



**D6.4.2**

# **OVERVIEW OF THE FIC2LAB PLATFORM**

**June 2015**

## **ABSTRACT**

This document accompanies the prototype FIC2Lab, an online platform where third parties can discover, try, tweak and run Specific Enablers produced by the FI-Content2 project.

This document is a deliverable of the FI-CONTENT 2 integrated project supported by the European Commission under its FP7 research-funding programme, and contributes to the FI-PPP (Future Internet Public Private Partnership) initiative.

## DISCLAIMER

FI-CONTENT 2 consortium members own all intellectual property rights protected by the applicable laws. Except where otherwise specified, all document contents are “© FI-CONTENT 2 project - All rights reserved”. Reproduction is not authorised without prior written agreement.

All FI-CONTENT 2 consortium members have agreed to full publication of this document.

The commercial use of any information contained in this document may require a license from the owner of that information.

All FI-CONTENT 2 consortium members are also committed to publish accurate and up to date information and take the greatest care to do so. However, the FI-CONTENT 2 consortium members cannot accept liability for any inaccuracies or omissions nor do they accept liability for any direct, indirect, special, consequential or other losses or damages of any kind arising out of the use of this information.

## DELIVERABLE DETAILS

[Full project title]:	Future media Internet for large-scale <b>CONTent</b> experim <b>ENT</b> ation 2
[Short project title]:	FI-CONTENT 2
[Contract number]:	603662
[WP n°]:	WP6 – Technology Enablers
[WP leader]:	Mario LOPEZ-RAMOS, Thales
[Deliverable n°]:	D6.4.2
[Deliverable title]:	Overview of the FIC2Lab Platform
[Deliverable nature]:	Report (R)
[Dissemination level]:	Public (PU)
[Contractual delivery date]:	M27 – June 2015
[Actual delivery date]:	6 July 2015
[Editors]:	Peter Muryshkin and Heike Horstmann, Fraunhofer IAIS Assisted by Pieter van der Linden – VIVITnet.
[Internal Reviewers]:	Mario LOPEZ-RAMOS, Thales / Dirk KRAUSE, Pixelpark
[Suggested readers]:	Accelerators, SMEs and web application developers
[Keywords]:	FIC2Lab, discover, test, tweak and run Specific Enablers
[File name]:	FI-CONTENT 2_WP6-008_D6.4.2_V1.0.docx

## EXECUTIVE SUMMARY

FIC2Lab is an online platform with interactive demonstration of Specific Enablers produced by the FI-CONTENT 2 project.

This document introduces to the FIC2Lab prototype, targeting Phase 3 accelerators and sub-grantees. The platform provides convenient means for discovering, trying, tweaking and enablers produced by the FI-CONTENT2 project.

## LIST OF AUTHORS

Organisation	Author
Thales	Mario LOPEZ-RAMOS
Thales	Geoffroy CHOLLON
eBiz	Olivier DUVOID
eBiz	Loïc ORTOLA
Pixelpark	Dirk KRAUSE
Fraunhofer IAIS	Peter Muryshkin, Heike Horstmann
VIVITnet	Pieter van der Linden

## TABLE OF CONTENTS

<b>EXECUTIVE SUMMARY .....</b>	<b>3</b>
<b>LIST OF AUTHORS.....</b>	<b>4</b>
<b>TABLE OF CONTENTS.....</b>	<b>5</b>
<b>LIST OF FIGURES AND TABLES .....</b>	<b>7</b>
<b>1 - INTRODUCTION .....</b>	<b>8</b>
<b>2 - DISCOVER AND TRY SEs.....</b>	<b>9</b>
2.1 - Catalogue .....	9
2.1.1 - <i>The discover web page</i> .....	9
2.1.2 - <i>Selection procedure of enablers</i> .....	9
2.2 - Per enabler discover page .....	10
2.3 - Per enabler demo page.....	10
2.4 - Interactive API description using Swagger .....	11
2.4.1 - <i>Introduction</i> .....	11
2.4.2 - <i>Wording</i> .....	11
2.4.3 - <i>Swagger for FIC2Lab – SE developer guide</i> .....	11
2.4.3.1 - <i>Procedure</i> .....	12
2.4.3.2 - <i>Sample</i> .....	12
2.5 - Interactive API description using Doxygen .....	13
2.5.1 - <i>Using Doxygen with FIC2Lab - SE developer guide</i> .....	13
2.6 - Documentation generation procedure using <a href="http://wiki.mediafi.org">wiki.mediafi.org</a> .....	13
<b>3 - TWEAK CLIENT-SIDE SEs.....</b>	<b>15</b>
3.1 - Design .....	15
3.2 - Implementation.....	15
3.2.1 - <i>One click deployment, random hashes and no-auth</i> .....	15
3.2.2 - <i>Configuration</i> .....	16
3.2.3 - <i>Immediate feedback on tweaks</i> .....	16
3.2.4 - <i>Download, Github integration and shares</i> .....	16
3.2.5 - <i>Initialization and maintenance</i> .....	16
3.3 - User guide .....	16
<b>4 - RUN SERVER-SIDE SEs.....</b>	<b>19</b>
4.1 - Design and implementation.....	19
4.1.1 - <i>GEs considered in the design process</i> .....	19
4.1.2 - <i>High-level architecture</i> .....	19
4.1.3 - <i>Authorization (using OAuth2 implicit flow)</i> .....	22

4.1.4 - Deployment phase.....	23
4.1.5 - Running phase.....	25
4.1.6 - Authorization using an IDM without OAuth2 implicit flow .....	26
4.2 - User guide .....	28
4.2.1 - Introduction .....	28
What is Docker?.....	29
How can I use the Docker images of the Specific Enablers? .....	29
4.2.2 - Run SEs using the FIC2Lab runner web-based tool (on top of FIWARE Lab Cloud).....	30
4.2.3 - Run SEs in your own machine .....	34
Installing Docker .....	34
Running a SE from the command line .....	34
4.2.4 - Run SEs using command-line tools in a remote machine in the FIWARE cloud .....	36
<b>5 - PACKAGING AND DELIVERY OF SERVER-SIDE SEs .....</b>	<b>37</b>
5.1 - Initial situation .....	37
5.2 - Problem analysis .....	37
5.3 - Methodology and systematic problem assessment .....	39
5.4 - Solution design and architectural concept .....	42
5.5 - Implementation.....	42
5.6 - Docker script runner.....	44
5.7 - Live component status report via Jenkins CI .....	45
5.8 - Proposals for further improvement for FIC2Lab acceptance checks.....	47
<b>6 - CONCLUSION .....</b>	<b>49</b>

## LIST OF FIGURES AND TABLES

### LIST OF FIGURES

Figure 1-1 - lab.mediafi.org home page .....	8
Figure 2-1 - FIC2Lab Discover function overview .....	9
Figure 2-2 - FIC2Lab Discover - per enabler presentation page .....	10
Figure 2-3 - FIC2Lab Try - Example of demo link (3DMap Tiles enabler). .....	11
Figure 2-4 - The swagger API browsing interface provided on FIC2Lab (example of context aware recommendation). .....	13
Figure 2-5 - Document creation and update overview page hosted on wiki.mediafi.org .....	14
Figure 2-6 - Document creation and update page on wiki.mediafi.org (OCDB example) .....	14
Figure 3-1 - Example of use of playground – 3DMapTiles .....	17
Figure 3-2 – Playground: configuration details of a project .....	17
Figure 3-3 - Playground showing the result of a tweak .....	18
Figure 4-1 – FIC2Lab runner & static files .....	20
Figure 4-2 – “Middle man” setup .....	21
Figure 4-3 – “Delegated” setup .....	22
Figure 4-4 - Runner authentication sequence: Implicit Grant Flow .....	23
Figure 4-5 - FIC2Lab Runner architecture (deployment) .....	24
Figure 4-6 – Automation of the FIWARE OpenStack .....	25
Figure 4-7 - FIC2Lab Runner architecture (runtime) .....	26
Figure 4-8 - Port mapping .....	26
Figure 4-9 – Architecture changes for the IDM update .....	27
Figure 4-10 – Workflow changes for the IDM update .....	28
Figure 4-11 - The FI-CONTENT 2 public repository of Docker images .....	29
Figure 4-12 - FIWARE Lab portal to sign up and request community upgrade .....	30
Figure 4-13 - FIC2Lab runner (beta) - Landing page .....	31
Figure 4-14 - FIC2Lab runner (beta) - Deployment wizard .....	31
Figure 4-15 - FIC2Lab runner (beta) - User-dedicated management console .....	32
Figure 4-16 - FIC2Lab runner (beta) - Social Network Enabler SE launched from the console .....	33
Figure 4-17 - FIC2Lab runner (beta) - Changing port mapping of the SNE SE instance .....	34
Figure 4-18 - DockerHub page of ppnet, the Social Network Enabler .....	35
Figure 4-19 - Docker CLI - Downloading Social Network SE .....	35
Figure 4-20 - Docker CLI - Social Network SE downloaded and running .....	36
Figure 4-21 - Docker CLI - Checking running containers .....	36
Figure 5-1 – Packaging and delivery – a naïve approach .....	37
Figure 5-2 – Packaging and delivery of SEs – Problem analysis .....	38
Figure 5-3 – Delivery and QA concept for server side SEs .....	42
Figure 5-7 – Architecture specification .....	44
Figure 5-5 – Live status of SE stability via Jenkins dashboard .....	46
Figure 5-6 – Jenkins dashboard for integration of Generic Enablers .....	46
Figure 5-7 – Jenkins build statistics since January 2015 .....	47

## 1 - INTRODUCTION

FIC2Lab addresses the needs of Phase 3 projects and developers by means of four support levels:

- **Discover:**
  - Access information about Specific Enablers
  - Description / Programmer guide / Installation guide / Tutorial/ Usage videos / T&C
  - How to get support
- **Try:** Test live demos of Specific Enablers
  - Demo servers provided by the enabler owners (May use or not use FIC2Lab servers)
- **Tweak:** Use our developer playground and access our support portal
  - Real-time try/modify/run loop
- **Run:** various SysOp tools for starting and monitoring instances of enablers
  - Layered on top of FIWARE LAB
  - Use of advanced tools for facilitating usage and deployment

The functions of FIC2Lab are proposed to the public via the <http://lab.mediafi.org> website.

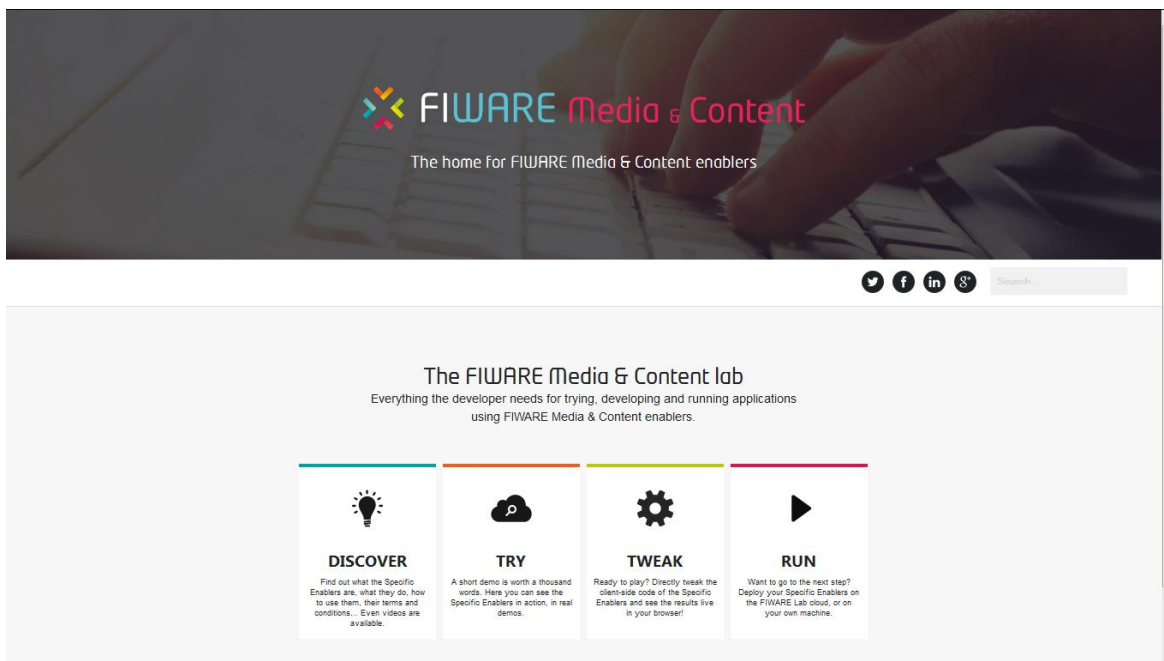


Figure 1-1 - lab.mediafi.org home page



## 2 - DISCOVER AND TRY SES

### 2.1 - Catalogue

#### 2.1.1 - The discover web page

The discover page targets developers with a centralized repository of data and resources. The main information provided is:

- descriptive data,
- programming guides,
- examples,
- demo endpoints,
- terms and conditions for use in FI-PPP and beyond,
- source code when applicable (open source),
- links to running instances and or links to executable images.

The discover page provides the lists of enablers available on FIC2Lab and allows the user to search for enablers or to select them via a menu listing using tags provided by the designers of the enabler.



Figure 2-1 - FIC2Lab Discover function overview

#### 2.1.2 - Selection procedure of enablers

In the course of the FI-Content2 project, its participants produced 29 Specific Enablers. Developers tested this software in small, medium or large scale experimentations conducted on one or several of the experimentation sites. Based on criteria established by the partners individually, we selected a subset of 23 Specific Enablers as candidates to be featured on FIC2Lab.

For being actually featured on FIC2Lab a number of formal acceptance criteria have been defined. These criteria are:

- Availability of presentation,
- Availability of thumbnail image,

- Availability of stable Docker Image OR a productive SaaS instance,
- Availability of clear Terms a Conditions,
- Availability of the programming guide,
- Availability of the installation guide,
- Availability of the API description,
- Availability of a Bug Tracking infrastructure,
- Availability of Demo instance end-point,
- Availability of test and/or demo script.

Specific Enablers that do not comply with those conditions cannot be featured on FIC2Lab.

We believe that this setup allows to gain more control of quality across FIC2Lab.

## 2.2 - Per enabler discover page

The discover interface features one global information page for each of the enablers featured on FIC2Lab.



**Figure 2-2 - FIC2Lab Discover - per enabler presentation page**

In addition to providing general information, terms and conditions, and links to code and documentation, the per enabler discover page provides links:

- to demo instances (mandatory);
- links to the playground tweak environment (optional);
- and links to support and bug tracking.

## 2.3 - Per enabler demo page

Each SE needs to provide the endpoint of a running demo. The 'try' link opens a browser tab and displays the demo provided by the enabler owner.

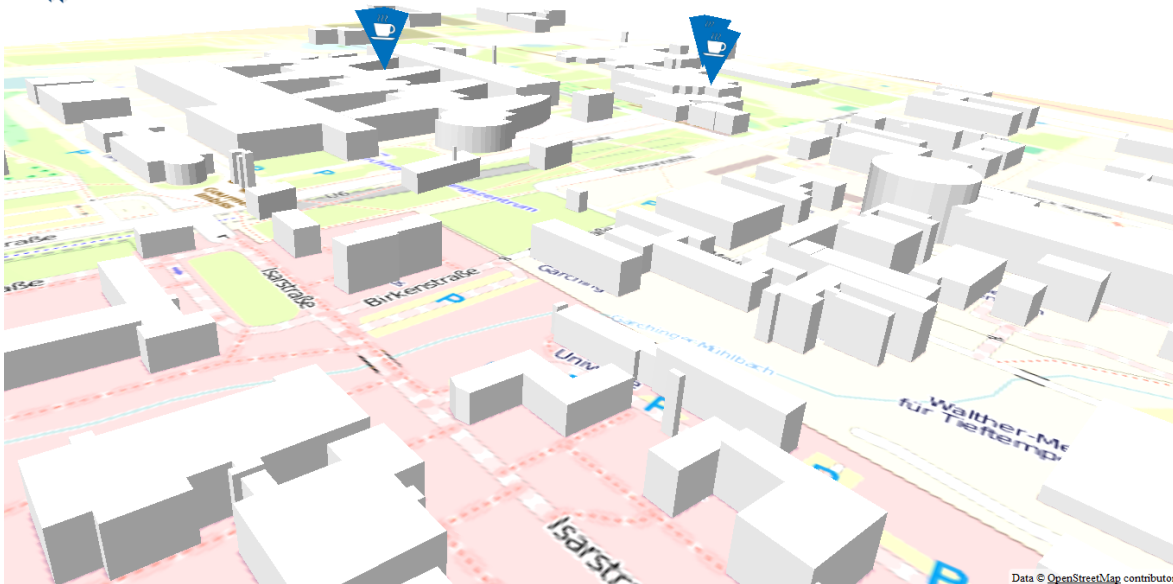


Figure 2-3 - FIC2Lab Try - Example of demo link (3DMap Tiles enabler).

## 2.4 - Interactive API description using Swagger

### 2.4.1 - Introduction

Swagger is a specification designed to describe RESTful APIs. To do so, it provides a very simple and intuitive meta-model. This document aims at understanding both the wording and the procedure to jumpstart the creation of your own API description.

### 2.4.2 - Wording

- **Swagger:** a specification, which provides a meta-model to describe APIs. The second major release (2.0) was launched in September 2014. The spec can be found at <https://github.com/swagger-api/swagger-spec/>
- **Swagger Implementation:** Swagger supports multiple programming languages. An implementation is usually used through your existing application code, and contains two main sides: A simple endpoint which exposes the swagger API, and additional parameters at every endpoint you produce (description, attribute description etc...). Multiple implementations are available and listed on the swagger spec GitHub project. An alternative is to produce your own swagger implementation (understand make your own API description manually, and expose it on a webserver as static json files). If this procedure was particularly heavy in Swagger v1, the v2 specification allows anyone to produce an implementation from a very simple YAML file.
- **Swagger UI:** A set of static files (HTML, JS, CSS...) which provides a web UI rendering multiple APIs in a fancy and well-readable way and allows you to interact with them in a frontend sandbox space. The demo is available on <http://petstore.swagger.io/>

### 2.4.3 - Swagger for FIC2Lab – SE developer guide

We developed a multi-source version of Swagger UI for FIC2Lab. It looks like a standard Swagger UI (a web page displaying in a friendly UI your own API description). Technically speaking, the static configuration files provided by SE developers are REST client configuration stubs, which enable the Swagger software to query

multiple REST endpoints. In order to simplify the procedure, every partner should create a dedicated folder on GitHub to post their own implementation JSON file (no matter how this implementation is generated) <https://github.com/fic2/swaggerfiles>

The name of the folder must be the same as the wiki official name of the related SE. This is important because we use automatic validation of this resource.

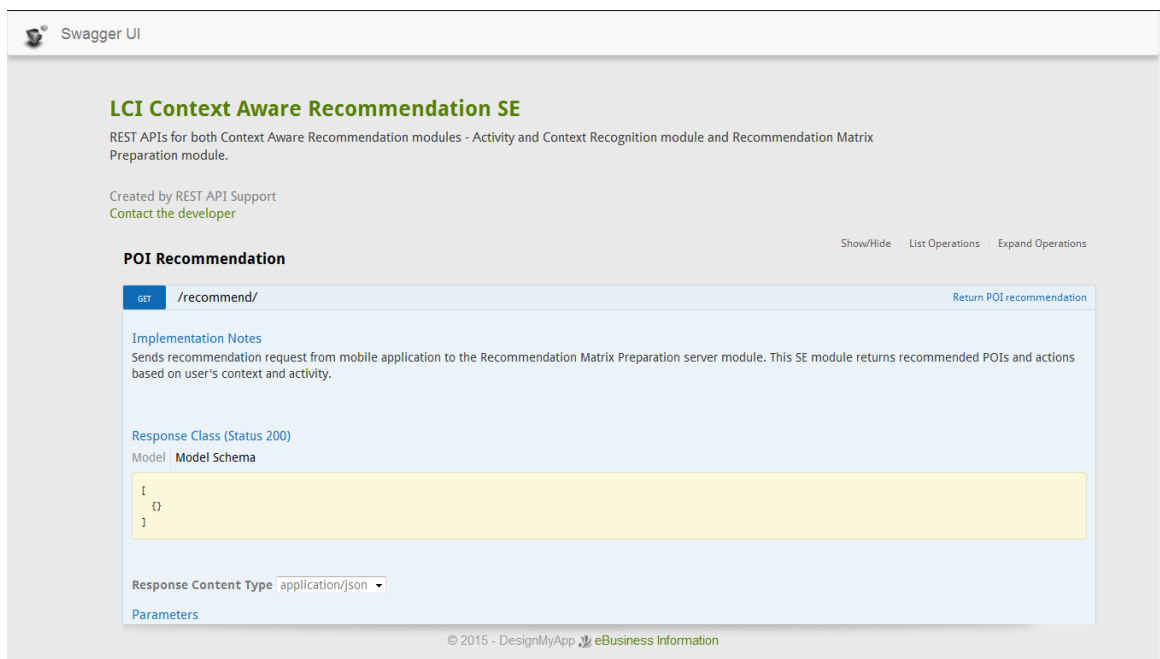
#### 2.4.3.1 - Procedure

1. Each partner needs to provide one JSON file (called the swagger implementation). Since 2.0, you can now generate the JSON file using a simple YAML model. Learning how to use it should be straightforward with those two links: <http://editor.swagger.io/#/> (a web-editor tool made by swagger, with a default sample being the Uber API description). <https://github.com/swagger-api/swagger-spec/wiki>Hello-World-Sample> (a hello world sample which lays out exactly the requirements of a swagger spec, and explains roughly the concepts put in motion. Warning: the sample is still using the 1.2 API, so it is not completely up to date.) eBiz recommends using the solution above-mentioned to make your own descriptor.
2. The file should be made available on GitHub as: [https://github.com/fic2/swaggerfiles/\[yourenabler\]/swagger.json](https://github.com/fic2/swaggerfiles/[yourenabler]/swagger.json). If you want so, you can also store your swagger.yaml file there for maintenance purposes. Please push on the *gh-pages* branch.

#### 2.4.3.2 - Sample

In order to help with the integration procedure, there is a sample which can be used as a starting codebase, available at <https://github.com/fic2/swaggerfiles/tree/gh-pages/sample>

To learn the language and see more samples, we recommend going through the Uber API swagger implementation, which appears by default on the swagger web-editor. All partners shall refer to the main specification README.md <https://github.com/swagger-api/swagger-spec/blob/master/versions/2.0.md> for an exhaustive description of the service.



**Figure 2-4 - The swagger API browsing interface provided on FIC2Lab (example of context aware recommendation).**

## **2.5 - Interactive API description using Doxygen**

While Swagger provides a convenient way to describe RESTful API, client-side enablers, particularly Unity-based ones (written in C#) do not fit the model of a RESTful API. Therefore, another tool is necessary to describe the API. We have chosen Doxygen [<http://www.stack.nl/~dimitri/doxygen/>], for its widespread use and accessibility.

### **2.5.1 - Using Doxygen with FIC2Lab - SE developer guide**

This guide assumes that your enabler is open source and on github. As it applies primarily to Unity-based enablers, it is most relevant in the context of the Pervasive Game Domain, and therefore the following documentation assumes that the enabler is for that Domain.

For another platform, please adjust the URL links.

For the API documentation, we decided to use Doxygen to document Unity-based enablers. To do so, we will upload the generated documentation on github using gh-pages. To create gh-pages, please follow this guide: <https://help.github.com/articles/creating-project-pages-manually/>

Then add the generated documentation from Doxygen which is in `doxygen/html` to the root of the *gh-pages* branch. You can see the Game Synchronization enabler as a example:

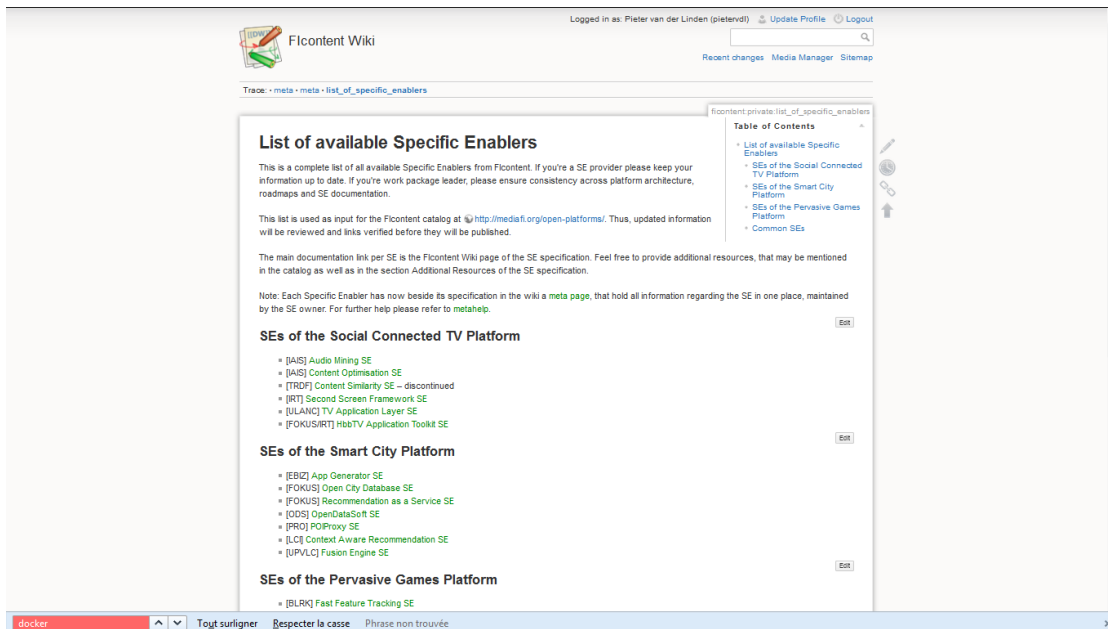
- <https://github.com/fi-content2-games-platform/FIcontent.Gaming.Enabler.GameSynchronization/tree/gh-pages>, whose API is visible at:
- <http://fi-content2-games-platform.github.io/FIcontent.Gaming.Enabler.GameSynchronization/>

Then add the latter link to the meta page at that entry:

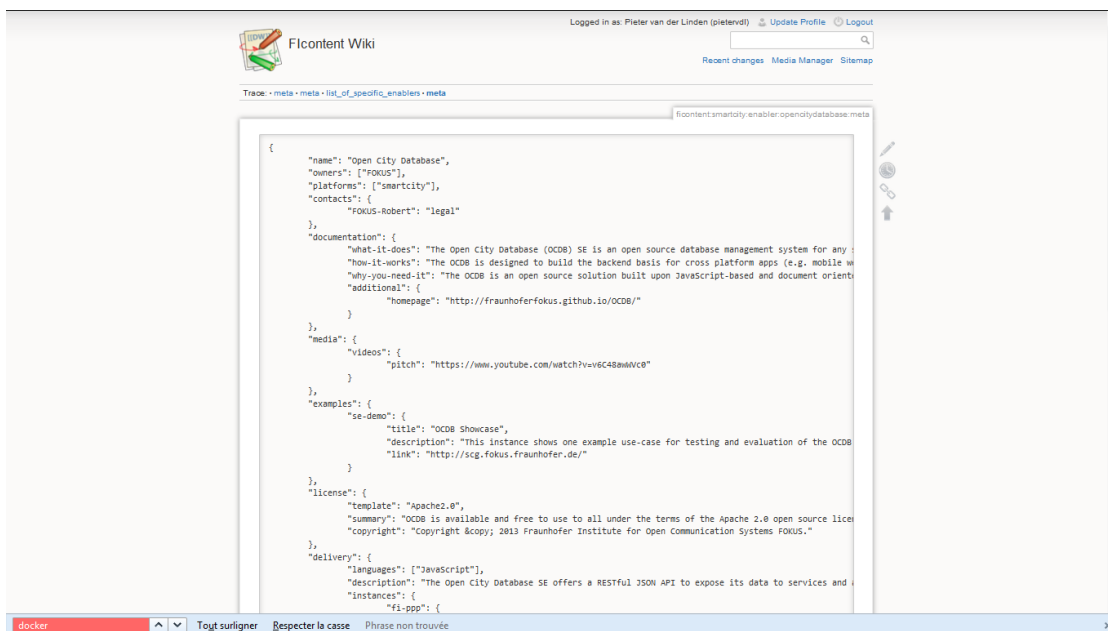
- `/documentation/api/doxygen`

## **2.6 - Documentation generation procedure using wiki.mediafi.org**

The document update procedure introduces template information pages on the FI-CONTENT2 wiki and a series of scripts processing the data provided into a JSON structure. FIC2Lab periodically scans contents and forwards information to the main online portal. The portal uses a simple MySQL database and PHP scripts for displaying the information pages.



**Figure 2-5 - Document creation and update overview page hosted on wiki.mediafi.org**



**Figure 2-6 - Document creation and update page on wiki.mediafi.org (OCDB example)**

### 3 - TWEAK CLIENT-SIDE SES

Developers can tweak enablers, interactively customize them and change their behavior to see if they match their expectation.

This explorative interactive aspect is one of the most innovative aspects of FIC2Lab: the Playground. It displays simultaneously the JavaScript code tree, an editing window to change the code in real-time and a window displaying the running enabler. Playground supports currently a subset of the enablers. More enablers may come be added over the next months.

#### 3.1 - Design

The idea behind the Playground is to give developers the means to try the Enablers in the browser

- without the need to install anything locally on their computers
- with the least amount of effort and time

This is especially useful when you have Enablers that are web applications which run mainly in a web browser (such as SPA - Single Page Applications). These applications typically are either self-contained and do not need no backend at all, or will connect to a backend (typically a database) via a REST (representational state transfer) interface. Since all FIWARE Enablers should implement a backend as a REST interface, the idea of the playground is a logical consequence.

The main requirements for the playground were:

1. one click deployment of Enablers
2. no need for authentication
3. ease of configuration
4. immediate feedback on tweaks
5. shareable URLs via random hashes
6. ease of cooperation
7. one click download of results
8. GitHub integration
9. social share

Initially we modeled the Playground after services like JSFiddle or JSBin, since these fulfill the requirement of ‘tweaking’ code in the browser. These services provide the tools to edit source code in the browser by building a web page virtually. However, they lack the ability to write the results to a file system – they build a virtual webpage that gets assembled in-memory and displayed in an iframe. We discovered we needed a solution that creates a real file tree that the user can traverse, edit and finally download or fork.

As result, the playground was created as a NodeJS application, which was finally deployed to FIWARE Lab

#### 3.2 - Implementation

We implemented all of the requirements above.

##### 3.2.1 - One click deployment, random hashes and no-auth

By simple clicking the Enablers tile on the FIC2lab page, the playground is called with a URI like `http://PLAYGROUNDIP/ENABLERNAME`. The playground invokes a new instance by generating a random hash string and adding this to the URI, which resembles this pattern: `http://PLAGROUNDIP/HASH/ENABLERNAME`. It additionally creates a folder with the hash name that holds all files needed by the enabler. As such, the user does not need no login because the only thing to be

remembered is the hashed URI, which can easily be done by bookmarking the page. Additionally, this link can be shared to possible co-operators.

### 3.2.2 - Configuration

If the Enabler provides a configuration file (optional), the playground lets you tweak this configuration, so that the user doesn't even need to edit source code. Also the configuration file holds the information which file to load as default.

### 3.2.3 - Immediate feedback on tweaks

We chose to not auto-save every tweak the user makes which would result in constant updates of the iFrame holding the result and putting stress on server, browser and user perception. We implemented a classic 'save' button instead which provides the means to write back tweaks to the file system. Also, this encourages cooperation with other users, since every user with the URI is editing the same folder.

### 3.2.4 - Download, Github integration and shares

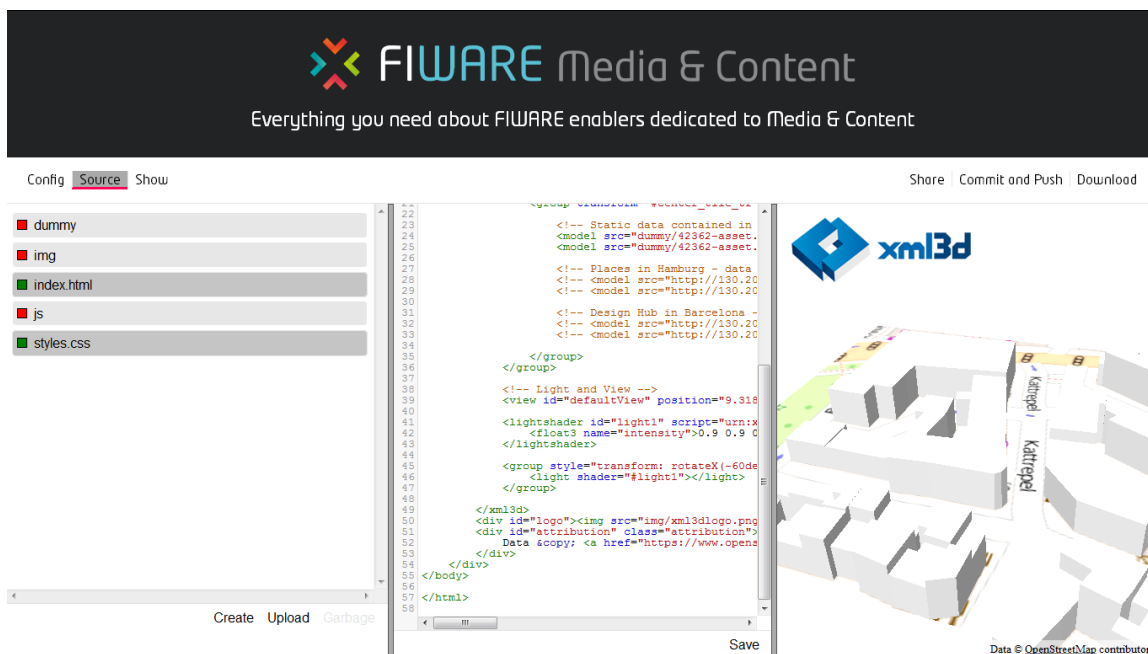
The user can at any time download the whole enabler as a ZIP file to immediately be able to access the files and try to run the code locally. We additionally implemented the ability to fork the tweaked code on Github which is the first and only time the user needs to give credentials. To share the URI that points to the tweaked playground, we provide a QR Code (for easy access via smart phones) and Twitter and Facebook share links,

### 3.2.5 - Initialization and maintenance

The initial instances of the Enablers are copies of their subsequent repositories on Github. Folders that nobody accesses over a given period are purged from the file system for maintenance reasons.

## 3.3 - User guide

As described above, the playground is accessed by simply following a specific URI that leads to the generation of a new instance by copying the repository. After that, the user discovers a screen that initially shows three panes:





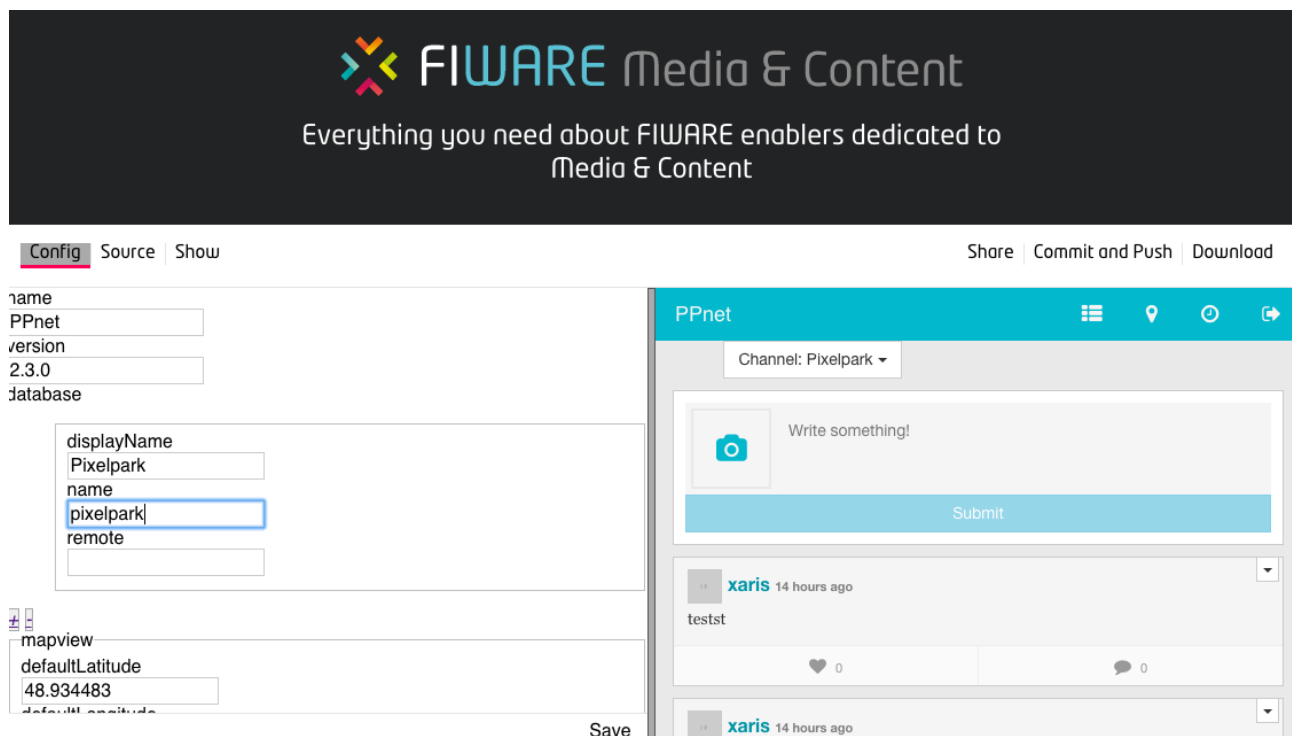
**Figure 3-1 - Example of use of playground – 3DMapTiles**

The left pane shows a classic tree view of the file system that makes the client side of the enabler. The idea is that the user only needs to edit existing files, but the playground holds the ability to upload additional files by using the buttons at the bottom.

Per default, the middle pane shows the source code of the file selected by user to tweak on. The Enabler provider can customize the name of the default file through the configuration file on GitHub. On the bottom is the 'Save' button that uploads the tweaked source of the edited file to the playground.

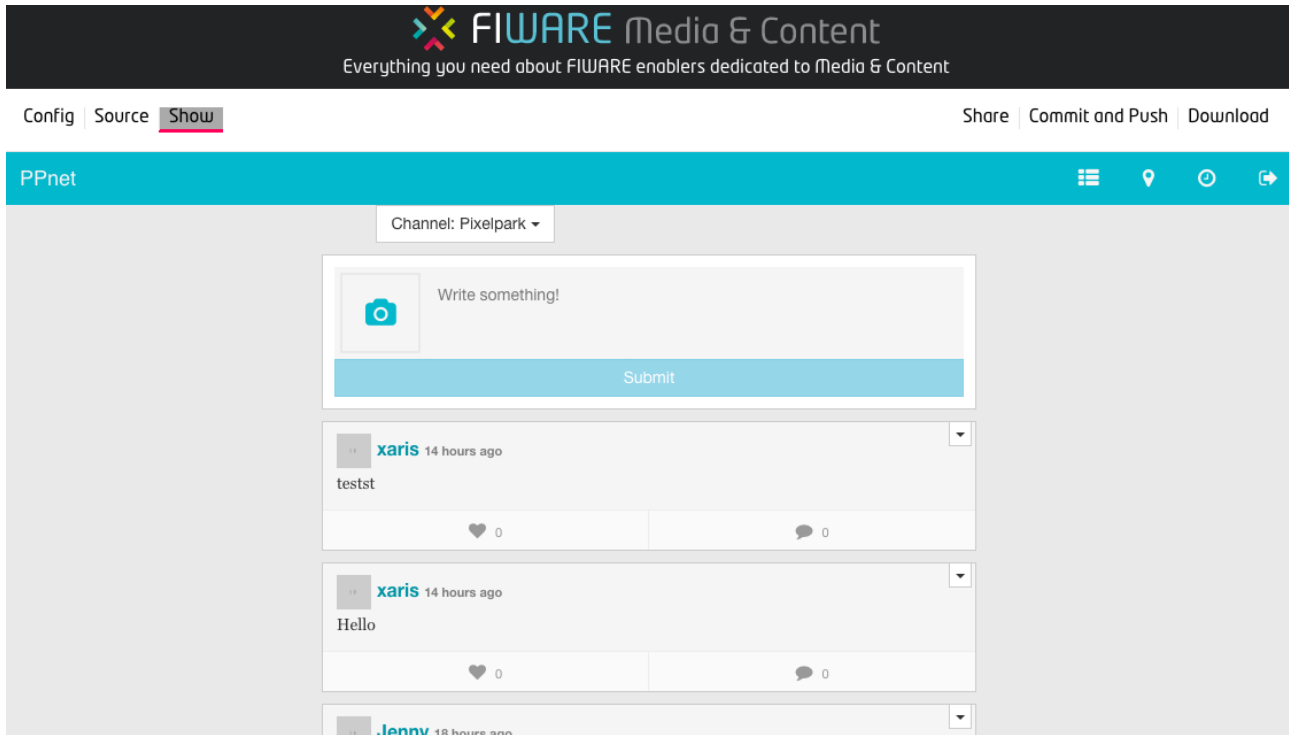
Once user saves the view, the iframe to the right reloads. It holds the result of the tweaked source code and shows the current state of the tweaks.

The 'Config' button on top changes the three-pane view to a two-pane one, showing a form for configuration details



**Figure 3-2 – Playground: configuration details of a project**

The 'Show' button gives the user the full, one-pane view of the iframe, i.e. the full result of the tweak.



**Figure 3-3 - Playground showing the result of a tweak**

It is worth mentioning that although the playground lets users edit only the client (i.e. browser) sided portion of the enablers, he can still address any server sided backend via JavaScript/AJAX calls. The server side should be aware of possible CORS (cross-origin resource sharing) issues though, and preferably provide a REST interface (though this is not mandatory for the playground).

## 4 - RUN SERVER-SIDE SES

### 4.1 - Design and implementation

#### 4.1.1 - GEs considered in the design process

The FIC2Lab team considered a number of alternatives based on FIWARE technology during the design of the FIC2Lab runner (as described on D6.2.2):

- The FIWARE Cloud Portal offers virtual infrastructure in an on-demand basis and is therefore a convenient hosting solution. It is an Infrastructure-as-a-Service (IaaS) solution, a basic alternative to Amazon EC2. However:
  - it requires an understanding of virtual machines, security key pairs, network interfaces, public IP addresses, firewalls, Linux administration, etc. which is a barrier for certain developers;
  - FIWARE Lab offers limited resources, e.g. only 1 public IP. This could be circumvented by using NAT (e.g. by setting up a reverse proxy) but adds additional complexity;
  - it is slow for frequently testing and updating new versions.
- FIWARE's SDC and PaaS Management GE (and its integration within the Cloud Portal as Blueprints) manages the lifecycle of applications and software platforms on top of VMs. Its target users are system administrators that have the knowledge of how applications integrate on top of sets of VMs and want to move from a manual process to an automated, repeatable and predictable way. However:
  - They still need to care about virtual machines, security key pairs, network topology, public IP addresses, firewalls, Linux administration, etc. and in addition, they need to understand the concept of blueprint templates, blueprint instances, tiers and software releases.
  - This approach aims to go in the same direction of Amazon Elastic Beanstalk, which automates the process of creating virtual machines with prepackaged runtime containers, and setup security groups, load balancers, auto-scaling groups and monitoring alarms. Unfortunately, these features are missing in FIWARE Lab Cloud and the user experience is far from being as simple as using Amazon Elastic Beanstalk. This GE provides greater functionality through automation, but increases the entry barrier in terms of skills instead of lowering it. This can be quite frustrating for end users, especially creative SMEs with little resources to invest on highly technical tasks.
  - In addition, the Blueprint solution's design aims to deploy multiple VM instances organized in tiers in a coherent way. This is a problem given the limited amount of resources granted to third parties on FIWARE Lab Cloud (two VM instances for Trial users as of this writing).

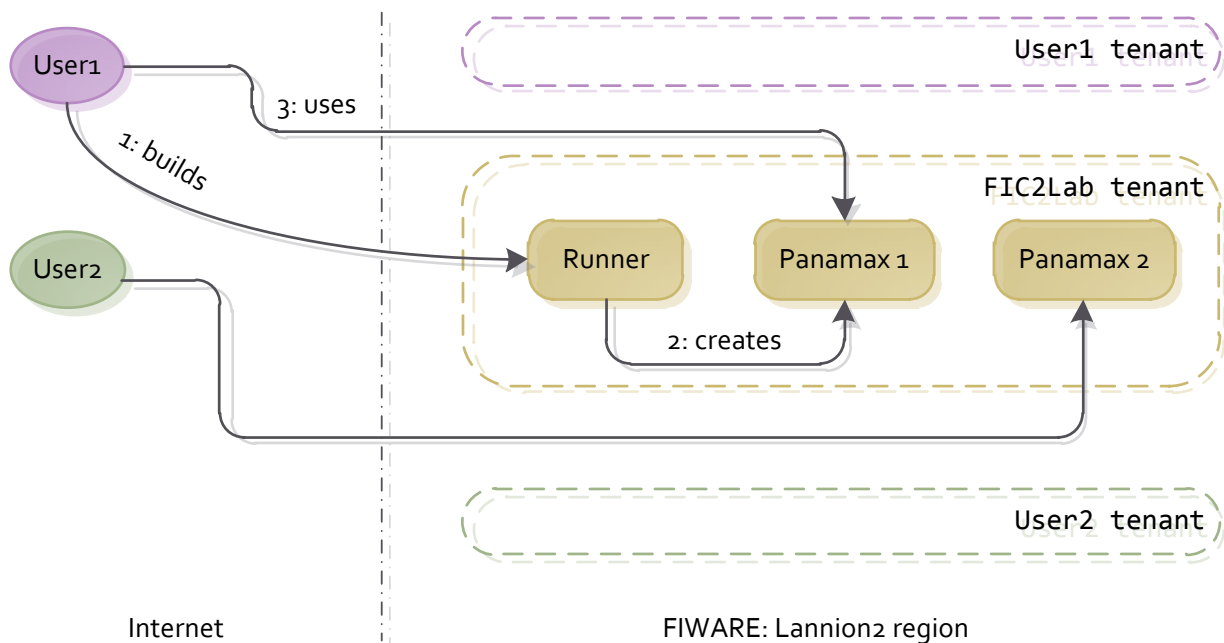
The FIC2Lab taskforce agreed on a common way to package and deploy Specific Enablers. Instead of heavy VM images, we decided that Docker images are much more lightweight and convenient for developers be able to access and try this already virtually installed software quickly.

#### 4.1.2 - High-level architecture

Firstly, the main goal of the FIC2Lab runner is to automate the construction of an environment to run SEs packaged as Docker containers. This automation resides in an elaborate sequence of API calls to the FIWARE Lab's Cloud platform, which is based on OpenStack. These API calls are made on behalf of the user, emulating what he would do on the Cloud Portal or using the OpenStack command-line tools inside a terminal, and doesn't require any administrator rights on the FIWARE Lab infrastructure. Therefore, no server-side component is required. This design choice makes the FIC2Lab runner a client-side application. Everything is done inside the end user's browser (see Figure 4-1) and targets the various remote API endpoints. With this approach, the FIC2Lab application consists solely of a set of static files. For now, these static pages are served by a web server but ideally they could have been deployed to a CDN.

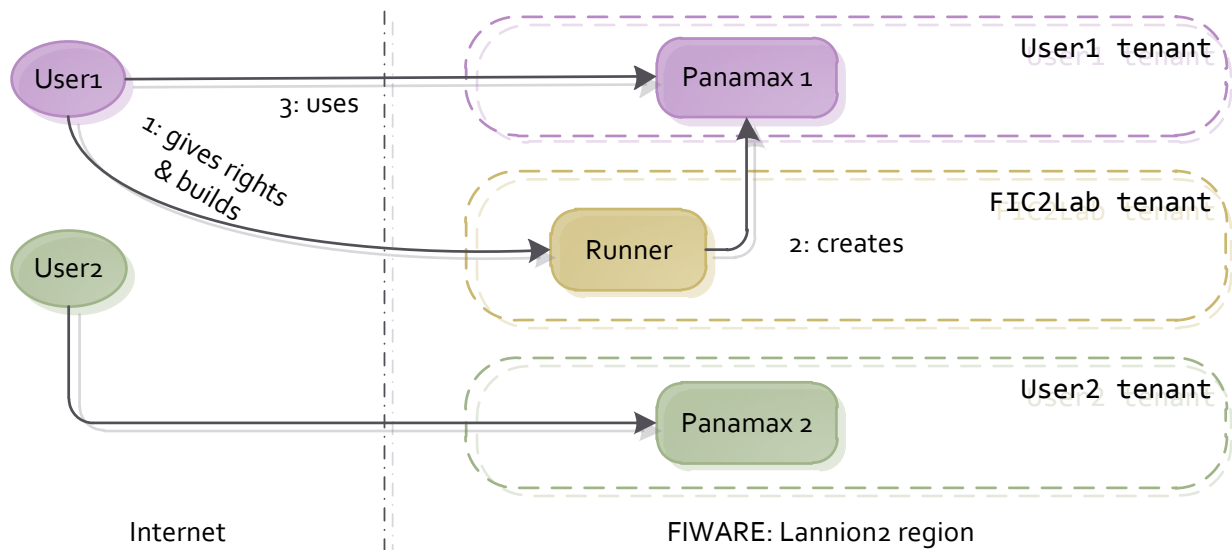
Secondly, there are two possibilities for the architecture of the FIC2Lab runner: the “Middle man” and the “Delegated” setups. The “Delegated” setup was chosen for the actual implementation.

In the “Middle man” setup (see Figure 4-2), the FIC2Lab runner creates every Panamax (SE hosting environment) instance in the FIC2Lab’s tenant. Then, after completion, it provides the end user with a URL to access the Panamax UI. This architecture puts a lot of pressure on the FIC2Lab runner. Any malicious actions taken by an end user will be run inside a Panamax instance owned by the FIC2Lab tenant. As a result, the FIC2Lab will engage its own responsibility. In such an event, a termination of the FIC2Lab account would have jeopardized every end user. More practically, the FIC2Lab tenant couldn’t be dimensioned for adequate quotas as the number of end users is unbound. Finally, the FIC2Lab runner must maintain and save a global state to correctly handle the isolation of each Panamax.



**Figure 4-2 – “Middle man” setup**

In the “Delegated” setup (see Figure 4-3), the FIC2Lab runner “borrows” an end user account to instantiate a Panamax instance. This instance resides inside the end user tenant. All of his actions remain under his own responsibility. This architecture has the benefit to distribute all Panamax instances across multiple tenants and is directly compatible with a multi-region deployment. Moreover, this approach reduces the impact of unused resources because a basic deletion of an end user’s account will terminate the useless Panamax instances.



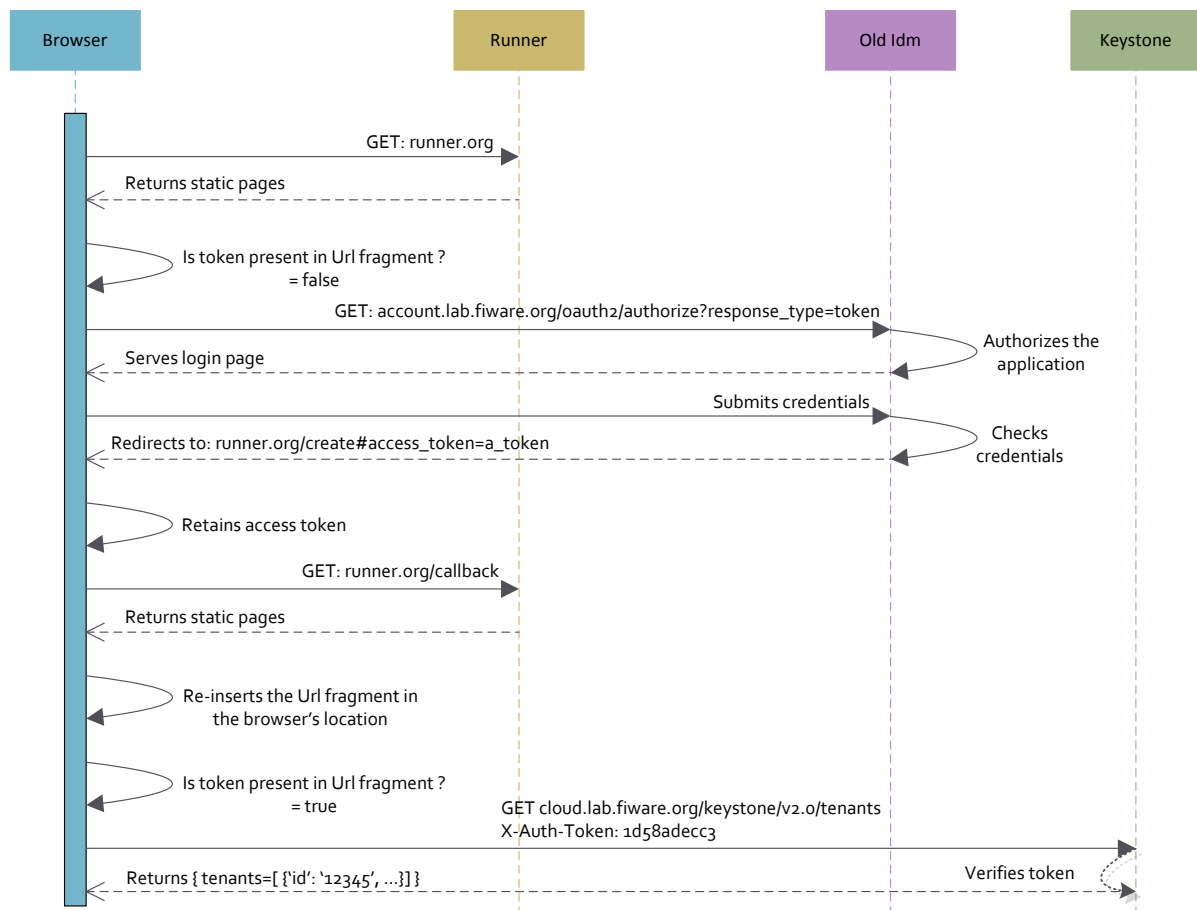
**Figure 4-3 – “Delegated” setup**

#### 4.1.3 - Authorization (using OAuth2 implicit flow)

The FIC2Lab runner requires some information to operate on another user tenant. Directly asking for their credentials can be a security flaw because they may be leaked through a bug. Moreover the end users might be reluctant to trust the application.

The FIWARE platform offers the IDM (Identity Management) component to leverage this problem. The IDM implements the OAuth 2 protocol to enable a third party application (such as the FIC2Lab runner) to obtain a delegated access to FIWARE services on behalf of a user. This process has the benefit to never share the user's credentials.

The OAuth2 protocol proposes several authorization grant flows. The FIC2Lab runner uses the Implicit Grant (see Figure 4-4), which is optimized for clients implemented in a browser using a scripting language. This flow doesn't require any server side computation, simplifying the release and the maintenance of the FIC2Lab runner.

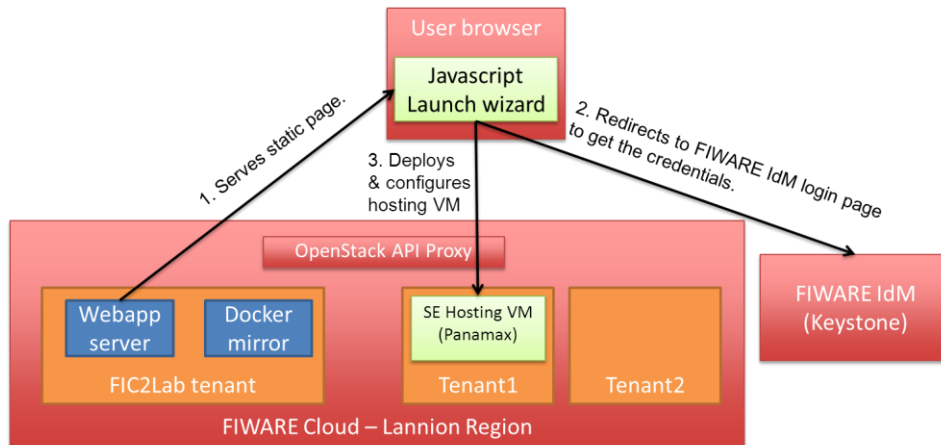


**Figure 4-4 - Runner authentication sequence: Implicit Grant Flow**

#### 4.1.4 - Deployment phase

When a user has fully logged into the FIWARE platform, then the FIC2Lab runner has the credentials required to use FIWARE Cloud to deploy the VM with the SE hosting environment. Using the authentication token acquired through the OAuth workflow, the FIC2Lab runner starts by converting it to a Keystone token. When this task is completed, the FIC2Lab runner is able to leverage usage of the OpenStack API on the user's behalf (see Figure 4-5).

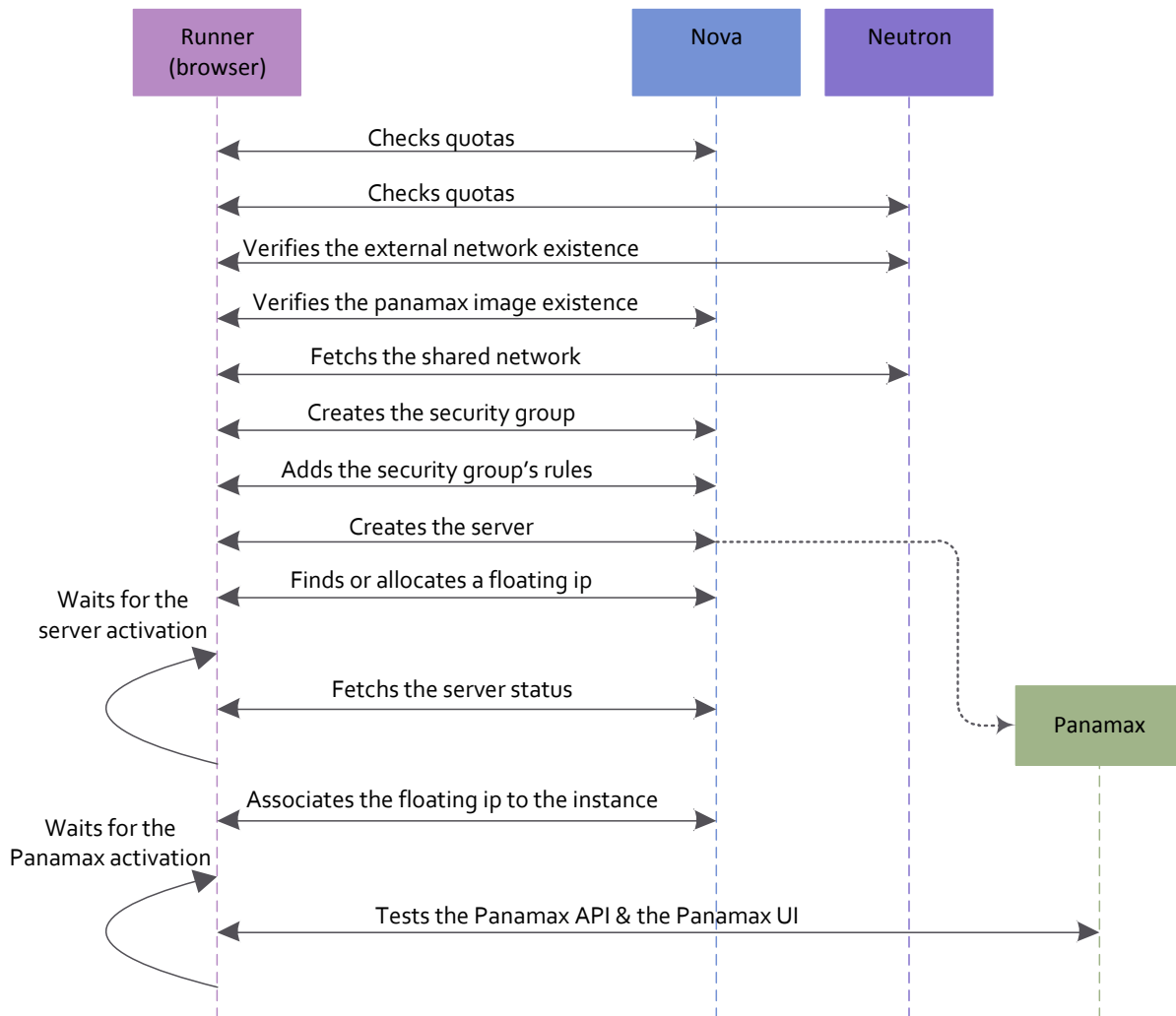
To setup a SE hosting environment, the FIC2Lab will deploy and configure a VM. If this instance is already running (due to a previous deployment) then the FIC2Lab runner will reuse it after a check and a repair phases.



**Figure 4-5 - FIC2Lab Runner architecture (deployment)**

Please consider the sequence diagram (see Figure 4-6) for the main process executed by the FIC2Lab runner in the deployment phase. During this process, the application tries to detect the possible problems, in particular related to the user's account status.



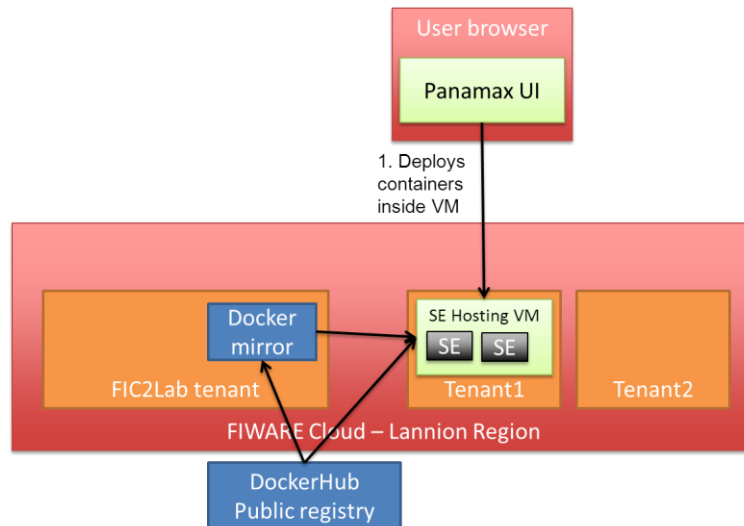


**Figure 4-6 – Automation of the FIWARE OpenStack**

#### 4.1.5 - Running phase

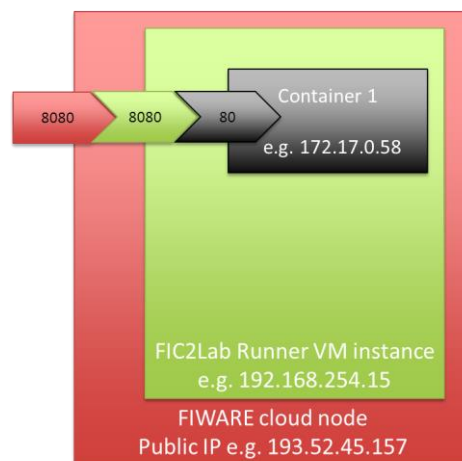
While the SE hosting environment (Panamax) is running, the Runner can be accessed to reuse/repair a previous setup. This process is equivalent to the previous workflow.

During the running phase (see Figure 4-7), the end user accesses and uses the SE hosting environment to create and delete Docker containers running SE instances. They are stored inside a Docker registry (mainly the Docker Hub, but private registries could be used). The container images could take long to download from the remote registry, so a local Docker mirror has been setup. This mirror is a local proxy of the remote registry (in this case inside the Lannion2 region) that accelerates the download of containers and reduces the external traffic of the FIWARE infrastructure.



**Figure 4-7 - FIC2Lab Runner architecture (runtime)**

The Panamax used as the medium for the SE hosting environment has been customized to fit the FIC2Lab project a little more. The user interface was updated to directly display links to the services provided by a container. Another modification was to add the support for publishing ports on a container. Historically, a container started in Panamax was neither exposed (accessible internally) nor published (accessible remotely). The end user was expected to customize and restart a running container to fit his needs. To avoid this manipulation, the support for an automatic publishing was added in Panamax. With this modification, the services provided by a container are directly accessible on the host. Then the FIC2Lab runner is synchronized with this feature to open the required ports on the FIWARE platform, making the services public. The Figure 4-8 shows the 3 levels of ports manipulated when a container runs.

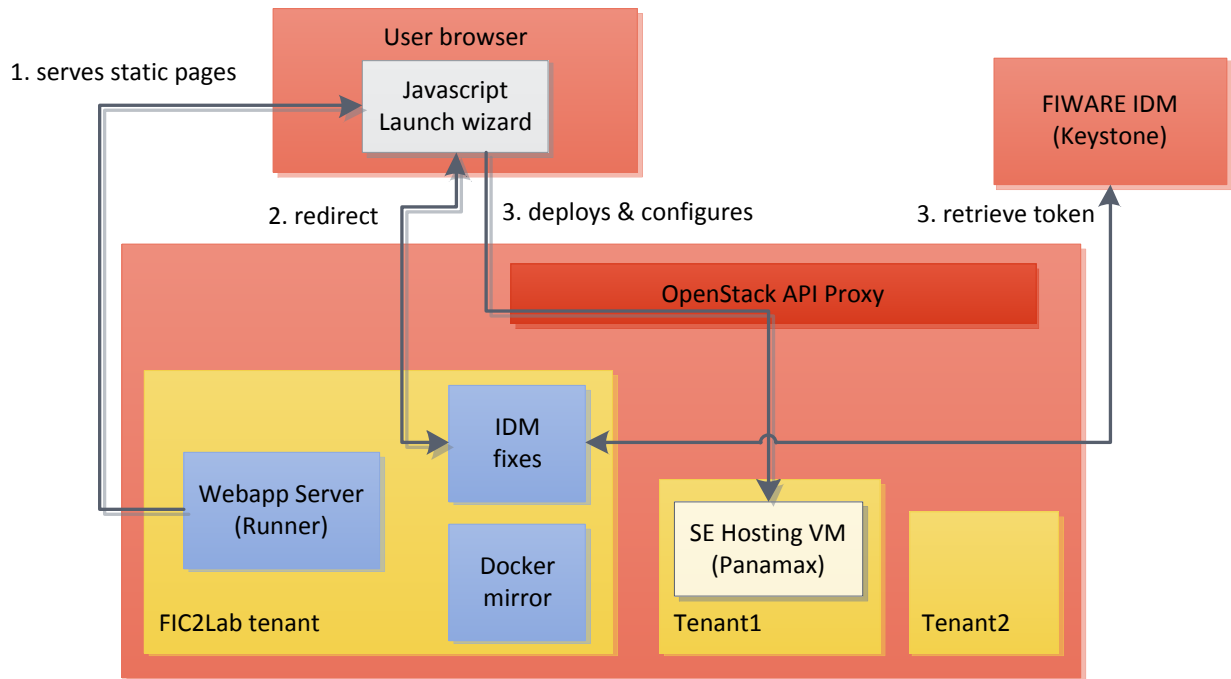


**Figure 4-8 - Port mapping**

#### 4.1.6 - Authorization using an IDM without OAuth2 implicit flow

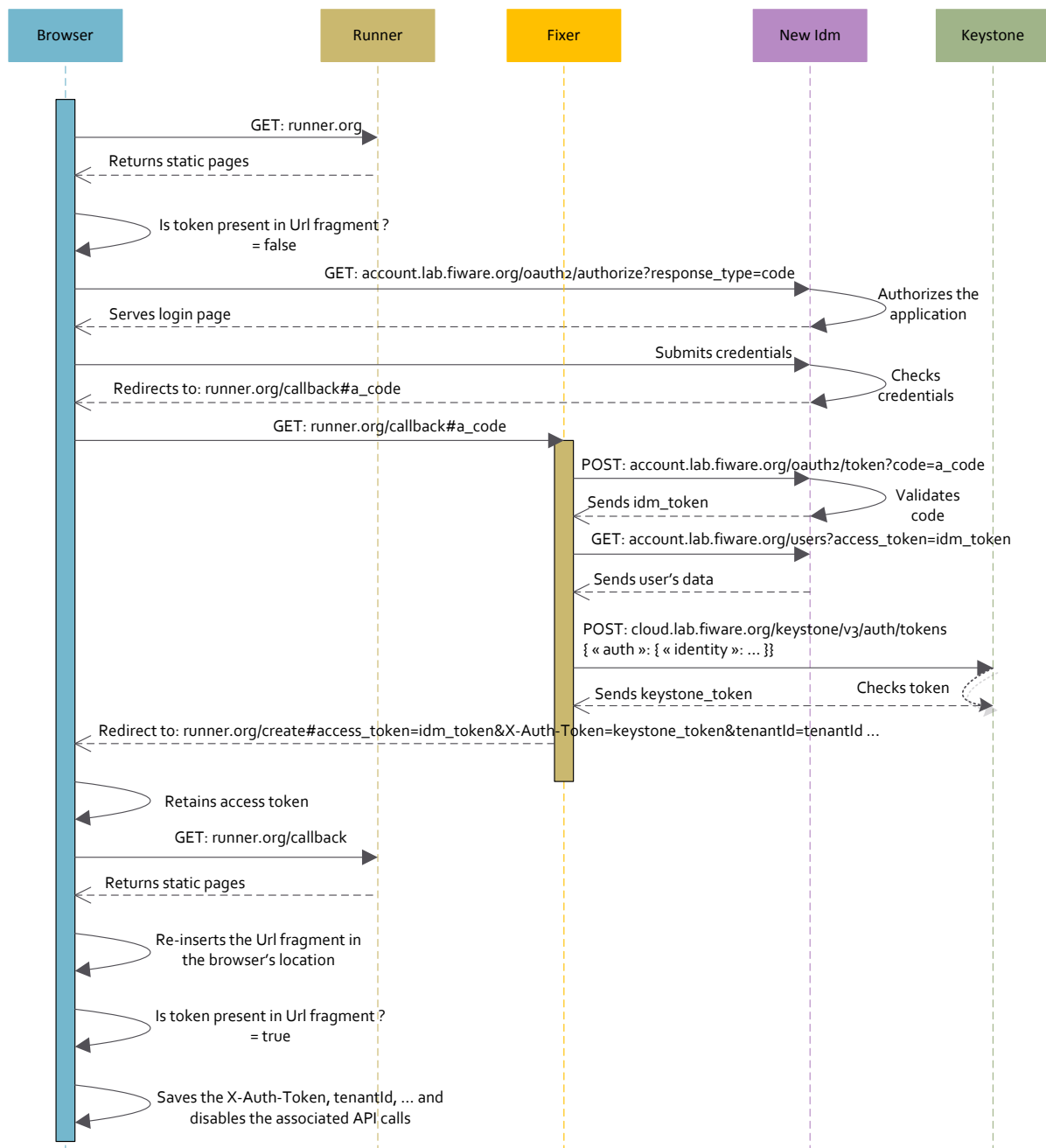
During a period of time after a version upgrade, FIWARE's IDM removed an important feature used by the FIC2Lab runner: the implicit grant flow. Only the classical Authorization grant was supported. This grant requires an active server side component to deal with the manipulation of tokens. The FIC2Lab runner architecture was adapted to change the used grant, even if such a modification was breaking the main runner's aspect: being a full client-side application.

In this architecture, a NodeJS application is added to handle the Authorization grant flow (see Figure 4-9 – Architecture changes for the IDM update). When a user authenticates, the FIWARE IDM gives him an access code. The user transmits this code to the new component. Then this component will verify the code with the IDM to obtain the final authentication token corresponding to the user. Finally the token is sent back to the application mimicking the implicit grant flow.



**Figure 4-9 – Architecture changes for the IDM update**

In addition, in that version upgrade, the IDM update was shipped with more restrictive CORS directives, making the browser block some API calls. The call made by the FIC2Lab runner to retrieve the user's tenant and the call made to convert the IDM token to a Keystone token were impacted. To counter this problem, these calls were relocated to the new component. Being a server, it was out of the scope of the CORS restrictions. The results were inserted inside the response. The FIC2Lab runner's code was modified to remove the problematic calls and retrieve the missing information directly in the fragment URL. The new authentication workflow is showed in the Figure 4-10 – Workflow changes for the IDM update.



**Figure 4-10 – Workflow changes for the IDM update**

## 4.2 - User guide

### 4.2.1 - Introduction

Many of the FI-CONTENT 2 Specific Enablers are server-side open-source software that you can easily reuse to build your own applications. For simplicity, these enablers have been packaged as Docker images that are publicly available. You can run them in your own machine or in the cloud.

## What is Docker?

Docker is an open-source project for shipping and running applications inside lightweight software containers that can run anywhere. Any application that runs on Linux can be packaged as a Docker container.

Docker is very popular, and there are more than 70,000 Dockerized applications at the DockerHub Registry: <https://registry.hub.docker.com/>

## How can I use the Docker images of the Specific Enablers?

You can deploy and manage our Docker images in different ways:

- using command-line tools in their own machine
- using command-line tools in a remote machine in the Cloud (any provider, including FIWARE Lab Cloud)
- using the FIC2Lab runner web-based tool to manage an instance on FIWARE Lab Cloud (best for beginners, limited for advanced users)

Our Specific Enablers are available as public Docker images:

<https://registry.hub.docker.com/repos/fic2/>

**Please note** that this solution does not bind anybody to use Docker, as a Dockerfile is at same time a formal instruction how to install software. So it can be also used just as reference to build own infrastructure without a vendor lock to Docker,

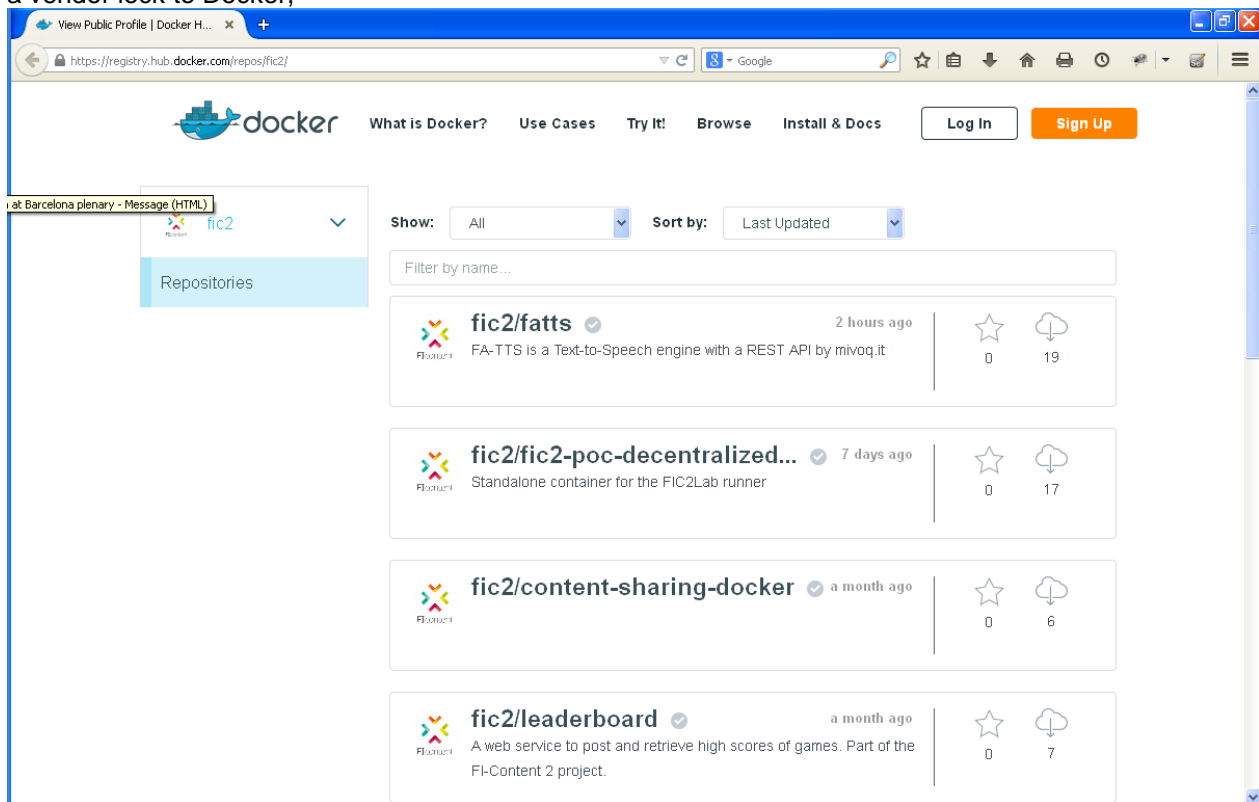


Figure 4-11 - The FI-CONTENT 2 public repository of Docker images

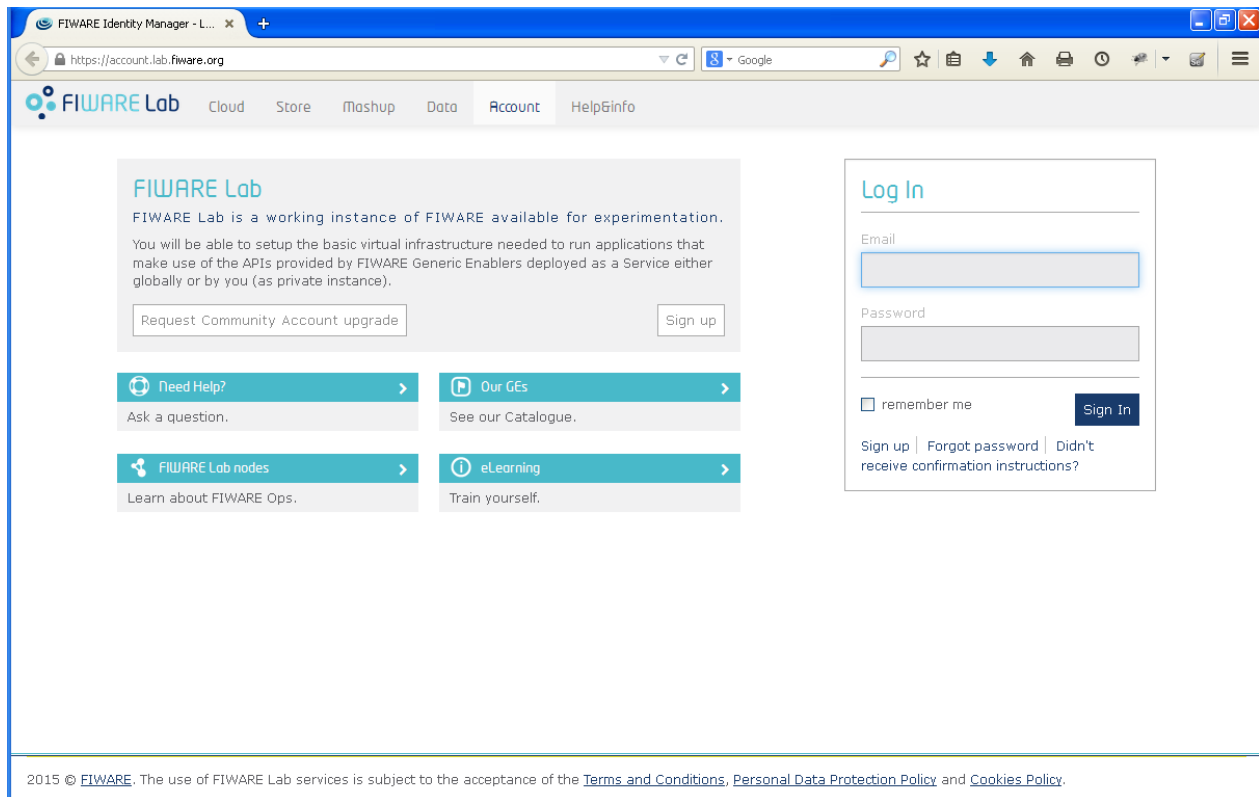
The following subsections will show the different ways in which you can run SEs.

#### 4.2.2 - Run SEs using the FIC2Lab runner web-based tool (on top of FIWARE Lab Cloud)

FIC2Lab runner is a tool that automatically sets up for you a running VM instance with a Docker host and a web UI on top of the FIWARE Lab Cloud infrastructure. You do not need to know anything about the FIWARE Lab Cloud, it will configure all the infrastructure, network, security, etc.

However, you will need to have a FIWARE Lab account and request a community upgrade in the Lannion region. The community upgrade grants you the right to allocate a public IP address within the cloud.

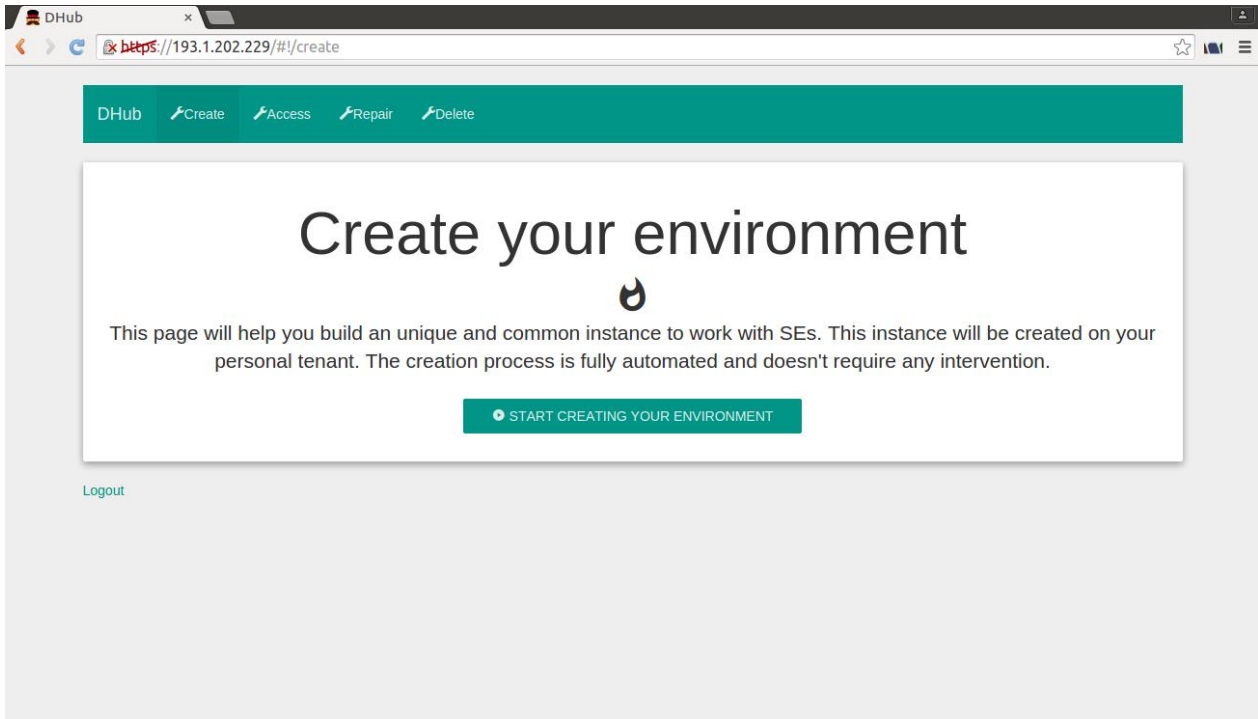
You can request your account and the community upgrade at <https://account.lab.fiware.org>



The screenshot shows the FIWARE Lab portal at <https://account.lab.fiware.org>. The page has a navigation bar with links: Cloud, Store, Mashup, Data, Account, and HelpGinfo. The main content area is divided into two columns. The left column features a 'FIWARE Lab' section with a description: 'FIWARE Lab is a working instance of FIWARE available for experimentation. You will be able to setup the basic virtual infrastructure needed to run applications that make use of the APIs provided by FIWARE Generic Enablers deployed as a Service either globally or by you (as private instance).' Below this is a 'Request Community Account upgrade' button and a 'Sign up' button. Further down are four links: 'Need Help?' (Ask a question), 'Our GEs' (See our Catalogue), 'FIWARE Lab nodes' (Learn about FIWARE Ops), and 'eLearning' (Train yourself). The right column contains a 'Log In' section with fields for 'Email' and 'Password', a 'remember me' checkbox, and a 'Sign In' button. At the bottom of the right column are links for 'Sign up', 'Forgot password', and 'Didn't receive confirmation instructions?'. The footer of the page states: '2015 © FIWARE. The use of FIWARE Lab services is subject to the acceptance of the Terms and Conditions, Personal Data Protection Policy and Cookies Policy.'

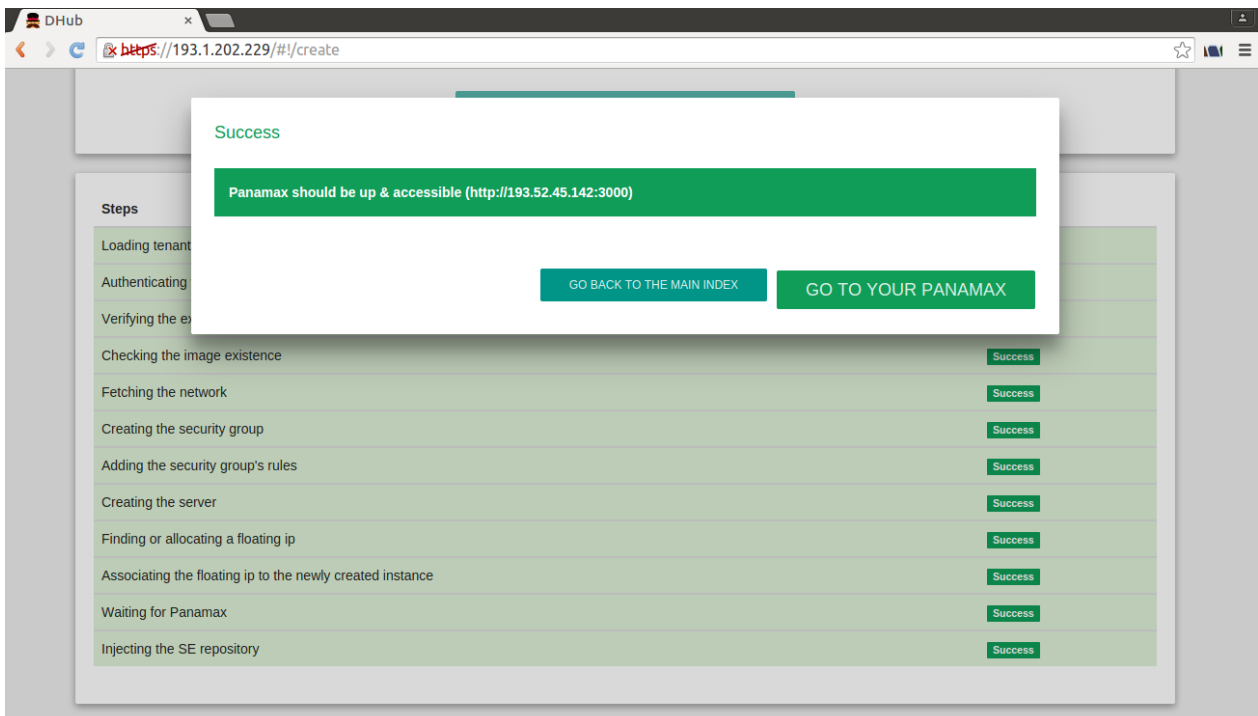
**Figure 4-12 - FIWARE Lab portal to sign up and request community upgrade**

Once you have received a confirmation of your upgrade to a community account, you can go to the FIC2Lab runner wizard at <https://runner.developer.mediatech.org> where the system guides you through the steps to create your own run environment.



**Figure 4-13 - FIC2Lab runner (beta) - Landing page**

Once you click on the button to instantiate your environment, the wizard will go through a number of steps that set up everything that you need on FIWARE Lab Cloud. If everything goes right, you should see a pop-up with the URL to your dedicated management console.



**Figure 4-14 - FIC2Lab runner (beta) - Deployment wizard**

Some of the things that can go wrong are:

- You need to log in with your FIWARE Lab account.

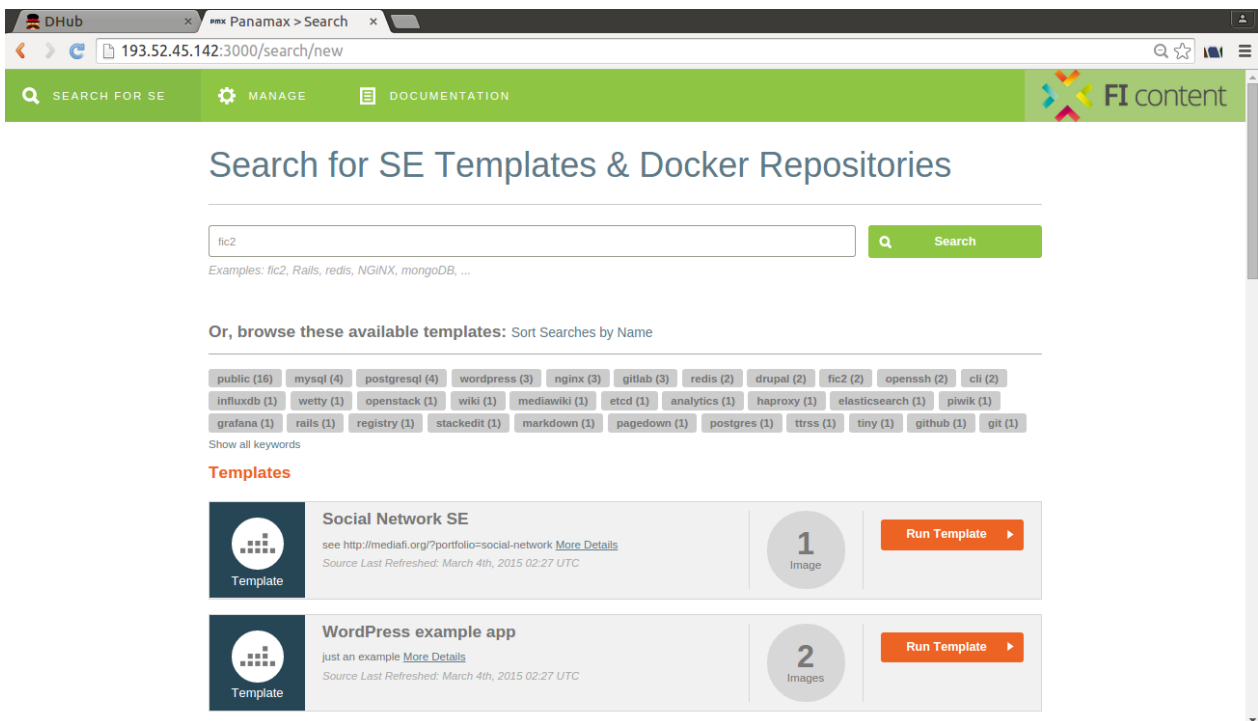
- You do not have a community account in the selected region.
- Your quota of public IP addresses or VM instances is full.
- There is an issue in the selected region. Try again in a few minutes or file a ticket.

Once you go to your dedicated Panamax management console, you will be able to deploy and manage applications that will be publicly available online, listening on different ports of your public IP address.

There are two types of artifacts developers can deploy and access in the UI:

- **Images:** individual Docker images. When deployed, they are **services**.
- **Application Templates:** linked sets of Docker images. When deployed, they are **applications**, composed of one or more **services**.

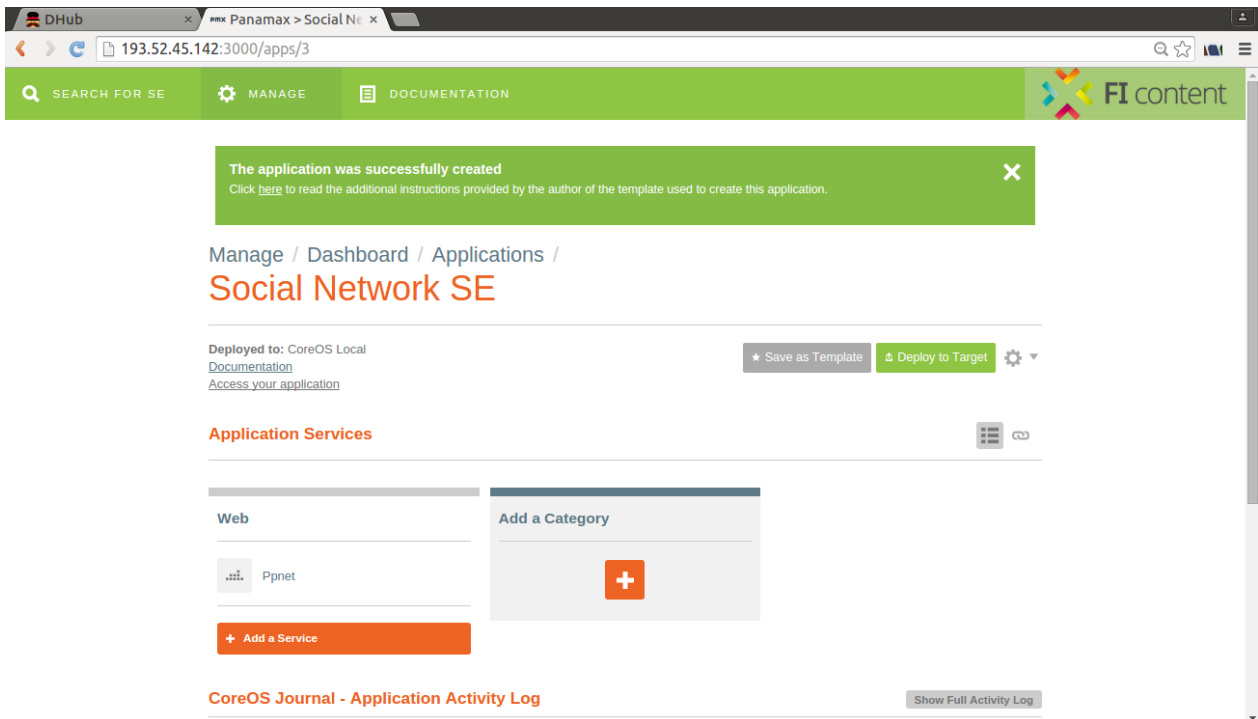
The user interface provides a built-in search capability that can search any public Docker image repository through Docker Hub, including the FI-CONTENT2 repository. It will search as well public Application Templates from Panamax.io. You can also select from the list the featured templates and images.



**Figure 4-15 - FIC2Lab runner (beta) - User-dedicated management console**

Let us select the Social Network SE and click on “Run Template”. This will trigger under the hood the download and launch of an application with one image: “fic2/ppnet”. It will open the management dashboard for this application and show a spinning wheel that means that the deployment is taking place.





**Figure 4-16 - FIC2Lab runner (beta) - Social Network Enabler SE launched from the console**

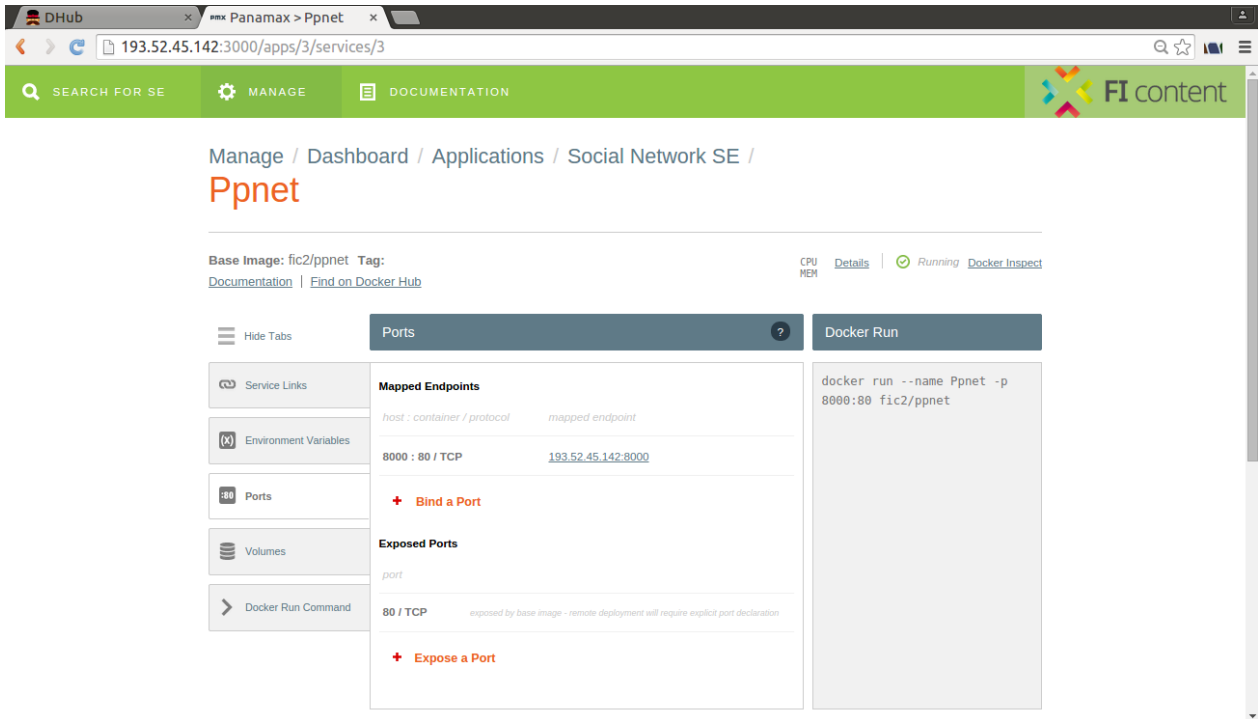
Categories help you structure your application in tiers. If you are creating a multi-tiered application using multiple Docker images, you can add more images to your application here.

At the bottom of the screen, the application activity log shows the detailed status of your application (the CoreOS Journal).

You can manage the settings of the services within your application. Let's click on the only service deployed in this application: ppnet. From this page, you can:

- Bind ports
- Link services
- Set environment variables
- Mount host volumes
- Pass arguments to the services entry point

For convenience, all the exposed ports of a Docker image are bound to the outside in a randomly generated port (in the 49000-50000 range). You can change that in the "Ports" tab of the service configuration, as shown below.



**Figure 4-17 - FIC2Lab runner (beta) - Changing port mapping of the SNE SE instance**

### 4.2.3 - Run SEs in your own machine

We packaged our software as a series of standard Docker images. You can easily launch them from a terminal in your own machine.

If you are familiar with Docker, you can find the Docker images and the instructions in the FIC2Lab DockerHub repository: <https://registry.hub.docker.com/repos/fic2/>

#### **Installing Docker**

Docker runs natively on any recent Linux. You can use it on Windows and Mac OS X using boot2docker, which installs VirtualBox with a guest Linux for you.

To get it running on your machine, follow the Docker installation instructions:

<http://docs.docker.com/installation/>.

Under Mac OS X, you can use Kitematic, a nice user interface (also installs VirtualBox with a guest Linux):

<http://kitematic.com>

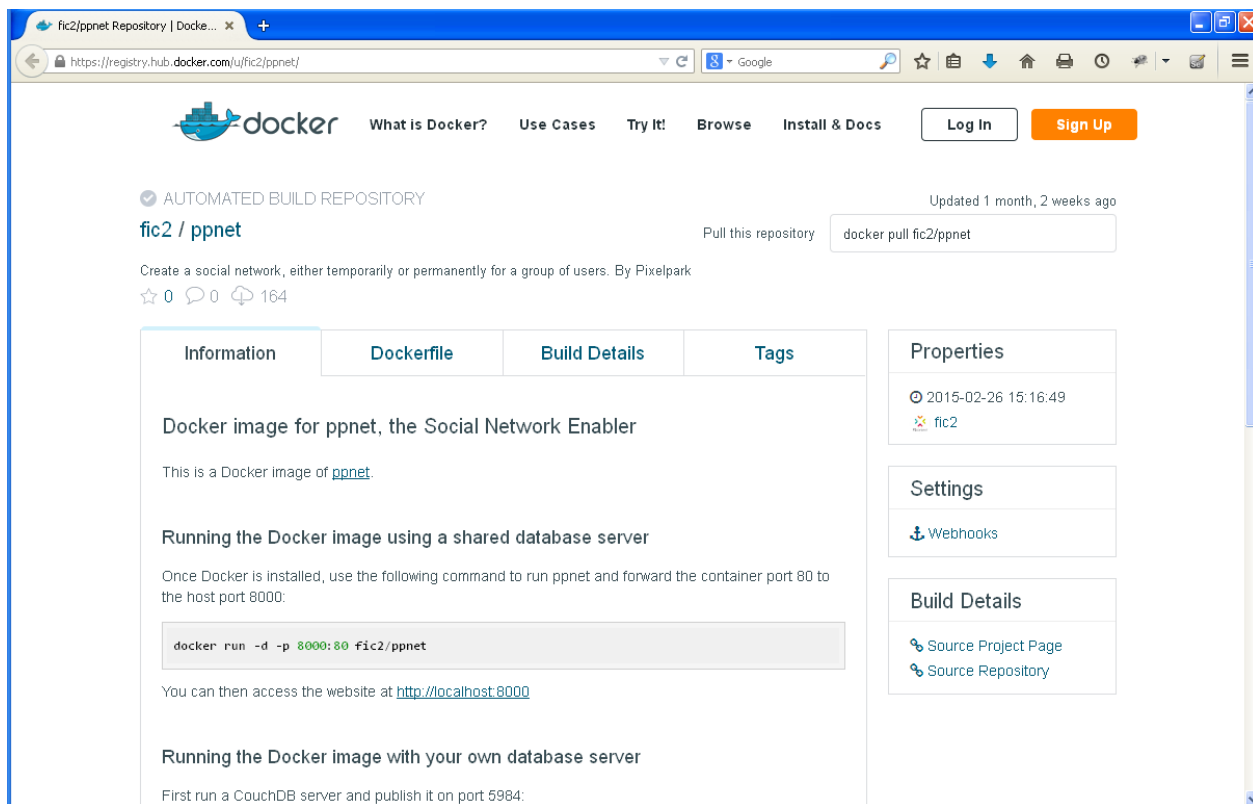
You can also deploy it to different Cloud providers easily using Docker Machine:

<https://docs.docker.com/machine/>

#### **Running a SE from the command line**

Find your SE on the public FI-CONTENT 2 repository <https://registry.hub.docker.com/repos/fic2/>

Then follow the instructions there. For instance, here are the instructions to run the Social Network SE:



**Figure 4-18 - DockerHub page of ppnet, the Social Network Enabler**

Open a terminal and type:

```
docker run -d -p 8000:80 fic2/ppnet
```

This will download the Docker image for ppnet, run it in a Docker container and forward the container port 80 to the host port 8000:

```
tal@Parme: ~
tal@Parme:~$ docker run -d -p 8000:80 fic2/ppnet
Unable to find image 'fic2/ppnet:latest' locally
Pulling repository fic2/ppnet
b454781957ed: Pulling dependent layers
511136ea3c5a: Download complete
a5b60fe97da5: Download complete
390a00bdb439: Download complete
af7c9ae40363: Downloading 1.867 MB/2.019 MB 0
```

**Figure 4-19 - Docker CLI - Downloading Social Network SE**

Once the download is complete, the container will be up and running and we will get a container ID in the console.

```

tai@Parme: ~
tai@Parme:~$ docker run -d -p 8000:80 fic2/ppnet
Unable to find image 'fic2/ppnet:latest' locally
Pulling repository fic2/ppnet
b454781957ed: Download complete
511136ea3c5a: Download complete
a5b60fe97da5: Download complete
390a00bdb439: Download complete
af7c9ae40363: Download complete
84cde1578c13: Download complete
d81ff9276ee7: Download complete
9520277322c9: Download complete
Status: Downloaded newer image for fic2/ppnet:latest
86d7fb126ee54ece80e83c826a58ab4ca13ef2e29d2cad35c18eb6af86c2df1d
tai@Parme:~$

```

**Figure 4-20 - Docker CLI - Social Network SE downloaded and running**

To check all the running containers, type:

```
docker ps
```

```

tai@Parme: ~
tai@Parme:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
86d7fb126ee5       fic2/ppnet:latest  "/usr/local/bin/run. 44 seconds ago     Up 43 seconds      0.0.0.0:8000->80/tcp mad_goldstine
tai@Parme:~$

```

**Figure 4-21 - Docker CLI - Checking running containers**

You will notice the port redirection from host machine (0.0.0.0) port 8000 to container port 80. Docker generates a random container name if you choose to specify none.

#### 4.2.4 - Run SEs using command-line tools in a remote machine in the FIWARE cloud

You can follow the steps in the previous section on any Linux machine with recent kernel (from 3.10) in the FIWARE cloud.

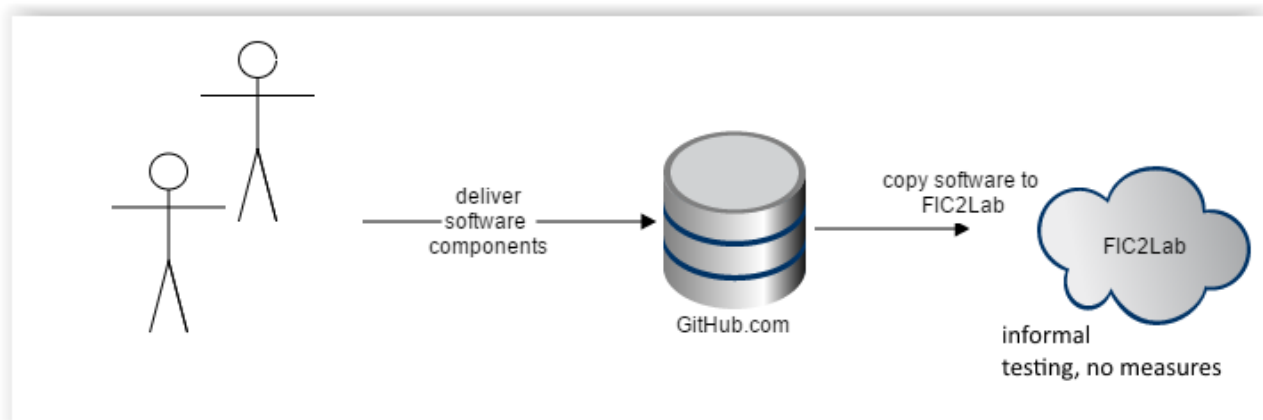
## 5 - PACKAGING AND DELIVERY OF SERVER-SIDE SES

### 5.1 - Initial situation

FIC2Lab is technically seen a pool of software components (SEs). Their authors published their source code, or in some cases only executable package on GitHub.

As FIC2Lab is an interactive online resource, providing as described in Chapter “2 -Discover and try SEs” interactive use cases TWEAK, TRY and RUN for technically interested users. “Interactive” requires in this context a live SE execution environment for a private user session. The primary execution environment is thus, for most SEs, a web browser of a person accessing an SE.

A naïve approach to deliver SEs to the FIC2Lab server (see Figure 5-1) would assume that there should be a quite simple way to copy delivered software from GitHub over to the FIC2Lab system, which would be enough. This assumption does not concern, however, that in case of diversity and system dependencies of the SEs the delivery process could be more complex than that.



**Figure 5-1 – Packaging and delivery – a naïve approach**

The naïve approach implies but not ensures also a structurally simple and non-mandatory quality assurance process:

- System integration: after copying of files to a specified location on disk of a FIC2Lab server or after any change of an SE, somebody might want to check that all SE files have been copied.
- Functional testing: at least a quick visual check after the copy step, which is in informal environments the most common way of testing software, being however not simply measurable.

### 5.2 - Problem analysis

FIC2Lab is actually a typical example of complex modern software delivery and execution environments where many parties provide software components with diverse software architecture, implying different workflows to put software into operation.

An architectural survey of the available SEs revealed that the following types of them are available in FICONTENT2:

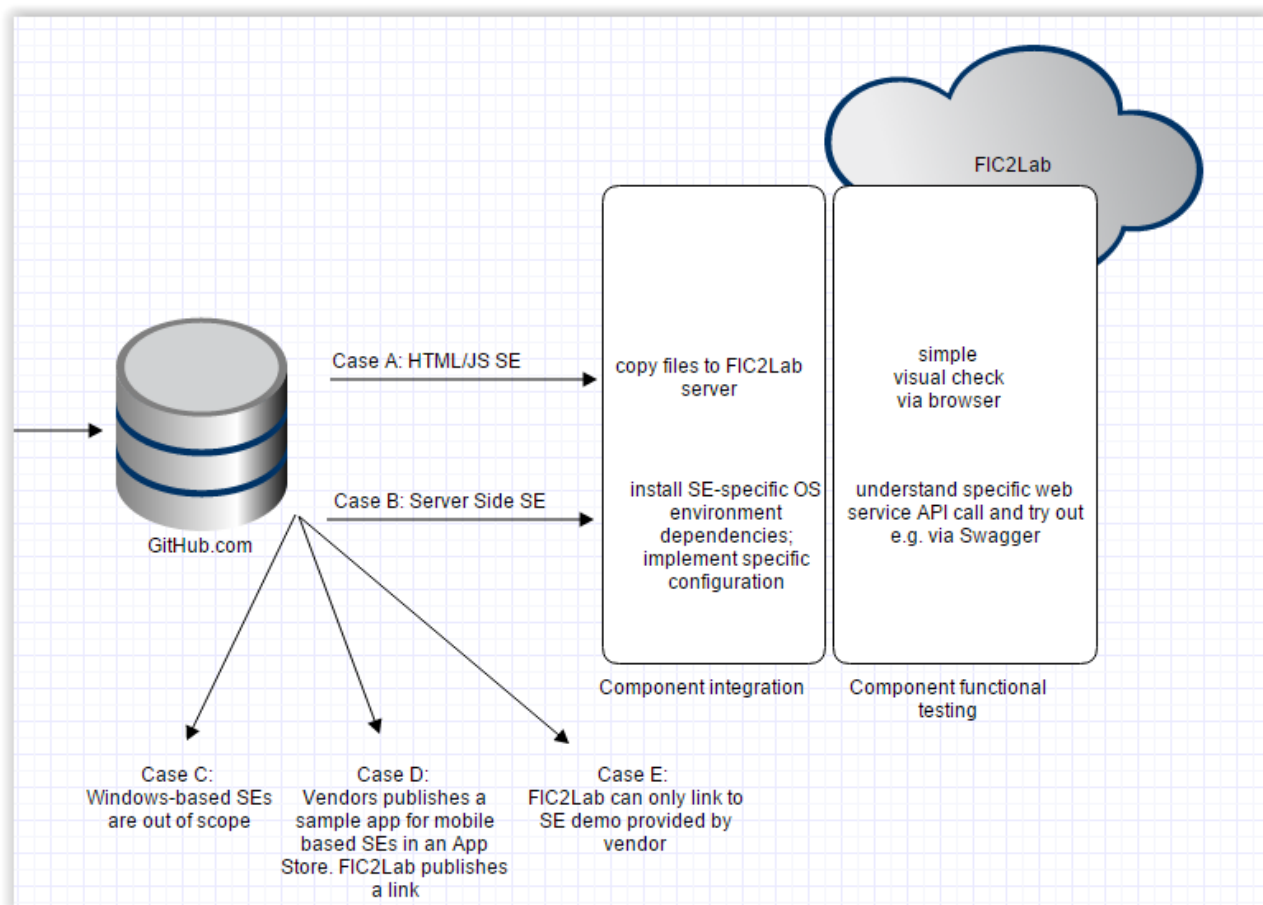
- Interactive web page setups based on classic HTML and Javascript code; (A)
- Server-side software providing web service APIs, runnable in a Linux environment (B)

- Server-side software runnable in Windows environment only (C)
- Mobile OS based software runnable on iOS or Android (D)
- Vendor would like to link his SE via FIC2Lab without providing a copy of software (E)

As we can see, the simple delivery process described above covers only the case A, which is generally compatible with the Playground solution. A simple quality check is trivial; it is enough to check if the UI of an SE loads – assumed that the vendor has performed a more in-depth quality control.

Cases C and D has been decided to be out of scope for FIC2Lab representing only smaller groups of SEs but introducing questions about Windows licenses for FIC2Lab or about the technical challenge to build a mobile version of FIC2Lab. Actually, the pragmatic solution for case D is to use the corresponding App Store and follow the scenario of case E (just putting a link on FIC2Lab). Case E is a content management task and therefore not relevant in this context.

However, delivery of components in case B is not trivial, because server side software requires specific system environment components and their configuration, which is not only additional work, but also a subject to change. As you can see in Figure 5-2, case B requires more communication to clarify possible problems. Also testing using the FIC2Lab demonstration approach via Swagger interface is not trivial, because in order to test a web service API you need to understand at least one or two specific call procedures.



**Figure 5-2 – Packaging and delivery of SEs – Problem analysis**

Please consider the complexity levels between component delivery cases A and B:

- In case of Specific Enablers (A), which are frontend components viewable in a web browser:

- More or less standard runtime environment is generally available, assumed SE vendor supports cross-browser compatibility;
- A manual simple quality check is possible;
- In case of software errors, communication and retesting is quite simple, additionally to an error screenshot it is enough to communicate browser product name, version and operating system name and version.
- In case of Specific Enablers (B), implemented as server-side software:
  - No standard operation environment configuration is available for all Specific Enablers;
  - Even simple quality checks require:
    - Detailed up-to-date documentation of all required environment components and configuration;
    - successful installation and operation of SE;
    - at least minimal knowledge/understanding of SE's API to execute test call;
  - in case of a software error the vendor needs to be sure that FIC2Lab has properly followed the installation and configuration instructions;
  - problem analysis is not trivial as it involves knowledge of software components, on which an SE depends, and log analysis as well as much communication with the vendor to understand what needs to be changed;
  - use of one FIC2Lab installation environment might cause SE dependency / SE interoperation problems; whereas use of many installation environments increases required effort to manage and maintain these resources;
  - Additionally, not all SEs support multiple tenants, which means, SE operation on the same environment in FIC2Lab could corrupt other sessions as soon as more than one user accesses the same SE on FIC2Lab.

We can conclude that delivery and operation of server side SEs are a complex process, which can be time-consuming and not easy to handle. However, these challenges are not new in the software industry.

To face them, an existing systematic IT process management methodology can be used, which provides standardized ways and best practices of managing software delivery and operation processes beyond the software development processes. We introduce this approach in the next section.

### 5.3 - Methodology and systematic problem assessment

One of the existing industrial methodologies, providing approaches to solve complex problems in large-scaled software delivery, is ITIL (IT Infrastructure Library), featured as *"a set of practices for IT service management (ITSM) that focuses on aligning IT services with the needs of business. [...] ITIL describes processes, procedures, tasks, and checklists, which are not organization-specific, but can be applied by an organization for establishing integration with the organization's strategy, delivering value, and maintaining a minimum level of competency. It allows the organization to establish a baseline from which it can plan, implement, and measure. It is used to demonstrate compliance and to measure improvement."*<sup>1</sup>

In fact, ITIL is a relatively large and complex management framework geared towards steering value delivery in very large distributed IT organizations. At first glance, this appears to be a serious constraint in terms of over-engineering and corresponding overhead risks. However, FIC2Lab is in fact distributed, and as of this writing, it spans over more than 10 distributed software delivery teams. Thus, our working assumption has

---

<sup>1</sup> Source: Wikipedia.org (as of June 20th 2015) <https://en.wikipedia.org/wiki/ITIL>

been that in this context we can borrow those best practices, which focus on those industry-proven methods for implementing and managing the quality of the IT delivery process.

We will use a series of terms<sup>2</sup> from ITILv3 standard language in order to approach the description and analysis work regarding the delivery of server-side Specific Enablers in a formal way. Then, we will put these terms in the FIC2Lab context and show how FIC2Lab implements an engineered solution based on state of the art industrial IT operations best practices.

Term	ITIL definition	FIC2Lab context
Service	A Service provided to one or more Customers, by an IT Service Provider. An IT Service is based on the use of Information Technology and supports the Customer's Business Process. An IT Service is made up from a combination of people, Processes and technology and should be defined in a Service Level Agreement.	As our context is delivery of server side SEs, it is the IT Service in question, and the Business Process is operation of these SE at FIC2Lab internally, and externally presentation of these SEs to Phase3 members or any other online FIC2Lab visitor.
SLA	Service Level Agreement is a formal, negotiated document that defines (or attempts to define) in quantitative (and perhaps qualitative) terms the service being offered to a Customer.	As no formal SLAs are available, FIC2Lab has derived these based on delivery requirements, which follow in further assessment below.
Incident	An event which is not part of the standard operation of a service and which causes or may cause disruption to or a reduction in the quality of services and Customer productivity.	For some reason, an SE does not work in FIC2Lab environment as expected.
Problem	A cause of one or more Incidents.	In FIC2Lab, we have three general classes of problems: a) Delivered SE is corrupt b) Integration process has not been successful c) FIC2Lab operation environment unstable itself
Change Management	The Process responsible for controlling the Lifecycle of all Changes. The primary objective of Change Management is to enable beneficial Changes to be made, with minimum disruption to IT Services.	
Change	The addition, modification or removal of anything that could have an effect on IT Services.	Problem of class a) requires changes to the corrupt SE, which implements a re-run of the delivery process and quality checks.
Request for Change	A formal proposal for a Change to be made.	Implementing a Change is in responsibility of SE vendor. However, as described above in the complexity analysis, specifying and managing a Request for Change implies

<sup>2</sup> Source: [http://www.knowledgetransfer.net/dictionary/ITIL/en/IT\\_Service.htm](http://www.knowledgetransfer.net/dictionary/ITIL/en/IT_Service.htm) (as of June 22th 2015)



		analysis and communication efforts for all parties.
Continual Service Improvement	Continual Service Improvement is responsible for managing improvement to IT Service Management Processes and IT Services. The Performance of the IT Service Provider is continually measured and improvements are made to Processes. IT Services and IT Infrastructure in order to increase Efficiency, Effectiveness, and Cost Effectiveness.	In context of FIC2Lab, implementing SEs as runnable software artifacts delivered by SE vendors. In case of repeated Requests for Changes across many SEs, the question arises how to keep problem analysis and communication efforts as low as possible.

In the next table, the workflow steps of the process required to deliver a server side SE are presented and corresponding SLAs are defined.

Workflow step	Assessment	SLA / QA objectives
Collect available SE releases from corresponding GitHub repositories every time a Change has been made	As FIC2Lab team and SE vendors work under high time pressure and partly over different time zones, there is a need to decrease time spans between availability of new releases and their availability. Communication of new releases via email is a time cost factor and a considerable effort for driving a systematic approach because of many parties.	FIC2Lab delivery process needs to address challenges of irregular email communication in distributed teams to access new releases as soon as possible e.g. same business day or sooner, and in a plannable manner. In case of emergency, 24/7 delivery capability must be available (because of hard delivery deadline and risk of escalations). (SLA1)
Install SE on FIC2Lab (system integration)	Due to architectural diversity of components, assessment of all SE related incidents is a considerable work force effort. In addition, a corrupt SE version introduces a potential risk to FIC2Lab system. This risk adds across the variety of SEs.	Delivery process has to address architectural diversity and connected risks and to enable FIC2Lab to run one generic integration process for any server side SE in a standard secure environment. Communication regarding SE related incidents also has to be a generic and simple, repeatable and transparent process. This type of operation is called "system integration testing". (SLA2)
Functional check	FIC2Lab operators do not possess enough knowledge of product features of all server sides in question.	Enable FIC2Lab operators to deploy any number of SEs without understanding how they work in detail. Also here a generic and transparent process is required. (SLA3)
Repeatability of the process	The delivery process needs to be repeatable anytime as well for FIC2Lab operators as also	Reduce manual operation steps to a pragmatic minimum. While this is a measurable optimization goal, for

	for any FIC2Lab user, because an easy component installation is one of essential FIC2Lab requirements.	simplicity purpose FIC2Lab sets on full automation of delivery management where process automation appears to make sense based on our experience.(SLA4)
--	--	---

## 5.4 - Solution design and architectural concept

On a conceptual high-level, the delivery process implementing the SLA described above, looks like the following (Figure 5-3). The FIC2Lab team identified the following requirements for the delivery process.

- [Requirement R1] Event- or time-based monitoring and collection of SE releases from GitHub, or for proprietary components from a private FTP server; (SLA1)
- [Requirement R2] Setup of a standard and secured (i.e. isolated from FIC2Lab productive environment) execution environment for generic system integration and generic functional testing (so called “smoke test”<sup>3</sup> meaning a basic operational check of a component) (SLA2, SLA3)
- [Requirement R3] SE components qualified by the smoke test are stored together with the automated installation routine in a repository, which can be accessed by an FIC2Lab operator or in case of public component anybody having access to the Internet.(SLA4)
- [Requirement R4] (not depicted) in case of smoke test error, SE vendors receive test protocol citing occurred errors which is in this case a conventional Request for Change meaning “please fix this issue and deliver an update your latest SE version so that FIC2Lab can retest and accept your component.” (SLA1)

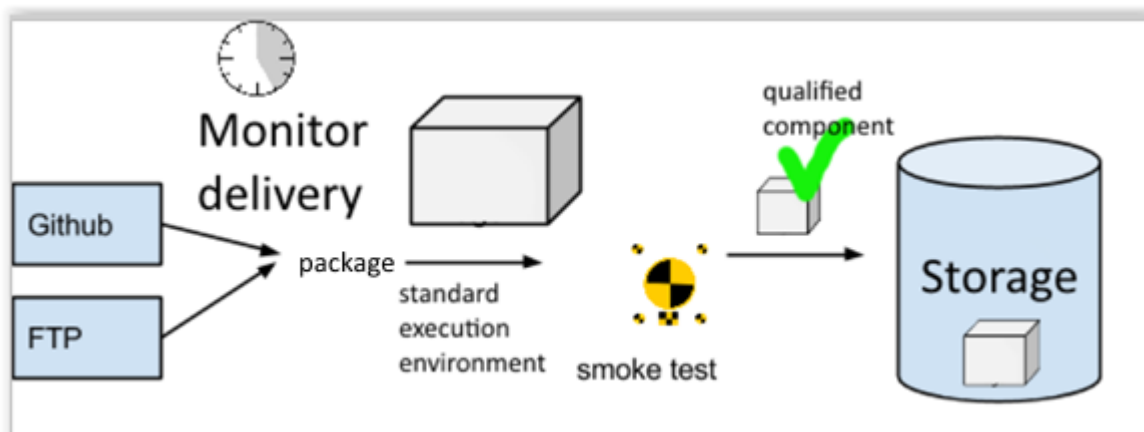


Figure 5-3 – Delivery and QA concept for server side SEs

## 5.5 - Implementation

A state of the art practice in complex software environments is to apply the method of *Continuous Integration* (CI), meaning, “integrating more than once per day, perhaps as many as tens of times per day”, which has recently evolved to *Continuous Delivery*. “[It] extends CI by making sure the software checked in on the

<sup>3</sup> Source: [https://de.wikipedia.org/wiki/Smoke\\_testing](https://de.wikipedia.org/wiki/Smoke_testing) (as of June 22th 2015)

mainline is always in a state that can be deployed to users and makes the actual deployment process very rapid.<sup>4</sup>

The Continuous Integration method is in a software delivery environment a very effective enabler of the ITIL Continual Service Improvement process, as this can help to run acceptance checks many times a day and drive the delivery loop.

A necessary technical side note in this context is to say that in the context of FIC2Lab for most Specific Enablers it is not necessary to merge different sources per SE together as most SE vendors publish packaged software artefacts. However, in some cases FIC2Lab takes SE sources and performs the classical integration process. On the other hand, we can also define the integration goal mainline to be “a set of stable, quality-proven server side Specific Enablers”.

A typical continuous integration tool is a batch job scheduler and runner, providing access to configuration and managing batch jobs in a more convenient and abstract manner as e.g. common operating system scheduled Cron jobs<sup>5</sup>.

Well-known and freely available software CI products are Hudson<sup>6</sup> and Jenkins<sup>7</sup>, which both implement features to enable the following requirements:

- Release monitoring [R1] and batch job automation are core features of these tools per definition;
- Available extensions support configuration and delivery of automated job execution reports [R4]

FIC2Lab uses Jenkins, as Hudson is a commercial trademark owned by Oracle, and Jenkins is a trademark owned by a public domain organisation (SPI). From a technical point of view, there is in this context no considerable difference.

The solution to Requirements R2 and R3 is a system virtualization solution. The overall FIC2Lab design already includes this aspect by using a Linux-native virtualization tool called Docker.

As explained in Chapter 4 -, Docker provides isolated virtual machines (containers) which reduce the consequences of broken installation to be no more a problem: it is much cheaper to rerun a virtual installation on a clean virtual environment template than to assess and repair a corrupt system. Docker uses a simplistic generic language (about 20 generic instructions<sup>8</sup>) to define virtual environments. Since many Open Source products provide reference virtual environments specified via Docker, this has been a good argument for FIC2Lab to define the availability of a functional SE Dockerfile written by the SE developer to be the acceptance criteria for a Specific Enabler.

Another extension to this setup is to have functional acceptance criteria. Therefore, all SE developers have to deliver a functional test script for their SE. This test script should perform a basic functional test in order to make sure that the SE is performing as expected.

---

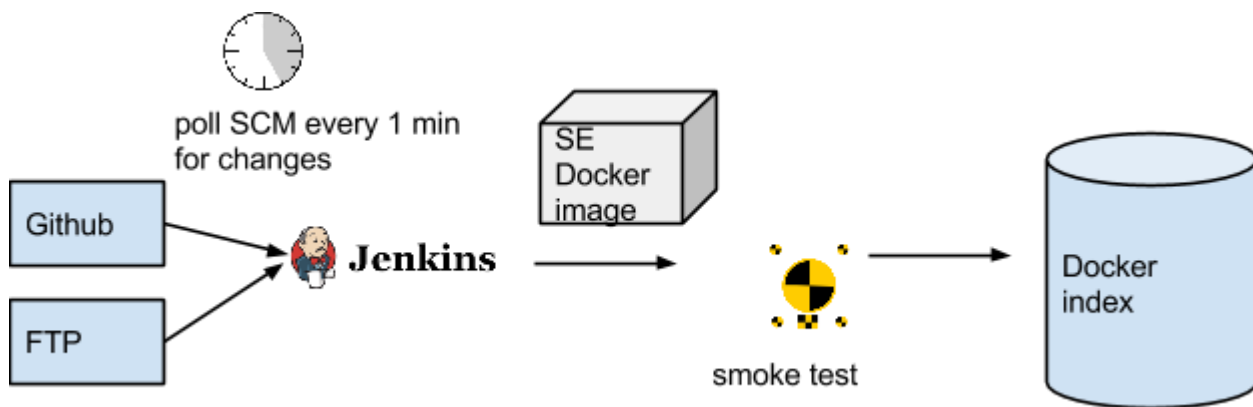
<sup>4</sup> Source: [https://en.wikipedia.org/wiki/Continuous\\_integration](https://en.wikipedia.org/wiki/Continuous_integration) (as of June 22th, 2015)

<sup>5</sup> <https://en.wikipedia.org/wiki/Cron>

<sup>6</sup> <https://eclipse.org/hudson/>

<sup>7</sup> <https://jenkins-ci.org/>

<sup>8</sup> <https://docs.docker.com/reference/builder/>



**Figure 5-4 – Architecture specification**

The described two generic approaches based on Jenkins and Docker software allow for a simple, effective, rapid, low-cost and scalable deployment process, which works also for Generic Enablers or any other type of server-side software runnable in a Linux environment.

To run the functional acceptance test, FIC2Lab has developed a standard procedure covered by the next chapter. This procedure applies to all server-side Specific Enablers, which use only one network port (currently, all of them).

As this setup allows for a rapid creation of test environments as VM images in large amounts, a referenced term for this architecture is “image bakery”<sup>9</sup>.

## 5.6 - Docker script runner

This component is an advanced shell script aiming to manage container-based SE “functional crash testing” in a systematic and system-friendly manner, while being configurable and suitable for easy debugging. In this approach, basic functional testing can be fully automated and iterated as frequently as required.

The test runner script automates acceptance checks for server-side SEs as following:

- Accept standard interface call from a Jenkins job representing one sample of test process iteration per SE.
- Ensure that input is correct and environment contains all required dependencies.
- Provide verbose output where essential to easy troubleshooting; Jenkins has an auto-purge mode for obsolete logs so disk space is not a restriction if using proper configuration.
- Reserve a free system port mapping for the new test asset. This is a very important concept in this setup to avoid random port conflicts between any two SEs using say e.g. 8081 for their REST API.
- Instantiate a Docker container (“setup test fixture”).
- SE vendors provide runnable functional test specifications by themselves following the testing interface agreement. This means that FIC2Lab operators only need to review the functional test if they make sense in general. This simple acceptance check has proven to be already very useful to detect a series of software errors.
- On error, the system generates bug reports and sends them per email to the persons in charge.
- Host system destroys container with the tested virtual SE environment to free operational resources (“tear down test fixture”).

<sup>9</sup> In a 2012 blog entry, the author references a similar setup to generate Amazon cloud VM images <http://richsteer.com/netflix-aws-architecture-talk-from-adrian-cockcroft/>

Finally, and only in case of success, the system pushes non-proprietary SE Docker VM images to the official Docker Hub repository: <https://registry.hub.docker.com/repos/fic2/>. Images of proprietary SEs can go also to a privately hosted local Docker repository.

### 5.7 - Live component status report via Jenkins CI

As Jenkins provides a live dashboard, which lists existing jobs, this output is a convenient tool to get a high-level impression of what is going in terms of server-side components' stability without going into much detail or having to test components manually. Figure 5-5 shows two SE components (at the time of the screenshot capture) being marked as having an unstable state as of latest integration process run (red colored balls); this situation needs an assessment together with the SE developers. Weather symbols depict builds stability across the last five acceptance check iterations<sup>10</sup> (sunny weather symbol appears if all of them have been successful).

**Please note that any versions of components failing acceptance checks do not go to FIC2Lab.**

Figure 5-6 shows another dashboard, this one being used to track the status of GE integration. Although FIC2Lab aims at SEs coming from FICONTENT2, there are SEs, which depend on a GE. Additionally, partners from the accelerator project EUROPEAN PIONEERS<sup>11</sup> provided a proof of concept Docker integration for more GEs.

---

<sup>10</sup> Historically, one iteration of a job is called *build* in CI context because CI was primarily used to merge source code delivered in a collaborative development process. In this context, Jenkins works on a higher abstraction level because built items are whole Linux virtual operation system environments.

<sup>11</sup> <http://europeanpioneers.eu/en/>

Dockerizable SEs in FIC2Lab

[All](#)
[Batches](#)
[FIC2Lab descoped](#)
[FIC2\\_SE](#)
[GE](#)
[custom\\_endpoints](#)
[-FIC2Lab- dockerizable](#)

S	W	Name	Last Success
		<a href="#">se_3d_map_tiles</a>	5 hr 21 min - #153 <a href="#">fraunhoferiais/se_3d-map-tiles.latest</a> <a href="#">fraunhoferiais/se_3d-map-tiles.latest</a>
		<a href="#">se_audiomining-de</a>	12 hr - #322
		<a href="#">se_audiomining-en</a>	18 days - #5
		<a href="#">se_car_acr</a>	12 hr - #279 <a href="#">fraunhoferiais/se_car_acr.latest</a> <a href="#">fraunhoferiais/se_car_acr.latest</a>
		<a href="#">se_car_rmp</a>	5 days 16 hr - #129 <a href="#">fraunhoferiais/se_car_rmp.latest</a> <a href="#">fraunhoferiais/se_car_rmp.latest</a>
		<a href="#">se_content_optimisation</a>	12 hr - #263 <a href="#">fic2/cose.latest</a> <a href="#">fic2/cose.latest</a>
		<a href="#">se_fatts</a>	12 hr - #128 <a href="#">fraunhoferiais/se_fatts.latest</a> <a href="#">fraunhoferiais/se_fatts.latest</a>
		<a href="#">se_fusion_engine</a>	12 hr - #238 <a href="#">fraunhoferiais/se_fusion_engine.latest</a> <a href="#">fraunhoferiais/se_fusion_engine.latest</a>
		<a href="#">se_leaderboard</a>	12 hr - #207 <a href="#">fraunhoferiais/leaderboard.latest</a> <a href="#">fraunhoferiais/leaderboard.latest</a>
		<a href="#">se_leaderboard-mysql</a>	10 hr - #70 <a href="#">fraunhoferiais/se_leaderboard_mysql.latest</a> <a href="#">fraunhoferiais/se_leaderboard_mysql.latest</a>
		<a href="#">se_open_city_database</a>	11 hr - #249 <a href="#">fraunhoferiais/se_open_city_database.latest</a> <a href="#">fraunhoferiais/se_open_city_database.latest</a>
		<a href="#">se_poi_proxy</a>	11 hr - #251 <a href="#">fraunhoferiais/se_poi_proxy.latest</a> <a href="#">fraunhoferiais/se_poi_proxy.latest</a>
		<a href="#">se_social_network</a>	5 days 11 hr - #299 <a href="#">fraunhoferiais/se_social_network.latest</a> <a href="#">fraunhoferiais/se_social_network.latest</a>
		<a href="#">se_tv_application_layer</a>	11 hr - #196 <a href="#">fraunhoferiais/se_tv_application_layer.latest</a> <a href="#">fraunhoferiais/se_tv_application_layer.latest</a>

Icon: [S](#) [M](#) [L](#)

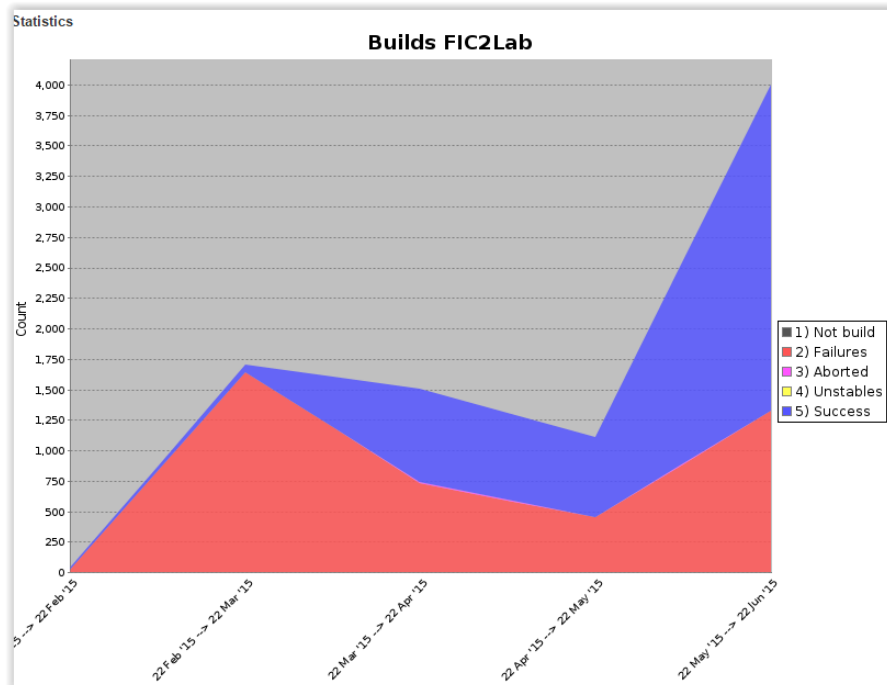
Figure 5-5 – Live status of SE stability via Jenkins dashboard

[All](#)
[Batches](#)
[FIC2Lab descoped](#)
[FIC2\\_SE](#)
[GE](#)
[custom\\_endpoints](#)
[-FIC2Lab- dockerizable](#)

S	W	Name	Last Success
		<a href="#">ge_poi_dataprovider</a>	10 hr - #19 <a href="#">fraunhoferiais/ge_poi_dataprovider.latest</a> <a href="#">fraunhoferiais/ge_poi_dataprovider.latest</a>
		<a href="#">ge_cosmos_master</a>	N/A
		<a href="#">ge_cep</a>	1 mo 18 days - #13 <a href="#">fraunhoferiais/ge_cep.latest</a> <a href="#">fraunhoferiais/ge_cep.latest</a>

Icon: [S](#) [M](#) [L](#)

Figure 5-6 – Jenkins dashboard for integration of Generic Enablers



**Figure 5-7 – Jenkins build statistics since January 2015**

As shown in Figure 5-7, since January 2015 the integration machine's operational performance scaled towards about **4000 full automated acceptance check cycles a month**, delivering as of June 2015 quality assurance for about 20 virtual environment specifications of all available server side SEs and some GEs deployments.

One such cycle is following the concept as explained in Figures Figure 5-2, Figure 5-3 and Figure 5-4:

- load a new version of a delivered SE artifact;
- perform automated system integration test and functional test **as formally specified by the SE vendor**;
- push container to Docker Hub, or send error report to FIC2Lab operations team and SE vendor.

Thus, probability of SE installation problems in the productive operational system decreases for FIC2Lab or any other Linux compatible OS environment (as in the FIC2Lab use case RUN), given that only quality proven images are used for deployment and still possible errors are communicated to FIC2Lab for further quality check improvements.

An additional benefit to be mentioned is that users, who prefer not to use Docker for operation e.g. for security or any other reasons, still can use the created Dockerfiles as a formal reference specification of the last known working environment. This approach can significantly reduce efforts connected to maintenance of written installation manuals, which in IT environments are well known to outdate very soon due to the rapid evolution of software implementations and growing dependencies complexity.

## 5.8 - Proposals for further improvement for FIC2Lab acceptance checks

As software changes and improves over time and increases in complexity, a system integration QA procedure should also follow the ITIL Continual Service Improvement process to catch up with these changes and keep quality at an acceptable level.

For example, a series of load tests to measure scalability of components under very high load might make sense, or automated security penetration tests to discover possible vulnerabilities (which however partly depend on vulnerabilities of OSS components used beneath FIWARE Enablers).

The presented system integration solution is not limited to Specific Enablers; it can be used also for GEs or any other software beyond FIWARE context. Linux environment is preferred, but the solution concept is transferrable also to probably virtually any other platforms and tools, with more or less effort in each case.



## 6 - CONCLUSION

FIC2Lab has been designed as a comprehensive and easy to use platform for understanding and experimenting results of the FIWARE use case project FI-CONTENT. The platform hosts currently 21 operational software modules, called FIWARE Media&Content enablers, each of them realizing one or several functions in relation to the three main technical and business areas covered by the project (Social Connected TV, Smart City Services and Pervasive Gaming). The main entry point to FIC2Lab is the one stop [lab.mediafi.org](http://lab.mediafi.org) portal that allows the user/developer to access the 4 main functions provided – Discover, Try, Tweak and Run.

The platform targets IT professionals who plan or are considering to develop applications (web applications, or mobile device apps). First priority potential users of the platform are the FI-PPP Phase 3 accelerators and sub grantees. The platform is specifically geared towards giving information and allowing saving time to these developers. By no means is the platform meant to be used as an operational environment for mission critical application. FIC2Lab targets application development, not operational deployment.

FIC2Lab supports a rather large variety of technologies and delivery models:

- Web Services: Part of the FIWARE Media&Content enablers are implemented by means of Web Services which can be delivered via a SaaS Model, or dedicated instances started on demand by the developers – some enablers support both delivery models.
- Client side libraries: In order to reach the high level of responsiveness required, the Gaming enablers are mostly implemented as client side modules. These modules are mostly delivered as add-ons on the Unity 3D gaming platform via open source repositories.
- End-User application sites: Some FIWARE Media&Content enablers (App Generator, ARTool, Second Screen Framework, OpenDataSoft...) are plain Web application environments allowing the developers to design and configure applications. These applications are hosted on dedicated servers or – commercial - cloud services.

Ease of access to - presumably correct and useful - information and ease use for the target population (i.e. developers not end-users of Internet services) are the main design principles for FIC2Lab. The efforts of the group of partners involved in readying the platform have mainly concentrated on the following aspects:

- 1) Realizing the FIWARE Media&Content portal.  
A rather basic PHP portal using a SQL database.
- 2) Setting up of a documentation workflow allowing to structure the information and to dispatch it in a comprehensive way to the target population.  
Based on DokuWiki scripts.
- 3) Setting up the runner environment allowing shielding the developers from presumably nonproductive Openstack/FIWARE Lab related hassle on experimenting and doing test deployments of software modules.  
The runner environment is layered on Docker.
- 4) Setting up of an automated system integration workflow – based on Jenkins-, allowing to port the enablers to the target operation environment and to perform preliminary functional testing.  
The porting effort is referred to as “dockerization”.
- 5) Realizing the developer Playground allowing developers to “tweak” the look&feel and behavior of FIWARE Media&Content enablers.  
The playground allows developers to interactively change the client side Java code of enablers, assess results and save the code if appropriate.
- 6) Qualifying the different enablers and make sure they comply to a minimum set of compliance standards.  
The qualification process is partly automatic and partly manual.

Along this roadmap, quite some effort has been invested to work around shortcomings of the FIWARE Lab environment, and lack of stability. In particular the Playground and the Runner have required several stages of rather heavy technical realization work.

Finally, several actions were conducted for promoting the platform and the FIWARE Media&Content enablers to the Phase 3 accelerators and SMEs potentially concerned by using the enablers. Among these actions the partners organized a meeting with the accelerators begin March in Barcelona, and several presentations to sub grantees on accelerator premises (Hamburg, Nantes, Cologne). FIC2Lab was presented at ECFI and at the Fokus Web symposium. And finally several partners toured incubators and technology hubs.

We believe that FIC2Lab is a big step towards a sustainable environment of Media&Content enablers allowing the emergence of a Europe wide eco-system of Media&Content application and service developers. Yet the work is not finished and a series of enhancement, including SEO, GUI enhancement, more functionality, more participation (via forums, rating, etc.), more reliability, more openness is being considered. The partners involved are excited to continue the journey towards building a genuine community of developers and serve them with more information, more functionality, more reliability, higher visibility and crowd sourced self-support.