Advanced Engineering of Microservices and Serverless Applications: The RADON approach
RADON Consortium

- ICT-16-2018: Software Technologies
- 30 months project (Jan 2019 - Jun 2021) – 8 organizations
Serverless Function-as-a-Service

• FaaS: function calls served from the cloud, event-driven paradigm
• Quick prototyping and demonstration without infrastructure management issues

• A way to reduce costs
  • Fine-grained billing
  • Automated deallocation

• Natural to combine with microservices-based architectures
  • Fine-grained software architecture
  • Automated autoscaling
  • Flexibility and responsiveness
  • High-degree of reuse of platform services
Some challenges

• How to deploy and update in continuous, DevOps fashion, hybrid serverless-based applications?
• How to choose an optimal deployment configuration respecting QoS?
• How to debug infrastructure code?
• How to best do testing and monitor outcomes?
Value proposition:

Offer an open source DevOps framework to help the EU software industry adopting serverless FaaS without vendor lock-in.

Tools at advanced state.

Open source releases.
RADON framework overview

Constraints Definition
Graphical Modelling
Decomposition Tool
Defect Prediction
Verification Tool
Continuous Testing Tool
Template Library

TOSCA Blueprint

CI/CD
Function Hub
Orchestrator
Monitoring
Data Pipelines
Data-driven serverless computing
Graphical Modeling Tool
RADON: modelling for serverless FaaS

- Easy-to-use composition of functions, microservices, storage, VMs, ...
- Reuse modular element to assemble complex applications

1. RADON Models
2. Graphical Modeling Tool (+ Blueprint Generation)
3. Constraint Definition Language (+ Verification Tool)
Models automatically deployable using the RADON orchestrator.
Quality Guardrails in RADON

1. Create model
2. Run verification
   e.g., GDPR constraints

3. Extract product metrics
   (e.g., # lines of code)
4. Extract delta metrics
   (between two successive releases)
5. Extract process metrics
   (e.g., # modifications to the file in a release)
6. Run detection

[All constraints verified]

1. Extract product metrics
2. Extract delta metrics
3. Extract process metrics
4. Run detection

DEPLOY

Application Source Code
```
int div(a, b):
    return a/b
```
Possible division by zero

- name: "foo"
  include: es-template.yml
  when: es_templates
  when: es_templates | bool

Infrastrucutre Code
this makes the application behave wrongly

Continuous Testing Tool
1. Create tests
2. Run tests
Continuous Testing

CTT

Parser
Generator
Updater
Optimizer
Executor

xOPERA

Test deployment specification (RADON/TOSCA)

Test results

Test deployment

Application under test (AuT)
Test annotations

Test infrastructure (TI)

Application project (RADON/TOSCA)

QoS Engineer

Test report
Demo

Unlocking the benefits of serverless FaaS for the European software industry

- Graphical Modelling

RADON 2020 - Graphical Modelling
20 views · Apr 6, 2020
Unlocking the benefits of serverless FaaS for the European software industry

- Decomposition
Unlocking the benefits of serverless FaaS for the European software industry - Continuous Testing
RADON: optimization & decomposition trade-offs

- What is the optimal size for a service taking into account for constraints?
- How do we converge through development cycles towards an optimal architecture?
- How to model and predict QoS?

<table>
<thead>
<tr>
<th>Monolith</th>
<th>SOA</th>
<th>Microservices</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+ Separation of concerns + Specular to business - Pre-cloud - No infrastructure focus</td>
<td>+ Container-based + Easy to migrate + Reproducible + Vendor-agnostic - Manual admin - Running costs</td>
</tr>
<tr>
<td></td>
<td>+ Simplified arch. + Less to deploy + Less to manage - Inflexible - Slow updates</td>
<td>+Scalability +Cost +Zero admin -Resource limits -Size limit -Vendor lock-in</td>
</tr>
</tbody>
</table>

- We have different architectures: Monolith, Service-Oriented Architecture (SOA), Canonical (container based), Serverless FaaS (platform).

- Each architecture has its own benefits and drawbacks.
Decomposition & Optimization

- Optimization problem:
  - **type**: non-linear integer programming (NLIP);
  - **variables**: memory and concurrency;
  - **constraints**: average response time less than 2.5 sec

- Performance modeling:
  - **benchmarking**: service demand estimation
  - **formalism**: layered queueing networks (LQNs)

- LQN for the thumbnail generation example (simplified)

  - clients (open workload)
  - create_thumbnail (Lambda function)
  - thumbnails (S3 bucket)
  - uploads (S3 bucket)
T3.3: Continuous Testing

Results after Y1

- Design of continuous testing workflow, tool architecture, and integration;
- Support for modeling and executing selected test types (focus: performance tests);
- Initial research contributions on:
  - Tailored testing (MASCOTS 19);
  - Regression testing of microservices (accepted for ICPE 20);
- Application to RADON examples (SockShop and Thumbnail);
- Started interaction with use cases (ATC, PRQ) and tools (monitoring, CI/CD);
- Prototypes being made available as open-source:
  - CTT server: https://github.com/radon-h2020/radon-ctt
  - CTT agent: https://github.com/radon-h2020/radon-ctt-agent
  - https://hub.docker.com/r/ustctt/
T3.4: Defect Prediction Tool

Why?

"Infrastructure-as-code (IaC) ⇒ managing and provisioning compute datacenters through machine-readable definition files"

Cit. TOSCA Simple Profile Yaml v1.3, CSD2

- As any other source code artifact, IaC files may contain defects that can preclude their correct functioning and operations;

- The tool is intended for detecting defect-prone IaC blueprints at the end of a release cycle;

- Defect-Prediction SoTA from Dev. source code is well-established in the use of Machine Learning techniques:
  - Scripts prone to contain imperfections or deficiencies cause them not to meet their requirements or specifications;
  - Metrics identify such qualities, so that smells or bug-proneness can be detected and possibly repaired;
Continuous Testing Tool (CTT)

- Functionalities grouped into 3 usage scenarios:
  i. Test case definition
  ii. Test execution
  iii. Test maintenance
- CTT modules
  i. Microservices/FaaS
  ii. Data pipelines
- Usage:
  i. Standalone tool (open-source)
  ii. Invocation via the RADON IDE or CI/CD