



Rational decomposition and orchestration for serverless computing

Deliverable D2.7

RADON integrated framework 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 | D2.7 |
| Title: | RADON integrated framework II |
| Editor(s): | Stefania D'Agostini (ENG) |
| Contributor(s): | Matija Cankar (XLB), Stefania D'Agostini (ENG), Stefano Dalla Palma (TJD), Thomas F. Düllmann (UST), Pelle Jakovits (UTR), Chinmaya Kumar Dehury (UTR), Anže Luzar (XLB), Law Mark (IMP), Hans Georg Næsheim (PRQ), Michael Wurster (UST), Ahmad Alnafessah (IMP), Lulai Zhu (IMP) |
| Reviewers: | Lulai Zhu (IMP), Vladimir Yussupov (UST) |
| Type: | R |
| Version: | 1.0 |
| Date: | 31-March-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 deliverable presents the final release of the RADON integrated framework. The RADON integrated framework consists of an integrated methodology and an open source toolchain, to define, evolve, and operate event-centric applications that consume serverless functions.

Glossary

| | |
|-------|--|
| CDL | Constraint Definition Language |
| CI/CD | Continuous Integration/Continuous Delivery |
| CLI | Command Line Interface |
| CSAR | Cloud Service Archive |
| CTT | Continuous Testing Tool |
| DPT | Defect Prediction Tool |
| DT | Decomposition Tool |
| FaaS | Function-as-a-Service |
| GMT | Graphical Modeling Tool |
| IaC | Infrastructure-as-Code |
| IDE | Integrated Development Environment |
| OIDC | OpenID Connect |
| SUT | System under test |
| TI | Test infrastructure |
| TL | Template Library |
| TPS | Template Publishing Service |
| VM | Virtual Machine |
| VSC | Visual Studio Code |
| VT | Verification Tool |

Table of contents

| | | |
|---------|---|----|
| 1. | Introduction | 8 |
| 1.1. | Deliverable objectives | 8 |
| 1.2. | Overview of main achievements | 8 |
| 1.3. | Structure of the document | 8 |
| 2. | Final RADON Framework Overview | 10 |
| 3. | Integrated development environment based on Eclipse Che | 11 |
| 3.1. | IDE deployment | 11 |
| 3.2. | IDE Customization and RADON Devfile | 14 |
| 4. | Final release implementation | 16 |
| 4.1. | Implementation goals and roadmap (Update) | 16 |
| 4.2. | Integrated RADON tools (Update) | 18 |
| 4.2.1. | Graphical Modeling Tool | 18 |
| 4.2.2. | Verification Tool | 20 |
| 4.2.3. | Decomposition Tool | 20 |
| 4.2.4. | Defect Prediction Tool | 21 |
| 4.2.5. | Continuous Testing Tool | 22 |
| 4.2.6. | xOpera SaaS | 22 |
| 4.2.7. | Template Library | 23 |
| 4.2.8. | Monitoring System | 23 |
| 4.2.9. | CI/CD | 24 |
| 4.2.10. | Data Pipeline plugin | 25 |
| 5. | Testing | 27 |
| 5.1. | Functional Testing (Update) | 27 |
| 5.2. | Integration Testing (Update) | 35 |
| 5.2.1. | Continuous Testing Tool - Graphical Modeling Tool/xOpera Orchestrator | 36 |
| 5.2.2. | Continuous Testing Tool - RADON IDE | 37 |
| 5.2.3. | Orchestrator - RADON IDE | 38 |
| 5.2.4. | Orchestrator - Monitoring | 38 |
| 5.2.5. | Decomposition Tool - RADON IDE | 39 |

| | | |
|---------|---|----|
| 5.2.6. | Defect Prediction Tool - RADON IDE | 40 |
| 5.2.7. | GMT - RADON IDE | 41 |
| 5.2.8. | Function Hub - GMT | 42 |
| 5.2.9. | Verification Tool - RADON IDE | 42 |
| 5.2.10. | Verification Tool - Graphical Modeling Tool | 43 |
| 5.2.11. | Monitoring - RADON IDE | 44 |
| 5.2.12. | Template Library - RADON IDE | 44 |
| 5.2.13. | CI/CD Plugin- RADON IDE | 45 |
| 5.2.14. | Data Pipeline Plugin | 46 |
| 6. | Conclusions | 47 |
| 7. | References | 51 |

1. Introduction

This document presents the final results of the activities performed within task T2.3: Integrated development environment (IDE) of WP2. In particular, this document updates the deliverable [D2.6](#) describing the final implementation of the RADON web-based IDE to support development activities and integration of the final version of the RADON tools, developed in WP3-WP4-WP5.

1.1. Deliverable objectives

This deliverable has the following main objectives:

- Describe the technical decisions to implement the final release of the RADON integrated framework;
- Outline the set of capabilities realized by the final release according to the requirements presented in [D2.2](#) and the updated integration plan described in [D2.6](#);
- Describe the performed functional and integration tests;
- Provide a user guide of the final release of the RADON integrated framework. To this scope, a companion document is supplied with this deliverable.

1.2. Overview of main achievements

The main achievements of the work reported in this deliverable are:

- The release of the final version of the RADON integrated framework, including:
 - a web-based IDE (built on top of Eclipse Che) supporting development activities in a multi-user context;
 - integration of the RADON tools in the IDE;
 - customization of the IDE with graphical elements to enable the interactions with the shared spaces of the RADON artifacts and with the RADON tools, according to the RADON methodology ([D3.1](#));
- Final integration and testing process.

1.3. Structure of the document

The rest of this deliverable is structured as follows:

- **Section 2** gives an overview of the final RADON integrated framework;
- **Section 3** provides a summary of the final set-up of the RADON IDE based on the Eclipse Che technology and describe the final RADON IDE architecture;
- **Section 4** describes the technical decisions in detail to implement the final release of the RADON integrated framework;
- **Section 5** describes the functional and integration testing performed to verify the functionalities provided by the individual components and their interactions;

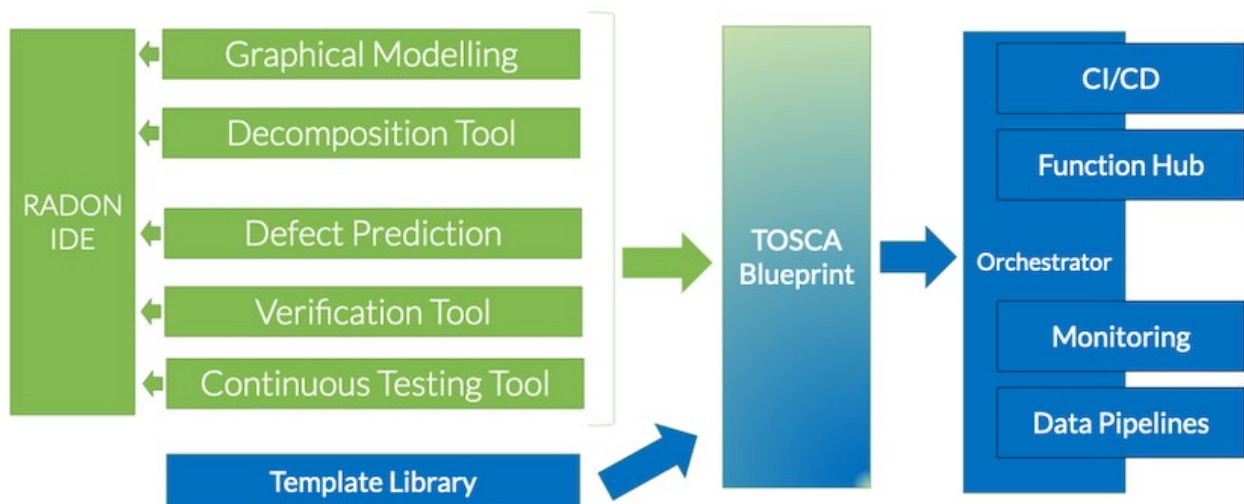
- **Section 6** draws the conclusions by summarizing its contents and the level of RADON IDE's compliance with the requirements.

2. Final RADON Framework Overview

This section provides an overview of the framework features. The RADON architecture relies on the combination of modeling environment, runtime environment, and coding environment. In RADON, we envision a model-based approach to manage and orchestrate modern, distributed, cloud-native application systems that will typically apply a microservice architecture and exploit the FaaS cloud service model. The framework uses OASIS TOSCA as a baseline to define the RADON models. A TOSCA Blueprint (shown in Figure 1) describes the topology (generated via the RADON GMT) and the orchestration of cloud applications in a declarative manner. The orchestration process occurs using the RADON orchestrator (i.e., xOpera). The deployed applications, which can support data pipelines as well, are monitored in real-time.

As described in deliverable [D3.1](#), the RADON IDE enables improving the quality assurance of serverless applications by (i) decomposing architectures and deployments, (ii) computing quality metrics and detecting code smells, (iii) applying constraints to verify, (iv) continuously testing the applications. To foster code reuse, ‘plug-and-play’ application artifacts (i.e., FaaS artifacts) can be saved and loaded using the Function Hub, a serverless package manager integrated with RADON through GMT. Developed TOSCA modules and blueprints can be stored, published, and shared via RADON Particles on Github or the Template Library Publishing Service.

Figure 1. Final RADON framework overview.



3. Integrated development environment based on Eclipse Che

As already highlighted in the deliverable [D2.6](#), the role of the IDE in the context of the RADON framework is twofold: (i) to support the development activities in a team-based context and (ii) to provide an access point for the interactions with the shared spaces of RADON artifacts and with the RADON tools.

The RADON IDE has been realized on top of the Eclipse Che¹ technology, a web-based development environment for multi-user usage, where developers can create applications without the need to install any software on their local system. As described in the previous deliverable, this technology allowed us to achieve the RADON IDE's requirements reported in the deliverable [D2.2](#). In particular, we were able to achieve both the requirements related to standard functionalities for the support of the development activities (e.g., support from different programming languages, debugging functionalities and error-checking capabilities), that the requirements strictly related to the RADON's purpose (thanks to its customization aspects).

In order to make this deliverable self-contained and up-to-date, Section 3.1 provides a summary of the final set-up of the Eclipse Che instance, while Section 3.2 describes in detail the development and integration activities performed to customize the final release of the RADON IDE, according to the project needs.

3.1. IDE deployment

Eclipse Che is a Kubernetes² native IDE, available in two modes: (i) *single-user mode (non-authenticated Che)*, and (ii) *multi-user mode (authenticated Che)*. The former is a lighter option most suited for personal desktop environments, whereas the latter option is more suited for organizations and developer teams. An instance of Eclipse Che in multi-user mode has been deployed in a virtual machine (VM) hosted by ENG using Minikube (a tool that runs a single-node Kubernetes cluster in a virtual machine) to set-up Kubernetes. A detailed description of the deployment steps was provided in [D2.6](#) (Appendix 1).

This VM has a public IP address and the following main characteristics:

- a Centos 7 machine with 8 GB RAM and 80 GB disk space;
- the installed software is: Java 8, Docker³, Kubectl (the Kubernetes command-line tool), Minikube, chectl (the command-line tool for managing a Che server and its development workspaces).

Figure [2](#) depicts the main dashboard of Che instance, reachable by the authorized users. A member of the ENG team, as the administrator of the Che instance, has created a set of accounts for the different units involved in the project in order to have access to it. Moreover, a form⁴ has been

¹ Eclipse Che - <https://www.eclipse.org/che/>

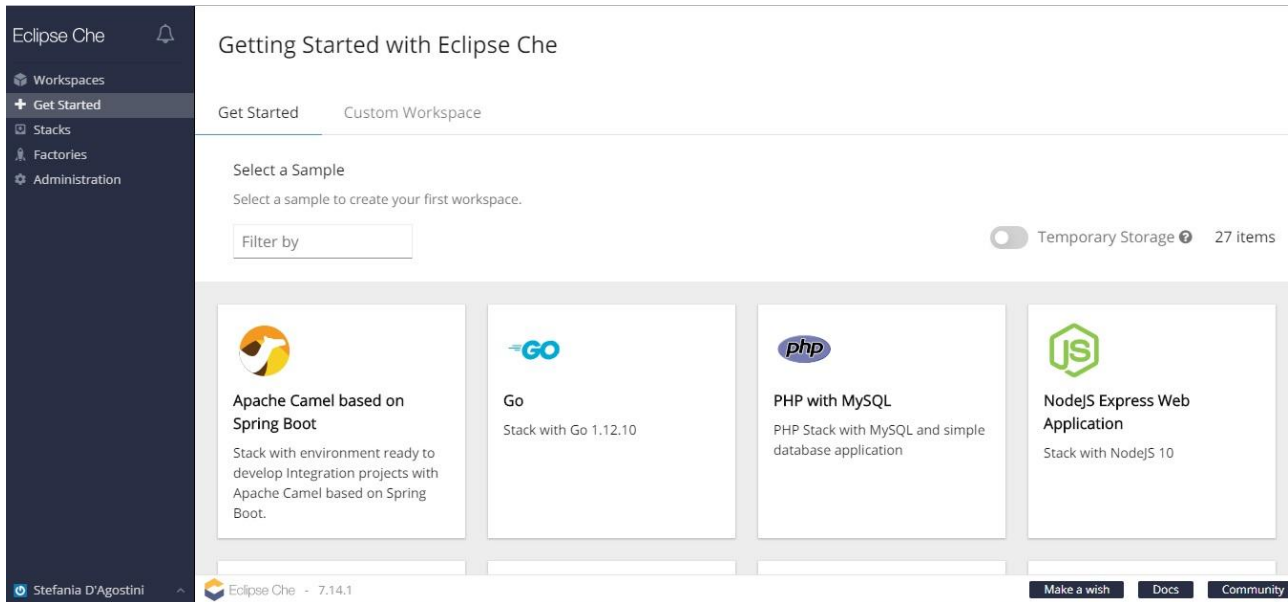
² Kubernetes - <https://kubernetes.io/>

³ Docker - <https://www.docker.com>

⁴ <https://mailchi.mp/fe5357445dba/radon-ide-access-request/>

added in the RADON web site to enable people external to the consortium to make RADON IDE's access requests. Once a new access request arrives to the administrator, a new account is created and the credentials are sent to the email specified in the form.

Figure 2. RADON IDE Dashboard.



The user accounts and security aspects are managed by Keycloak⁵, the identity and access management tool used by Eclipse Che to create, import, manage, delete, and authenticate users. Figure 3 depicts the Keycloak dashboard from which the administrator can configure users, groups, authorization and authentication capabilities for the RADON IDE.

Keycloak has also been used to secure the interaction with the *xOpera SaaS Orchestrator* and *Monitoring Tool* integrated in the RADON IDE. Indeed, these tools require authentication for the users that invoke them from the RADON IDE. As depicted in Figure 4, a new *Client* configuration has been defined for each of these tools (i.e., *xopera-saas-client* and *monitoring-client*) so that Keycloak provides security for them using the OpenID Connect⁶ protocol.

⁵ Keycloak - <https://www.keycloak.org/>

⁶ https://www.keycloak.org/docs/latest/server_admin/#_oidc

Figure 3. RADON IDE Administration Dashboard (i.e., Keycloak).

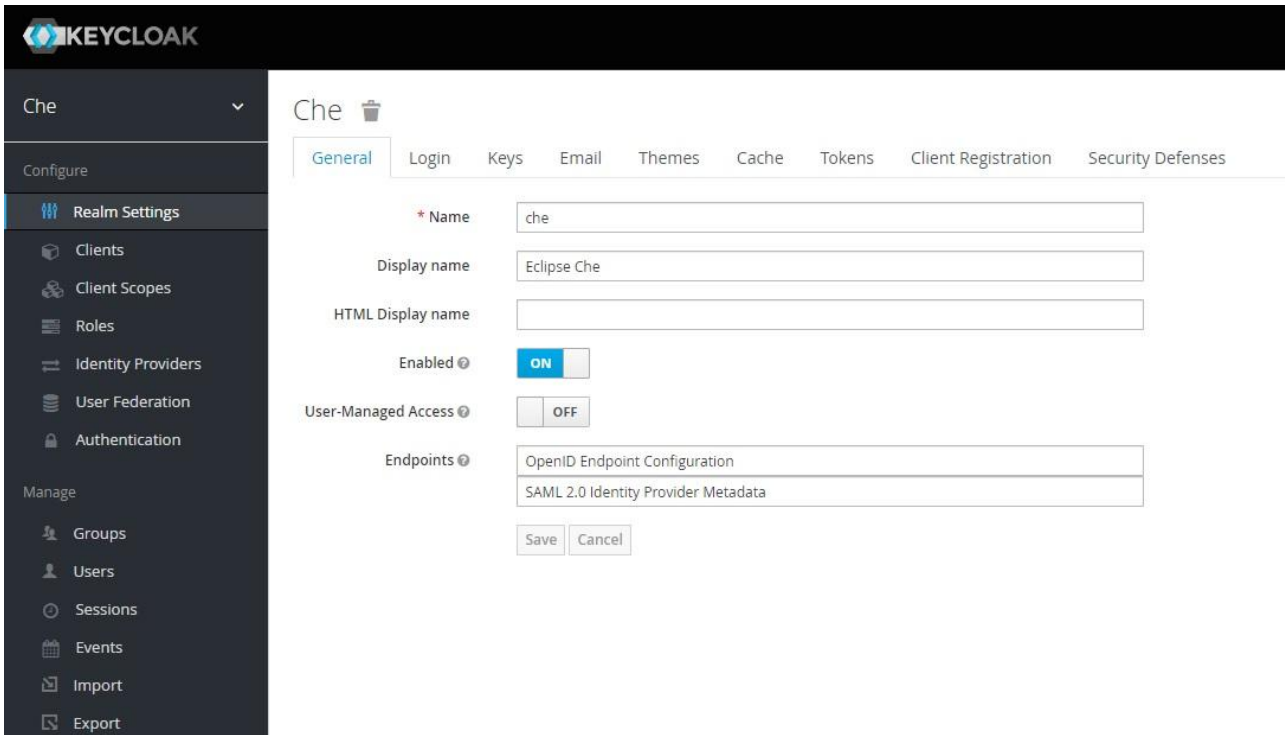
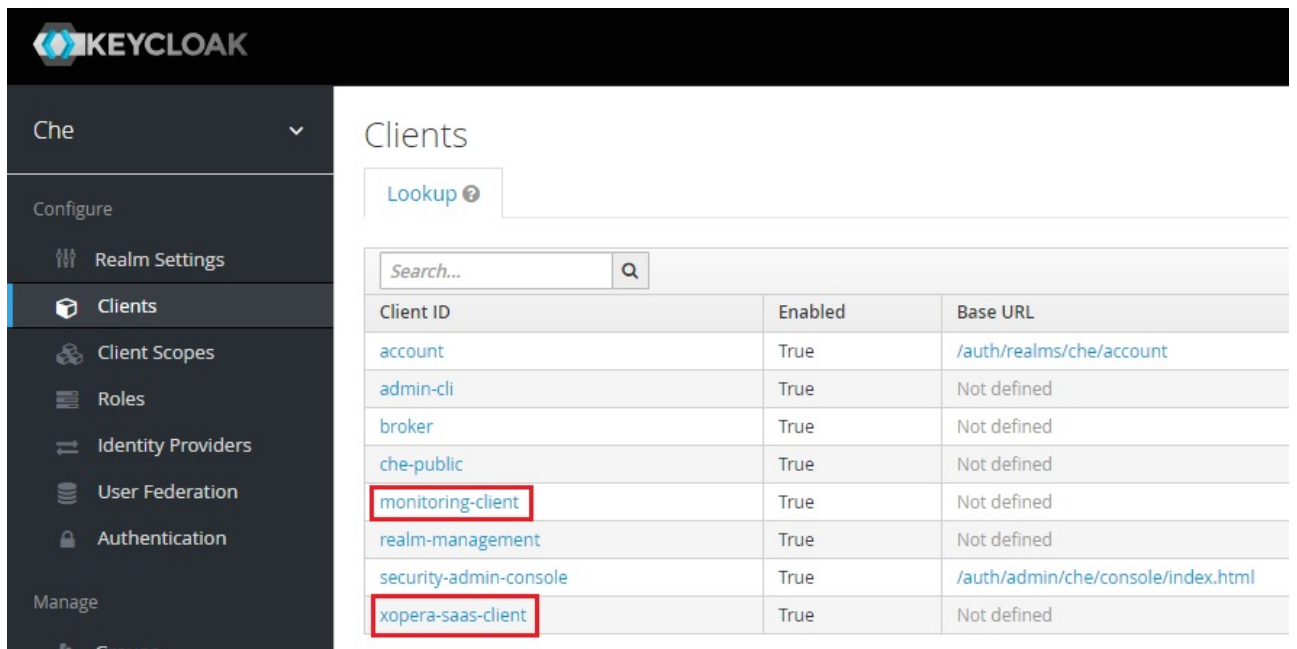


Figure 4. Keycloak Clients configuration.



3.2. IDE Customization and RADON Devfile

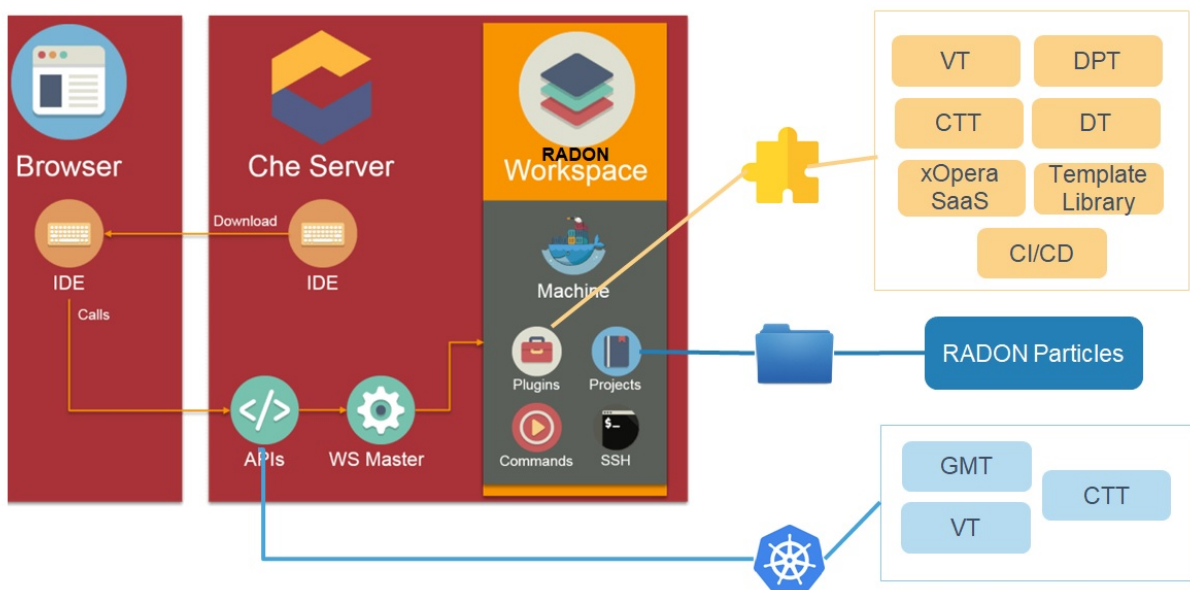
As mentioned in [D2.4](#), Eclipse Che permits to customize the development environment using a so-called *devfile*, where the configurations of the workspaces are defined to include projects, components, and commands. A devfile essentially is a portable Infrastructure-as-Code definition (IaC) which enables bootstrapping an Eclipse Che workspace. For example, a workspace can be initialized with a list of projects (i.e., Git projects) that are automatically cloned during startup. Further, additional development components (i.e., *cheEditor*, *chePlugin*, *kubernetes* containers, *docker* images) and user runtimes can be added to provide even more functionalities.

In the deliverable [D2.6](#) we presented the first version of the new Che *stack* (i.e., a runtime configuration for workspaces) that was implemented by defining a custom devfile (i.e., a *RADON devfile*) to bootstrap a RADON workspace. Figure 5 depicts the final RADON IDE architecture showing the set of projects and components (i.e., *chePlugin* and *kubernetes* containers) we implemented and included in the RADON devfile to realize the RADON workspace, according to our requirements.

In particular, the final RADON stack defined via the devfile includes the following elements:

- A project (named *radon-particles*) that clones in the RADON workspace the TOSCA modeling entities from the RADON Particles⁷ GitHub repository;
- The set of Che plugins (i.e., VT, DT, DPT, CTT, xOpera SaaS, TL, CI/CD *chePlugins*) and Kubernetes components (i.e., GMT, VT, CTT *Kubernetes container*) that have been developed to implement custom commands and backend services to interact with the integrated RADON tools.

Figure 5. Final RADON IDE architecture.



⁷ <https://github.com/radon-h2020/radon-particles>

The Che plugins and Kubernetes components developed to interact with the integrated RADON tools are described in detail in Section 4.2. Besides the Che plugins and Kubernetes components implemented to integrate the RADON tools on the RADON IDE, another Che plugin (implemented as a Visual Studio Code extension) has been developed to add a *RADON menu* on the IDE command palette. This RADON menu defines the following commands:

- A command to open the RADON documentation page by selecting the *Open Help Page* option;
- A command to open a page to show monitoring data by selecting the *Open Monitoring Page* option;
- A command to open a page to show the deployment status by selecting the *Show Deployment status* option.

The RADON menu plugin is integrated in the RADON IDE using the *chePlugin* type.

4. Final release implementation

This chapter describes in detail the technical decisions made to implement the final release of the RADON integrated framework according to the IDE requirements presented in [D2.2](#). In particular, the Che instance has been customized with the implementation of a new *RADON Stack* (i.e., a runtime configuration for workspaces), as described in the previous section.

4.1. Implementation goals and roadmap (Update)

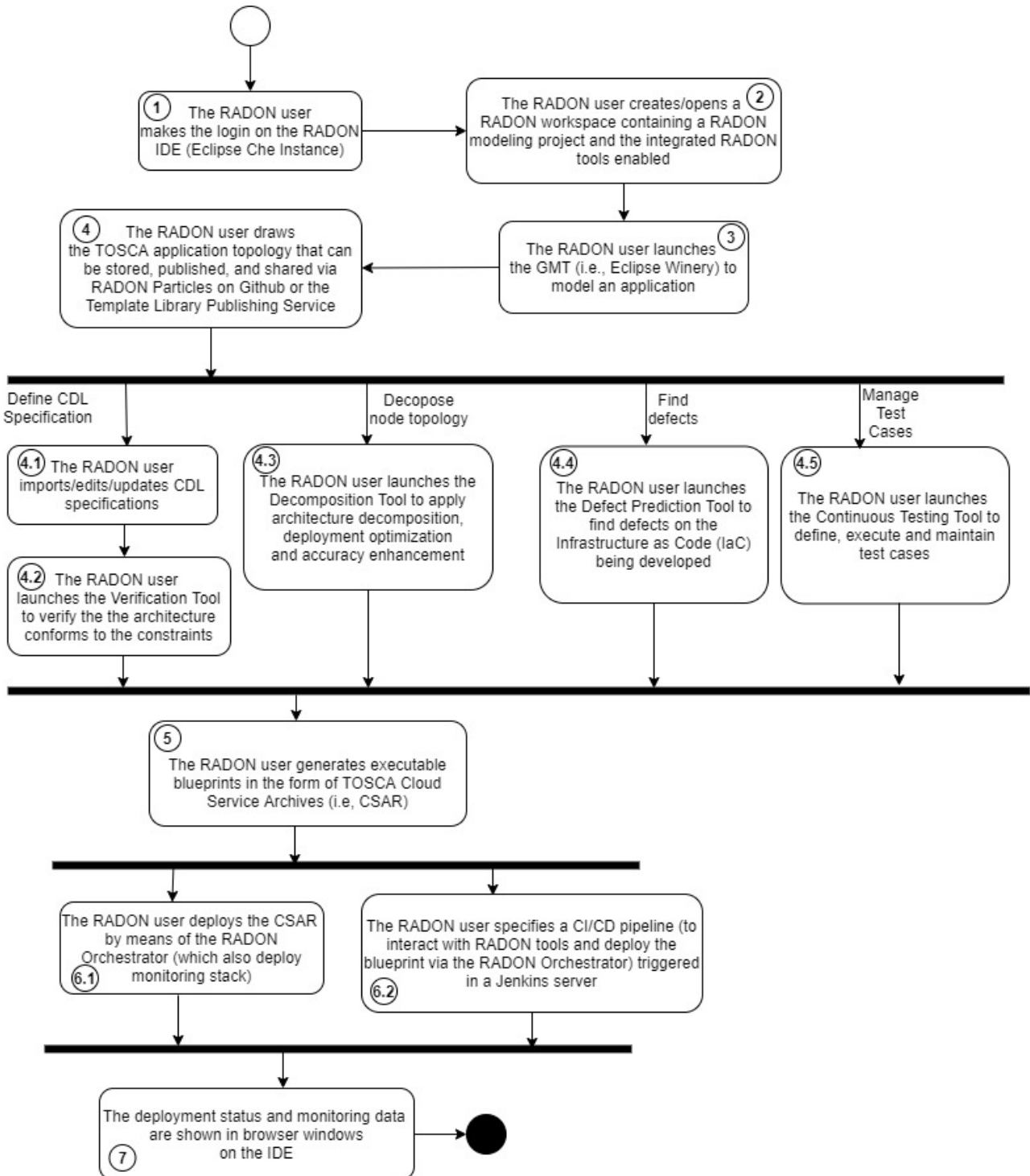
The activity diagram presented in [Figure 6](#) describes the set of actions discerned from the RADON IDE's requirements ([D2.2](#)) and the RADON workflows ([D3.1](#)), we planned to be accomplished by a RADON user interacting with the final release of the RADON framework using the RADON IDE. In particular, a RADON user can:

- create new or open existing RADON workspaces comprising modeling projects, providing access to the shared repositories of RADON artifacts and the integrated RADON tools;
- invoke the GMT from the RADON IDE to model application topologies that can be stored, published, and shared via RADON Particles on GitHub or the Template Library Publishing Service;
- run the Verification Tool, the Decomposition Tool, the Defect Prediction Tool, and the Continuous Testing Tool to verify CDL specifications on the RADON models, decompose and optimize the deployment of the applications, find the defects on IaC blueprints, and manage test cases, respectively;
- export the modeled application as deployable TOSCA CSAR files;
- deploy TOSCA CSAR files using xOpera, the RADON Orchestrator;
- start CI/CD pipelines on Jenkins servers to interact with the RADON tools and deploy the blueprint via the RADON Orchestrator;
- visualize the deployment status and monitoring data.

The above capabilities have been implemented by customizing an Eclipse Che instance to realize the new RADON Stack, which defines a RADON workspace to access RADON artifacts and provides custom menus and commands to invoke the integrated RADON tools as described in [Section 3.2](#). The RADON devfile implementing the final RADON Stack will be provided only to the authorized users after the registration process (see [Section 3.1](#)).

A RADON workflow is a repeatable sequence of actions that involves the interaction of several RADON tools. The RADON workflows are described in detail in [D3.1](#). In [Figure 6](#), we used a numbered label to map the actions a user can perform in the RADON IDE with the RADON workflows.

Figure 6. RADON workflows within the RADON IDE.



[Table 1](#) summarizes the RADON workflows supported in the final release of the RADON integrated framework. Due to the fact that all the RADON tools have been integrated in the RADON IDE, the RADON workflows are all supported.

Table 1. Support of RADON workflows (Update).

| RADON Workflow | RADON Tools | Involved steps (from Figure 4) |
|-----------------------------------|--|---------------------------------------|
| Verification | Graphical Modeling Tool Verification Tool | 1, 2, 3, 4, 4.1, 4.2 |
| Decomposition | Graphical Modeling Tool Decomposition Tool Monitoring Tool | 1, 2, 3, 4, 4.3, 5, 6.1, 7 |
| Defect Prediction | Defect Prediction Tool | 1, 2, 3, 4, 4.4 |
| Continuous Testing | Continuous Testing Tool Orchestrator Monitoring Tool | 1, 2, 3, 4, 4.5, 5, 6.1, 7 |
| Continuous Integration/Deployment | CI/CD Plugin | 5, 6.2, 7 |
| Monitoring | Monitoring Tool Graphical Modeling Tool Orchestrator | 5, 6.1, 7 |

4.2. Integrated RADON tools (Update)

This section describes the RADON tools that have been integrated in the final release by means of *Che plugins* and/or as *Kubernetes containers*.

4.2.1. Graphical Modeling Tool

Overview of provided features:

The Graphical Modeling Tool (GMT) is a web-based software solution to create, develop, and model TOSCA applications and its required ingredients. It is mainly used to compose TOSCA service templates, representing the applications that are deployed using the RADON Orchestrator. These TOSCA service templates (aka. RADON Models) consist of TOSCA entity types, which are reusable modeling entities. Both, the RADON Models as well as the TOSCA modeling entities, can be maintained using the GMT. To make it easier for RADON users to launch the GMT, we integrated it into Eclipse Che, the RADON IDE. The GMT runs as a Kubernetes container inside an Eclipse Che workspace and can be started respectively. The GMT is able to interact with files and

folders from the underlying workspace. Upon startup, the GMT initializes a "modeling project" inside the workspace, which becomes available in Che's project structure. RADON users are now able to create or adapt existing TOSCA modeling entities as well as to compose new applications. All changes are reflected in the files and folders of the created "modeling project" and can be versioned by using a public or private Git repository. Further, RADON users may want to package their applications as a CSAR before it is deployed into production by using the RADON Orchestrator. GMT offers the feature to package and save a CSAR of a selected RADON Model to Eclipse Che's workspace. This provides the possibility to process it using other RADON tools, e.g., analyze using the Defect Prediction Tool or deploy using the RADON Orchestrator.

Once a user has modeled an application using the GMT it may be required to open the underlying TOSCA file inside the RADON IDE, e.g., to modify some complex properties using a source code editor or to create a file containing CDL statements used together with the VT. The Topology Modeler component as well as the Type Management component of GMT got extended with a button which is able to launch the IDE in a new browser window with an additional URL parameter containing the information to which type or template to navigate. A respective plugin in the Che-Theia editor will take this information to directly open the folder in the project explorer.

Integration details:

The GMT is based on the open-source project Eclipse Winery. All enhancements in the course of the RADON project have been merged into Eclipse Winery's master at the time of writing. Further, a Docker build automatically builds a new Docker image⁸ of the GMT whenever new enhancements or defect fixes are pushed to Winery's master branch. Based on this, GMT can be started as a separate Docker container for each created RADON workspace. To integrate the GMT, we defined GMT as a Kubernetes component (which means that it is possible to apply existing configuration for Kubernetes) in RADON's stack "devfile".

In GMT's case, we have created a common Kubernetes deployment descriptor⁹ file (cf. winery.yaml) that essentially defines the deployment configuration of GMT.

Notably, setting the "mountSources" flag to "true" will make the project's source code available inside the underlying container. With this setting, GMT is able to access and modify the files and folders in Eclipse Che's workspace when composing new RADON Models or creating new TOSCA modeling entities. Further, GMT defines one public endpoint to reach the user interface. This configuration is required to expose GMT's user interface and public REST API to be accessible for RADON users.

To implement the "Open in IDE" button (aka. "jump to code" feature) we had to develop and register an Eclipse Che extension, which is registered within Eclipse Che's native source code editor Eclipse Theia. GMT has been extended to open a new browser window with the following URL scheme when users click on the "Open in IDE" button:

⁸ <https://hub.docker.com/r/opentosca/radon-gmt>

⁹ <https://raw.githubusercontent.com/eclipse/winery/master/deploy/che/winery.yaml>

`http://<che-theia-url>:<che-theia-port>?path=folder/containing/a/tosca/definition`

The extension is able to retrieve and parse the “path” parameter from the browser URL and tries to open the respective file in Che-Theia. It is required to package such an extension with the Eclipse Theia editor. Therefore, we set up an automated build to clone the eclipse-theia/theia project, register our custom extension, and package it as Docker container. This Docker container can be launched by a respective configuration in the RADON devfile.

4.2.2. Verification Tool

Overview of provided features:

The Verification Tool (VT) enables a user to verify that a RADON model conforms to a set of requirements expressed as a specification in the Constraint Definition Language (CDL). Users can run the *verify*, *correct* and *learning* modes of the VT by right-clicking on any file with the extension ".cdl" and clicking on the "Verify", "Correct" or "Learn" buttons in the resulting context menu. The result of the *verification*, *correction* or *learning* task is then displayed in a new output sub-window. In addition, when the correction mode is called, the VT produces a modified CSAR containing the corrected TOSCA model.

Integration details:

The VT integration has two parts: (1) VT plugin and (2) VT Kubernetes container. The VT plugin is responsible for the extensions to the Eclipse Che GUI; specifically, it adds the buttons to the context menu, which appears when a user right-clicks on a file with the extension ".cdl". The plugin is implemented as a VS Code extension. The VT Kubernetes container is the "backend" of the tool. It is running a very lightweight Sinatra web server. When a user triggers a task, the VT plugin sends a request to the Sinatra application on the container, which then calls the command line VT. The result is sent back to the VT plugin and is then displayed to the user.

Similarly to the GMT, the VT plugin and VT container are both integrated in the RADON IDE using the *chePlugin* and the *kubernetes* types in the RADON devfile.

The flag "mountSources: true" allows the VT container to read the files in the user's projects. This is necessary because the VT needs to read the files in order to verify them. The endpoint definition ensures that the VT container is available on port 5000, which enables communication between the VT plugin and VT container.

4.2.3. Decomposition Tool

Overview of provided features:

The decomposition tool is present to help in finding the optimal decomposition solution for an application based on the microservices architectural style and the serverless FaaS paradigm. It is envisioned to support four typical usage scenarios: (1) architecture decomposition, (2) deployment

optimization, (3) accuracy enhancement, (4) interference detection. The latest version allows the use of the tool to decompose the architecture of a monolithic or microservice application at the abstract level and to optimize the deployment of an application comprising Lambda functions and S3 buckets. To these ends, one can right-click on the service template (.tosca) and then respectively click the “Decompose” and “Optimize” buttons in the pop-up menu. After the corresponding procedure completes, the service template will be backed up and updated in place, and the execution information will be printed in the “Output” window (View → Output).

Integration details:

An instance of the decomposition tool is deployed and made available as a public service with a REST API. We develop a Visual Studio Code extension for users to access that service through different procedures, namely decomposition and optimization. For example, the decomposition procedure calls the API of the tool to upload the model, decompose it and download the result. The Visual Studio Code extension is installed in the RADON IDE by adding a *chePlugin* type in the RADON devfile.

4.2.4. Defect Prediction Tool

Overview of provided features:

The Defect Prediction (DP) tool enables operators to identify potentially defective IaC blueprints and their defect type. It provides a graphical user interface to run the detection on a given Ansible or Tosca blueprint and display the results. The results consist of a table showing the values for each of the extracted metrics (highlighting those that might be critical because diverging from the community standard) and the blueprint’s defect type, if any, with a set of rules to interpret the decision.

Integration details:

The Defect Prediction consists of a Visual Studio Code extension integrated into an Eclipse Che environment. The extension is packaged as an Eclipse Che Theia plugin into a sidebar container. An operator can interact with it by right-clicking on a YAML-based Ansible file and click “*Run detection*”. The results (i.e., the metrics extracted from the script and defect-proneness) will be displayed in a new active tab (commonly known as webview in VSC).

Similarly to the previous tools, the DP plugin is integrated into the RADON IDE using the *chePlugin* type in the RADON devfile.

4.2.5. Continuous Testing Tool

Overview of provided features:

The CTT IDE plugin enables RADON users to execute CTT's functionalities via the RADON IDE in a project workspace. Users specify the configuration of the CTT execution by creating a CTT configuration file in the project workspace of the system under test (SUT). A CTT execution can be started by right-clicking on a configuration file and selecting the respective menu entry. The IDE plugin then interacts with the CTT server in order to perform the usual CTT steps, i.e., deploying the SUT and the test infrastructure (TI), executing the test, and collecting the test results. During the execution, the progress is logged in the output panel, and a progress bar appears on the bottom right of the RADON IDE. After a successful execution, users will find the test results in the configured location in their workspace.

Integration details:

The CTT IDE plugin is written in Typescript, and it is integrated in the RADON IDE with a Che plugin and a Kubernetes component. A CTT server instance is automatically deployed into the RADON IDE. The communication between the plugin and the CTT server is performed via the REST API.

Similarly to the previous tools, the CTT plugin is integrated into the RADON IDE using the *chePlugin* type in the RADON devfile.

4.2.6. xOpera SaaS

Overview of provided features:

Where developers would normally use xOpera locally using the CLI, xOpera SaaS offers users a hub and execution environment for executing TOSCA modules. Offered as a multi-tenant service with full integration with RADON's authentication providers, the SaaS version of xOpera is an API and browser interface for execution in the cloud. Users can manage secrets and share workspaces with other users, thereby facilitating cooperation and enabling users to have an easier overview of their many active deployments. A feature exclusively enabled by xOpera SaaS is support for out-of-the-box scaling operations, as monitoring services can automatically be bound to URL callbacks, thus eliminating the need of provisioning a separate service for receiving scaling operation callbacks.

Integration details:

A Visual Studio Code and Eclipse Che plugin is provided to integrate the functionality of xOpera SaaS into the RADON IDE. Users are able to author and subsequently deploy TOSCA artefacts seamlessly onto xOpera SaaS by right-clicking on a CSAR and selecting "Deploy CSAR with xOpera SaaS". A prompt is then shown for the user to choose and/or create a workspace and project for deployment, and a choice to deploy the project immediately is displayed. Upon completion of

this flow, the user is presented with information about their actions and also redirected to the xOpera SaaS dashboard for further interaction and lifecycle management of the deployed artefact.

As with other such integrations, the plugin is integrated into the RADON IDE using the *chePlugin* type in the RADON devfile.

4.2.7. Template Library

Overview of provided features:

The Template Library offers a place where users can store, manage and retrieve RADON modules (TOSCA templates and their implementations) and blueprints (TOSCA CSARs) and facilitates the collaboration between TOSCA module creators. The users within the RADON IDE are able to distinct their modules by their TOSCA types (such as node or relationship types) organize them into template groups which are then accessed by the groups of users. When the users build complex applications based on TOSCA models, the strict dependencies can tightly couple specific groups of modules. New versions of modules can result in new issues that make applications, based on them, defective. The goal of Template Library is to simplify the complexity of managing multiple versions of TOSCA templates and to serve as a standalone publishing service that does not permit changes of published content and persuades users to organise each modification in a new template version. Moreover this approach also allows to have a private content and sharing of the content inside template and user groups.

Integration details:

The Template Library Visual Studio Code/Eclipse Che plugin is used for communication between Template Library and the RADON IDE. The plugin is invoked by right clicking on the file from file explorer or in the editor. For better user experience and more flexible interaction with the Template Library, the plugin has two modes of operation. With the first config mode the actions can be invoked via JSON config file, and the second is the interactive mode that will guide the user through an interactive Eclipse Theia tasks, where the users are able to create templates, upload template versions or download a specific template version from the Template Library.

Similarly to the previous tools, the Template Library plugin is integrated into the RADON IDE using the *chePlugin* type in the RADON devfile.

4.2.8. Monitoring System

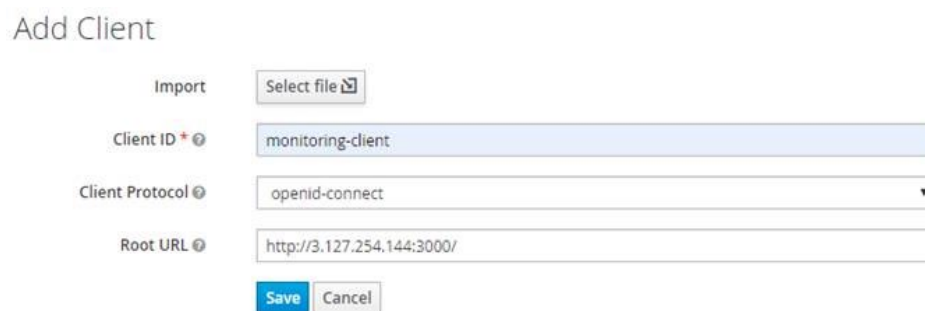
Overview of provided features:

A user can configure the monitoring of a FaaS' resource consumption by using the corresponding TOSCA monitoring nodes and relationships directly through the RADON IDE's GMT instance. Once a service template configured with the monitoring stack is deployed, a dedicated visualisation dashboard is created that can be only accessed by the user who created it.

Integration details:

The visualization dashboard of the Monitoring tool is a shared instance based on the Grafana platform. It exposes the metrics that can be configured through the TOSCA monitoring relationships. The user management of the RADON IDE is fully integrated with the Monitoring dashboard through the OpenID Connect (OIDC) protocol, which is an identity layer built on top of the OAuth2 framework. This way, the user authentication is delegated to the RADON Keycloak and upon successful authentication the RADON user is redirected to the Monitoring dashboard.

Figure 7. Keycloak configuration for adding the Monitoring dashboard endpoint as an OIDC client.



The screenshot shows the 'Add Client' configuration page in Keycloak. It features the following elements:

- Import:** A button labeled 'Select file' with a download icon.
- Client ID:** A text input field containing 'monitoring-client'.
- Client Protocol:** A dropdown menu currently set to 'openid-connect'.
- Root URL:** A text input field containing 'http://3.127.254.144:3000/'.
- Buttons:** 'Save' (in blue) and 'Cancel' (in grey) buttons at the bottom.

Moreover, a plugin that provides on the command palette of the RADON IDE a command *Open Monitoring Page* connecting with the Monitoring system, has been implemented (see Section 3.2).

4.2.9. CI/CD

Overview of provided features:

CI/CD templates are predefined pipeline templates for the execution of the RADON tools in a remote CI/CD environment. Using them, a RADON user can easily combine pipeline snippets together in order to construct a desired CI/CD pipeline reducing documentation study time and technical implementation efforts. The templates can be found in RADON's Github repository¹⁰ and are available for two different CI/CD technologies: Jenkins¹¹ and CircleCI¹². The tools covered by the templates repository are:

- CTT - Continuous Testing Tool
- VT - Verification Tool
- DPT - Defect Prediction Tool
- TL - Template Library

¹⁰ <https://github.com/radon-h2020/radon-cicd-templates>

¹¹ <https://www.jenkins.io/>

¹² <https://circleci.com/>

- xOpera - Orchestrator

A CI/CD plugin is offered in IDE and is used to invoke a CI/CD pipeline on a remote platform. The integrated plugin provides access to a Jenkins server and executes a project pipeline as it has been preconfigured using the CI/CD pipeline templates. Users generate a configuration (i.e., a yaml file) of the CI/CD by right-clicking on a CSAR from the file explorer and selecting the respective menu entry. Once the configuration file has been created in the workspace, users can edit it to specify several properties needed to trigger the CI/CD pipeline for the selected blueprint. The parameters to set are listed below:

- *CSAR_name*: The name of the CSAR as uploaded in the Template Library;
- *CSAR_version*: The version of the CSAR as as uploaded in the Template Library;
- *Jenkins_URL*: The URL of the Jenkins server;
- *Jenkins_username*: The username of Jenkins credentials;
- *Jenkins_password*: The password of Jenkins credentials;
- *Jenkins_job*: The job (i.e., CI/CD pipeline) that must be triggered;
- *Jenkins_job_token*: The Authentication Token associated to the job;
- *cookie_jar*: Parameter used to get a Jenkins crumb. Use the value `/tmp/cookies`

Then, users can trigger the CI/CD pipeline in the Jenkins server specified in the configuration file by right-clicking on this configuration and selecting the respective menu entry.

Integration details:

Similarly to the previous tools, the CI/CD plugin is integrated into the RADON IDE using the *chePlugin* type in the RADON devfile.

4.2.10. Data Pipeline plugin

Overview of provided features:

The Data Pipeline orchestration plugin is a standalone component that checks for and automatically fixes potential problems in the user-defined data pipeline service templates before orchestration. As an example, if a user creates a pipeline, which migrates data between different cloud providers or availability zones, it is important to assure that the data encryption has been turned on to avoid leaking sensitive data during transport.

The input to the Data Pipeline plugin is a CSAR container of a service template exported from Graphical Modelling tool. The Data Pipeline plugin is deployed inside the RADON IDE as a Kubernetes pod containing a Docker container. The plugin itself provides a REST interface for integration, and implements a single API function: *convert CSAR*.

Integration details:

To integrate the plugin directly with the RADON IDE, a separate Data Pipeline (Che) plugin¹³ has been implemented, which allows users to right-click on CSAR files in the RADON IDE workspace to submit the file to be converted by the Data Pipeline plugin. Once the file is converted, it is written back to RADON IDE workspace and can be submitted to the RADON orchestrator for deployment.

¹³ <https://github.com/radon-h2020/datapipeline-che-plugin>

5. Testing

This section outlines the testing activities performed during the project in order to test the RADON framework. Section 5.1 provides an update of the testing activities of individual components to verify the correctness of the expected functionalities while Section 5.2 provides an update of the testing activities concerning the interaction between the components.

5.1. Functional Testing (Update)

In the deliverable [D2.4](#) we provided a first overview of the functional testing performed to validate the functional specifications (described in deliverables [D2.3](#) and [D2.4](#)) of the RADON tools developed within WP3-WP4-WP5. In this section, we provide an update of the performed functional testing.

We adopted a test-driven development approach in which several unit tests have been defined to validate the set of functionalities implemented by the RADON tools. Moreover, code coverage tools have been used to determine how much of the source code has been exercised during the execution of the test suites.

In particular, the test process has been characterised by the following activities:

- Define unit tests: in order to test the functionalities implemented by the individual RADON components, several unit tests have been defined using unit test frameworks according to the different programming languages;
- Automate testing: testing has been automated using CI/CD technologies. For instance, CI/CD pipelines have been defined and triggered on the Jenkins¹⁴ server hosted by ENG (see [D2.4](#) for details) or using other CI/CD systems (e.g., CircleCI¹⁵);
- Generate code coverage: code coverage reports have been generated using several code coverage tools according to the different programming languages. Then, the code coverage percentages have been indicated in the GitHub repositories using badges.

Table [2](#) briefly describes, for each RADON component, the adopted testing approach, also reporting the unit test frameworks and coverage tools that have been used. Table [3](#), instead, details the performed functional tests. Only for a few RADON tools it was not possible to define unit tests (and test coverage) due to their characteristics, but the alternative testing approach has been indicated in Table [2](#).

¹⁴ <https://www.jenkins.io/>

¹⁵ <https://circleci.com/>

Table 2. Tools for testing.

| Component | Unit Tests <i>[if applicable]</i> | Unit Test Framework | Coverage Tool |
|--|---|------------------------|------------------------|
| Constraint Definition Language and Verification Tool | Cannot be meaningfully applied. Each of the three VT modes transform a task into a set of constraints/rules which are then processed by an external solver. It is very difficult to break the code for the VT into units which can be tested independently in any meaningful way. We have instead produced a large set of functional tests. | N/A | N/A |
| Continuous Testing Tool | Tests have been generated based on the OpenAPI definition extended with pre- and post-conditions needed for the tests. In the Jenkinsfile the following stages have been added: (i) a stage to run unit tests and generate code coverage reports (ii) a stage to publish the reports to junit and coverage Jenkins plugins. | unittest ¹⁶ | coverage ¹⁷ |
| RADON IDE | <i>Can not be applied.</i> The RADON IDE is a particular tool based on the Eclipse Che technology which integrates the RADON tools developed on WP3-WP4-WP5. The RADON IDE specifications have been tested verifying that a RADON user is able to interact with the set of RADON tools integrated with it. | N/A | N/A |
| Graphical Modeling Tool | Unit tests have been created based on a test-driven development approach. The unit tests validate certain backend features of GMT, e.g., parsing and processing of TOSCA files. Further, many unit tests also validate the API layer based on API-driven tests. | JUnit ¹⁸ | Codacy ¹⁹ |
| Template Library | <i>Can not be fully applied.</i> Template library is a tool that mostly brings the storage and the API, where the most important part is the integration between these components and | N/A | N/A |

¹⁶ <https://docs.python.org/3/library/unittest.html>
¹⁷ <https://pypi.org/project/coverage/>
¹⁸ <https://junit.org/junit5/>
¹⁹ <https://www.codacy.com/>

| | | | |
|------------------------|--|-------------------------------|--|
| | the validation of TOSCA templates that are used to deploy the service. The unit tests are limited to testing the basic storage functions like upload and download of modules to S3 storage and managing modules' metadata. | | |
| Decomposition Tool | The latest version of the DT supports architecture decomposition and deployment optimization. Tests have been generated for these two features using models from the decomposition tool sample project. Since the DT is not fully open source like other RADON tools, testing is not executed based on a public CI/CD server or system. | matlab.unittest ²⁰ | matlab.unittest.plugins.CodeCoveragePlugin ²¹ |
| Defect Prediction Tool | Tests have been generated using a test-driven approach. First, we documented the Ansible and Tosca metrics with the intended behavior and examples. Then, we implemented the test cases before the production code to improve our confidence in the metric miners. The DPT uses the metrics miner. Therefore, at that point, we were sure the miner behaved as intended. Finally, we generated tests for each of the DPT commands (i.e., train, download model, and predict) to ensure that the tool provided the relevant and expected outputs. For example, check that a model is saved on the machine after "training" or "downloading" it. | unittest | pytest-cov |
| Orchestrator | The unit tests are used to test the xOpera TOSCA parser and other Python objects, modules and internal components such as template and instance creator and comparer, Ansible executor, threading and concurrent execution, standard TOSCA normative types from TOSCA Simple Profile in YAML v1.3 and so on. | pytest ²² | pytest-cov ²³ |

²⁰ <https://www.mathworks.com/help/matlab/ref/matlab.unittest-package.html>

²¹ <https://www.mathworks.com/help/matlab/ref/matlab.unittest.plugins.codecoverageplugin-class.html>

²² <https://pypi.org/project/pytest/>

²³ <https://pypi.org/project/pytest-cov/>

| | | | |
|----------------------|---|----------|----------|
| Monitoring Tool | <i>Cannot be applied.</i> The monitoring tool is mostly a TOSCA wrapper on the Prometheus monitoring and alerting components, that enables the RADON users to add monitoring capabilities on a FaaS and an OS level. | N/A | N/A |
| Function Hub Cli | The unit tests cover the full code collection with potential edge cases, exception handling and return codes. By testing the clients functionality, we also validate Function Hub API endpoints. | pytest | coverage |
| Data Pipeline plugin | The unit tests are used to verify that the REST API backend of the Data pipeline plugin converts CSAR files correctly. Separate unit tests are also applied on the non-API version of the plugin, to validate specific conversion Python methods are working as intended. | unittest | coverage |

Table 3. Functional testing of RADON tools (update).

| RADON tools | Functional Testing |
|--|---|
| Constraint Definition Language and Verification Tool | The functional tests for the VT consist of a set of 80 CDL specifications for verification, correction and learning tasks. The functional tests check that the VT correctly classifies the consistent/inconsistent CDL specifications in the verification test set and is able to return at least one inconsistency for the inconsistent cases. For the correction mode, the functional tests check that the VT is able to return at least one valid correction for each test case. For learning, the functional tests check that the VT learns a valid CDL specification from the examples in the specification. |
| Continuous Testing Tool | The focus of CTT's functional tests comprised the validation of the tool's (i) REST-based interaction workflow as well as the included abilities to (ii) read test-augmented RADON models, (iii) deploy and execute tests (including the CTT agents), and (iv) collecting and providing the results. We focused on component/system-level and integration-level testing (see Section 5.2), which is due to the decision to start with an early end-to-end prototype of the tool as well as the early and incremental integration with the other RADON tools to mitigate potential risks. The functional tests have been integrated into |

| | |
|---|---|
| | <p>CTT's CI/CD infrastructure and are executed on each commit.</p> |
| <p>RADON Integrated Development Environment</p> | <p>The functionalities of the RADON IDE have been tested defining an activity diagram (see Section 4.1) and verifying that a RADON user is able to perform the set of expected actions defined on it. A registered user connected to the RADON IDE instance and we successfully tested the following actions:</p> <ul style="list-style-type: none"> ● User authentication on the IDE; ● Possibility to create a RADON workspace providing to the RADON user access to the shared repositories of RADON artifacts and enabling interaction with the RADON tools that are invoked via custom commands; ● Modeling of an application by means of the GMT and store/publish/share it via RADON Particles on Github or the Template Library Publishing Service; ● Edit CDL specifications for a RADON model and verify them by means of the VT; ● Decompose and Optimize a RADON model by means of the DT; ● Verify defects of an (ansible) IaC script by means of the DPT; ● Define, execute and maintain Test Cases by means of the CTT; ● Export into the workspace the modeled application in the CSAR format for deployment; ● Start deployment process by means of the RADON Orchestrator (along with the deployment of monitoring stack); ● Configure and trigger from the RADON IDE a predefined CI/CD pipeline in a Jenkins server; ● Visualization of the deployment status and monitoring data by opening a browser window. |
| <p>Graphical Modeling Tool</p> | <p>Eclipse Winery, on which the RADON GMT is based, already had a good starting point in terms of unit testing as it has been under active development since 2013. At the time of writing, Winery employs and executes 526 unit tests during each CI build. With up to 350 unit tests, Eclipse Winery focuses on testing the RESTful API layer as well as the backend functionality to ensure the proper serialization of maintained TOSCA entities. In the course of the project we steadily extended the number of unit tests to 587.</p> <p>Further, while developing new functionality to create and export TOSCA YAML application blueprints, we regularly performed manual end-to-end testing between the RADON GMT and the RADON Orchestrator to ensure that modeled application deployments can be successfully executed.</p> |
| <p>Template Library</p> | <p>The functionalities of Template Library were tested through Template Publishing Service (TPS) where we provided and uploaded several TOSCA modules (these included TOSCA templates and their Ansible</p> |

| | |
|-------------------------------|---|
| | <p>playbook implementations) to the public instance²⁴ using REST API, CLI and RADON IDE plugin. We included cloud modules for AWS, Azure, GCP and OpenFaaS along with the ThumbnailGenerator application blueprint (CSAR) for every aforementioned cloud provider and some example blueprints that connect multiple cloud providers (for instance Azure and AWS). We also published some other common modules such as MinIO, Docker and Rancher for Kubernetes. All these modules can be now accessed and downloaded. Template library's content gets deployed using xOpera orchestrator within CI/CD configurations. Its source code that currently resides in a private GitLab repository is constantly being tested by GitLab CI scheduled pipelines that track linting errors and therefore maintain code quality, validate TPS TOSCA templates and backup the production version of TPS data nightly so that the content can be restored in case of any future accidents.</p> |
| <p>Decomposition Tool</p> | <p>The DT is unit tested using models from the decomposition tool sample project to ensure the robustness of features released in the latest version, namely architecture decomposition and deployment optimization. Additionally, the backend solver of the DT integrates a collection of about 300 unit tests that verify the consistency of the performance predictions on models of varying size, with different types of invocations between software components, scheduling at the resources, and workload intensities.</p> |
| <p>Defect Prediction Tool</p> | <p>The DPT relies on external tools created in the context of RADON (radon-ansible-metrics and radon-iac-miner) that already provide a good suite of unit tests to test their intended behavior to collect and extract data from Ansible blueprints. Therefore, the focus of DPT functional tests comprised the proper working of the REST-based interaction workflow of the tools starting from the IDE and the included abilities to collect and provide the results.</p> |
| <p>Orchestrator</p> | <p>xOpera TOSCA orchestration has already been tested with different application blueprints like ThumbnailGenerator and SockShop. It was also used to deploy TPS on a public VM. Apart from all that, opera's source code on GitHub²⁵ is continuously tested by CircleCI. With every push on any branch we perform unit and integration tests that assure stability of the new code, i.e., new changes do not break any functionality. Every night the integration tests are performed on the latest xOpera version where we execute tests on prepared TOSCA CSARs that contain common TOSCA entities. With CI/CD we also</p> |

²⁴ <https://template-library-radon.xlab.si/swagger/>

²⁵ <https://github.com/xlab-si/xopera-opera>

| | |
|----------------------|--|
| | <p>ensured that with every push on the master branch a testing opera Python package is deployed on testing PyPI environment²⁶. In case when a new tag is pushed to the application release, the new official release is published on production PyPI server²⁷.</p> |
| Function Hub Cli | <p>Unit tests are written as the code base increases. The test suite is added in the test stage of the Jenkinsfile, the declarative CI pipeline. The code coverage percentage is visible on the project's github page. The intended functionality described in the requirements is fully test covered. See previous deliverable D2.4 and closed issues in gitHub projects for full list of requirements and features. The architectural decision of the app and client with the API driven functionality means that the functional testing can be scripted²⁸ and automated.</p> |
| Delivery Toolchain | <p>The Delivery Toolchain targets TOSCA Orchestrator, CI/CD and RADON Monitoring tool. The first two tools are being tested separately. The Monitoring tool has been tested with a reference service model, GCP monitoring²⁹. It consists of shared instances of a Prometheus server, a Consul service discovery cluster and a visualization dashboard, while it offers the Prometheus PushGateway and NodeExporter components. These services are set up using prepared TOSCA template and Ansible playbooks which have been regularly tested with xOpera.</p> |
| Data Pipeline plugin | <p>The Data Pipeline plugin takes a CSAR generated by the GMT as input. It checks the CSAR file for any inconsistencies related to data pipeline node types and modifies the CSAR file if anything needs to be fixed. The Data Pipeline plugin is deployed as a Docker container and provides an API method for submitting and converting CSAR files. API level unit-testing is used to verify that a prepared CSAR file is modified correctly by the API. The achieved coverage was 67% for the API, and the coverage.report is available on the tool GitHub³⁰ page. Separate unit-tests verify the internal conversion function on specific issues in the CSAR.</p> |

²⁶ <https://test.pypi.org/project/opera/>

²⁷ <https://pypi.org/project/opera/>

²⁸ <https://github.com/radon-h2020/radon-functionhub-client/tree/master/test>

²⁹

https://github.com/radon-h2020/radon-particles/tree/master/servicetemplates/radon.blueprints.monitoring/GCP_Monitoring_Example

³⁰ <https://github.com/radon-h2020/radon-datapipeline-plugin/tree/master/datapipeline-server>

Table 4 reports for each RADON tool, where it is applicable (see Table 2), the percentage of the code coverage resulting from the testing activities.

Table 4. Code coverage of RADON tools.

| RADON tools | Code coverage |
|-------------------------|----------------------|
| Continuous Testing Tool | 77% |
| Graphical Modeling Tool | 49% |
| Decomposition Tool | 67% |
| Defect Prediction Tool | 82% ³¹ |
| Orchestrator | 79% |
| Function Hub | 92% |
| Data Pipeline plugin | 67% |

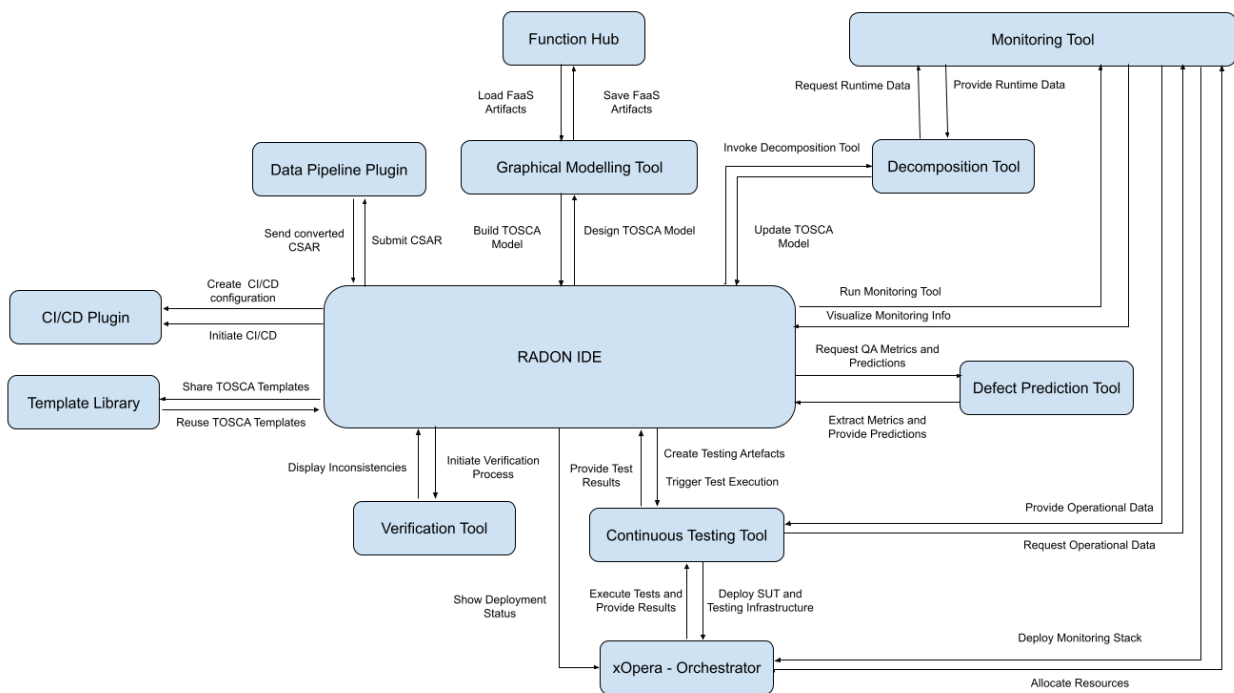
³¹ This value is the average of the code coverage percentages of the DPT's source code, available in the GitHub repositories.

5.2. Integration Testing (Update)

In the deliverable [D2.4](#) (M18) we provided a first description of the integration tests performed to verify the interactions between the RADON tools released in the alpha release of RADON integrated framework. As described in the previous deliverable, we adopted an incremental testing approach and we performed the integration tests considering pairs of components identified by examining their interactions. In this section we provide an update of the performed integration tests taking into account the interactions between the components involved in the final release of the RADON integrated framework.

The overall interactions between the RADON tools are depicted in [Figure 8](#). In particular, these interactions have been defined as a result of the six different workflows entailed in the process of designing, developing, and operating a RADON application, as described in the deliverable [D3.1](#). The next subsections describe the final integration tests performed to test the main interactions.

Figure 8. Final interactions between RADON tools.



5.2.1. Continuous Testing Tool - Graphical Modeling Tool/xOpera Orchestrator

| <i>Integration Testing Scenario CTT-GMT-Orchestrator</i> | |
|--|---|
| Description | <p>The integration test between CTT and GMT tests the ability of GMT to export a service template as a CSAR and the ability of CTT to take this as an input for further processing, i.e., deploying the service template using the xOpera orchestrator.</p> <p>The following test execution steps explain the essential steps needed to test the integration between CTT, the artifacts exported from GMT, and the xOpera orchestrator.</p> |
| Pre-Conditions | In order to run this test, there needs to be a TOSCA service template that can be exported. |
| Post-Conditions | The SUT and TI are deployed and the test results have been downloaded from CTT. |
| Test Execution Steps | <p>The following steps illustrate the test execution:</p> <ol style="list-style-type: none"> 1. Clone integration test repository. 2. Pull all containers needed for the GMT application. 3. Clone RADON Particles repository. 4. Start the GMT application. GMT application is running and web interface is available at http://localhost:18080/ 5. Start the CTT server. CTT application is running and web interface is available at http://localhost:7999/RadonCTT/ui/ 6. Clone the SockShop Demo repository. SockShop Demo repository available locally. 7. Use the GMT REST API to export the system under test (SUT) CSAR to the local file system. CSAR file containing the TOSCA blueprints for the SUT and its dependencies in the local file system. 8. Use the GMT REST API to export the test infrastructure (TI) CSAR for the JMeter Master Only Agent to the local file system. 9. Install the CTT CLI Tool 10. Execute the CTT Testing Workflow using the CTT CLI Tool <ol style="list-style-type: none"> a. Create Project b. Create TestArtifact c. Create Deployment (SUT and TI) d. Create Execution (Execute Test) e. Create Results f. Obtain/Download Test Results |
| Test Results | A test report has been downloaded from CTT. |

| Integration Testing Scenario CTT-Orchestrator (Data Pipelines case) | |
|--|---|
| Description | This is an additional integration test between CTT and the xOpera orchestrator which tests the ability of CTT to properly interact with the orchestrator for deploying a TOSCA data pipeline service template. |
| Pre-Conditions | In order to run this test, the data pipeline application needs to be available in the GitHub repository. Also there needs to be a TOSCA software under test (SUT) service template and testing infrastructure (TI) service template available in the repository. |
| Post-Conditions | The SUT and TI are successfully deployed and a test report could be downloaded from CTT. |
| Test Execution Steps | <p>The test execution steps of this integration test can be reused from <i>CTT-GMT and CTT-Orchestrator test scenarios</i>. To avoid repeating their description, we only provide their sequence here:</p> <ol style="list-style-type: none"> 1. Start the CTT server 2. Create a CTT project using the CTT REST API 3. Create a CTT test artifact using the CTT REST API 4. Create a CTT execution using the CTT REST API 5. Create a CTT result using the CTT REST API 6. Download the CTT result using the CTT REST API <p>The main difference from the previous scenarios is that the software under test (SUT) is a data pipeline application (e.g., Thumbnail generation with data pipelines³² described in the D6.2 Initial Validation Results deliverable [7])</p> |
| Test Results | If successful, the result is a ZIP-file containing the test results of the test that has been executed by the TI. |

5.2.2. Continuous Testing Tool - RADON IDE

| Integration Testing Scenario CTT- RADON IDE | |
|--|---|
| Description | The integration test between the Continuous Testing Tool and the RADON IDE tests the ability of the user to interact effectively with the tool through the RADON IDE. |
| Pre-Conditions | The RADON IDE has started. A RADON SUT model with test annotations and a corresponding TI model is available. |

³² <https://github.com/radon-h2020/demo-lambda-thumbgen-tosca-datapipeline>

| | |
|-----------------------------|--|
| Post-Conditions | The test results are located in the configured file system directory. |
| Test Execution Steps | <ol style="list-style-type: none"> 7. Create a CTT configuration file. 8. Right-click on the configuration file and select the option to run CTT. 9. The CTT workflow is executed and can be inspected in the log messages and the progress bar. 10. Once the CTT workflow has finished, the test results ZIP file is placed in the RADON IDE workspace. |
| Test Results | The CTT IDE plugin properly communicates with the CTT Server to execute the workflow of CTT. |

5.2.3. Orchestrator - RADON IDE

| <i>Integration Testing Scenario Orchestrator- RADON IDE</i> | |
|---|--|
| Description | The integration test between the xOpera SaaS and the RADON IDE tests the ability of the user to interact effectively with the tool through the RADON IDE. |
| Pre-Conditions | The RADON IDE has started. |
| Post-Conditions | The xOpera SaaS console is available to show the deployment status. |
| Test Execution Steps | <ol style="list-style-type: none"> 1. Press <i>Ctrl+Shift+P</i> to open the command palette. 2. Select the <i>RADON: Show Deployment status</i> command. 3. Allow to open the external web site https://saas-xopera.xlab.si/ui/. 4. Click on the appropriate option on the login screen. |
| Test Results | The RADON menu plugin properly opens a browser page connecting to the main xOpera SaaS console. |

5.2.4. Orchestrator - Monitoring

| <i>Integration Testing Scenario Orchestrator- Monitoring</i> | |
|--|---|
| Description | The integration test between the xOpera SaaS and the Monitoring tool tests the ability of the user to deploy a model and monitor the runtime resource consumption. The reference example model provided for testing refers to a GCP monitored FaaS. |

| | |
|-----------------------------|--|
| Pre-Conditions | Configure a reference example model ³³ and prepare it for deployment. |
| Post-Conditions | The runtime metrics are available in the visualization dashboard. |
| Test Execution Steps | <ol style="list-style-type: none"> 1. Deploy the reference example model using the xOpera SaaS 2. Access the Grafana visualization dashboard and check that the metrics flow at runtime. |
| Test Results | The runtime metrics are monitored as a result of a successful deployment. |

5.2.5. Decomposition Tool - RADON IDE

| <i>Integration Testing Scenario DEC- RADON IDE 1 (Architecture Decomposition)</i> | |
|---|--|
| Description | This integration test verifies the ability of the Decomposition Tool (DT) to properly interact with the RADON IDE for architecture decomposition. |
| Pre-Conditions | An abstract RADON model with interface annotations is available on the codebase, but a monolithic/microservice application needs to be decomposed in order to migrate it to a container/FaaS platform. |
| Post-Conditions | The execution of the DT decomposition procedure is correctly displayed in the <i>Output</i> window of the RADON IDE (View → Output). The RADON model is updated with the desired decomposition solution for the monolithic/microservice application. |
| Test Execution Steps | <ol style="list-style-type: none"> 1. Configure the domain name and public port of the DT server instance (Open Preferences → Radon Dec Plugin → Server) 2. Clone the decomposition tool sample project from https://github.com/radon-h2020/demo-decomposition-tool-sample-project.git. The <i>mono-app</i> and <i>micro-app</i> folders provide sample service templates, <i>model.tosca</i>, for an abstract monolithic and an abstract microservice application respectively. These service templates contain the required interface annotations. 3. Right-click on either of the service templates in the file explorer. 4. Click "Decompose". |
| Test Results | The decomposition result returned by the DT server instance is deterministic and can be used for regression testing. |

| <i>Integration Testing Scenario DEC- RADON IDE 2 (Deployment Optimization)</i> | |
|---|---|
| Description | This integration test verifies the ability of the Decomposition Tool (DT) to properly interact with the RADON IDE for deployment optimization. |
| Pre-Conditions | A concrete RADON model with performance annotations is available on the codebase, but the memory and concurrency level of a serverless function need to be optimized in order to minimize the total operating cost. |
| Post-Conditions | The execution of the DT optimization procedure is correctly displayed in the <i>Output</i> window of the RADON IDE (View → Output), showing particularly the minimum total operating cost. The RADON model is updated with the optimal memory and concurrency level for the serverless function. |
| Test Execution Steps | <ol style="list-style-type: none"> 1. Configure the domain name and public port of the DT server instance (Open Preferences → Radon Dec Plugin → Server) 2. Clone the decomposition tool sample project from https://github.com/radon-h2020/demo-decomposition-tool-sample-project.git. The <i>demo-app</i> folder provides two sample service templates, <i>open_model.tosca</i> and <i>closed_model.tosca</i>, for a concrete demo application (thumbnail generation). These service templates contain the required performance annotations but define different types of workloads. 3. Right-click on either of the service templates in the file explorer. 4. Click "Optimize". |
| Test Results | The optimization result returned by the DT server instance is deterministic and can be used for regression testing. |

5.2.6. Defect Prediction Tool - RADON IDE

| <i>Integration Testing Scenario DPT-RADON IDE</i> | |
|--|--|
| Description | The integration test between the Defect-Prediction Tool (DPT) and the RADON IDE tests the ability of the DPT to properly interact with the RADON IDE to run detection on a given Ansible blueprint. The following test execution steps explain the essential steps needed to test the integration between DPT and the RADON IDE. |
| Pre-Conditions | Running CTT with at least one test artifact (as it would result at the end of Integration Testing Scenario 1). |
| Post-Conditions | The Ansible blueprint is successfully analyzed and a report is shown on |

| | |
|-----------------------------|---|
| | the Graphic User Interface. |
| Test Execution Steps | <ol style="list-style-type: none"> 1. Create an Ansible, a Tosca blueprint, or a CSAR archive containing Tosca blueprints. The expected output is a well-formatted yaml file. 2. Call the APIs to run the detection by right-clicking on the yaml file. The inputs are the Ansible/Tosca source code metrics extracted by the plugin locally. The expected output consists of a json report with information about the final decision (i.e., the defect type: conditional, service, configuration data), an interpretation of the prediction (i.e., why the tool predicted that class of defect, and a value for each of the extracted metrics). 3. Display the results in a Webview. This step shows the results of the detection on a Graphical User Interface in the RADON IDE. The input is the json report from step 2. The expected output is a two columns table of (key, value) pairs where the <i>key</i> represents a metric extracted from the script. The final detection decision with its interpretation is prepended to the metrics list. |
| Test Results | <p>If successful, the result is a table containing information about the characteristics of the script (i.e., the metrics extracted) along with the final decision (defective or not defective).</p> <p>If unsuccessful, the result is a pop-up message indicating the cause of the failure: Ansible file not valid or server error.</p> |

5.2.7. GMT - RADON IDE

| <i>Integration Testing Scenario GMT - RADON IDE</i> | |
|---|--|
| Description | Being able to start GMT from the RADON IDE, open a given TOSCA Service Template (aka. RADON Model) and export it as a CSAR to be available in RADON IDE's project folder. |
| Pre-Conditions | User is logged-in into RADON IDE and created a RADON workspace. |
| Post-Conditions | A CSAR file is available in the RADON IDE's project folder "radon-csars". |
| Test Execution Steps | <ol style="list-style-type: none"> 1. Open GMT: Go to <i>My Workspace</i> on the right hand side of the RADON IDE, click on the <i>radon-gmt</i> entry; a new browser opens 2. In the list of Service Templates, click on "EC2_on_AWS" 3. In the header component, click on "Export" and select "Export to Filesystem" 4. GMT shows a successful message in the top-right corner of the browser window |

| | |
|---------------------|---|
| | <ol style="list-style-type: none"> 5. Go back to RADON IDE 6. In the project explorer, a new directory named “radon-csars” is available 7. A new file named “EC2_on_AWS.csar” is available, which a user can download or process further with other RADON tools. |
| Test Results | <p>If successful, GMT shows a successful message after exporting the CSAR to the RADON IDE. Further, the respective CSAR is accessible in the RADON IDE.</p> <p>If unsuccessful, GMT shows an error during export and no CSAR will be present.</p> |

5.2.8. Function Hub - GMT

| <i>Integration Testing Scenario Function Hub - GMT</i> | |
|--|--|
| Description | Being able to select URL from Function Hub when creating a FaaS model. Treat the function as a ‘black box’ and insert the necessary parameters like <i>runtime</i> , <i>handler name</i> , and <i>environment variables</i> . |
| Prerequisite | <ul style="list-style-type: none"> ● Running GMT instance ● Preselected, valid URL from Function Hub |
| Postrequisite | GMT allows script input both as ‘on local machine’ and ‘URL’. |
| Test Execution Steps | <ol style="list-style-type: none"> 1. Select Function from Function Hub and copy URL 2. Use path as input for FaaS model in GMT 3. Fill in the necessary fields describing the function 4. Export CSAR |
| Test Results | Successfully generated CSAR with FaaS source code and dependencies included. |

5.2.9. Verification Tool - RADON IDE

| <i>Integration Testing Scenario VT-RADON IDE</i> | |
|--|--|
| Description | The integration test between the Verification Tool (VT) and the RADON IDE tests the ability of the VT to properly interact with the RADON IDE. Specifically, it checks that a user can use the RADON IDE to call the verification mode of the VT and view the result in the RADON IDE. |

| | |
|-----------------------------|--|
| Pre-Conditions | None |
| Post-Conditions | The output of the VT is correctly displayed with the RADON IDE |
| Test Execution Steps | <ol style="list-style-type: none"> 1. Clone the verification tool sample project from https://github.com/radon-h2020/demo-verification-tool-sample-project.git. This contains a sample TOSCA model and a CDL specification in "main.cdl". 2. Right-click on the main.cdl file in the file explorer. 3. Click "Verify". |
| Test Results | A sub-window should appear in the bottom of the RADON IDE. After a few seconds the output of the VT should be displayed. If everything has worked correctly, the VT will show that it has found an inconsistency in the specification. |

5.2.10. Verification Tool - Graphical Modeling Tool

| <i>Integration Testing Scenario VT- GMT</i> | |
|---|---|
| Description | The integration test between the Verification Tool (VT) and the Graphical Modeling Tool (GMT) tests the ability of the VT to properly interact with the GMT. Specifically, it checks that the output of the VT's correction mode can be displayed in the GMT. Note that the interaction between the two tools is indirect (both tools interact with the RADON IDE). |
| Pre-Conditions | None |
| Post-Conditions | The diff between the original model and the corrected model (found by the VT) is correctly displayed with the GMT |
| Test Execution Steps | <ol style="list-style-type: none"> 1. Clone the verification tool sample project from https://github.com/radon-h2020/demo-verification-tool-sample-correction-project.git. This contains a sample CSAR and a CDL specification in "main.cdl". 2. Right-click on the main.cdl file in the file explorer. 3. Click "Correct". <ul style="list-style-type: none"> ○ This triggers the VT to build a new CSAR object containing the corrected model. 4. Right-click on the original CSAR in the file explorer. 5. Click "Upload to GMT" <ul style="list-style-type: none"> ○ This is then imported in the GMT. 2. Right-click on the new CSAR in the file explorer. 3. Click "Upload to GMT" <ul style="list-style-type: none"> ○ This is then also imported in the GMT. |

| | |
|---------------------|---|
| | <ol style="list-style-type: none"> 4. Open the GMT. 5. Navigate to "Service Templates" and find "VTSample". 6. Expand the "VTSample" nodes and click "(+) Differences". <ul style="list-style-type: none"> ○ This will display a diff between the two models, showing that two AWS Buckets have been moved to different AWS Platforms. |
| Test Results | The GMT should display the diff between the original and corrected versions of the RADON model. |

5.2.11. Monitoring - RADON IDE

| <i>Integration Testing Scenario Monitoring - RADON IDE</i> | |
|--|--|
| Description | The integration test between the Monitoring tool and the RADON IDE tests the ability of the user to interact effectively with the tool through the RADON IDE. |
| Pre-Conditions | The RADON IDE has started. |
| Post-Conditions | The Monitoring dashboard is available to show the monitoring information. |
| Test Execution Steps | <ol style="list-style-type: none"> 1. Press <i>Ctrl+Shift+P</i> to open the command palette. 2. Select the <i>RADON:Open Monitoring page</i> command. 3. Allow to open the external web site http://3.127.254.144:3000/ 4. Click on the appropriate option on the login screen. |
| Test Results | The RADON menu plugin properly opens a browser page connecting to the main Monitoring dashboard. |

5.2.12. Template Library - RADON IDE

| <i>Integration Testing Scenario Template Library - RADON IDE</i> | |
|--|---|
| Description | The integration test between the RADON IDE and the Template library (Particles and TPS) tests the user' ability of interaction with the TPS through the RADON IDE, which means that the user is able to effectively upload and download modules to TPS. |
| Pre-Conditions | The RADON IDE has started. |
| Post-Conditions | <ol style="list-style-type: none"> 1. The templates and modules are accessible inside IDE. 2. The modules/templates are published in TPS |

| | |
|-----------------------------|---|
| Test Execution Steps | Particles case: <i>RADON IDE should automatically pull the content from the particle GitHub the last version of RADON particles</i> TPS case: <ol style="list-style-type: none"> 1. Navigate to the folder and file section in the IDE. 2. Right click on the editor or file explorer 3. Click on “Template library interactive actions”. 4. Click “DownloadTemplateInteractiveAction”. 5. Select the module to download from dropdown. 6. Type in the path where you want to download the module to. |
| Test Results | The plugin downloads the module file (usually a zip archive) to a specified location and the module is then visible in the file explorer. |

5.2.13. CI/CD Plugin- RADON IDE

| <i>Integration Testing Scenario CI/CD plugin - RADON IDE</i> | |
|--|--|
| Description | The integration test between the CI/CD plugin and the RADON IDE tests the ability of the CI/CD plugin to invoke a CI/CD pipeline on a remote platform. |
| Pre-Conditions | <ul style="list-style-type: none"> ● The RADON IDE has started. ● A configured Jenkins server. ● A user with execution access to jobs. ● A configured CI/CD pipeline. ● A CSAR file is available on the RADON IDE workspace. The CSAR has been uploaded to the Template Library. |
| Post-Conditions | The preconfigured CI/CD pipeline has been triggered and executed within the Jenkins server. |
| Test Execution Steps | <ol style="list-style-type: none"> 1. Right-click on the CSAR file available in the file explorer. 2. Click “Configure CI”. A configuration file (i.e. a yaml file) with some configuration parameters has been created. 3. Edit the created configuration yaml file to specify the name and version of the CSAR uploaded to the Template Library, the Jenkins instance and the user credential, the CI/CD pipeline to trigger and its token. 4. Right-click on the configuration file edited in the previous point. 5. Click “Trigger CI”. |
| Test Results | The CI/CD plugin properly triggers the CI/CD pipeline on the Jenkins server according to the parameters specified on the edited configuration |

| | |
|--|------------|
| | yaml file. |
|--|------------|

5.2.14. Data Pipeline Plugin

| <i>Integration Testing Scenario Data Pipeline plugin-RADON IDE</i> | |
|--|---|
| Description | The integration test between the Data Pipeline plugin and the RADON IDE tests the ability of the Data Pipeline plugin to properly interact with the RADON IDE. Specifically, it checks that a user can use the RADON IDE to call the Convert CSAR operation of the Data Pipeline plugin the converted CSAR file is accessible inside the RADON IDE. |
| Pre-Conditions | CSAR file available in the RADON IDE “radon-csars” workspace folder. |
| Post-Conditions | Converted CSAR is written into the RADON IDE “radon-csars” workspace folder. Converted CSAR is deployable by RADON orchestrator. Original CSAR still exists. |
| Test Execution Steps | <ol style="list-style-type: none"> 1. Right click on the CSAR file 2. Select Convert CSAR with Data pipeline plugin |
| Test Results | A terminal window should appear in the bottom of the RADON IDE. After a few seconds the output should be displayed. If everything has worked correctly, a converted CSAR file will be created in the RADON IDE “radon-csars” workspace folder. Source CSAR file should not be overwritten. |

6. Conclusions

This deliverable presents the final release of the RADON integrated framework by describing the technical decisions in detail to implement the RADON IDE based on the Eclipse Che technology.

This document concentrates on how the IDE has been customized in order to (i) integrate the RADON tools on it according to the RADON IDE’s requirements ([D2.2](#)) and the RADON workflows ([D3.1](#)), (ii) enable the interaction of these tools with the shared spaces of the RADON artifacts, and (iii) add new graphical elements (e.g. menus, commands etc.) to interact with the overall RADON framework and visualize results (e.g., show deployment status and monitoring data).

Table [5](#) summarizes the level of RADON IDE’s compliance with the requirements for this final release. In particular, we focus on the requirements concerning the customization of the development environment for RADON’s purposes (e.g. integration on the IDE of the RADON tools developed in the technical work packages). The remaining requirements are achieved by means of Eclipse Che. In parentheses in gray, we list the level of compliance as reported in the previous deliverable [D2.6](#). Moreover, in each case, we provide a brief statement explaining the (changed) level of compliance.

The labels specifying the “Level of compliance” are defined as follows:

- (i) ✓ (partially-low supported): a few of the aspects of the requirement is fulfilled by the current version;
- (ii) ✓✓ (partially-high supported): most of the aspects of the requirement is fulfilled by the current version;
- (iii) ✓✓✓ (fully supported): the requirement is fulfilled by the current version.

Table 5. Overview of RADON IDE’s requirement compliance level

| ID | Requirement Title | Priority | Level of fulfillment |
|---|-------------------------------|-----------|----------------------|
| R-T2.3-6 | Secure application workspaces | Must have | ✓✓ (✓) |
| The requirement is now partially-high supported by the extensions regarding the adoption of Keycloak to secure the interaction with some RADON tools. | | | |
| R-T2.3-7 | Integration of TESTING_TOOL | Must have | ✓✓✓ (-) |
| The requirement is now fully supported by the extensions regarding the integration of the CTT. | | | |
| R-T2.3-8 | Access to test reports | Must have | ✓✓✓ (-) |

| | | | |
|---|--|-----------|-------------|
| The requirement is now fully supported by the extensions regarding availability of the test results in the RADON workspace. | | | |
| R-T2.3-9 | Access to the shared repositories | Must have | ✓ ✓ ✓ |
| The requirement was fully supported with the previous deliverable. | | | |
| R-T2.3-10 | Provide connectors to CI/CD tools | Must have | ✓ ✓ ✓ (✓) |
| The requirement is now fully supported by the extensions regarding the integration of CI/CD plugin. | | | |
| R-T2.3-11 | Support for the deployment process | Must have | ✓ ✓ ✓ (✓ ✓) |
| The requirement is now fully supported by the extensions regarding the integration of xOpera SaaS Orchestrator. | | | |
| R-T2.3-12 | Creation of a RADON-based workspace | Must have | ✓ ✓ ✓ (✓ ✓) |
| The requirement is now fully supported by the extensions regarding the integration of xOpera SaaS Orchestrator. | | | |
| R-T2.3-13 | Creation of a RADON modeling project | Must have | ✓ ✓ ✓ |
| The requirement was fully supported with the previous deliverable. | | | |
| R-T2.3-14 | Integration of GMT | Must have | ✓ ✓ ✓ |
| The requirement was fully supported with the previous deliverable. | | | |
| R-T2.3-15 | Access projects inside a workspace from the GMT | Must have | ✓ ✓ ✓ (-) |
| The requirement is now fully supported by the extensions regarding the implementation of the “jump to code” feature. | | | |
| R-T2.3-16 | Synchronization of changes made within the IDE in the GMT and vice versa | Must have | ✓ ✓ ✓ (-) |
| The requirement is now fully supported by the extensions regarding the container-based integration of GMT into the RADON IDE. | | | |

| | | | |
|---|---|-----------|-------------|
| R-T2.3-18 | Support in launching the TOSCA management UI | Must have | ✓ ✓ ✓ |
| The requirement was fully supported with the previous deliverable. | | | |
| R-T2.3-19 | Export the application blueprints as TOSCA CSAR | Must have | ✓ ✓ ✓ |
| The requirement was fully supported with the previous deliverable. | | | |
| R-T2.3-20 | Source code editor support for TOSCA grammar | Must have | ✓ ✓ (-) |
| The requirement is now partially-high supported by the extensions regarding the integration of the radon-tosca-yaml plugin | | | |
| R-T2.3-21 | Integration of Defect Prediction Tool | Must have | ✓ ✓ ✓ |
| The requirement was fully supported with the previous deliverable. | | | |
| R-T2.3-22 | Support in showing monitoring information | Must have | ✓ ✓ ✓ (-) |
| The requirement is now fully supported by the extensions regarding the possibility to connect with the Monitoring dashboard within the RADON IDE. | | | |
| R-T2.3-23 | Availability of a RADON menu | Must have | ✓ ✓ ✓ |
| The requirement was fully supported with the previous deliverable. | | | |
| R-T2.3-24 | RADON tool UI elements | Must have | ✓ ✓ ✓ (✓) |
| The requirement is now fully supported by the extensions presented in this deliverable (Section 4.2). | | | |
| R-T2.3-25 | Availability of custom menus and commands | Must have | ✓ ✓ ✓ (✓ ✓) |
| The requirement is now fully supported by the extensions presented in this deliverable (Section 4.2). | | | |
| R-T2.3-26 | Integration of the Verification Tool | Must have | ✓ ✓ ✓ |

| | | | |
|--|-----------------------------------|-----------|-------|
| The requirement was fully supported with the previous deliverable. | | | |
| R-T2.3-27 | Integration of Decomposition Tool | Must have | ✓ ✓ ✓ |
| The requirement was fully supported with the previous deliverable. | | | |

7. References

- [D3.1] RADON Consortium, Deliverable D3.1: RADON DevOps methodology, 2021
- [D2.4] RADON Consortium, Deliverable D2.4: Architecture and integration plan II, 2020
- [D2.3] RADON Consortium, Deliverable D2.3: Architecture and integration plan I, 2019
- [D2.6] RADON Consortium, Deliverable D2.6: RADON integrated framework I, 2020
- [D2.2] RADON Consortium, Deliverable D2.2: Final requirements, 2020