0), 8GB of RAM, 4x2.5ghz processors, 20GB of disk and Docker 18.0.

The Rancher platform v1.6 is runnable directly via Docker container:

```
$ sudo docker run -d --restart=unless-stopped -p 8080:8080 rancher/server:stable
.... Startup Succeeded, Listening on port...
```

The default port for Rancher UI is 8080, so it is available on `http://<SERVER_IP>:8080`

Now the container management platform is available and we can proceed to the host configuration.

⁵ <https://rancher.com/docs/rancher/v1.6/en/hosts/#supported-docker-versions>

⁶ <https://docs.docker.com/install/linux/docker-ee/ubuntu/>

2.2 CONFIGURATION

After Rancher installation, it was possible to create a Rancher instance directly via web UI.⁷

The first step was to create an administration user with the permission to create a new environment. As already discussed for this release of the platform Cattle was used as orchestration engine on the default environment.

The second step was to associate a physical or virtual host to new Rancher environment. In CPN platform the host used is that described in the installation paragraph.

Once associated the host, Rancher start to monitoring this host and all the information was provided through the UI as shown in the figure below:

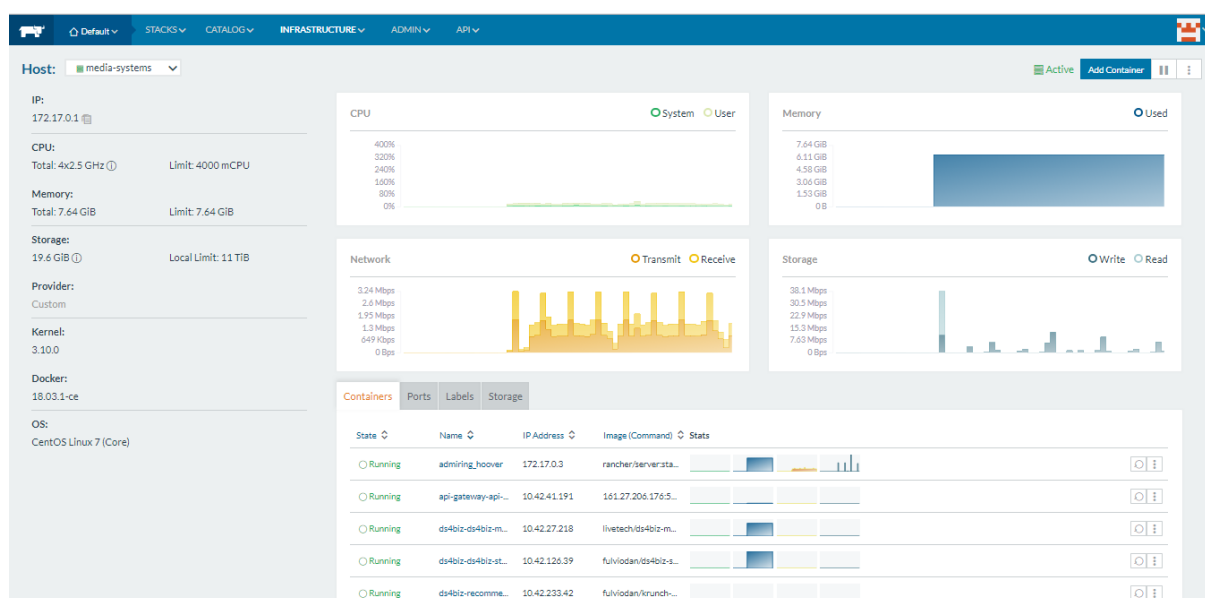


Figure 2: Rancher host information and statistics

Now the Rancher environment is ready to deploy CPN modules, divided into stacks, as Docker container, including both core components and technology bricks.

2.2.1 Docker private registry

Docker makes available to all the community a public repository, named DockerHub⁸, with 100.000+ apps, official and unofficial.

⁷ <https://rancher.com/docs/rancher/v1.6/en/quick-start-guide/>

⁸ <https://hub.docker.com/explore/>

From DockerHub, we can retrieve the basic images for our platform (e.g. MongoDB or Node.js official images) but for the modules developed by the partners, we need a private Docker registry.

Rancher allow integrating a private registry and using it as repository for platform containers, accessible only for users of CPN. In this version of the platform the registry was deployed as Docker container in the same host and is available at address: `http://xxx.yyy.32.194:5000`

In order to make available a module within the CPN platform you need to push your Docker image into this private registry.

```
$ docker push xxx.yyy.32.194:5000/myapp:version
```

It is mandatory to tag the image with private registry URL before to push it.

The registry is now not under TLS and therefore needs to be added as an insecure registry in the Docker configuration. In order to make this, edit the `daemon.json` file, whose default location is `/etc/docker/daemon.json` on Linux or `C:\ProgramData\docker\config\daemon.json` on Windows with the following content:

```
{ "insecure-registries" : [ "xxx.yyy.32.194:5000" ] }
```

You can verify if the registry is setted as insecure with:

```
$ docker info
```

```
Insecure Registries:
xxx.yyy.32.194:5000
127.0.0.0/8
```

Following version of the platform should provide a dedicated host for the private registry under TLS connections.

2.2.2 CPN Catalog

The Rancher platform makes available a series of applications through its catalog. In the figure below an example of Rancher's catalog.



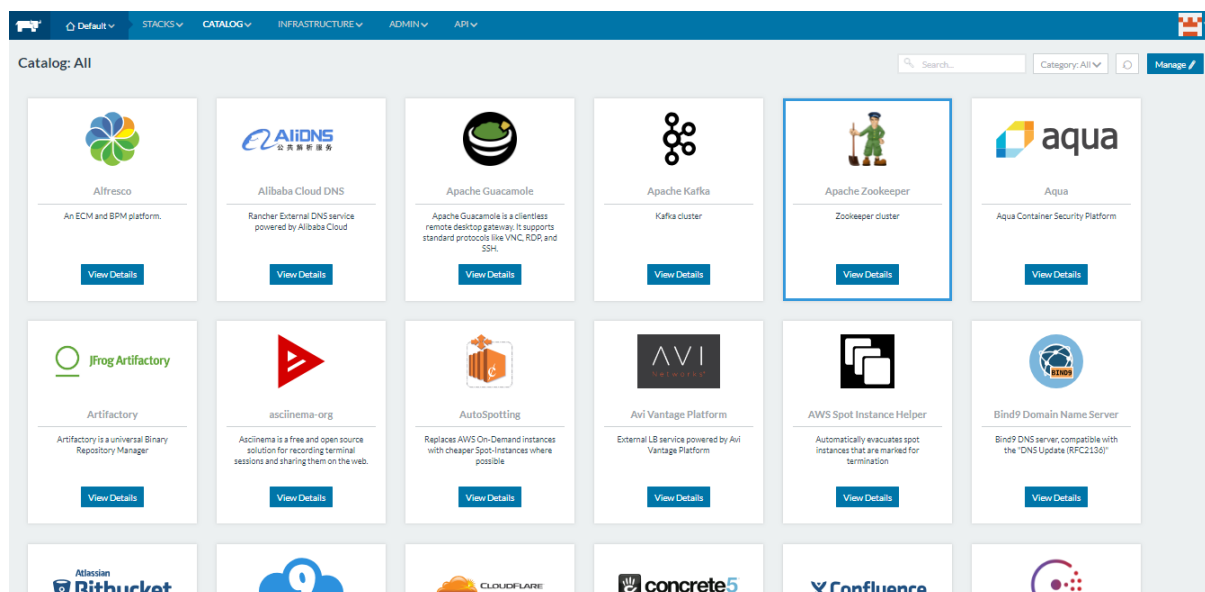


Figure 3: Rancher default application catalog

An additional feature of Rancher is the possibility to create your own application catalog. This catalog is a sort of marketplace for your modules, provided with description, technical documentation and usage guidelines, from which the user can choose the modules to be installed in the platform.

In the figure below the CPN catalog:

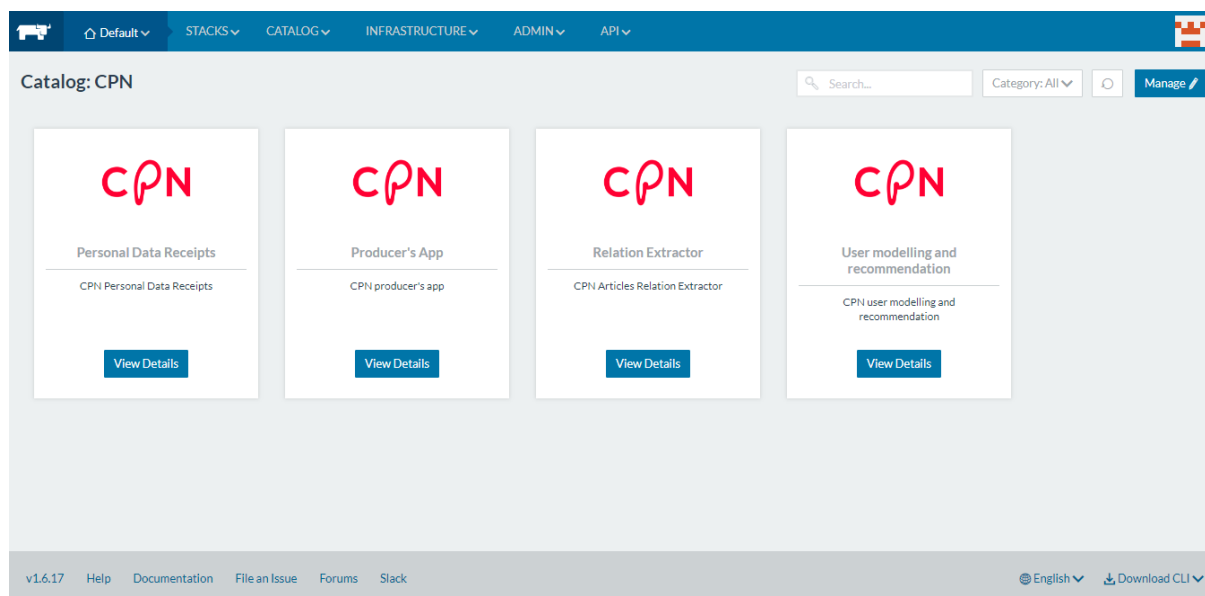


Figure 4: CPN application catalog

2.3 CORE COMPONENTS

Even if the CPN Platform is open and extensible and the modules installed may be replaced or improved according to customer needs, some core components are necessary to have a basic configuration and deployment.

As described into the Reference Architecture, the CPN microservices architecture implies the application of API Gateway pattern and two kind of communication systems: messaging and orchestration. In order to satisfy these architectural requirements three core components have been deployed within the platform:

- ➔ API Gateway
- ➔ Orchestrator
- ➔ Message Broker

In the following paragraphs these components are described in detail.

2.3.1 API Gateway

The API Gateway represents the access door for external application that want to exploit CPN innovative services.

This component serve all different client applications (mobile, web, etc.) and centralize some middleware functionalities as authentication, logging, security, etc.

In CPN platform a microservices API gateway, based on Express.js framework has been deployed, namely the Express Gateway module.

Express Gateway is open source and distributed under Apache 2.0 License⁹ and it was chosen for its ease of use and flexibility and in particular for the following benefits:

- ➔ Configurable via YAML
- ➔ Language agnostic
- ➔ Extensible with any Express.js middleware
- ➔ Run anywhere with Docker
- ➔ Many authentication and authorization method supported (e.g. JWT)

The CPN API Gateway was extended with a GUI that permit to view all documented APIs and test them directly via browser.

⁹ <https://github.com/moby/moby/blob/master/LICENSE>

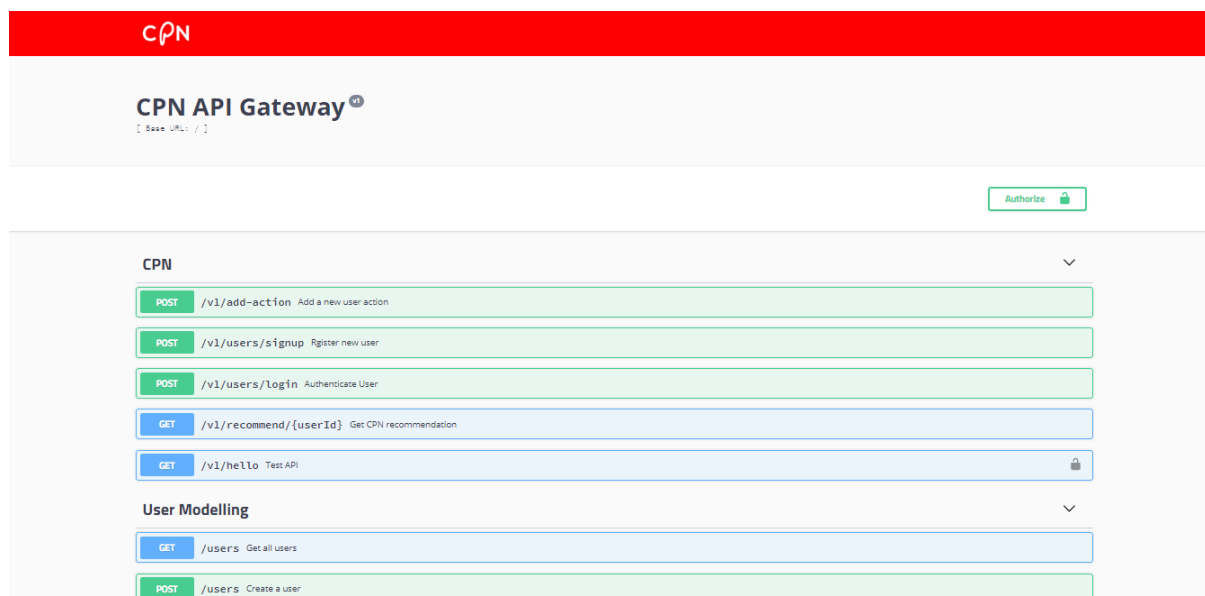


Figure 5: CPN API Gateway GUI

2.3.2 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.

We developed a customized orchestrator module and deployed it as Docker container, 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

The orchestration module cover all the synchronous processes inside the platform, exploiting the APIs exposed by technology bricks.

2.3.3 Message Broker

The messaging pattern allow the microservices to collaborate minimizing their coupling and improving the flexibility of the platform.

To implement this pattern a message broker is needed. 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.

In CPN platform, Apache Kafka and Zookeeper have been chosen to allow messaging and asynchronous communication among microservices.

Apache Kafka and Zookeeper are both open sources and distributed under Apache 2.0 License.

Apache Kafka is a distributed streaming platform that allow publishing and subscribing streams of records, very similar to a standard message queue. Kafka is deployable as a cluster of one or more servers that stores stream records in categories called topics. Each record consists of a key, a value and a timestamp.

Zookeeper is a software tool for distributed services coordination. In particular, it is a service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

Both the modules were deployed into the CPN platform as Docker containers and are available to all the other modules.

Details on integration with the CPN core components will be described in Section 3.2.



3 CPN TECHNOLOGY BRICKS

The CPN technology bricks are a series of modules developed by CPN partners to offer a series of innovative services for news recommendation.

As defined into D3.1: Initial Design and APIs of Technology Bricks¹⁰ they are divided into three layers as following:

Layer	Technology Bricks	Partner responsible of the provision
Content	Semantic Lifting	Imec
	Relation Extraction	Imec
	Topic Extractor	LiveTech
	Uplifting/Depressing Article Classifier	Imec
	Frame Based Slot-Filling System	Imec
	Sentiment	LiveTech
User	UserModelling	LiveTech
	Reader's App	ATC
	Personal Data Receipts	DigiCat
Mapping	Producer's App	ENG
	Reward Framework	DigiCat
	Twitter Analytics	ATC
	Recommender	LiveTech

Table 1: List of CPN Technology Bricks

In order to be included into the CPN platform, all these modules need to be implemented as microservices and containerized with Docker.

Furthermore, as already highlighted in Reference Architecture, each microservice have to manage its own database, must be autonomous and must be accessible for communication with other microservices through REST interfaces or through Messaging.

In the following paragraphs are described the guidelines for container deploying and integration with the core components of CPN platform.

¹⁰ <https://owncloud.cpn.lab.vrt.be/index.php/f/2548>



3.1 DEPLOYMENT

As already discussed in the previous paragraphs, to be deployed into the CPN platform a module must be provided as Docker container and the Docker version required is almost v1.12.

To start the containerization a complete guide is provided on Docker documentation¹¹.

In order to create a Docker container we need a Dockerfile:

```
FROM node:carbon

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm install --only=production

# Bundle app source
COPY . .

EXPOSE 8080

CMD [ "npm", "start" ]
```

Figure 6: Example Dockerfile for Node.js application

After the creation of a Dockerfile, to deploy a module into the CPN platform, three steps are needed:

¹¹ <https://docs.docker.com/get-started/part2/>



- ➔ Build an image of the container

```
$ docker build -t myapp .
```

- ➔ Tag the image with private registry URL

```
$ docker tag myapp xxx.yyy.32.194:5000/myapp:latest
```

- ➔ Push the image into the CPN private registry

```
$ docker push xxx.yyy.32.194:5000/myapp:latest
```

After these steps, it will be possible to deploy the module into the CPN platform directly through the Rancher web UI.

If a module is stand-alone and it does not need to communicate with other modules, a stack can be created in a simple way starting from the image pushed on the platform Docker images registry.

In case of modules that need to be integrated with others, integration guidelines must be followed.

3.2 INTEGRATION

3.2.1 Data schema

To better understand how the modules interact with each other for each Technology Bricks was released a document with input/output data schema and shared with other partners.

Each document includes the format for input/output data as needed and a data example.

The following figures represent the data schema document of a CPN technology Brick:



```
articleSchema:
{
  _id : String, // Internal Id
  originId : String, // Original Id from source
  origin : String, // Source
  url : String //original item url
  category : String, // Category of article
  multimedia : [{ // List of media objects (images, video, etc..)
    type: String,
    url: String,
    name: String,
  }]
  title : String,
  text : String,
  language : String,
  author : String,
  date : Date, // date in Date format
  dateStr : String, // date in String format
  timestamp : Number, // date in timestamp format
  location : { //Location in geojson format
    type: Object,
    index: '2dsphere',
    sparse: true },
  tags : [String] //List of tags
}
```

Figure 7: Output data schema of a CPN Technology Brick



Example :

```
{
  "url": "https://api.dw.com/api/detail/article/43703889?apiVersion=1.3.0",
  "date": "2018-05-08T15:41:57.140Z",
  "text": "Belgian politicians and animal rights activists have slammed the gassing of 20,000 chicks at Brussels airport. The animals were meant to be flown to Kinshasa. Airport firefighters refused to take part in the killing.",
  "title": "Belgium: Gassing of baby chicks at Brussels airport sparks outrage ",
  "originId": "43703889",
  "origin": "DW",
  "_id": "5af1c61d3ecbb81a00bf0e8b",
  "tags": [],
  "multimedia": [
    {
      "name": "Frisch geschlüpfte Hühner-Küken",
      "url": "https://api.dw.com/image/18380179_301.jpg"
    }
  ]
}
```

Figure 8: Output example of a CPN Technology Brick

3.2.2 Documentation

For a complete integration of a module into the CPN platform, it is crucial to have clear documentation for each service that the module implements, whether it is exposed via API or communicating through message broker.

In the case of APIs, a standard format for the documentation is Open API¹² and in particular, a YAML file with Open API specification Swagger v2.0 is needed.

Since we will have multiple API in CPN platform to better identify the relevant module and to not create overlapping among them, a module must follow this naming convention:

`/[module-identifier]/[service-name]/`

All the APIs documentation is included in the API Gateway swagger.yaml to have a complete description of all platform modules and in order to show the documentation in the API Gateway GUI.

In the case of module that communicate via message broker, a common pattern is to send the text of the message using a structured and easily parseable format, preferably JSON. Each

¹² <https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md>

consumer can therefore receive the message, extract and parse the content of the body and access/process the information of interest.

3.2.3 Integration with core components

Thanks to sharing of data schema information and interfaces documentation all the Technology Bricks can collaborate among them.

The core components deployed within the CPN platform allow this collaboration adding an inter-communication level.

To interface a module to external application through API Gateway it is enough expose the documented APIs and define a new routing in the gateway configuration file.

In case of integration through the orchestrator, a new process definition is needed.

Finally, in case of integration through the message broker, the module itself must implement the rule of integration and setup as environment variable the local address of Kafka.



4 TEST AND COMMISSIONING

A microservices architecture brings many benefits such as the ability to independently deploy, scale and maintain each component and parallelize development. However, it needs new testing strategies to be defined, in a different way respect of monolithic approach.¹³

A microservices architecture builds software as suites of collaborating services, so it is important testing the architecture starting from the single service (Unit test), proceeding with their collaboration and integration (Integration Test) and ending with testing the entire functionalities (End-to-end test)

4.1 UNIT TEST

A unit test exercises the smallest piece of testable software in the application to determine whether it behaves as expected.¹⁴

For any test to be effective, you first have to find the boundaries of that test. The goal of the test is to verify all behaviour inside the “black box” of the test boundaries by manipulating the inputs to the black box, and verifying that the black box for each set of inputs produces the correct output.

In order to reproduce input conditions that come from side effects it is necessary to implement function stubs, also called mocks. Many test frameworks and tools help to implement the function stubs¹⁵

It is worth noting that unit testing alone does not provide guarantees about the behaviour of the system. We need other types of testing for microservices as integration and contract tests.

4.2 INTEGRATION TEST

An integration test verifies the communication paths and interactions between components to detect interface defects.¹⁶

Verification of the services that have been individually tested must be performed. This critical part of microservice architecture testing relies on the proper functioning of inter-service communications. Service calls must be made with integration to external services, including error and success cases. Integration testing thus validates that the system is working together seamlessly and that the dependencies between the services are present as expected.

¹³ <https://martinfowler.com/articles/microservice-testing/>

¹⁴ <https://martinfowler.com/bliki/UnitTest.html>

¹⁵ <https://medium.com/@nathankpeck/microservice-testing-unit-tests-d795194fe14e>

¹⁶ <https://www.techopedia.com/definition/7751/integration-testing>



4.2.1 CONTRACT TEST

An integration contract test is a test at the boundary of an external service verifying that it meets the contract expected by a consuming service.¹⁷

Contract testing should treat each service as a black box and all the services must be called independently and their responses must be verified. Any dependencies of the service must be stubs that allow the service to function but do not interact with any other services. This helps avoid any complicated behaviour that may be caused by external calls and turn the focus on performing the tests on a single service. A “contract” is how a service call (where a specific result or output is expected for certain inputs) is referred to by the consumer-contract testing. Every consumer must receive the same results from a service over time, even if the service changes. There should be the flexibility to add more functionality as required to the Responses later on. However, these additions must not break the service functionality. If the service is designed in this manner, it will stay robust over longer durations and the consumers will not be required to modify their code to take into account the changes made later on.

4.3 END-TO-END TEST

An end-to-end test verifies that a system meets external requirements and achieves its goals, testing the entire system, from end to end.¹⁸

End-to-end testing verifies that the entire process flows work correctly, including all service and DB integration. Thorough testing of operations that affect multiple services ensures that the system works together as a whole and satisfies all requirements. Frameworks like JBehave¹⁹ help automate functional testing by taking user stories and verifying that the system behaves as expected.

4.4 OTHER TESTS

Besides the aforementioned tests, that could be functional tests (i.e. testing the functional aspects of the systems) another important class of tests is the one addressing the problems connected to the global behaviour of the systems in usage contexts for evaluating non-functional properties such as the resilience of the system, load testing and response time. Such tests can be broadly classified as being “non-functional” tests and, in microservices architecture, are often run both in test/preproduction environments and continuously executed and monitored also in production environment (the so-called “shift-right testing” depicting a situation in which the “production” environment is represented as the last environment on the right in an architecture diagram).

4.4.1 Scale testing

In a microservices application, there may also be supporting services or resources that operate faster or slower, depending upon total application traffic or the state of the resource.

¹⁷ <https://martinfowler.com/bliki/ContractTest.html>

¹⁸ <https://martinfowler.com/bliki/BroadStackTest.html>

¹⁹ <http://jbehave.org/>



As an example, if a caching layer is present in the application topology, calls to data may run slower early in the operation of an application. That is because not much data is cached yet, so the application has to make calls into a relatively slower database. Later in the operation of an application, calls to data may run much faster, because most data can be retrieved in a call to the caching layer rather than requiring a call to the database. In some respects, it is probably better to think of a microservices application as a dynamic environment with constant change occurring. It is critical to go beyond simple functionality testing and implement load testing to observe how well the application performs when a high number of calls are made to services, or large amounts of data are transferred on the network between individual services. Again, the network will often be the bottleneck. Load testing will expose parts of the application that are not designed to scale and can prevent meltdowns associated with high amounts of user traffic in production. Do not be tempted to have a few colleagues run some tests on the application and call that load testing. There are a number of sophisticated load-testing solutions available, and they excel at generating enough virtual traffic to truly test how well an application stands up to heavy load.

4.4.2 Resilience testing

Microservices applications operate on an endlessly evolving infrastructure environment, and sometimes portions of those environments may encounter failures. For example, individual servers that are running part of a specific service may crash or become unavailable. Alternatively, network segments may stop reliably passing traffic. Larger aggregations of resources (e.g., entire racks or even entire data centers) may stop working.

Microservices applications must be resilient in the face of infrastructure failures. However, production operation—especially during heavy heavy load—is the wrong time to evaluate just how resilient your application is. An appropriate approach to evaluating application resilience is to test whether it can continue operation if the underlying resources fail. Netflix pioneered the practice of suddenly removing portions of an application's infrastructure, or portions of the application itself, and evaluating how well the application performed. Netflix dubbed this (now open source) tool for sudden, random resource destruction Chaos Monkey.



5 CPN PLATFORM - 1ST PROTOTYPE DELIVERY

The first delivery of CPN platform is probably the most important because it includes the delivery of core components and the first deployment and integration of technology bricks. Moreover, first version of the pilots applications will rely on this version of the platform for their execution.

In this paragraph, the process followed to release the first prototype of platform is described.

5.1 APPROACH AND METHODOLOGY

In order to implement and delivery the CPN platform, starting from the bases provided by the already mentioned deliverables, d2.1 Reference Architecture and d3.1 Technology Bricks, an agile methodology and in particular the Scrum framework was applied.

A 3-steps process was defined: two starting activities steps and one cyclic step for task development.

- ➔ Requirements prioritization
- ➔ Mapping Requirements/Technology Bricks
- ➔ Sprints

5.1.1 Requirements prioritization

The first step of the process was the requirements prioritization. In this phase starting from the list of requirements expected for the 1st pilot iteration, as described in d3.1, the media partners (VRT, DW, DIAS), that in the case of the Scrum methodology assumed the role of Product Owners, gave a priority for each user requirement expected. The result of this activity was an ordered list of requirements, named Prioritized Backlog.

Below the Prioritized Backlog:

1. UR- UP1.2 - The system should create/refine interests based on the user's consumption habits
2. UR- UP1.6 - The system should assign preferences (1-5) to categories based on the users behaviour
3. UR- AF4.1 - The system should be able to personalise news from/for the CPN media partners (VRT, DIAS, DW)
4. UR- UP9.2 - The system should require informed and explicit consent for processing of personal user data, beyond those required for the provisioning of the agreed service
5. UR- UP9.1 - The system must provide transparent, simple and easy-to-understand information on what user data are collected, for what purpose and how they are stored



6. UR- UP1.8 - The system must allow users to completely turn off the personalisation algorithm and receive content as is and vice versa
7. UR- UP3.2 - The system should create/refine time frames based on the user's consumption habits
8. UR- AF2.4 - The system should show users only a limited number of items at once
9. UR- UP5.2 - The system should allow the user to set a home/main interest location
10. UR- AF7.2 - The system should include guided feedback for specific elements of the system, allowing users to (help) improve it
11. UR- UP1.4 - The system should refine the user's interests through frequent interaction with the user (talkback)
12. UR- UP3.3 - The system should refine the user's time frames through frequent interaction with the user (talkback)
13. UR- AF2.5 - Once all articles proposed have been consumed, the system should only offer more content upon request by the users
14. UR- UP1.7 - The system should allow users to assign and change preferences (1-5) to categories themselves
15. UR- UP3.1 - The system must allow the user to choose a preferred time frame or frames to consume content
16. UR- UP3.5 - The system must allow the user to postpone a time frame for a chosen amount of time.
17. UR- UP3.6 - The system must allow the user to ignore a time frame completely
18. UR- AF3.4 - The system should be able to offer both news content and entertainment
19. UR- AF3.5 - The system should be able to offer both locally and globally relevant content
20. UR- UP2.7 - The system should allow users to share content from the CPN system to social networks
21. UR- AF4.2 - The system should allow for additional content sources, outside the consortium
22. UR- AF1.5 - The system should allow users to choose favourite sources
23. UR- AF3.8 - The system should allow users to filter content by language

5.1.2 Mapping Requirements/Technology Bricks

The second step after the definition of Prioritized Backlog was mapping between these requirements and Technology Bricks. This phase involved both media and technical partners, which, in the Scrum Framework, represent the Scrum Team.

The Technology Bricks expected in this phase are the following:



- ➔ User Modelling (LiveTech)
- ➔ Recommender (LiveTech)
- ➔ Producer's App (ENG)
- ➔ Reader's App (ATC)
- ➔ Personal Data Receipts (DigiCat)
- ➔ Relation Extractor (Imec)

Starting from this list of a basic workflow was created to define the process for their integration within the platform. The workflow includes the interaction among modules and the data exchanged.

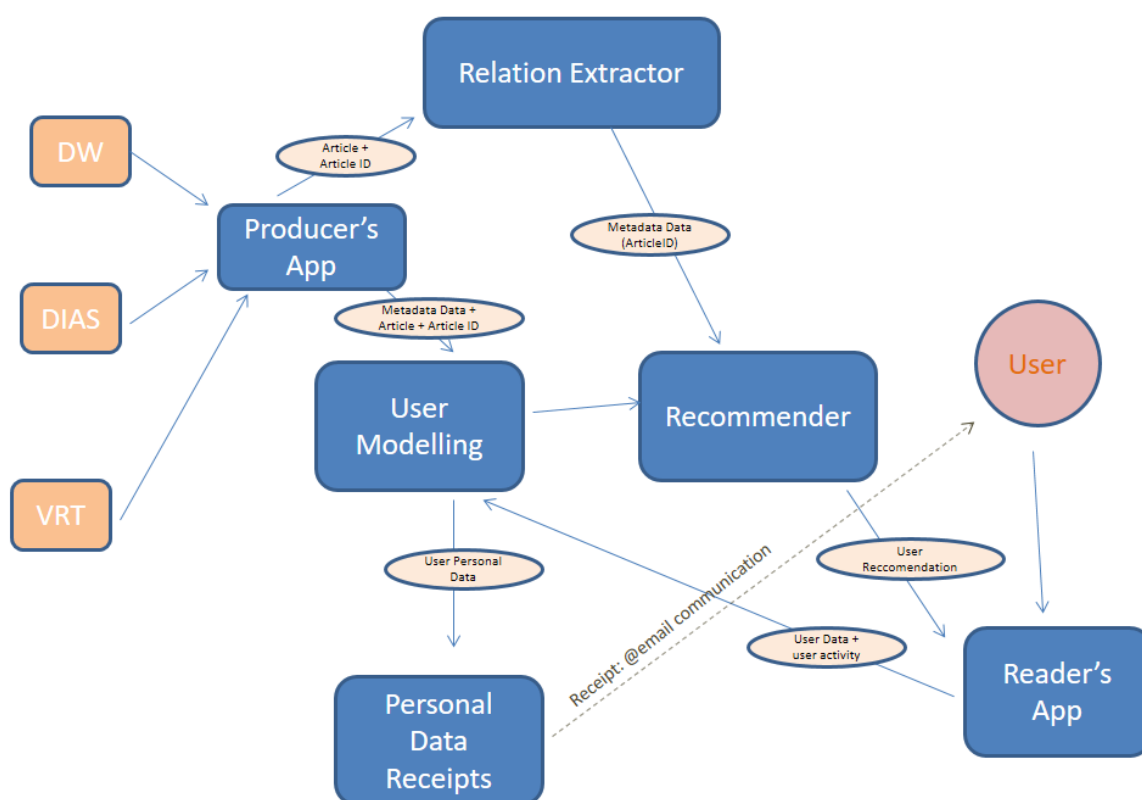


Figure 9: 1st Prototype basic workflow

Once it became clear how all the components were integrated with each other, the requirements were transformed in tasks and mapped with related Technology Bricks.

5.1.3 Sprints

The third step was a cyclic one. In fact, after the starting steps, following the Scrum Framework, a series of sprints was defined.

Each sprint duration was of two weeks and each sprint consisted of:

- ➔ Sprint planning
- ➔ Sprint execution
- ➔ Sprint review

The sprint planning is the starting phase of a sprint. During this activity, the tasks to be completed in the sprint are selected.

The sprint execution is the second phase, which continue for all the duration of the sprint (two weeks). During this phase, the tasks are executed and completed.

The sprint review is the final phase of the sprint. In this phase tasks completed and not are discussed. The completed tasks are demonstrated by the Scrum Team. The Scrum Team also discusses what went well during the Sprint, what problems it ran into, and how those problems were solved. Finally, the sprint review provides valuable input to subsequent sprint planning.

5.2 RESULTS

The first platform delivery includes some of the requirements expected for the first pilot iteration, the overall requirements set will be completed in next two months.

The main goal of this version of platform was demonstrate the potentiality of micorservices architecture and in particular of the architectural choices made for CPN.

All the core components and the Technology Bricks expected for the first pilot were deployed, integrated and tested successfully and all the services were documented and exposed through the API Gateway.

The result of this deployment is represented in the figure below:



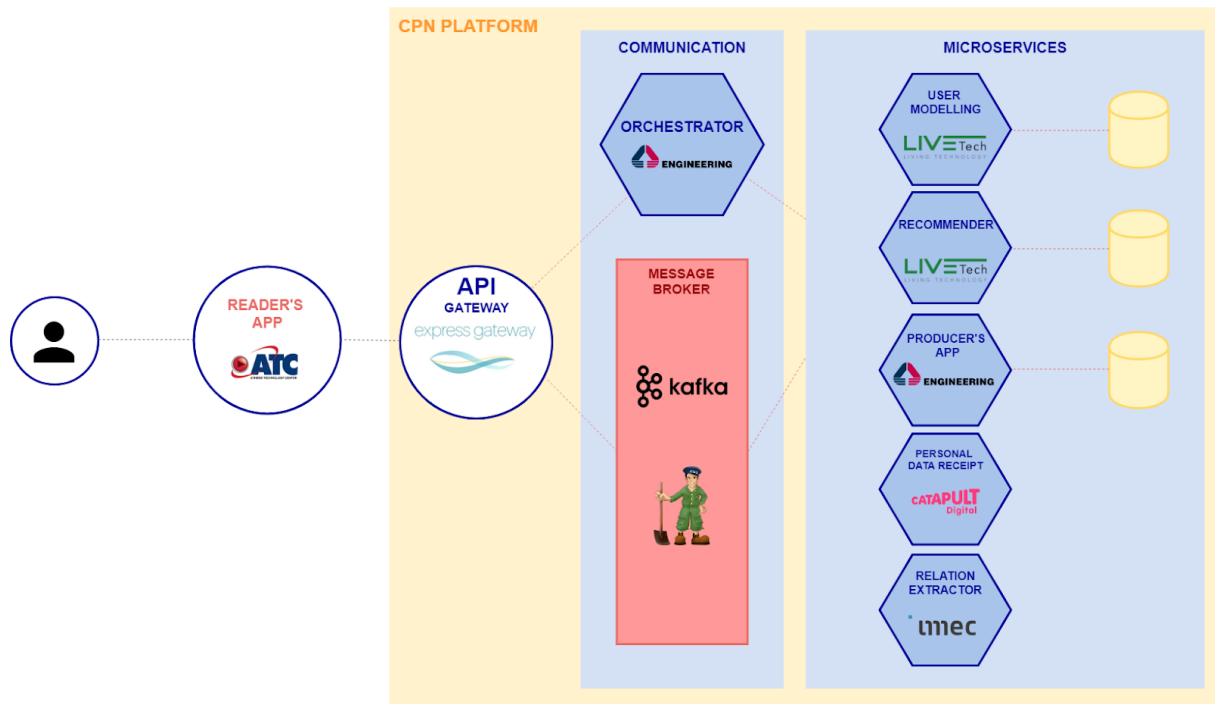


Figure 10: Deployment of CPN platform v1

More in detail, the following functionalities were implemented:

- ➔ Signup and login system
- ➔ Authentication and authorization system managed by API gateway with JWT tokens
- ➔ Background process for scheduled contents extraction from media partners (DW, VRT, DIAS)
- ➔ Background process for content analysis and recommendation process
- ➔ Basic recommended system and news visualization
- ➔ Collection of user interests and activities
- ➔ Production of User Data Receipts
- ➔ Analysis and extraction of additional information from contents

6 CONCLUSIONS

This document represents the report of all the activities conducted for the release of the first version of CPN Open Virtual platform.

The CPN platform v1 was packaged and it is available for the partners in internal repository.

In order to test the platform and verify the status of delivery as described in this report, the following software prototypes are available to internal partnership:

- ➔ CPN Microservices Platform: container management platform (see figures 1-4)
- ➔ CPN API Gateway: gateway with APIs for the client applications (see figure 5)
- ➔ CPN Microservices Registry: public registry of versioned Technology Bricks
- ➔ Test Client Application

The next activities will be focussed on Technology Bricks and platform improvements, in order to satisfy the complete set of all the requirements of both first pilot execution (planned for early September) and the second version of the platform (planned for the end of May 2019).

