This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement n: 761999

**EasyTV: Easing the access of Europeans with disabilities to converging media and content.**

# D5.1 - Mid-term report on the set up and implementation of the EasyTV multi terminal technical platform

**EasyTV Project**

*H2020. ICT-19-2017 Media and content convergence. – IA Innovation action.*

**Grant Agreement n°: 761999**

Start date of project: 1 Oct. 2017

Duration: 30 months

Document. ref.: D5.1

## Disclaimer



The EasyTV Consortium consists of the following partners:

| | Partner Name | Short name | Country |
|---|---|---|---|
| 1 | Universidad Politécnica de Madrid | UPM | ES |
| 2 | Engineering Ingegneria Informatica S.P.A. | ENG | IT |
| 3 | Centre for Research and Technology Hellas/Information Technologies Institute | CERTH | GR |
| 4 | Mediavoice SRL | MV | IT |
| 5 | Universitat Autònoma Barcelona | UAB | ES |
| 6 | Corporació Catalana de Mitjans Audiovisuals SA | CCMA | ES |
| 7 | ARX.NET SA | ARX | GR |
| 8 | Fundación Confederación Nacional Sordos España para la supresión de barreras de comunicación | FCNSE | ES |
| 9 | Unione Italiana dei ciechi e degli ipovedenti | UICI | IT |

| | |
|---|---|
| **PROGRAMME NAME:** | **H2020. ICT-19-2017 Media and Content Convergence – IA Innovation Action** |
| **PROJECT NUMBER:** | 761999 |
| **PROJECT TITLE:** | EASYTV |
| **RESPONSIBLE UNIT:** | ENG |
| **INVOLVED UNITS:** | ENG, UPM, ARX |
| **DOCUMENT NUMBER:** | D5.1 |
| **DOCUMENT TITLE:** | Mid-term report on the set up and implementation of the EasyTV multi terminal technical platform |
| **WORK-PACKAGE:** | WP 5 |
| **DELIVERABLE TYPE:** | Report |
| **CONTRACTUAL DATE OF DELIVERY:** | 31-10-2018 |
| **LAST UPDATE:** | 25-10-2018 |
| **DISTRIBUTION LEVEL:** | PU |

**Distribution level:**

**PU** = *Public,*

**RE** = *Restricted to a group of the specified Consortium,*

**PP** = *Restricted to other program participants (including Commission Services),*

**CO**= *Confidential, only for members of the LASIE Consortium (including the Commission Services)*

# Document History

| VERSION | DATE | STATUS | AUTHORS, REVIEWER | DESCRIPTION |
|---|---|---|---|---|
| v. 0.1 | 20/07/2018 | Draft | Giuseppe Vitolo ENG | Table of Contents definition and document structure |
| v. 0.2 | 23/07/2018 | Draft | Giuseppe Vitolo ENG | Table of Contents changes |
| v. 0.3 | 11/09/2018 | Draft | Giuseppe Vitolo ENG | First draft |
| v. 0.4 | 17/09/2018 | Draft | Giuseppe Vitolo ENG | Draft corrections |
| v. 0.5 | 03/10/2018 | Draft | Giuseppe Vitolo ENG | Added more information (infrastructure, containerization, API management) |
| v. 0.6 | 18/10/2018 | Draft | Silvia Uribe UPM | Added "API Design" chapter |
| v. 0.7 | 24/10/2018 | Draft | Stavros Skourtis ARX | Review |
| v. 0.8 | 24/10/2018 | Draft | Kosmas Dimitropoulos CERTH | Review |
| v. 0.9 | 25/10/2018 | Final Candidate | Giuseppe Vitolo ENG | Minor corrections |

# Definitions, Acronyms and Abbreviations

| ACRONYMS / ABBREVIATIONS | DESCRIPTION |
|---|---|
| API | Application Programming Interface |
| CLI | Command Line Interface |
| CNM | Container Networking Model |
| DASH | Dynamic Adaptive Streaming over HTTP |
| GUI | Graphical User Interface |
| HTML | HyperText Markup Language |
| IDE | Integrated Development Environment |
| IP | Internet Protocol |
| IT | Information Technology |
| JSON | Javascript Object Notation |
| OS | Operative System |
| RAML | RESTful API Modeling Language |
| REST | Representational State Transfer |
| SLT | Sign Language Task |
| SQL | Structured Query Language |
| TCP | Transport Control Protocol |
| UDP | User Datagram Protocol |
| URI | Uniform Resource Indicator |
| URL | Uniform Resource Locator |
| vCPU | Virtual Central Processing Unit |
| VM | Virtual Machine |
| vRAM | Virtual Random Access Memory |
| YAML | YAML Ain't a Markup Language |

# Table of Contents

# List of Figures

# List of Tables

# Executive Summary

This document is the first deliverable in WP5. It has been written by ENG with the contribution of UPM. It focuses on the description of technologies, technical choices and strategies that will be employed to implement the EasyTV multi terminal technical platform.

The first chapter is an introduction to the concept of “platform” and it outlines the main technologies on which the platform will be built.

The second chapter (“Infrastructure”) compares two main infrastructure types (on-premises and cloud), highlighting their pros and cons, and it subsequently describes the infrastructure choice made for the EasyTV project.

The third chapter (“Containerization platform”) introduces the concept of “application containerization” and describes Docker, which is the leading application containerization solution on the market. Moreover, it describes a generic application development lifecycle based on Docker and lists the EasyTV modules to be containerized, giving a detailed description of each module. Finally, the concept of container orchestration is addressed.

The fourth chapter (“Service API management”) deals with API management, which will be supported by the Kong platform. After an introduction to the main API management ideas, the Kong platform is described, followed by an example of service configuration.

The fifth chapter (“API design”) introduces the concept of API design in the context of the EasyTV project, giving insights on how a RESTful API should be designed and documented. An example is given, too.

Finally, conclusions and future work are outlined in the fifth chapter.

# 1. INTRODUCTION

A **platform** is a set of hardware and software technologies which enable the execution of domain-specific services and applications. It can be based on two types of infrastructure model: "**on-premises**" and "**cloud**".

The **EasyTV multi terminal technical platform** is the platform that is going to host the **EasyTV services** and make them available to end users by leveraging application **containerization**, which sits at the core of the EasyTV multi terminal technical platform implementation. Currently, the leading software containerization platform on the market is **Docker**: it allows to run multiple applications or services on a single machine or in a cloud environment by efficiently sharing the services on containers, virtual volumes and virtual networks.

On top of that, an **API management** solution will be deployed, offering a reliable and consistent way of managing the APIs exposed by the services running on the platform. The proposed solution will be based on **Kong**, one of the most popular API management tools. In addition to the basic API gateway features, it offers many other ones like rate limiting, monitoring, caching, logging and analytics. In addition, Kong is built on a plugin-based architecture, so it can be customized and fine-tuned to obtain the desired behaviour. Of course, it is of foremost importance that each component exposes a well-designed RESTful API: **Swagger** will be the main tool used to support the API design and documentation process.

By employing the previously mentioned technologies and tools and following the architecture evolution from D1.3 ("First release of the EasyTV system architecture") and D1.4 ("Final release of the EasyTV system architecture"), the EasyTV multi terminal technical platform will take care of the runtime aspects of the EasyTV services running on it, ensuring flexible provisioning and managing the whole lifecycle of the services. Services will be offered in a reliable, secure and scalable way, regardless of whether the final infrastructure be a server farm at the content broadcaster's premise or a private/public cloud environment.

# 2. INFRASTRUCTURE

**IT infrastructure** is defined as “the system of hardware, software, facilities and service components that support the delivery of business systems and IT-enabled processes” [1].

Within the context of the EasyTV project the **IT infrastructure** allows the **EasyTV multi terminal technical platform** to run in a smooth, consistent and reliable manner.

Infrastructure can be centralized in a data centre (content broadcaster’s premises) or spread across multiple data centres. These decentralized data centres can be controlled by the broadcaster or by a third party, such as a **cloud provider**.

## 2.1. Infrastructure types

### 2.1.1. On-premises

**On-premises** (often abbreviated as “on-prem”) refers to private data centres that companies house in their own facilities and maintain themselves.

Software runs on the premises of the owner company, thus allowing more control on the underlying resources, since the IT staff has physical access to the infrastructure and can directly control the configuration, management and security of the computing infrastructure and data.

However, since this kind of infrastructure is owned by the company, the latter is responsible for maintaining and upgrading the whole system and for ensuring high availability and disaster recovery.

### 2.1.2. Cloud

In a **cloud infrastructure** model a third-party service provider makes computing resources available for consumption on an as-needed basis.

As noted in [2], some of the benefits of **cloud computing** are:

- Flexibility: users can scale services to fit their needs, customize applications and access cloud services from anywhere with an internet connection.
- Efficiency: enterprise users can get applications to market quickly, without worrying about underlying infrastructure costs or maintenance.
- Strategic value: cloud services give enterprises a competitive advantage by providing the most innovative technology available.

Of course, there are also some downsides related to cloud computing: customers have limited control over the infrastructure, since they can only control and manage applications, data and services operated on top of it and not the backend infrastructure itself. Also, data security could be an issue if the third-party service provider does not implement the best security standards and policies.

## 2.2. EasyTV infrastructure hosting

The EasyTV multi terminal technical platform will be hosted at ENG premises, as ENG’s data centres will provide the highest standards in terms of security, reliability and efficiency. The architecture is based on the latest generation Intel blade systems, optimised to support environment virtualisation and allowing both vertical and horizontal scalability.

The initial deployment will be done on an environment configured as specified in the following table:

| No. of vCPU | vRAM (GB) | Storage (GB) | OS |
|---|---|---|---|
| 4 | 8 | 100 | CentOS Linux 7.x |

**Table 1. Server configuration**

Both a public IP address and a public hostname will be assigned to the server, thus allowing it to be reached from the outside. Network firewall will be configured accordingly to accept traffic on specific TCP/UDP ports.

Finally, it is worth mentioning the fact that the infrastructure will support daily hot (online) disk backups to ensure data safety.

# 3. CONTAINERIZATION PLATFORM

## 3.1. Introduction to app containerization

Containerization is an innovative kind of technology that uses the kernel on the host operating system to run multiple root file systems. Each root file system is called **container**: containers offer a logical packaging mechanism in which applications can be abstracted from the environment in which they actually run.

Containerization provides a clean separation of concerns, as developers can focus on their application logic and dependencies, while IT operations teams can focus on deployment and management without bothering with application-specific details such as library versioning and configuration. Moreover, it offers a lightweight alternative to full machine virtualization: in fact it is important to note that containers are not VMs. Containers leverage shared resources, do not require a hypervisor and provide greater portability while VMs, on the other hand, are hypervisor-based and require a full OS to run.

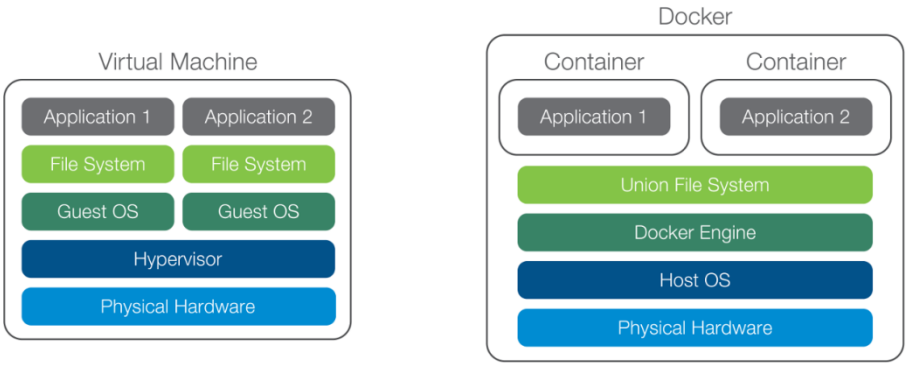


**Figure 1 - A comparison of virtual machines and Docker[1]**

## 3.2. Docker overview

The server-side modules that will be running on the **EasyTV multi terminal technical platform** will be "packaged" using the Docker container technology. Docker is "a platform for developers and sysadmins to develop, deploy, and run applications with containers" [3].

The core of the Docker platform is the **Docker Engine**, which is a client-server application consisting of the following major components [4]:

- A server which is a type of long-running program called a daemon process (the *dockerd* command).
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client (the *docker* command).

[1] https://www.f5.com/content/dam/f5-com/page-assets-en/home-en/resources/white-papers/using-docker-container-technology-with-f5-products-and-services-wp-ag-amer-54662827-dockers-tech-diagrams-v7-diag-1.png (retrieved 14/09/2018)

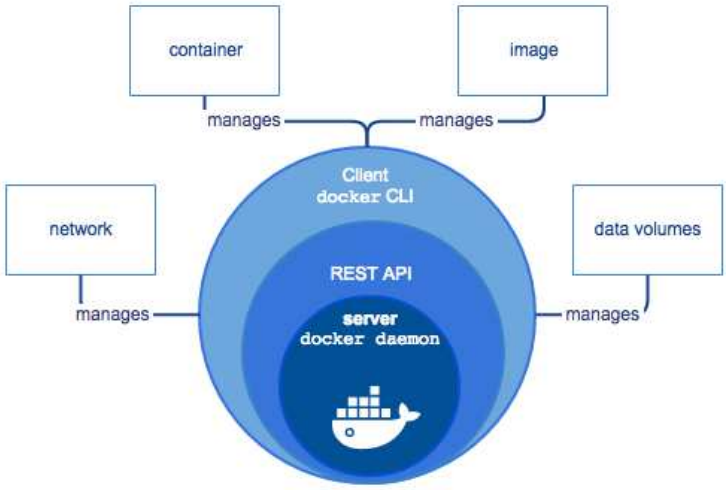


**Figure 2 – Docker Engine components**[2]

Docker employs different underlying technologies to deliver its functionality. Some of them that are worth mentioning are **control groups**, **kernel name spaces** and the **union file system**. Control groups ("cgroups") are "a Linux kernel feature which allow processes to be organized into hierarchical groups whose usage of various types of resources can then be limited and monitored [5], allowing Docker to share available resources to containers and to enforce limits and constraints (e.g. CPU usage limits and memory usage limits). A kernel namespace (or simply namespace), on the other hand, "wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource" [6]. In particular, Docker Engine on Linux uses the following namespaces: *pid* (process ID), *net* (networking), *ipc* (inter process communication), *mnt* (mount), *uts* (Unix timesharing system). Finally, union file system (UnionFS) is a particular kind of filesystem which amalgamates a collection of different file systems and directories (called *branches*) into a single logical file system (more details can be found in [7]).

## 3.3. Docker objects

### 3.3.1. Images

An image is a snapshot of an application and serves as the basis of a container. It includes all of the requirements for running a single Docker container, as well as metadata describing its needs and capabilities. Often, an image is based on another image, with some additional customization.

Images are stored in a Docker **registry**, which can be used to store and retrieve images in a reliable and consistent way, thus simplifying the whole deployment process.

### 3.3.2. Containers

A container is a runnable instance of an image. It is possible to create, start, stop, move or delete a container using the Docker API or CLI. A container is defined by its image as well as any configuration options provided at creation and/or starting time.

[2] https://docs.docker.com/engine/images/engine-components-flow.png (retrieved 11/09/2018)

By default, a container is isolated from other containers and the host machine, but of course it is possible to control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

#### 3.3.3. Networks

Networks are based on a set of interfaces called the **Container Networking Model** (CNM), which brokers connectivity for the containers and abstracts away the networking diversity and complexity [8]. The abstraction provided by the CNM can be used to support multiple **network drivers**, which provide the actual network implementation.

Several drivers exist by default, and provide core network functionality [9]:

- *bridge*: this is the default network driver. It creates a private network internal to the host so containers on this network can communicate, and provides isolation from containers which are not connected to that bridge network.
- *host*: for standalone containers, remove network isolation between the container and the Docker host and use the host's network directly.
- *overlay*: the overlay network driver creates a distributed network among multiple Docker daemon hosts. This network sits on top of the host-specific networks, allowing containers connected to it to communicate securely.
- *macvlan*: macvlan networks allow to assign a MAC address to a container, making it appear as a physical device on the network. The Docker daemon routes traffic to containers by their MAC addresses.
- *none*: this driver disables networking for a container.

#### 3.3.4. Volumes

Volumes are the preferred way to persist data generated by and used by Docker containers. Volumes are necessary because data created inside a container doesn't persist when the container is no longer running (while data stored inside a volume exists outside the lifecycle of a given container).

## 3.4. App development on Docker

#### 3.4.1. App development process overview

The application development process steps are clearly outlined in following figure:

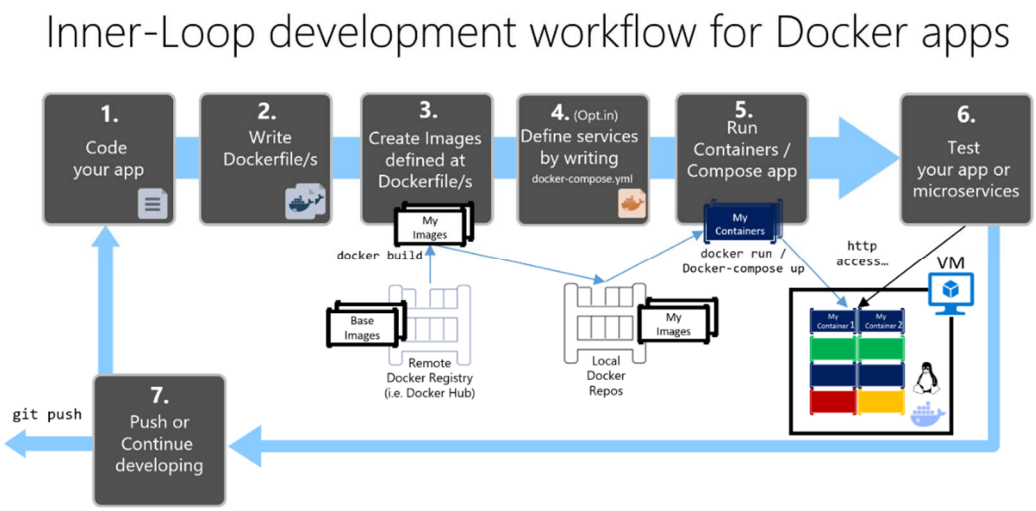


**Figure 3 - Docker development workflow**[3]

The first step is to create the initial application/service baseline, from which all subsequent development can take place. While developing a Docker app/service, the developer deploys and tests it in a suitable local Docker environment.

Once the code baseline has been established, the developer has to create a suitable **Dockerfile**, a file which contains the commands that tell Docker how to setup and run the app/service in a container. The Dockerfile is placed in the root folder of the app/service project.

After the Dockerfile has been defined, an image should be created (some IDEs automatically build a Docker image, otherwise the Docker CLI is needed).

It is also possible, as an optional step, to define a set of services in a **docker-compose.yml** file, which defines a set of related services to be deployed as a composed application.

At this point, the app/service can be started. If it only has a single container, it can be started by deploying it to the local Docker host. On the other hand, if it contains multiple services, it can be deployed as a composed application.

Finally, the app/service should be tested using a local Docker environment (at the developer machine) and then the final Docker image can be pushed to a registry, so that it can be pulled and executed on any machine.

### 3.4.2. EasyTV module containerization

In this subsection an example of containerization and deployment of an EasyTV module is given. Since the whole process is similar for each EasyTV module, a significant example is given (the complete list of the EasyTV modules to be containerized, along with their description but without the steps used to produce and deploy the docker image, is presented in the next section).

The target application to be containerized is an EasyTV module based on node.js which exposes an HTTP REST API and stores data on MongoDB, a NoSQL JSON oriented database.

First of all, an image should be created. There are two possibilities to package the application: either by extending an official Linux distribution image (like Alpine or Ubuntu) and install the node.js runtime or by using the official node.js image. Let's suppose the second option is chosen.

The Dockerfile looks like this:

[3] https://docs.microsoft.com/it-it/dotnet/standard/microservices-architecture/docker-application-development-process/media/image1.png (retrieved 11/09/2018)

```
FROM node:8.12.0

# Copy source code
COPY . /app

# Change working directory
WORKDIR /app

# Install dependencies
RUN npm install

# Expose API port to the outside
EXPOSE 80

# Launch application
CMD ["npm","start"]
```

The Dockerfile instructs Docker to use node 8.12.0 as the base image, to copy the application sources and to install dependencies as well as to expose port to the outside from the Docker host. The last command defines how to execute the application when instantiating the image.

To create the image (named “easytvserviceA”) the following command is issued from a command shell:

```
$ docker build -t easytvserviceA .
```

The image can be pushed to Docker Cloud (if the developer has a Docker Hub account) or to another remote Docker registry with the *docker push* command. It can then be pulled and used on any other host with the *docker pull* command:

```
$ docker pull developer/easytvserviceA
```

A user-defined network should be created (named, for example, “easytvnet”) to allow the application to communicate with the database. To do so, the docker *network create command* is used:

```
$ docker network create easytvnet
```

The application container and the database container can be finally started (note that each container should be connected to the “easytvnet” user-defined network):

```
$ docker run --name mongo --net easytvnet -d mongo:3.6
$ docker run –name easytvserviceA –net easytvnet –p 8000:80 –d –e ‘MONGO_URL=mongodb://mongo/easytvdb’ easytvserviceA
```

By using the *–e* flag, environmental variables can be set for the container. In the above example the URL of the database is set at runtime as an environmental variable. It is worth noting that it is possible to define a *docker-compose.yml* file which defines both the application and the database configuration (the docker compose tool is described in subsection 3.6.1).

## 3.5. Modules to be containerized

Descriptions of the EasyTV modules to be containerized are presented in this section. The information is drawn from D1.4 ("Final release of the EasyTV System Architecture").

### 3.5.1. Service Manager

The Service Manager will be the main communication hub between the broadcasters' premises, and the different components, modules, and services at the internal of the EasyTV platform. It will act as a gateway and orchestrator of the full platform and it will allow an abstraction of the work and processes that can involve multiple modules.

The Service Manager will have two main components:

- A **web graphical user interface** (GUI), which will allow the broadcaster to request content from the EasyTV platform in a centralized and unified way. This web GUI will keep track of their request, status, and serve the contents to the broadcaster premises in a user-friendly way without the need of knowing the underlying structure of the EasyTV modules, services, and components.
- A **REST API**, acting as a middleware between the user interface (and other possible services that the broadcasters may deploy in the future) and the EasyTV services.

The Service Manager will be able to address the specific content (accessibility and audiovisual contents) request to the appropriate EasyTV component where the repository for that content resides. If the requested content is available, it will be directly served to the broadcaster. If the requested content is not available (or doesn't exists yet), the Service Manager will generate the needed tasks in the EasyTV modules, allowing the broadcaster to know in every moment the status of the request.

As some of the tasks could be dependant of previous tasks, the Service Manager will take care of the correct workflow progress and monitoring.

### 3.5.2. Automatic Descriptive Narratives Module

This module generates a textual file similar to a subtitles file that contains valuable automatically generated data obtained from the intrinsic information of the multimedia contents and from different external information sources, such as the metadata associated to the file and the television guides. This information is related to the preferences of the user and it is suitable for obtaining real-time data about the contents that the user is watching. Moreover, this process will be completed with the textual information extraction from the video in order to provide more data that can improve the experience of the users, especially if they are visually impaired.

Once the audionarrative file is generated, the file is stored in the accessibility contents database through the service manager.

### 3.5.3. Automatic Voice Synthesis of Subtitles Module

The aim of this module is to provide spoken subtitles by using a TTS (Text-To-Speech) service available on the server side and another TTS service on the cloud for a more flexible and effective solution. Subtitles will be provided by the subtitling production platform available in EasyTV or directly through a Web Based application that can be used by professional users. The Automatic voice synthesis of subtitles will include a set of functionalities available through two different interfaces: the Web API interface and the Web GUI interface. The Text To Speech Service (TTS) (local or on the cloud) will be also used for translating any text in an audio track to be included or mixed to an audio video streaming content, for example for audio narratives.

### 3.5.4. Clean Audio Module

This module allows the user to change the balance of the audio mix according to the listening

environment or personal needs, obtaining important benefits for the hearing-impaired, including the people affected by hearing loss that increases gradually with age.

The clean-audio component interacts with the sound from the source multimedia content and develops a process for equalization of frequencies and advanced techniques of filtering for enhancing the intelligibility of the audio by identifying, processing and reinforcing the bandwidth

where the most important audio information is located, especially the voices.

### 3.5.5. **Image Enhancement Module**

The main objective of image enhancement is the processing of an image for obtaining a more suitable result that the original image/video, which is adapted to the requirements of the user, especially when this user has impaired vision. Digital image enhancement techniques provide a multitude of choices for improving the visual quality of images. Appropriate choice of such techniques is greatly influenced by the imaging modality, task at hand and viewing conditions. The variety of algorithms commonly used for image enhancement is large, depending on the available processing resources and the real-time limitation. There are multiple techniques based on spatial domain techniques, with special reference to histogram processing and point processing methods.

On the EasyTV Cloud Platform this module that automatically detects texts and faces of an audiovisual content. This module generates a text file with the position where the content to be magnified is located. *Once* the text file is generated, the file is stored in the accessibility content database through the service manager. Then, the client application can obtain the text file and magnifies the content with the information obtained.

### 3.5.6. **Subtitle Production Module**

The aim of the Subtitle Production Module is to facilitate the translation jobs in order to achieve a good quality of translations in a reasonable term. With this objective in mind, two components have been included in the workflow of subtitle translation: the **Automatic Subtitles Translation Component** and the **Human Subtitling Production Tool**.

The Automatic Subtitles Translation Component provides a first translated version of the subtitles in the languages of interest indicated by the broadcaster/content owner. This is achieved by using free translation services available in the cloud. The translation process takes place as soon as a new job is mandated to the platform, and its results are also stored in the Data Storage Component and used as a starting point by the editorial users of the crowdsourcing platform in the Human Subtitling Production Tool.

The Human Subtitling Production Tool consists of a web-based interface that allows the collaborative users to perform the requested translation tasks. The users will be granted to the tool by means of the crowdsourcing platform.

### 3.5.7. **Sign Language Production Module**

The Sign Language Production module is responsible for the creation, validation and exploitation of the sign language content in different languages for easing the access of hearing impaired individuals to media content and services.

This module is made up of the following components:

- Sign Language Crowdsourcing: the purpose of this procedure is to allow users to contribute with sign languages and translations of available signs, creating a multilingual sign language repository. The administrators of the EasyTV Crowdsourcing platform will be responsible for defining Sign Language Tasks (SLTs) by proposing words/sentences that need to be translated in sign language and distribute these SLTs to expert sign language users. The users will then use a capturing setup in order to perform the requested signs, and by using a capturing component, record and upload the data to the Crowdsourcing Platform. The administrators will validate the users' contributions by stating an SLT as completed.

Afterwards, the multilingual ontology will be responsible for annotating data in order to create a 1-to-1 mapping between the recorded signs, the text corresponding to the signing, and the concepts in the multilingual ontology. This will allow to automatically establish translation relations among signs in different languages.
The enriched multilingual ontology can also be linked with existing repositories that contain signs in different languages (e.g. Spreadthesing), laying the foundations for the creation of a sub-cloud of resources that contain sign language data. Finally, the motion data along with the corresponding text and concept translation will be stored in a sign language repository for use by the other EasyTV services/modules. The Sign Language Crowdsourcing will also allow a user to search and view already recorded signs.

- Sign Language Capturing component: the purpose of this component is to allow users to record signs and upload them to the EasyTV Crowdsourcing Platform. Initially, the user will connect to the crowdsourcing platform and through his/her account, accept the assigned SLTs which recommend the signing of specific words or sentences. However, the user is not limited to completing the SLTs but he/she can perform his/her own signs given that he/she annotates them correctly. The Sign Language Capturing component will be responsible for processing the recorded data in order to extract valuable motion information that will be used for the correct differentiation among different signs. The recorded data will also accompany a text describing the signing sequence (word/sentence).
  At the next phase, there is an optional possibility of building a semi-automatic annotation algorithm that will accept the recorded data and the text describing them and output a data annotation that breaks down the video in frames and provides starting and ending frame numbers for each recorded sign. The semi-automatic annotation tool will be based on accurate and robust machine learning techniques applied on already learned signs. Finally, the expert user/signer will be able to view data annotation and perform corrections before uploading the data on the Sign Language Crowdsourcing component.
- Realistic Avatar: the purpose of this component is to allow the visualization of the recorded signs using an avatar. This component will communicate directly with the data repository of the Sign Language Crowdsourcing component and will acquire and play the sign representations using 3D motion data of face and hands.

### 3.5.8. **Hyper-personalization Module**

The EasyTV hyper-personalisation module will be built on top of profiling- user experience mining techniques; extraction, abstraction, homogenisation of dynamic user profiles and innovative interaction techniques for interface adaptation. This module will be able to handle dynamic and continuous changes in user experience in a user-transparent way in order to recommend new personalised services, dynamically adapted to the current context and device of the user.

The EasyTV hyper-personalisation module will consist of the following sub-components:

- EasyTV user models: the EasyTV user models will define user needs and preferences, including also the disabilities and functional limitations of the user. The structure of the EasyTV user models will be based on previous work conducted in previous projects, such as Cloud4All and VERITAS, as well as on the user models proposed by the VUMS cluster. Further extensions of the aforementioned user models will be developed to cover the EasyTV specific needs.
- User model editor**:** the EasyTV User Model Editor will be a web-based tool that will enable the easy creation and editing of EasyTV end user models through intuitive web forms.
- Hybrid matchmaker: the EasyTV hybrid matchmaker will perform matchmaking between user needs and preferences defined in user profiles and UI capabilities, accessibility features specifications and DASH streaming services specifications.
- UI adaptation and accessibility features configuration module: this module will take as input the results of the hybrid matchmaker and will perform automatic turn on and configuration of accessibility features (e.g. volume, rate, pitch, colour preferences, etc.) that are built into different TV operating systems, applications and embedded devices that will be supported.

- Content adaptation module: the EasyTV content adaptation module will perform content adaptation based on the results of the hybrid matchmaker in order to offer streaming content in the best possible form for a specific user (e.g. by selecting the audio that corresponds to user's language).

### 3.5.9. Crowdsourcing Platform

The Crowdsourcing Platform provides an infrastructure for implementing the sign language production and subtitle production procedures. More specifically, it includes functionality for task definition, distribution and validation, and its purpose is to allow the creation and management of professional user profiles and the access of users in the sign language and subtitle production procedures through web Graphical User Interfaces (GUIs).

## 3.6. Container orchestration

Container orchestration defines both the initial deployment of the containers and the management of multiple containers as a single entity. Specifically, container orchestration encompasses a certain number of features, including container instantiation, container execution rescheduling, container linking through interfaces, cluster scaling.

The Docker ecosystem includes tools which enable container orchestration: the **Compose** tool, "which is a tool for defining and running multi-container Docker applications" [10], and the **Swarm mode**, which is based on SwarmKit [11] and allows to orchestrate multiple containers running on different Docker hosts.

### 3.6.1. Docker Compose

The Compose tool enables the definition and the execution of multi-container Docker applications. The services that make up an app/service are defined in a *docker-compose.yml* file which is subsequently used by the **docker-compose** application to start and run the entire app/service. Compose can be useful in every environment (production, staging, development and testing) and it can manage the whole app/service lifecycle.

A *docker-compose.yml* should contain at least a "version" key (the Compose file format version) and a "services" key (the services that make up the whole app/service, each one configured accordingly).

As an example, consider the following *docker-compose.yml* file:

```
version: "3"

services:
  db:
    image: postgres:9.4
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - easytvnet
    deploy:
      placement:
        constraints: [node.role == manager]

  easytvserviceA:
```

```
    image: easytv/serviceA:stable
    ports:
      - "5000:80"
    networks:
      - easytvnet
    deploy:
      replicas: 2
      restart_policy:
        condition: on-failure

  easytvserviceB:
    image: easytv/serviceB:stable
    ports:
      - "5001:80"
    networks:
      - easytvnet
    depends_on:
      - db
    deploy:
      replicas: 1
      restart_policy:
        condition: on-failure

networks:
  easytvnet

volumes:
  db-data:
```

In the above example three services are defined: a database and two EasyTV modules. Each module has different configuration options, but it is mandatory to declare the image used to start the container.

Multiple volumes and networks can also be defined and configured: here we have one volume called “db-data”, which is used by the “db” service, and one network called “easytvnet” to which all services are connected.

### 3.6.2. **Swarm mode**

Swarm mode is a feature embedded in the Docker Engine which enables cluster management and orchestration. A **swarm** “consists of multiple Docker hosts which run in swarm mode and act as managers (to manage membership and delegation) and workers (which run swarm services). A given Docker host can be a manager, a worker, or perform both roles” [12].

Hosts which participate in a swarm are called **nodes**:

- **Manager nodes** dispatch tasks to worker nodes and also perform the orchestration and cluster management functions required to maintain the desired swarm state.
- **Worker nodes** receive and execute tasks dispatched from manager nodes and report the current state of its assigned tasks to the manager node.

Swarms can be managed by using the Docker CLI: the main commands are *'docker swarm init'* and *'docker swarm join'* to initialize a swarm and to make a node join an existent swarm respectively.

When Docker is running in *swarm mode* it is also possible to take advantage of **stacks**, which are groups of interrelated services that share dependencies, and can be orchestrated and scaled together [13]. The command *'docker stack deploy'* serves this purpose by accepting a stack description in the form of a Compose file.

## 3.7. Docker Management UI

It is convenient to have a graphical management tool which can efficiently interact with the deployed Docker platform. Portainer [14] is management UI of choice, since it is an open-source, lightweight and easy solution to deploy and use. It provides a detailed overview of images, containers, networks and volumes and it offers advanced management capabilities.

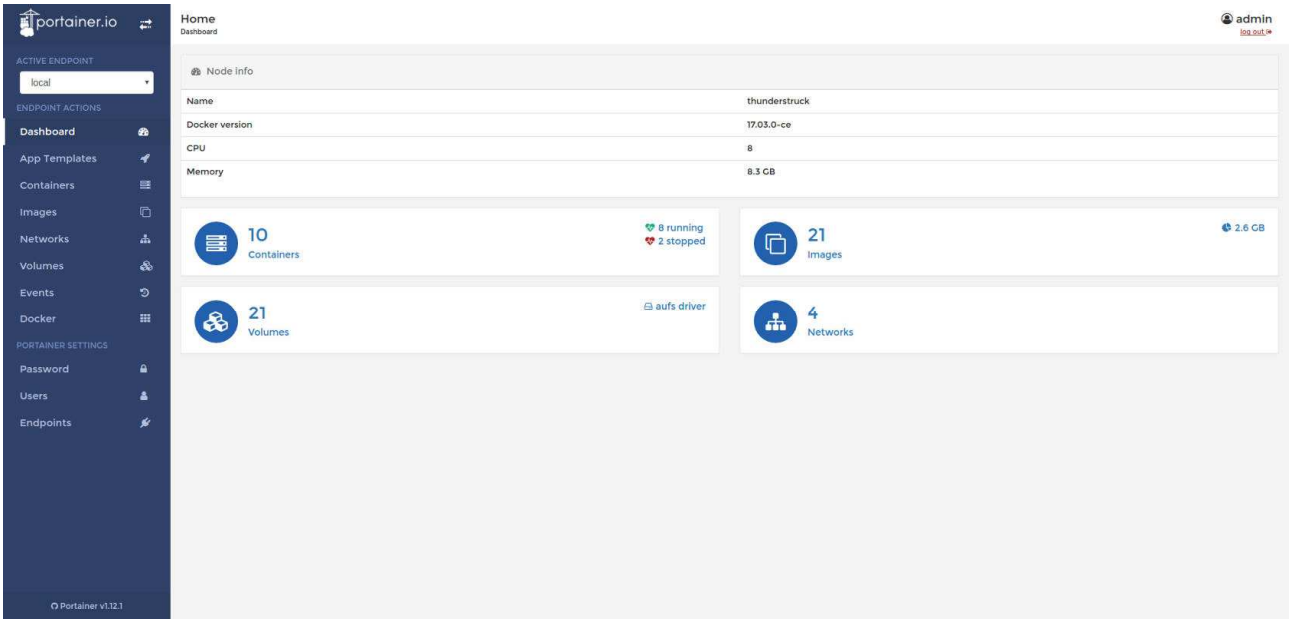


**Figure 4 - Portainer dashboard**

An exhaustive list of Portainer's features is given here [15]:

- containers: list, management, details, statistics, logs, console, creation;
- images: list, management details;
- networks: list, management;
- volumes: list, management;
- container templates;
- cluster overview;
- services management;
- endpoint management;
- user management and user access control.

Portainer itself runs as a Docker container, so it seamlessly integrates with any project ecosystem based on Docker. It is possible to deploy Portainer by using the following commands:

```
$ docker volume create portainer_data
$ docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer
```

Portainer UI will then be accessible on TCP port 9000.

# 4. SERVICE API MANAGEMENT

## 4.1. Introduction to API management

Each EasyTV service module exposes its functionality through an API, so that other modules and client applications can interact with it.

In order to simplify the whole service management lifecycle and to ensure greater scalability, it is useful to provide an **API management** mechanism, usually delivered through a platform or application.

Some of the key features offered by an API management tool are:

- API orchestration
- API identity & security
- Versioning
- Monitoring and logging
- Analytics

## 4.2. Kong API platform overview

**Kong** is an open source API management solution which comes both in a community edition and in an enterprise edition. It leverages **Nginx** capabilities by using **OpenResty**, a dynamic web platform based on **Nginx** and **LuaJIT** [16].

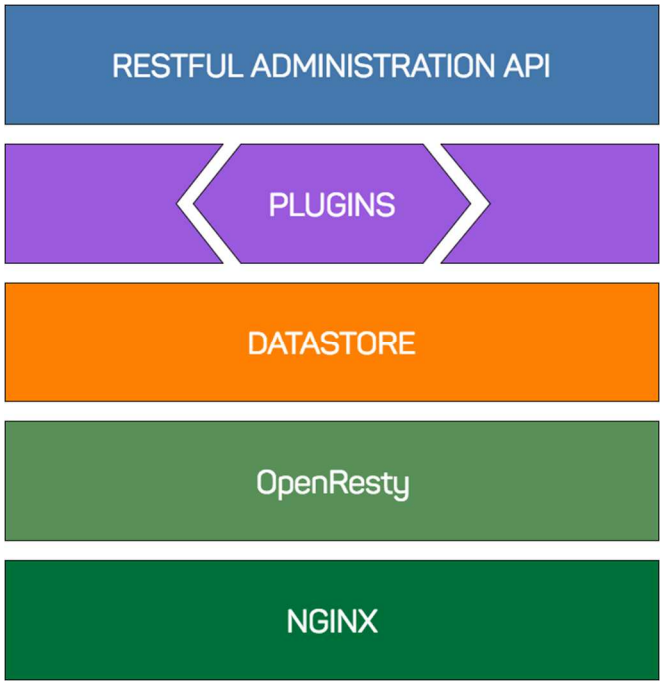


**Figure 5 - Kong architecture**[4]

Kong is built on a pluggable architecture which allows to extend its capabilities by means of **plugins** based on the Lua scripting language: plugins can be injected anywhere into the request lifecycle. Kong core encompasses database abstraction, routing and plugin management, while any other

[4] https://blog.codecentric.de/files/2017/10/Kong-Architecture-250x258.png (retrieved 11/09/2018)

feature is implemented through a specific plugin.

One way to deploy Kong is in a centralized pattern fashion, as shown here:

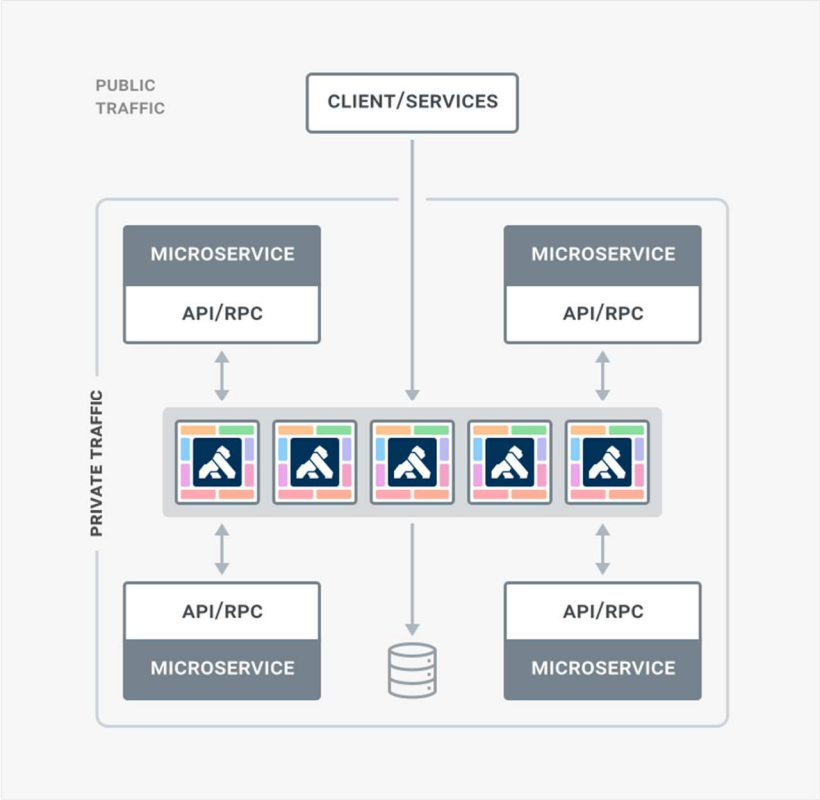


**Figure 6 - API Gateway Pattern**[5]

In this way, Kong can act as an **API gateway**, essentially becoming the single entry point for a group of services running on the platform.

In addition to the basic API gateway features, the available Kong plugins cover a broad range of features such as monitoring, authentication, logging, ACL, rate-limiting, caching and analytics.

## 4.3. Kong Service configuration

As an example, a new service configuration will be shown. The service will be configured by using the Kong Admin API. The configuration is shown below:

As a prerequisite, suppose Kong and an EasyTV service are running in two different Docker containers connected to the same Docker bridge network. The important thing to note is that on a user-defined bridge network, containers can resolve each other by name or alias, so if you call your containers *kong* and *easytvserviceA*, the *kong* container can connect to the EasyTV service container at *easytvserviceA*.

### 4.3.1. Add a Service

The following *curl* (but it is also possible to use another tool such as *wget*) command is used to add a service to Kong (note that the command is used outside any container and the Admin API port

[5] https://konghq.com/ (retrieved 11/09/2018)

8001 inside the Kong container is mapped to the 8001 port on the host system):

```
$ curl -i -X POST \
  --url http://localhost:8001/services/ \
  --data 'name=easytv-service-A' \
  --data 'url=http://easytvserviceA '
```

The response received should be something similar to:

```
HTTP/1.1 201 Created
Content-Type: application/json
Connection: keep-alive

{
   "host":"easytvserviceA",
   "created_at":1519130509,
   "connect_timeout":60000,
   "id":"92856672-f5ea-4e9a-b196-667bf55ba40c",
   "protocol":"http",
   "name":"easytv-service-A",
   "read_timeout":60000,
   "port":80,
   "path":null,
   "updated_at":2019130509,
   "retries":5,
   "write_timeout":60000
}
```

### 4.3.2. **Add a route for a Service**

The following *curl* command is issued to add a Kong route for the previously configured service:

```
$ curl -i -X POST \
  --url http://localhost:8001/services/easytv-service-A/routes \
  --data 'paths[]=/easytvserviceA'
```

The response received should be something similar to:

```
HTTP/1.1 201 Created
Content-Type: application/json
Connection: keep-alive

{
```

```
  "created_at":2019131139,
  "strip_path":true,
  "hosts": null
  "preserve_host":false,
  "regex_priority":0,
  "updated_at":2019131139,
  "paths": [
    "\/easytvserviceA"
  ],
  "service":{
    "id":"92856672-f5ea-4e9a-b196-667bf55ba40c "
  },
  "methods":null,
  "protocols":[
    "http",
    "https"
  ],
  "id":"f9de2ed6-c06e-4e16-bd5d-3a82daef3f9e"
}
```


In this way an URL path match has been added, thus allowing Kong to route the incoming request accordingly. Port 8000 is used by Kong for proxying and it listens on it for HTTP traffic, therefore any request to http://localhost:8000/easytvserviceA will be forwarded to the *easytvserviceA* container.

More specifically, suppose that the EasyTV service *easytvservice1* exposes three API endpoints. Kong will forward the requests as follows:

- http://localhost:8000/easytvserviceA/endpoint1 -> http://easytvserviceA/endpoint1
- http://localhost:8000/easytvserviceA/endpoint2 -> http://easytvserviceA/endpoint2
- http://localhost:8000/easytvserviceA/endpoint3 -> http://easytvserviceA/endpoint3

# 5. API DESIGN

## 5.1. Introduction to API design

As it was abovementioned, the EasyTV service modules will expose their functionality through different APIs with the aim that other modules and client applications are able to interact with them.

In order to make this interaction easier, an effective API design within EasyTV project is expected. In fact, providing an effective interface will help to understand, use and integrate the APIs, as well as facilitating their maintenance during the project and beyond. In this regard, a good design will increase their usability, which includes aspects such as:

- Helping in better implementation: an effective API design should provide a comprehensive overview on what the API does and in which way, so its use should be easier and prevent complicated configurations.
- Allowing incremental development: considering that API development is a continuous process, a good design will help evolution by preventing confusion and errors.
- Easing better documentation: the better an API is designed, the easier it is to be well documented, allowing self-optimized processes.
- Enhancing developer experience: this is vital for assuring the use of each API, since a good design will help to integrate the APIs into the platform.

Taking the advantages provided by a good API design into account, a special effort will be done along the project in order to achieve it. In this regard, and with the aim of providing a starting point, let's say that some of the most important principles to be considered when designing a RESTful API for the EasyTV platform should be:

| Principle | Comments |
|---|---|
| **Keep it simple** | In order to facilitate its use and understanding, the API URLs should be simple |
| **Use nouns instead of verbs** | Since the action is already described with the HTTP method used |
| **Use the correct HTTP method for each action** | GET: for getting a resource or collection of resource<br>POST: for creating a resource or collection of resources<br>PUT/PATCH: for updating an existing resource or collection of resources<br>DELETE: for removing an existing resource or collection of resources |
| **Use parameters** | In order to avoid using long URLs and keep the simplicity, some type of information must be included as parameters, not as new methods |
| **Use proper HTTP codes** | To provide a more complex information about the response of the system |
| **Use proper error messages** | To include enough information about the error to understand what happened. |

**Table 2. Principles when designing a RESTful API**

On the other hand, the correct API design explained before comprises three different aspects that are often used interchangeably but should be explained in depth in order to understand their differences:

- API documentation: it is the reference manual for an API, and it explains how it can be used, providing examples and the constraints that the API allows (APIs requests, error messages, etc.). The information provided can be presented into three different ways: as top level information (including a simple guide with some examples in different forms), as functional understanding (which removes the abstraction from the rest of the documentation for a meaningful understanding) or as technical understanding (it represents a reference document, not a functional catalogue).
- API specification: it is usually related to the explanation of the overall behaviour of the API and how it can be linked to other solutions. It also includes its general design philosophy and supported types.
- API definition: it defines the backbone, organization and function of each API at a base-machine readable level.

These three levels should be considered during the design of each API for the EasyTV project in order to guarantee correct solutions that are easy to use and understand.

## 5.2. Solutions for API documentation

The API design process comprises different tasks that have to be overtaken in detail. To facilitate this process there are several documentation generators that could help with the conversion from the definition into structured and easy to use API documentation for developers. This section is in charge of presenting the most important open source ones according to the three main different specifications, with the aim of having enough information for selecting the most adequate for the EasyTV environment.

### 5.2.1. API documentation generators using Swagger/OpenAPI[6] specification

The OpenAPI specification is a powerful bottom-up definition format for describing RESTful APIs and for mapping all the resources and operations related to the RESTful interface in order to facilitate its development and consumption. It provides a fast setup and a large support community.

| Pros | Cons |
|---|---|
| - Mature solution: a large community and support base.<br>- High adoption rate<br>- Strong framework support<br>- Highest language support<br>- Can execute API calls from the documentation<br>- JSON with YAML compatibility | - Requires multiple specifications for some tools.<br>- The code can't be reused.<br>- Requires schemas for all responses |

**Table 3. Swagger/OpenAPI specification Pros&Cons analysis**

The main solutions based on this specification are mentioned hereinafter:

- Swagger[7]: it is a complete framework for describing, producing, consuming and visualizing RESTful web services. The API documentation can be dynamically created by means of JSONs files and its own interface, which is able to also be displayed it in a well-structured

[6] https://www.openapis.org/

[7] https://swagger.io/

manner. It is free to use and licensed under the Apache 2.0 License, and many open source projects as well as commercial vendors provide different Swagger integrations since it is today's leading API ecosystem with the best documentation and support.
- DapperDox[8]: this solution helps with the process of creating readable guides for APIs documentation, allowing the inclusion of relevant documentation into the rendered specification page.
- ReDoc[9]: by using OpenAPI specification, ReDoc is able to generate responsive sites and support deep linking. The code examples can be included by making use of a third party extension.

### 5.2.2. API documentation generators using RAML specification

RAML (RESTful API Modeling Language)[10] is built on standards such as YAML and JSON and helps with the whole API lifecycle with a language agnostic solution, since it includes tools for Java, PHP, Python, Ruby, etc. It excels at supporting the entire API's lifecycle with a top-down specification. Its main advantages and disadvantages are shown below:

| Pros | Cons |
|---|---|
| - Single specification to maintain<br>- Strong and visual-based IDE<br>- Allows for design patterns<br>- Easy to get started<br>- Human readable<br>- Unit testing | - Lacks strong documentation and tutorials<br>- Limited code reuse<br>- Multiple specifications required for several tools.<br>- Poor tooling support for newer versions<br>- Only YAML |

**Table 4. RAML specification Pros&Cons analysis**

Two main solutions are developed on the basis of this specification:

- RAML 2 HTML[11]: it is a Node js solution for HTML documentation generation.
- RAML API Console[12]: it allows to create HTML documentation from a RAML specification, as well as browsing of API documentation and testing of API methods.

### 5.2.3. API documentation generators using API blueprint specification

API Blueprint is based on Markdown format and it is used for both writing API descriptions and documentation. It allows designing, documenting and testing of already deployed APIS from a top-down point of view.

| Pros | Cons |
|---|---|
| - Easy to use<br>- Ecosystem of tools<br>- Markdown makes the file easy to read regardless of technical ability | - Slow adoption due to the lack of advanced construct and code level tooling.<br>- It is specially focused on C++ through Node-js and C# |

[8] http://dapperdox.io/

[9] https://github.com/Rebilly/ReDoc

[10] https://raml.org/

[11] https://github.com/raml2html/raml2html

[12] https://raml.org/blogs/raml-console-20

**Table 5. Blueprint specification Pros&Cons analysis**

The main solutions based on this specification are:

- Snowboard[13]: it is a parser and renderer tool which can be customizable.
- Aglio[14]: it renders HTML from API Blueprint files, with different colours, themes and templates.

## 5.3. EasyTV APIs design

According to the specification generator tools analysis from the previous sections, there are several options that could be used in the project for API design. Nevertheless, the decision about which one to select is based upon two main aspects: on one hand, we need to select a mature and wide adopted solution in order to guarantee a good market acceptation and, on the other, the use of JSON is needed for easy integration. For that reason, the tool selected is Swagger. Along the following sections we are going to present an example of a specific API design for EasyTV with it.

### 5.3.1. EasyTV User API

In order to facilitate user management for the companion screen app within the platform, an API has been created. Its main methods are explained in the following sections.

#### *5.3.1.1.* POST/login

This is a user login request within the EasyTV platform.

The input is a JSON with the authentication data, which includes user name and password.

There could be three different responses:

- Code 200: access granted.
- Code 403: access denied.
- Code 500: server error

The output is a JSON that contains the following data:

| Output | Type | Description |
|---|---|---|
| Message | String | Success or error message |
| statusCode | Int | Http response code |
| Token | String | Authentication key (for access granted) |
| Role | String | User role (for access granted) |
| Expires | Date | Expiration date (for access granted) |

**Table 6. POST/login output response data**

An example for each kind of response can be seen below:

| 200 Success | 403 Error | 500 Error |
|---|---|---|
| {<br>"message": "Success",<br>"statusCode": 200, | { | {<br>"message": "Error: Server internal error", |

[13] https://github.com/bukalapak/snowboard

[14] https://github.com/danielgtaylor/aglio

| | | |
|---|---|---|
| "token": "5a68a7f27f4f6d120881b6e7", "role": "admin", "expires": DD/MM/YYYY } | "message": "Error:resource not found", "statusCode": "403", } | "statusCode": "500", } |

**Table 7. Examples of POST/login response**

#### 5.3.1.2. GET/users

This is a request for retrieving users from EasyTV platform. It does not contain any input data. An authentication key (token) should be passed in the request header to verify that the caller is authorized to get this information.

There could be three different responses:

- Code 200: success.
- Code 403: access denied.
- Code 500: server error

The output is a JSON that contains the following data:

| Output | Type | Description |
|---|---|---|
| Message | String | Success or error message |
| statusCode | Int | Http response code |
| User | Array | Array with users' information such as name, id, email, etc. |

**Table 8. GET/users output response data**

An example for each kind of response can be seen below:

| 200 Success | 403 Error | 500 Error |
|---|---|---|
| { "message": "success", "statusCode": 200, "users": [ { "_id": "string", "user": "name", "email": "email", "enabled": false, "role": "user", "created": "2016-06-14T10:24:11.387Z", "updated": "2016-06-14T10:24:11.387Z" } ] } | { "message": "Error:resource not found", "statusCode": "403", } | { "message": "Error: Server internal error", "statusCode": "500", } |

**Table 9. Examples of GET/users response**

#### 5.3.1.3. GET/emails

This is a request for retrieving users' mails from EasyTV platform. It does not contain any input data. An authentication key (token) should be passed in the request header to verify that the caller is authorized to get this information.

There could be three different responses:

- Code 200: success.
- Code 403: access denied.
- Code 500: server error

The output is a JSON that contains the following data:

| Output | Type | Description |
|---|---|---|
| Message | String | Success or error message |
| statusCode | Int | Http response code |
| User | Array | Array with users' mails. |

**Table 10. GET/emails output response data**

An example for each kind of response can be seen below:

| 200 Success | 403 Error | 500 Error |
|---|---|---|
| { "message": "success", "statusCode": 200, "users": [ { "user": "user", "email": "email@host.es" } ] } | { "message": "Error:resource not found", "statusCode": "403", } | { "message": "Error: Server internal error", "statusCode": "500", } |

**Table 11. Examples of GET/emails response**

#### 5.3.1.4. POST/user

This is a request for creating new users within EasyTV platform, by means of a JSON which includes the user's info: id (string), user (string), email (string), password (string), repeat_password (string). An authentication key (token) should be passed in the request header to verify that the caller is authorized to execute this operation.

There could be three different responses:

- Code 200: user created.
- Code 500: server error.
- Code 400: client error (e.g. user already exists, password and repeat_password don't match).

The output is a JSON that contains the following data:

| Output | Type | Description |
|---|---|---|
| Message | String | Success or error message |

| | | |
|---|---|---|
| statusCode | Int | Http response code |

**Table 12. POST/user output response data**

#### 5.3.1.5. DELETE/user/{id}

This is a request for removing any user from the EasyTV platform by making use of his/her id, which is the input parameter. An authentication key (token) should be passed in the request header to verify that the caller is authorized to execute this operation.

There could be three different responses:

- Code 204: removed successfully.
- Code 404: user not found in the database
- Code 500: server error.

The output is a JSON that contains the following data:

| Output | Type | Description |
|---|---|---|
| Message | String | Success or error message |
| statusCode | Int | Http response code |

**Table 13. DELETE/user/{id} output response data**

#### 5.3.1.6. POST/user/forgotpassword

This is a request for retrieving the user's password from the database by making use of his/her email address, which is the input parameter.

There could be three different responses:

- Code 200: success.
- Code 404: user not found in the database
- Code 500: server error.

The output is a JSON that contains the following data:

| Output | Type | Description |
|---|---|---|
| Message | String | Success or error message |
| statusCode | Int | Http response code |

**Table 14. POST/user/forgotpassword output response data**

#### 5.3.1.7. POST/user/resetpassword/{token}

This is a request for resetting the user's password from the database by making use of an authentication token, which is the input parameter.

There could be three different responses:

- Code 200: success.
- Code 404: user not found in the database
- Code 500: server error.

The output is a JSON that contains the following data:

| Output | Type | Description |
|---|---|---|
| Message | String | Success or error message |
| statusCode | Int | Http response code |

**Table 15. POST/user/resetpassword/{token} output response data**

#### *5.3.1.8.* POST/user/changepassword

This is a request for changing the user's password from the database by making use of an JSON which the following fields: "old_password", "password", "repeat_password" as the input data.

There could be three different responses:

- Code 200: success.
- Code 404: user not found in the database
- Code 500: server error
- Code 400: client error (e.g. old password is wrong, password and repeat_password don't match)

The output is a JSON that contains the following data:

| Output | Type | Description |
|---|---|---|
| Message | String | Success or error message |
| statusCode | Int | Http response code |

**Table 16. POST/user/changepassword output response data**

# 6. CONCLUSIONS AND FUTURE WORK

This document presented strategies and technical choices on which the EasyTV multi terminal technical platform will be based. In addition to the description of the adopted technologies, configuration examples have been presented, along with a comprehensive list of the EasyTV modules that will be containerized. Two chapters were devoted to API management and API design, respectively. The setup and implementation of the EasyTV multi terminal technical platform follows the evolution of the architecture from D1.3 (“First release of the EasyTV system architecture”) to D.1.4 (“Final release of the EasyTV system architecture”).

The next steps involve the actual deployment of the whole EasyTV platform solution, along with the EasyTV modules that will be subsequently integrated and tested. Platform components will be fine-tuned to meet requirements and to solve any issue that will arise in the process. Particular attention will be given to proper API design, which will be carefully taken into consideration by each partner, so to avoid any inconsistencies between multiple EasyTV module interfaces.

Final results will be presented in D5.7 (“Final report on the set up and implementation of the EasyTV multi terminal technical platform”).

# 7. REFERENCES


[1] Gartner. [Online]. https://www.gartner.com/it-glossary/it-infrastructure (last accessed 18 Oct 2018)

[2] IBM. [Online]. https://www.ibm.com/cloud/learn/benefits-of-cloud-computing (last accessed 18 Oct 2018)

[3] Docker. [Online]. https://docs.docker.com/get-started/#docker-concepts (last accessed 18 Oct 2018)

[4] Docker. [Online]. https://docs.docker.com/engine/docker-overview/#docker-engine (last accessed 18 Oct 2018)

[5] Linux Man Pages. [Online]. http://man7.org/linux/man-pages/man7/cgroups.7.html (last accessed 18 Oct 2018)

[6] Linux Man Pages. [Online]. http://man7.org/linux/man-pages/man7/namespaces.7.html (last accessed 18 Oct 2018)

[7] UnionFS. [Online]. http://unionfs.filesystems.org/ (last accessed 18 Oct 2018)

[8] Docker. [Online]. https://blog.docker.com/2016/12/understanding-docker-networking-drivers-use-cases/

[9] Docker. [Online]. https://docs.docker.com/network/#network-drivers (last accessed 18 Oct 2018)

[10] Docker. [Online]. https://docs.docker.com/compose/overview/ (last accessed 18 Oct 2018)

[11] Github. [Online]. https://github.com/docker/swarmkit (last accessed 18 Oct 2018)

[12] Docker. [Online]. https://docs.docker.com/engine/swarm/key-concepts/#what-is-a-swarm (last accessed 18 Oct 2018)

[13] Docker. [Online]. https://docs.docker.com/get-started/part5/#introduction (last accessed 18 Oct 2018)

[14] Portainer. [Online]. https://portainer.io (last accessed 18 Oct 2018)

[15] Portainer. [Online]. https://portainer.io/overview.html (last accessed 18 Oct 2018)

[16] Kong. [Online]. https://docs.konghq.com/0.14.x/getting-started/introduction/ (last accessed 18 Oct 2018)