# Deliverable D3.2. In vivo Testing Service for Microservice Quality Assessment Accompanying document.

June 2023

## 1 Scope

This is the accompanying document of Deliverable D3.2 of the µDevOps project entitled, "In vivo Testing Service for Microservice Quality Assessment". The type of the deliverable is marked as *Other*, and is made up of software artifacts, along with this accompanying document. The delivered artifacts are made available on the project website `www.udevops.eu`, as well as on the following GitHub repositories:

`https://github.com/uDEVOPS2020/MacroHive`
`https://github.com/uDEVOPS2020/EMART`
`https://github.com/uDEVOPS2020/ReliabilityAssessment`

and at the linked Zenodo repository:
`https://doi.org/10.5281/zenodo.8143905`, indexed by OpenAIRE

## 2 Introduction

This document describes the artifacts implemented for the testing service under development for the Microservice quality assessment. It integrates the developed testing algorithms based on sampling and on machine learning. Specifically, the artifact implements a prototype testing service embedding the testing algorithms designed and described in Deliverable D3.1: *Sampling-based Testing Techniques Design and Algorithms for QoS Testing* and embedding teh solutions developed and described in the WP2 Deliverables D2.1 and D2.2.

Instructions are provided for using the algorithms of the testing service and reproduce the examples developed for illustrative purpose. The artifact will be extended during the project, as the Consortium will advance with WP4 and WP5.

## 3 Automated Functional Testing

This artifact is the prototype implemented for automatic tests generation for Microservice applications. It featues tests generation and monitoring via a trace-based technique. It is described in Section 4 of Deliverable 3.1. The artefact along with instructions for its usage are available at:
`https://github.com/uDEVOPS2020/MacroHive`

Hereafter, we report the main content and steps: The repository is divided into four folders: *uProxy*, *uSauron*, *uKnows* and *uTest* are the 4 main components of MacroHive.

*uTest* is the test generator, it can be deployed in docker with the "buildandstart" script. It exposes by default port "11111". It is possible to edit the default configuration with the docker-compose file. All commands are HTTP requests to this microservice. Requests are collected in scripts (ClientCommands folder) to perform functionalities. In particular:

- *init_xx.sh*: sends all selected files for the testing session to the microservice environment;
- *retrieveSpec_xx.sh*: automatically retrieve microservice's specification, through information defined in a json file (e.g., ports_ftgo.json). Represent an alternative to the previous instruction, where specification files are needed;
- *execute.sh*: start the testing session;
- *clear.sh*: clear the microservice's environment (e.g., to perform a new testing session);
- *getOutput.sh*: retrieve all output information after a testing session, and put the files in output folder. Must be called before clear.

Initialization files (in initFiles) must be configured before script calls.

*uSauron* collect the traces from proxies. It can be deployed in docker with the "buildandstart" script. It exposes by default port "11112". It is possible to edit the default configuration with the docker-compose file. Scripts in "ClientCommands" folder can be executed in order to perform functionalities, in particular:

- *proxycompose.sh*: automatically upload a docker-compose file in input and gives in output the same file manipulated to add proxies (experimental);
- *clear.sh*: clear the microservice's environment;
- *getX.sh*: retrieve computed metrics, in output folder.

*uProxy* contains the implementation of the reverse proxy to be deployed along every microservice.

*uKnows* performs the failure propagation analysis. It can be deployed in docker with the "buildandstart" script. It exposes by default port "11113". It is possible to edit the default configuration with the docker-compose file. Scripts in "ClientCommands" folder can be executed in order to perform functionalities, in particular:

- *setup.sh*: retrieve data from uSauron;
- *compute.sh*: compute activities for test reporting;
- *getOutput.sh*: retrieve the output;
- *clear.sh*: clear the microservice's environment.

## 4 Reliability and Performance Testing

This artifact is the prototype of the automatic reliability and performance tests generation for Microsrevice applications. The goal of testing in this case is to estimate the expected reliability or performance of the microservice application. This prototype exploits sampling-based algorithms described in Section 3 of Deliverable 3.1 and implements the estimator described in Section 3.5. The full support for performance assessment is under completion. The artefact along with instructions for its usage are available at:
`https://github.com/uDEVOPS2020/EMART`

These two tests generators will be integrated with algorithms for tests prioritization and failure propagation analysis analysis developed in WP2.

Hereafter, we report the main content and steps: all the results obtained in the experimentation of the proposed technique (called EMART) are collected and reported in the repository. Each folder in the repository is coupled with the corresponding Research Question (RQ) as described in the original paper. In the RQ1 and RQ2 folders, results in terms of MSE and Variance are organized considering:

- the true operational profile: profile1, profile2, profile3 and variable profile;
- the error attached to relative estimated profiles: 10%, 90%.

In the pictures of the repository, both EMART results and Operational Testing (OT) results, used as baseline, are reported. The RQ3 contains all the results of the cost-benefit analysis considering both MSE and Variance. All considerations on the additional results with respect to those reported in the paper confirm the main conclusions described in the paper.

In the folder "code", the source code of the EMART engine is available. To run EMART, these steps need to be followed:

1. Import the source code in an IDE (e.g., Eclipse);

2. Populate the data structure Test Frame;
3. Define a Weight Matrix to build connections among Test Frames;
4. Set the values of n (number of samples to be selected), and d (a weight parameter representing the probability of adopting the weight-based sampling in the adaptive sampling algorithm, 0.5 is the default).

To repeat the experiments, it is necessary to:

1. Download and run the application defined in the paper as ”experimental subjects”;
2. Generate requests according to a desired profile (defined in the test frame data structure) to the interface of the components;
3. Run a monitoring infrastructure (like Metro Funnel https://github.com/dessertlab/MetroFunnel.git) to collect the methods invocation and update the information in input to the EMART engine.

## 5 Reliability assessment from data

This artifact reports Bayesian algorithms that we made available to provide a reliability assessment functionality from data. Such data could be collected from testing (e.g., from the two tests generator above) or from online observations. Data regard the observed failures per "partition", where a partition is, in the case of an MSA, a microservice or an equivalence class in the microservice' API input space. This is described in Section 3.4 of Deliverable 3.1

The artefact along with instructions for its usage are available at:
`https://github.com/uDEVOPS2020/ReliabilityAssessment`

Hereafter, we report the main content and steps:

- The Matlab code under the BayesianReliabilityAssessment folder.
- The file ReliabilityAssessment.m for the assessment of reliability in one iteration. It takes as input:
  - demands: an (1 x m) matrix of demands (with m being the number of partitions);
  - failures: an (1 x m) matrix of failing demands (with m being the number of partitions);
  - dirichlet_step: the step used for discretization in the Dirichilet PDF computation;
  - beta_step: the step used for discretization in the Beta PDFs computation.

It provides:

  - marg_distr: The output PFD distribution (n-by-1 matrix);
  - cond_distr_params: The conditional PFD distribution, one per partition;
  - mean of marg_distr.

- The file IterativeAssessment.m is used for a continuous assessment over k iterations, either by the model selection approach or without the model selection approach. It takes as input:

  - cumulative_demands: an (n x m) matrix of cumulative demands (with n being the number of iterations and m the number of partitions);
  - cumulative_failures: an (n x m) matrix of cumulative failing demands (with n being the number of iterations and m the number of partitions);
  - dirichlet_step: the step used for discretization in the Dirichlet PDF computation;
  - beta_step: the step used for discretization in the Beta PDFs computation;
  - model_selection: if set to 1, the dynamic model selection approach (by Bayes factor) is applied to select the model; if different than 1, the dynamic model selection is disabled.

  It provides:

  - marg_distr: The output PFD distribution (n-by-1 matrix);
  - mean of marg_distr.

  The other files in the folder which are support functions used by these two files.

- The Matlab code under the KalmanFilterAssessment folder. The file KalmanReliabilityAssessment.m for the iterative assessment of the expected PFD by means of a discrete time-varying Kalman filter formulation. takes as input:

  - demands: an (n x m) matrix of (non-cumulative) demands (with n being the number of iterations and m the number of partitions);
  - truePFD (optional): for our experimentation purpose, the value of the true PFD to compute so as the offset and the plot.
  - % failures: an (n x m) matrix of (non-cumulative) failing demands (with n being the number of iterations and m the number of partitions);

  It provides:

  - offset: The offset (difference between the estimated and true PFD). If the truePFD is not provided, offset=[];
  - sPFD: structure containing the output of the Kalman filter at each iteration (see the auxiliary file klamanf.m for details about the structure)

  The folder contains kalmanf.m, a support function used by KalmanReliabilityAssessment.m.

  In both cases, reliability is simply R = 1 - PFD (namely, "1 - marg_distr" to obtain the distribution, or "1 - mean(marg_distr)" to obtain the expected reliability)

- The file InputForAssessment.xlsx contains the demands and the failing demands obtained after testing the subject under test described in the paper, which is NLP-Building-blocks. These are the inputs to be provided to the Matlab files to reproduce the results in the paper.

- The JavaCode folder contains experimental code to ease the execution of tests on a generic subject (a REST web service). To use it: - Import the source code in an IDE (e.g., Eclipse). - Select a subject. - Modify the endpoints of the service API to invoke. - Optionally, define a profile for the partitions (if not defined, a uniform profile is generated). - Optionally, modify the type of input classes generated. The TestFrame class defines some example classes to generate Strings with different features. This can be customized depending on the need. - Deploy the subject - Run the main.class file. Results are printed to a file output.txt. The response code is used as oracle to get the number of demands and of failed demands.

- Finally, the source code of NLP-Building-Blocks is uploaded, with instructions on how to deploy it (via docker) and how to invoke it. The name of the APIs are in the source folder.