Deliverable

Project Acronym:	**ImmersiaTV**
Grant Agreement number:	688619
Project Title:	*Immersive Experiences around TV, an integrated toolset for the production and distribution of immersive and interactive content across devices.*

D3.5 Distribution and Reception

Revision: 0.6

Authors:

David Gomez (i2CAT), Einar Meyerson (i2CAT)

Delivery date: M25

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement 688619

Dissemination Level

P	Public	x
C	Confidential, only for members of the consortium and the Commission Services	

Abstract: This deliverable describes the hardware and software components inside the distribution and reception pipeline of media content delivered for Pilot 1.

REVISION HISTORY

Revision	Date	Author	Organisation	Description
0.1	09/01/2017	David Gomez, Einar Meyerson	i2CAT	Software description
0.2	11/01/2017	Joan Llobera	i2CAT	Document review
0.3	10/10/2017	Alexandr Kelembet	CGY	Live production tools section is added
0.4	20/10/2017	Maciej Glowiak, Szymon Malewski	PSNC	Final formatting
0.5	22/12/2017	Juan Antonio Núñez	i2CAT	Document review
0.6	24/01/2018	Szymon Malewski	PSNC	Formatting

CONTRIBUTORS

First Name	Last Name	Company	e-Mail
Joan	Llobera	i2CAT	Joan.llobera@i2cat.net
Alexandr	Kelembet	CGY	kelembet@cinegy.com
David	Gómez	i2CAT	David.gomez@i2cat.net
Einar	Meyerson	i2CAT	Einar.meyerson@i2cat.net
Maciej	Glowiak	PSNC	mac@man.poznan.pl
Szymon	Malewski	PSNC	szymonm@man.poznan.pl

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF ACRONYMS

Acronym	Description
API	Application Programming Interface
BW	Bandwidth
DDoS	Distributed denial-of-service
GUB	GStreamer - Unity Bridge
HMD	Head Mounted Display
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MPD	Media Presentation Descriptor
MPEG-DASH	Moving Picture Expert Group - Dynamic Adaptive Streaming over HTTP
REST	Representation State Transfer
URI	Uniform Resource Identifier
VOD	Video on Demand
XML	Extensible Markup Language

1. INTRODUCTION

ImmersiaTV offers a whole distribution and reception pipeline of MPEG-DASH (Moving Picture Expert Group - Dynamic Adaptive Streaming over HTTP) content. Dynamic Adaptive Streaming over HTTP (DASH) [1] specifies a XML (MPD) and binary formats that enable delivery of media content from standard HTTP servers to HTTP clients.

The offline production tools pipeline is implemented as a set of modules linked and built inside a virtualization tool (Docker), with a web application to set up, transcode, publish and distribute the MPEG-DASH content through an API REST[1].

The online production tools pipeline is implemented as a set of native Windows OS applications providing user interface and configuration options to set up, transcode and publish the MPEG-DASH content.

Last but not least a MPEG-DASH client has been implement in order to make the reception and visualization of MPEG-DASH content in VOD (Video on Demand).

[1] Application program interface that uses HTTP requests to GET, PUT, POST and DELETE data.

2. ARCHITECTURE AND FUNCTIONAL OVERVIEW

2.1. Offline publication and distribution

The publication and distribution component is a web application running in a Linux server using the Docker virtualization tool that is able to:

- List the content exported from the Premiere plugin and the content converted to MPEG-DASH
- Handle the MPEG-DASH transcoding of Premiere content
- Give user info about the transcode status of the Premiere content (Ready, Done, Processing, Exporting, Error)
- Publish MPEG-DASH content in order to view it through the multi-platform players.

The architecture of the web application server is divided into four modules. Each module is built as a Docker image and runs in a unique Docker container. The modules are: Nginx web server, web application, database and MPEG-DASH transcoder see Figure 1.

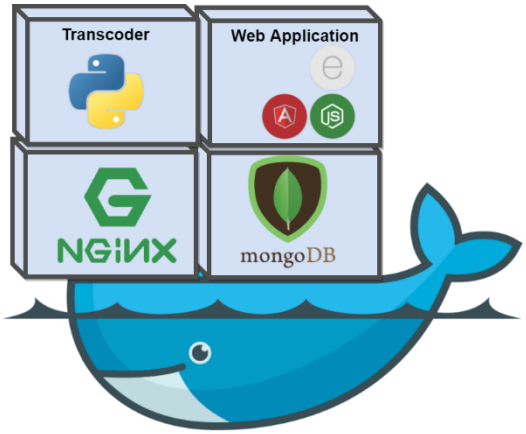

Figure 1 Modules of the Publication and distribution server

2.1.1. Web Application

The web application offers the possibility of performing a series of actions listed above in a visual and more user friendly way. The module is divided into 2 different sub-modules connected together: Client and Server.

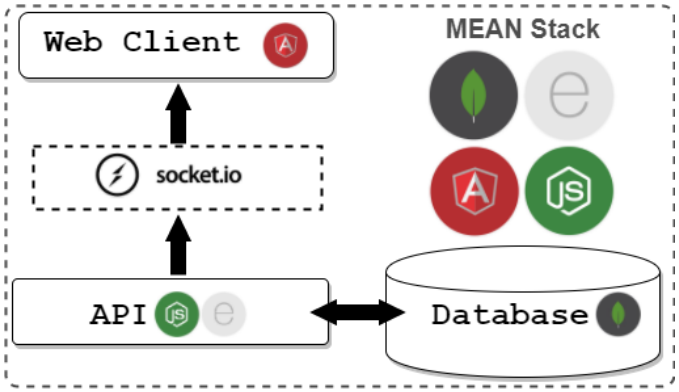

Figure 2 Web application module structure and connection

All together with the database module they constitute the **MEAN** stack (**M**ongoDB database system, **E**xpress.js back-end web framework, **A**ngular.js front-end web application framework and **N**ode.js back-end runtime environment). This module connects with all the other modules.

2.1.1.1. Server

The server sub-module is developed with NodeJS[2] together with Express.js. An API REST service is configured in order to create, read, update, and delete (CRUD functions) content and transcoding information entries in the database. The different datasets will be explained later in this document in section 1.1.2. The database. The server is also in charge of making sure that everything is working correctly during all the different process and in restart mode make sure the database is coherent with the content located in the folders in case someone uploaded content when turned off. Everything that can be done through the web client can also be execute through the API REST using software applications like e.g. Postman. Any action that produces changes in the database needs to be notified to the client in order to update the web page view dynamically. Socket.io[3] is used in order to accomplish this.

The server is prepared for deployment in any OS and Docker. By changing the variables in the .env file the server will work in any environment. There is also available a light version of the server which only has the option of publishing content (The transcoding option will be blocked for users using the light version for security purposes) and it is also configured in the .env file. The .env file gives the server the option to set a custom configuration according to the user's need.

2.1.1.2. Client

The client sub-module is developed with AngularJS[4]. AngularJS is responsible of the web application front-end. In conjunction with Bootstrap[5] it gives the web application a client-side visual interface. Input content, exported from Adobe Premiere, is shown in the "Transcode" sub-menu. Output content, once transcoded to MPEG-DASH format, is shown in the "Publish" sub-menu (See figure 3).

[2] Open-source, cross-platform JavaScript runtime environment for developing a diverse variety of tools and applications. https://nodejs.org/en/

[3] Socket.io

[4] JavaScript-based open-source front-end web application framework. https://angularjs.org

[5] Bootstrap is a free and open-source front-end web framework for designing websites and web applications. https://getbootstrap.com/

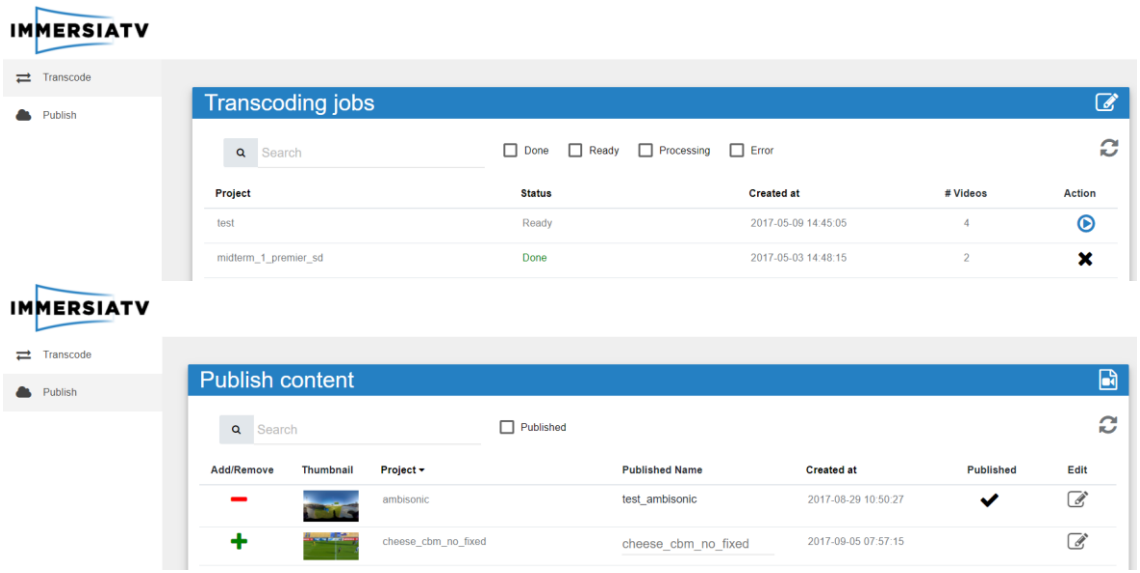

Figure 3 The distribution and publication web application view. Transcode to MPEG-DASH section on top, publication content below.

Despite there is no option for dynamic content upload. Users can export their content from Premiere to the configured input folder and refresh the input list on the web page by clicking the refresh button on 'Transcode' panel. In order to transcode, content needs to be in Ready status (content can also be transcoded in Error status if the error has been solved). Previous to the transcoding process users are able to select the transcoding parameters before transcoding through a menu (see Figure 2) by clicking the icon at the top right corner of the panel. Segment size, aspect ratio of the video, resolution and bitrate are the different configuration options users have. The configuration accepts having 2 representations in the same video type with same resolution but different bitrates. In the case of the omnidirectional videos we created a new projection type in addition to the EQR (Equirectangular projection). This new projection uses a Cube instead of a Sphere for the projection Mesh.

The media templates set by default it see in the Table 1:

Profile	Television	Spherical	Portal
High	4096x2160	4096x2160	426x240
Mid	2048x1080	2048x1080	426x240
Low	1920x1080	2048x1080	426x240

Table 1 Media Templates

The fragment length has a range from 1 to 15 seconds being the optimal values 1, 3, 5, 10 and 15. The segment length is global between all the video types, Television, 360-degree video and Portal will have the same segment length.

Users have no obligation on configuring the transcoding parameters every time they want to transcode a project as there is a default configuration shown in Figure 4. Once the configuration is done, by clicking the apply button users will be redirected back to the 'Transcode' menu where the MPEG-DASH transcoding is done by pressing the 'Play' button under the Actions column.

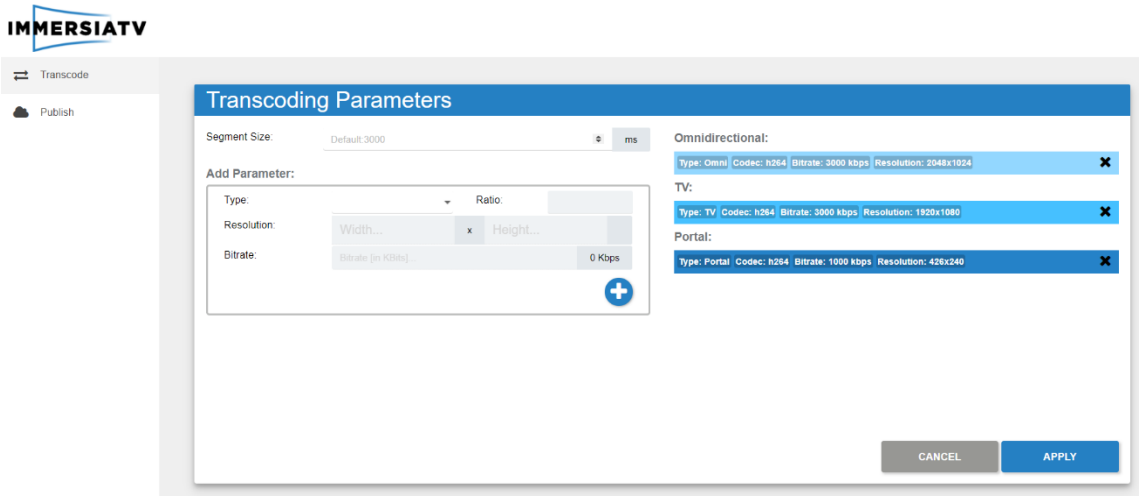

Figure 4 The Transcoding Parameters menu where the user can select which resolutions transcode for each type of video (Omnidirectional, TV and Portal)

This action will activate the transcoder container via the API REST, and generate the MPEG-DASH output content according to the previews configuration. The transcoding process is viewed through the status column. A process bar will appear giving the users the feedback on how the transcoding process is going on, see Figure 5.

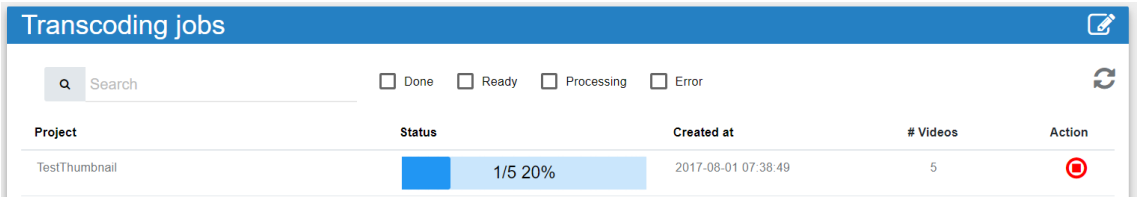

Figure 5 Process bar appears when a transcoding process is executed for user information.

By clicking the red stop button below the Action column, users have the option to stop the transcoding process. Stopping the transcoding process will remove all the created files until that instance. Users will have to transcode the content from scratch. Once the transcoding job has ended the new DASH content will appear in the publish view. By clicking on the 'Done' content, users will navigate automatically to the 'Publish' panel, and the new content will appear in first position for fast accessibility. Before publishing, a new name can be introduced below the 'Published name' column. This will be the name that will appear in the published list and later in the players. The default name introduced is the same as project XML file. Content can be published by clicking the green plus button and removed from the published list clicking the red minus button, see Figure 1. Published content is marked with a tick under the 'Published' column. The information of the published content (name, id, preview, thumbnail and url) is saved in a JSON file inside the NGINX web server module. Later in this document all the remaining functionalities will be detailed.

2.1.2. The database

The database module uses the official MongoDB docker image. MongoDB is an open-source cross-platform document-oriented database program. It is classified as a NoSQL[6] database program that uses JSON-like documents with schemas. The web app server uses the mongoose

[6] A NoSQL database provides a mechanism for storage and retrieval of data that is modelled in other ways than the tabular relations used in relational databases.

package from the NodeJS package manager (npm) in order to communicate with the MongoDB docker container. The database module is configured in a persist mode, this means that the information is not lost although the server is restarted. This is done by configuring a docker volume[7] folder from the MongoDB docker container to the local filesystem.

In this section of the document the different data models used and its schemas will be detailed. The two principal models are the input content and output content along with the transcoder, job and job update models. Firstly, the input content model has the following schema structure it see in the Table 2:

Key	Description
input_id {String}	This id identifies the input content and it is also used in order to connect with its associated output content in future transcoding processes.
xml_path {String}	Path for the XML that describes the whole content structure, also need for transcoding purposes.
xml_name {String}	XML name.
parent_folder {String}	The content folder name. This is used in order to detect duplicates.
Status {String}	Information on the transcoding status: Ready: Content ready for transcoding. Processing: Content in a transcoding process. Done: Content successfully transcoded. Error: Transcoding process failed.
error_message* {String}	If the transcoding process fails, Users can hover over the status 'Error' in order to view the error message.
last_modified {String}	The last time the content was modified.
mediaFiles {Number}	Number of media files in the content that can be transcoded.
Process* {String}	Indicates the stage in the transcoding process.

Table 2 Input content data model structure description

*Not mandatory in order to add a new entry to the BBDD.

Secondly, the output model has the following schema structure it see in the Table 3.

Key	Description
outputId {String}	This id identifies the output content and it is also used in order to connect with extra info on the transcoding job model.
inputContentId {String}	This identifier corresponds to the Input content associated.
xml_path {String}	XML path.
xml_name {String}	XML name.

7

thumbnail_path* {String}	Path for the thumbnail image used in the web server and web player.
preview_path* {String}	Path for the mini clip reproduced in the web page when navigating to the output content info. (See figure X.)
hasParentInput {Boolean}	Boolean that describes if the content has a parent input content. If 'False' no content editing can be executed due to missing Input files.
parent_folder {String}	Output content folder name
last_modified {String}	The last time the content was modified. This time also gives information of the last time this content was transcoded/edited.
Published {Boolean}	Boolean that describes if the content has been published or not.
published_name** {String}	The name introduced in the web server to be shown in the different players.
published_url** {String}	The path to the XML file consumed by the different players. This XML also contains the paths of the different MPDs of the transcoded media files allocated in the NGINX server.

Table 3 Output content data model structure description

**Mandatory when output content is published.

Last but not least we have data models related to the transcoder. To transcode a new project, the MPEG-DASH transcoder module needs specific information in order to execute correctly the transcoding process. Since you cannot send the whole data structure via command line in a clean way, it was agreed to add this information as a new data model although this means the MPEG-DASH transcoder module has to send a petition via the API REST. The transcoder model has the following schema structure it see in the Table 4, 5:

Key	Description
maxSimultaneous {Number}	This value limits the maximum simultaneous transcoding processes that can be executed in the server.
running {Number}	Number of running process been executed. This value has to be lower or equal to the maxSimultaneous in order to execute a new transcoding process.
Dasher_JSON {Object}	Parent key that wraps the Dasher, MediaInfo and MediaResolution keys.
Dasher **{Object}**	Parent key for the child key 'Version'. The version key is used to query the desired transcoder information in case there are several versions.
MediaInfo **{Object}**	Parent key for the child keys 'NumberOfVideos' and 'SegmentLength'. NumberOfVideos is the number of media files that will have to be transcoded. SegmentLength (in ms) is the length of the MPEG-DASH fragments.

MediaResolutions **{Object}**	Parent key for the keys 'tv', 'spherical' and 'portal'. Each of these keys are objects that contain information of the 'Resolutions', 'Bitrate' and 'SourceName' to use in the transcoding job.

Table 4 Transcoder data model structure description

Once the transcoding process has ended, the information of the transcoding job is saved in the BBDD using the transcoder job data model. The transcoder job model has the following schema structure:

Key	Description
job_id {Number}	This id identifies the transcoding job. This Id is used for future editing purposes.
outputContentId {Number}	This identifier corresponds to the Output content associated.
MediaInfo {Object}	Same as the previous transcoder schema.
MediaResolutions {Object}	Same as the previous transcoder schema.

Table 5 Transcoder job data model structure description

The transcoding job information of each output content can be visualized in the web page by clicking on top of any element in the 'Publish content' panel. A new view will appear (See Figure 6) with the representation information along with a short preview clip of the media content.

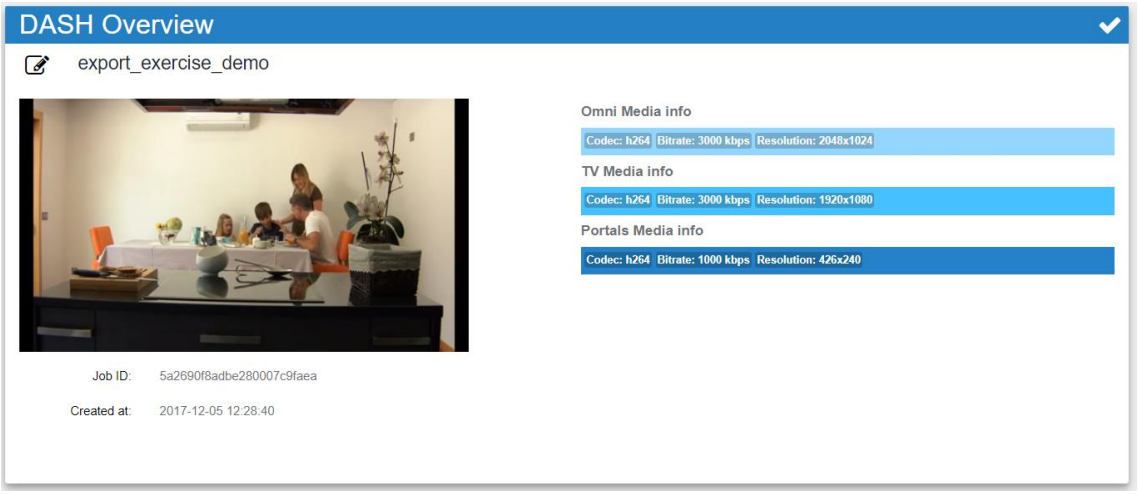

Figure 6 DASH Overview panel. This panel shows the transcoding information of the selected output content.

Both in this view 'DASH Overview' and in 'Publish content' the user can edit the content by adding or removing representations. These 2 processes will be described later in section 2.1.4.2. Add and Remove Qualities. In order to edit a content, the MPEG-DASH module, like in the transcoding process, needs the specific information of the new qualities to add or the ones to remove. This information is added in a new database modal and the module has to send a petition in order to get this information.

The transcoder job update model has the following schema structure:

Key	Description

jobContentId {String}	This identifier corresponds to the transcoding job associated.
AddResolutions* {Object}	New representations to be added will be listed in this object.
RemoveResolutions* {Object}	Old representations to be removed will be listed in this object.

Table 6 Transcoder job update model structure description

*Both objects have the same schema as 'MediaResolutions'. Adding and removing can be done in the same update process, on the contrary, only the object associated with the action will be necessary.These are the different models and schemas all the data is organized in. MongoDB was chosen due to the good synergy it has with NodeJS and Express.js. Furthermore, this project is evolving and due to MongoDB is a NoSQL database system the database models and schemas can be upgraded without any big issues.

2.1.3. Nginx web server

The Nginx web server container connects the web application and the multi-platform players distributing the content. To provide a more robust infrastructure the content consumed by the players is delivered through the Nginx in a different port of the web application. This secures media distribution regardless of the web application's state and therefore makes it more robust, particularly to the common DDoS attacks. The output content folder is also added as a volume in the configuration of the nginx docker container. Although the 4 different modules usually run in the same host, the nginx web server can be configured in a separate host by editing the variables in the *.env* file.

2.1.4. MPEG-DASH transcoder.

The MPEG-DASH content through an API REST. This API it can be used regardless of the Web App, due to implements a CLI input. The API implements a third party's framework orchestrated by the Main Script. This script is developed in Python language don't needs compilation, the execution of the program is in Realtime and exist extended community users.

- FFmpeg[8] Encoding Opensource Framework.
- JVET360Lib[9] Encoding and projection transformation Opensource Framework..
- GPAC[10] Encoding Opensource Framework, includes the MPEG-DASH gegenerator

2.1.4.1. Create Content.

The transcoder gets the encoding parameters of the Web Application (Resolutions, Codec type, Bitrate, Framerate, Bandwidth and Segment length) and the media source to the XML Premiere file. Once obtain all the parameters, the API create the folder structure and generate all the resolutions needed using the parameters obtained, once finished this process, the API generate the Equirectangular projection and the Cubemap projection it sees

[8] https://ffmpeg.org
[9] https://jvet.hhi.fraunhofer.de/
[10] https://gpac.wp.imt.fr/

an example of this projections in the Figure 7. When the encoding process is completed, the media source is fragmented and create the MPEG-DASH content. Finally, the temporal files generated are removed. Also creates a short preview of the content (Tv mode) and a thumbnail, this media is needed to the content serve to present a preview of the media. In addition all the media properties of the media (name, type projection, resolution, bitrate, codec, etc) are saved in a Json files. The complete ouput structure of the MPEG-DASH transcoder it see in the Figure 8

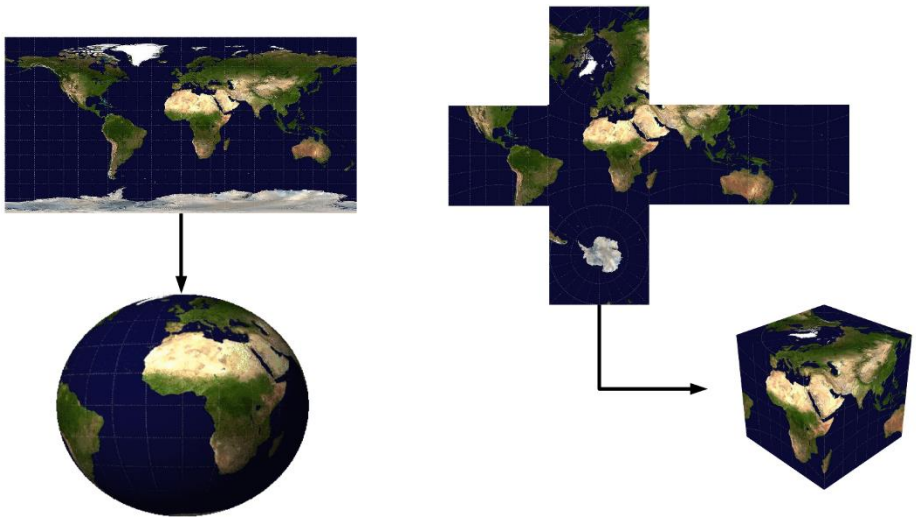

Figure 7 Equirectangular Projection (left) and the Cube Map Projection (right)

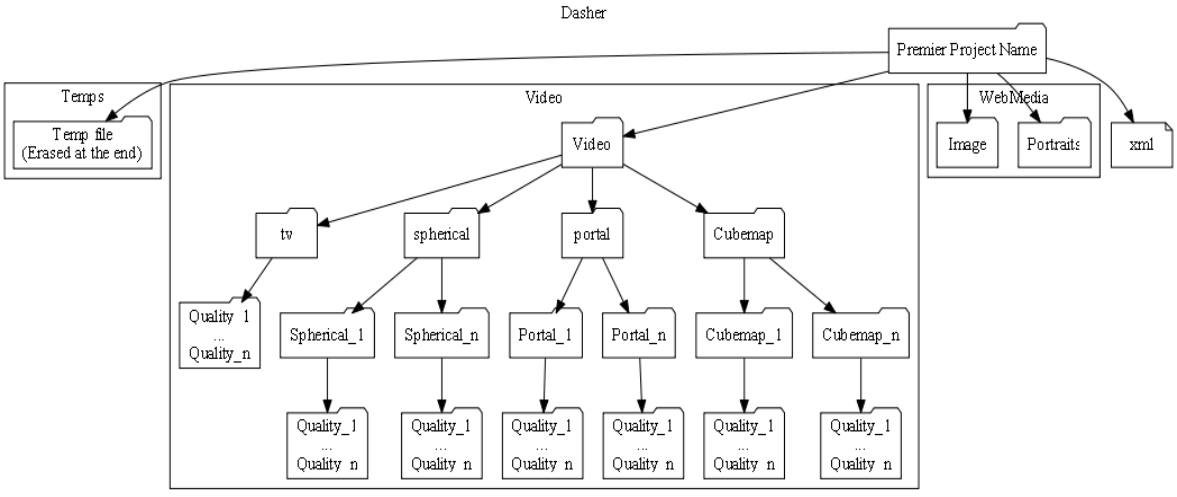

Figure 8 Output Dasher folder structure

2.1.4.2. Add and Remove Qualities.

The API implements an Edit mode, this mode it allows add new qualities or remove existing qualities. A quality in terms of MPEG-DASH makes reference to a resolution, bandwidth or a combination of both. Once the conversion has ended the new DASH content will appear in the publish view and is possible access to the edit mode, it see in the Figure 9.

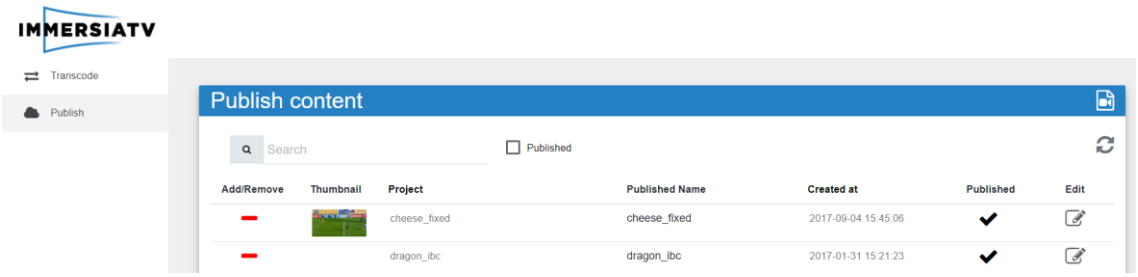

Figure 9 Publish content panel where the transcoded content is listed. Users can publish and/or edit content.

The edit mode contains a resume of the content created in the right column: Type of video (TV, Omni, and Portal), the codec used, bitrate and Resolution. In the Left column, the Add a resolution its possible add new resolutions for every type of content, defining the parameters and the last click in the "+" button.

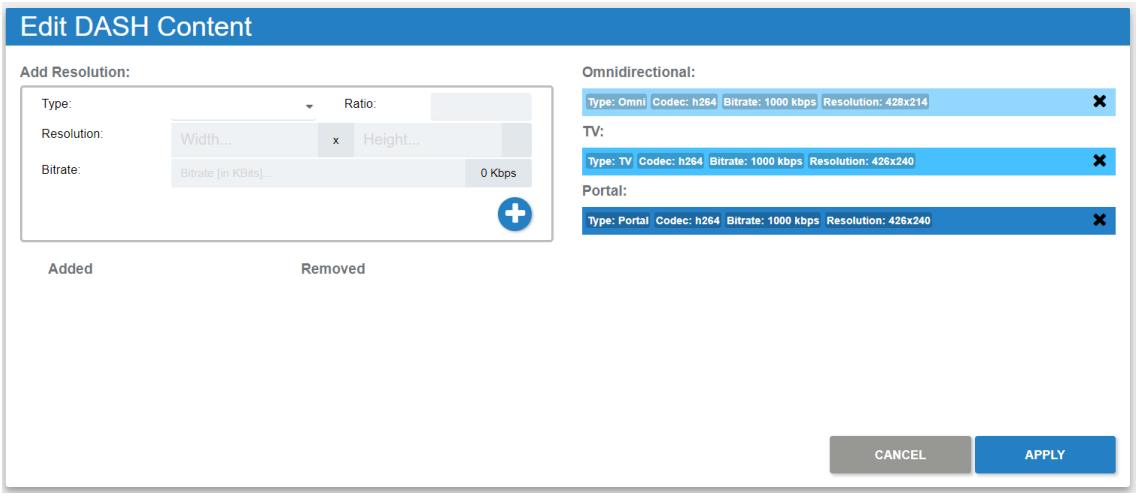

Figure 10 Panel for Add or Remove representations of previously created MPEG-DASH Content.

Once the new resolution is added, it's possible start the conversion process click on the "APPLY" button. The web return to the Transcode instance, and start the encoding process like the Create content.

2.1.5. Player

The client is implemented in two different technologies, the first one uses a 3D engine environment, Unity3D, which generates the Windows and Android player. The web implementation uses the DASH reference player. In this section a brief description of both players will be described. (See, for further details, deliverable D3.6, which introduces the ImmersiaTV player).

The methods used by the three players for media content access is exactly the same, for this reason the description of these process will be described in general terms.

2.1.5.1. Connect to the Nginx web server

Once the content is created and published, the player is configured (See section 2.3 for more details)

2.1.5.2. Parse the ImmersiaTV XML

Once the content is selected in the ImmersiaTV player, the player obtains the Metadata of the content. The metadata consist in a description of the media content (TV, HMD, Tablet) and the structure (Portals, Interactions, Timeline of the events, Transitions). This metadata is needed to perform the scene and generate the components. (More details in the metadata)

2.1.5.3. Access to MPEG-DASH

For the display the media content the player will obtain the DASH Manifest Uri. The DASH Manifest, called Media Presentation Description (MPD) is an XML document that describes different resolutions available for the player consumption.

2.1.6. Monitoring module

An extra module was developed in order to monitor the resources consumed by the different modules. This module was also developed with the open-source monitoring and alerting toolkit named Prometheus[11]. Prometheus uses exporters in order to grab metrics and visualize them later in a web interface. Node exporter[12] is used to collect metrics for hardware and OS. For docker metrics cAdvisor[13] (Container Advisor) was used. This exporter provides the users with the information about the resources usage and performance characteristics of their running containers. The Prometheus module along with the two different exporters run in three different docker containers connected together. At the moment this monitoring module only works in a Linux OS.

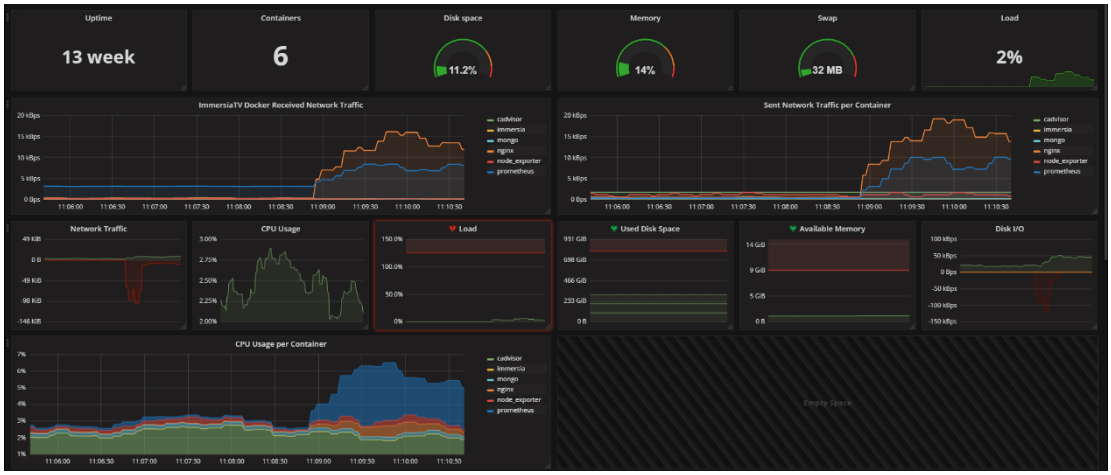

Figure 11 Docker container, hardware and OS metrics view through the Grafana interface.

As we described earlier, all the metrics are visualized in a web interface served by Grafana[14] (See Figure 9). Grafana is an open source metric analytics and visualization platform. In future releases Grafana will be removed and the web application will have its own Monitoring panel.

[11] https://prometheus.io/
[12] https://github.com/prometheus/node_exporter
[13] https://github.com/google/cadvisor
[14] https://grafana.com/

2.2. Online publication and distribution.

The publication of the live content is done via 2 applications:

- Cinegy Live VR
- Cinegy Transport

Cinegy Live VR is the primary control interface for the live production tools operator. This application generates the ImmersiaTV scene description XML file based on operator input that defines the playback behaviour on the player side. The XML file is regularly updated and stored in the location defined in the configuration:

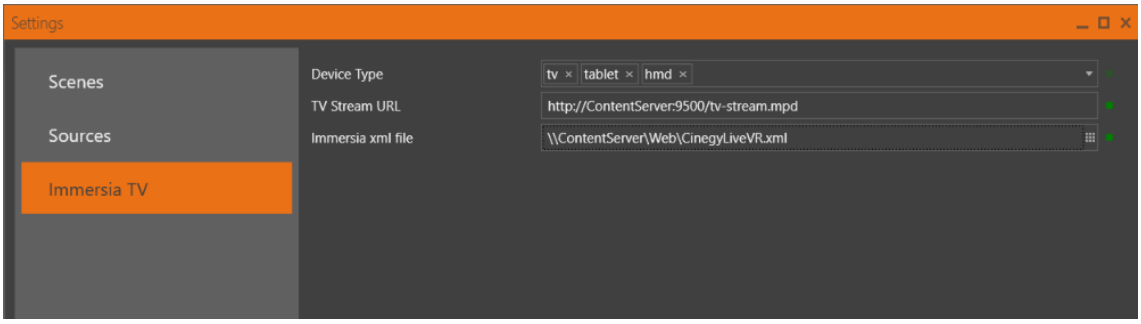

Figure 12 Settings menu

The hosting of the corresponding scene XML file should be done by 3rd party WEB server (for example, Nginx).

Cinegy Live VR is also responsible for configuring the live streams transcoding, synchronization and publication via Cinegy Transport service.

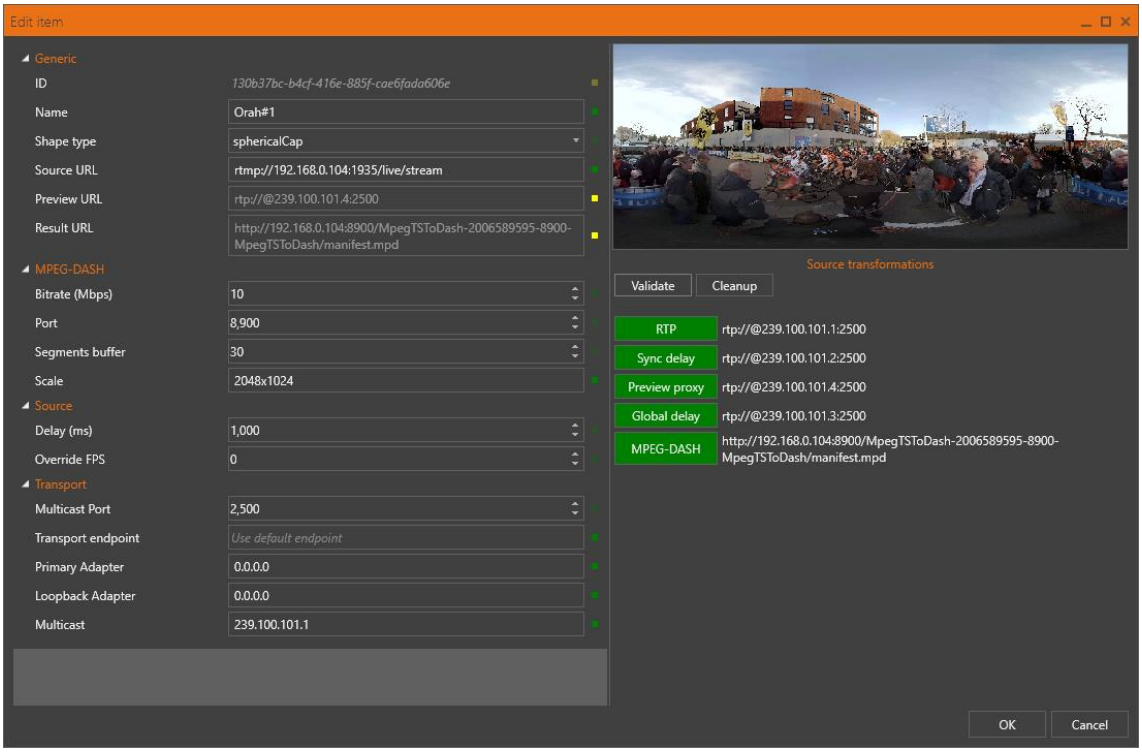

Figure 13 Edit item menu

The sources configuration is done within the same configuration dialog and allows defining required stream input settings like:

- Stream type (RTMP, RTSP, RTP)
- Transcoding parameters (bitrate, scale, buffer size)
- Synchronization offset

Cinegy Transport application is responsible transcoding and publishing the live source streams as MPEG-DASH ones. The configuration is done primarily via Cinegy Live VR configuration or via REST API:

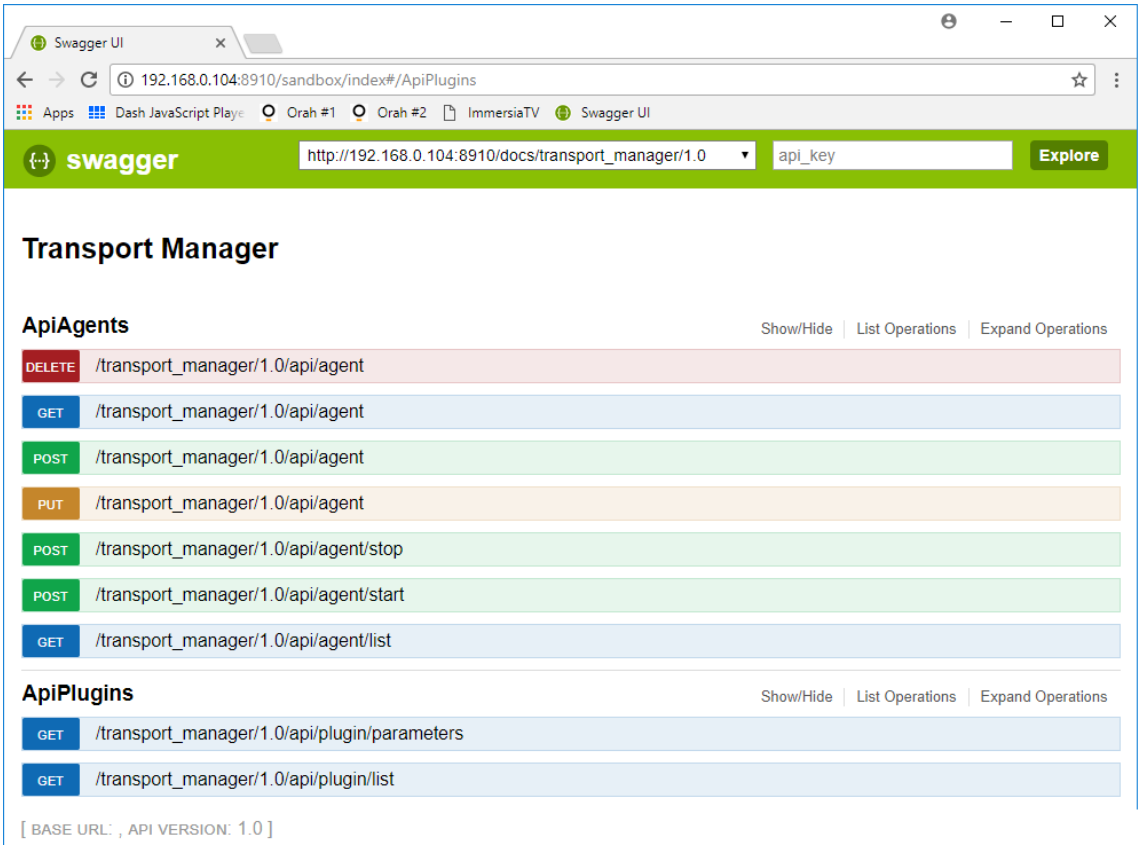

Figure 14 Swagger UI transport manager

Cinegy Transport also produces scaled preview stream to be displayed in Cinegy Live VR that provides real-time feedback for the operator.

2.3. Future work.

2.3.1. Client

The main problem of content playout is selecting the content, because this depends of the device where the player is run. For example, the reproduction of 4K content gives a good experience on a TV, but is not appropriate for a Smartphone that has a screen resolution of 1080p.

Currently, the stream quality selection of the player follows strictly the DASH standard, and therefore only considers the estimation of the bandwidth (BW) available, at playout. The selection is based in the effective throughput of the BW available at the moment of download the MPEG-DASH content.

For this reason, the player needs to integrate an adaptive algorithm, to enable switching between the available representations of the media content depending on the device.

2.3.2. Encoder

Implement H265 encoding profile for UHD.

Implement MPEG-DASH SRD for VR content using TiCMP (Tiled Cube map projection).

Port the code from Python to NodeJS for veter integration in the Server.

2.3.3. Online production tools

In order to simplify deployment of the live production tools the self-hosted WEB service will be integrated into Cinegy Live VR in order to provide default access to ImmersiaTV scene XML description file and JSON for the player. In such case no additional 3rd party components will be required to serve content for reasonably small number of clients (for example, up to 10). For large scale deployments 3rd party proxy server will be required (for example, Nginx) that will cache the requests for the live content and provide scaling/load balancing features.

3. INSTALLATION GUIDE

3.1. Offline distribution and reception service.

MATERIALS

To install the ImmersiaTV distribution and reception service, you will need the following hardware components:

- A server (Windows, Mac OSX or Linux)
- Reception devices (Android TV Box Setup, Android mobile phones and/or a windows PC)

You will also need the following software:

- Docker
- ImmersiaTV web application installer YAML script
- ImmersiaTV players

PROCEDURE

Docker tools installation:

For Windows and Mac OSX, this comes in the package Docker Toolbox. To install it:

https://www.docker.com/products/docker-toolbox

Docker Toolbox is installed together with VirtualBox. In order for the server to work correctly we will have to open the next ports (8080, 8083) in:

- Settings > Network > Advanced > Port Forwarding (See figure 2)

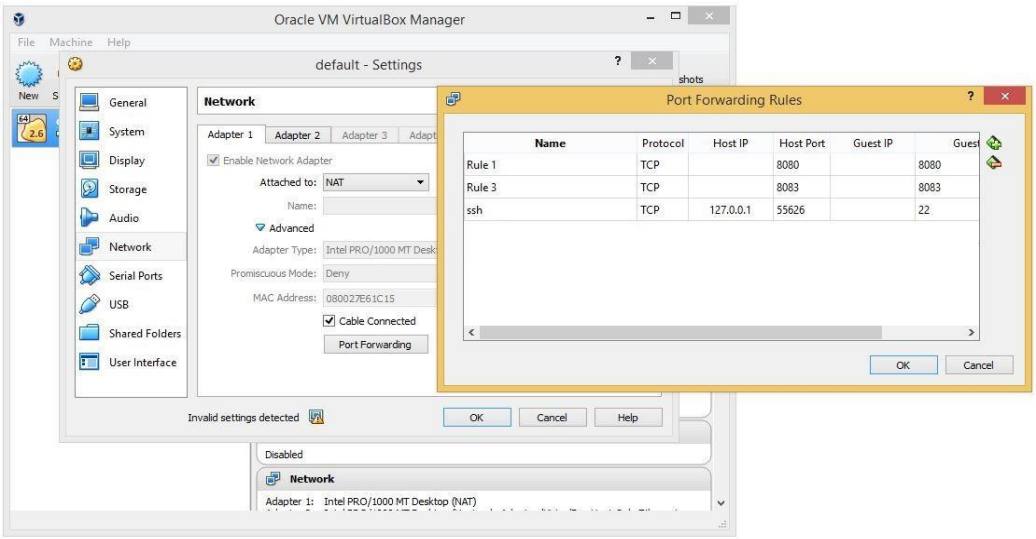

Figure 15 VirtualBox port opening menu interphase

For Linux we will have to install Docker and Docker Compose following the instructions:

https://docs.docker.com/engine/installation/linux/ubuntulinux/

Ports don't have to be opened for this case. In the host, where the web application is been deployed, create three folders in any known location:

- Folder for exported projects from Premiere (e.g. ~/User/input).
- Folder for the DASH content (e.g. ~/User/output)
- Folder for database files (e.g. ~/User/mongodb)

Download the generic YAML script from the ImmersiaTV ftp server:

ftp://ftp.immersiatv.eu/releases/0.2/docker-compose.yml

Edit the **highlighted** parts with your personal settings:

```
nginx:
    container_name: nginx
    image: livemediastreamer/nginx:${NGINXTAG}
    volumes:
        - ${OUTPUTPATH}:/data/dash
    ports:
        - "${CONTENT_SERVER_PORT}:80"
immersia:
    container_name: immersia
    image: livemediastreamer/immersia-server:${SERVERTAG}
    volumes:
        - ${OUTPUTPATH}:/data/dash
        - ${INPUTPATH}:/data/input
        - /var/run/docker.sock:/var/run/docker.sock
    environment:
        - DOCKERIZED=true
        - API_IP=${API_IP}
        - API_PORT=${API_PORT}
        - DASHERTAG=${DASHERTAG}
        - VERSION=${VERSION}
        - CONTENT_SERVER_IP=${CONTENT_SERVER_IP}
        - CONTENT_SERVER_PORT=${CONTENT_SERVER_PORT}
    ports:
        - "${API_PORT}:8083"
    links:
        - mongo:mongo
mongo:
    container_name: mongo
    image: livemediastreamer/immersia-mongo:${MONGODBTAG}
    volumes:
        - ${MONGODBPATH}:/data/db
    ports:
        - "27017:27017"
```

Figure 16 docker-compose.yml

In order to fill in the variables of the YAML script download the *.env.example* file in the same location the *docker-compose.yml* is located and make your own configuration:

ftp://ftp.immersiatv.eu/releases/0.2/.env.example

```
## Uncomment for local setup
#DOCKERIZED=false
#DASHERPATH=

## Select between 'full' and 'light'. Light version doesn't permit ## transcoding job,
VERSION=full

## Absolute path for the input and output content folders.
INPUTPATH=~/User/input/
OUTPUTPATH=~/User/output/

## Absolute path where the database data should be stored.
MONGODBPATH=~/User/mongodb/

## Port and ip address for the transcoding server.
API_PORT=8083
API_IP=

## Add the port and ip address for the content server (ip address ## might be the same as the
transcoding server
## if they are running in the same host).
CONTENT_SERVER_PORT=8080
CONTENT_SERVER_IP=

## Docker image version (Tag).
NGINXTAG=0.20
DASHERTAG=0.20
```

Figure 17.env.example used to fill the variables in the docker-compose.yml

Open a Terminal (Docker Terminal for Windows and Mac OSX), navigate to the path where the *docker-compose.yml* is located and insert the command:

```
docker-compose up -d
```

The '-d' tag is used in order to run the docker container in background. If something goes wrong press [CTRL + C] twice and insert the next command followed by the list of container names separate by spaces:

```
docker rm -f CONTAINER_NAME1 CONTAINER_NAME2 ...
docker-compose up -d
```

Navigate to http://[HOST_IP]:8083 and the ImmersiaTV content distribution web application has to be shown.

3.2. Online production tools

MATERIALS

To install the ImmersiaTV Live production tools, you will need the following hardware components:

- A server/workstation (Windows OS)
- Reception devices (Android TV Box Setup, Android mobile phones and/or a windows PC)

You will also need the following software:

- 3rd party WEB hosting service (Microsoft IIS, Nginx, etc)
- ImmersiaTV players

PROCEDURE

In order to install Cinegy Live VR and Cinegy Transport applications run the corresponding installer and follow the setup wizard steps:

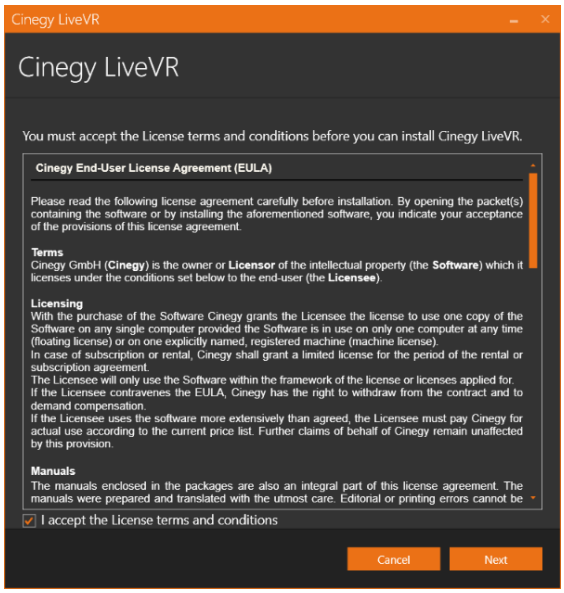

Figure 18 Cinegy LiveVR installation setup wizard

Once installed start Cinegy Live VR application to configure the live production environment.

Open Settings dialog using the “cog” button in the bottom right corner of the application window:

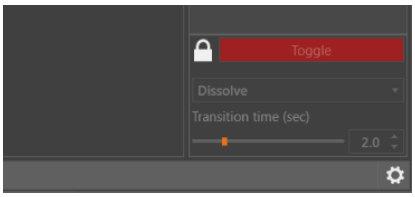

Figure 19 “Cog” button for settings menu

In the settings dialog activate “Scenes” tab and setup as many ImmersiaTV player scenes are required:

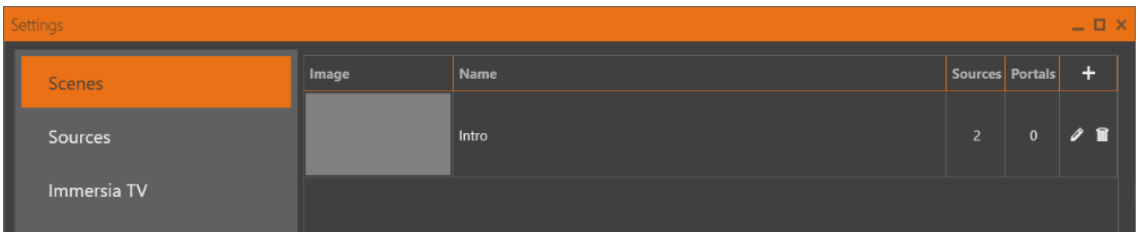

Figure 20 Settings - create scene

Each scene represents a separate set of source videos and portals that can be activated on operator request.

Once scenes setup is complete, activate “Sources” tab and define sources (input live streams or static images) that will be used during the live production:

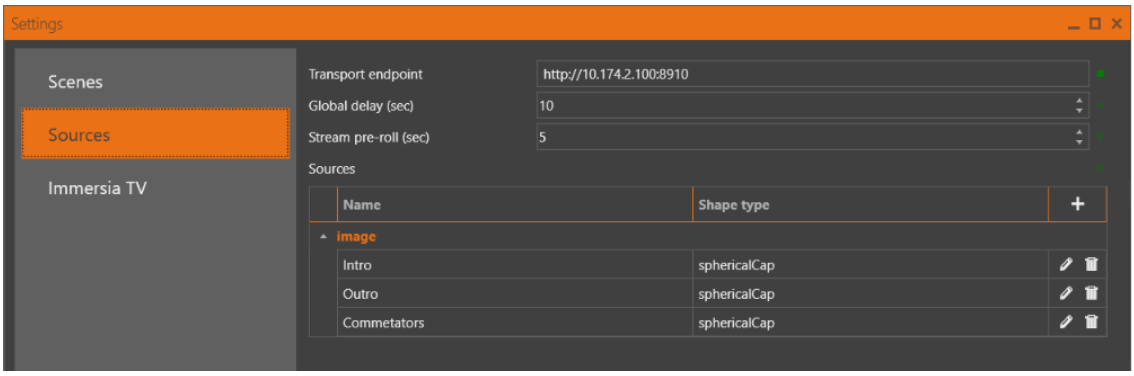

Figure 21 Settings - create sources

For each of the source define the required stream input settings like:

- Stream type (RTMP, RTSP, RTP)
- Transcoding parameters (bitrate, scale, buffer size)
- Synchronization offset

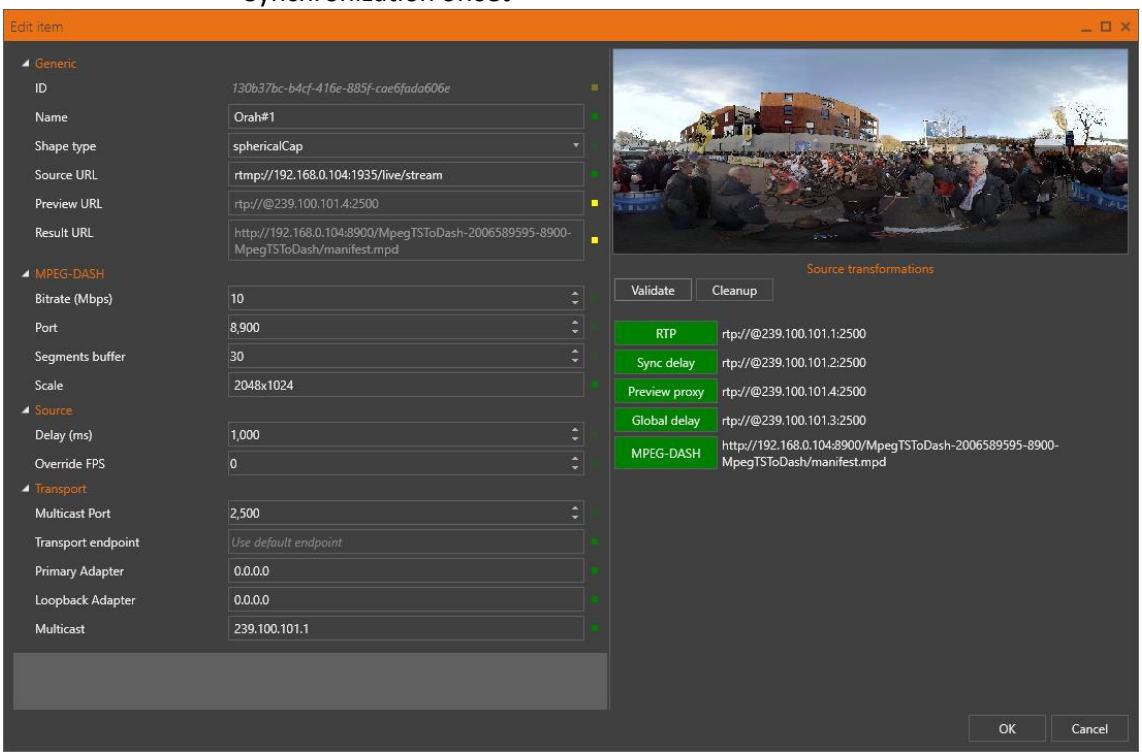

Figure 22 Settings – source parameters

When configuration is complete press “Validate” butting and follow the validation wizard steps to configure Cinegy Transport transcoding pipeline for the selected source:

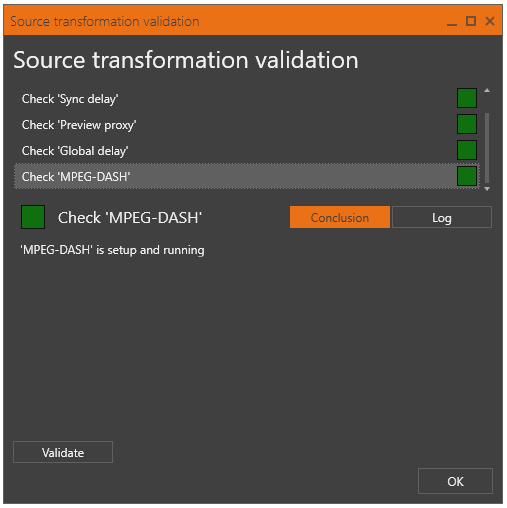

Figure 23 Source information validation

When all required sources are defined and configured, activate “ImmersiaTV” tab to configure playback related settings:

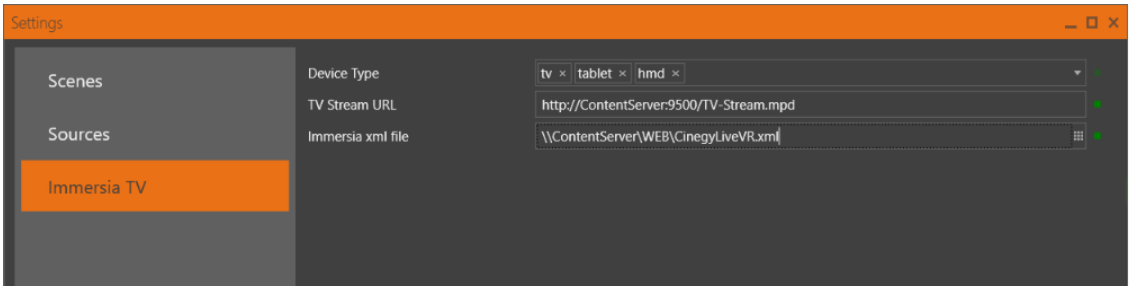

Figure 24 Immersia TV tab

Define here the list of devices to generate scene description for, URL for the directional TV stream and storage location for the ImmersiaTV scene XML file.

The setup of live production tools and transcoding service is complete. In order to access the live content in ImmersiaTV player the player content description JSON file and scene description files should be accessible via HTTP. This should be done via your favourite WEB hosting service, for example Nginx:

- In the nginx.conf define the additional *location* section to host scene description file and player content description JSON file
- Restart the service in order to apply the settings

3.3. Client configuration

Follow the instructions for Client installation in Deliverable D3.6 Interaction and Display. In order to play the content published in section 2.1 Publication & Distribution:

1. Launch the app in your Smartphone.
2. When the app is running the player's main screen will be displayed (See figure 25):

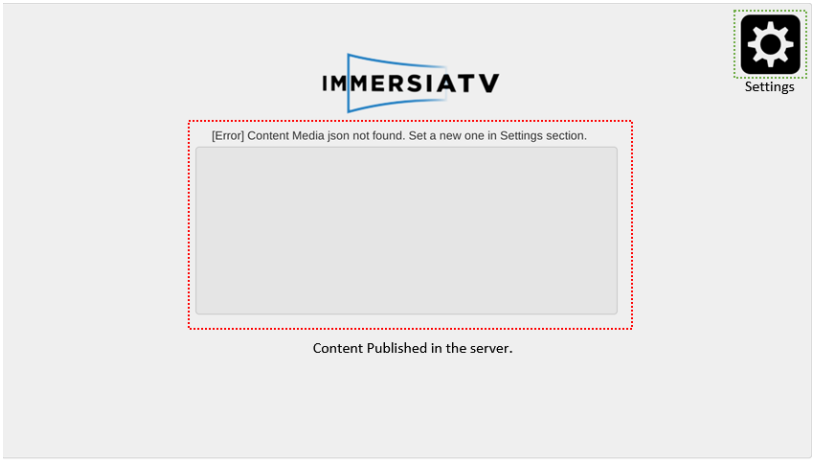

Figure 25 Player's main screen

3. Tap the button settings, introduce the content media JSON's URI in the next format (See figure 26) and press connect:
   - http://[HOST_IP]:8080/dash/JSON_file

Figure 26 Settings view

4. If the connection to the JSON file is established, the published content will be listed (See figure 27):

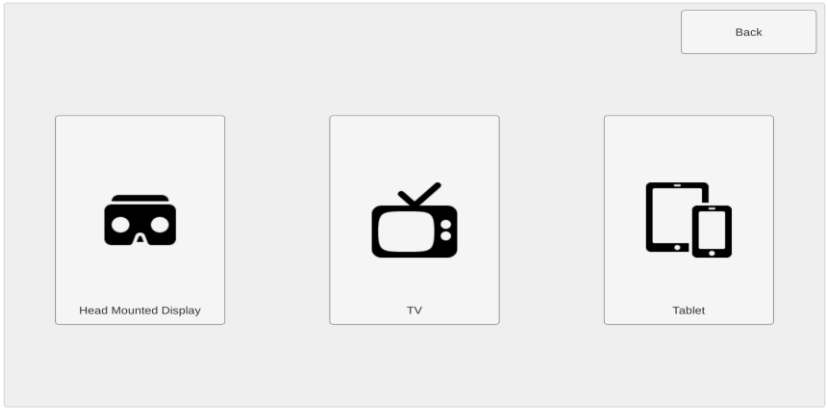

Figure 27 Video format election menu

5. Select one of the listed content and choose the viewing format between HMD, TV or Tablet (See figure 28).

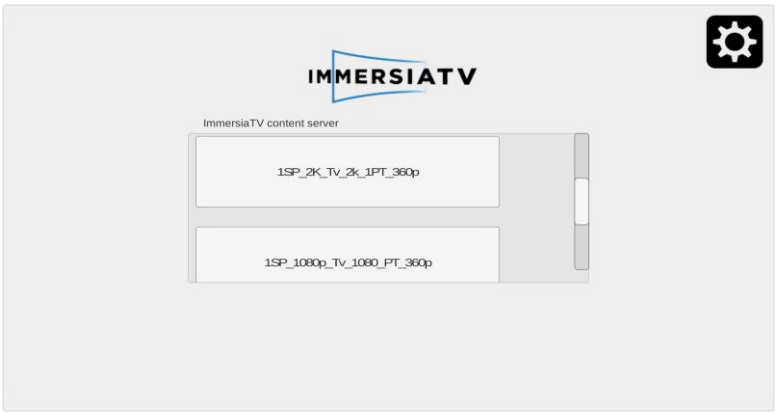

Figure 28 Content list view

4. REFERENCES

[1]. **International Organization of Standardization**. ISO/IEC 23009-1:2014. *Information technology - Dynamic adaptive streaming over HTTP (DASH) -- Part 1: Media presentation description and segment formats.*