SYNCHRONICITY

GA no:	732240
Action full title:	SynchroniCity: Delivering an IoT enabled Digital Single Market for Europe and Beyond
Call/Topic:	Large Scale Pilots
Type of action:	Innovation Action (IA)
Starting date of action:	01.01.2017
Project duration:	36 months
Project end date:	31.12.2019
Deliverable number:	D3.4
Deliverable title:	Common methodology and toolset for city service customization
Document version:	1.0
WP number:	WP3
Lead beneficiary:	35-NEC
Main author(s):	Flavio Cirillo, Detlef Straeten (NEC), David Gomez Fernandez, Jose Gato (ATOS), Giuseppe Ciulla (ENG), Daniel Puschmann (DigiCat), Ignacio Elicegui Maestro (UC)
Internal reviewers:	Reza Akhavan (FCC), Luis Muñoz (UC), Martin Brynskov (AU)
Type of deliverable:	Other
Dissemination level:	PU
Delivery date from Annex 1:	M16
Actual delivery date:	M20 (30.08.2018)

Co-funded by the European Union

This deliverable is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732240.

Executive Summary

As stated in the Framework Agreement, Task 3.2 of the SynchroniCity efforts provide reference implementations Baseline Services which will support the City Services designed in effort Task 3.1 in the form of generic software enablers that are compatible and easily integrated with the reference architecture for the targeted applications and services. Baseline Services themselves do not represent actual end user facing City Services, instead they build a first number of reusable service modules to compose City Services faster and more consistently.

This effort will be executed progressively to make optimal use of the increasing functionality of the SynchroniCity framework developed in WP2 over time, deploying consecutive implementations for successive framework feature sets. For details of the actual Baseline Services (Basic and Advanced) portfolio sets see deliverables [1] and [2].

Those Baseline Service implementations will be one basis for the foreseen City Services leveraging most relevant data streams and functionalities to be exposed. The intended comprehensive nature of the baseline implementations, in turn, facilitates reference zone customization and stimulates replication based on common solutions and designs.

Together with the SynchroniCity architecture framework, these common solutions prevent both “vendor lock-in” from the cities’ perspective and “city lock-in” from the perspective of IoT device, service and middleware providers.

While deliverable D3.2 described the first portfolio release of mentioned Baseline Services, the deliverable D3.4 as part of Task 3.2 eventually describes procedures and tooling to facilitate customization and replication of the Baseline Services into Reference Zones (RZs).

The target audiences for this deliverable are exploiters of Baseline Services, e.g. RZs application developers, third parties joining from the Open Calls, and external users of SynchroniCity.

The Baseline Services owner will need to stick to these guidelines presented here in order to expose the Baseline Services according to one common standard.

Consequently, will we first introduce the aspect of Baseline Services and position content presented here to the other deliverables of Task 3.2 and beyond.

This introduction will be followed by an overall description of the practices and processes suggested to deploy and customize Baseline Services including the actual tooling implementing those practices.

As usual we expect the practices we describe here will evolve based on experiences gathered during the large scale adoption of the Synchronicity technical framework and related Baseline Services portfolio over time. Therefore this document represents a statement in time of reasonable practices agreed with the Reference Zones, Application-level Service Developers and Baseline Services providers.

The same holds true for tooling we identified here to handle a first release of Baseline Services complexities.

Abbreviations

API	Application Programming Interface
D	Deliverable
EC	European Commission
IoT	Internet of Things
NGSI	Next Generation Service Interface
REST	REpresentational State Transfer
RZ	Reference Zone
WP	Work Package
WT	Work Task

Contents

List of Figures

1 Introduction

1.1 Definition of "Baseline Service"

Services in scope of this document are those published in [1] as 'Baseline Services'. A baseline service can be seen as a very abstract concept, in the scope of Synchronicity, we understand it as an atomic block using data input to return any kind of feature, either managing, enriching, joining or filtering the input data from SynchroniCity infrastructure, and/or data coming from other sources. Other specific characteristics of a SynchroniCity's baseline-service are outlined below:

- They are built on top of the SynchroniCity framework and exploit the different interaction points offered by the platform, e.g. NGSI interfaces and APIs, historical data collection, Data Marketplace API [3].
- Data input will be NGSI-compliant (in terms of data models) so as to respect OASC's principles [3]. Consequently, interoperability and replicability will be guaranteed.
- The outcome generated by these Baseline Services must carry out a concrete (and focused) functionality over the input data
- An API to interact with the service and obtain this extra functionality.
- Enabling the development of other applications and services, including those ones addressed in Task 3.3 [4], and opening the door to new integration possibilities for the Open Call.
- It is envisioned that these services could be straightforwardly downloadable and deployable in different environments.

'Baseline Services' are agnostic to the delivery model they depend on.

The envisioned Delivery models for Baseline Services are:

- Local dedicated code deployments by city/RZ (creating an individual instance locally)
- Remote dedicated code deployments in a SaaS (Software as a Service) model by city (creating an individual instance remotely)
- Remote Shared code deployment in a SaaS model across multiple Cities

To simplify our approach and accelerate time–to-market, we treat each Baseline Service like a FIWARE Generic Enabler[1], which offer a number of general-purpose functions, offered through well-defined APIs, easing development of smart applications in multiple sectors.

1.2 Common Methodologies and Tooling

Considering the emergence of a significant number of Baseline Services to be exploited by Synchronicity users from various third parties, a key enabler for adoption is the use of common methodologies and toolsets to deploy and customize these services.

This document introduces the suggested methodologies and defines the selected toolset for realization. In line with the overall approach in the Synchronicity efforts the current best practise were identified together with the RZs, analized and as a consequence described as recommended practises for a Baseline Service deployment.

The following disciplines in deploying a Baseline Service will be discussed:

- 'Learn' about a Baseline Services

[1] https://catalogue-server.fiware.org/

- ‘Contracting’ or ‘Licensing’ of Baseline Services
- ‘Acquiring’ the related technical artefacts of Baseline Services
- ‘Installation of Baseline Services
- ‘Configuration’ or customization of Baseline Services
- ‘Basic Validation Test’ of Baseline Services install
- ‘Integration’ approach with other services
- ‘Support’ of respective Baseline Services

1.3 Quality Assurance & Compliancy

A technical ‘Baseline Service’ is considered complete, if each of those services comes with the following documentation.

Minimum requirements, in order to consider a service a “SynchroniCity Baseline Service” are:

- Baseline Service description
- Docker image & related technical artefacts
- Administration and Installation Guide
- User and Developers/Programmers Guide
- License Model description

The initial set of Baseline Services were identified and prioritized together with RZs from use cases published in [5].

For understanding the maturity level of a specific Baseline Service implementation registered in the Docker repository please see the actual documentation of the baseline service [1] or the exploitation results gained by SynchroniCity City Services implementations [4] [6].

In addition, we leave it open to the baseline service owners to go beyond satisfying these requirements for example by sharing the actual source code.

2 Processes & Tooling Approach

In this section, we will describe the processes and related tooling how a baseline service user can leverage the provided implementation and exploit it to compose higher level services.

Main objectives of described processes are (in no specific order):

- learning about available Baseline Services and their scope
- learning about the license, terms and conditions and contract for using the Baseline Services
- acquiring the software and the related documentation
- installing the component(s)
- configuring the component(s)
- validating that the component is correctly installed and all the functions are working
- learning how the user can integrate the component with other components: e.g. API documentation, integration diagram
- validating that the Baseline Services is correctly interacting with other components
- getting support from Baseline Service owner or other experts.

Whenever possible this document suggests to reuse procedures and practices found successful in similar areas. See Bibliography for more details.

2.1 Learn about Baseline Service Specification

In this section we present how the Baseline Services that will be part of the SynchroniCity project have to be described in a 'catalogue' to ensure the successful adoption of the services by application developers within the reference zones and third-party users (e.g. coming from open calls).

For this purpose, we provided a template for the service description which covers the necessary information from a users perspective, as described below:

- Overall Description & Introduction
- Synchronicity Application Theme functional requirements (identified in [5]) addressed
- Service Interface
- API
- Data Model
- Interaction diagram (with the external world)
- Service Architecture
- Service components
- Operation model (Message flows between components)
- Non-functional requirements
- Service implementation
- Service implementation IT environment (dependencies like SQL Database)
- Service deployment dependencies (e.g. Orion Broker, oAuth server etc.)
- Installation and admin guide
- Implementation Roadmap

Above template addresses the following questions:

Purpose: The Baseline Service specification has to start with a short description of the Baseline

Service. This includes the scope of the service, the intended user group and the overall purpose and goals as well as the interfaces to be exploited. This should give the potential user enough information on the benefit of the service, in order to understand if the service is solving the needs of the users.

Inputs: Technical description of the inputs that are required to be provided by the user in order to make use of the service. The data format of the inputs is described as well as the optional and mandatory fields.

Outputs: Technical description of the outputs that the service produces. The data format of the outputs is described so that the user can make use of the outputs for further processing.

Information model: In order to be compliant with the SynchroniCity project, the Baseline Services are advised to utilize NGSI v2 information models as much as possible. A detailed description of the specific information models that are being utilized within the context of this service has to be provided here.

Functional requirements: The functional requirements that the service is addressing are linked here, the functional requirements have been defined in [7] in the context of each application theme.

Non-functional requirements: Any non-functional requirement that the baseline service addresses are specified here.

Pre-requirements: This describes the necessary pre-requirements that are needed by the application developers and third-party users in order to make use of the service. This includes third party components such as external databases and other technical infrastructures that have to be present to make use of the service.

Please refer also to [1] for the list of initial Baseline Services and latest service description template

Baseline Services will be indexed and made available through the Synchronicity IoT Docker Hub.[2]

2.2 Contracting and Terms and Conditions

The usage of the Baseline Services will be subject to the Terms and Conditions that have to be specified by the Baseline Service asset owner. Baseline Service providers are able to provide the Terms and Conditions through the Baseline Service Description published as part of the Docker Hub boarding effort[2].

2.3 Acquiring Baseline Service Package

All SynchroniCity Baseline Services will be available via Docker Hub[2]. Therefore, this guide will only focus on pulling the image, the container configuration and how to run it. After this, we can also expand to another multi-container solution.

Synchronicity's Baseline Services will lead to the generation and further distribution of Docker[3] images. A sample of this is presented in

Figure 1, where all the off-the-shelf Baseline Services can be found.

It goes without saying that this repository will be appended with more application throughout the project's lifetime.

[2] https://hub.docker.com/u/synchronicityiot/
[3] https://www.docker.com

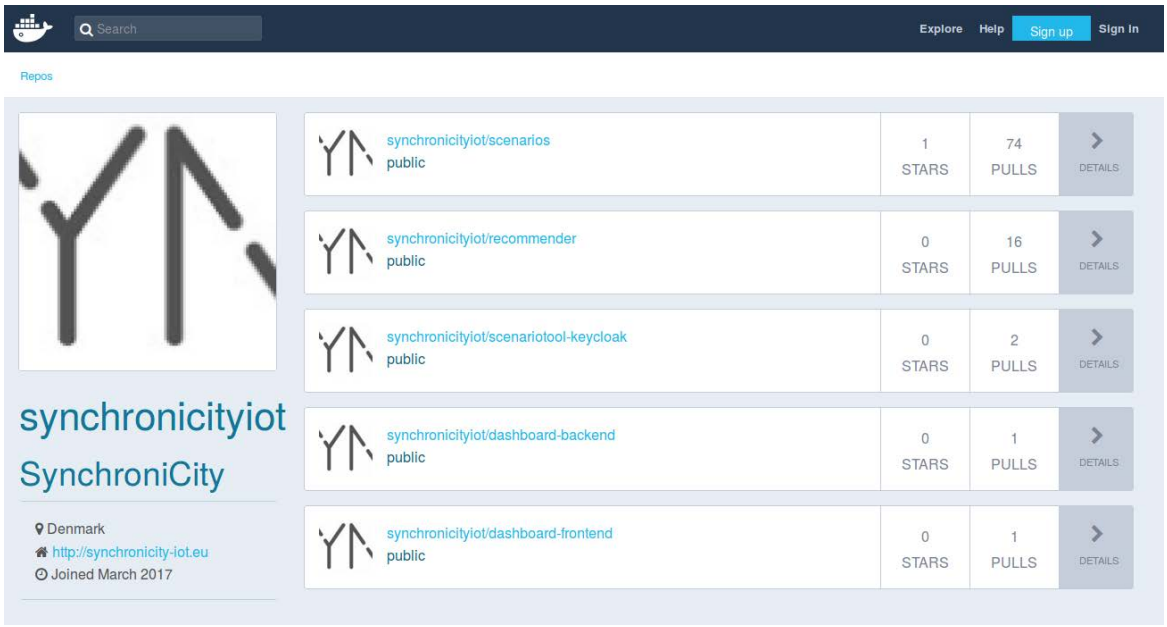

Figure 1. Synchronicity's Docker Hub[2]

By clicking on any of the different images of the registry, we get redirected to its subsequent documentation page, where we can see a full description of the Docker image, the command that has to be run in order to get it, the source code repository (in case it is an open source application), etc.

In essence, the Docker concept leads to the straightforward generation and deployment of Linux containers of (almost) any kind of application and micro services.

Just to settle down a couple of concepts before getting to a misunderstanding, let us introduce the following definitions, excerpted from the official Docker site:

- **Image:** Read-only package that contains all the artefacts necessary to run an application or service. This encompasses the code, a runtime, required libraries, environment variables and configuration files.
- **Container:** Technically speaking, it is the running (or stopped) instance of an image. It is also worth highlighting that a container runs natively on Linux and shares the kernel with other containers.

2.4 Apply Installation Procedure

As it can be easily inferred, the first requirement to run a Docker container is to have Docker installed in the system. Fortunately, there are tailored versions for almost every system, spanning from the classical Windows/macOS/Linux distributions for users' personal computers to cloud solutions (e.g. Amazon Web Services, Microsoft Azure, etc.) and more operating systems (CentOS, Microsoft Windows Server, etc.). As the installation of this package is out of the scope of this document, we recommend that the reader visit the official installation tutorial[4] and follow the steps detailed therein.

Henceforth, we stick to a practical example to help us define and showcase the different steps to be done in order to get a service running in an arbitrary host. In this section we outline how to proceed

[4] https://docs.docker.com/install/

to install (or pull) an image and might be already installed in the host, or not.

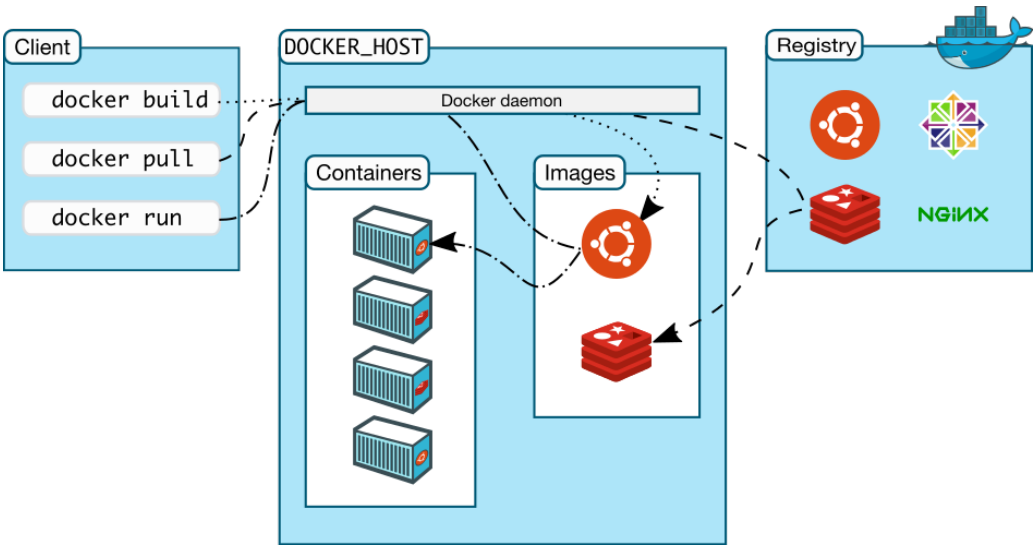

Figure 2. Docker architecture[5]

Figure 2 illustrates how the Docker client-server architecture deals with the creation/downloading of an image and how it is instanced into a container.

Without entering too much into the details, we outline here the process. Basically, this consists of different parts:

(1) a client that works on a Command Line Interface (CLI) basis (which, at the same time, relies on a REST API for the remote calls);
(2) the Docker host, that is, the core of the system in the host machine. In a nutshell, the main component is the Docker daemon, which listens to the Client (Docker API) and manages the different Docker objects, i.e., images, containers, etc.
(3) Remotely, there are the Docker Registries (publics or privates) that behave as images repositories, which are invoked when a Docker Daemon searches for an image and it is not already present in the host.

As for the main Docker API commands needed for getting an image and running a container, we have the following three (left side of the figure):

- **`docker build`**: This former command addresses the creation of an image from a so-called Dockerfile, which will define the image as a combination of off-the-shelf images (this is optional) together with the source code present in the same folder (and subfolders). From this deliverable standpoint, this step is something that should have been already addressed by the services' developers or maintainers.
- **`docker pull`**: This command fetches the image from the Docker registry, as long as it has not been previously pulled (or locally built). Here, it is worth highlighting that a version check is carried out, as new updates might have occurred. On this, besides the name of the image, the Docker API also allows to specify a version, which is done through a tagging mechanism (by default, the system sets the "latest" tag). As an illustrative example, Docker Hub displays the available tags (if any) of the stored images[6].
- **`docker run`**: Finally, this last command runs a container based on an image. Nonetheless, in most of the cases (mainly in single container applications, some configuration would be

[5] https://docs.docker.com/engine/docker-overview/#docker-architecture
[6] https://hub.docker.com/r/fiware/orion/tags/

deemed necessary.

Section 2.5 outlines the main highlights of the different parameters to be defined at runtime (container names, labels, volumes, exposed ports…).

2.5 Configure

At this point we assume that we have successfully installed Docker in our system and we have also pulled the corresponding Docker images from Synchronicity's Docker Hub. Now, it is time to configure the environment in which we want the container to operate.

Single container applications

In the simplest cases, applications or in our case Baseline Services are made of a single container. On its own, the concept of application / baseline service may be extremely abstract, as it may be a micro service, an API endpoint, a web app, and a long et cetera.

To run a container/instance an image, the basic CLI command looks like this (even though a full description can be found in the official documentation[7], in this deliverable we outline the most highlighting aspects to be taken into account):

```
$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG…]
```

It is worth remarking that, when this command is executed, the container process builds its own filesystem, networking, environment variables, etc., all of them independent from the host. It is at the command configuration where we have to set the necessary bindings between host and container.

Getting back to the *run* command, we must specify the base image that will be used to instance the container. As hinted before, beside the image name we can also pinpoint to a particular version using a tag. Moreover, as of Docker v2, users can select the image by means of a unique content-addressable identifier, also called digest.

2.6 Installation validation test

Service's end user depends heavily on service documentation to make the most of its integration within their final applications.

Due to this, Synchronicity Baseline Services must provide clear, concise and easy-to-follow documentation that:

1. details the integration with other required services and/or components
2. defines a set of minimum set of test to verify and validate the proper installation and functioning of the service
3. helps the end user to identify and fix the errors found during the service installation or its usage

[7] https://docs.docker.com/engine/reference/run/

The Sanity Check Procedures are the steps that a System Administrator will take to verify that an installation is ready to be tested. This is therefore a preliminary set of tests to ensure that obvious or basic malfunctioning is fixed before proceeding to unit tests, integration tests and user validation.

We expect the following procedures to be described in each Baseline Service description:

- End-to-End Testing mechanism
- List of running processes
- Network Interfaces Up/Open list
- Databases access testing

2.7 Integration

Apart of its proper installation and setup, in order to validate the Baseline Service instance, a verification of its integration with defined services or components (or its capability of being integrated) is needed, since this integration should be required to provide proper service outcomes. Finally, its functional validation will check the functional requirements specifically covered by the baseline service (detailed in [1]) that guarantee its proper execution.

We expect the following integration procedures to be described in each Baseline Service description:

- Integration with other Synchronicity architecture components (internal dependencies)
- Integration with other required components (external dependencies)
- Service functional requirements validation

2.8 Support

This section reports and describes a generic support procedure for Baseline Services; the specific support plan for each baseline service should be defined, operated and maintained by the asset owner. Information reported in this section represents a general guideline for the implementation and operation of the support procedures and for the communication with the end users those require support.

Once the specific support procedure will be defined for a specific Baseline Service, it will be very important that they will be implemented and operated according to their definition. This is essential to avoid confusion, misunderstandings and waste of time both internally in the management of support requests as well as externally during interaction between the user requesting support and the support provider.

Indeed communication channel to interact with the end users should be well defined and published (e.g.: emails address, forms on web pages, forums, wiki and F.A.Q. pages, etc.); once a support request has been submitted, it should be well identified by a unique code representing a ticket (sometimes called ‘issue’) associated to the support request itself. The ticket/issue can keep track of all the information related to the request, its status, progress and resolution.

Figure 3 depicts a suggested process to provide end users with support; for each incoming support request, in consequence of the first of the process "User send a support request", the first action to

be performed is to assign a ticket/issue at the support request and then to check if the specific support request is about a problem previously faced and solved; if a solution for the problem exists and it is reported in a wiki or in a F.A.Q page then the solution is notified to the end user. After that, it will be necessary to check if the proposed solution solves the problem. If yes the ticket/issue is closed.

If not or if a solution for the problem does not exist, it will be necessary to check if more details about the problem are needed to identify a proper solution. Then the technical support will be provided and a solution to the problem identified. This phase will be conducted in contact with the user in order to check at the same time the resolution of the problem. When the problem is solved it is documented and the relative solution is saved for future use.

Finally, the ticket/issue is closed.

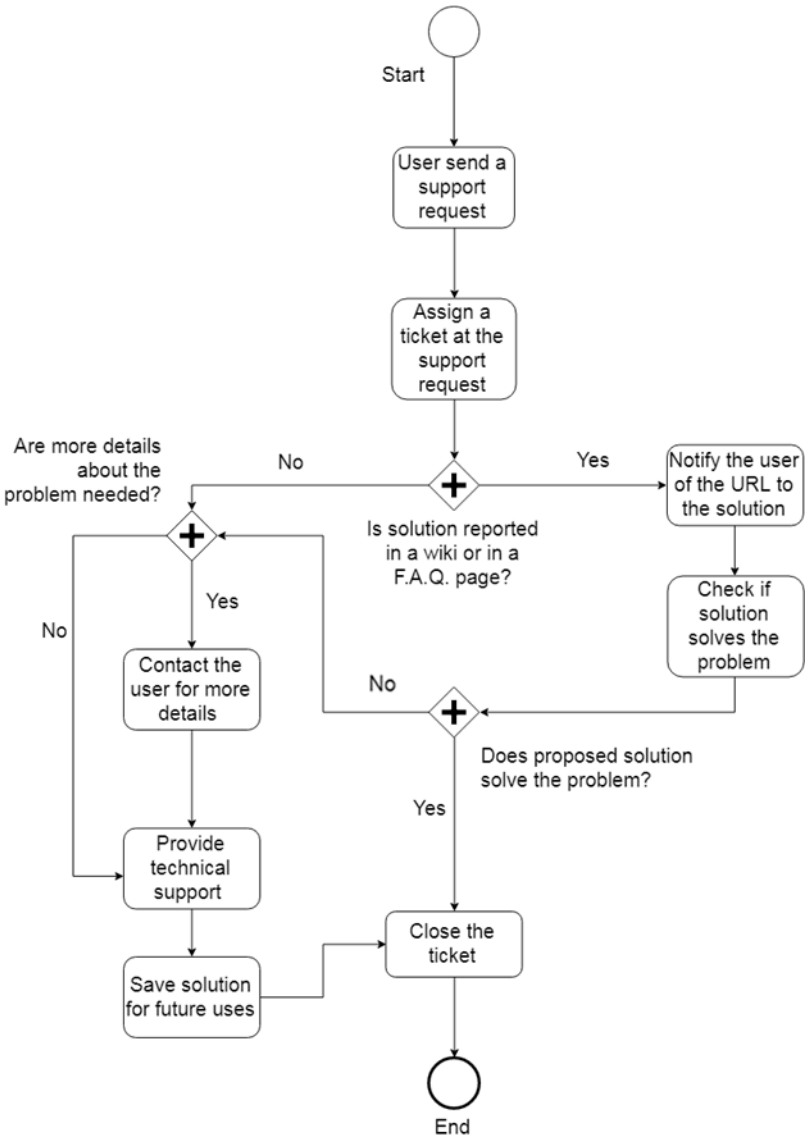

Figure 3: Support plan - sample support process for baseline services

In order to provide service users with technical support, different tools can be used. It is important to underline that it is not possible to indicate which the best one is; indeed it depends on different factors, such as:

- Who is providing the technical support (e.g.: a large company, a small one, an open source community, etc.)
- Complexity of the service for which the technical support is provided
- Number of the expected users of the service
- Number of expected technical support request
- etc.

So, it is important to evaluate carefully all the factors that can influence the technical support.

For complex services used by a large number of users it is recommended to adopt a dedicated ticketing system that helps both the organization providing the service to keep trace of support requests and the users of the service to submit their requests.

Two examples of these systems are *Bugzilla* [8] and *MantisBugTracker* [9]; both are open source and freely available. Indeed, MantisBugTracker is released under the term of *GNU General Public License* [10], whereas Bugzilla is released under the *Mozilla Public License* [11]

For simple Baseline Services, characterised by a low complexity and by a limited number of functionalities, it could be possible to adopt more simple solutions, for instance a web form of an email address through which end users will be able to send support requests.

In any case, some solutions and tools can be considered transversal, and their adoption is recommended, such as a list of the most common problems users may encounter or experience and how to solve them (e.g.: a web page reporting FAQ - Frequently Asked Questions) or public forum to provide a fast and efficient support leveraging also the community of developers that could evolve around a baseline service. An example of this kind of forum is *Stack Overflow* [12], that represents a well know Web site for questions and answers where computer programmers exchanges experiences and competence.

3 Tooling

The tools shown here are the ones currently used as exposers to external users. It might happen that in the future the tools change or upgrade to an enhanced “look and feel”. Nevertheless the concepts and the contents will stay as shown in the following sub-sections.

To make the adoption easier and drive consistency we provide an example template to fill in and extend.

We will start with some specifics on DockerHub followed by GitHub as the tool for baseline service support questions.

3.1 DockerHub

As stated earlier the SynchroniCity DockerHub repository will be used for hosting the service description, and the image of the service. It serves as catalogue for the SynchroniCity baseline services, as shown in Figure 4, where we illustrate the current Docker images already available in the “marketplace”.

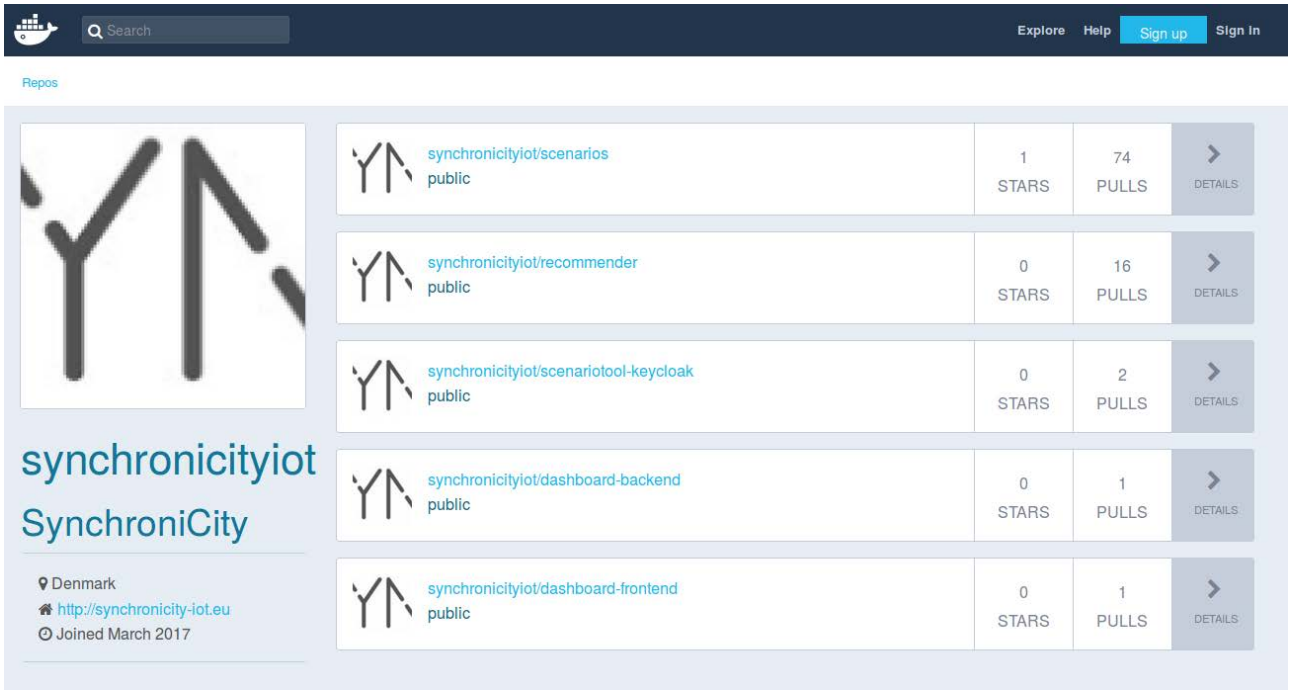

Figure 4: Synchronicity’s Docker Hub

The following shows an overview of the provided ‘dummy baselines service’ template provided as an example[8]:

[8] https://hub.docker.com/r/synchronicityiot/dummy-baseline-service-description/

PUBLIC REPOSITORY

synchronicityiot/dummy-baseline-service-description

Last pushed: never

Repo Info Tags

Short Description

Template for documenting a Docker image

Full Description

Description

This baseline service is offering the means to aggregate and compute data for the purpose of....

Application theme functional requirements

SynchroniCity project has the objective to offer smart services to citizens, namely "city services". Those services are categorized in different "application themes" which have been defined in (SynchroniCity, D3.1, 2018). This baseline service covers the following functional requirements identified in (SynchroniCity, D3.1, 2018).

Interface and API

The API is based on the OMA NGSI. The data format is based on JSON.
The service acts synchronously and each request will be answered with a single response.
For further information please refer to the APIary link

Service Architecture

This baseline services is composed by X atomic blocks: block A that does ...; block B that does ...
Block A interacts with block B...
For more detailed information of the service specification please refer to the link...

Service Deployment

Here will go a quick installation and admin guide for having the service up and running trough docker.
A link to more comprehensive documentation about configurations and features might be provided.

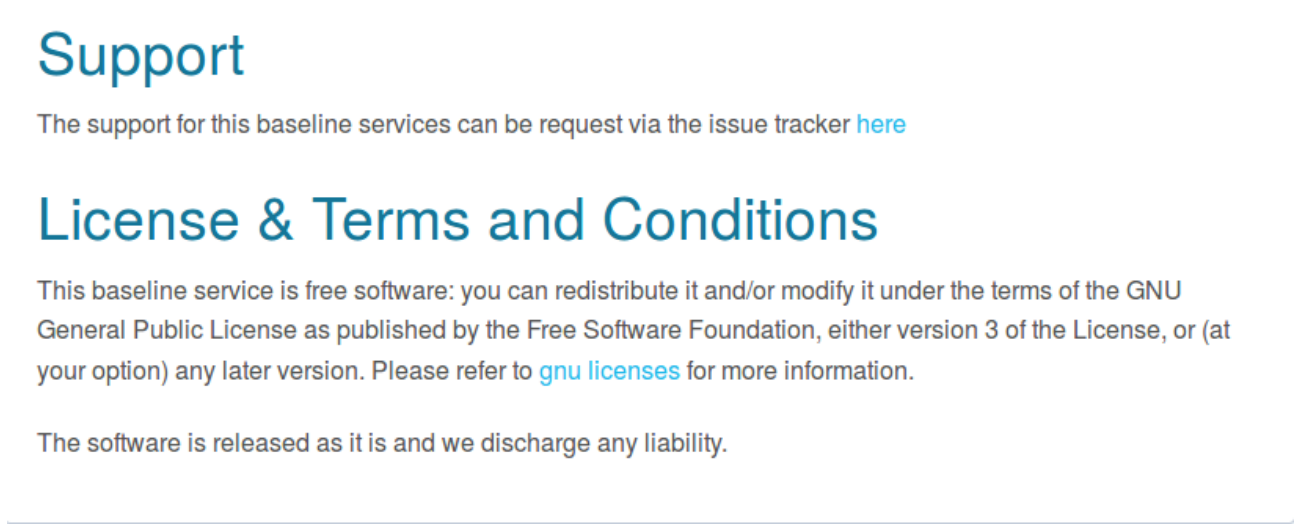

Figure 5: Dummy Baseline Service Template

3.2 Git Repository

Concerning the handful of options to share and expose the source code of the different baseline services (although this is not something mandatory), we rely on GitLab[9] for deploying all the developments (and e.g. wikis) that have been/are being developed in SynchroniCity. Moreover, apart from the evident storage and sharing of the different software projects, GitLab also offers other services that can be harnessed in the scope of the project. In the following sections we focus on two of them: an automated license compatibility finder and checker, and a fully-fledged issue tracker.

3.2.1 License

At the time of writing this deliverable, there is no official agreement on the license(s) that will be used on the different developments carried out under the umbrella of the project. However, this does not mean that partners do not have to pay attention to this. Obviously, the fact of including a software license within every project brings about advantages, not only from a legal standpoint, but also from compatibility, availability, etc.

As for the way GitLab manages all what has to do with licenses, and for the sake of complying the terms that the libraries are exposing, they run an automate process[10] that is executed after any commit is pushed. There, it asserts that all gems and node modules within the project do not generate a conflict with GitLab Community Edition's or GitLab Enterprise Edition's licensing. Here, as can be inferred, this feature does not only check the typical *LICENSE.md* file that is allocated at the project's root path, but also the underlying files that belong to the dependencies.

Giving more context on this, this process manages two different lists (some examples are outlined below; for the complete list, the reader should refer to GitLab's documentation site[11]):

1. *Acceptable licenses*: e.g. MIT, LGPL, Apache 2.0 Licence, Ruby 1.8 License, etc.
2. Unacceptable licenses: e.g. GNU GPL, GNU AGPL 3.0, Open Software

[9] https://gitlab.com/synchronicity-iot/
[10] https://github.com/pivotal-legacy/LicenseFinder
[11] https://docs.gitlab.com/ee/development/licensing.html

However, a library may not be covered by any of the ones that are included in these two lists. In this case, this new License file can be submitted to GitLab's legal team for a review.

3.2.2 Support

In the previous chapter we motivated and discussed the need for a support procedure so that baseline service users can interact effectively with the respective baseline service owners or providers. The way to deal with the issues raised among either consortium partners or externals coming from the Open Calls will be handled via GitLab's Issue Tracker[12]. For a thorough description of this, the reader should refer to [13].

Said in layman's terms, this issue manager allows teams and collaborators to spread and discuss ideas, suggestions, etc. before or while the code is implemented. Obviously, reporting bugs, errors is also a key part of the service, where other users can advise developers that the software is misbehaving.

On top of a raw "title + description" tuple that describes the issue itself, GitLab (as well as any other regular issue tracker) does provide another level of information, where the person who generates the issue can specify the following "overhead":

- *Assignee*: a user can directly pinpoint a developer/maintainer (or more), who will become the responsible (team) for moving the issue forward. As a matter of fact, this process automatically binds mail notifications to the events that may occur with regards to the issue.
- *Milestone*: issues can be grouped under a common milestone; for example, a project, feature or time period.
- *Labels:* another way of classifying the issues is by means of labels, where project administrators might keep a better track on what is actually happening in the repository. As an illustrative example, some typical labels might be: "bug", "help wanted", "duplicate", "invalid", "question", etc.
- *References:* For issues that depend on already existing ones, it is possible to link them. By typing a hash ("#") symbol plus the issue number we can "connect" issues.
- *Mentions*: Yet it is not a standalone feature, this intrinsic (and well-known) operation allows to directly "call" other GitLab users inside of an issue. As a matter of fact, the operation is alike to that of Twitter (i.e. prepend an "@" symbol before someone's username; "*@username*"). Moreover, it also works for Teams within the GitLab organization.
- *Due date:* it is easy to infer what this one is about. In case the publisher of the issue wants to set a deadline in which he/she wants to limit its duration, this is the right place. Internally, underlying events (i.e. reminder mail) will be accordingly triggered.

As for the issue tracking, maintainers can choose between "Issues per project" or "Issues per group".

For general inquiries and defect support we demand baseline service owners to adopt the broadly used Git Repository approach. This is also consistent with the overall Synchronicity Technical Framework Implementation assets support tooling.

For the sake of displaying an illustrative example, we show in Figure 6 a generic view of an arbitrary project managed by GitLab version control system.

[12] https://gitlab.com/synchronicity-iot/dummy-baseline-service/issues

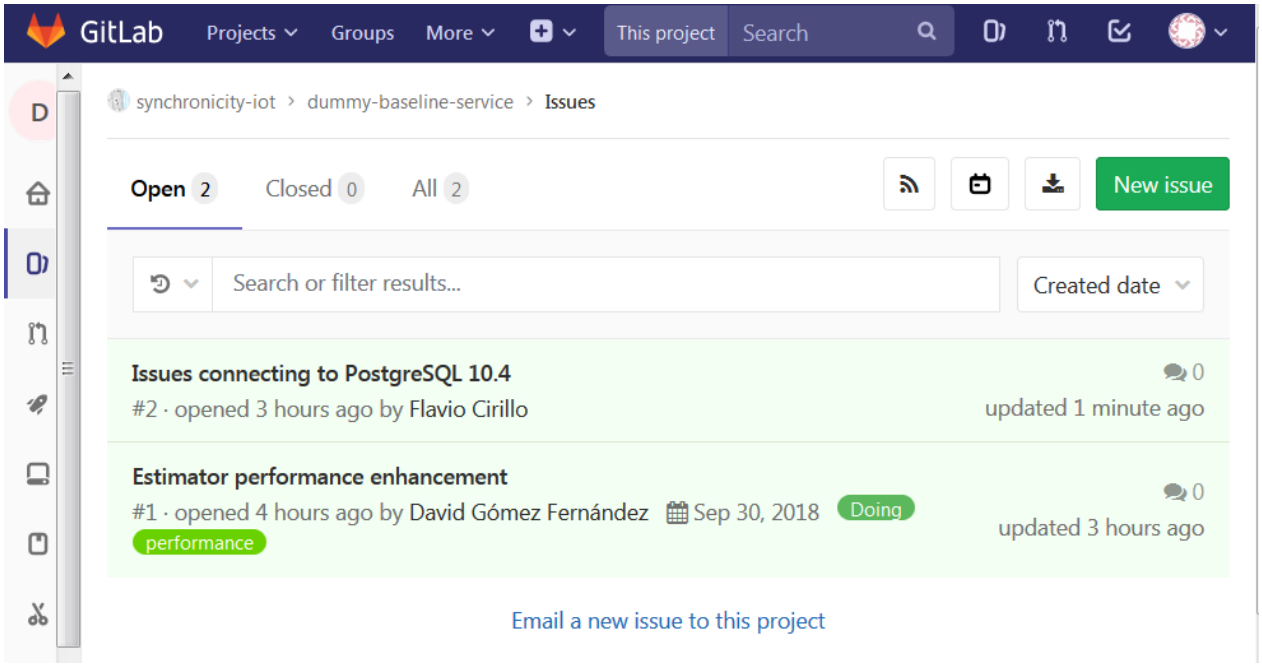

Figure 6: GitLab issue tracker illustrative example (main view)

Last, but not least, it is worth remarking that the mere fact of using this tool includes intrinsically a fully-fledged accounting framework, where consortium administrator are able to monitor the activities that are inherent to the technical implementation. This means that they will be aware of the behaviour of the people responsible for answering to these issues (for example, reaction time, average time needed to close an issue, who responds more/less, and a long etcetera). As a matter of fact, GitLab incorporates a built-in "Board" graphical interface where all this information is displayed in a friendly manner, following a Kanban-like style, as shown in Figure 7.

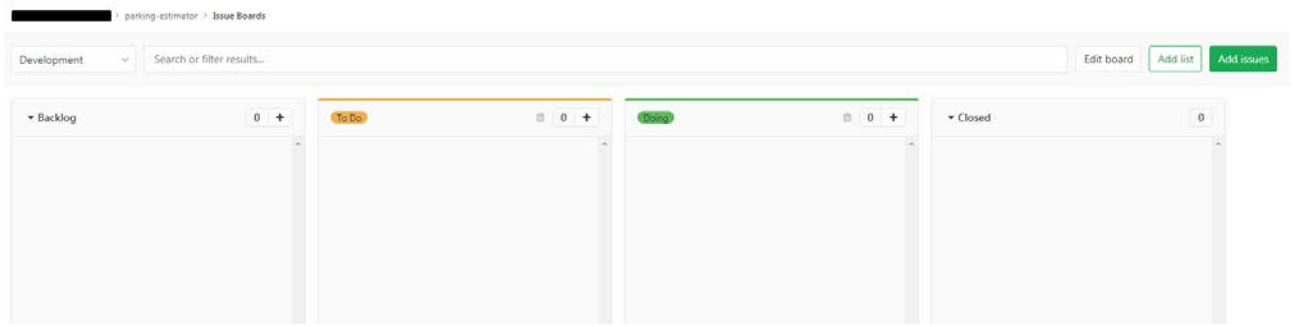

Figure 7: GitLab issue tracker Board sample (Kanban view)

4 Conclusions and Outlook

As motivated at the beginning, this document describes a minimum set of procedures and tooling to accomplish a standardized yet practical way to acquire Baseline Services. Its recommendations

were derived from practices adopted in other areas of the Synchronicity Technical Framework for consistency.

The deployment mechanisms identified were selected from existing best practices in large scale deployment of IT technical services. There is valid believe those have the maturity and robustness also to support initial SynchroniCity Service composition and initial deployment.

With the objective of the document in mind, we have described mainly technical service deployment and assurance procedures. Nevertheless presented approach allows the establishment of any complementary Marketplace model related business provisioning processes including fulfilment and billing.

For further reading on complementary installation and deployment procedures, please refer to “Customized IoT service prototypes for lead Reference Zones” [4] [6]. [13]

5 Bibliography

[1] SynchroniCity, "D3.2 - Suite of baseline implementations - basic," SynchroniCity, 2018.

[2] SynchroniCity, *D3.3,* SynchroniCity, 2018.

[3] SynchroniCity, “D2.1 Reference Architecture for IoT Enabled Smart Cities,” August 2017.

[4] SynchroniCity, *D3.5,* SynchroniCity, 2018.

[5] SynchroniCity, *D3.1,* SynchroniCity, 2018.

[6] SynchroniCity, *D3.6,* SynchroniCity, 2019.

[7] SynchroniCity, “D3.1 - Specification and design of initial IoT applications,” 2018.

[8] “Bugzilla,” [Online]. Available: https://www.bugzilla.org/.

[9] “Mantis Bug Tracker,” [Online]. Available: http://mantisbt.org/.

[10] “GNU General Public License,” [Online]. Available: https://www.gnu.org/licenses/licenses.en.html.

[11] “Mozilla Public License,” [Online]. Available: https://www.mozilla.org/en-US/MPL/.

[12] “Stack Overflow,” [Online]. Available: https://stackoverflow.com/.

[13] “GitLab Issue Tracker Documentation,” [Online]. Available: https://docs.gitlab.com/ee/user/project/issues/.

[14] “General Data Protection Regulation - Official Journal of the European Union, L 119,” 4 May 2016.

[15] SynchroniCity, *D3.4,* SynchroniCity, 2018.

[16] SynchroniCity, *D2.2,* SynchroniCity, 2018.

[17] SynchroniCity, *D2.1,* SynchroniCity, 2017.