A Framework for SLO-driven Cloud Specification and Brokerage

Abdessalam Elhabbash, Yehia Elkhatib, Gordon S Blair
MetaLab, SCC, Lancaster University, UK
{*i.lastname*}@lancaster.ac.uk

Yuhui Lin, Adam Barker
School of Computer Science, University of St Andrews, UK
{yl205,adam.barker}@st-andrews.ac.uk

***Abstract*—The diversity of cloud offerings motivated the proposition of cloud modelling languages (CMLs) to abstract complexities related to selection of cloud services. However, current CMLs lack the support for modelling service level objectives (SLOs) that are required for the customer applications. Consequently, we propose an application- and provider-independent SLO modelling language (SLO-ML) to enable customers to specify the required SLOs. We also sketch the architecture to realise SLO-ML.**

***Index Terms*—Cloud Computing, Cloud Modelling Languages, Service Level Agreements, Service Level Objectives.**

I. Introduction

The growing expansion of the cloud market poses a challenge to its customers who are already overwhelmed with a wide choice of cloud service [1]. The scale as well as diversity of the range of offerings, and their real time performance variation are adding more challenges to the selection decision [2], [3]. One approach to facilitate such decision is to base the selection on provider guarantees regarding service performance that is defined as a set of service level objectives (SLOs) that are part of the service level agreements (SLAs).

In order to make it easier for customers to select services and deploy applications, cloud modelling languages (CMLs) were proposed (*e.g.,* [4]). They provide means for high level description of a cloud application's topology, then automate their deployment. With respect to SLO modelling, a few of the proposed CMLs support such modelling through standards that are designed primarily for providers to specify their services levels (*e.g.,* WS-Policy). In other words, they lack support for customer-oriented SLOs modelling.

As a result, we propose a design of a new language for SLO modelling, *SLO-ML*, that provides a comprehensive syntax for capturing service level requirements, supporting all SLOs currently used by IaaS providers and those specified in industry standards. SLO-ML provides customers with a high level of abstraction, whereby they can specify SLOs for required cloud services regardless of the low level details of those services. More importantly, SLO-ML supports applications in both single- and multi-cloud environments. In addition, we present the architecture of cloud brokerage system that realises the SLO-ML based selection of services.

II. SLO Modelling Language

The key aim of SLO-ML is to provide a comprehensive syntax to capture all possible SLOs that customers may require to specify service levels of their applications, taking into account the requirement for multi-cloud deployment. For this purpose, SLO-ML enables customers to specify SLOs for each application component. Therefore, SLO-ML supports SLO specification in both single- and multi-cloud applications.

A. Design principles

The design of SLO-ML is based on the following principles:

1) **Customer-oriented.** SLO-ML is designed to enable customers to specify their high-level operational requirements in a simple declarative syntax.
2) **Independence**. To avoid vendor lock-in, SLO specification should be independent of cloud service specification. Furthermore, it needs to be independent of cloud application development technology and implementation details.
3) **Abstraction**. Customers should be able to specify SLOs regardless of the required type of cloud service, such as SaaS, PaaS, FaaS, etc.
4) **Separation of concerns**. It should be possible to maintain and adapt isolated SLO specification at an application component level. For example, a load-balancing component's SLOs should be separate from those of a data storage element.
5) **Mapping SLOs** A high-level SLOs which specified by users should be broken down to low-level ones, and then further mapped to the application component level. For example, the response time of a three-tier application consists of processing time for each layers.

B. Key elements of SLO-ML

We adopt JSON syntax for representing SLOs. The elements of the current syntax are: **Names**, **Value types**, **Units**, and **Operators**. An illustrative exmaple is given in Listing 1.

A unique keyword name is used to refer to each SLO. The keywords are self-explanatory, making it simple for developers to understand. For example, the keyword `Response_Time` is used to refer to the response time SLO.

SLO-ML supports three types of the SLO values: scalar, interval, and categorical. The scalar type is used to specify a numerical value (*e.g.,* availability = 0.9999). The interval type is used to specify an upper- and lower-bound of SLO value (*e.g.,* response time between 5ms and 10ms). Categorical types provide a higher level of abstraction for SLO value specification, allowing customers to specify a category (*e.g.,*

Listing 1 SLO-ML file structure example

```
{   "aws_compute_web": [
      { "name": "response_time",
        "unit": "ms",
        "value": "5-10",
        "operator": "in"
      },
      { "name": "availability",
        "unit": "",
        "value": "0.999",
        "operator": ">="
      }
    ],
    "aws_db": [
      ...
    ]
}
```

low, medium, high) instead of specific values or a predefined range, relieving customers from specifying an exact value in case they are not certain. For example, for memory-intensive application, a customer can specify the category 'high' for the `Memory_Size` SLO.

SLO-ML uses a set of keywords that specify the units of measurement of the each SLO. For example, the `Migration_Time` SLO is specified using the hours unit. In addition, SLO-ML contains rules for unit-to-unit conversion between units of the same kind.

SLO-ML defines a set of operators to specify relational SLO values. This set includes: *less_than*, *less_than_or_equal*, *greater_than*, *greater_than_or_equal*, *equal*, and *in*. For instance, *in* can be used to indicate that *response_time* should be in the interval `[5ms,10ms]`.

III. Broker Architecture

We provide a preliminary architecture for a cloud broker that realises cloud deployment based on user-provided SLO-ML descriptions (Fig. 1). Our approach views the cloud application as a set of components, each of which requires a set of SLOs to be specified. The approach builds on existing approaches of modelling cloud applications such as Terraform HCL [1], TOSCA [2], etc. We assume that the customer request consists of two models, namely the SLO model defined using SLO-ML and the Infrastrucure as Code (IaC) model defined using an application description language (*e.g.,* Terraform HCL, TOSCA, etc.), which is basically the application description model. Our proposed broker architecture consists of the following main components:

Parsing and Validation. This component parses both the SLO model and the IaC model to extract the required SLOs for each component. The validation intends to evaluate the SLO specification by checking the correctness of the (i) syntax, (ii) units, and (iii) consistency of the configuration. Syntax validation aims at inspecting the syntax for any errors in using SLO-ML keywords. Unit validation aims to check for any improper use of units. Consistency validation ensures that component references in the SLO file correspond to the application components described in the IaC model.

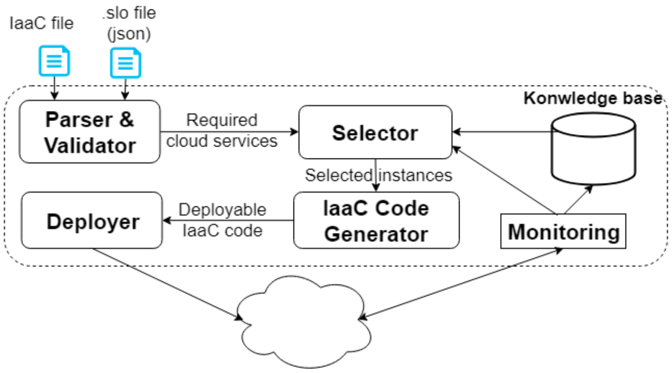

Fig. 1: Broker architecture to realise SLO-ML based selection

Knowledge base is a repository the stores information of the cloud instances such as their type, provider, and the service levels. The knowledge base contains also monitoring data that represent the real time performance of the cloud services.

Selector selects services that match the required SLOs for each component of the application. In its simplest implementation, the selection is based on the providers SLAs. Other implementations include intelligent selection using the monitoring data.

IaC Code Generator generates the deployment code of the application based on the selected instances. The generated code is a tuned version of the IaC model submitted by the user where the information of the selected services are injected in the model.

Deployer receives the deployment code and automates the deployment of the application on the selected cloud instances.

Monitoring monitors the performance of the selected cloud services in terms low level metrics. The collected data are stored in the Knowledge base. The metrics are then mapped to the high level SLOs. If the mapping results in violation of an SLO, the violation is reported to the selector to re-select new instances and adapt the application.

IV. Future Direction

Our immediate next step is to address the implementation of the brokerage system. This includes developing mechanisms for (i) mapping between cloud service SLAs of different providers, (ii) monitoring cloud service performance using low-level metrics, and (iii) mapping low-level metrics to high-level SLOs required by cloud customers.

References

[1] Cloud Standards Coordination (CSC), "CSC Phase 2: Cloud computing users needs - analysis, conclusions and recommendations from a public survey," The European Telecommunications Standards Institute (ETSI), Special Report 003 381 V2.1.1, Feb 2016.

[2] F. Samreen, Y. Elkhatib, M. Rowe, and G. S. Blair, "Daleel: Simplifying cloud instance selection using machine learning," in *IEEE/IFIP NOMS*, April 2016, pp. 557–563.

[3] C. Kilcioglu, J. M. Rao, A. Kannan, and R. P. McAfee, "Usage patterns and the economics of the public cloud," in *WWW*, Republic and Canton of Geneva, Switzerland, 2017, pp. 83–91.

[4] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, *TOSCA: Portable Automated Deployment and Management of Cloud Applications*. Springer, 2014, pp. 527–549.

[1] https://www.terraform.io/docs/configuration/syntax.html

[2] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca