Separation of Concerns in Heterogeneous Cloud Environments

Dapeng Dong, Huanhuan Xiong and John P. Morrison
Boole Centre for Research in Informatics, University College Cork, Western Gateway Road, Cork, Ireland

Keywords: Service Management, Resource Management, Cloud, HPC.

Abstract: The majority of existing cloud service management frameworks implement tools, APIs, and strategies for managing the lifecycle of cloud applications and/or resources. They are provided as a self-service interface to cloud consumers. This self-service approach implicitly allows cloud consumers to have full control over the management of applications as well as the underlying resources such as virtual machines and containers. This subsequently narrows down the opportunities for Cloud Service Providers to improve resource utilization, power efficiency and potentially the quality of services. This work introduces a service management framework centred around the notion of Separation of Concerns. The proposed service framework addresses the potential conflicts between cloud service management and cloud resource managment while maximizing user experience and cloud efficiency on each side. This is particularly useful as the current homogeneous cloud is evolving to include heterogeneous resources.

1 INTRODUCTION

There are many cloud service management frameworks in existence today. They provide the means to specify and to compose services into applications for deployment in cloud environments. In essence, these frameworks embody service descriptions, deployment specifications, and the specific resources required to run each service. Current frameworks support a combination of both the IaaS and PaaS interaction models: resources for each service are acquired separately without reference to the needs of other services, and once acquired; the framework manages both the application lifecycle and the underlying resources. Consequently, the Cloud Service Provider loses control over the management of these resources until they are freed by the framework.

Current trends in cloud computing is to create a service-oriented architecture for the evolving heterogeneous cloud. In this respect, it is imperative to maintain a separation between application lifecycle management and resource management. This separation of concerns makes it possible for the user to concentrate on "*what*" needs to be done and for the cloud service provider to concentrate on "*how*" it should be done making it possible to implement continuous improvement, in terms of resource utilization and service delivery, at the resource level. The proposed framework is twofold including a Service Description Language and its associated implementation. They include facilitating application lifecycle management by the cloud consumer and resource management by the Cloud Service Provider to ensure a proper separation of concerns between both; and creating application Blueprints consisting of many services and taking account of the entire collection of services to determine an optimal, and potentially, a heterogeneous, set of resources to implement them.

The remainder of the paper is organized as follows. We discuss the related work in Section 2. The proposed concept of separation of concerns is elaborated in Sections 3. The realization of the proposed concept is presented in Section 4. In Section 5, conclusions are drawn and future directions are discussed.

2 RELATED WORK

According to the cloud computing service delivery model, Infrastructure as a Service (IaaS) is described as the capability of processing, storage, network, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, such as operating system and applications. Generally speaking, the management and control of these resources will be given to the consumer as well, such as

Dong, D., Xiong, H. and Morrison, J.
Separation of Concerns in Heterogeneous Cloud Environments.
DOI: 10.5220/0006385507750780
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 747-752
ISBN: 978-989-758-243-1

initiating, starting and/or terminating virtual machines and networks. A large number of cloud providers offer infrastructure resources as services to consumers in their specific ways, which commonly causes "vendor lock-in" issues. Fortunately, several efforts were conducted towards a general framework/description of resource management across different cloud vendors, in order to improve the interactions among various cloud providers in an abstract way. Some of them have been commonly adopted by cloud computing communities, such as Apache jClouds (jCloud, 2017) and Apache Libcloud (Libcloud, 2017). For example, Apache jClouds offers a number of services for resource management operations in Java, providing support for around 30 providers including most of the major cloud vendors such as Amazon Web Services, Microsoft Azure Storage, OpenStack, Docker, and Google App Engine.

On the other hand, Platform as a Service (PaaS) allows the consumers to deploy applications onto the cloud infrastructure in two different ways: (1) the consumers manage the cloud application and the providers manage the underlying infrastructure; (2) the consumers manage the lifecycle of cloud applications together with their associated underlying resources. Most existing PaaS framework are using the second approach.

Existing frameworks manage the lifecycle of cloud applications together with their associated underlying resources. In this section, we use three representative frameworks to highlight the tightly coupled nature of their integrated management approaches at two levels within the cloud. Figure 1 shows the lifecycle management architectures for the OpenStack Solum (Solum, 2017), Apache Brooklyn (Brooklyn, 2017), and OpenStack Heat frameworks (Heat, 2017). These frameworks provide a set of tools and APIs in the form of a self-service interface to allow cloud consumers to interact with the cloud. Solum and Brooklyn operate at the PaaS level while Heat operate predominantly at the IaaS level.

The Solum and Brooklyn frameworks allow cloud consumers to create blueprints using an appropriate Service Description Language (SDL), and to deploy blueprints in clouds. SDLs are used to describe the characteristics of application components, deployment scripts, dependencies, locations, logging, policies, and so on. The Solum engine takes a blueprint as an input and converts it to a Heat Orchestration Template (HOT), this template can be understood by the application and resource management engine (Heat). The Heat engine, thereafter, carries out the application and resource deployment processes by calling the corresponding service APIs that are provided by the underlying cloud infrastructure framework (such as OpenStack).

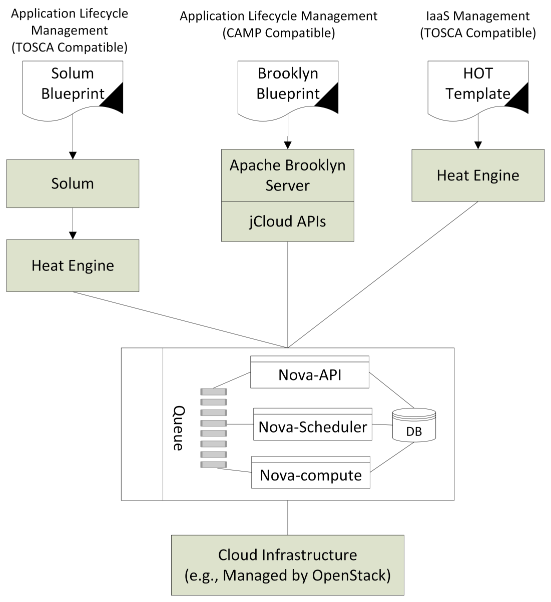
Application Lifecycle Management (TOSCA Compatible)
Application Lifecycle Management (CAMP Compatible)
IaaS Management (TOSCA Compatible)
Solum Blueprint
Brooklyn Blueprint
HOT Template
Solum
Apache Brooklyn Server
jCloud APIs
Heat Engine
Heat Engine
Queue
Nova-API
Nova-Scheduler
Nova-compute
DB
Cloud Infrastructure (e.g., Managed by OpenStack)

Figure 1: Overview of cloud application/resource lifecycle management of OpenStack Solum, Apache Brooklyn, and OpenStack Heat.

In contrast, Brooklyn engine converts a blueprint into a series of jCloud API calls that can be used to directly interact with the underlying cloud infrastructure. For example, an API call for creating a Virtual Machine (VM) in OpenStack will be sent to the *nova-api* service. The *nova-api* service subsequently notify the *nova-scheduler* service to determine where the VM should be created. The final process is to notify the *nova-compute* service for actual VM deployment. This "*Request and Response*" approach is simple, robust, and efficient. However, it should be noted that each request is processed independently, making it impossible to consider relative placement of VMs associated with multiple requests. Consequently, this traditional approach does not support the optimal deployment of a group of services, such as those in a Brooklyn blueprint, for example. This limitation is not specific to VM placement, but also applies to current container deployment technologies.

3 THE CONCEPT OF SEPARATION OF CONCERNS

From the previous section, two striking characteristics of traditional frameworks are apparent, the first is that they do not support the separation of concerns between the various actors of the system, and the second

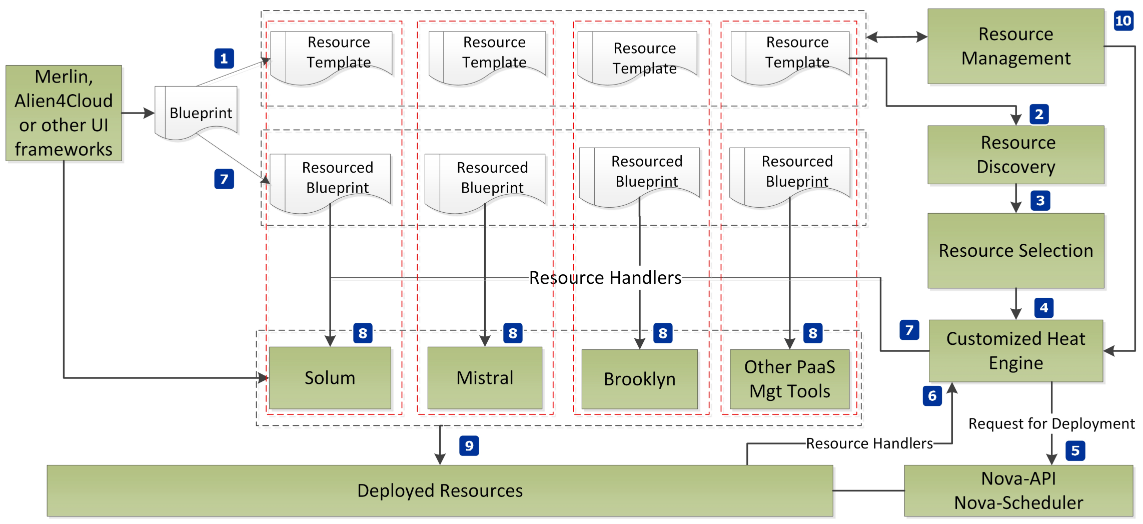

Figure 2: The Service Delivery Model with Regards to the Concept of Separation of Concerns.

is that global optimizations between multiple services are not possible due to the way in which resource requests are individually processed.

The proposed approach tackles both these limitations directly. It explicitly introduces separation of concerns between cloud consumers and cloud service providers, and it provides direct architectural support for considering optimal resource requests from multiple interacting services simultaneously.

In order to separate the concerns of cloud application lifecycle management and resource lifecycle management, a Blueprint in conjunction with a Service Catalogue is used to construct a Resource Template, which will be used to identify/create appropriate resources. This process is shown in Figure 2 (label 1). This also implies that a Service Description Language (SDL) is needed and created in such a way that a Blueprint written in the SDL can be transformed to framework-specific blueprints without losing generality.

A Blueprint deployment starts by sending the Resource Template to the Resource Discovery and Resource Selection components as indicated in Figure 2 (label 2 & 3). After the resources have been identified, they are used together with the information from the Blueprint and Service Catalogue to construct a Resourced Blueprint. Forming a Resourced Blueprint is done in the following manner.

The Resource Template is sent to the resource lifecycle management engine (OpenStack Heat as its default engine, but it is not limited to this choice). The modified Heat engine will carry out the actual *resource deployment* on the infrastructure. Some of the resource information (for example, the physical server on which a VM should be located) must be embedded into appropriate resource request API calls to the infrastructure management components, such as OpenStack's *nova-api* and *nova-scheduler*. The resource deployment process results in the return of a number of resource handlers (A resource handler can be a login account with username, access key, and IP address to a virtual machine, a container, a physical machine with a pre-installed operating system, or an existing HPC cluster). These resource handlers are sent to the service management framework, which, in turn, will use them to construct the Resourced Blueprint.

The Resourced Blueprint will then be given to the corresponding workflow/application lifecycle management framework to carry out the *application deployment* on the pre-provisioned resources. This process is shown in Figure 2 (label 6, 7, and 8).

The described service delivery model is more sophisticated than current self-service models using a vertical management approach. The application management and resource management operate independently. Nevertheless, this does not preclude the exchange of information between the application and resource management layers. For example, to terminate a Resourced Blueprint, a notification can be readily sent from the application management layer to the resource management layer, to free the underlying resources.

This approach has both advantages and disadvantages that must be carefully managed. A disadvantage is that cloud service provider has to manage a more complex system. However, this disadvantage can be offset by the cloud service provider having more control over resource utilization, service delivery, and optimal use of heterogeneous resources.

In contrast to existing frameworks, the proposed service delivery model will facilitate blueprint developers to specify comprehensive constraints and quality of service parameters for services. Based on the specified constraints and parameters, in contrast to existing solutions, can provide an initial optimal deployment of the resources. For example by creating/identifying resources on adjacent physical servers to minimize communication delay or by allocating containers with attached GPUs or Xeon Phis to balance performance and cost.

4 REALIZATION OF SEPARATION OF CONCERNS

The concept of the separation of concerns has been realized in the CloudLightning project (CloudLightning, 2015). In the service provider-consumer context, CloudLightning defines three actors including End-users (application/service consumers), Enterprise Application Operator/Enterprise Application Developer (EAO/EAD), and IaaS resource provider (CSP). These three actors represent three distinct domains of concern.

- For the end-user, the concerns are cloud application continuity, availability, performance, security, and business logic correctness;
- For the EAO/EAD, the concerns are cloud application configuration management, performance, load balancing, security, availability, and deployment environment;
- For the CSP, the concerns are resource availability, operation costs such as power consumption, resource provisioning, resource organization and partitioning (if applicable).

CloudLightning is built on the premise that there are significant advantages in separating these domains and the SDL has been designed to facilitate this separation. Inevitability, there will always be concerns that overlap the interests of two or more actors. This may require a number of actors to act together, for example, an EAO may need to configure a load-balancer and a CSP may need to implement a complementary host-affinity policy to realize high-availability. These overlapping concerns are managed by CloudLightning by providing vertical communications between the application lifecycle management and the resource management layers.

EADs/EAOs are responsible for managing the lifecycle of Resourced Blueprint at the application level, using frameworks such as Apache Brooklyn and OpenStack Solum. At the same time, the underlying resources are managed independently by the CloudLightning system. As a result, the following advantages accrue:

- Continuous improvement on the quality of the Blueprint services delivery;
- Reducing the time to start a service and hence improve the user experience by reusing resources that have already been provisioned;
- Resource optimizations and energy optimizationl
- Creating a flexible and extensible integration with other management frameworks such as the OpenStack Mistral workflow management system.

In CloudLightning, the functional components that realize the concept of the ”Separation of Concerns” is shown in Figure 3. The components responsible for application lifecycle management includes

- Blueprint: is used to represent specific application parameters, constraints and metrics defined by users, identifying services and their relationships.
- Resourced Blueprint: represents a fully qualified Cloud Application Management Platform (CAMP) (Jacques Durand and Rutt, 2014) document with specific resource handlers.
- Service Catalogue: is a persisted collection of versioned services, each of which includes service identification, application deployment mechanism and resource specification.
- Blueprint Decomposition Engine: handles the transformation of Blueprints to Resourced Blueprints according to provided requirements.
- Brooklyn: is used for deploying and managing the applications via Resourced Blueprints.

The components responsible for resource lifecycle management includes

- Infrastructure Management: a set of resource management frameworks for managing varying hardware resources.
- Service Orchestration Template: describes the infrastructure resource (such as servers, networks, routers, floating IPs, volume, etc.) for a cloud application, as well as the relationships between resources.
- Service Orchestration Interface: automatically generate HOT template in terms of the results from Resource Allocation Strategy component, or dynamically modify HOT template based on the results from the Continuous Improvement component.

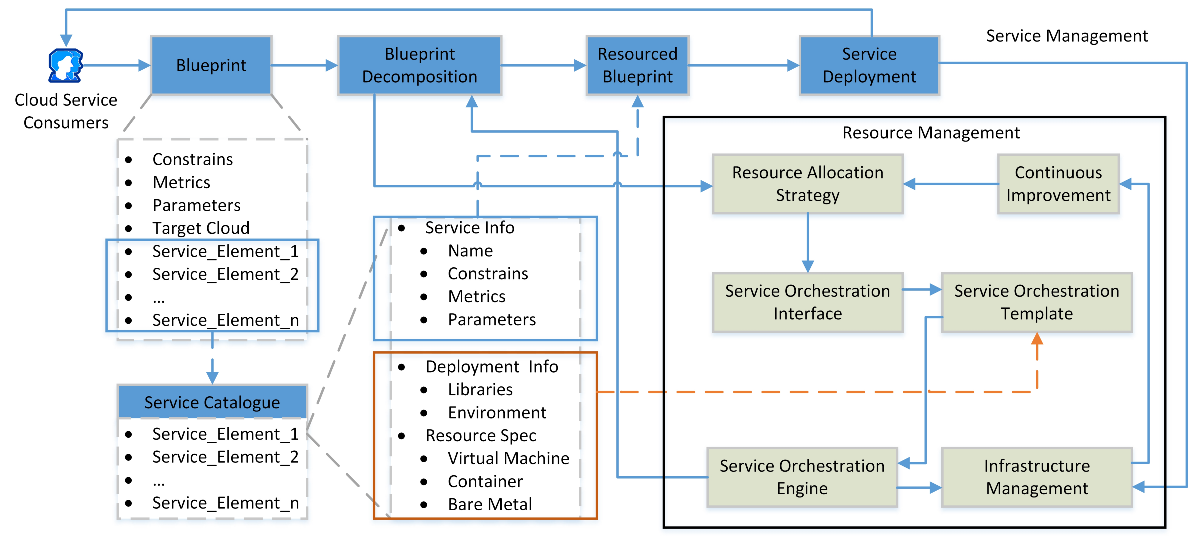

Figure 3: Realization of "Separation of Concerns" The Blueprint Decomposition Engine handles the transformation of Blueprints to Resourced Blueprints. Brooklyn is used for deployment and application lifecycle management. The CloudLightning identifies and creates/allocates optimal Resources for applications. The Heat Orchestration Template (HOT), specific to CloudLightning system, describes resources and their relationships. The Heat Engine manages the resource lifecycle. The Continuous Improvement component together with Heat and telemetry attempts to perform continuous resource improvement over the lifetime of the deployed blueprint.

- Service Orchestration Engine: responsible for resource lifecycle management.
- Continuous Improvement: this management component together with Heat and telemetry continuously improve the quality of the deployed blueprint during its lifetime.

The approach taken by the CloudLightning project is to try to re-align the evolving heterogeneous cloud with the Services Oriented Architecture of the homogeneous cloud. The first step in this process is to establish a clear services interface between the service consumer and the service provider. The essence of this interface is the establishment of a separation of concerns between the consumer and the provider. Thus, consumers should only be concerned with *what* they want to do, and providers should be concerned only with *how* that should be done. This simple step removes all direct consumer interaction with the provider's infrastructure and returns control back to the provider. In this view, various service implementation options are assumed to already exist and the consumer no longer has to be an expert creator of those service implementations. Consumers should not have to be aware of the actual physical resources being used to deliver their desired service, however, given the fact that multiple diverse implementations may exist for each service (each on a different hardware type, and each characterized by different price/performance attributes) consumers should be able to distinguish and choose between these options based on service delivery attributes alone. Service creation, in the approach proposed here, remains a highly specialized task that is undertaken by an expert. An expert will create a service solution for a specific hardware platform, will profile that service and will register the service executable and meta information with the CloudLightning system.

5 CONCLUSION

This paper concentrates on providing a separation of concern between application lifecycle management and resource management to maximize user experience and cloud efficiency on each side. A realization of separation of concern was developed, a collaborative interaction between the Brooklyn system and the OpenStack Heat, component was designed and integrated into the CloudLightning architecture. A coherent framework for the concept of separation of concerns will be provided in the future work.

ACKNOWLEDGMENT

This work is funded by the European Unions Horizon 2020 Research and Innovation Programme through the CloudLightning project under Grant Agreement Number 643946.

REFERENCES

Brooklyn, A. (2017). https://brooklyn.apache.org/. [Accessed on 16-February-2017].

CloudLightning (2015). http://cloudlightning.eu/. [Accessed on 16-February-2017].

Heat (2017). https://github.com/openstack/heat. [Accessed on 10-February-2017].

Jacques Durand, Adrian Otto, G. P. and Rutt, T. (2014). Cloud application management for platforms version 1.1. In *OASIS Committee Specification 01*.

jCloud (2017). https://jclouds.apache.org. [Accessed on 05-February-2017].

Libcloud (2017). https://libcloud.apache.org. [Accessed on 07-February-2017].

Solum (2017). https://github.com/openstack/solum. [Accessed on 01-February-2011].