CTT: Load Test Automation for TOSCA-based Cloud Applications

Thomas F. Düllmann
University of Stuttgart
Germany

André van Hoorn
University of Hamburg
Germany

Vladimir Yussupov
University of Stuttgart
Germany

Pelle Jakovits
University of Tartu
Estonia

Mainak Adhikari
Indian Institute of Information Technology Lucknow
India

ABSTRACT

Despite today's fast and rapid modeling and deployment capabilities to meet customer requirements in an agile manner, testing is still of utmost importance to avoid outages, unsatisfied customers, and performance problems. To tackle such issues, (load) testing is one of several approaches. In this paper, we introduce the Continuous Testing Tool (CTT), which enables the modeling of tests and test infrastructures along with the cloud system under test, as well as deploying and executing (load) tests against a fully deployed system in an automated manner. CTT employs the OASIS TOSCA Standard to enable end-to-end support for continuous testing of cloud-based applications. We demonstrate CTT's workflow, its architecture, as well as its application to DevOps-oriented load testing and load testing of data pipelines.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Software and its engineering → Software testing and debugging**; *System modeling languages*; • **General and reference** → *Performance.*

KEYWORDS

continuous testing, continuous integration, TOSCA, agile, DevOps

ACM Reference Format:
Thomas F. Düllmann, André van Hoorn, Vladimir Yussupov, Pelle Jakovits, and Mainak Adhikari. 2022. CTT: Load Test Automation for TOSCA-based Cloud Applications. In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering (ICPE '22 Companion), April 9–13, 2022, Bejing, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3491204.3527484

1 INTRODUCTION

Testing is the prevalent quality assurance technique in practice. Being an essential part of the software development lifecycle, testing plays one of the key roles in the RADON framework [1, 2], which provides a model-based software development ecosystem for cloud applications. RADON bundles a set of tools for application modeling, deployment, quality assurance, and operations that utilize the Topology and Orchestration Specification for Cloud Applications (TOSCA) standard by OASIS [3]. For quality assurance, RADON includes the continuous testing workflow [4] that particularly aims to support software developers, QoS engineers, and release managers in producing high-quality applications. The core component implementing the continuous testing workflow is the Continuous Testing Tool (CTT), which provides the functionalities for defining, generating, and executing continuous tests of application functions, data pipelines [5], and microservices, as well as for reporting test results. These results provide further information about the outcome of the tests. CTT employs the TOSCA standard to extend RADON by end-to-end support for continuous testing of microservice-based (including serverless [6] components hosted using Function-as-a-Service (FaaS) offerings) and data pipeline applications in DevOps [7].

CTT is the first tool of its kind that supports the whole workflow — from test specification over execution and reporting — that is also extensible to custom needs, e.g., integration of other types of tests or tools. CTT and its respective TOSCA types allow the specification of tests within the model. During the modeling phase, tests can be attached to the respective applications, providing developers a good overview and self-contained artifacts.

This paper presents a concise summary of CTT and selected application scenarios. In particular, this paper (i) outlines the CTT tool's architecture and integration into the RADON workflow and tool ecosystem, (ii) presents the tool's extension points, (iii) and shows selected applications of CTT to DevOps-oriented FaaS/microservices load testing and data pipeline load testing. This paper is based on the cited technical deliverables (particularly, [8] and [9]), to which we refer for further reference.

2 BACKGROUND: TOSCA AND RADON

TOSCA [3] is a YAML-based standard by OASIS for describing cloud application deployments in a vendor- and technology-agnostic manner. TOSCA enables modelers to specify so-called Service Templates that describe desired application topologies — i.e., components and their relationships — of an application to be deployed to a cloud infrastructure. The core modeling constructs in TOSCA are Node and Relationship Types: the former describe distinct component types such as "Relational Database" or "Web Server", whereas the latter represent types of relations among them, e.g., "Hosted On" or "Connects To". Each Node or Relationship Type can be instantiated in Service Templates in a form of Node or Relationship Templates to

ICPE '22 Companion, April 9–13, 2022, Bejing, China

ACM ISBN 978-1-4503-9159-7/22/04…$15.00
https://doi.org/10.1145/3491204.3527484

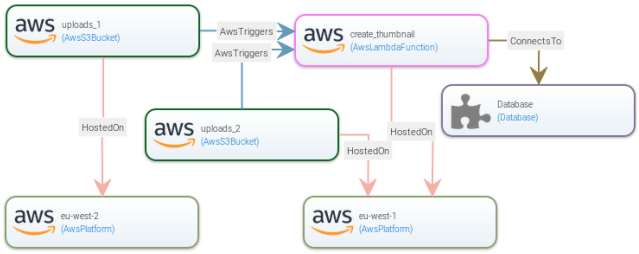


Figure 1: TOSCA model with RADON extensions in GMT

represent concrete application component instances, e.g., a MySQL database hosted on a Ubuntu virtual machine. Moreover, TOSCA introduces so-called Policy Types for modeling non-functional requirements that can be associated with application components.

RADON [1] is a DevOps framework building on top of TOSCA with a specific focus on cloud applications using microservices, FaaS, and data pipelines. While in principle TOSCA models can be edited in any text editor, the RADON framework also provides a web-based graphical modeling tool (GMT) based on Eclipse Winery[1] for more convenient modeling of TOSCA application topologies. Resulting TOSCA models can be processed by TOSCA-compliant orchestrators, which are responsible for the automated deployment and management of the specified application topologies. One such orchestrator is xOpera[2]. RADON extends the TOSCA modeling capabilities and tools (particularly Winery/GMT and xOpera), and adds additional tools to form a respective ecosystem. Figure 1 shows a TOSCA topology for a FaaS-based application modeled in GMT also containing the newly developed RADON TOSCA types. All tools of the RADON ecosystem are integrated in the RADON Integrated Development Environment (IDE) which is a development environment based on Eclipse Che[3]. The RADON software ecosystem is publicly available on GitHub [2].

3 RELATED WORK

CTT is a model-driven and TOSCA-based software testing approach with a focus on DevOps-oriented load testing of cloud applications.

Model-based quality assurance of TOSCA-based cloud applications While there are other tools for quality assurance in general and for using TOSCA entities for specifying test properties [10], there are none that allow continuous testing in TOSCA and the RADON ecosystem [11]. RADON itself provides several other quality assurance tools in its ecosystem: Decomposition Tool, Defect Prediction Tool, and Verification Tool. The Decomposition Tool shares some performance modeling concepts with CTT.

DevOps-oriented load testing In contrast to regular software testing, load testing [12] ensures that the behavior of an application is as expected, even under high load, e.g., high number of concurrent users or number of concurrent requests, etc. Despite CTT being a tool for (load) testing, its purpose is not to add another (load) testing tool in itself, but to provide the framework to use existing and established tools like JMeter[4], Gatling[5], etc. in the context of TOSCA deployments. Also, approaches and tools developed specifically for load testing in DevOps can be integrated into CTT, e.g., following approaches for automating load testing in continuous software engineering [13] and insights from performance practitioners in DevOps [14]. CTT itself provides a framework for setting up the environment for different testing scenarios while the mentioned approaches [13, 14] provide testing strategies. We will cover this topic also in Section 6.

4 CONTINUOUS TESTING TOOL

This section describes the workflow, architecture, modeling, and extension points of the Continuous Testing Tool (CTT).

4.1 Workflow

CTT'S continuous testing workflow comprises three usage scenarios (US), namely Define Test Cases (US 1), Execute Test Cases (US 2), and Maintain Test Cases (US 3). which are overviewed in Figure 2(a), and detailed in Figure 2(b) to Figure 2(d).

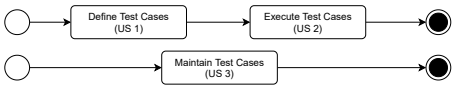
The three usage scenarios can be summarized as follows:

Define Test Cases (US 1). In parallel to the regular development, the Software Developer or QoS Engineer can define test specifications (e.g., deployment and load tests) for their application, referred to as the system under test (SUT). The definition of test specifications is done via the RADON IDE by adding respective TOSCA Policy Types to the SUT's Service Template as described in Section 4.3. Moreover, the developer defines an additional Service Template for the test infrastructure (TI). The resulting artifacts, comprising the SUT's and TI's executable TOSCA artifacts (CSAR) files and input definitions, can be exported from the RADON IDE. Figure 2(b) shows an activity diagram for the usage scenario.

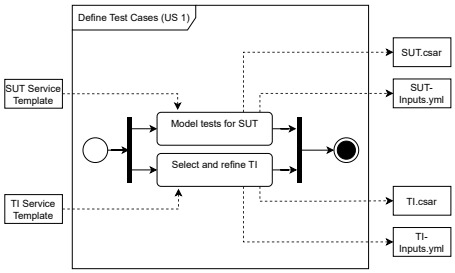
Execute Test Cases (US 2). During development or before deploying to production, actors such as the Developer or Release Manager can manually trigger the execution via the RADON IDE or the standalone interface or automatically via continuous integration (CI) or continuous delivery (CD) [15] for integration into DevOps processes. In each case, the Continuous Testing Tool conducts a series of steps for each selected test case, namely preparing the project context, generating the executable artifacts (CSARs), deploying the SUT and the TI via the orchestrator, executing the tests, and collecting the results. Afterwards, the test results can be inspected. Figure 2(c) shows an activity diagram for the usage scenario.

Maintain Test Cases (US 3). Once the application is deployed in a production environment, operational data can be used for reporting, refining, generating new tests, or updating the existing tests as presented in Section 3 and Section 6, to fit into DevOps contexts and constraints, such as evolving system usage and limited test budgets in CI/CD. Even though different approaches for maintaining test cases are provided in the continuous testing workflow, they share a similar process of analyzing the intended user request, querying the monitoring tool for the required monitoring data, and providing the generated/refined test artifacts. Figure 2(d) shows an activity diagram for the usage scenario.

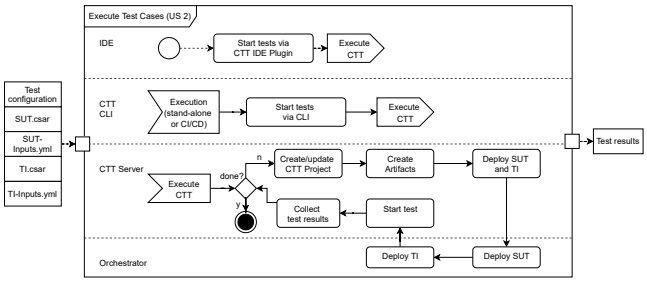
[1]Eclipse Winery – https://eclipse.org/winery/
[2]xOpera – https://github.com/xlab-si/xopera-opera
[3]Eclipse Che – https://www.eclipse.org/che/
[4]Apache JMeter – https://jmeter.apache.org/
[5]Gatling – https://gatling.io/



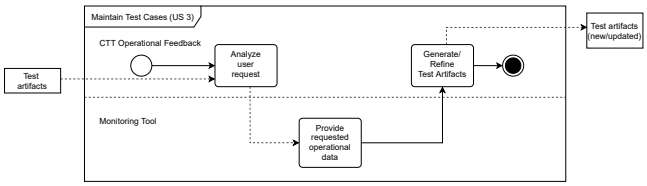
(a) Overview of Usage Scenarios



(b) Usage Scenario 1



(c) Usage Scenario 2



(d) Usage Scenario 3

Figure 2: CTT Usage Scenarios

4.2 Architecture

Figure 3 depicts the overall architecture of CTT and the external components and tools from the RADON ecosystem it makes use of. Tests and test infrastructures are defined in TOSCA models, using CTT-based (extensible) modeling types (as detailed in Section 4.3), e.g., supporting performance or deployment tests. The CTT server is responsible for managing the test execution workflow, including the generation of executable TOSCA models, the deployment of the test infrastructure (TI) and the system under test (SUT) using the xOpera (SaaS) orchestrator, running the tests via CTT test agents, as well as collecting and providing the test results. A CTT agent executes the actual test, e.g., a load test with a load driver such as JMeter. The RADON IDE (including RADON GMT) can be used to define the test-related information, such as the tests and the TIs, and to interact with the CTT server via the CTT IDE plugin. CTT CLI provides an alternative command-line interface to interact with the CTT server. CTT CLI is the preferred way

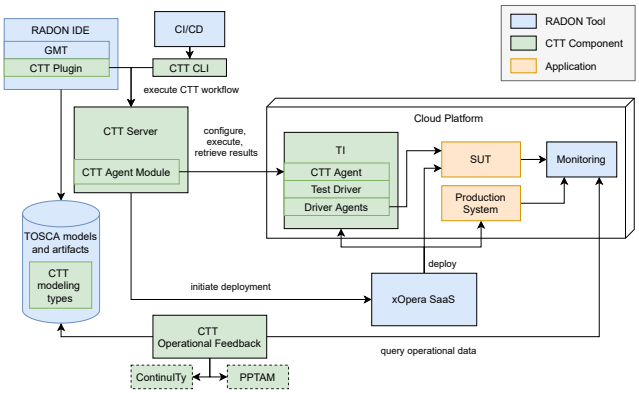


Figure 3: CTT Architectural Overview

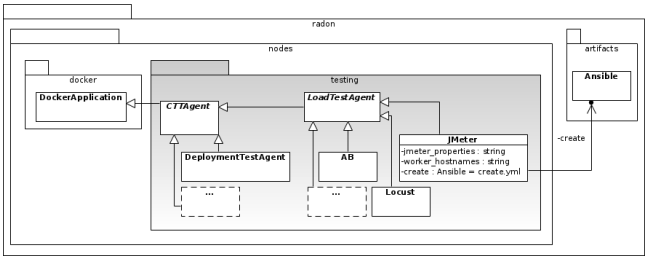
to use CTT in CI/CD pipelines. Moreover, CTT provides various approaches for generating and updating load tests from operational data, particularly to obtain tailored tests for microservices, FaaS, and data pipelines. Therefore, it interfaces with a monitoring tool, such as the one included in the RADON ecosystem.

4.3 CTT TOSCA Modeling

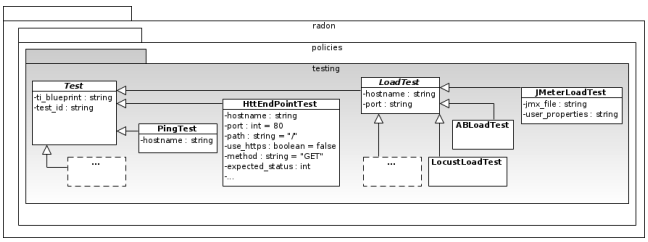
CTT relies on TOSCA modeling concepts to augment the TOSCA models of the SUT as well as to define the TI. Therefore, we have defined a CTT-specific modeling type hierarchy. Test case types are modeled using TOSCA Policy Types. Reusable component types for the test infrastructures, e.g., test drivers, are modeled as TOSCA Node Types. Reusable test infrastructures are provided as blueprints, which essentially are ready-to-deploy sets of TOSCA models. CTT's type hierarchy is integrated into RADON's modeling profile [16]. The CTT types are implemented in RADON Particles, which is the RADON template library containing reusable definitions and extensions, in this case, for testing. Following the RADON modeling approach [16], we provide abstract (AEML) and deployable modeling entities (DEML).

CTT Policy types. The class diagram in Figure 4(b) shows CTT's Policy Type hierarchy. The (abstract) parent type of any CTT test type and test case is *Test*. The type includes a reference to a CTT TI blueprint and a test identifier as attributes. As detailed below, the blueprint is a TOSCA Service Template defining the infrastructure that executes the test. The Policy Types *PingTest* and *HttpEndPointTest* are concrete Policy Types that can be used to define deployment test cases of an SUT to test whether they respond to requests on different protocol levels. The type hierarchy includes another (abstract) Policy Type for load tests, namely *LoadTest*. Example concrete Policy Types for load tests include those for the JMeter (*JMeterLoadTest*), Apache Bench (*ABLoadTest*), and Locust (*LocustLoadTest*) tools. As detailed for *JMeterLoadTest*, the test cases include attributes such as a reference to a load test script and additional properties. Additional types are available and can be added at the places indicated by "..." on the diagram.

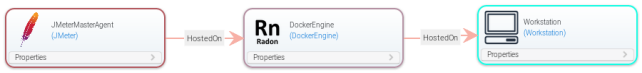
CTT Node types. The (abstract) parent of any test infrastructure Node Type is *CTTAgent*, which derives from RADON's type



(a) CTT Node Types



(b) CTT Policy Types



(c) *JMeterMasterOnly* blueprint as shown in GMT

Figure 4: CTT Modeling

for Docker applications (*DockerApplication*). The class diagram in Figure 4(a) shows CTT's Node Type hierarchy.

Currently, CTT's Node Type hierarchy includes Node Types for executing deployment tests and load tests. The concrete Node Type (*DeploymentTestAgent*) is able to execute deployment tests such as the previously introduced *PingTest* and *HttpEndPointTest*. The type *LoadTestAgent* is abstract and serves as the basis for concrete Node Types for representing agents for load testing tools such as JMeter (*JMeter*), Apache Bench (*AB*), and Locust (*Locust*), corresponding to the respective Policy Types introduced before. As exemplified for JMeter, the Node Type includes attributes such as additional configuration properties or the list of worker nodes (for high-scale load test experiments). Moreover, the Node Types require an implementation artifact so that CTT can trigger the deployment of the respective node in the test infrastructure via the orchestrator. More types are available and can be added at the places indicated by "...".

CTT Blueprints. CTT provides a set of TI blueprints, i.e., TOSCA Service Templates that represent the test infrastructure needed to execute the modeled tests. For example, the blueprint *JMeterMasterOnly* is a TOSCA Service Template that comprises the JMeter Node Type deployed on a Docker engine (*DockerEngine*) hosted on a workstation (Workstation), where both *DockerEngine* and *Workstation* are Node Types already available in RADON Particles. For the JMeter case, various other blueprints are possible, particularly a JMeter master/slave setting with several agents enabling a load test with higher workload intensity. Figure 4(c) shows the *JMeterMasterOnly* blueprint in GMT.

4.4 Extensibility

While CTT contains a range of modules and features out-of-the-box, it has been designed to be extensible to custom test types and test agents, e.g., load generators.

- New test types and test tools are defined by adding or refining (extending) CTT's TOSCA Policies, Node Types, and Service Templates in RADON's TOSCA modeling type hierarchy and by providing Ansible deployment artifacts.
- The CTT server must be extended by adding a new server module that is able to interpret the new test Policies and to communicate with a suitable test agent, which essentially wraps the actual test tool (e.g., load generator). The CTT test agent may be an existing one for an existing tool or a newly developed one for new test tools.
- As the server and the test agent communicate via a REST API, both parts can be replaced or extended accordingly, while the APIs of both sides need to be in sync.

The respective extension points in the CTT architecture are described in detail in CTT's technical documentation [8, 9]. The CTT data pipeline module, as described in Section 7 makes use of CTT's extension points. CTT has only a limited dependency on the RADON ecosystem. It only depends on the TOSCA models used for specifying SUT, TI, as well as the testing configuration as TOSCA policies. Therefore, CTT can most likely be integrated into other frameworks.

5 VALIDATION

The lab validation efforts related to CTT were two-fold: (i) functional validation of the CTT tool implementation including the integration with other RADON tools; (ii) quantitative validation of novel approaches for continuous testing. The remainder of this section provides an overview of the functional lab validation activities. As a combination of (i) and (ii), we demonstrate quantitative results of using CTT in combination with the developed approach for domain-based scalability analysis in Section 6.

The overall goal of the functional validation was to exercise the feasibility of the continuous testing workflow and its implementation by CTT in combination with the related RADON tools. The functional lab validation was conducted using the following RADON microservices/FaaS demo applications: Thumbnail generation, SockShop, and TODOListAPI as SUTs:

- *ThumbnailGenerator*[6] The application demonstrates the usage of FaaS (Function-as-a-Service), comprising two Amazon S3 buckets and an Amazon Lambda function. Once an image file is uploaded to the first bucket, the function is triggered, creating a thumbnail version of the uploaded image and saving it to the second bucket.
- *SockShop*[7] A shop system implemented with DevOps and microservice concepts in mind. Researchers have identified the SockShop as a suitable microservices benchmark application [17].
- *TODOList-API*[8] A ToDo-list application based on FaaS features, comprising a set of Amazon Lambda functions and a MongoDB database.

[6] https://github.com/radon-h2020/demo-faas-thumbnail-generator-python
[7] https://github.com/microservices-demo/microservices-demo
[8] https://github.com/iaas-splab/todo-api-nodejs

We have conducted a corresponding validation with use cases from industrial partners involved in the project. For each SUT, we defined test cases by adding CTT modeling annotations (via TOSCA Policies) to the SUT's TOSCA model. Regarding test types, we focused on deployment tests and load tests, covering scenarios for microservices (SockShop), FaaS (Thumbnail and TODOListAPI), and data pipelines (Thumbnail generation and File Transfer between different Cloud Storage).

Apart from modeling, the validation per SUT comprised the deployment and execution of the tests using the exported CSAR of the SUT and TI. Overall, we were able to demonstrate the realization of a continuous testing workflow that comprises the modeling of testing-related information using GMT and the CTT-related types from RADON particles in GMT, as well as letting CTT process the models exported from GMT including the deployment using the xOpera orchestrator, the execution using a CTT agent, as well as making the results available via CTT. Moreover, we could demonstrate the feasibility of using CTT from the RADON IDE via the CTT plugin, as well as using CTT as a standalone tool and via the CI/CD setting using CTT's command-line interface.

The results show evidence about CTT's feature support based on applying CTT with three applications. CTT is able to express and execute different test types, namely deployment, load, and data pipeline tests. Regarding load testing tools, CTT fully supports JMeter (for microservices/FaaS and data pipelines) and NiFi, and additional modeling concepts for Locust. CTT's extension mechanisms turned out to be very useful. The mechanisms have been used to implement both CTT's core features and external extensions, e.g., for the data pipeline testing (Section 7).

We have used the RADON demo applications from the very beginning of the CTT development, which turned out to be extremely useful for validating the conceptual workflow, deriving tool requirements, defining the architecture, as well as developing and validating the implementation — including its integration with other tools. Moreover, the resulting artifacts served as regression tests that were executed as part of CTT's CI/CD pipeline. It also turned out to be very useful that other partners have used CTT using the example use cases, e.g., for adjusting the setups to the industrial use cases. Regarding the deployment environments for the tests, we have started with local deployments and have incrementally added cloud-based deployment, particularly for AWS.

The artifacts for the mentioned validation using the three benchmark applications are provided as reusable artifacts (Appendix A). The respective SUT models are also included in the RADON particles along with a set of reusable TI models.

6 USING CTT FOR DEVOPS-ORIENTED LOAD TESTING

In the context of the activities around CTT, we have developed novel approaches for continuous testing of performance-related properties to be used in the DevOps context. The approaches are aligned with our vision of automating representative load testing in continuous software engineering [13] and the findings of an industry survey [14].

Key characteristics of the approaches are: (i) Inclusion of operational profile data on system usage in the production environment. (ii) Inclusion of architectural knowledge obtained from production monitoring data. (iii) Option to rely on (semi-)automatically extracted and evolved test scripts. (iv) Selecting and prioritizing relevant test cases with a focus on representativeness. (v) Automation in executing tests.

The developed approaches are microservice-tailored workload generation [18], context-tailored workload generation [19], domain-based scalability testing [20]. In the remainder of this section, we will focus on the latter approach. Details about the integration of all approaches can be found in the CTT technical delivery [9].

CTT Integration with Domain-based Scalability Analysis and PPTAM. Assessing the performance of architecture deployment configurations — e.g., with respect to deployment alternatives — is challenging and must be aligned with the system usage in the production environment. We introduced an approach [20] for using operational profiles to generate load tests to automatically assess scalability pass/fail criteria of microservice configuration alternatives. The approach provides a domain-based metric for each alternative that can, for instance, be applied to make informed decisions about the selection of alternatives and to conduct production monitoring regarding performance-related system properties.

The first step is the collection of data about the system's operational profile, which can be obtained from common APM tools [HHM+17]. The operational data of interest are time series of workload intensities, e.g., number of concurrent sessions, request rates to microservices or functions. This operational data is transformed into an empirical distribution of workload situations, which essentially models the probability of occurrence of the respective workload levels, e.g., 200-400 requests per second in 4% of the time. In parallel, the assumption is that load test scripts are (semi-)automatically created and evolved using the established approaches [21, 22]. The extracted scripts serve as load test templates, which are parameterized by the respective workload level to be tested. The load test tool handles the execution of the test series, comprising the loop of deploying the SUT and the TI, the actual load test execution, and the data collection. Finally, the domain-based metric is calculated from the test results and visually presented in a dashboard.

In this section, we present selected results of using CTT together with the developed research approach for domain-based scalability analysis, implemented in the PPTAM tool [20]. The SUT is the TODOListAPI demo application. The goal is to compare two configurations of the SUT with respect to scalability using operational profile data: (i) the standard configuration from the RADON particles; (ii) the standard configuration but auto-scaling being enabled for the included DynamoDB instance. The data for this experiment is also included in the demo resources dataset (appendix A).

Experiment setup. The load test is implemented as a parameterized JMeter test plan that targets the five endpoints of the TODOListAPI, denoted as *ToDo-Create*, *ToDo-Get-Single*, *ToDo-Get-All*, *ToDo-Update*, and *ToDo-Delete*. The test plan implements a closed, session-based workload with n concurrent users iterating through a sequence of the aforementioned endpoints with some probabilistic branches. The JMeter test plan is integrated into the actual test definition in the TODOListAPI's TOSCA model. The experiment duration is configured to be 840 seconds (14 minutes) with a 2-minute linear ramp-up period.

For both system configurations, we configured CTT to execute a series of tests for eleven different workload intensities (i.e., number of users), namely 1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 users. Hence, we obtained results for 22 different experiments. We removed the first 4 minutes and the last 2 minutes from the raw data of each experiment to eliminate warmup and cooldown effects.

The results of the test with 1 user serve to obtain the baseline criteria for the pass/fail criteria of the scalability assessment.

After CTT has executed successfully, the test results are imported into the PPTAM tool. As the operational profile, we use Wikipedia traces according to the original publication on our approach [20].

Experiment results. Figure 5(a) lists the response time (rt) and error rate results of the baseline experiment (1 user).

Figure 5(b) depicts the response time results over the different workload levels per endpoint and configuration. It can be observed that for both configurations, the response times increase with increasing workload intensity. The configuration without auto-scaling performs worse than the one with auto-scaling.

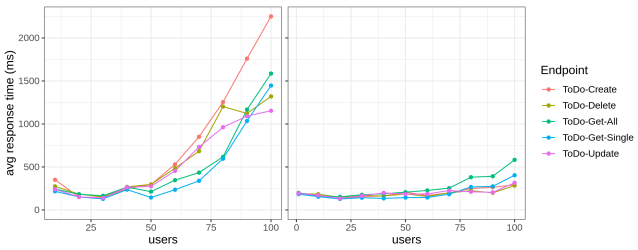
Accordingly, Figure 5(c) shows the error rates per endpoint and configuration. It is clearly visible that again the numbers increase with increasing workload intensity and the configuration without auto-scaling performs worse than the one with auto-scaling.

Figure 5(d) depicts the detailed domain results for the response time pass/fail criteria. The outer polygon represents the theoretically best domain metric per workload level (black line). The inner polygons represent the scalability of the two configurations (red line: without auto-scaling; yellow line: with auto-scaling). It can be observed that both configurations show degradations for increasing load levels. Moreover, the polygon for the configuration without auto-scaling clearly shows that the configuration performs poorly for workload levels greater than or equal to 70 users.

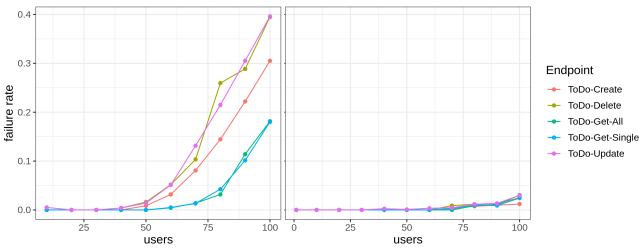
The aggregated domain metric for the two configurations, being the sum of the values per workload level, are: 0.3879 for the configuration without auto-scaling and 0.6630 for the configuration with auto-scaling. Note that a value of 1.0 would be the maximum score, representing that situation that a configuration meets the pass/fail criteria for each workload situation and endpoint.

Endpoint	Avg. rt [ms]	Stddev rt [ms]	Errors [%]
ToDo-Create	197.92	33.70	0.00
ToDo-Delete	191.17	16.11	0.00
ToDo-Get-All	198.75	29.44	0.00
ToDo-Get-Single	183.25	28.89	0.00
ToDo-Update	192.56	26.30	0.00

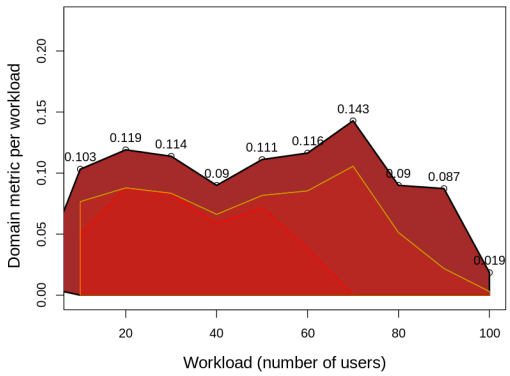
(a) Baseline experiment results with 1 user.



(b) Response times per endpoint and configuration (w/o auto-scaling left; w/ auto-scaling right)



(c) Failure rates per endpoint and configuration (w/o auto-scaling left; w/ auto-scaling right)



(d) Domain metric per workload level for both configurations (w/o auto-scaling: red line; w/ auto-scaling: yellow line)

Figure 5: Domain-based scalability results

7 USING CTT FOR DATA PIPELINE TESTING

In addition to typical cloud services, we also extended CTT to support the testing of data-focused cloud services in the form of data pipeline applications, which are designed for migrating, processing, and storing data in multi-cloud environments. RADON supports modeling data pipelines using TOSCA which are based on either open-source NiFi-based data flow platform or the AWS data pipeline service. RADON data pipelines can be designed, deployed, and managed just like any other cloud service using RADON tools and also required testing to be available, which CTT was extended to support. In order to test data pipeline applications (as SUT) using CTT, the data pipeline TOSCA Service Template should be provided or designed using RADON GMT. This SUT is then annotated with details about which tests to execute by defining a TOSCA testing Policy in GMT. Based on the types of tests specified in the Policy, the TI should also be chosen (or designed), which supports deploying the infrastructure and tools required for the respective tests (e.g. NiFi or JMeter based TI for data pipeline tests).

To support the testing of data pipelines, the capabilities of the CTT were extended with a new CTT server module for data pipelines, TOSCA Policy Types, a custom TI consisting of a Python Agent for communicating with CTT server and a NiFi-based agent for generating test data for load testing of the data pipelines. In addition, a Prometheus [9] server is used for collecting performance metrics.

[9] https://prometheus.io/

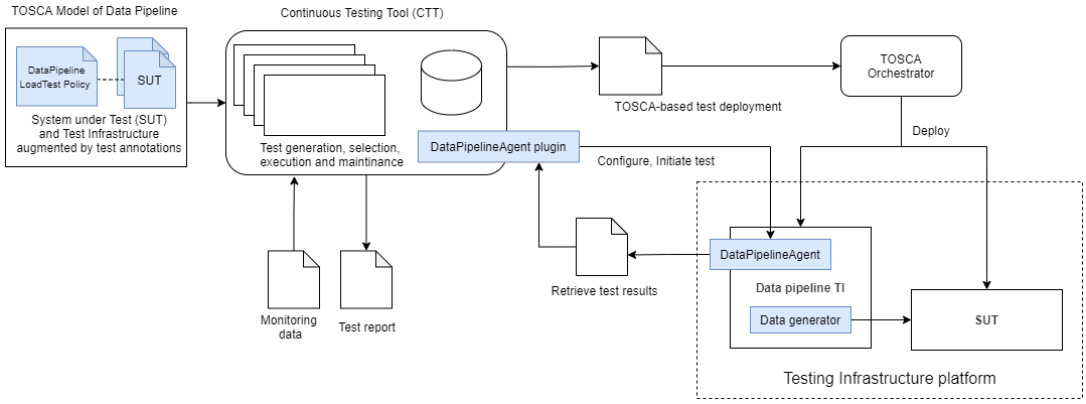


Figure 6: High-level architecture of CTT data pipeline module

The CTT data pipeline module provides support for exposing monitoring data and generating metrics at the level of individual data pipeline blocks, efficiently generating data for load testing data pipelines, and defining and setting up corresponding test infrastructures. This is achieved by integrating the NiFi run-time (used for hosting and executing data pipelines) directly with the Prometheus monitoring data collection server.

CTT Data Pipeline Testing Module. A high-level architecture of the CTT data pipeline module and its interaction with the data pipeline application in terms of artifacts, tools, and infrastructures is depicted in Figure 6. The main purpose of the CTT data pipeline testing module is to provide support for exposing monitoring data and generating metrics at the level of individual data pipeline blocks, efficiently generating data for load testing, defining and setting up corresponding test infrastructures, and collecting test result metrics. Initially, a user defines a data pipeline test application by adding them to a TOSCA Service Template for the SUT. The CTT data pipeline module-specific TOSCA Node Types and Policy Types are defined and made publicly available in the RADON Particles GitHub repository [23] for expressing different types of tests and TIs.

Experimental Analysis of the CTT Data Pipeline Module. To verify that the designed CTT data pipeline module works as intended, we apply it to evaluate the performance of two different data pipeline applications: (i) *Serverless image Thumbnail Generation*: data pipeline, which listens for images uploaded to an AWS S3 bucket, applies an AWS Lambda function to generate a thumbnail, and stores results into another S3 bucket. (ii) *File Transfer*: data pipeline which transfers uploaded files from an AWS S3 bucket into a Google Cloud storage bucket.

We have used the CTT tool to deploy this application together with a testing infrastructure, which generates three different types of input files for the data pipelines: (i) Image files for the thumbnail generation data pipeline. (ii) IoT data in the form of JSON files for the file transfer data pipeline. (iii) Tweets in the form of CSV files for the file transfer data pipeline.

For the image datasets, we used an INRIA Holiday dataset containing 500 image groups, each group representing a distinct scene or object [24]. For testing the File Transfer data pipeline service,

Table 1: Load testing of Thumbnail generation with CTT data pipeline module using multiple image datasets

Dataset	Images	Load Time (ms)	Throughput	Success Rate (%)	Avg. Proc. Time (s)
D-1	25	25694	2.2/min	100	8.31
D-2	50	51238	1.2/min	100	14.29
D-3	100	60283	0.99/min	100	58.48

two different datasets have been used that included the Twitter Dataset[10], which consists of 100 CSV files, each containing more than 300 tweets, and the IoT Sensor dataset, containing 100 readings of temperature and humidity sensors, and is generated synthetically using a modified IoSynth data generator[11] in the form of JSON files.

We verify the load testing of the Thumbnail generation with the CTT data pipeline module by using a JMeter-based data pipeline TI which generates REST requests uploading multiple images to the data pipeline input S3 buckets. To test the application performance, we defined three separate size image datasets, each with a different number of images: 25, 50, and 100. The CTT data pipeline testing agent uploaded the images to the source S3 bucket of the data pipeline. During load testing, we verified that the data pipeline services still perform as expected in terms of multiple performance metrics, i.e., load time, throughput, and processing time. Further, the success rate of the SUT is tested by comparing the uploaded images on the source S3 bucket with the thumbnails in the other data endpoint or destination S3 buckets. Table 1 reports the final validation results of the Thumbnail generation data pipeline application. The experiments demonstrated that the success rate of the Thumbnail generation data pipeline application is 100% for all three datasets, which shows that there were no failures.

During load testing, we verified that the data pipeline services still perform as expected in terms of success rate and processing time. Table 2 reports the final validation results of the File Transfer data pipeline by using the CTT agent. Through the experiments, it has been demonstrated that the success rate of the File Transfer

[10]kaggle: Twitter sentiment analysis — https://www.kaggle.com/c/twitter-sentiment-analysis2/data
[11]IoSynth IoT synthetic data generator — https://github.com/Afsana2910/iosynth

Table 2: Load testing of File Transfer data pipeline between different Cloud Storage application with CTT data pipeline module using multiple file datasets

Dataset	Data uploaded	Data in destination bucket	Success Rate (%)	Avg. Processing Time (sec)
Twitter Data	100	100	100	0.47
Sensor Data	100	100	100	0.49

data pipeline is 100% for all two datasets, which indicates that there were no failures in the data pipeline.

8 CONCLUSION

The Continuous Testing Tool provides the means for testing deployments modeled with the OASIS TOSCA standard. CTT supports generating, executing, and refining continuous tests of application functions, data pipelines, and microservices, as well as for reporting test results. While aiming to provide a general framework for continuous quality testing in TOSCA and its RADON extension, a particular focus of CTT lies on testing workload-related quality attributes, such as performance, elasticity, and resource/cost efficiency. Due to CTT's modular architecture, it can be extended and customized easily. In this paper, we presented the CTT tool and its application to DevOps-oriented load testing and load testing of data pipelines. Promising future work on CTT comprises the development of extensions for additional test types and tools.

A SUPPLEMENTARY MATERIAL

The technical deliverables [8, 9, 25] from the RADON project include additional details about CTT and its lab validation. The source code of CTT is open-source and available on GitHub [26]. Also, videos have been produced to show the features and usage of CTT in the form of a teaser video [27], as well as a webinar recording [28]. An archival version of the artifacts is provided on Zenodo [29]. In addition, the three data pipeline testing datasets (CSV, Images, JSON) are also available separately on Zenodo [30–32]

ACKNOWLEDGMENT

This paper has been partially supported by the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 825040 (RADON). We thank Andrea Janes from the Free University of Bozen-Bolzano for his support in integrating CTT with PPTAM.

REFERENCES

[1] G. Casale *et al.*, "RADON: Rational decomposition and orchestration for serverless computing," *SICS Software-Intensive Cyber Physical Systems*, 2020.
[2] RADON Consortium, "RADON tools," 2020. [Online]. Available: https://github.com/radon-h2020/
[3] Organization for the Advancement of Structured Information Standards (OASIS), "TOSCA Simple Profile in YAML Version 1.3," 2019. [Online]. Available: https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/TOSCA-Simple-Profile-YAML-v1.3.html
[4] D. D. Nucci, "Deliverable D3.1: RADON methodology," 2021. [Online]. Available: https://radon-h2020.eu/wp-content/uploads/2021/09/D3.1-RADON-methodology.pdf
[5] C. K. Dehury, P. Jakovits, S. N. Srirama, G. Giotis, and G. Garg, "TOSCAdata: Modeling data pipeline applications in tosca," *Journal of Systems and Software*, 2022.
[6] S. Kounev *et al.*, "Toward a definition for serverless computing," *Report from Dagstuhl Seminar 21201*, 2021.
[7] L. J. Bass, I. M. Weber, and L. Zhu, *DevOps — A Software Architect's Perspective*, ser. SEI Series in Software Engineering. Addison-Wesley, 2015.
[8] A. van Hoorn and T. F. Düllmann, "Deliverable D3.4: Continuous testing tool I," 2020. [Online]. Available: https://radon-h2020.eu/wp-content/uploads/2020/07/D3.4-Continuous-testing-tool-I.pdf
[9] A. van Hoorn *et al.*, "Deliverable D3.5: Continuous testing tool II," 2021. [Online]. Available: https://radon-h2020.eu/wp-content/uploads/2021/09/D3.5-Continuous-testing-tool-II.pdf
[10] M. Wurster *et al.*, "Modeling and Automated Execution of Application Deployment Tests," in *Proc. of the IEEE 22nd Int. Enterprise Distributed Object Computing Conf., EDOC*. IEEE Computer Society, 2018.
[11] A. U. Gias *et al.*, "Performance engineering for microservices and serverless applications: The RADON approach," in *Comp. 2020 ACM/SPEC Int. Conf. on Performance Engineering, ICPE*. ACM, 2020.
[12] Z. M. Jiang and A. E. Hassan, "A survey on load testing of large-scale software systems," *IEEE Trans. Software Engineering*, 2015.
[13] H. Schulz *et al.*, "Towards automating representative load testing in continuous software engineering," in *Comp. of ACM/SPEC Int. Conf. on Performance Engineering, ICPE*. ACM, 2018.
[14] C. Bezemer *et al.*, "How is performance addressed in devops?" in *Proc. of ACM/SPEC Int. Conf. on Performance Engineering, ICPE*. ACM, 2019.
[15] J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.
[16] M. Wurster, V. Yussupov, and D. Tamburri, "Deliverable D4.3: RADON Models I," 2019. [Online]. Available: https://radon-h2020.eu/wp-content/uploads/2019/11/D4.3-RADON-Models-I.pdf
[17] C. M. Aderaldo *et al.*, "Benchmark requirements for microservices architecture research," in *1st IEEE/ACM Int. Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering, ECASE@ICSE*. IEEE, 2017.
[18] H. Schulz *et al.*, "Microservice-tailored generation of session-based workload models for representative load testing," in *27th IEEE Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS*. IEEE Computer Society, 2019.
[19] ——, "Context-tailored workload model generation for continuous representative load testing," in *ACM/SPEC Int. Conf. on Performance Engineering, ICPE*. ACM, 2021.
[20] A. Avritzer *et al.*, "Scalability assessment of microservice architecture deployment configurations: A domain-based approach leveraging operational profiles and load tests," *Journal of Systems and Software*, 2020.
[21] C. Vögele *et al.*, "WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction - a model-driven approach for session-based application systems," *Software and Systems Modeling*, 2018.
[22] H. Schulz *et al.*, "Reducing the maintenance effort for parameterization of representative load tests using annotations," *Software Testing, Verification & Reliability*, 2020.
[23] RADON Consortium, "RADON particles," 2020. [Online]. Available: https://github.com/radon-h2020/radon-particles
[24] H. Jegou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *European Conf. on Computer Visualization, ECCV*. Springer, 2008, pp. 304–317.
[25] V. Yussupov, "Deliverable D6.5: Final Assessment Report," 2021. [Online]. Available: https://radon-h2020.eu/wp-content/uploads/2021/09/D6.5-Final-Assessment-Report.pdf
[26] Thomas F. Düllmann and André van Hoorn and Pelle Jakovits, "GitHub: RADON Continuous Testing Tool (CTT)." [Online]. Available: https://github.com/radon-h2020/radon-ctt
[27] RADON H2020, "RADON 2020 | Continuous Testing Tool," 7 2021. [Online]. Available: https://www.youtube.com/watch?v=WyLSG9rrpaA
[28] R. H2020, "RADON Webinar 2 | Introduction to Continuous Testing," 7 2021. [Online]. Available: https://www.youtube.com/watch?v=DsShmfUih2c
[29] T. F. Düllmann and A. van Hoorn, "RADON Continuous Testing Tool (Final)," 6 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4973131
[30] M. Adhikari, A. Khan, and P. Jakovits, "Data pipeline validation and load testing using multiple CSV files," 3 2021.
[31] M. Adhikari, A. Khan, and S. Narayana, "Datapipeline execution validation and load testing of multiple images," 6 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3884525
[32] M. Adhikari, A. Khan, and P. Jakovits, "Data pipeline validation and load testing using multiple JSON files," 3 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4636789