

RADON: Open Source DevOps for Serverless Applications

Embracing microservices and serverless technology with the open source RADON framework





RADON: OPEN SOURCE DEVOPS FOR SERVERLESS APPLICATIONS

@Copyright 2021 RADON Consortium

E-mail: info@radon-h2020.eu

Edition: 1.0.0 (30 June 2021)

Editor

Giuliano Casale, Imperial College London, UK

Author List

Matija Cankar, XLAB, SI

Chinmaya Dehury, University of Tartu, EE

Stefania D'Agostini, Engineering Ingegneria Informatica, IT

Stefano Dalla Palma, JADS, NL

Dario Di Nucci, JADS, NL

Georgios Giotis, ATC, GR

André van Hoorn, University of Stuttgart, DE

Pelle Jakovits, University of Tartu, EE

Mark Law, Imperial College London, UK

Anže Luzar, XLAB, SI

Zifeng Niu, Imperial College London, UK

Domenico Presenza, Engineering Ingegneria Informatica, IT

Anestis Siridopoulos, ATC, GR

Alexandros Spartalis, Eficode, NO

Sašo Stanovnik, XLAB, SI

Damian Tamburri, JADS, NL

Giorgos Triantafyllou, ATC, GR

Michael Wurster, University of Stuttgart, DE

Lulai Zhu, Imperial College London, UK

Table of contents

| | |
|--|----|
| 1. Introduction | 6 |
| 1.1 Benefits of serverless computing | 6 |
| 1.2 Challenges posed by serverless computing | 8 |
| 1.3 Why RADON? | 9 |
| 1.4 An overview of the RADON framework | 10 |
| 1.5 Structure of the book | 14 |
| 2. RADON Workflow-driven Methodology | 15 |
| 2.1 The RADON lifecycle model | 15 |
| 2.2 RADON Workflows | 18 |
| 3. Tools Overview | 21 |
| 3.1 RADON Integrated Development Environment | 21 |
| 3.2 Graphical Modeling Tool | 22 |
| 3.3 Verification Tool | 25 |
| 3.4 Decomposition tool | 27 |
| 3.5 Defect Prediction Tool | 31 |
| 3.6 Continuous Testing Tool | 34 |
| 3.7 xOpera SaaS Orchestrator | 36 |
| 3.8 Template Library | 40 |
| 3.9 Monitoring Tool | 42 |
| 3.10 Function Hub | 45 |
| 3.11 CI/CD Plugin | 48 |
| 3.12 Data Pipeline Plugin | 51 |
| 4. Industrial Use Cases | 54 |
| 4.1 Travel Technology | 54 |
| 4.2 Assisted Living | 57 |
| 4.3 Artifact Management | 60 |
| 5. Conclusion | 63 |
| References | 63 |

Glossary

| | |
|-------|--|
| CDL | Constraint Definition Language |
| CI/CD | Continuous Integration/Continuous Delivery |
| CTT | Continuous Testing Tool |
| DPT | Defect Prediction Tool |
| DT | Decomposition Tool |
| FaaS | Function as a Service |
| GMT | Graphical Modeling Tool |
| IaC | Infrastructure as Code |
| VT | Verification Tool |
| WP | Work Package |

1. Introduction

Emerging serverless computing technologies, such as function-as-a-service (FaaS) offerings, have emerged in recent years to enable developers to virtualize the internal logic of an application, simplifying management of cloud-native applications and allowing cost savings through billing and scaling at the level of individual function calls. Therefore, serverless computing is rapidly shifting the attention of software vendors to developing complex industrial cloud applications that can use this new technology and the underpinning platforms optimally.

This handbook addresses this problem by presenting RADON, a DevOps framework to create and manage microservices-based applications that can optimally exploit serverless computing technologies. Applications built with RADON include fine-grained and independently deployable microservices that can efficiently exploit FaaS and container technologies. The vision of the RADON platform is to broaden the adoption of serverless computing technologies with a methodology that strives to tackle software complexity, avoid FaaS lock-in, harmonize the abstraction and actuation of action-trigger rules handled with serverless functions, and optimize decomposition and reuse through model-based development and orchestration for FaaS.

This book aims to present the benefits and challenges for software engineers and their managers that arise from serverless technologies and explain how these, in practice, can be tackled using the methods and tools developed within RADON. The RADON framework is illustrated through the handbook on concrete examples and the lessons learned in its application to several industrial use cases. This introductory chapter begins this journey by presenting the broader benefits and challenges posed by serverless computing for companies and software developers.

1.1 Benefits of serverless computing

In the last decade, most organizations have embraced cloud technologies, greatly reducing operation costs through infrastructure scaling and improving service delivery agility through novel engineering practices (e.g., DevOps). However, the first generation of cloud solutions has still seen engineers developing applications designed to run as *cloud-based servers*, a situation that does not eliminate *server management and maintenance costs*.

Instead, *Serverless computing* offers a new delivery model for enterprise software where SMEs and large companies alike can run business logic in the cloud by relying on cheap,

provider-managed servers. Business logic is packaged in simple functions, whose execution is entirely outsourced to the provider for execution, relieving companies from taking care of performance, scaling, and availability issues on their own.

Some of the most significant business benefits of adopting serverless in place (or on top) of existing cloud offerings such as IaaS or PaaS include:

- *Cost reduction.* Serverless functions can be started and stopped by the provider upon demand, reducing to a minimum the overhead for the companies, such as expenses arising from periods of server inactivity while waiting for sensor data.
- *Self-managed scalability.* Scaling complex enterprise applications can involve tens or even hundreds of interacting nodes. Serverless exposes minimum configuration, installation, and operations knobs, freeing companies from the need to provision teams of operations specialists to manage such large-scale applications.
- *Shorter time-to-market.* Serverless is developer-centric, promoting the definition of large collections of reusable APIs that can be easily reused across products, allowing to prototype new functions in a matter of hours.

There are already many business stories that indicate success in Serverless adoption, and that can motivate an organization to look at the RADON technology stack with interest. Some sources indicate that, at this time, up to 50% of Amazon AWS users have adopted Lambda¹.

According to market analysis firms such as Gartner², typical business case requirements that motivate the adoption of serverless include:

- Application reactive behaviour in response to user or environment generated events.
- Sporadic execution, long periods of idleness.
- Applications that undergo periods with large uncertainties on the scaling requirements (e.g., promotional campaigns to attract more customers to a service).
- Short-running workloads with limited resource footprint
- Highly-parallel stateless execution
- Light-weight integration method between services with the ability to produce and pass data

Many such examples arise in practical, day-to-day, technical operations across different vertical domains. Technical use cases of serverless computing span across multiple domains

¹ <https://www.bmc.com/blogs/state-of-serverless/>

² Gartner, A CIO's Guide to Serverless Computing, G00465766, 28 April 2020.

and management issues. Some representative ones include: Asynchronous processing, Real-time data analytics, IoT edge orchestration, Account creation, Extract, transform, load data (ETL), Test automation, among many others. This book will overview examples arising from healthcare, travel technology, and managed DevOps domains.

Overall, the above benefits justify the considerable interest in serverless computing technologies that have emerged in recent years. At the same time, to see the whole picture, it is essential to complete this introduction by looking at open technical, research, and business-driven problems surrounding this emerging technology.

1.2 Challenges posed by serverless computing

Despite the above business case for adopting serverless, several business shortcomings remain for organizations that want to adopt serverless technology. These challenges still represent a risk and limiting factor for serverless FaaS adoption. The following major shortcomings of serverless computing may be identified.

Code and data lock-in. Developers and organizations have different preferences and constraints when storing data and choosing the serverless platform to run functions in the cloud. Investing in a single serverless stack (e.g., Azure Functions) can restrict the added-value services that companies can offer to their customers and lose business opportunities if a new customer uses an incompatible technology stack. However, maintaining products that use multiple clouds, if not done automatically, can duplicate efforts and increase maintenance costs, decreasing the cost-benefits of adopting serverless.

Early-stage adoption issues. FaaS is a game-changer in the software industry, shifting the attention of many IT-driven organizations to adopt this technology. However, there is an inherent lag in adopting new technologies in less technology-driven organizations due to skill shortage and the necessity to build internal proof-of-concept prototypes to persuade management to invest in more advanced IT technologies. In this lies a problem: adopting a new technology stack inherently creates a skill gap in organizations and requirements to define new best practices. There is a shortage in the market of serverless expertise due to the recency of the technology, which means that initial attempts at defining proof-of-concept solutions may be misguided and result in problems or lower-than-expected performance. Adopting serverless without appropriate tools can result in losing time, effort, and a misperception of the issues and weaknesses. Integrating a DevOps culture can also be an important factor in reducing the time to adopt this technology in the organization.

Portability across customer use cases. Applying serverless computing across customer use cases can result in situations where the technology assumptions taken upon creating the product are far from the customer reality, requiring significant reengineering or customization to implement the solution in production. This risk can be mitigated by increasing the level of abstraction in the product design.

Need for architecture reengineering. Serverless FaaS requires packaging an application into functions, requiring a considerable time to decompose a monolith or service-oriented system into a finer grained decomposed architecture. Besides rewriting portions of the code, one issue is the change in design pattern required to adopt serverless due to its innovative event-driven nature without servers that differs from the prior art. For example, the stateless pattern of functions is not suitable for all workloads, and therefore identifying where and when to apply serverless requires considerable conceptual effort in the adopters.

1.3 Why RADON?

To tackle the challenges described in the previous section, RADON proposes a DevOps-oriented framework that enables stakeholders in the software development industry to create and manage applications using microservices and serverless computing technologies. The goal set for the RADON methodology is to tackle complexity, harmonize the abstraction and actuation of action-trigger rules, avoid Function-as-a-Service (FaaS) lock-in, and optimize decomposition and reuse through model-based FaaS-enabled development and orchestration.

As such, the RADON key business value proposition is to provide a DevOps framework to rapidly develop novel FaaS-based software products without incurring code and data lock-in.

The RADON framework is a holistic software environment built upon the open-source Eclipse Che environment. The framework leverages the DevOps nature of the Che environment to supply tools as dynamically retrievable plug-ins and containers that are instantiated at runtime in the end-user environment upon creation of a new Integrated Development Environment (IDE) workspace. In other words, the RADON solution is modular and easy to maintain and evolve because each tool in the IDE is dynamically retrieved and configured with the IDE itself pulling it from a remote Docker or Github repository. On the one hand, this integration approach simplifies the continuous evolution of the framework. On the other hand, this delivery style for an integrated framework is considerably more advanced than the canonical style of extending a pre-DevOps environment such as the classic Eclipse IDE.

RADON couples the standard Eclipse Che IDE with a new range of design and analysis tools. In particular, RADON supports the graphical modeling of applications, combining FaaS, microservices, and data pipelines through models that generalize the TOSCA baseline. TOSCA, an acronym for *Topology and Orchestration Specification for Cloud Applications*, is a standard language proposed by the OASIS standardization consortium to describe a topology of cloud-based services (and microservices), together with their components, relationships, and related lifecycle management functions. Within RADON, an extended version of TOSCA has been supplied, which provides for the first time support for FaaS across multiple providers such as AWS, Azure, and Google clouds. We shall refer to this extended language as RADON TOSCA.

The project introduces on top of RADON TOSCA a constraint definition language (CDL) that allows to define requirements in the form of constraints and desired properties and employs tools for assuring the quality of development with respect to the fulfillment of these requirements and the associated service level agreements, using specialized supporting tools. At runtime, the project supports the orchestration of the application deployment and delivery processes. In this direction, RADON provides a library of reusable and actionable templates and plugins for serverless FaaS and data pipelines. At the same time, it employs monitoring mechanisms to collect evidence towards measuring conformance to the controls and constraints already defined at design time.

This handbook aims to explain these developments in detail, offering a journey to the reader and adopters within a modern research-oriented solution for software engineering of serverless applications.

1.4 An overview of the RADON framework

The RADON framework provides a set of components that realize a set of tools, modules, and services covering both the design and runtime phases of microservices and serverless-oriented application development and deployment. The Architecture Diagram in Figure 1.1 illustrates the broader technical architecture of the framework.

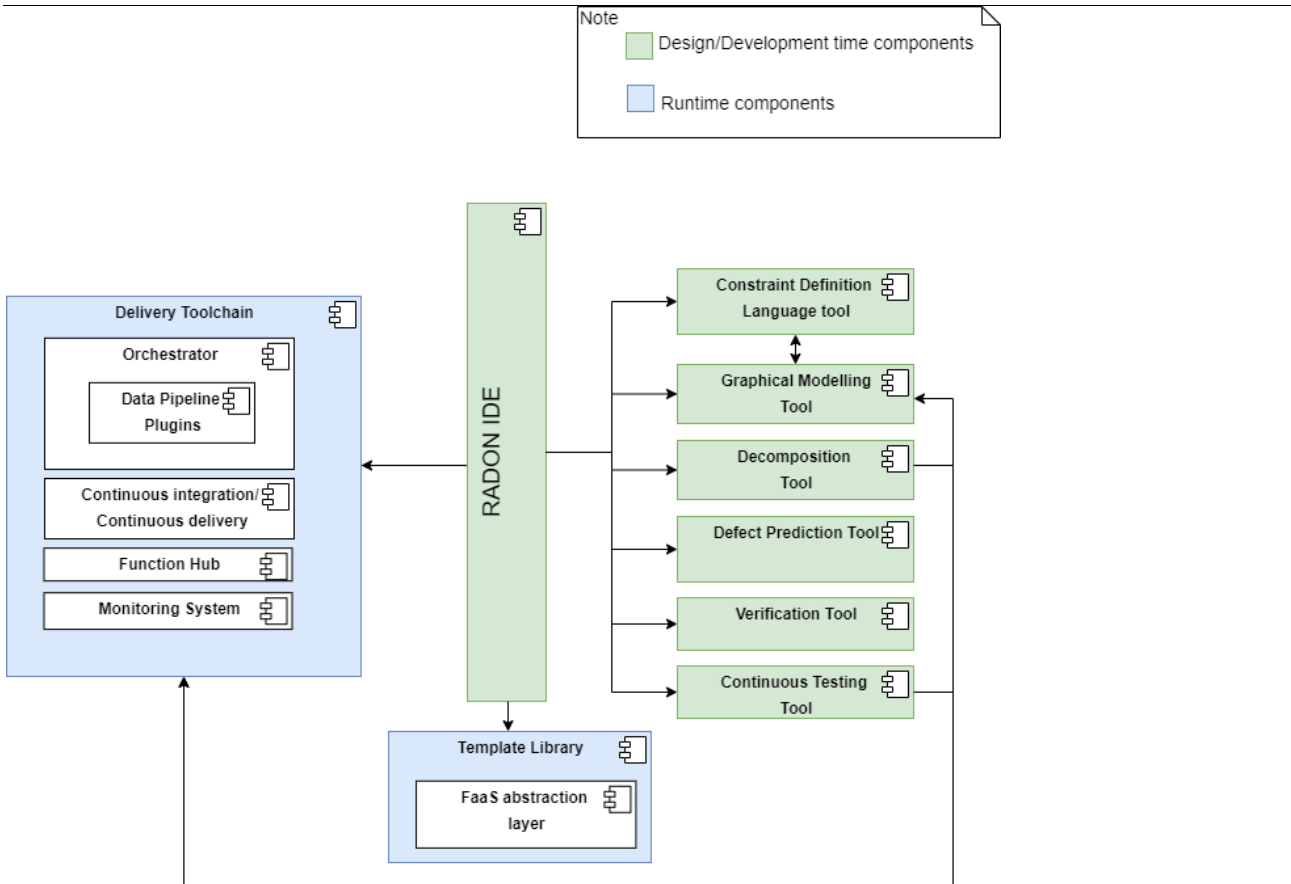



Figure 1.1 - RADON Architecture Diagram

The Architecture Diagram depicts the connections among the RADON components. The design-time components interact with each other and with the runtime components to design, prototype, deploy and test applications built on serverless FaaS. The RADON Workflows define such interaction(s) in the context of the RADON methodology, which are described later in this handbook.

In this context, a particular role is played by the RADON IDE. This component, as mentioned, is based on Eclipse Che and provides a multi-user development environment to access the RADON artifacts. Indeed, as depicted in the architecture diagram, the RADON IDE interacts also with the Template Library. This is done to access the reusable base types, abstractions, and TOSCA extensions and make them available to the RADON tools that require them to model a RADON application. Moreover, the RADON IDE acts as the front-end of the RADON





methodology by enabling users to invoke RADON tools supporting both the design and runtime phases of application development.

The RADON DevOps methodology, described in a dedicated chapter, consolidates the user workflow for using RADON tools and the DevOps paradigm for software delivery and evolution. In the context of a DevOps lifecycle, we have defined several workflows as abstractions to organize and present the possible interactions of the different tools within the RADON framework and with the identified actors. DevOps actors, as described above, are fundamental to reason about the existing development and operations roles and re-assign them for the continuous delivery of software in the context of RADON.

Furthermore, the defined workflows help understand and further refine the application development lifecycle with the RADON framework, considered an iterative process involving Design, Development (Deployment), and Runtime.

The table shown in the next page, Table 1.2, provides the role of each tool in these two phases, together with a brief description of their intended purpose.

Table 1.2 - RADON Tools

| Name | Description | Phase |
|-------------------------|---|------------------------|
| RADON IDE | A web-based multi-user development environment that integrates the RADON tools. | Design time |
| Graphical Modeling Tool | A web-based tool to graphically model TOSCA applications. | Design time |
| Verification Tool | A tool for verifying whether a RADON model conforms to a specification expressed in the Constraint Definition Language. | Design time |
| Decomposition Tool | A tool for architecture decomposition, deployment optimization, and accuracy enhancement. | Design time Runtime |
| Defect Prediction Tool | A tool that focuses on IaC correctness and smells detection. | Design time |
| Continuous Testing Tool | A tool for continuous design, evolution, deployment, and execution of tests. | Design time Runtime |
| xOpera SaaS | A tool for processing and executing the RADON TOSCA service templates packaged in a compressed archive called Cloud Service Archive (CSAR). | Runtime |
| Template Library | A shared repository for templates, blueprints, and modules required for the application deployment. | Design Time Runtime |
| Monitoring System | A back-end service-based system to collect evidence from the runtime environment to support quality assurance. | Runtime |
| Function Hub | A repository to store versioned plug-and-play FaaS packages. | Design time |
| CI/CD Plugin | A plugin to enable CI/CD based on predefined RADON tool pipeline templates. | Runtime |
| Data Pipeline Plugin | A plugin that ensures the functioning of data-pipeline-based CSAR files before they are deployed. | Runtime |

In the next chapters, we will see a description of the RADON tools and the main workflows they operate in.

1.5 Structure of the book

The structure of the book chapters is as follows:

- Chapter 2 offers a methodological overview of RADON explaining the stakeholders of the methodology and the workflow for their coordination and simultaneous use of the RADON framework.
- Chapter 3 introduces one by one the RADON tools describing the problem they solve and their benefits and aims. In most cases, these are coupled with concrete examples.
- Chapter 4 discusses the applicability of RADON to 3 industrial use cases in the domains of travel technology, healthcare and managed DevOps.

The above chapters are then followed by conclusions that point the reader to additional web resources for the reader.

2. RADON Workflow-driven Methodology

In this chapter, we begin our journey by looking at the holistic methodology that RADON proposes to develop serverless based applications. The chapter aims at distilling in compact form the lifecycle model used in RADON, namely, an abstract representation of the high-level phases and their logical interconnections. Afterwards, these high-level phases are decomposed into methodological sub-units, also called fragments, implemented as RADON tools. The usage of each of such tools is further materialized as one or multiple lower-level workflows for designing, developing, and operating RADON applications.

2.1 The RADON lifecycle model


RADON intends to deliver a DevOps-inspired methodology. On that basis, RADON proposes to identify a few reference DevOps actors. A RADON actor defines a role -- not a single human or software -- therefore the same person can potentially act as multiple RADON actors and the same role could be split across multiple actors. The RADON actors are as follows:

- Software Designer: this actor is responsible for the application architecture and data lifecycle design.
- Software Developer (Dev): responsible for business logic coding and testing.
- Operations Engineer (Ops): responsible for delivery on the infrastructure and infrastructure testing.
- QoS Engineer: responsibility for ensuring performance/reliability/security/privacy/access control properties of the application,
- Release Manager: team leader that authorizes major changes and their release to production.

The lifecycle model defines the sequencing of method actions and associated RADON workflows in an application-organization-and-technology-agnostic fashion.

The model is developed applying "*situational method engineering*".³ This tactic allows us to cater to community-, organization-, and project-specific requirements and constraints. It is not geared toward one single base method but rather at synthesizing several "method parts" (sometimes referred to as method chunks or fragments) in an application context-specific

³ Details about the process can be found in D3.1



manner. The key fabric of these method chunks/fragments constitutes the RADON workflows designed by RADON industry partners, grounded on the body of literature.

In contrast to other software lifecycle models, the feedback does not take place only toward the end of a cycle but takes place virtually constantly, which means that the actors involved need intense communication and coordination amongst themselves. The RADON Monitoring Tool plays a center-court role in such coordination. For example, it provides feedback on resource consumption (CPU/memory usage), notifying RADON developers, integrators, and testers to act upon such information with the actual build, decomposition, and verification of the code. Furthermore, the RADON lifecycle model neither prescribes a specific order in which the cycles may be traversed nor demands all lifecycle phases to be activated.

It is also important to understand that the RADON lifecycle methodology does not herald the "one-size-fits-all" mentality. This equally applies to staff involved in executing projects adopting the RADON methodology. Typically, a single RADON "DevOps" team is thus composed of a mixture of staff involved in the entire end-to-end application lifecycle, from development and test to deployment to operations. This implies that application design, quality assurance and security roles are more tightly integrated with development, monitoring, testing and operations roles throughout the application lifecycle. In addition, it requires staff to develop a mixture of skills not restricted to a single traditional function, such as would be the case in more conventional software development lifecycle models. Some (larger) organizations applying the RADON methodology might adopt all roles, done by different staff members, whilst other (smaller) organizations might allocate all these responsibilities to one single staff member.

The RADON lifecycle methodology, as depicted below, has been defined at the macro-level pertaining to the global lifecycle model and their interrelationships, the meso-level referring to organization-specific instantiations, and the micro-level denoting specific serverless application development projects. Organization specific instantiations of the RADON methodology take into account specific resources and domain-specific constraints imposed by the organization, including its size (SME- or large IT departments), maturity (experience level of DevOps), existing IT landscape (legacy system environment), and toolbase and pre-existing IT development and maintenance infrastructure. The project-specific instantiations are typically developed ad-hoc, taking into consideration situational characteristics like application size, complexity and type (e.g., event-driven IoT systems, or production administrative systems), dependencies on other applications, resource capabilities and capacity, and timing and scope.

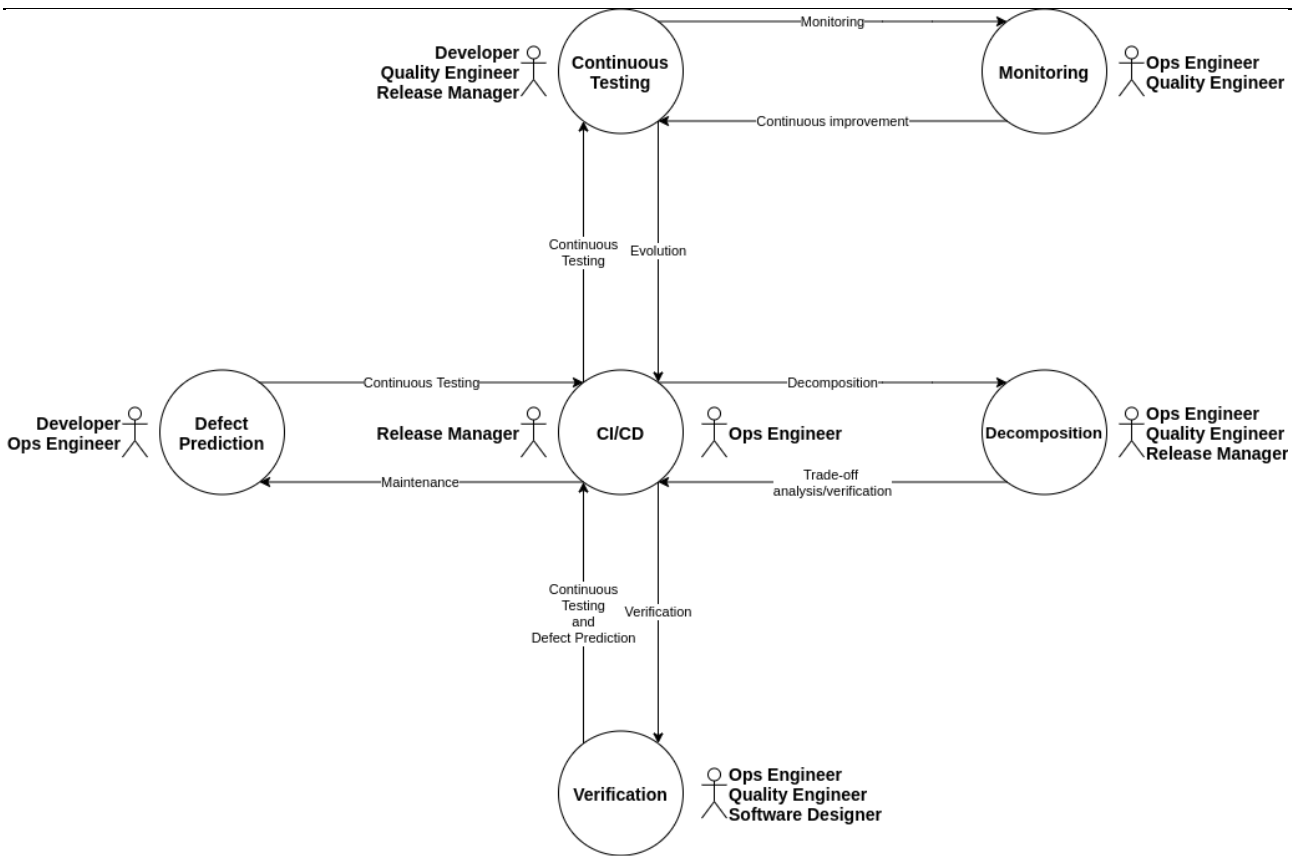


Figure 2.1 - RADON methodological workflow

The entry point of the RADON workflow is the application development that allows users to define FaaS-based applications using a graphical modeling tool, save and reuse previously created templates, and deploy the obtained results.

Starting from this entry point, there are at least six critical phases in a standard RADON-supported lifecycle model, namely, from abstract-level and design time to code-level and run-time:

- The **verification workflow** allows users to define several constraints and verify whether the serverless application complies with such constraints with the final goal of refactoring it to comply with requirements.
- The **decomposition workflow** allows users to decompose a monolithic application from both an architectural and a deployment perspective.
- The **defect prediction workflow** allows users to improve the quality of the codebase by visualizing code metrics, localizing defects, and detecting code smells.

-
- The **continuous testing workflow** allows users to automate the testing activities by continuously generating and testing the applications.
 - The **monitoring workflow** allows users to real-time monitor their applications at runtime.
 - The **CI/CD workflow** allows users to integrate RADON within their CI/CD platform configuration.

Once a serverless application enters the RADON lifecycle model, it is organized as a continuous loop. This process comprises several phases, from code development to its continuous integration and delivery. It exploits a series of feedback loops to incorporate new measured insights (e.g., about quality attributes such as performance and security). The cycle is started anew until the application is discarded for whatever reason.

2.2 RADON Workflows

The RADON methodology adapts to the purpose of the user through the exploitation of the situational method engineering approach.

RADON supports and advocates a comprehensive approach to Microservices and FaaS-based application development. However, it also acknowledges that some users are not developing applications from scratch using the complete RADON methodology, and could be only looking for the mere use of discrete RADON workflows, e.g., rapid prototyping functionality or defect prediction facilities upon existing applications.

The RADON methodology is composed of the six key RADON workflows and associated tools needed to conduct a specific (serverless) application development project. The RADON workflows methods impose structure on specific software development tasks with the goal of making the activity (more) disciplined, systematic, repeatable and predictable. The RADON tools have thus been designed for the explicit support of the RADON workflows, maximizing the level of automated support.

In the following, we consider using the RADON tools, which the next figure depicts in an integrated way, in the context of six different workflows and the methodology entry-point that illustrate alternative ways to exploit the RADON framework.

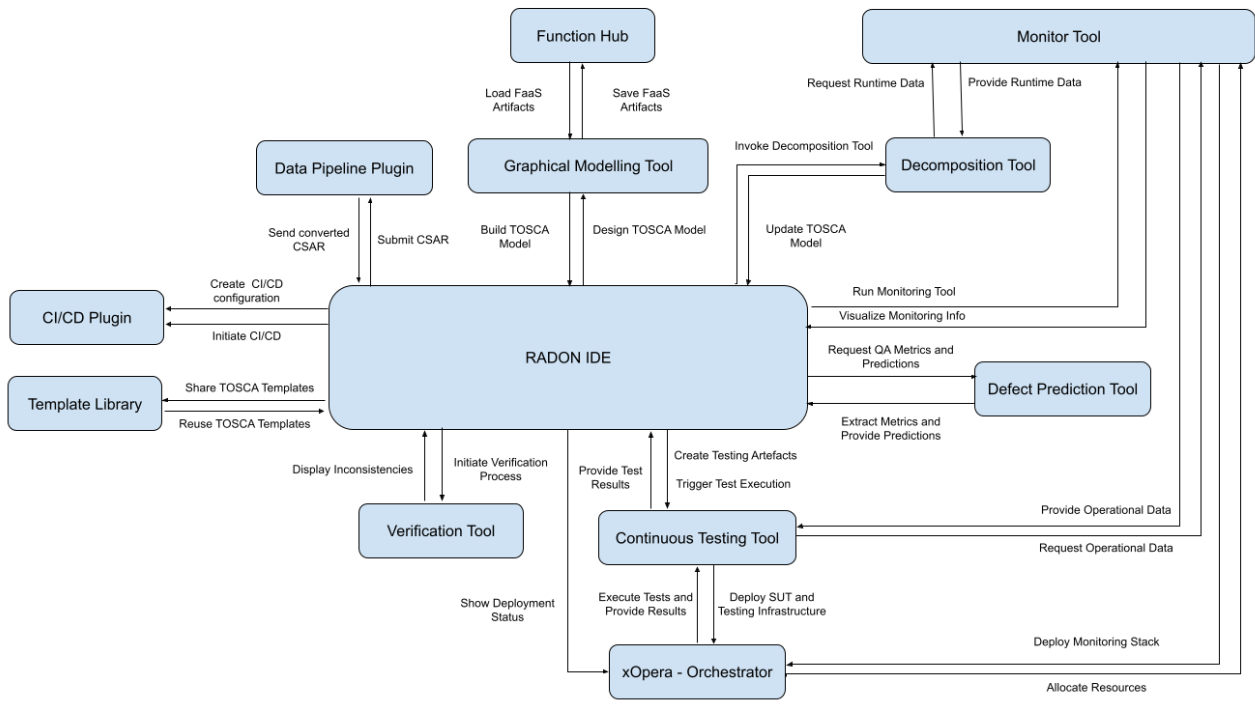


Figure 2.2 - RADON integrated tools and related workflows

The table in the next page provides an overview of RADON workflows with involved actors and tools.⁴

⁴ Full details concerning each workflow are provided in [D3.1](#)

Table 2.3 - RADON Roles

| RADON workflows | Description | Roles⁵ | Tools |
|--------------------------------------|---|---|--|
| Entry-point: Application Development | It allows users to define FaaS-based applications using a graphical modeling tool, save and reuse previously created templates, and deploy the obtained results. | Software Developer Release Manager | IDE Graphical Modeling Tool Data Pipeline Plugin Function Hub Template Library Orchestrator |
| Verification | It allows users to define several constraints and verify whether the serverless application complies with such constraints with the final goal of refactoring it to comply with requirements. | Software Designer QoS Engineer | IDE Verification Tool Graphical Modeling Tool |
| Decomposition | It allows users to decompose a monolithic application from both an architectural and a deployment perspective. | Software Designer Ops Engineer QoS Engineer | IDE Decomposition Tool Graphical Modeling Tool Monitoring Tool |
| Defect Prediction | It allows users to improve the quality of the codebase by visualizing code metrics, localizing defects, and detecting code smells. | Software Developer Ops Engineer | IDE Defect Prediction Tool |
| Continuous Testing | It allows users to automate the testing activities by continuously generating and testing the applications. | Software Developer Release Manager QoS Engineer | IDE Continuous Testing Tool Orchestrator Monitoring Tool |
| Monitoring | It allows users to real-time monitor their applications at runtime. | Ops Engineer QoS Engineer | IDE Monitoring Tool Graphical Modeling Tool Orchestrator |
| CI/CD | It allows users to integrate RADON within their CI/CD platform configuration. | Ops Engineer Release Manager | IDE CI/CD Plugin |

⁵ Roles were identified in [D2.1](#)

3. Tools Overview

This section summarizes the RADON tools we have introduced earlier in the handbook. Some of the tools are design-focused, while some are runtime-oriented. Finally, some have both design and runtime aspects.

3.1 RADON Integrated Development Environment

Overview. The RADON Integrated Development Environment (IDE) provides a development environment for multi-user usage. Based on the Eclipse Che technology, the RADON IDE supports standard (web-based) development activities (such as support for different programming languages, debugging functionalities, source code editors). Furthermore, it provides a front-end to interact with the RADON framework and its tools and access to the shared spaces of the RADON artifacts (i.e., RADON models).

High-level architecture. The Eclipse Che development environment has been customized to realize a new RADON Stack (i.e., a runtime configuration) defining a RADON workspace. A RADON workspace is characterized by a set of plugins, projects, and Kubernetes containers implemented to customize the development environment and integrate the RADON tools according to the project needs.

The RADON IDE comprises several Eclipse Che plugins that integrate the RADON tools. These plugins add capabilities to the Eclipse Che GUI and permit interaction with the RADON tools. Moreover, some Kubernetes components have been defined in the RADON workspace to integrate services of some RADON tools (i.e., GMT, VT, CTT) in the IDE "backend".

Finally, the RADON workplace is also characterized by a project (named “**radon-particles**”) that clones in the RADON workspace the TOSCA modeling entities from the RADON Particles GitHub repository. A detailed description of the RADON IDE is provided in the deliverable [D2.7](#).

3.2 Graphical Modeling Tool

Overview. The Graphical Modeling Tool (GMT) is a web-based environment to graphically model TOSCA topologies. The environment includes a type and template management component to offer creation and modification of all elements defined in the TOSCA specification. All information is stored in a repository, which allows importing and exporting using the TOSCA packaging format.

Business Purpose. GMT provides an usability layer on top to maintain your TOSCA files in a graphical and intuitive user interface. It provides a graphical web-editor with which you can create and maintain all TOSCA entities. Thereby, GMT stores all TOSCA entities in a defined folder structure that fosters the reusability of TOSCA types, while validating and storing all TOSCA entities in the syntax defined by the standard. Further, the graph-based representation of TOSCA topologies in GMT provides a quick overview of the entire system and offers a communication basis for the cooperation with other parties. It therefore offers a quicker introduction to modeling with TOSCA and provides newcomers with necessary guidelines.

How the tool works. GMT is published as a Docker container. The image contains two web applications for TOSCA type and template management as well as topology modeling. Further, the backend is exposed as a RESTful HTTP API, which is used by its provided web applications and can be used by external tools for integration purposes. GMT expects that the Template Library is mounted as a directory into the container at runtime. This assumes that either the RADON Particles (RADON's public version of the Template Library) GitHub repository is cloned locally or the content from the Template Publishing Service is exported to a directory and respectively mounted into the container. GMT is designed and developed loosely-coupled from the actual content that is used by users to model TOSCA application deployments. The content is provided by the Template Library containing different TOSCA entities, such as node types, policy types, requirement types, and even service templates. As a result, users may extend the content of the Template Library to introduce new types and entities for modeling. For this purpose, users may also use the GMT to create and define new TOSCA types. The created content can be pushed to the RADON Template Publishing Service or even pushed to the RADON Particles on GitHub.

A practical example. Start GMT either through the RADON IDE or as a standalone tool following our [user guide](#). The main entry point is the TOSCA Management UI, which can be used to manage all of the TOSCA entities inside the current data repository. The data

repository, however, can be mounted into the container, e.g., based on RADON Particles, or, in case of RADON IDE, the types and templates from the “radon-particles” project folder are configured by default. From here, users may create new TOSCA node types or start to develop a new deployable blueprint by creating a TOSCA service template. When creating new TOSCA blueprints, the TOSCA Topology Modeler of GMT is used. Once the TOSCA Topology Modeler UI is opened, it starts with an empty modeling canvas. Users can drag-and-drop components from the palette on the left-hand side of the editor to the canvas to model their intended application structure. For example, you can drag the “AwsPlatform” entry to the canvas to create a new TOSCA node template of this type. In addition, you may want to add an additional node template of type “AwsLambdaFunction”. By selecting a node, users are able to edit the node’s properties in the edit. Further, the editor helps to create relationships between nodes, e.g., to express that the Lambda function is hosted on the AWS platform node. The “Requirements & Capabilities” view (activated by the top menu) lets users drag from the “HostedOn” relationship entry of the requirement “host” to the “host” capability of the “AwsPlatform” node to establish a relationship of this type between these nodes. Lastly, the TOSCA Management UI of GMT is able to “Export” a TOSCA service template as a CSAR file, either to *download* the CSAR or to save it to the filesystem (relative to the configured repository path). The generated CSAR is self-contained and contains all type definitions, implementation, and deployment artifacts to deploy the application by a TOSCA orchestrator.

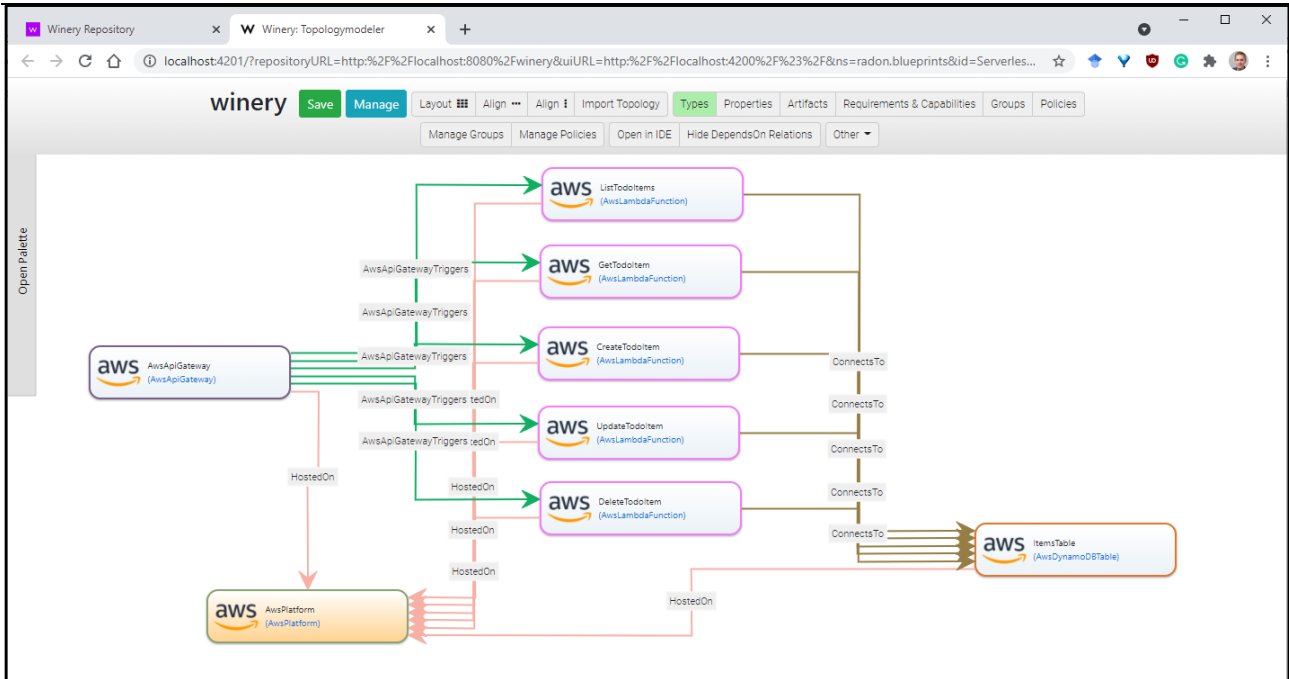


Figure 3.1 - GMT example

Open challenges. GMT can currently only be operated based on a file-based data repository. This means that either the RADON Particles need to be cloned locally or a respective directory needs to be provided following the expected directory and file structure (see D4.4 RADON Models II) and mounted into the provided Docker container. Further, a future work item for GMT is to provide support for TOSCA YAM imperative workflows. Currently GMT only supports the creation of BPMN-based workflows and would require an extension to also model deployment and management workflows based on the YAML syntax specified in the standard.

Getting started. A detailed description of the Graphical Modeling Tool is provided in deliverables [D4.5](#) and [D4.6](#). A user guide is also available online showing how to start GMT in Docker (<https://winery.readthedocs.io>). Further information can be found at the Eclipse Winery project site (<https://projects.eclipse.org/projects/soa.winery>) as well as in the respective GitHub repository (<https://github.com/eclipse/winery>) since GMT has been developed based on Eclipse Winery and all advancements have been merged back to the official Eclipse repository.

3.3 Verification Tool

Overview. The primary purpose of the Verification Tool (VT) is to allow a user to check that a given TOSCA model conforms to a set of functional and non-functional requirements, expressed in the RADON Constraint Definition Language (CDL). In addition, its correction and learning modes allow it to repair invalid TOSCA models and extend incomplete CDLs specifications (respectively).

Business Purpose. One target market for the CDL and VT is cloud and Internet of Things (IoT) providers. The CDL is predominantly aimed at Quality of Service (QoS) Engineers, who define requirements as CDL specification, which can then be verified using the VT. Software designers can use the VT to aid the design process, using it to suggest extensions and corrections to the current architecture. A second potential target market is policy management platforms. Software designers are able to use the CDL to define access control policies, both manually and through machine learning, which can then be verified using the VT.

How the tool works. The verification mode of the VT works by translating a CDL specification and a RADON model into the language of answer set programming (ASP). The translation to ASP ensures that the solutions of the resulting ASP program correspond exactly to the inconsistencies between the CDL specification and the TOSCA model. An off the shelf ASP solver is used to search for these ASP solutions, which are then post-processed and displayed to the user. ASP solvers are used to solve a complete ASP program. The latter two modes of the VT require extending/modify the ASP translation of a CDL specification and TOSCA model (the correction mode requires the VT to modify or extend the part corresponding to the TOSCA model and the learning mode requires the VT to extend the part corresponding to the CDL specification). For this purpose, we use the ILASP system for Inductive Learning of Answer Set Programs, which is capable of extending (and through standard meta-level representations) modifying ASP programs.

A practical example. Start VT either through the RADON IDE or as a standalone tool following our [user guide](#). Download the [service template](#) and the simple set of constraints in the CDL file [main.cdl](#). The constraints state that if a lambda function accesses sensitive data (where the nodes which contain sensitive data are given in the CDL specification), then it should be hosted on the same AwsPlatform as the data. In this simple case, the constraints mean that the resource components in the model have to be hosted on the same AwsPlatform. ServerlessToDoListAPI has been implemented as such, so when we invoke the

verification test importing the service template of the created application, no inconsistencies are found as expected. After importing this file into the IDE workspace, the VT can be invoked choosing the option "Verify" by right clicking on the CDL file. The example returns that no inconsistencies have been detected.

In order to put the VT to the test, we will slightly modify the topology model of the ServerlessToDoListAPI hosting 1 of the 5 lambda functions (DeleteToDoItem) to a different region than the rest of the components, resulting in the topology below.

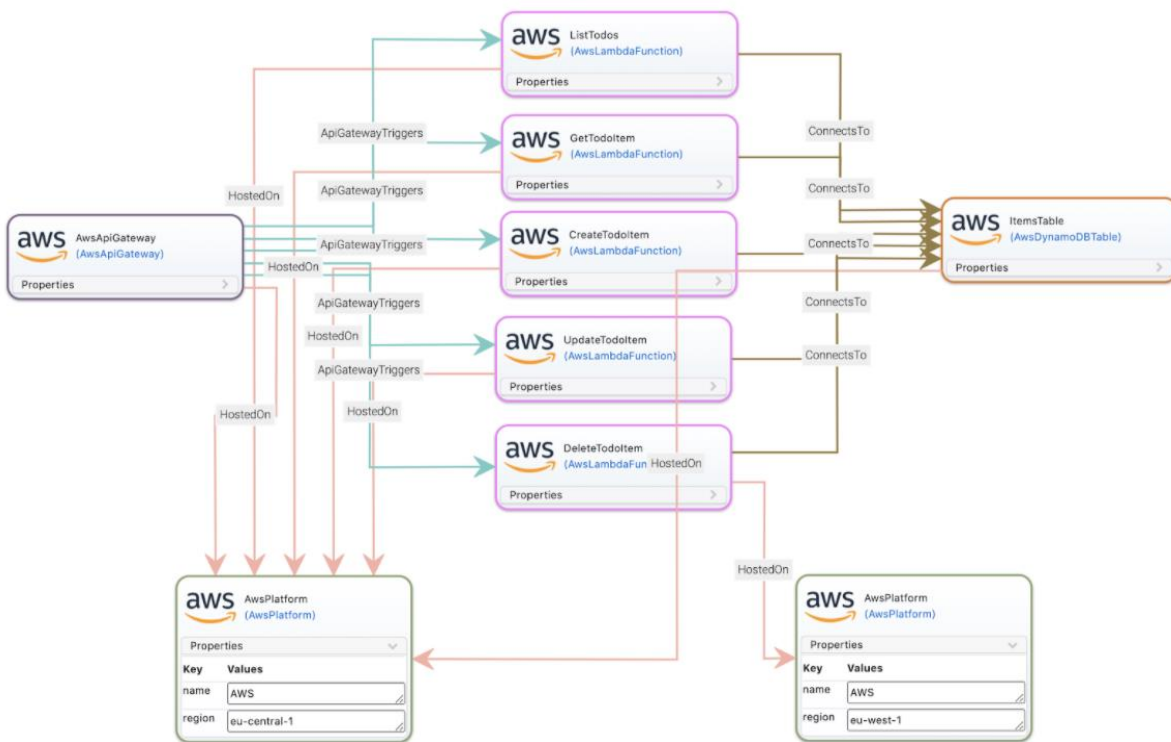


Figure 3.2 - VT example

As we can see, the modified topology does not comply with the CDL specification (**AwsLambdaFunction_4** is stored on a different platform to **AwsDynamoDBTable_0**, and because the table contains sensitive data and is accessed by the lambda, this is a violation of the constraints). If we invoke the VT again (again by right-clicking on **main.cdl** and selecting the "Verify" option), we will see the following output from the VT.

```
Problems >_ RADON Verification Tool x
===== Inconsistency 1 =====
Detected sensitive_data_issue inconsistency. The following assertions are sufficient to demonstrate the inconsistency:
(base[0]) "lambda" = "AwsLambdaFunction_4"
(base[0]) "lambda" = fn((fn("AwsApiGateway_0","requirements"),(at,6)),"invoker").node
(base[0]) "sensitive_node" = "AwsDynamoDBTable_0"
(base[0]) "sensitive_node" = fn((fn("AwsLambdaFunction_0","requirements"),(at,1)),"endpoint").node
(base[0]) "sensitive_node" = fn((fn("AwsLambdaFunction_1","requirements"),(at,1)),"endpoint").node
(base[0]) "sensitive_node" = fn((fn("AwsLambdaFunction_2","requirements"),(at,1)),"endpoint").node
(base[0]) "sensitive_node" = fn((fn("AwsLambdaFunction_3","requirements"),(at,1)),"endpoint").node
(base[0]) "sensitive_node" = fn((fn("AwsLambdaFunction_4","requirements"),(at,1)),"endpoint").node
```

Figure 3.3 - VT output

This prompts the modeler to make modifications accordingly to resolve the inconsistency between the CDL specification and the model.

Open challenges. The main limitation of the VT is the lack of scalability of the latter two modes with respect to the size of the search space for possible corrections and learned constraints. Both of these modes depend on the ILASP system. Current versions of ILASP are known to suffer from such scalability issues (although ILASP is being actively developed and improved with the aim of overcoming such issues). Another recent system for learning answer set programs called FastLAS is far more scalable, but current versions are less general than ILASP and unable to solve the tasks required by the VT. In current work, we are aiming to extend the FastLAS approach to full answer set programs, which could automatically improve the scalability of the VT.

Getting started. A detailed description of the Verification Tool is provided in deliverables [D4.1](#) and [D4.2](#). Information on getting started with the VT is available on Read The Docs (<https://radon-vt-documentation.readthedocs.io/en/latest/>).

3.4 Decomposition tool

Overview. The Decomposition Tool (DT) helps RADON users to find the optimal decomposition solution for an application based on the microservices architectural style and the serverless FaaS paradigm. It is typically used in four different scenarios: (i) architecture decomposition, (ii) deployment optimization, (iii) accuracy enhancement.

Business Purpose.

The first purpose is to help software designers who want to decompose the architecture of a monolithic application. The DT targets to generate a coarse-grained or fine-grained TOSCA model from such application. The second purpose is to make it possible for operations engineers to optimize the deployment of platform-independent or platform-specific TOSCA models on a particular cloud platform. With DT, they can obtain the optimal deployment scheme that minimizes the operating costs on the target cloud platform. The third purpose is to help operations engineers to enhance the accuracy of the description according to runtime monitoring data so that a better decomposition or optimization result may be achieved afterwards.

How the tool works. The implementation of the decomposition tool is based on a chain of tools and data structures illustrated in the following Figure 3.4. Given a RADON model, the tool applies a built-in YAML processor to import the service template into MATLAB and generates its topology graph and the embedded LQN automatically through model-to-model transformation. As shown in the left figure, an optimization problem is then created from the topology graph and solved by invoking the GA solver and the LINE engine. As shown in the right figure, a decomposition problem is then created from the topology graph and solved by invoking the Schema.org2 dataset and the DKPro tool. When the optimal solution or decomposition solution is found, the tool returns the result back to the original service template.

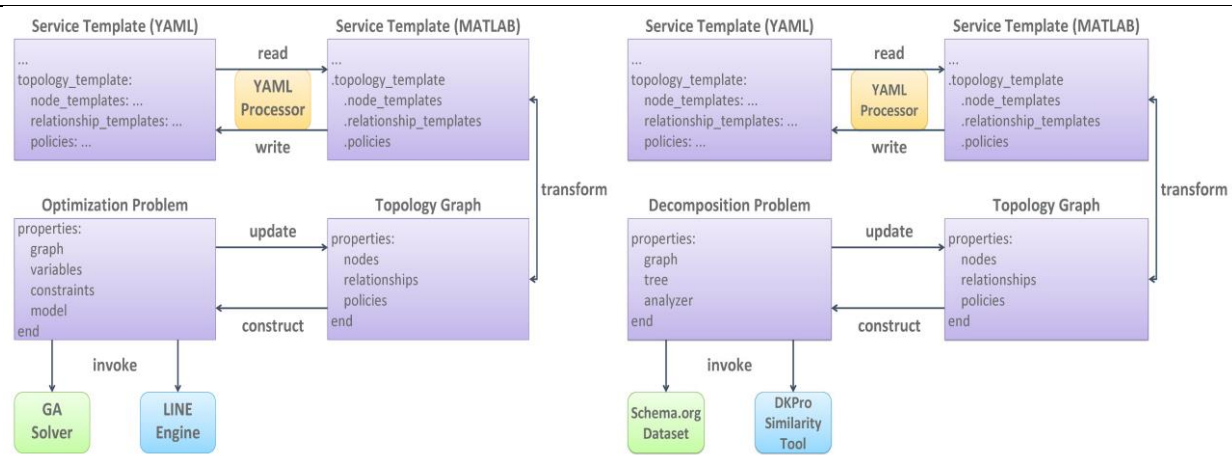


Figure 3.4 - DT technical architecture

A practical example.

The Decomposition Tool is used within the RADON IDE to decompose the architecture of an abstract RADON model and to optimize the deployment of a concrete RADON model. To get started, you can clone the decomposition tool sample project in the workspace: 1.Press *Ctrl+Shift+P* to open the command palette. 2.Select the Git:Clone command. 3.Type the repository URL of the decomposition tool sample project. 4.Press Enter to clone the project in the workspace. An example output showing the cost per year of running with the target deployment option is shown below:

```
Problems Output x
Extension-Host:Start deployment optimization of open_model.tosca
Extension-Host:Successfully uploaded the original model to the server
Extension-Host:Successfully optimized the deployment of the model
Extension-Host:Successfully downloaded the resultant model from the server
Extension-Host:Deployment optimization of open_model.tosca complete
Extension-Host:{
  "total_cost": 0.03059409192044063
}
Total operating cost per year: 261.39592136824473
```

Figure 3.5 - DT output for optimization feature

The example project includes three folders, namely mono-app, micro-app and demo-app. The mono-app and micro-app folders provide sample service templates, model.tosca, for an



abstract monolithic and an abstract microservice application respectively. The demo-app folder provides two sample service templates, open_model.tosca and closed_model.tosca, for a concrete demo application (thumbnail generation).

To invoke the decomposition functionality of the DT, right-click on model.tosca in either the mono-app or the micro-app folder and select the Decompose option. The execution of the decomposition procedure will be displayed in the Output window (*Ctrl+Shift+U* to open). After the decomposition procedure completes, the service template will be updated according to the desired decomposition solution. An example of the tool output is given below.

```
Extension-Host:The RADON decomposition plugin is now active!  
Extension-Host:Start architecture decomposition of model.tosca  
Extension-Host:Successfully uploaded the original model to the server  
Extension-Host:Successfully decomposed the architecture of the model  
Extension-Host:Successfully downloaded the resultant model from the server  
Extension-Host:Architecture decomposition of model.tosca complete  
Extension-Host:{}
```

Figure 3.6 - DT output for decomposition feature

Open challenges. To simplify architecture decomposition, the Decomposition Tool ignores specific technologies in use and works on abstract RADON models, which are not deployable. As for deployment optimization, the Decomposition Tool only supports a limited range of AWS services at present. Further extension of this feature to other AWS services and cloud platforms is desirable. Extra assumptions are also made in the optimization program due to the inability of LQNs to capture the exact behavior of certain node types, for example, cold start and retrieval of Lambda functions.

Getting started. A detailed description of the Decomposition Tool is provided in deliverables [D3.2](#) and [D3.3](#). Information on getting started with the DT is available on Read The Docs (<https://radon-ide.readthedocs.io/en/latest/>).

3.5 Defect Prediction Tool

Overview. The RADON IaC Defect Prediction Tool strives to tackle correctness in designing applications based on serverless computing. In particular, it is designed to help DevOps engineers to allocate effort and resources more efficiently during Quality Assurance activities by prioritizing their inspection efforts for IaC scripts that might be failure-prone.

Business Purpose. The defect prediction tool is the first-of-its-kind tool to ensure the software quality of infrastructure code (IaC) with the final aim of supporting its maintenance and evolution. The tool consists of four individual components. The Github IaC Repositories Collector collects active IaC repositories on GitHub. The Repository Scorer computes repository metrics based on best engineering practices used to select relevant repositories. The IaC Repository Miner mines failure-prone and neutral IaC scripts from a repository together with a broad set of IaC-oriented metrics computed upon the collected IaC scripts to predict their failure-proneness. The IaC Defect Predictor pre-processes the datasets and trains the Machine Learning models. Given an unseen IaC script, this component classifies it as failure-prone or neutral. Software Developers and Ops Engineers can rely on such models to refactor Ansible and TOSCA blueprints to make them more comprehensible and maintainable.

How the tool works. From a backend perspective, the tool uses Firebase to store failure-prone data related to a repository and the built models. After the user has collected relevant data through the RADON Defuse GUI (see link below), a model is automatically built for the language and defect of choice. For the sake of performance and explainability, Decision Tree is used to build the defect prediction models. However, new algorithms will be added soon along with the possibility of tuning them. Once a model is created, the user can access it through the GUI, see its performance and download it. The download .joblib file can be integrated in a CI/CD pipeline or deployed online to be used by the plugin.

A practical example. The Defect prediction tool analyzes the characteristics of IaC blueprints to predict their defect-proneness. Its models can be used within the RADON IDE, via a dedicated plugin, or in Continuous Integration pipelines. The tool's output consists of TOSCA and Ansible files metrics along with hints on the blueprints to investigate.

The models can be invoked within the plugin by performing "Run detection" on the csar final of the final application to be deployed.

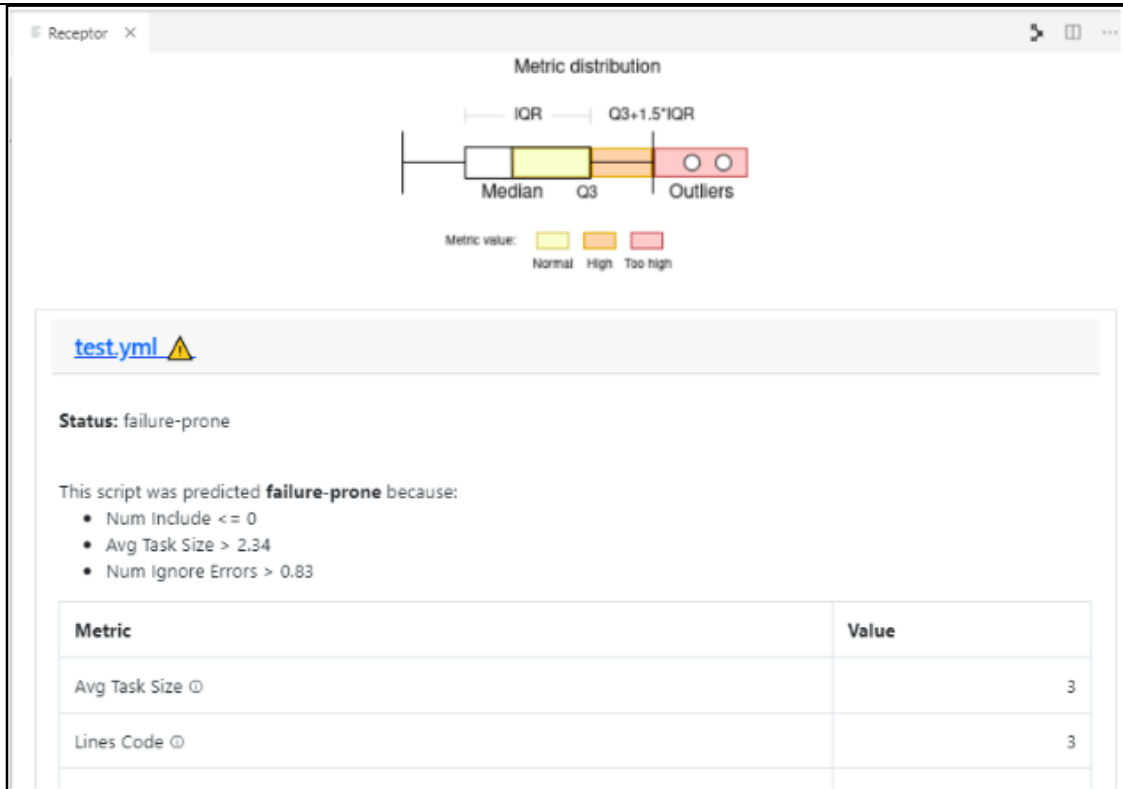


Figure 3.7 - Defect prediction output

A new tab opens in the IDE workspace, showing a list with all the files suitable for the analysis. For each file, the plugin provides a set of metrics by clicking on them, highlighting whether each particular file is defective or not, and providing an interpretation of the prediction.

Open challenges. The tool currently supports only two IaC languages (i.e., Ansible and TOSCA) and five kinds of defects. However, more languages and defects could be easily integrated in the future. The defect prediction tool identifies defects at the file level. Future implementation will target lower levels of granularities (e.g., feature level, line level).

Furthermore, because the tool relies on Firebase, the end-user must stick to the Firebase billing plans. The current version is suitable for the main usage scenario and uses the Firebase Spark plan, which offers generous limits for getting started with Firebase. Therefore, it does not require the user to subscribe to any billing plan. In case the end-user wants to increase the current quota and storage limits, she can subscribe to a paid-tier plan⁶ without affecting the tool functioning.

⁶ <https://firebase.google.com/pricing>

Getting started. A detailed description of the Graphical Modeling Tool is provided in deliverables [D3.6](#) and [D3.7](#). Below you can find the most relevant defect-prediction tool-related repositories:

- *RADON Defuse*⁷, the commit annotator and model builder to assist developers and operators to collect failure-prone data of infrastructure code, build, and evaluate models through a graphical user interface.
- *RADON Repository Collector*⁸, a tool to crawl Github for IaC repositories.
- *RADON Repository Miner*⁹, a tool to mine IaC scripts contained in a repository.
- *RADON Ansible Metrics*¹⁰, a tool to extract source code metrics from Ansible playbooks.
- *RADON Tosca Metrics*¹¹, a tool to extract source code metrics from Tosca blueprints.
- *RADON Defect Predictor Plugin*¹², the plugin integrated in Eclipse Che.

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

⁸ <https://github.com/radon-h2020/radon-repositories-collector>

⁹ <https://github.com/radon-h2020/radon-repository-miner>

¹⁰ <https://github.com/radon-h2020/radon-ansible-metrics>

¹¹ <https://github.com/radon-h2020/radon-tosca-metrics>

¹² <https://github.com/radon-h2020/radon-defect-prediction-plugin>

3.6 Continuous Testing Tool

Overview. The Continuous Testing Tool (CTT) provides the functionality for defining, generating, executing, and refining continuous tests of application functions, data pipelines, and microservices and reporting test results. CTT integrates with other RADON tools and can be used as a standalone tool.

Business Purpose. CTT enriches the TOSCA ecosystem by end-to-end support for continuous testing of microservice-based, FaaS, and data pipeline applications in DevOps. It is the first tool of its kind that supports the whole workflow — from test specification over execution and reporting to automated updates based on production data — that is also extensible to custom needs, e.g., integration of other types of tests or tools.

How the tool works. A user defines tests by adding them to a TOSCA service template for the system under test (SUT) — most conveniently via GMT. Via the RADON Particles, CTT provides tailored TOSCA node types, relationship types, and policy types for expressing different types of tests and including suitable test drivers. For instance, CTT allows the definition of a load test to be executed using a configured load driver such as JMeter. Also the test infrastructure (TI) is defined as a TOSCA service template.

The deployment and execution of tests is managed by the CTT server. Via the REST-based interface, users can execute the continuous testing on-demand via the RADON IDE or the CTT command-line tool, or include it as a part of the CI/CD process. After being deployed by a TOSCA orchestrator such as xOpera, the tests are executed and the test results are made available to the user.

CTT is designed as an extensible framework that allows the definition of new test types, metrics, and tools. CTT integrates with novel research approaches for DevOps-oriented load testing in continuous software engineering.

A practical example. We will demonstrate the main steps of using CTT with an example. The SUT is a FaaS-based implementation of a ToDo-list using AWS services, especially AWS-Lambda functions. We will define an endpoint test that assesses whether the deployment was successful.

We assume that a TOSCA model for the SUT exists. Within the model of the SUT, so-called policies add the information about the tests that CTT will later execute. Figure 3.8 shows an excerpt of a test policy (for a deployment test) assigned to the SUT in GMT.

The RADON IDE is a possible interface to execute the tests via CTT. Figure 3.9 show a workspace with the SUT and test-related artifacts, as well as the editor with a CTT configuration to be defined.

The configured tests can be deployed and executed by selecting the respective option from the configuration file's context menu. CTT's RADON IDE plugin communicates with the CTT server.

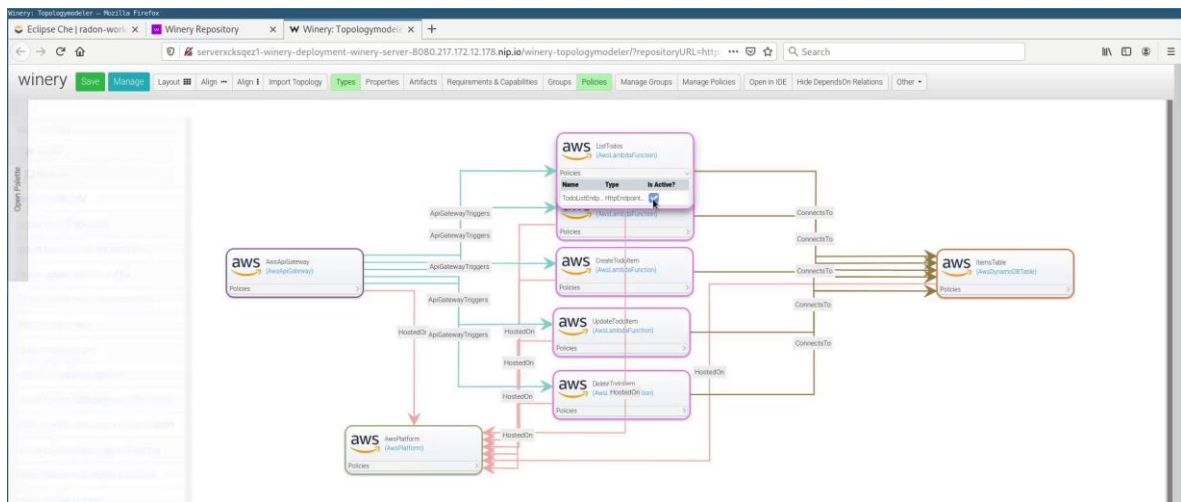


Figure 3.8 - Deployment test configured in GMT


```

1  {
2    "name": "ServerlessToDoList-DeploymentTest",
3    "repository_url": "ServerlessToDoList",
4    "sut_tosca_path": "todolist.csar",
5    "ti_tosca_path": "deploymentTestAgent.csar",
6    "ti_inputs_path": "inputs.yaml",
7    "test_id": "test_1",
8    "result_destination_path": "serverless-test-results.zip"
9  }
10

```

Figure 3.9 - SUT and test-related artifacts in the RADON IDE

Alternatively, for example in CI/CD pipelines, CTT can be triggered via a dedicated command-line tool with a simple command like `./ctt_cli.py -u "http://localhost:18080/RadonCTT" -c ctt_config.yaml`.



In any case, the test results will be provided in the configured output folder for further inspection.

Open challenges. Currently, CTT supports only a basic set of test types and test infrastructures. However, CTT has been designed explicitly for custom extensions.

Getting started. A detailed description of the Continuous Testing Tool is provided in deliverables [D3.4](#) and [D3.5](#). Step-by-step instructions are provided in CTT's Read the Docs page¹³. Various GitHub repositories provide the source code of CTT's components and examples; the starting point is CTT's main repository.¹⁴

¹³ <https://continuous-testing-tool.readthedocs.io>

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

3.7 xOpera SaaS Orchestrator

Overview. xOpera SaaS is an advanced TOSCA orchestrator available as a service and built on top of the xOpera orchestrator engine. The SaaS component provides an isolated xOpera orchestrator environment to each deployment project, to which the owner can attach secrets and share with other users. The ability to run *as a service* makes the deployment projects available to teams of users. They can directly use or monitor the project through the GUI or integrate xOpera SaaS using the API into their CI/CD workflow.

Business Purpose. The xOpera orchestrator aims to be a lightweight orchestrator compliant with [OASIS TOSCA](#) and the current compliance level is with the [TOSCA Simple Profile in YAML v1.3](#). `opera`, the CLI tool, is a (TOSCA) cloud orchestrator which enables orchestration of automated tasks within cloud applications for different cloud providers such as Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), OpenFaaS, OpenStack and so on. The tool can also be used and integrated into other infrastructures in order to orchestrate services or applications and therefore reduce the human factor. The service oriented version, xOpera SaaS, includes advanced features such as secret management, user, access and multi-tenancy management and receiving notification callbacks.

How the tool works.

Using the browser version is straightforward. The basic workflow is simple, and includes that you:

1. Have secrets you need to define prior to deployment.
2. Author a new workspace to contain your project.
3. Register your secrets to be available in the workspace.
4. Use the browser to create a new xOpera project from a CSAR.
5. Have to specify which service template and inputs you are using, then validate them.
6. In the end, deploy the project.

xOpera SaaS can also be used through a HTTP API.

A practical example.

The first thing we need to do is create whatever secrets are necessary for your deployment to run. For example, these are your cloud provider secrets, SSH public keys, among others. The way they are provided is through files - with each secret, you declare a file (and contents) that will be present in your project when you create it.

Creating a workspace is simple, you just need to choose a name. You are assigned owner privileges automatically, and you can share this workspace with other users, who can then also create projects in it.

The next thing we need to do is to assign the secrets we created in the previous step to this workspace. This is the only way they are applied to projects within this workspace. As with sharing workspaces, this is done through the dropdown on the right of each workspace's row.

All that is left is to create and deploy a project. To do this, click the Add Project button, choose a name and select your CSAR file.

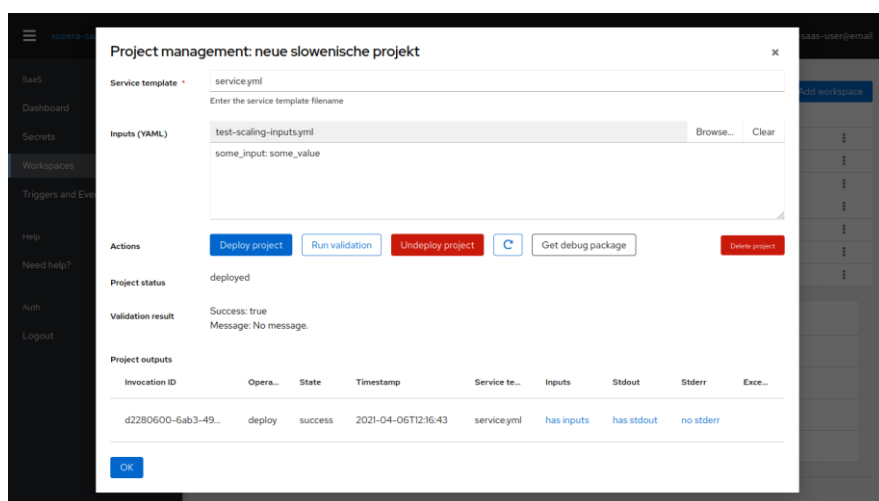



Figure 3.10 - The main xOpera SaaS project management screen

To deploy the project, open the management window, input your service template filename and upload your inputs file using the Browse button. You can Run validation on the service template and inputs prior to deploying as a basic sanity check.



Open challenges. xOpera project consists of multiple pieces that co-exist together. The core engine used through a CLI or the HTTP API is open sourced and gives you the ability to orchestrate any supported provider. However xOpera SaaS is currently proprietary software that extends the possibilities to use the xOpera and extends the functionalities with OpenID user management, secret management and more. In the future the orchestrator will be supported and extended by the needs of the community and business clients.

Getting started. A detailed description of the use is described in [D5.1](#) and [D5.2](#) or in the online documentation¹⁵ and project github pages¹⁶.

¹⁵ <https://xlab-si.github.io/xopera-docs/saas.html>

¹⁶ <https://github.com/xlab-si/xopera-opera>

3.8 Template Library

Overview. The Template Library is a place for storing, publishing, and sharing TOSCA modules and blueprints. Users can manage their templates in their local storage, community repositories, e.g., RADON particles on GitHub or in the Template Publishing Service (TPS). As the management of the content in local folder or git repository is quite straight forward for the GMT and RADON IDE, we focus here on the TPS, which provides the ability to search for templates, filter them by different parameters, and download/publish TOSCA content.

Business Purpose. The main purpose of the Template Library is the management of the TOSCA component and service templates. Currently, we are targeting two different scopes. First is the community one, which can use the GitHub repository and Template Library Publishing System, to develop, share and publish the templates. For the more advanced users, we provided a special feature in TPS, where a user can privately share the content or use xOpera SaaS orchestrator to directly create a project from a template published in TPS. For easier search through the content the TPS has a web GUI interface.

How the tool works. Template Library is a standalone service and is exposed through a RESTful HTTP API that comes with an OpenAPI specification and Swagger UI. The backend is written in the KTOR Kotlin framework and can be set up in a Docker container. We use Traefik reverse proxy to expose all services and Keycloak for IAM. On top of the API we have a GUI, which was designed with PatternFly. The API enables users to publish templates and then view them in the GUI. All these actions can be also initiated from the terminal with the TPS CLI package that is available and versioned on PyPI. The API is also used by the TPS RADON IDE plugin that offers invoking TPS actions such as publishing and downloading TOSCA templates directly from Eclipse Che IDE.

A practical example. For the purposes of this booklet, we will present simple functionality of TPS CLI and GUI. For other options please, visit our documentation pages.

The user first needs to install the TPS CLI from PyPI¹⁷ and then the *setup* command will guide him through the process of configuring the necessary TPS API and IAM endpoints. After that the user can initiate the *template* CLI command to view the available CLI actions for templates. Templates can be uploaded with *template save* and downloaded with *template get* command. The *template create* command generates all directories and files that are needed to upload a Template library model. After generating a model (TOSCA template) or

¹⁷ <https://pypi.org/project/xopera-template-library/>

a blueprint (TOSCA CSAR) user can add in his own code. He will be asked for the template's name and (TOSCA) type such as capability, requirement, relationship, node, etc. To create and upload a new template to the TPS, the user needs to provide a unique name of template, template's privacy and semantic version of the template. Here's a simple example of the CLI command:

```
# save a private CSAR to Template library (with README)
$ xopera-template-library template save --name TestCSAR --path examples/csar -
-readme examples/README.md --version 0.0.1
Description of the template: Testing TPS CLI TOSCA CSAR upload
Template has been added successfully!
Version insertion was successful!
```

After uploading a new template, the user can preview and download the files from the CLI, or even better, he can open the web browser and navigate to the public TPS GUI¹⁸, where he will see a list of publicly available templates. These can be filtered to find the right one and display all its versions and README and can be then downloaded with a single click.

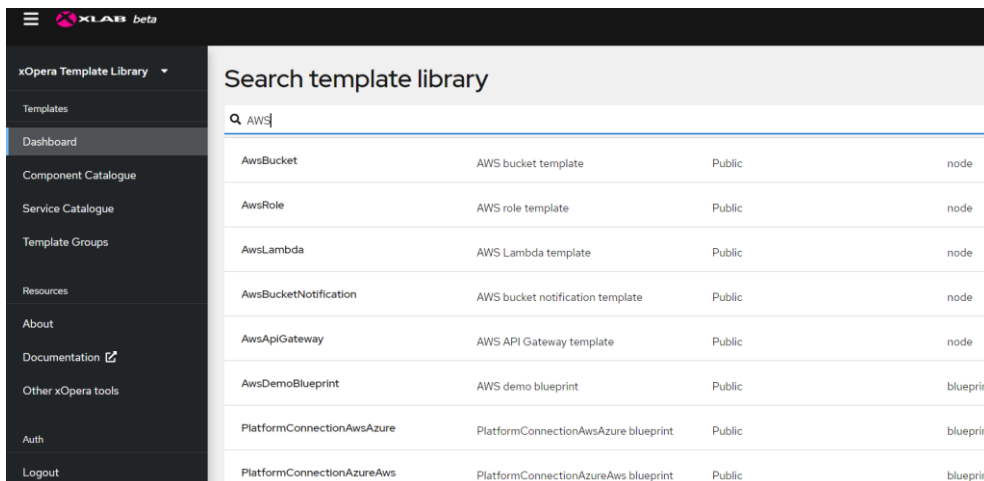


Figure 3.11 - The main template library GUI screen with the list of stored templates

Open challenges. Currently the CLI version provides all features, while GUI and Eclipse Che plugin still lack some functions, which will be extended and improved in the future.

Getting started. A detailed description of the use is described in [D5.3](#) and [D5.4](#) or in the online documentation¹⁹.

¹⁸ <https://template-library-xopera.xlab.si/>

¹⁹ <https://template-library-xopera.xlab.si/docs/>

3.9 Monitoring Tool

Overview. The Monitoring Tool (MT) provides all functionality for defining all necessary infrastructure to monitor the efficiency and performance of the applications. In addition to that, the Monitoring Tool triggers a mechanism that generates Grafana User personalized dashboards and Alarm events based on defined Policies. The Monitoring Tool integrates with other RADON tools (CTT, DT, Orchestrator), but can also be used as a standalone tool.


Business Purpose. Monitoring the runtime behavior of a serverless application implies that either a cloud platform native monitoring service (AWS CloudWatch or GCP Cloud monitoring) is used or a cloud platform agnostic solution is applied. The latter is promoted based on the fact that the RADON framework relies on TOSCA to model FaaS implementations on multiple cloud FaaS runtimes. It is built for DevOps engineers, developers and IT managers.

How the tool works. The Monitoring tool is based on the Prometheus open source monitoring framework and relies on the Prometheus PushGateway component that allows ephemeral and batch jobs to expose their metrics on a Prometheus server. Moreover, the collected metrics from FaaS instances are exposed as visualization dashboards on a Grafana instance. A user can attach the Monitoring stack on a TOSCA service template and link it to the FaaS instances that he/she wants to get monitoring insights on while runtime execution. The linking is implemented through TOSCA relationships that targets the FaaS runtime execution environment on both the AWS and the GCP platforms (monitoring TOSCA relationships: `awsIsMonitoredBy` and `gcplsMonitoredBy`). On every FaaS invocation the metrics are pushed to the PushGateway component. The Prometheus server periodically scrapes the PushGateway endpoint to gather any metric data.

Furthermore, the aforementioned TOSCA relationships can be used to set up the generation of Alerts through the Grafana API whenever rules about metric-constraints are violated. The alerts can be forwarded towards the xOpera SaaS. In combination with a policy defined on TOSCA node level the generated alerts can be handled by xOpera to trigger resource scaling actions.

In a cluster, the Monitoring tool can be used to monitor, alert and trigger the scale of resources.

A practical example.



(e.g. EC2 instances) Moreover, code injection is required if one wishes to handle the exposed metrics in a fully automated way.

Getting started.

A detailed description of the Monitoring Tool is provided in deliverables [D5.1](#) and [D5.2](#). Moreover, the Grafana Dashboards API can be located in the following github [repo](#).

3.10 Function Hub

Overview. The Function Hub supports storing of versioned, ‘plug-and-play’ FaaS packages. It has been designed as an integral part of Pragma’s use case, Cloudstash.io - a serverless package manager. The function hub has been integrated with RADON through GMT at the design phase.

When creating FaaS objects, the user can select any available URL from Function Hub. These can be found by browsing the web app.

Business Purpose. Function Hub offers a secure space for organizations and individuals to store and maintain their artifacts all aggregated in a versioned manner. Essentially, any actor in need of managing and provisioning artifacts such as source code files, dependency packages, etc is a potential user of Function Hub. The content is easily accessible and shareable to any persons of interest providing access to multiple users across big organizations, making sure to use the same versions uniformly. Further, making use of the versioning system helps to keep the organization's codebase neat and robust.

How the tool works. Function Hub is an integral part of Cloudstash.io, a serverless package manager. It offers a web application UI where the user can browse and interact with the application's environment but also offers a PyPI CLI package with which the user can interact with the application programmatically. The backend is exposed with a RESTful HTTP API which can be accessed either from the web UI or the PyPI package for storing and accessing reusable functions. The system offers free access to publicly uploaded functions but requires user authentication to access privately stored data. Integration with RADON is taking place in the modeling phase using GMT. There the user can use Function Hub URL annotations as references to stored functions resulting in the creation of simple and lightweight TOSCA service templates.

A practical example. Access the UI interface of the application in cloudstash.io and create a user. Then create a repository and follow the instructions in the quick start guide that you can find in the comprehensive documentation of the tool in <https://functionhub-cli.readthedocs.io/en/latest>. After having successfully uploaded the artifact of your choice using the pip package of Function Hub found in <https://pypi.org/project/functionhub>, you can browse the UI and locate the files you have just uploaded in the repository of your choice. In figure 3.13 you can see an example of an uploaded function named “create”. The description panel of the function provides various information about the artifact, while the entry “artifactID” can be used as a reference to that particular item.

Figure 3.14 depicts the artifact attachment window of a Lambda function where we use a Function Hub URL as reference for the source code the Lambda function will execute. As you can see the URL consists of a simple path containing the artifact ID and it triggers an API call to locate and download the desired item.

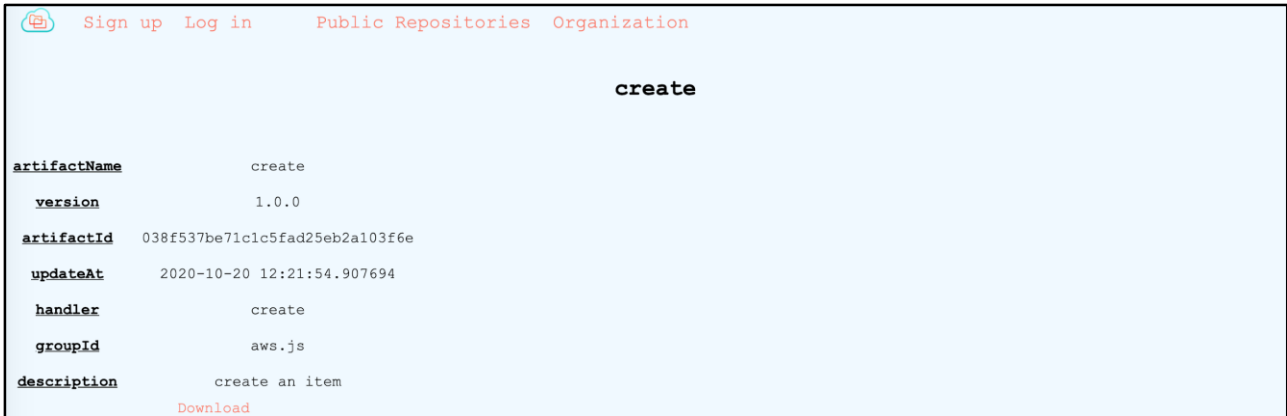


Figure 3.13 - Detailed information on the “create” function

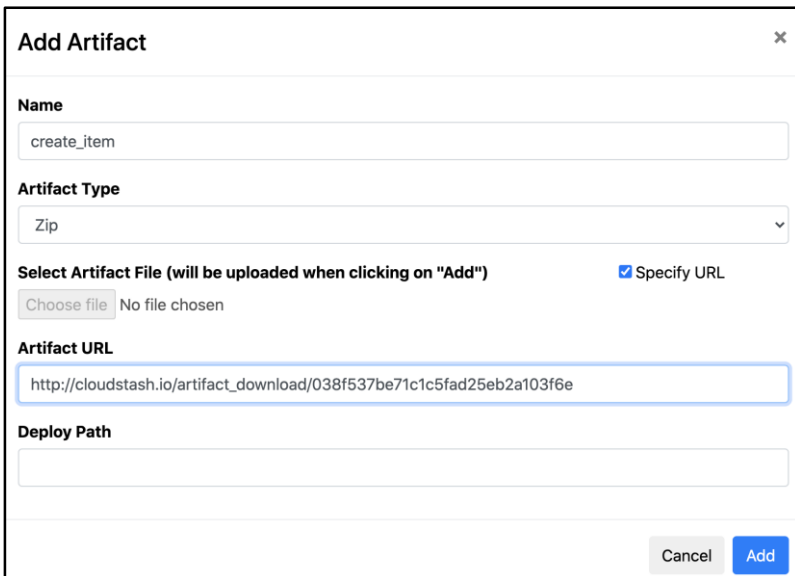



Figure 3.14 - Artifact attachment as Function Hub URL

Open challenges. Function Hub can currently be used using its authentication system that adds another layer of user management to the RADON end-user. This could be avoided in the future by integrating “keycloak” as a method of authentication for RADON users as they



should already have an account there with their subscription to the IDE. A limitation that Function Hub has so far is the allowance of uploading and hosting artifacts of maximum 10MB in size. This is a decision that has been made so that the service is offered for free to users, but a subscription-like schema could be set up in the future so that we offer the possibility of storing bigger artifacts to individual users and enterprises.

Getting started. A detailed description of Function Hub, with its content and functionality is found in deliverables [D5.3](#) and [D5.4](#). A user guide including a quick start can be found at <https://functionhub-cli.readthedocs.io/en/latest> while step-by-step tutorial can be found at <https://github.com/radon-h2020/RADON-workshop/blob/main/labs/functionhub.md>

A PyPI package is available at <https://pypi.org/project/functionhub> while issues and pull requests can be requested straight from our Github repository at: <https://github.com/radon-h2020/radon-functionhub-client>

3.11 CI/CD Plugin

Overview. The CI/CD plugin integrated into the IDE provides all necessary connections and functionality to enable CI/CD in the application development process. The plugin is available for Jenkins, the RADON officially supported CI/CD platform but can be extended to other platforms.

Business Purpose. Continuous Delivery is an essential part of RADON which is translated to rapid, incremental changes to Serverless Applications. Doing so, demands a specific focus on automation. For Continuous Integration, a user should be able to automate the necessary tollgates for quality validation and branch integration, while for Continuous Deployment, a user might want to automate the necessary release criterias and create a fail-safe environment. Getting those two elements combined in a single project by enabling CI/CD in a project, we can reduce lead time and bring value to the end user.

How the tool works. Technically, this is achieved by utilizing the full framework of Radon. Within the framework you find all the necessary components for quality assurance, deployment, testing and monitoring. Including these tools in a CI/CD pipeline will assure automation and increase development velocity.

The CI/CD plugin that is integrated in the IDE triggers a pre-configured CI/CD job in the user's Jenkins environment and there, the selected RADON tools will be automatically executed in a robust and secure environment ensuring that all new contributions deriving from the application development team won't break the system at the production deployment time. Feedback will be provided back to the user in Jenkins so that potential issues can be further investigated for troubleshooting, while a thorough history backup of the various builds will provide confidence and roll-back possibility.

A practical example. To enable CI/CD a user should invoke a pre-configured Jenkins job through the IDE plugin. To do so, must choose "Configure CI" on any CSAR file and after providing all the necessary configuration in the YAML configuration file you can see in figure 3.15, the job can be triggered by choosing "Trigger CI" in the IDE contextual menu.


```

Eclipse Che  CI_config_ServerlessToDoListAPITestingExample.yaml x
1  {
2  "CSAR_name": "",
3  "CSAR_version": "",
4  "Jenkins_URL": "",
5  "Jenkins_username": "",
6  "Jenkins_password": "",
7  "Jenkins_job": "",
8  "Jenkins_job_token": "",
9  "cookie_jar": "/tmp/cookies"
10 }

```

Figure 3.15 - YAML CI configuration file

After that, a remote agent in Jenkins will perform all the specified jobs defined in the pre-configured pipeline. Examples of pipelines for the RADON tools can be found in <https://github.com/radon-h2020/radon-cicd-templates>. In Figure 3.16, you can see an example of the sequential execution of the VT, the DPT, the CTT, the TL and finally the deployment of an application with xOpera.

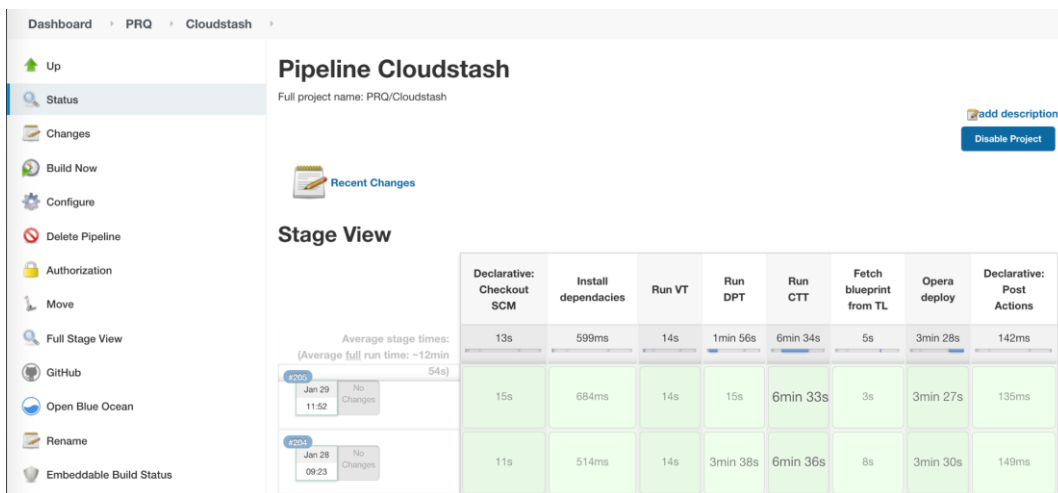



Figure 3.16 - Jenkins pipeline execution

Open challenges. The CI/CD system is fully customizable but currently the IDE supports only invocation of CI/CD projects configured in the Jenkins environment.



Getting started. A detailed description of the CI/CD plugin is provided in deliverables [D5.1](#) and [D5.2](#). CI/CD templates²⁰ for Jenkins and CircleCI and step-by-step tutorials²¹ can be found online.

²⁰ <https://github.com/radon-h2020/radon-cicd-templates>

²¹ <https://github.com/radon-h2020/RADON-workshop/blob/main/labs/cicd.md>

3.12 Data Pipeline Plugin

Overview. TOSCA service blueprint with data pipeline-based nodes may need to be updated at runtime to ensure consistency and may consist of various user-made errors (e.g., incorrect relationships among the data pipeline nodes, erroneous configuration of nodes for encryption). Therefore, we designed and developed the data pipeline plugin to allow users to work with data pipeline-based TOSCA service blueprint.

Business Purpose. Designing data pipelines using Topology and Orchestration Specification for Cloud Applications (TOSCA) standard language enables the ability to easily compose data driven applications from independently deployable, schedulable and scalable pipeline tasks, such as microservices, serverless functions or self-contained applications.

The aim is to provide standards based methodology and tools for controlling the life-cycle of such composable data pipelines in a DevOps manner and to enable companies to move from monolithic data management applications to freely reusable, composable, and scalable data pipeline services.

How the tool works. This consortium has designed and developed a set of TOSCA based pipeline node types available in radon particles repository. RADON data pipeline provides an environment for building serverless data-intensive applications and handling data movement between different clouds efficiently. The TOSCA based data pipeline service template can be generated using the RADON Graphical Modelling Tool (GMT), Winery. The service template developed using those data pipeline nodes is forwarded to the data pipeline plugin, making sure that the user-designed service template is workable and the pipelines can be deployed in the required cloud or local environment. The pipeline plugin can be invoked through a command-line interface or a REST-based interface.

Upon receiving the service template in CSAR format, the data pipeline plugin unzips the CSAR file and finds the service blueprint, which is in .tosca format. The plugin then parses the .tosca file and understands the node topology, makes any changes/modification to the .tosca file itself, if needed. For instance, the plugin will fix the wrong relationship types among pipelines in the service template. After updating the templates, the plugin Zip all again and create the CSAR file. The modified CSAR can now be passed to the RADON Orchestrator.

A practical example. The TOSCA based data pipeline service template can be generated using the RADON Graphical Modelling Tool (GMT), Winery. You can follow this step to set up RADON GMT. Follow our [user guide](#) to create a new service template. Open the Winery window from the RADON IDE and click on the *Service Templates* manu. Create a new service

template by clicking on the *Add New* button. Now provide a suitable name and click on the *Add* button. Here you can see the list of service templates. Now select and open the newly created service template. Select the *Topology Template* menu item followed by the *Open Editor* button. In the Winery: *topology modeler* window, find the suitable data pipeline TOSCA nodes, as shown in below figure. Drag the required TOSCA nodes from the palette area, as shown in Figure 3.17(a), and set the properties and make the connection with other pipeline nodes. The service template can now be exported in CSAR format.

The exported CSAR now can be sent to the data pipeline plugin. To invoke the data pipeline plugin with exported CSAR, right click on the csar and select “Convert CSAR with Data pipeline plugin” option, as shown in Figure 3.17(b). The converted csar will be exported to the same folder structure.

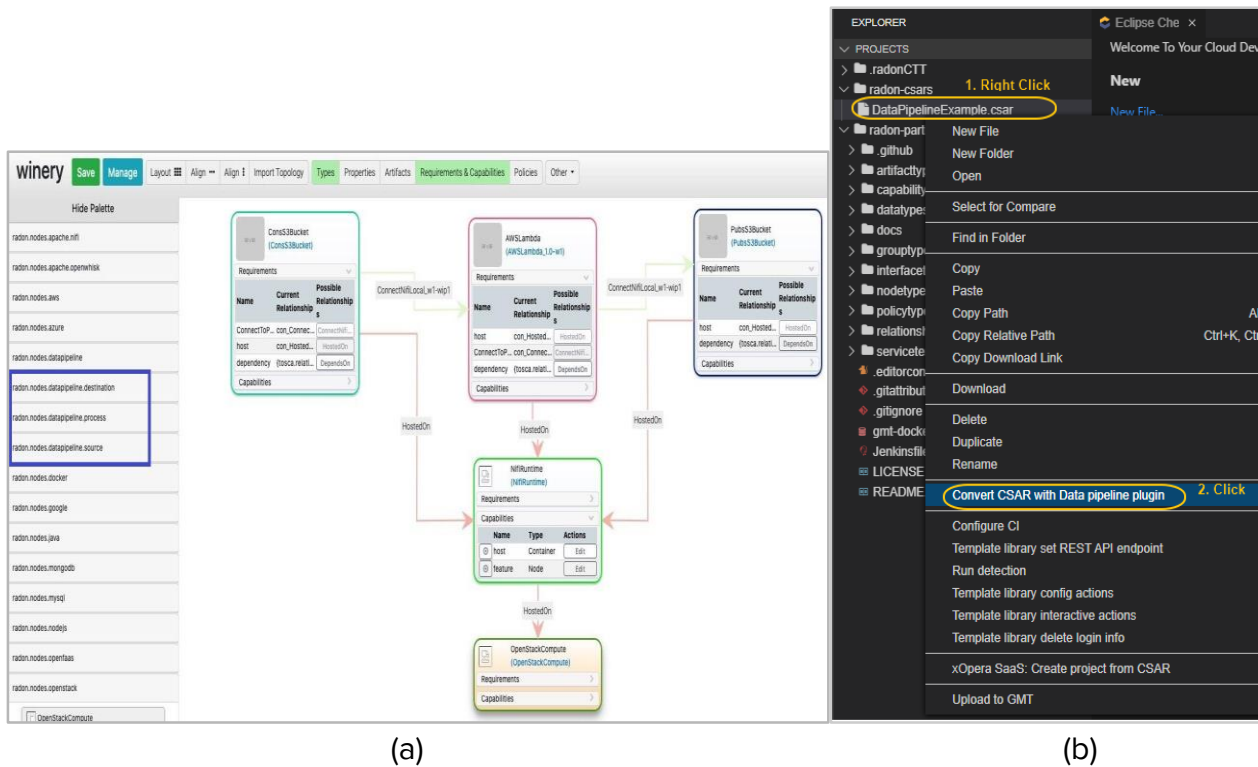



Figure 3.17 - Designing and verifying TOSCA based data pipeline service template

Open challenges. RADON data pipeline is built atop one open source (Apache Nifi) and one commercial data management solution (AWS DP) and currently there is no support for the other open-source and commercial solutions.



Getting started. A detailed description of the data pipeline plugin can be found in deliverables [D5.5](#) and [D5.6](#). The detailed description of the data pipeline can be found [here](#). The GitHub repo for data pipeline plugin is available [here](#). The developed TOSCA based pipeline node types are available in [radon particles](#) repository. Additionally, one can refer to the [RADON workshop](#) repo and the [RADON data pipeline webinar](#) for practical use cases.

4. Industrial Use Cases

4.1 Travel Technology

The problem.

The ATC use case refers to the travel and tourism industry, and it aims to provide a testbed for validating the concepts and the tools that RADON is delivering to a number of stakeholders being involved in the development of FaaS-enabled serverless computing applications. This use case is based on the commercial offering of ATC in the travel and tourism industry, which is called Viarota²², and it delivers a mobile application that assists tourists in a destination to automatically or manually set up personalized city break tours and runs algorithms in the backend for matching available places to visit with tourist preferences. Eventually, the tourists get place descriptions and ratings, based on a workflow analysing experiences for past visitors, in which the Viarota platform integrates the information collected from various sources, such as travel blogs and social media, and uses AI to extract experiences lived in specific places.

Whilst validating the benefits of the DevOps framework proposed by the RADON project, ATC applied the principles of the RADON methodology to partially refactor the initial monolithic architecture (containing potential bottlenecks) of the Viarota app. Also ATC achieved to enhance the functionality of the app, by introducing a multi-cloud ML pipeline related to providing qualitative information on the Points of Interest (POIs) included in a user's personalized tour planning. This type of information derives from content aggregated from the Twitter API (focused crawling) and contextual mining on text coming from the Twitter streams, allows the Viarota user to better understand the social sentiment around the places he/she intends to visit. Finally, the hate speech detection FaaS component addressed the need for moderation capabilities on the incoming content.

How RADON addresses it.

The RADON framework helped the Viarota application to modernize its architecture by introducing an event-driven ML/AI data pipeline. The Viarota DevOps team managed to model a cross cloud pipeline consisting of ML/AI functions hosted on different FaaS runtimes, namely the AWS, GCP and Azure platforms.

²² <https://viarota.com>

The RADON framework supports the modelling of serverless components through the GMT that exposes the various node definitions. The GMT allows to model not only the FaaS components but also events linked to storage components. The model can be exported in YAML format and can be executed by the TOSCA compliant runtime, the xOpera orchestrator. The Viarota model is shown in Figure 4.2.

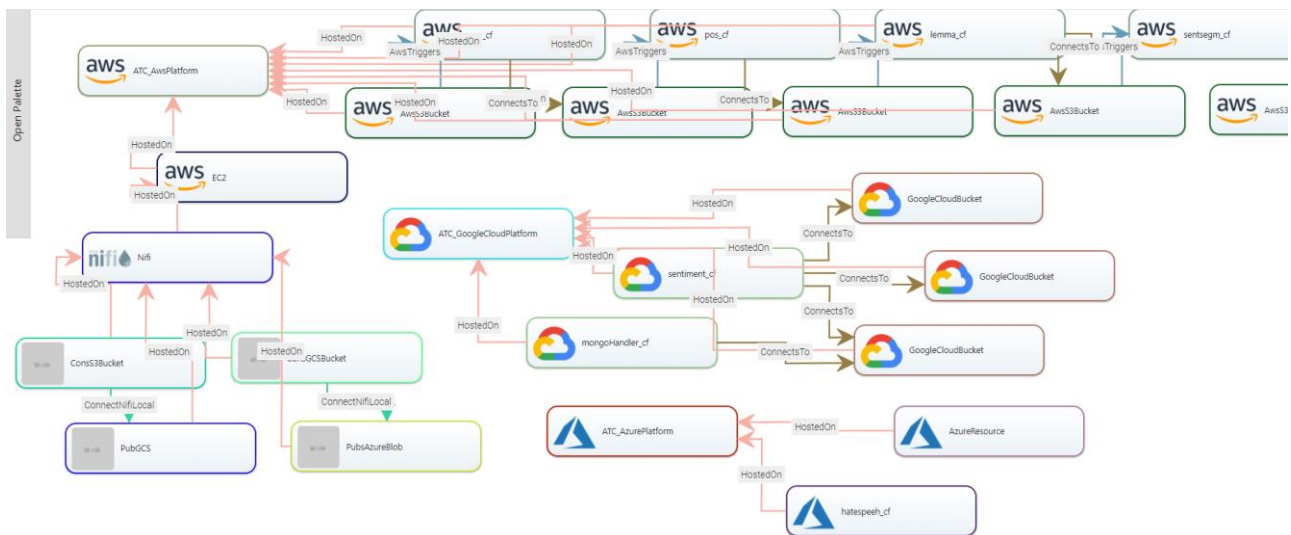



Figure 4.2 - Viarota model

The need to model, verify and if possible correct non functional requirements like the instance type of the AWS EC2 machines used in the Viarota topology, is addressed by the verification workflow. The ability to identify quality related inconsistencies at design time contributes to the quality of the Viarota software. Moreover, by configuring a decomposition policy the Viarota dev team is able to find the optimal deployment configuration for functions that can be potential bottlenecks. The Decomposition tool suggestions refer to either the FaaS memory settings or the concurrency level. Since the provisioned concurrency level is a paid cloud resource it is important to configure a proper value so as to avoid excessive costs.

The data flows implemented in the social feedback analysis feature are integrated using the Data Pipelines components. Specifically, a data link between an AWS S3 storage bucket and a GCP cloud storage and another between a GCP cloud storage and an Azure blob storage are modelled and implemented. Although the self scaling nature of the public cloud serverless components makes the Viarota topology elastic and contributes to the total performance, there is still the need for auto scaling capabilities. This is addressed in the context of RADON by the joint execution of the Monitoring and the orchestrator tools. A scale



up policy is configured to constantly monitor the functions memory consumption and when a threshold is reached an event notification is sent to the orchestrator triggering a redeployment with updated configuration.

Since the Viarota consumes events from social networks like Twitter, burst events are expected to happen unexpectedly. The RADON framework offers a way to perform load testing to simulate such an event and check how the Viarota topology behaves and how the total performance/response time is affected. Finally, the CI/CD tool allows to automate the execution of the various RADON tools as a build and orchestration template.

Lessons learned from using RADON.

The pay as you go pricing model that the public cloud environments offer seems quite appealing in situations when you need to deliver a solution without managing the underlying infrastructure, as it is the case in the Viarota application. Nevertheless, this implies that the Varota dev team is familiar with the particularities and complexities of the public cloud FaaS runtimes. The RADON framework fills this gap by hiding away most of the vendor specific particularities, contributing this way in vendor lock in avoidance, simplified development processes and increased Quality of product. Also, a common backend is shared among all customers to provide for common Database and application components, whilst at the same time functions in FaaS architecture are deployed to serve each customer's specific requirements and customer specific changes can be applied only to lambdas, leading to reduced operating costs. Finally, Time-To-Market is significantly reduced in all types of development whether new products are conceptualized and developed, or existing ideas are upscaled to lead them to the market.

4.2 Ambient Assisted Living

The problem.

Because of their event driven nature applications in the domain of IoT are indicated among the ones that can greatly benefit from the adoption of the FaaS programming model. The RADON Assisted Living use case aimed to evaluate the capacity of the RADON tools to deal with development of a solution in the domain of Ambient Assisted Living and involving an event-driven environment consisting of robotic and Internet-of-Things (IoT) devices.

SARA, this is the name of the solution, provides health monitoring and (socially interactive) assistance in daily living tasks to the elderly (and their caregivers) at home, in order to prolong autonomy and delay institutionalization of elderly. The SARA system is designed to provide assistance to the elderly (and their caregivers) through Assistive Tasks (AT) falling in four areas of intervention: physical decline prevention and therapy, cognitive decline prevention, health management, and psychological needs.

In order to provide the aforementioned functionalities the SARA system coordinates the following computing nodes: a Smart Phone that acts as hub of a Body Area Network (BAN) of wearables (e.g. sensors and identification tags carried or worn on the patient's person) for fall detection, fall risk assessment and other mobility related data; a Robotic Rollator (RR) providing physical support for the patient and offering patient monitoring and autonomous navigation capabilities; a Robotic Assistant (RA) connected to a network of embedded devices and services for monitoring the patient's activities, health status and for supporting the notification/reminder of upcoming treatments (e.g. medication, training schedules); a Smart Environment Gateway acting as a hub of smart devices embedded in the local physical environment in support of patient monitoring and rollator navigation functions.

The SARA solution is built on top of the ENG CloE-IoT platform, which aims to simplify the integration of highly distributed, complex and robust IoT solutions exploiting computational resources both in the cloud and at the edge. The CloE-IoT platform offers a set of functionalities specifically targeting common IoT requirements allowing developers to focus on their domain specific requirements.

The objective of the Assisted Living use case was the development of a FaaS-based implementation of the SARA solution starting from an existing prototype built using a mix of traditional and microservices approach. The FaaS-based implementation of S-SARA was codenamed S-SARA. The target FaaS platform selected for hosting the SARA functions was OpenFaaS.

Using the RADON Constraints Description Language (CDL) it was possible to formalize the SARA security and privacy requirements. This formalization enabled the use of the RADON Verification Tool for the verification of the consistency of the requirements against the TOSCA models of S-SARA. The RADON Defect Prediction Tool (DPT) was used to continuously monitor the quality of the Ansible scripts developed to automate the configuration of ROS (Robotic Operating System) services. In fact, the SARA solution relies on the services offered by ROS for implementation of some of the functionalities provided by the robotic components of SARA (i.e. the Robotic Rollator and the Robotic Assistant).

iv) Packaging and deployment. The description of the artifacts that need to be installed and configured along with their dependencies were packaged in a collection of CSAR files suitable to be processed by the RADON TOSCA Orchestrator (xOpera). The Continuous Testing tool (CTT) was used for executing JMeter load tests of serverless versions of CloE-IoT. For the definition of the JMeter Load test policies we reuse the test scripts (*.jmx) already developed for CloE-IoT.

Lessons learned from using RADON.

The RADON Orchestrator reduced the operational costs of the S-SARA solution by means of the automation of the deployment process enabled by. The GMT reduced the technical implementation overhead in terms of both (a) simplified management of the execution environment brought in by the adoption of the FaaS model and (b) reduced effort for the development and maintenance of deployment bundles. The RADON Decomposition Tool (DT) simplified the development process by avoiding starting from scratch the refactoring of SARA in terms of serverless functions. The Verification tool (VT) improved the quality of S-SARA verifying the consistency between the constraints and the S-SARA application structure defined by means of the GMT. The DPT contributed to the overall quality of the product as well by predicting possible defects within the Ansible scripts for the configuration of the Robotic Operating Systems nodes.

4.3 Artifact Management

The problem.

The EFI/PRQ UC refers to building and offering an artifact management system based on the serverless technology. This artifact management system addresses complexity and reliability issues by centralizing your artifacts in a single location. As a result the end-user gains more control over your artifacts and how they are used, while the artifact repository itself acts as a single source of truth and CI/CD integration point for your artifacts.

Existing players in the artifact management market tend to offer costly and unsustainable packages especially for small and medium-sized enterprises and so EFI/PRQ sees clear business opportunity offering a serverless, scalable and affordable solution.

Within RADON, the UC acts as a validation story of using RADON framework to entirely create such an artifact manager from scratch. Focus has been given mostly to modeling and orchestrating the application onto the cloud vendor rather than developing the logic of the application as RADON's offering strongly highlights the benefits of the framework from a DevOps perspective.

The key factor to our product's success is the serverless nature of the application. Serverless refers to an event-driven application design and deployment model that automates back-end resource provisioning and enables developers to focus on the application development rather than the hosting infrastructure. It provides various advantages over traditional server-centric or cloud-based infrastructure.

The product offers developers with greater scalability, quick time to release, more flexibility and all this at a reduced cost as the user pays only for the services used. A serverless function is a programmatic function written by a software developer for a single purpose. It's then hosted and maintained on infrastructure by cloud computing companies. These companies take care of code maintenance and execution so that developers can deploy new code faster and easier. In our UC we chose Amazon Web Services to host our application but in general all cloud services providers offer similar functionalities.

Creating "Cloudstash" as a serverless application we can achieve building a self-maintained and self-funded application with the possibility of scaling it up or down painlessly according to demand.

How RADON addresses it.

RADON's most valuable asset is that the framework provides every tool and plugin a developer would need to complete a robust and reliable development workflow following DevOps practices.

For our UC development, a big advantage was the ability to model our application's topology graphically and export the orchestration templates automatically into YAML executable files. That way it's easier and more user-friendly to develop and maintain big and complex topologies integrating multiple resource nodes. Combined with the orchestrator's ability to take care of the resources deployment and put the application into the runtime environment effortlessly, it already delivers value to the development lifecycle. On top of those, RADON framework offers a couple more tools that can add value to the development workflow offering different functionalities. In EFI/PRQ UC we used plenty of them listed below:

- IDE - Integrated Development Environment
- GMT - Graphical modeling Tool
- CTT - Continuous Testing Tool
- VT - Verification Tool
- DPT - Defect Prediction Tool
- Function Hub - Functions repository
- xOpera - Orchestrator
- MonT - Monitoring Tool
- CI/CD - Continuous Integration/Continuous Deployment

All the above tools contributed to various aspects related to application quality, availability, adaptability, and led us to the final version of Cloudstash's model presented in Figure 4.4.

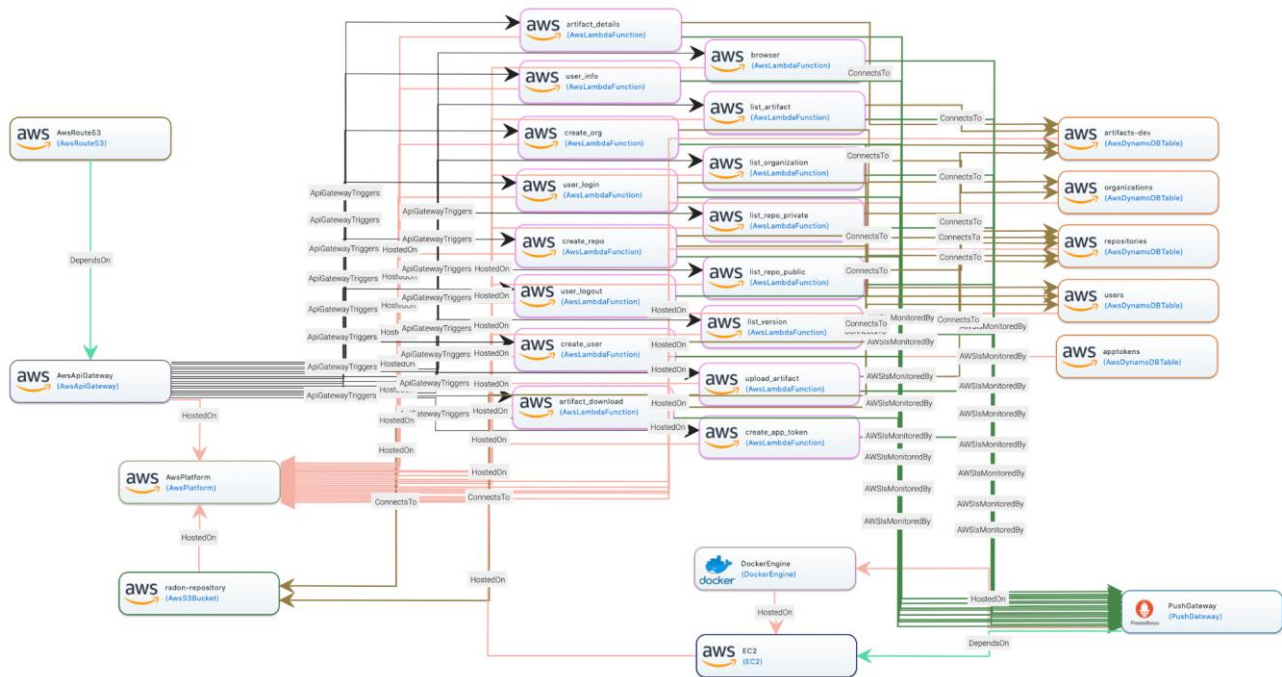


Figure 4.4 - Cloudstash model

Lessons learned from using RADON.

RADON framework was used as the main development platform and brought significant benefits to our development and operation processes. Acting as a single portal of development we managed to broaden our horizons in QA, testing, and maintenance. We realized the importance of having a common workspace for all development operations as we spent less time exploring solutions but instead we used RADON's in-house offerings. All together added up to a more simplified development process compared to our previous practices.

Having chosen RADON to "go serverless" we saw clear benefit from operational costs reduction for the development and the maintenance of our application. Even though there will be charges for using RADON as a final product, a lot of operational expenses can be saved using the RADON framework instead of subscribing to separate solutions. Moreover, being a DevOps orientated framework encourages an incremental cycle of development and significantly reduces technical implementation overhead. In our research to identify concrete benefits compared to a similar tool in the market, we spotted improvements in time-to-market as less time and effort was needed to build Cloudstash using the "RADON framework" rather than using "Serverless framework" and additional testing tools.



5. Conclusion

In this booklet, we have presented a summary of the RADON methodology and its underpinning tools. A set of industrial use cases as well as focused examples of application of the tools have been developed throughout. The interested reader can find additional getting started information about the RADON framework in a walk-through demonstrator available at:

<https://github.com/radon-h2020/RADON-Demonstrator>

A list of tools and detailed videos that illustrate the RADON methodology are available at the following resource:

<https://github.com/radon-h2020/radon-methodology>

Acknowledgement

We gratefully acknowledge the European Commission under the Horizon 2020 Funding Program for supporting this booklet under grant agreement H2020-ICT-2018-2-825040.

References

- [D1.2] RADON Consortium, Deliverable D1.2 - Period Report II, 2020
- [[D2.1](#)] RADON Consortium, Deliverable D2.1 - Initial Requirements and Baselines, 2019
- [[D2.2](#)] RADON Consortium, Deliverable D2.2 - Final Requirements, 2020.
- [[D2.4](#)] RADON Consortium, Deliverable D2.4 - Architecture & Integration Plan II, 2020.
- [D2.7] RADON Consortium, Deliverable D2.7 - Integrated Framework II, 2021.
- [D3.1] RADON Consortium, Deliverable D3.1 - RADON methodology, 2021.
- [[D3.2](#)] RADON Consortium, Deliverable D3.2 - Decomposition Tool I, 2019.
- [D3.3] RADON Consortium, Deliverable D3.3 - Decomposition Tool II, 2020.
- [[D3.4](#)] RADON Consortium, Deliverable D3.4 - Continuous Testing Tool I, 2020.
- [D3.5] RADON Consortium, Deliverable D3.5 - Continuous Testing Tool II, 2021.
- [[D3.6](#)] RADON Consortium, Deliverable D3.6 - Defect Prediction Tool I, 2020.
- [D3.7] RADON Consortium, Deliverable D3.7 - Defect Prediction Tool II, 2021.
- [[D4.1](#)] RADON Consortium, Deliverable D4.1 - Constraint Definition Language I, 2019.
- [D4.2] RADON Consortium, Deliverable D4.2 - Constraint Definition Language II, 2020.
- [[D4.4](#)] RADON Consortium, Deliverable D4.4 - RADON Models II, 2020.
- [[D4.5](#)] RADON Consortium, Deliverable D4.6 - Graphical Modelling Tool I, 2019.
- [D4.6] RADON Consortium, Deliverable D4.6 - Graphical Modelling Tool II, 2020.
- [[D5.1](#)] RADON Consortium, Deliverable D5.1 - Runtime Environment I, 2019.
- [D5.2] RADON Consortium, Deliverable D5.2 - Runtime Environment II, 2020.
- [[D5.3](#)] RADON Consortium, Deliverable D5.3 - Technology Library I, 2020.
- [D5.4] RADON Consortium, Deliverable D5.4 - Technology Library II, 2021.
- [[D5.5](#)] RADON Consortium, Deliverable D5.5 - Data Pipeline Orchestration I, 2019.
- [D5.6] RADON Consortium, Deliverable D5.6 - Data Pipeline Orchestration II, 2020.
- [D6.5] RADON Consortium, Deliverable D6.5 - Final Assessment Report, 2021.

[Baldini2017] Baldini I, Castro P, Chang K, Cheng P, Fink S, Ishakian V, Mitchell N, Muthusamy V, Rabbah R, Slominski A, Suter P. Serverless computing: Current trends and open problems. In *Research Advances in Cloud Computing 2017* (pp. 1-20). Springer, Singapore.

[Brinkkemper1996] Brinkkemper, S. (1996). Method engineering: engineering of information systems development methods and tools. *Information and software technology*, 38(4), 275-280.

[Eyck2017] Van Eyk E, Iosup A, Seif S, Thömmes M. The SPEC cloud group's research vision on FaaS and serverless architectures. In *Proceedings of the 2nd International Workshop on Serverless Computing 2017* Dec 11 (pp. 1-4).

[Hendrickson2016] Hendrickson S, Sturdevant S, Harter T, Venkataramani V, Arpaci-Dusseau AC, Arpaci-Dusseau RH. Serverless computation with openlambda. In *8th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 16) 2016*.

[McGrath2016] McGrath G, Short J, Ennis S, Judson B, Brenner P. Cloud event programming paradigms: Applications and analysis. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD) 2016* Jun 27 (pp. 400-406). IEEE.

[Soldani2018] Soldani J, Tamburri DA, Van Den Heuvel WJ. The pains and gains of microservices: A Systematic grey literature review. *Journal of Systems and Software*. 2018 Dec 1;146:215-32.