Support Services for Applications Execution in Multi-Clouds Environments

Daniel Pop, Gabriel Iuhasz, Ciprian Crăciun, Silviu Panica

***Abstract*—Deploying and running applications in multi-cloud environments is a challenging task for a number of reasons: different configuration parameters are needed for different clouds, application's artifacts may vary across technologies or cloud providers, services provided at IaaS level vary from one cloud provider to another. This paper introduces a run-time platform that enables the deployment and execution of applications on multi-clouds with guaranteed QoS, and details the services of a unified layer responsible for connecting to several underlying IaaS cloud providers in order to avoid runtime lock-in and to simplify the management of cloud applications.**

***Index Terms*—multi-cloud execution, runtime, multi-cloud deployment**

I. Introduction

Cloud services promise flexible yet cheap solutions to end-users at a much larger scale than before. The proliferation of cloud service providers in all cloud computing business models (IaaS/PaaS/SaaS), doubled by the fact that cloud technologies are still in their early stages raises specific challenges and requires advanced software engineering methods and support tools in order to steer the domain to maturity age.

Deploying and running applications in multi-cloud environments is a challenging task for a number of reasons. Firstly, in order to deploy and run the application in a multi-cloud setup multiple parameters to configure the application on different clouds are needed, such as user credentials, service endpoints etc. In our approach, the runtime environment features an object store service designed to handle different configuration parameters and application-specific data. Secondly, a distributed application is composed of multiple binary artifacts that may vary across technologies, such as Ruby's gems, Java's jars and wars or Python's eggs. Artifact repository of ADDapters 4Clouds platform is a storage service designed to store, version and retrieve software artifacts, such as deployment recipes, Maven artifacts, software packages or binary data.

The runtime environment presented in this paper was developed as part of the MODAClouds [1] approach [1], [2] that combines model-driven development, risk analysis, quality prediction and deployment to support application developers and operators in the adoption of a multi-cloud strategy. There are three main ingredients of MODAClouds solution: Creator 4Clouds [3], inspired by OMG Model-Driven Architecture, is an environment comprising of design-time tools (such as IDE, UML meta-models, QoS and analysis tools etc.); Energizer 4Clouds (E4C), a rich execution platform that provides service discovery, packaging, configuration, enactment, monitoring, data synchronization and self-adaptation for multi-cloud applications; and Venues 4Clouds [4], [5], a Decision Support System designed to support organizations on the task of choosing the most suitable cloud provider for their Cloud applications.

The remaining of this paper is structured as follows. The next section section discusses the challenges of deploying and running multi-cloud applications and how our approach answers them. Section III overviews the overall architecture of Energizer 4Clouds, while Section IV details the ADDapters 4Clouds layer, responsible for connecting to several underlying cloud service providers at IaaS and PaaS levels. Finally, we conclude with a discussion of ongoing and future work.

II. Challenges and Opportunities

In this section, we highlight the challenges in deploying and running multi-cloud applications that are not holistically addressed by existing cloud environments.

Avoid runtime lock-in: Complex cloud services are distributed systems, which require complex coordination, configuration, packaging, binding, and discovery capabilities. cloud providers such as Amazon EC2 have included these services in their PaaS offerings, however, a service provider may face lock-in if it decides to adopt these PaaS solutions. One may therefore resort to existing open source software, but this lacks native multi-cloud support.
E4C's approach: In E4C, the ADDapters 4Clouds platform is a set of portable open-source support services that can be deployed on multiple IaaS Cloud to help package and deploy multi-cloud applications. These support services can be used to support applications running either on IaaS or PaaS. This is further detailed in section IV.

Simplify management in multi-clouds: When an application is deployed on IaaS platforms, application management is done through web-based consoles and APIs offered by the cloud provider. However, for a multi-cloud application, management would require one to use heterogenous consoles and APIs to continuously track and modify the state of the application. This puts significant overheads on management teams, which are required to develop ad-hoc interfaces to the target clouds. E4C aims at reducing this management burden by providing a unified view

This manuscript was submitted to Self Organizing Self Managing Clouds Workshop (SOSeMC 2016), affiliated with the 13th International Conference on Autonomic Computing (ICAC)

Daniel Pop and Gabriel Iuhasz are with the Department of Computer Science of West University of Timişoara

Ciprian Crăciun, Silviu Panica are with the research institute e-Austria Timişoara

[1] http://www.modaclouds.eu/

and semi-automated management toolchain for the execution environment.

E4C's approach: In E4C, the MODAClouds Models@Runtime engine [3], included in SpaceOps 4Clouds component, provides an abstraction layer to continuously track the application state across multiple cloud environments, leveraging the MODACloudML specification and dynamically updating such specifications as the application is changed at run-time. The Models@Runtime engine provides an API for enacting run-time changes, such as stopping, modifying or starting new VMs on a target cloud.

QoS/SLA monitoring and adaptation: Service-level agreements (SLAs) and, more in general, Quality-of-Service (QoS) requirements, can be very challenging to achieve for cloud applications, which may access a heterogeneous set of resources, such as virtual machines with very different amount of resources (CPU, memory, bandwidth, etc). Multi-cloud applications further increase heterogeneity, placing an even higher-bar for SLA management and QoS adaptation [6]. For example, a multi-cloud application requires multi-cloud load balancing to react to network congestion periods in a given cloud. However, techniques such as load-balancing, capacity scaling and reactive monitoring become much more difficult to operate in multi-clouds.

E4C's approach: E4C architecture features a self-adaptation platform, component of SpaceOps 4Clouds, that is able to leverage design-time information from the MODACloudML specifications to optimise run-time QoS management. The platform provides load-balancing capabilities, the integration with the SLA-aware monitoring rules and auto-scaling. The load-balancing techniques have, among others, a modus operandi that is agnostic of the utilization of virtual machines, thus they can be used also for PaaS deployments, in addition to the canonical IaaS ones.

Cloud bursting: An important application scenario for a multi-cloud application is that of hybrid cloud applications that can react to an increase in workload by obtaining additional resources from a different cloud provider. This may either be a burst from one private cloud to another private cloud, from one private cloud to a public cloud or between two different public clouds. This requires the capability, for the underlying management toolchain, to dynamically start and stop VMs or containers on a different cloud at runtime and to simultaneously track application states across multi-clouds.

E4C's approach: SpaceOps 4Clouds platform has been extended to operate in this scenario, developing the underlying adaptation process, the enactment mechanisms, and the capability in the Models@Runtime engine to consolidate the simultaneous application management across clouds. It simultaneously abstracts the distributed application state in a single model operated by the Models@Runtime engine, as opposed to tracking the state across clouds by two separate engines. This simplifies the operation and management of multi-cloud applications at runtime.

III. ENERGIZER 4CLOUDS ARCHITECTURE

The runtime environment focuses on deployment of applications on multi-clouds with guaranteed QoS, a first step towards the development of cloud-enabled future Internet applications [7]. Energizer 4Clouds runtime platform is composed of all services and resources involved in running and managing an application on a given cloud provider.

Figure 1 illustrates the overall architecture of E4C. The design goals of this environment are:

- to define a monitoring platform able to characterise the state of applications deployed on multi-clouds,
- to extend the design-time models for performance, reliability and scalability evaluation in the run-time environment,
- to use models for the definition of runtime policies for adaptive management,
- to develop a runtime environment framework for managing deployments and executing the run-time policies,
- to define a data synchronisation layer to keep consistency in multi-Cloud environments.

The environment is composed of two sub-platforms: the Monitoring Platform (Tower 4Clouds) [8] and the Execution Platform, which in turn is composed of two sub-platforms: ADDapters 4Clouds and SpaceOps 4Clouds.

SpaceOps 4Clouds [9] is a multi-cloud self-adaptation and policy reconfiguration engine providing QoS prediction and topology optimization. The design-time computed estimated optimization of the best deployment scenario is refined at runtime by SpaceOps 4Clouds using information gathered during the execution of the application. This allows interoperability at runtime preventing double handling of information between development and operations. SpaceOps 4Clouds is composed of two main parts: the Models@Runtime engine and the self-adaptation platform. The first component acquires data from the monitoring platform, keeps track and shows the status of the multi-cloud application its execution environment to its operators. Moreover, it allows performing reconfiguration actions. The self-adaptation platform acquires monitoring data from the monitoring platform and uses it to run performance and availability models that allow the platform to identify the best self-adaptation actions, which are then proposed to the Models@Runtime engine.

In the remaining of this paper we detail the ADDapters 4Clouds component of the runtime environment.

IV. ADDAPTERS 4CLOUDS

ADDapters 4Clouds (A4C) is the IaaS and PaaS unified layer responsible for connecting to several underlying cloud service providers at IaaS and PaaS levels. By creating an abstraction layer exposing a vendor-independent API to add different underlying providers, this platform avoids runtime lock-in and simplifies the management of cloud applications. In the context of Energizer 4Clouds, it enables the deployment and execution of the monitoring and self-adaptation components. As illustrated in Figure 2, A4C layer is composed of an object store, an artifact repository, a load-balancer controller and a lightweight cloud-tailored operating system (mOS), which are detailed in this section.

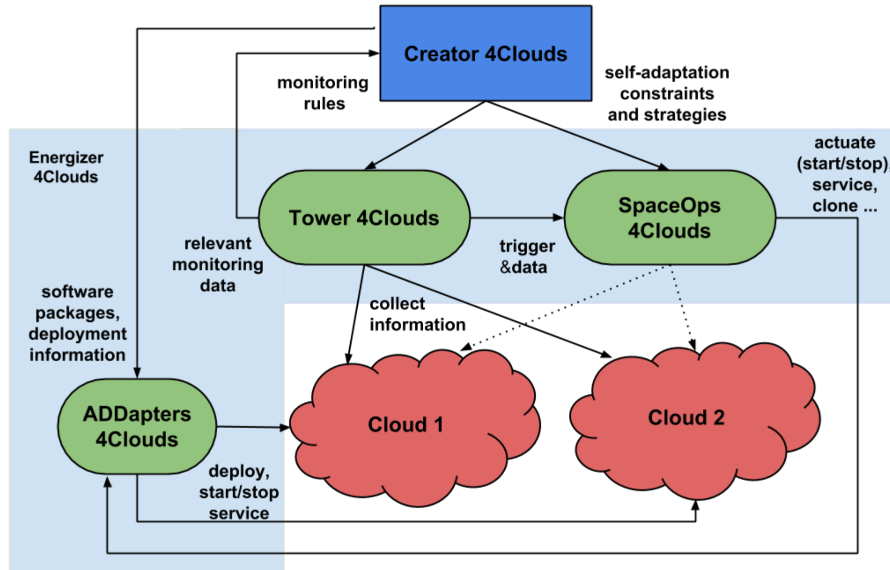

Fig. 1: Energizer 4Clouds Architecture

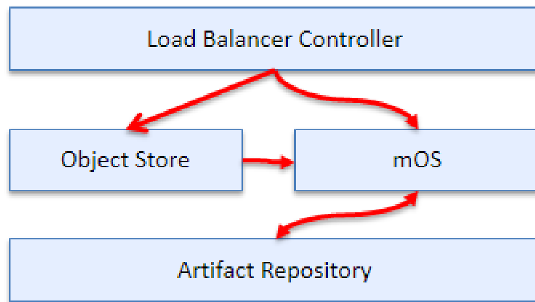

Fig. 2: ADDapters 4Clouds Architecture

A. Load Balancer Controller

The goal of Load Balancer Controller (LBC) is to provide a HTTP RESTful API able to control and configure HAproxy [2], which is a reliable, high performance TCP/HTTP load-balancer. Based on the end-user inputs, the LBC generates a configuration file for the HAproxy load-balancer that configures and controls proxy's behaviour. It exposes both the frontend and backend settings of HAproxy, with a limited support for ACL specifications. Currently each configuration file is saved into the database and can be accessed by querying the database. The API is designed to:

- *add, edit and delete resources*, so that pools, gateways, endpoints and targets can be defined. These represent direct representations of resources present in HAproxy. Each interaction is saved and versioned.
- *set policy*, load-balancing policies and their associated parameters can be set on each target. For example, in the case of round-robin we can set the weights for each target.
- *start HAproxy service*, after the configuration file is generated, it is then used to start the load-balancing service. Each time a new configuration is generated, it is reloaded into the already running service.

The LBC is designed to hide as much technical details of HAproxy as possible. This is done in order to make the REST API as agnostic as possible. For example, in LBC we use the term gateway to define a frontend server and pool to define the backend servers. This enables easy extension of the LBC and the REST resource structure can be easily mapped onto other load-balancer solutions.

Load Balancer Controller is implemented using Python `Flask` [3] micro-framework and `Werkzeug` web server gateway. It stores all interactions in a `Sqlite` database, which also serves as the basis for the configuration file. The `Jinja2` template engine is used to generate the configuration file which is then loaded into HAproxy.

B. Object Store

The Object Store provides an alternative to the more traditional locally stored configuration files for software configuration. This is necessary as the traditional approach is not well suited for cloud environments where VM templates are extensively used. This means that each VM upon start-up has to rewrite these configuration files. Although popular configuration management systems, such as Puppet [4] or Chef [5] provide advanced deployment and configuration management features, they require a central database where the actual

[2]http://www.haproxy.org

[3]http://flask.pocoo.org/docs/0.10/
[4]http://docs.puppetlabs.com/
[5]http://docs.chef.io/

configuration parameters are stored and agents (clients) on each node of the cluster.

An object store component in the design of a runtime environment is also motivated by the fact that some deployed services need to store small pieces of data for later retrieval, or implement some form of weak synchronisation within a cluster.

In both scenarios a distributed solution is needed, which is here provided by the Object Store component. In the context of Object Store, an *object* is a keyed container which aggregates various attributes that refer to the same subject. For example, an object can store the configuration parameters of a given service (or class of services), or the end-point (and other protocol parameters) where a given service can be contacted. Each object has the following attributes: data, indices, links, annotations and attachments.

The most basic usage of an object would be to store some useful information, and have it available for later access. The stored *data* can vary from JSON or XML to a binary file. Besides the actual data, stored data in an object is characterized by its content-type, which specifies how to interpret the data. Although in most of the cases each object will store a single data item, nothing prevents the usage of multi-part/mixed data that bundles together multiple data items. However, it is advisable to avoid such a scenario and use either links or attachments.

Access to the data is atomic, thus consistent, and concurrent updates are permitted without any locking or conflict resolution mechanisms, the latest update overriding previous ones, thus no isolation with lost-updates being possible. Although the data can be frequently accessed or updated without high overhead, it is advisable to cache operations by using the dedicated HTTP conditional requests.

In addition to its data, an object can be augmented with *indices* that enables efficiently selecting objects on other criteria than just the object key. An object can have multiple indices, each index being characterised by a label and a value, and it is allowed to have multiple indices with the same label. Using indices, one can select or delete objects that have an index with a given label and value, or objects that have an index with a given label, in increasing or decreasing order based on the value, or objects with an index with a given label, and a value in a given range.

The value of an index can be any valid JSON term, and it is allowed to have multiple indices with the same label but different value types. However the order between two values of different types is unspecified, including between floats and integers. If the value is a JSON object, the order in which the attributes are specified is important, therefore sorting the attributes lexicographically based on their keys improves lookup performance.

It has to be mentioned here that there is a major difference between the indices implemented in Object Store component and their counterparts in NoSQL or relational databases. In object store, indices are built based on meta-data attached to the actual data (using the `index` attribute), not on the actual data as in the case of databases management system. By separating indexing from the actual data we have greater control over how the data is stored and retrieved, and we can optimise the access to data for those access patterns where the data changes frequently, but the values used by the indexer stay the same.

Links is the feature allowing an object to reference other objects. For example, one could have a service configuration object storing specific parameter values and pointing to a global configuration object, which stores the default parameter values. A link is characterised by a label and the referenced object key and it is allowed to have multiple links with the same label or the same referenced object, therefore many-to-many relations are supported. Unlike indices, links are scoped under the object, are unidirectional (from the declaring object towards the referenced one), and are not usable in selection criteria. Therefore one can not deduce which objects reference a given target object (unless a full scan of the store is executed). The only operation, besides creation and destruction, that can be applied to a link is link-walking, where by starting from an object, one can specify a label (and an index in case there are multiple links with the same label) and gain access to the referenced objects attributes.

Data that logically belongs to the object, but which is either too large to be used as actual data or is static, can be placed as *attachment*. Attachments are created in two steps. First, the attachment is created by uploading its content and obtaining its fingerprint. Second, a reference to the attachment (i.e. its fingerprint) is placed within the object with a given label, together with the content-type and size. The same attachment can be referenced from multiple objects without uploading its data, provided that the fingerprint is known. Similar to links, attachments are scoped under an object, only their data being globally stored. In terms of efficiency, creating or updating attachments do not have high overhead (except the initial data upload) because the information related to a specific object, such as the actual data, meta-data, links, annotations or attachments, are not lumped together.

The *annotations* are meta-data specified for objects or attachments, and are characterised by a label (unique within the same object) and a JSON value. Annotations can be used to store ancillary data about the object, especially those used by operational tools, such as the creator, tool, the source, ACL, digital signatures, etc.

A *collection* groups objets together based on purpose or type (such as all configuration objects in one collection), or based on scope (such as all objects belonging to the same service).

The Object Store supports multi-cloud deployment via replication. The replication process has three phases: defining on the target (i.e. the server) a replication stream, which yields a token used to authenticate the replication; defining on the initiator (i.e. the client) a matching replication stream; and the actual replication which happens behind the scenes. The replication is one way: the target (i.e. the server) continuously streams updates towards the initiator (i.e. the client). If two-way replication is needed, the same process must be followed on both sides.

Regarding conflicts, and because internally the object store exchanges patches which only highlight the changes, any conflicting patch is currently ignored. It is therefore highly

recommended to confine updates to a certain object only to one of the two replicas. However, if multiple changes happen to the same object, and multiple patches are sent, and say the first one yields a conflict, but the rest do not, only the conflicting patch will be discarded, the others being applied. It is possible to obtain replication graphs or trees, including cycles, and the object store handles these properly.

C. Artifact Repository

The artifact repository component is designed to store, version and retrieve software artifacts, such as deployment recipes, Maven artifacts, software packages or binary data in general. An artifact may be composed of one or more binary files (BLOBs). Artifact Repository supports versioning at artifact level. A HTPP REST API allows consumers to manage and search (based on meta-data) artifacts stored in the repository. Internally, artifacts are stored as binary files in the file system with the following hierarchy: repositories/artifacts/versions/files. Synchronizing multi-clouds repositories is easily achieved using `rsync` command. Comparing to Object Store component, Artifact Repository was designed to support large binary files, contrasting the text-based approach used for object repository.

D. mOS

mOS is a lightweight operating system based on OpenSUSE 13.1[6] operating systems and it is used to host and run the MODAClouds platform. It is designed to run on private/public cloud infrastructures, pre-compiled kernels being available for Amazon AWS, Google Compute Engine and Flexiant Cloud Orchestrator. mOS supports Btrfs file system, XEN and HVM virtualization, LXC containers, KVM virtualization, Flexiant HVM and Amazon EC2. There are several services of mOS that are paramount to its functioning, which are briefly overviewed below.

The mOS bootstrap service is tasked to customise the runtime platform by starting required services at boot time. These services are in charge to create the run-time environment. ZeroConf services are specific services hosted by the cloud providers that enable the interaction between active VMs and a special service in order to obtain information about a specific resource, such as user-data, password-less SSH public key, user name and password pairs, or network information [10]. VM resource registration is handled by the naming service which generates unique name randomly and registers it with the DNS. There are other services, such as user-data service, package daemon and logging service, which are responsible for user scripts, package installation and event logging, respectively.

V. Initial Validation

The services described in this paper have been validated throughout the MODAClouds [7] project. Load Balancer Controller was used either for load balancing different components, or for service discovery. Artifact Repository was used by the controller for storing and retrieving checkpoint data. Multicloud load balancing is also considered and implemented using the same toolset. Object Store was used to store configuration data for all components running on the MODAClouds Runtime. This ensures a coherent view of the current configuration for all tools.

[6] http://opensuse.org

[7] http://www.modaclouds.eu/

VI. Conclusions

The contribution of this paper to advance of state-of-the-art is two fold: (1) it presents the overall architecture of a runtime environment for the execution of applications in multicloud environments, and (2) details the layer responsible for connecting to several underlying cloud service providers at IaaS and PaaS levels. Each support service is a self-contained software package, which can be easily reused and extended.

We are planning to investigate the ADDapters 4Clouds services in the context of Big Data applications. Object Store and Artifact Repository can be used to store data and artifacts needed for Big Data applications deployment and execution, while Load Balancer Controller can be used to mitigate the load on monitoring services of Big Data applications.

Acknowledgments

This work has received funding from the EC-funded projects MODAClouds (FP7-318484) and DICE (644869).

References

[1] D. Ardagna, E. D. Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D'Andria, G. Casale, P. Matthews, C. S. Nechifor, D. Petcu, A. Gericke, and C. Sheridan, "MODAClouds: A model-driven approach for the design and execution of applications on multiple Clouds," in *Modeling in Software Engineering (MISE), 2012 ICSE Workshop on*, June 2012, pp. 50–56.

[2] E. Di Nitto, M. Almeida da Silva, D. Ardagna, G. Casale, C. Dorin Craciun, N. Ferry, V. Muntes, and A. Solberg, "Supporting the Development and Operation of Multi-cloud Applications: The MODAClouds Approach," in *Proc. of. SYNASC 2013*, Sept 2013, pp. 417–423.

[3] N. Ferry, H. Song, A. Rossini, F. Chauvel, and A. Solberg, "CloudMF: Applying MDE to Tame the Complexity of Managing Multi-cloud Applications," in *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, Dec 2014, pp. 269–277.

[4] A. Omerovic, V. Muntes-Mulero, P. Matthews, and A. Gunka, "Towards a Method for Decision Support in Multi-cloud Environments," in *Proc. of 4th International Conference on Cloud Computing, Grids, and Virtualization (CLOUD COMPUTING 2013)*, 2013, pp. 162–180.

[5] B. Manate, V. I. Munteanu, T. F. Fortis, and P. T. Moore, "An Intelligent Context-Aware Decision-Support System Oriented towards Healthcare Support," in *Complex, Intelligent and Software Intensive Systems (CISIS), 2014 Eighth International Conference on*, July 2014, pp. 386–391.

[6] V. Munteanu, T. Fortis, and V. Negru, "An Evolutionary Approach for SLA-based Cloud Resource Provisioning," in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, March 2013, pp. 506–513.

[7] D. Petcu and A. V. Vasilakos, "Portability in clouds: approaches and research opportunities," *Scalable Computing: Practice and Experience*, vol. 15, no. 3, 2014.

[8] M. Miglierina, W. Wang, G. Casale, and V. I. Munteanu, "Monitoring Platform Final Release," *MODAClouds Deliverable D6.3.2*, 2014.

[9] M. Guerriero, M. Ciavotta, G. P. Gibilisco, and D. Ardagna, "SPACE4Cloud: A DevOps Environment for Multi-cloud Applications," in *Proceedings of the 1st International Workshop on Quality-Aware DevOps*, ser. QUDOS 2015. New York, NY, USA: ACM, 2015, pp. 29–30. [Online]. Available: http://doi.acm.org/10.1145/2804371.2804378

[10] S. Panica and D. Petcu, "Distributed Resource Identification Service for Cloud Environments," in *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2013 15th International Symposium on*, Sept 2013, pp. 448–453.