



Rational decomposition and orchestration for serverless computing

Deliverable 5.4

Technology Library II

Version: 1.0

Publication Date: 31-March-2021

Disclaimer:

The RADON project is co-funded by the European Commission under the Horizon 2020 Framework Programme. This document reflects only authors' views. EC is not liable for any use that may be done of the information contained therein.

Deliverable Card

Deliverable	5.4
Title:	Template Library
Editor(s):	Matija Cankar (XLAB)
Contributor(s):	Michael Wurster (UST), Vladimir Yussupov (UST), Matija Cankar (XLB), Anže Luzar (XLB), Špela Dragan (XLB), Chinmaya Kumar Dehury (UTR), Pelle Jakovits (UTR), Hans Georg Næsheim (PRQ), Giorgos Giotis(ATC), Deepsubhra Guha Roy (UTR)
Reviewers:	Damian A. Tamburri (TJD), Giorgos Giotis (ATC)
Type:	Report
Version:	1.0
Date:	31.3.2021
Status:	FINAL
Dissemination level:	Public
Download page:	http://radon-h2020.eu/public-deliverables
Copyright:	RADON consortium

The RADON project partners

IMP	IMPERIAL COLLEGE OF SCIENCE TECHNOLOGY AND MEDICINE
TJD	STICHTING KATHOLIEKE UNIVERSITEIT BRABANT
UTR	TARTU ULIKOOL
XLB	XLAB RAZVOJ PROGRAMSKE OPREME IN SVETOVANJE DOO
ATC	ATHENS TECHNOLOGY CENTER ANONYMI BIOMICHANIKI EMPORIKI KAI TECHNIKI ETAIREIA EFARMOGON YPSILIS TECHNOLOGIAS
ENG	ENGINEERING - INGEGNERIA INFORMATICA SPA
UST	UNIVERSITAET STUTTGART
PRQ	PRAQMA A/S

The RADON project (January 2019 - June 2021) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825040

Executive summary

This document presents the second and final iteration of the Technology Library deliverables, which focus on two main points: i) the presentation of the tools for TOSCA infrastructure as code (IaC) and FaaS content handling and ii) the short description of the TOSCA IaC and FaaS content.

The technology library tools and content were designed according to the user requirements and as such, it is divided into two main groups: serving TOSCA content on one end and application binary artefact on other. The first is the Template Library, which is composed of two parts: RADON particles and the Template Library publishing service. Both TOSCA artefact repositories can run independently or they can supplement each other. The Template Library focuses on the management of the TOSCA IaC artefacts and service templates. The second technology library tool is the Function hub. The function hub provides a place to store and manage reusable FaaS application artefacts, i.e., Functions. The Template Library and the Function Hub can be used by a developer as an artefact catalogue for developing large FaaS applications.

The tools are presented with main technical objectives, concepts and examples of usage, based on the standard interfaces they support, e.g. REST API, CLI or GUI. The document mostly focuses on the improvements of the tools from the previous, alpha version.

Glossary

FaaS	Function as a Service
IaC	Infrastructure as Code
RADON Module	A TOSCA entity type. This is one representative from the set of node types or policy types, etc.
Template Library	An umbrella term for managing (storing) the templates (entity or service).
RADON Particles	A public repository example of Template Library
TPS	Template (Library) Publishing Service
Application blueprint	TOSCA service template, application composed with TOSCA Modules.

Table of contents

Introduction	7
Deliverable objectives	7
Overview of main achievements	7
Structure of the document	7
Technology library	8
RADON Particles - Template library public repository	9
RADON Template library publishing service (TPS)	10
Objectives and access	10
TPS Architecture - update	11
TPS REST API	12
Endpoint group updates	13
TPS access and security updates	14
TPS REST API endpoint and usage changes	16
TPS CLI	19
TPS GUI	21
TPS RADON IDE plugin	23
Installation	23
TPS Plugin Usage	24
Template library content	28
FaaS abstraction layers	28
AWS modules	28
Azure modules	29
GCP modules	29
OpenFaaS modules	30
Other modules	30
Data Pipelines (UTR)	31
Monitoring (ATC)	32
Function Hub	33
Architecture	33
Interaction surface	34
Function Hub Content	36
Conclusions	37

Current requirement fulfilment status and future work.	38
References	39
Appendix A	40

1. Introduction

The new era of application development focuses not only on the reusability of code with libraries, but also on the possibilities to fast maintain and deliver the application to the end users. The application updates should be small, incremental and seamless for the user when possible [Lar03]. To achieve this level of high maintainability, reliability and flexibility, the application code and the code for application deployment must be well organised in small chunks and created on a strictly defined purpose. The RADON project follows the aforementioned design approach and as such, provides the Technology library tools for developers, to give them the function and IaC catalogues of reusable artefacts that developers can use while developing the applications and organise their code parts for the future.

1.1. Deliverable objectives

The main objective of this deliverable is systematically revisiting the Technology Library tools that were introduced in the previous deliverable D5.3, and emphasise the main updates. The list of objectives is the following:

- Present the general concepts of RADON technology library approach with a position of the following tools:
 - Template Library - RADON Particles
 - Template Library Publishing Services
 - Function hub
- For each tool present the
 - Requirements and basic concepts
 - Available services and interfaces
 - Content coverage
- Overview of achieved requirements in Y2.

1.2. Overview of main achievements

- Presenting the final design of the tools and content
- Presenting the tools capabilities and features.

1.3. Structure of the document

The document continues with a preface of division of Technology Library on crucial tools i.e., Template Libraries and Function Hub. Further, Section 3 is dedicated to the RADON particles, Section 4 to the Template Library Publishing system, section 5 presents the Template Library Content and Section 6 presents the advances in Function hub. Section 7 concludes the document.

2. Technology library

The final version of the technology library will explain the organisation of the application and IaC content inside the RADON environment. The technology content that is managed by the tools that we jointly call the Technology Library are the following (see [Table 1](#)).



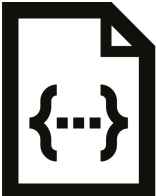
	<p>Community IaC repository - the RADON Particles provides the way to develop, test and contribute to the IaC modules and IaC service templates. The RADON particles are the community version of the RADON tool, that we refer to as Template Library.</p>
	<p>IaC Publishing repository - the Template Library Publishing System (TPS) is a set of tools that provide users an ability to share, search and reuse the TOSCA content in the form of IaC modules and IaC service templates. The improvement from the Community IaC repository is that the publishing system has a search system, special CLI and GUI that helps users to find and to standardise the process of publishing and sharing the TOSCA content. The TPS is an advanced and enterprise ready RADON tool that we refer to as a Template Library.</p>
	<p>FaaS artefact repository - the Function hub is a service for sharing the different FaaS applications that can be integrated into a larger FaaS workflow. This service eases the development of FaaS applications as developers are able to re(use) already developed functions from the catalogue.</p>

Table 1. Technology content managed by Technology Library tools

The following sections will in detail describe the final versions of the Technology Library components.

3. RADON Particles - Template library public repository

From a technical perspective, the RADON particles repository follows the same design decisions as discussed in the previous deliverables, namely D5.3 Technology Library I and D4.3 RADON Models I. Hence, the majority of efforts related to the RADON Particles were focused on (i) improving the existing types and extending the repository with new modeling constructs, and (ii) enhancing maintainability and reusability of available modeling constructs.

While the details related to the former aspect are already discussed in the deliverable D4.4 RADON Models II, it is worth highlighting the maintainability and reusability aspects of RADON Particles. The employed structure follows the same layout as required by the RADON GMT and supported by the TPS, meaning that it is possible to easily publish modeling constructs contributed by the community to the TPS, e.g., synchronising the state of TPS based on each new release of RADON Particles or by publishing a selection of distinct types using TPS' API. Furthermore, contributing new types essentially requires following the TOSCA specification and providing Ansible-based implementation logic to ensure types' compatibility with the RADON Orchestrator. More details on usage and extensibility of RADON Particles are provided in the form of guidelines in the D4.4 RADON Models II.

Additionally, types published in RADON Particles undergo a standard validation procedure which ensures that the contributed constructs (i) follow the repository structure required by the RADON GMT and supported by the TPS, and (ii) comply with the TOSCA YAML v1.3 specification including the YAML format requirements. Technically, this is achieved in a semi-automatic manner: pull requests have at least one reviewer who is also responsible for verifying the types' validity using RADON GMT which has a set of validators and a "touch" functionality that normalizes all imported types by adding the missing information and enforcing the same file layout. Moreover, there is a nightly automated test that executes the "touch" on a clone of the RADON Particles repository to validate the integrity of the types¹.

¹ <https://github.com/radon-h2020/radon-particles/blob/master/Jenkinsfile>

4. RADON Template library publishing service (TPS)

In this section we present the improvements of the Template library publishing system (TPS), which was introduced in D5.3 “Technology Library II” and which was incrementally improved following the revised requirements in the scope of WP5. From a high-level perspective, the TPS provides a standalone TOSCA IaC content publishing service that can be freely used by any TOSCA tool, for example RADON graphical modeling tool, RADON IDE, orchestrator or any other non-RADON TOSCA tool.

There have been significant updates to the TPS. Among the most significant ones are the introduction of the browser GUI (Section 4.5) and RADON IDE plugin (4.6), along with full integration with RADON authentication servers (4.3.2). The API and CLI have both also been updated with new features, such as public access (4.3.2) and the support for user groups (4.3.3). Usability and consistency were also significantly improved across all user experiences.

4.1. Objectives and access

From the previous version of TPS we revisited the requirements and discussed those with the users and the DevOps community. The result is the updated and more defined requirements that can be grouped in the following sets:

Advanced content management

- IaC content management actions:
 - Store & Publish & Versioning: stored content must be published with version to distinguish it with the updates and new releases).
 - Search: Users must be able to search through the content in a seamless way.
 - Dispatch: Users must be able to get the content through downloads or services.
- IaC content to be managed:
 - Module templates and implementations (TOSCA artefacts + executor instructions, and scripts, e.g. Ansible playbooks, Bash scripts, etc.).
 - Application blueprints (TOSCA service templates, e.g., CSARs).

Integration capacity

- Providing an API for integration with other tools and services.
- Providing a CLI that helps users or scripts to access it inside CI jobs (or better, they use API).
- Plugins: provide an Eclipse Che plugin.
- Integrate OpenID capabilities, e.g. KeyCloak integration.

Hide complexity

- Users should be able to get the content with simple CLI calls.

- Users should be able to search the content through CLI or GUI.

Business ready

- Open possibilities to group IaC content, publish it privately and share privately among the users, giving the opportunity for private dispatch of the content.

The presented requirements are well defined, except the business related one, which depends on the final business model that could be applied on the concept of the IaC sharing and developing. In this sense the business ready requirement is taken into account as a possible direction how the application can be tailored at the time when the amount of content is improved.

RADON Template Library publishing service consists of the following services:

- **REST API**² points to the REST API server, where API requests are processed;
- **Swagger UI**³ provides a detailed API documentation with WEB access for issuing the API commands to the Template Library publishing service;
- **CLI**⁴ is a PyPI package that brings the ability to invoke the REST API through CLI commands from the user's console;
- **GUI**⁵ visually presents the data stored in TPS;
- **RADON IDE plugin**⁶ (Eclipse Che plugin for RADON TPS) enables calling TPS actions from RADON IDE/Eclipse Che;
- **Sphinx docs**⁷ provides the general documentation and description of TPS together with user manual and examples.

4.2. TPS Architecture - update

Revisiting the objectives of the TPS and the discussions with the users and practitioners resulted in the updates of the TPS architecture. The TPS is a specialised and improved form of the RADON Particles repository. The updated architecture of the TPS is presented in [Figure 1](#), which we will present in bottom-up approach. The TPS core is presented in the right-hand of the picture showing the main parts, namely:

- File store for TOSCA IaC content.
- Database for user metadata and TOSCA IaC metadata, used for searching or describing the IaC to the user.
- Management tools for Users, IaC Modules, IaC Service Templates and Abstraction layers (set of modules to provide cover specific technology or provider).

² <https://template-library-radon.xlab.si/api/>

³ <https://template-library-radon.xlab.si/swagger/>

⁴ <https://pypi.org/project/xopera-template-library/>

⁵ <https://template-library-radon.xlab.si/>

⁶ <https://github.com/radon-h2020/radon-template-library-publishing-service-plugin>

⁷ <https://template-library-radon.xlab.si/docs/>

- An API module for integration of the TPS.

The left part of the [Figure 1](#) is presenting the user, services and interfaces to the TPS:

- CI/CD service
- User using:
 - RADON IDE (publishing/downloading TOSCA content directly through IDE)
 - TPS GUI (searching TOSCA IaC content)
 - CLI, console management tool for TOSCA TPS

In the middle is the Keycloak, which is used as a glue for integration of the TPS with the public or private (enterprise) identity and auth management user directories.

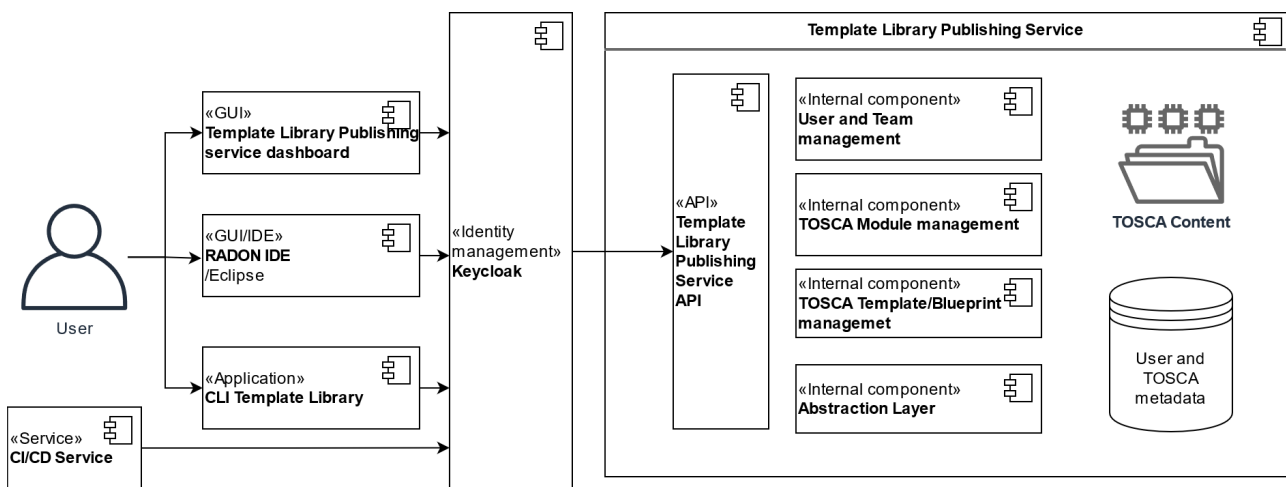


Figure 1. The component diagram of TPS showing the way user and CI/CD service can interact.

4.3. TPS REST API

This section explains the updates that were made for TPS REST API after the previous deliverable. REST API is currently accessible on template-library-radon.xlab.si/api/⁸.

Among the more significant updates are updated endpoint groups for more consistent interaction, full integration with RADON authentication servers and the support for user groups.

The OpenAPI specification for TPS REST API was also updated according to the REST API endpoint changes. All the TPS REST API endpoints can be accessed from the browser through Swagger UI on template-library-radon.xlab.si/swagger/⁹ after the user is successfully logged in. One small part of the API design organisation presented in Swagger UI server is shown in [Figure 2](#). In the following subsections we will discuss the updates of the TPS API.

⁸ <https://template-library-radon.xlab.si/api/>

⁹ <https://template-library-radon.xlab.si/swagger/>

Template Library REST API 0.6.0 OAS3

/radon/template-library-service/-/raw/master/rest-api/swagger/openapi-spec/openapi-spec-test.yml

Template Library is a place to manage your TOSCA templates of modules, blueprints and publish the implementations of the modules. The organisation and managing of the TOSCA content can be divided in three user scenarios. This website offers a Swagger UI interface for the Template Library REST API.

[Contact the developer - Website](#)

Servers

https://template-library-radon.xlab.si/api/ - template-library-public-api-server

users Operations about users

- GET /users Get list of all users or user info
- GET /users/current Get current user info
- GET /users/current/groups Get all groups for current user
- GET /users/current/templates Get a list of all templates created by the current user
- GET /users/{username}/groups Get all groups for user
- POST /users/{username}/groups Add user to group
- DELETE /users/{username}/groups Remove user from group
- GET /users/{username}/template_groups Get a list of all templates groups accessible by user

user_groups Operations about groups of users

- GET /user_groups Get all user groups or group's info
- POST /user_groups Add a new group
- PUT /user_groups Modify group
- DELETE /user_groups Delete group

Figure 2. Updated Template Library REST API design.

4.3.1. Endpoint group updates

As stated in the previous deliverable, D5.3 “Technology Library I”, TPS REST API has several REST endpoints that are grouped together so that it would be evident that they operate on the same database entities. Those endpoint groups (that are further explained in [Table 2](#)) received some minor updates - e.g. we removed the auth endpoint group after we integrated TPS with RADON

IAM because native users were no longer needed - to make the REST API more intuitive for common usage. We also added a new public endpoint group for publicly available endpoints.

Endpoint group	Purpose and description
public	Publicly available endpoints
users	Operations about users
userGroups	Operations about groups of users
templateGroups	Operations about groups of templates
templateTypes	Operations for TOSCA template types
templates	Access to TOSCA templates

Table 2. Updated REST API endpoints groups.

To improve readability and consistency we have transformed all snake_case endpoint group names, endpoint names and parameter names to camelCase, which is more natural in Java API designs.

4.3.2. TPS access and security updates

Previously, TPS was using its own native user auth for accessing the TPS services. These were no longer needed and were removed as TPS REST API is now fully integrated with RADON identity and authorization management (IAM) and is available through a RADON user registration process with Keycloak. If the user tries to initiate a request from her browser (for instance on `/users` from browser address bar) targeting any of the REST API endpoints, she needs to be logged in into her Keycloak account. When not logged in, the user will be redirected to the url (currently that is openid-radon.xlab.si/auth) where she can pick the appropriate identity provider on one of the RADON deployment (such as Azure or ENG, see [Figure 3](#)) and log in with her Keycloak user credentials. After that the user will be redirected back to the REST API endpoint the request was initiated from. Then the user is able to use all REST API endpoints manually or from Eclipse Che.

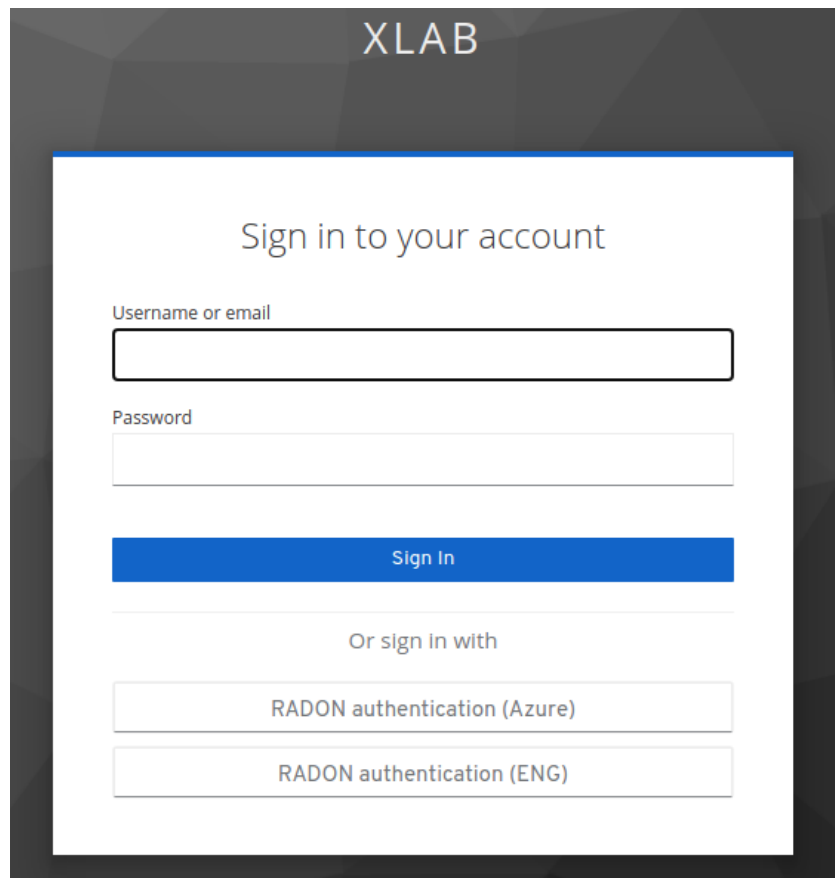
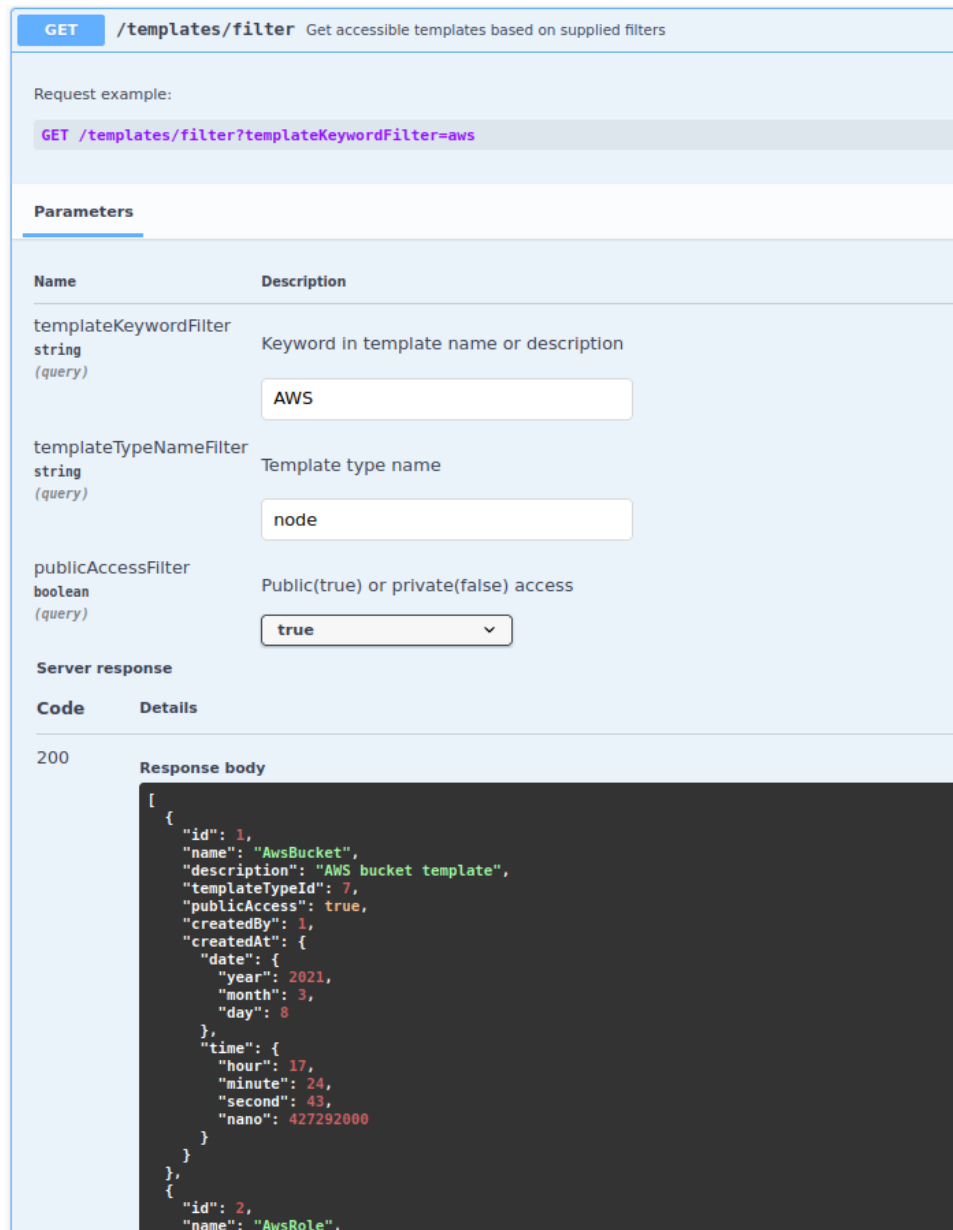


Figure 3. Keycloak authentication form for TPS.

Although all TPS REST API endpoints were initially locked with Keycloak auth, we decided to open some endpoints publicly, so that the users can browse public TOSCA IaC content without creating the account. Those endpoints start with */public* prefix and reside in a separate endpoint group. For example, one public endpoint is */public/templates/filter* which is used for retrieving and filtering public TPS modules (see [Figure 4](#)) and is also used in TPS GUI within the initial view of the web interface. Another public endpoint is */public/templateTypes* which returns all the possible types of templates (such as node, relationship, requirement and so on) that are based on different kinds of TOSCA entities from TOSCA Simple Profile in YAML v1.3¹⁰. The other public endpoints are for GET requests on */public/templates/versions* and are used to download public TPS modules.

¹⁰ <https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/cos01/TOSCA-Simple-Profile-YAML-v1.3-cos01.html>



GET /templates/filter Get accessible templates based on supplied filters

Request example:

```
GET /templates/filter?templateKeywordFilter=aws
```

Parameters

Name	Description
templateKeywordFilter string (query)	Keyword in template name or description
templateTypeNameFilter string (query)	Template type name
publicAccessFilter boolean (query)	Public(true) or private(false) access

Server response

Code	Details
200	<p>Response body</p> <pre>[{ "id": 1, "name": "AwsBucket", "description": "AWS bucket template", "templateTypeId": 7, "publicAccess": true, "createdBy": 1, "createdAt": { "date": { "year": 2021, "month": 3, "day": 8 }, "time": { "hour": 17, "minute": 24, "second": 43, "nano": 427292000 } } }, { "id": 2, "name": "AwsRole",</pre>

Figure 4. Filtering TPS modules using /templates/filter in Swagger UI.

4.3.3. TPS REST API endpoint and usage changes

One of initial goals of the Template library was to support collaboration between the users. In TPS it is possible that users, who work on common modules or are somehow connected, are added to proper teams that can be created along the way. Here we made some changes from the last deliverable as we have separated user groups and template groups. To create a new group of users someone can use a POST request on /userGroups (the params in the request are shown in [Table 3](#)).

Parameter	Description
name	Unique name of the group
description	Group description

Table 3. JSON request body params for adding a new user group.

The TOSCA templates in the TPS are organised by type (can be fetched through `/templateTypes` endpoint, see table [Table 4](#)). The main change in this grouping was renaming the CSAR type to blueprint, which is more general and is also more appropriate in our context. Blueprints contain all the TOSCA files, their implementations (e.g. Ansible playbooks) and all the other accompanying files and are not always compatible with TOSCA CSAR, which requires a special structure, e.g. TOSCA-Metadata/TOSCA.meta file most importantly.

Parameter	Description
data	TOSCA data type
artifact	TOSCA artifact type
capability	TOSCA capability type
requirement	TOSCA requirement type
relationship	TOSCA relationship type
interface	TOSCA interface type
node	TOSCA node type
group	TOSCA group type
policy	TOSCA policy type
blueprint	Full application blueprint (usually a zip archive or a compressed TOSCA CSAR)
other	Other definitions

Table 4. Template (TOSCA) types used in Template library.

The next step would be creating a new template and uploading its files to the TPS which is similar to the previous deliverable where we prepared an AWS S3 bucket module with a very simple TOSCA template including links to two implementations of Ansible playbooks. The `/templates` endpoint that is used for creating the template received some updates that are shown in [Table 5](#).

Parameter	Description
name	Module name (e.g. AwsBucket)
description	Module description
templateTypeName	Template type name (e.g. node)
publicAccess	Specify if module should be private/public (private templates are only visible in groups)

Table 5. Request body JSON key names for adding a new template.

Template groups present a new abstraction layer, which is a structure where the users can gather their templates and connect them with other users who will be able to add their own templates to the template group. To create a new group of templates a POST request on the `/templateGroups` endpoint can be used and the user will have to provide the params as in [Table 6](#):

Parameter	Description
name	Unique name of the group
description	Group description

Table 6. JSON request body params for adding a new template group.

To add template to the created template group use `/template/{templateName}/groups` endpoint and provide the request params below (currently only group owners can add new templates) where `groupId` and `groupName` params are mutually exclusive.

To be able to share private templates with other users it is possible to link groups of users to the groups of templates. To add access to templates in some template group for the whole user group use the `/userGroups/{userGroupName}/templateGroup/{templateGroupName}` REST API endpoint and provide the following params.

Templates can have multiple versions that get uploaded through time. The POST request on `/version/` that is used to create a new version and upload its files has been updated and the new params are shown in [Table 7](#).

Parameter	Description
templateName	Name of your created template
versionName	Unique semantic template version name (e.g. 2.5.8)

readmeFile	Optional markdown (README.md) file with instructions
templateFile	Compressed module (e.g. zip file) or single TOSCA YAML template file

Table 7. Request body JSON key names for adding a new version.

The main change above is that we removed the `implementationFile` parameter as it was unneeded since templates are usually uploaded as compressed archives using just `templateFile` parameter. The `implementationFile` was meant as an array of module's implementations (e.g. Ansible playbooks) but since modules often contain other file types it is better to have everything compressed and uploaded to the TPS without needing to link the separated implementations with the TOSCA later.

Published versions can be downloaded easily. To retrieve version info use a GET request on the `/templates/{templateName}/versions/{versionName}` endpoint and then you can use `/templates/{templateName}/versions/{versionName}/templateFile` request to get your compressed template file. To retrieve a README file for a specific template version use a GET request on `/templates/{templateName}/versions/{versionName}/readme`.

4.4. TPS CLI

With the respect to the updated TPS requirements the focus of the improvements on TPS CLI is on increasing the usability. The CLI was already introduced in previous deliverable D5.3 "Technology Library I", and since then the improvements were made in:

- More clarity and consistency of use commands
- Filters to display the IaC content (by keyword, template type and privacy setting)
- Improved and more compact outputs.
- Generation of a template for TOSCA service template.
- KeyCloak support.
- Public access, possibility of listing public templates without login.

In the previous versions of CLI, a Keycloak was already used for authentication of users along with native user authentication. As now there is no need for native users, they were removed and other RADON identity providers were enabled. When a user is authorizing in CLI, XLAB Keycloak is a default. Optional argument `--identity-provider` lists other possible providers (see [Figure 5](#)).

```

radon:$ xopera-template-library login -h
usage: xopera-template-library login [-h] [--username USERNAME]
                                     [--password PASSWORD]
                                     [--identity-provider [IDENTITY_PROVIDER]]

optional arguments:
  -h, --help            show this help message and exit
  --username USERNAME  Username of the user.
  --password PASSWORD  Password of the user.
  --identity-provider [IDENTITY_PROVIDER]
                       Specify KeyCloak identity provider ID. If argument and
                       the ID are present user will be logged in with the
                       chosen identity provider. If you do not know the ID of
                       the provider you can leave out the ID and the
                       available providers will be listed and you will be
                       able to choose one for authentication.

radon:$ xopera-template-library login --identity-provider
KeyCloak identity provider was not specified or is not available. You can pick your identity provider in the next steps.
Do you want to continue? (Y/n): Y
+List of available KeyCloak identity providers-----+-----+
| Provider number | Provider id | Provider description |
+-----+-----+-----+
| 1 | zocial-keycloak-xlab-oidc-provider-to-keycloak-radon | Log in with RADON credentials (ENG) |
| 2 | zocial-keycloak-xlab-oidc-provider-to-keycloak-radon-azure | Log in with RADON credentials (Azure) |
| 3 | zocial-keycloak-xlab-oidc-provider-to-keycloak-radon-azure-prod | Log in with RADON credentials (Azure - new) |
| 4 | zocial-provider-for-azure-new | azure new |
+-----+-----+-----+
Type in the valid provider number from the table above: 1
Using KeyCloak identity provider zocial-keycloak-xlab-oidc-provider-to-keycloak-radon (Log in with RADON credentials (ENG)).
Username: radon
Password:
  
```

Figure 5. Login options in CLI.

For creation of a service template a user can now use `template create` argument to generate a basic structure and files for TOSCA CSAR. See an example below ([Figure 6](#)) with tree structure of the created directory and files.

```

radon:$ xopera-template-library template create
Template type: blueprint
Template name: RadonBlueprint
Blueprint (TOSCA CSAR) 'RadonBlueprint' created in ./RadonBlueprint.
radon:$ tree ./RadonBlueprint
./RadonBlueprint
├── files
│   ├── create.yml
│   └── delete.yml
├── README.md
├── ServiceTemplate.tosca
├── TOSCA-Metadata
│   └── TOSCA.meta
└── 2 directories, 5 files
  
```

Figure 6. Creating a blueprint in CLI.

Another change is uniting `template` commands from `entity-template` and `service-template` to only `template` as the differences are taken into consideration with additional parameters for some operations and for others there is no difference.

A newer endpoint for filtering is used for listing templates. This enables filtering by keyword, template type and privacy of the template. See [Figure 7](#) below for an example.

```
radon:$ xopera-template-library template list --keyword aws --type node --public true
+-----+-----+-----+-----+-----+-----+-----+
| List of templates |
+-----+-----+-----+-----+-----+-----+-----+
| ID | Name | Description | Type | Public | Created by | Created at |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | AwsBucket | AWS bucket template | node | True | radon | 2021-03-08 17:24:43 |
| 2 | AwsRole | AWS role template | node | True | radon | 2021-03-08 17:30:26 |
| 3 | AwsLambda | AWS Lambda template | node | True | radon | 2021-03-08 17:30:42 |
| 4 | AwsBucketNotification | AWS bucket notification template | node | True | radon | 2021-03-08 17:31:01 |
| 5 | AwsApiGateway | AWS API Gateway template | node | True | radon | 2021-03-08 17:31:36 |
| 25 | AwsS3Bucket | AwsS3Bucket from radon particles | node | True | xopera-saas-user@email | 2021-03-11 09:57:11 |
+-----+-----+-----+-----+-----+-----+-----+
```

Figure 7. Listing templates using filters in CLI.

There are some public endpoints that enable searching and viewing information about public templates. One is `/templateTypes` that returns all types of templates that we distinguish and are based on TOSCA entities. Endpoint for listing templates is mentioned before and lists public templates using argument `template list` with filtering parameters `keyword`, `type` and `public`. A user can also view template's versions using arguments `template version` and download its files using `template get`.

4.5. TPS GUI

The Template library GUI is used for displaying and searching for IaC TOSCA modules and service templates published in the TPS. This GUI provides the user an alternative way to browse through the TPS content in addition to the other options as REST API endpoint calls, REST API Swagger UI, CLI, EclipseChe plugin. This addition enables an easier way to search for and view templates in a browser on the go.

TPS GUI is retrieving data through the TPS REST API, which was already presented in [Figure 2](#). User authorization is integrated with RADON IAM with Keycloak and is needed for accessing the private TPS content. No additional authorization is added for using TPS GUI.

The TPS GUI makes listing template information of public templates to users not logged in with Keycloak possible. All users can search public templates by keyword (see [Figure 8](#)), preview the template's last version's readme file and download compressed version files, while authenticated users can also view their private templates.

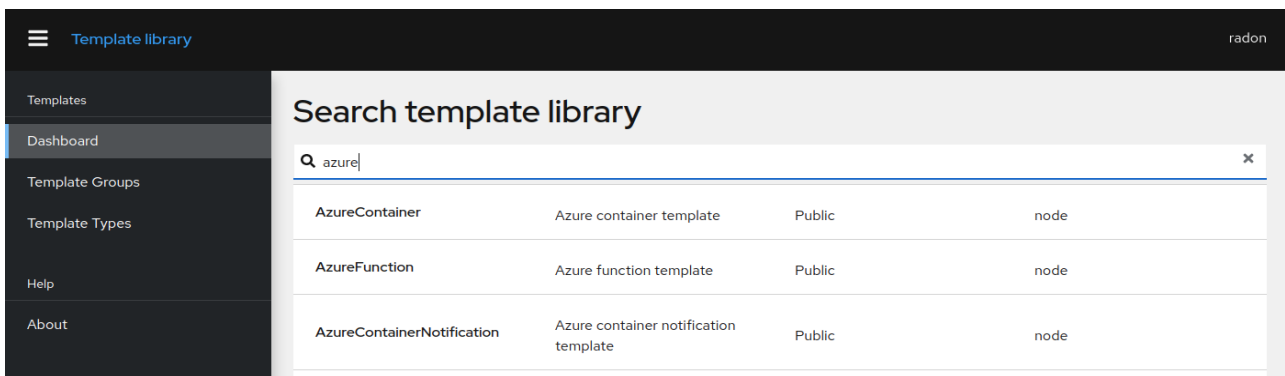
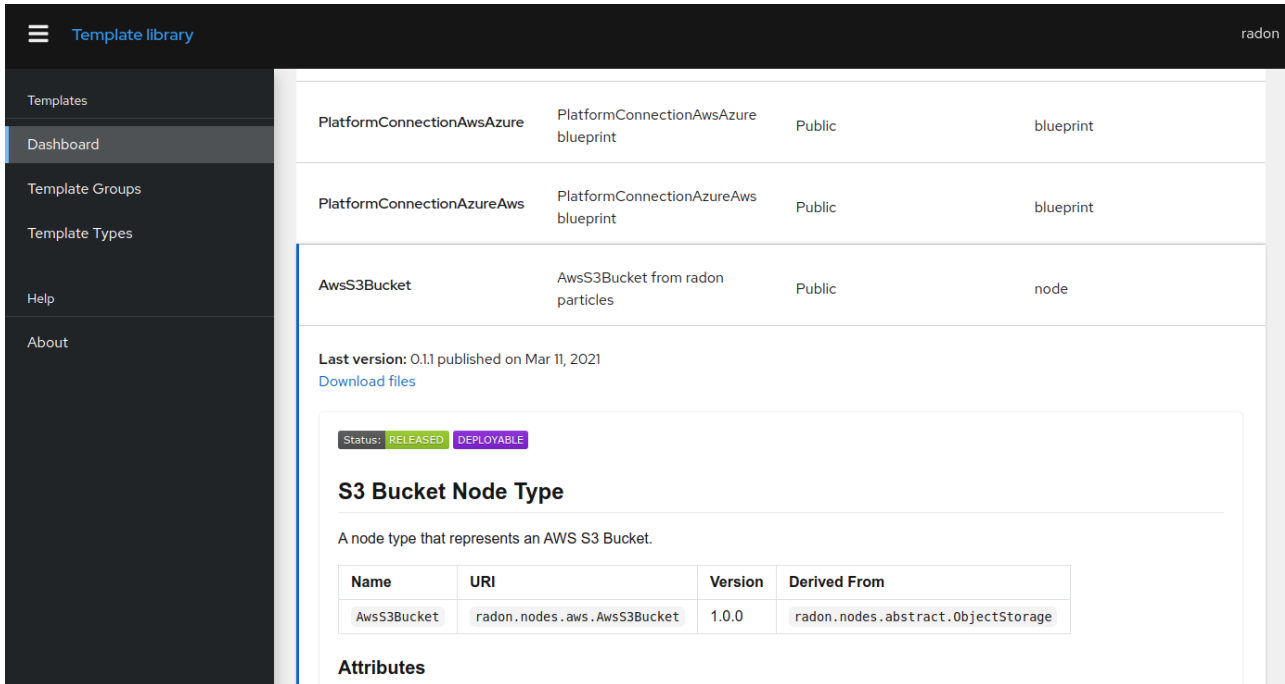


Figure 8. Searching templates by keyword.

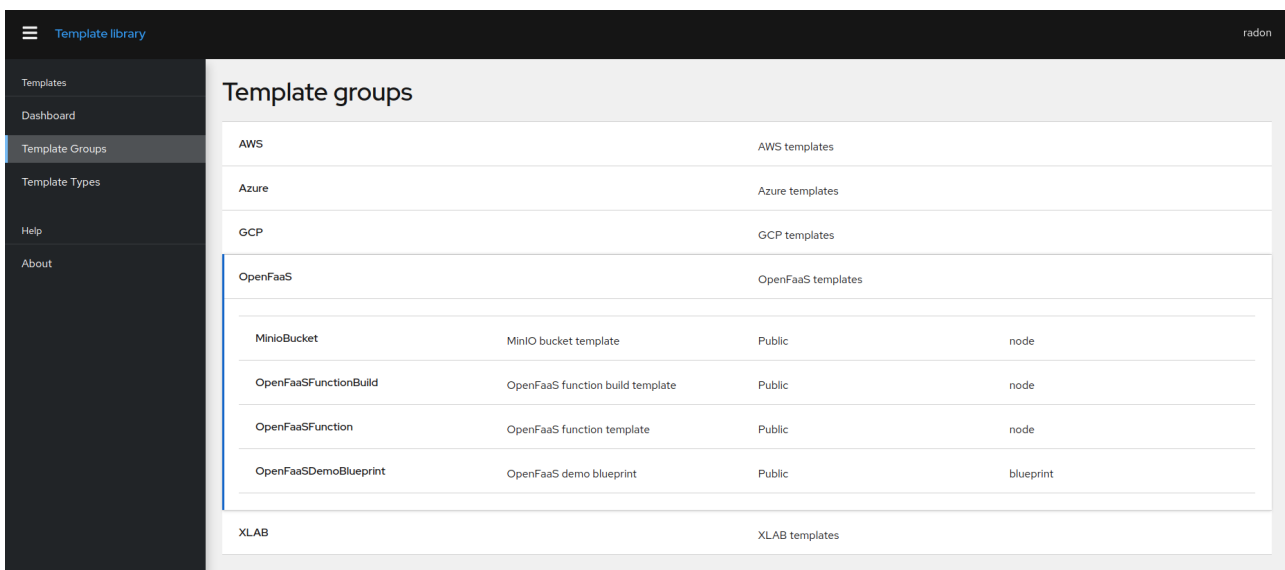
TPS GUI simplifies searching for and viewing templates in a clear way. For templates with a readme file it enables the preview of its content that usually contains information of what is included in the selected template and how to use it (see [Figure 9](#)). A click on the download link downloads compressed template files to the user.



Name	URI	Version	Derived From
AwsS3Bucket	radon.nodes.aws.AwsS3Bucket	1.0.0	radon.nodes.abstract.ObjectStorage

Figure 9. Viewing template details with readme file and download link.

All users can view lists of templates arranged by template type, while authenticated users can also view lists of templates in template groups they are members of (see [Figure 10](#)).



Name	Description	Version	Type
MinioBucket	MinIO bucket template	Public	node
OpenFaaSFunctionBuild	OpenFaaS function build template	Public	node
OpenFaaSFunction	OpenFaaS function template	Public	node
OpenFaaSDemoBlueprint	OpenFaaS demo blueprint	Public	blueprint

Figure 10. List of template groups and a list of templates in one of the groups

4.6. TPS RADON IDE plugin

The Template library RADON IDE plugin is used for communication between Template library services (mostly TPS REST API) and the RADON IDE (which is based on Eclipse Che). Using the plugin, the user is able to manage, store and retrieve his TOSCA modules (templates, blueprints - TOSCA CSAR files) and their implementations (e.g. Ansible playbooks) from Eclipse Theia or Visual Studio Code.

4.6.1. Installation

This is originally a Visual Studio Code extension/plugin, so the best way to test it locally is through VS Code's extension development host. RADON IDE is represented by Eclipse Che which uses an open-source cloud and desktop IDE framework called Eclipse Theia within the workspaces. The Eclipse Theia Cloud & Desktop IDE Platform that is used within Eclipse Che workspaces is very similar to VS Code and therefore VS Code extensions can be also used in Theia.

To install the plugin to your Visual Studio Code editor you will need the latest VSIX plugin package which you can get from RADON plugin registry GitHub repository¹¹. Then you just have to import the vsix file to the VS Code plugins.

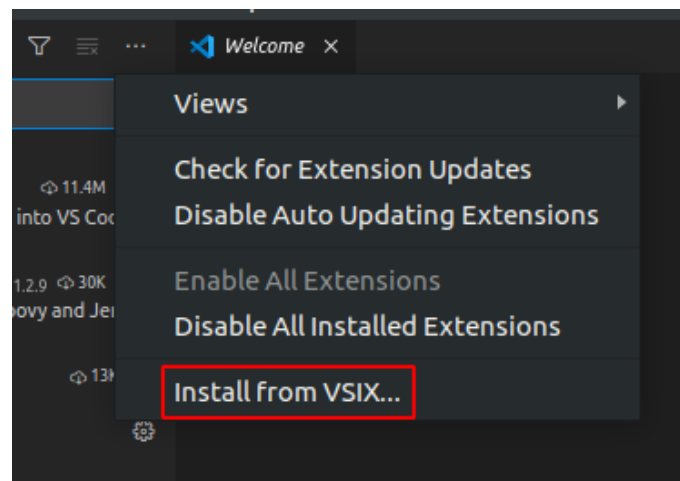


Figure 11. TPS plugin installation in Visual Studio Code.

TPS plugin is primarily meant for usage in Eclipse Theia which is the main editor in the Eclipse Che/RADON IDE. To install the plugin to Che you will need the prepared YAML devfile (which

¹¹ <https://github.com/radon-h2020/radon-plugin-registry>

also uses the prepared meta.yaml file). The files to try this can be found in Template library publishing-samples from GitHub¹².

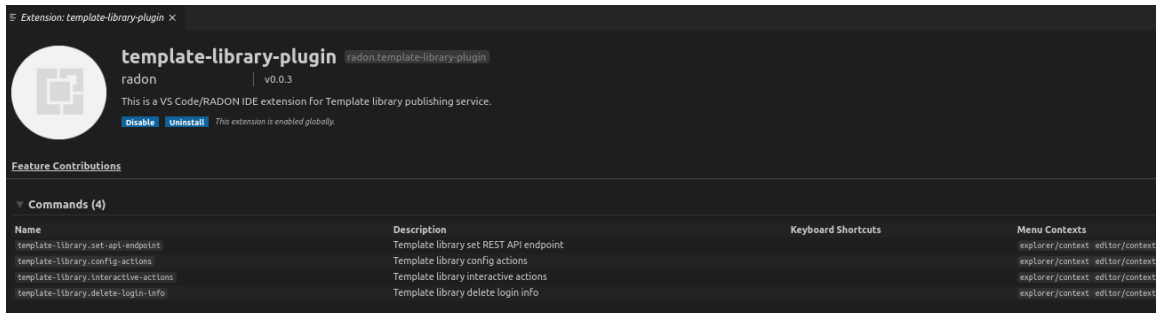


Figure 12. TPS plugin installed.

4.6.2. TPS Plugin Usage

The extension uses Template library REST API and can therefore invoke various Template library actions. Currently, supported actions are:

- setting Template library REST API endpoint
- creating and publishing TOSCA template or CSAR and its version
- downloading a specific template version files
- deleting saved login info (Keycloak cookies)

The plugin is invoked by right clicking on the file from file explorer or in the editor (see Figure 13). There are four commands that can be selected from the dropdown options and these are further explained within the next sections.

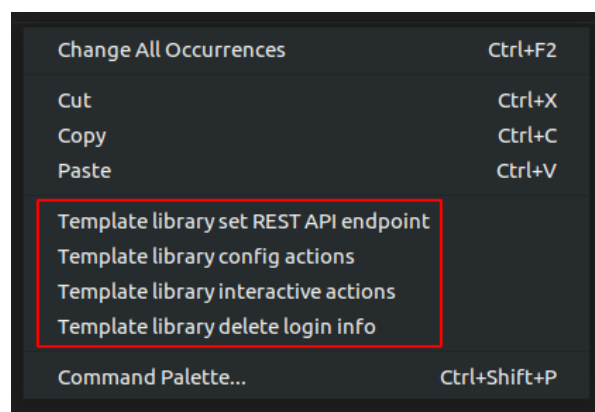


Figure 13. TPS plugin commands that show up after right click in RADON IDE.

When right clicking on any TPS plugin command (except from set API endpoint and clean login info commands), the extension will verify the user's credentials if the data has been saved. So, the first time when the user wants to use the plugin, he will be offered a set of options to select the

¹² <https://github.com/radon-h2020/radon-template-library-publishing-service-plugin>

preferred authentication method for the Template library. Since Template library auth works through Keycloak, there can be multiple login methods (see [Figure 14](#)). You can login with: XLAB Keycloak credentials RADON Keycloak/Che credentials and other identity providers that are connected to the XLAB Keycloak instance Native Template library credentials.

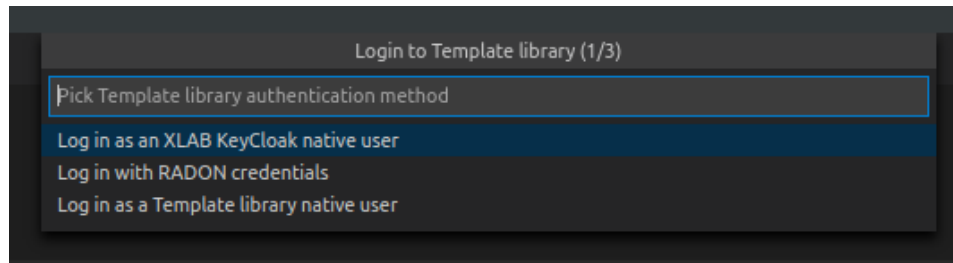


Figure 14. TPS plugin authentication methods.

If the login does not succeed, you will be warned and will have to login again. If the login succeeds, the Keycloak auth cookies will be stored into the local storage and next time you invoke any plugin command, you won't have to login again. But if you for instance set Template library API endpoint to something else or if you wish to login as another Keycloak user, then it is wise to clear saved login data by invoking the “Delete login info” action. If you log in as a native user, no data will be saved and you will have to login again every time you use the plugin.

The “Template library set REST API endpoint” command is used to set the TPS REST API endpoint that will be used for executing the TPS HTTP requests (see [Figure 15](#)). The default value here is `https://template-library-radon.xlab.si/api` which is pointing to the public TPS REST API URL. This command was meant mostly for testing different versions of TPS API so currently there is no need to change it.

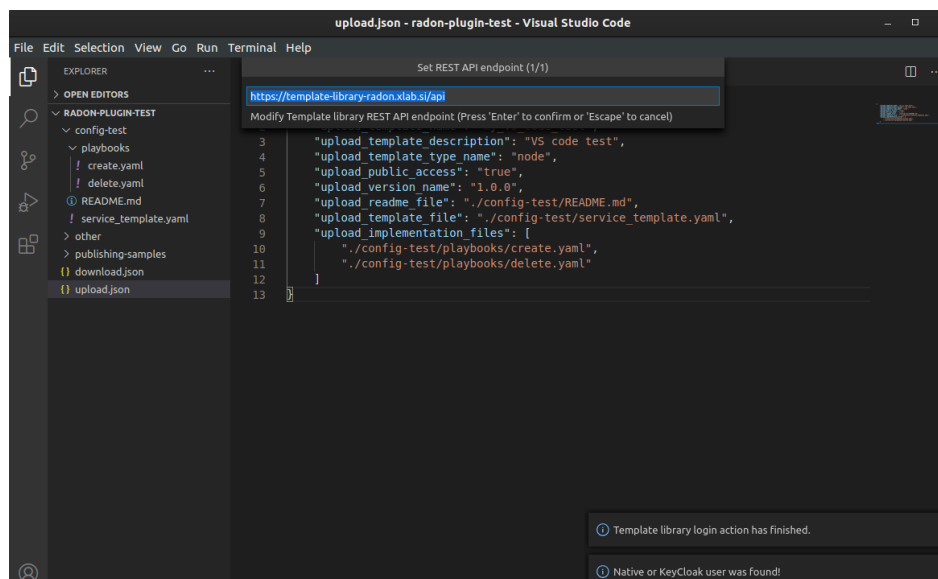


Figure 15. Set TPS API endpoint action.

If you choose the “Template library config actions” option (see [Figure 13](#) above) the TPS actions can be invoked via JSON config file. If you right clicked on the JSON file (from the editor or from the file explorer) you will be offered to choose it as a config file. If not, you will be asked to select this configuration file from other folders. A JSON object that is present in the config file should follow an exact structure which depends on the type of the action. The JSON keys specified are not mutually exclusive so you can execute multiple TPS actions with one JSON config file.

When uploading a template version you can use the keys presented in [Table 8](#):

JSON key	Description
uploadVersionName	Semantic version name (e.g. 0.5.0)
uploadReadmeFile	Optional path to README file to upload
uploadTemplateFile	TOSCA YAML service template file or compressed TOSCA CSAR

Table 8. Upload template version JSON config keys.

When downloading template version files (see [Figure 16](#)) you will get all version files (TOSCA template and playbooks) compressed in a zip file (if you provided just a CSAR without implementation files, you will get back this CSAR). For downloading template version files use the keys as in [Table 9](#).

JSON key	Description
downloadTemplateName	Name of the template you want to download
downloadVersionName	Semantic template version you want to get files from
downloadPath	Path where downloaded file will be stored

Table 9. Download template version JSON config keys.

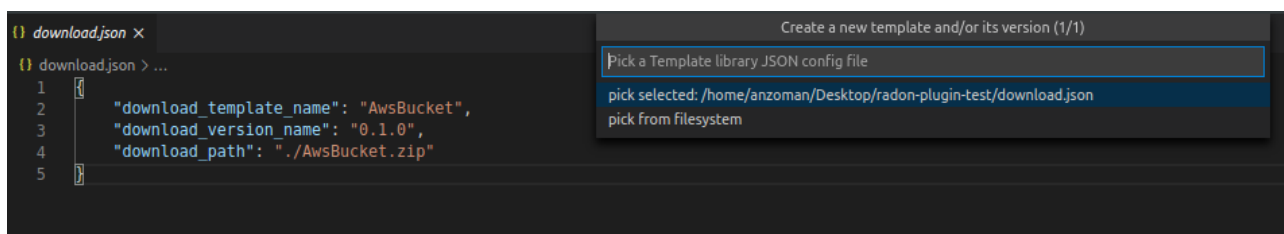


Figure 16. Download template version config action.

The “Template library interactive actions” (see [Figure 13](#)) TPS RADON IDE extension command will guide you through interactive Eclipse Theia tasks, where you will be able to create templates,

upload template versions or download version files from Template library service. An example is shown below in [Figure 17](#).

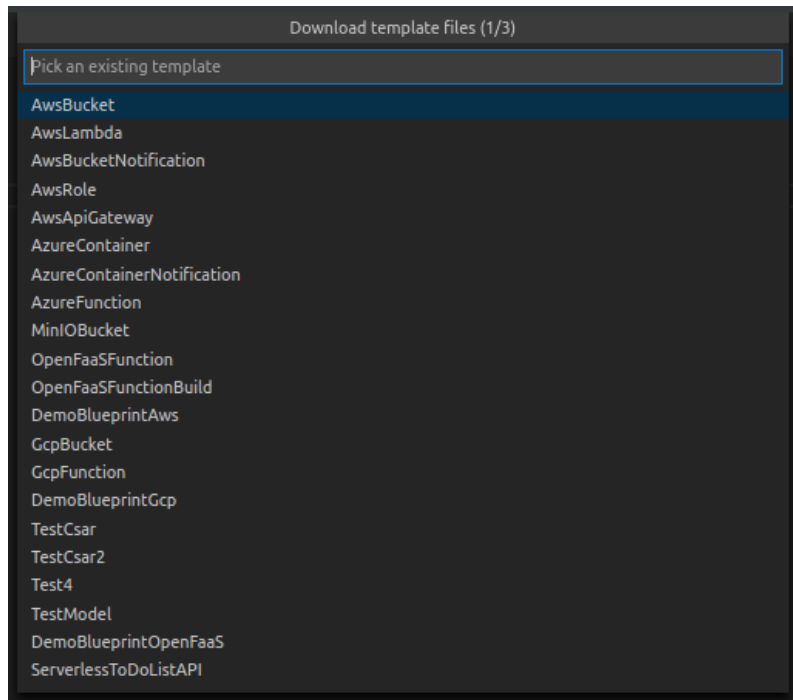


Figure 17. Interactive TPS plugin action where the user picks a template to download.

The “Template library delete login info” command will make sure that the saved login data gets deleted (e.g. Keycloak cookies). After that you will have to login again if you activate any TPS commands.

For more information take a look at [Template library Che plugin GitHub repository](https://github.com/radon-h2020/radon-template-library-publishing-service-plugin)¹³.

¹³ <https://github.com/radon-h2020/radon-template-library-publishing-service-plugin>

5. Template library content

The content that belongs to Template library is represented by modules of TOSCA templates and blueprints of TOSCA CSARs. The content is being developed within RADON Particles¹⁴ GitHub repository, where modules are separated into appropriate folders based on their TOSCA type. The Template library is used to store the metadata for modules and track their versions, so that each time when modules are updated with a new GitHub release, new versions of the modules get published to the TPS. Before the actual publishing to the TPS the modules go through a staging process, where they are validated with TOSCA orchestrator to ensure that they are deployable and compatible with TOSCA. The content of the Template Library is organised in different abstraction layers covering various technologies such as FaaS, docker or Data Pipelines. In the rest of the section we will present mentioned abstraction layers in detail.

5.1. FaaS abstraction layers

This section describes how modules in the TPS are organized into FaaS abstraction layers that correspond to different cloud providers.

5.1.1. AWS modules

AWS modules provide the necessary definitions (TOSCA templates and implementations) that are needed to deploy FaaS applications on Amazon Web Services, mostly focusing on AWS Lambda. The demo example of such an application is available on GitHub¹⁵ and the prepared CSAR is accessible in RADON particles repository¹⁶. The modules were updated and published to the public Template library instance where we included the modules shown in the [Table 10](#).

TOSCA template	Description
AwsRole	Creates a new AWS role
AwsBucket	Creates a new AWS S3 bucket
AwsLambda	Uses a zip file with function and deploys it to AWS Lambda
AwsBucketNotification	Creates bucket notification for triggering the lambda
AwsApiGateway	Prepares Swagger YAML file and deploys a new API Gateway
AwsDemoBlueprint	Contains a deployable AWS thumbnail generator application

¹⁴ <https://github.com/radon-h2020/radon-particles>

¹⁵ <https://github.com/radon-h2020/demo-tosca-blueprint-aws-lambda>

¹⁶ <https://github.com/radon-h2020/radon-particles/tree/master/servicetemplates/radon.legacy.blueprints/ImageResize>

Table 10. AWS modules that were published to the Template library.

5.1.2. Azure modules

TOSCA modules for Microsoft Azure are also accessible in the repository on GitHub¹⁷ along with the thumbnail generator blueprint. The modules that are centered around Azure Functions were published to the TPS and are listed in [Table 11](#).

TOSCA template	Description
AzureContainer	Creates storage account and necessary container on specified Azure storage account
AzureFunction	Creates FunctionApp and deploys a new function to Azure portal using Azure CLI
AzureContainerNotification	Creates an event subscription trigger for function
AzureDemoBlueprint	Contains a deployable Azure thumbnail generator application

Table 11. Azure modules that were published to the Template library.

5.1.3. GCP modules

The modules for Google Cloud Platform can be used to deploy FaaS applications to GCP using GCP Functions. These modules and the blueprint reside in GitHub¹⁸ repository and are also available in the TPS (see [Table 12](#)).

TOSCA template	Description
GcpBucket	Creates a new GCP bucket
GcpFunction	Creates and uploads GCP function
GcpDemoBlueprint	Contains a deployable GCP thumbnail generator application

Table 12. GCP modules that were published to the Template library.

¹⁷ <https://github.com/radon-h2020/demo-tosca-blueprint-azure-function>

¹⁸ <https://github.com/radon-h2020/demo-tosca-blueprint-gcp>

5.1.4. OpenFaaS modules

For OpenFaaS the modules that allow deploying FaaS applications by using OpenFaaS functions were supplied. Since OpenFaaS is a private cloud provider, we also supplied modules that are needed to set up the necessary infrastructure (such as Docker or MinIO). The modules are available on GitHub¹⁹ and are stored in the TPS (see [Table 13](#)).

TOSCA template	Description
OpenFaaSFunctionBuild	Builds and loads the given docker image to machine
OpenFaaSFunctionDeploy	Deploys docker image to OpenFaaS as a function
OpenFaaSDemoBlueprint	Contains a deployable OpenFaaS thumbnail generator application

Table 13. OpenFaaS modules that were published to the Template library.

5.2. Other modules

The next set of modules that are shown in [Table 14](#) contains simple TOSCA templates that either demonstrate some examples that are deployable with TOSCA orchestrators, or are needed as a requirement by some providers such as Docker for OpenFaaS. We have also included some useful application blueprints for connecting different cloud providers.

TOSCA template	Description
Docker	Installs docker on a target machine
Rancher	Runs Kubernetes in a Rancher Docker container and deploys the Prometheus helm chart for monitoring the Kubernetes cluster
MinioBucket	Creates necessary buckets on MinIO serve
MinioBucketNotification	Creates notification on bucket and configures MinIO server
PlatformConnectionAwsAzure	Connects the two providers where data flows from AWS to Azure using two functions (image-resize and image-watermark), each on one provider
PlatformConnectionAzureAws	Connects the two providers where data flows from Azure to AWS using two functions (image-resize and image-watermark),

¹⁹ <https://github.com/radon-h2020/demo-tosca-blueprint-openfaas>

	each on one provider
TestToscaCsar	A module to demonstrate a valid TOSCA CSAR example

Table 14. Additional modules that were published to the Template library.

5.3. Data Pipelines (UTR)

The following summarizes the list of the data pipeline TOSCA node types shown in [Table 15](#). All the node types mainly enable the users to smoothly move the data from one source location (i.e. consuming data) to another destination location (i.e. publishing data). The source and destination location can be the storage service provided by AWS, Google Cloud Platform, and Microsoft Azure cloud. In addition to that, users can use the storage unit over SFTP protocol or local file structure where. While moving the data among multiple clouds, users can process the data on-the-fly using a function as a service platform.

TOSCA template	Description
Consume GCS Bucket	Consume/Read data from Google Cloud Storage bucket.
Consume Azure Blob	Consume data from Microsoft Azure blob storage.
Consume S3 Bucket	Consume AWS data from AWS S3 bucket.
Consume MQTT	Subscribes to a topic and receives messages from an MQTT broker.
Consume Local	Consume data from the local file structure.
Consume SFTP	Consume files from the local or remote SFTP Server.
Publish GCS Bucket	Publish/Write data to Google Cloud Storage bucket.
Publish Azure Blob	Publish data to Microsoft Azure blob storage.
Publish S3 Bucket	Publish data to AWS S3 bucket.
Publish MQTT	Publish messages to an MQTT topic.
Publish Local	Publish/write data to the local file structure.
Publish SFTP	Publish data to the SFTP server.
Encrypt	Encrypt the content or the data on the local file structure.
Decrypt	Decrypt the encrypted content present in the local file structure.
Invoke AWS Lambda	Invoke AWS lambda function.

Invoke OpenFaaS	Invoke OpenFaaS function.
Copy DynamoDB to S3	Copy the data from AWS DynamoDB table to an AWS S3 bucket using AWS data pipeline.
Copy S3 to DynamoDB	Copy the data from AWS S3 bucket to an AWS DynamoDB table using AWS data pipeline.
Copy S3 to S3	Copy the data from one AWS S3 bucket to another. This can be used to copy data from one S3 directory to another directory in the same S3 bucket.
AWS SQL Activity	Runs an SQL query (script) on a database using AWS data pipeline.
AWS Shell Command	Runs a time-series or cron-like scheduled tasks using <i>ShellCommandActivity</i> in AWS data pipeline.
AWS EMR Activity	Creates a new AWS data pipeline with EMR activity which will create a new EMR cluster to run the script from AWS S3 bucket for Data analytics.

Table 15. Data Pipeline modules that were published to the Template library

5.4. Monitoring (ATC)

RADON users can use the RADON monitoring tool to monitor the resource consumption of serverless FaaS applications. The [Table 16](#) presents a list of monitoring TOSCA modules that were published to the TPS and can be used to establish the necessary components for monitoring.

TOSCA template	Description
PushGateway	Installs Docker, deploys Push Gateway Node for collecting the monitored metrics (RAM, CPU), injects Push Gateway instance to Consul service
AWSIsMonitoredBy relationship	Sets up user's proprietary Grafana dashboards, sets up the Cloud function to forward the metrics towards Prometheus Push Gateway

Table 16. Monitoring modules that were published to the Template library

6. Function Hub

Where Template Library handles deployable applications and a broader set of cloud resources following the CSAR format, Function Hub is more specific when focusing on Functions as a Service (FaaS). However, they share similarities in that they offer versioned packages in a ‘plug-and-play’ manner. The end user can browse and search for the required Function and deploy straight to any cloud vendor the Function is intended for.

In the scope of the RADON framework, function hub was intended as an open sourced repository of reusable functions. The rationale behind this collection of ready-made functions, was to simplify end-user interaction with the framework. From there, Eficode developed a state of the art serverless artifact manager. Now, the Function Hub’s functionality, with regards to RADON is encapsulated in the wider application of Cloud Stash.

6.1. Architecture

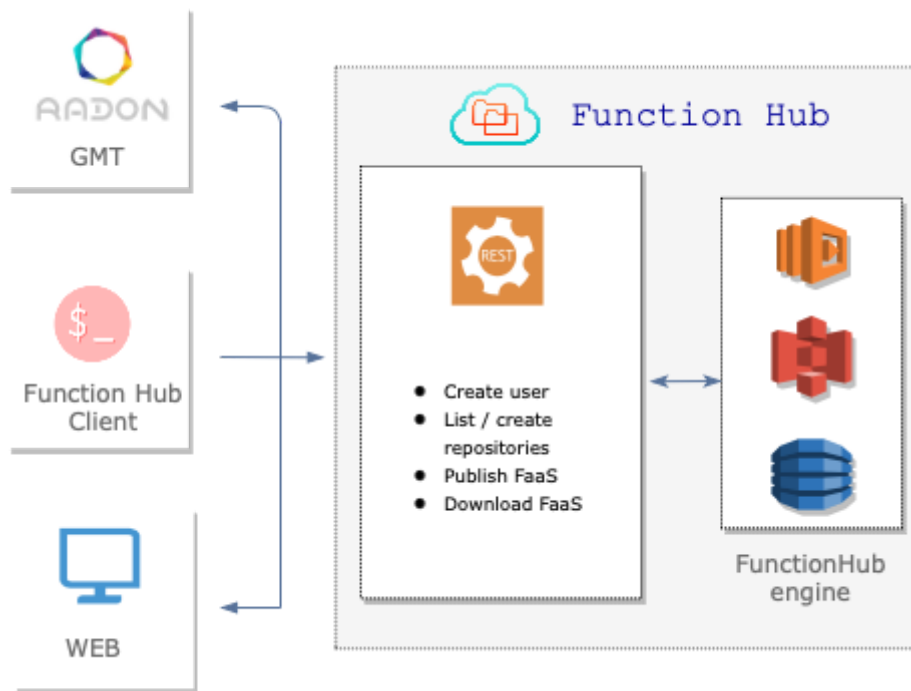


Figure 18. - High level architecture

As a result of being a part of Eficode’s use case, Serverless Artifact Manager, Function Hub shares the same architecture and design. For an in-depth description of the product, see D6.3. Hosted on AWS, the application is largely API driven, where each http endpoint constitutes a specific Lambda function with connection to table and blob storage.

6.2. Interaction surface

All interaction with Function Hub goes through the aforementioned API. From [Figure 18](#) on the left, we can see three different ways of interacting with Function Hub. In the following subsections we are presenting those in the order of the intended workflow.

6.2.1. Function Hub client

The Function Hub client was created in order to assure a consistent format of any uploaded function. The strict FaaS format ensures the ‘plug-and-play’ nature of all functions, resulting in that an end user being able to treat every function as a ‘black box’. The only necessary information can be deduced from the metadata. The content of the metadata is derived from a config.ini file which is created whenever a new function is created. Then it is up to the user to fill out the necessary information before deployment of the package. An example of such a config file can be seen in [Figure 19](#). Any missing fields or failing to authenticate throws an error.

```

functions > black_linter > ≡ config.ini
1  [REPOSITORY]
2  org = radon
3  repository = radon-functions
4
5  [FUNCTION]
6  name = black_linter
7  version = 0.0.1
8  description = Lints all python files in a git repository. Takes the env vars: git_repo and git_branch
9
10 [RUNTIME]
11 provider = aws
12 runtime = python3.8
13 handler = handler
14

```

Figure 19. - Example config.ini

The desktop client is available as a pip package²⁰ and support the following operations:

- **Create:** creates a new folder including the config.ini with the name passed as a parameter.
- **Package:** takes the folder name as an input and packages the necessary files into a zip.
- **Upload:** uploads the package to FunctionHub. Accept options like `--token`, which is used to authenticate ownership of requested the repository and `--endpoint`, for privately hosted instances of Function Hub.

6.2.2. Web interface

Once the Function is uploaded to the repository selected by the user, it will be available for browsing at cloudstash.io. The user needs to login depending on whether the repository is private or

²⁰ <https://pypi.org/project/functionhub/>

not. The same information defined in the config file upon creation is now visible through the web app. [Figure 20](#) shows how the content of a config file is presented at the web interface.



Figure 20. - Web overview

6.2.3. Integration with GMT

The relevance with RADON comes through its integration with the Graphic Modelling Tool. When creating a FaaS object, the user can either provide the source code, necessary modules and dependencies locally, or rely on the binary file and provided metadata from cloudstash. [Figure 21](#) shows how the web metadata is added to the configuration.



Figure 21. - Configure a Lambda with GMT

At the time of creating the FaaS object, the binary is downloaded and added to the final deployment package.

6.3. Function Hub Content

The purpose of Function Hub is twofold:

- a) To store personal functions in private repositories.
- b) To create a public pool of commonly useful functions.

The content of Function Hub is organized in a way following the general structure of Cloud Stash. A generic tree structure of a Function can look like this:

```

repository_name
|---function_name1
|-----|function1_version1
|-----|function2_version2
|---function_name2
|-----|function2_version1
  
```

The official RADON repository, *radon-functions*, offers quality assured functions available for the public. These are maintained by the RADON consortium and are labeled as stable. In addition we

open up for the community to build up a pool of reusable functions, freely contributing to the repository, *public-functions*.

Following the aforementioned example, we can look at the available function, *Black-Linter*. This function resides in the common repository “radon-functions”. It is a http triggered function that takes two parameters; `git_repo` and `git_branch`. Upon execution it traverses the repo passed as argument and performs code linting on all Python files. The suggested diff on the linting is then returned. Building up a library of reusable functions is a part of Eficode’s use case and will be further covered in D6.4, where more functions are explained in detail.

7. Conclusions

This deliverable gives an overview of the work done by the RADON partners inside the T5.2 task. The given results demonstrate RADON technology library services, namely Function Hub and Template Library, including the RADON Particles and Template Library Publishing Service (TPS). The services are in it’s final versions and available online to the rest of the RADON consortium and communities.

[Table 17](#) shows an overview of the level of fulfilment for each of the agreed requirements. The labels specifying the “Level of fulfilment” are defined as follows:

- (i) ✘ (unsupported): the requirement is not fulfilled by the current version
- (ii) ✓ (partially-low supported): a few of the aspects of the requirement is fulfilled by the current version
- (iii) ✓✓ (partially-high supported): most of the aspects of the requirement is fulfilled by the current version
- (iv) ✓✓✓ (fully supported): the requirement is fulfilled by the current version.

Id	Requirement Title	Priority	Level of compliance
R-T5.1-6	Support of FaaS deployment to OpenFaas	MUST_HAVE	✓✓✓
R-T5.1-7	Support of FaaS deployment to AWS cloud platform	MUST_HAVE	✓✓✓
R-T5.2-3	Support deployment to regular VMs	MUST_HAVE	✓✓✓
R-T5.2-8	Support of FaaS deployment to Google Cloud Platform	COULD_HAVE	✓✓✓
R-T5.2-9	Support of FaaS deployment to Azure	MUST_HAVE	✓✓✓

	cloud platform		
R-T5.2-11	Support deployment to microservices architecture	MUST_HAVE	✓✓✓
FR-T5.2-12	Template library publishing service filtering of entities	MUST_HAVE	✓✓✓
FR-T5.2-13	Template library publishing service should be RADON IAM compliant	SHOULD_HAVE	✓✓✓
FR-T5.2-14	Generating a basic entity with template library CLI	COULD_HAVE	✓✓
FR-T5.2-15	Publishing and retrieving entities with template library CLI	COULD_HAVE	✓✓
FR-T5.2-16:	Listing versions of a template with template library CLI	MUST_HAVE	✓✓✓
R-T5.3-3	The tool must be able to support configuring AWS EC2 auto-scaling service based on the TOSCA auto-scaling policy.	SHOULD_HAVE	✓✓

Table 17. Achieved level of compliance to RADON requirements

Current requirement fulfilment status and future work.

The requirements presented in Table 16 have been successfully fulfilled with the highest two levels of compliance. In comparison to the previous reporting period, the following requirements were updated:

- **R-T5.2-8:** Support for a FaaS deployment to a google Cloud platform has been achieved from the scratch, as the previous level of compliance was the lowest one.
- **R-T5.2-11:** The microservice applications are now supported and the deployment example is available in the xOpera documentation.
- **FR-T5.2-12:** Template library publishing service allows filtering in the API therefore it is available in CLI and GUI as well.
- **FR-T5.2-13:** TPS is fully integrated with RADON IDE. Users that use RADON IDE are automatically logged in the TPS also. Moreover, the TPS Che plugin allows RADON users to get or publish content from and to the TPS.
- **FR-T5.2-14:** The TPS CLI has a command that creates a template (skeleton of empty files) representing a new TOSCA service, which helps users to bootstrap development of new IaC projects.
- **FR-T5.2-16:** The TPS can list and manage multiple versions of one particular IaC module or service template.

- **R-T5.3-3** - Autoscaling has been described more in detail in the deliverable D5.2, where we also mention the paper²¹[Can20] of proposing TOSCA template for scaling. The autoscaling was addressed from multiple points, what we still try to achieve is to finalize this scaling approach with respect to the TOSCA standard, which currently does not propose or define any solid template, how the scaling should be done.

8. References

[Can20] Cankar M., Luzar A., Tamburri D.A. (2020) Auto-scaling Using TOSCA Infrastructure as Code. In: Muccini H. et al. (eds) Software Architecture. ECSA 2020. Communications in Computer and Information Science, vol 1269. Springer, Cham. https://doi.org/10.1007/978-3-030-59155-7_20

[Lar03] Larman, Craig, and Victor R. Basili. "Iterative and incremental developments. a brief history." *Computer* 36.6 (2003): 47-56.

[RADD6.1] RADON Consortium deliverable D6.1: Validation Plan

[RADD6.2] RADON Consortium deliverable D6.2: Initial Validation results

[TOS10] TOSCA Standards v1.3 available online:

<https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/csprd01/TOSCA-Simple-Profile-YAML-v1.3-csprd01.html>

²¹ https://doi.org/10.1007/978-3-030-59155-7_20

9. Appendix A

This appendix includes the updated list of TPS REST API endpoints sorted by their endpoint groups. The API documentation is available on <https://template-library-radon.xlab.si/swagger/>:

public:

- GET /public/templateTypes
- GET /public/templates
- GET /public/templates/filter
- GET /public/templates/{templateName}/versions
- GET /public/templates/{templateName}/versions/{versionName}
- GET /public/templates/{templateName}/versions/{versionName}/templateFile
- GET /public/templates/{templateName}/versions/{versionName}/readme

users:

- GET /users
- GET /users/current
- GET /users/current/groups
- GET /users/current/templates
- GET /users/{username}/groups
- POST /users/{username}/groups
- DELETE /users/{username}/groups
- GET /users/{username}/templateGroups

userGroups:

- GET /userGroups
- POST /userGroups
- PUT /userGroups
- DELETE /userGroups
- GET /userGroups/{groupName}/users
- GET /userGroups/{groupName}/templateGroups

- POST /userGroups/{userGroupName}/templateGroups/{templateGroupName}
- DELETE /userGroups/{userGroupName}/templateGroups/{templateGroupName}

templateTypes:

- GET /templateTypes
- GET /templateTypes/{typeName}/templates

templates:

- GET /templates
- POST /templates
- PUT /templates
- DELETE /templates
- GET /templates/filter
- GET /templates/{templateName}/groups
- POST /templates/{templateName}/groups
- DELETE /templates/{templateName}/groups
- GET /templates/{templateName}/versions
- POST /templates/{templateName}/versions
- GET /templates/{templateName}/versions/{versionName}
- DELETE /templates/{templateName}/versions/{versionName}
- GET /templates/{templateName}/versions/{versionName}/templateFile
- GET /templates/{templateName}/versions/{versionName}/readme

templateGroups:

- GET /templateGroups
- POST /templateGroups
- PUT /templateGroups
- DELETE /templateGroups
- GET /templateGroups/{groupName}/templates
- GET /templateGroups/{groupName}/userGroups