International Journal of Science Engineering and Advance Technology

Gusts of Alteration From Hawker To Meta Cloud

N.Srinath Reddy#1, G Sudhakar#2#1
Department Of Cse, Pbr Visvodaya Institute Of Technology & Science, Kavali.
#2student Of M.Tech(C.S) And Department Of Cse, Pbr Visvodaya Institute Of Technology & Science,Kavali.

ABSTRACT:

The emergence of yet more cloud offerings from a multitude of service providers calls for a meta cloud to smoothen the edges of the jagged cloud landscape. This meta cloud could solve the vendor lock-in problems that current public and hybrid cloud users face. The cloud computing paradigm has achieved widespread adoption in recent years. Its success is due largely to customers' ability to use services on demand with a pay-as-you go pricing model, which has proved convenient in many respects. Low costs and high flexibility make migrating to the cloud compelling. Despite its obvious advantages, however, many companies hesitate to "move to the cloud," mainly because of concerns related to service availability, data lock-in, and legal uncertainties. Lock in is particularly problematic. For one thing, even though public cloud availability is generally high, outages still occur. Businesses locked into such a cloud are essentially at a standstill until the cloud is back online.

Key words- meta cloud, gusts, data lock in.

1. INTRODUCTION

At the core of all these problems, we can identify a need for businesses to permanently monitor the cloud they're using and be able to rapidly "change horses" — that is, migrate to a different cloud if they discover problems or if their estimates predict future issues. However, migration is currently far from trivial. Myriad cloud providers are flooding the market with a confusing body of services, including compute services such as the Amazon Elastic Compute Cloud (EC2) and VMware vCloud, or key-value stores, such as the Amazon Simple Storage Service (S3). Some of these services are conceptually comparable to each other, whereas others are vastly different, but they're all, ultimately, technically incompatible and follow no standards but their own. To further complicate the situation, many companies not (only) build on public clouds for their cloud computing needs, but combine public offerings with their own private clouds, leading to socalled hybrid cloud setups.4 Here, we introduce the concept of a meta cloud that incorporates design time and runtime components. This meta cloud would abstract away from existing offerings' technical incompatibilities, thus mitigating vendor lock-in. It helps users find the right set of cloud services for a particular use case and supports an application's initial deployment and runtime migration.

Cloud Computing Use Case

Let's consider a Web-based sports portal for an event such as the Olympic Games, which allows users to place bets. An event this large requires an enormously efficient and reliable infrastructure, and the cloud computing paradigm provides the necessary flexibility and elasticity for such a scenario. It lets service providers handle short-term usage spikes without needing res- pective dedicated resources available continuously. The problem, however, is that once an application has been developed based on one particular provider's cloud services and using its specific API, that application is bound to that provider; deploying it on another cloud would usually require completely redesigning and rewriting it. Such vendor lock-in leads to strong dependence on the cloud service operator. In the sports portal example, in addition to the ability to scale applications up and down by dynamically allocating and releasing resources, we must consider additional aspects, such as resource costs and regional communication bandwidth and latency. Let's assume the sports betting portal application is based on a load balancer that forwards HTTP requests to numerous computing nodes hosting a Web application that lets users submit a bet. Request handlers place bet records in a message queue and subsequently store them in a relational database. Let's further assume a service provider realizes this scenario using only Amazon Web Services (AWS), EC2 to host applications, Simple Queue Service (SQS) as its cloud message queue, and the Relational Database Service (RDS) as

a database system. Instead of being bound to one cloud operator, however, the betting application should be hosted in an optimal cloud environment.
To leverage a more diverse cloud landscape, support flexibility, and avoid vendor lockin, the meta cloud must achieve two main goals:
•find the optimal combination of cloud services for a certain application with regard to QoS for users and price for hosting; and develop a cloud-based application once, then run it anywhere, including support for runtime migration.
Lately, the meta cloud idea has received some attention, and several approaches try to tackle at least parts of the problem.

Current Weather in the (Meta) Cloud

First, standardized programming APIs must enable developers to create cloud-neutral applications that aren't hardwired to any single provider or cloud service. Cloud provider abstraction libraries such as libcloud (http:// libcloud.apache.org), fog (http://fog. io), and jclouds (www.jclouds.org) provide unified APIs for accessing different vendors' cloud products. Using these libraries, developers are relieved of technological vendor lockin because they can switch cloud providers for their applications with relatively low overhead.

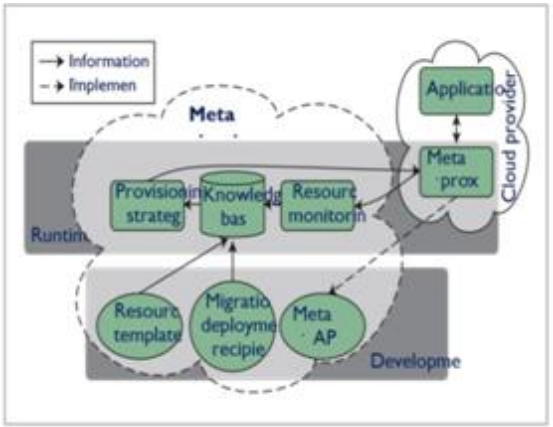
As a second ingredient, the meta cloud uses resource templates to define concrete features that the application requires from the cloud. For instance, an application must be able to specify that it requires a given number of computing resources, Internet access, and database storage. Some current tools and initiatives — for example, Amazon's Cloud Formation or the upcoming TOSCA specification are working toward similar goals and can be adapted to provide these required features for the meta cloud.

In addition to resource templates, the automated formation and provisioning of cloud applications also depends on sophisticated features to actually deploy and install applications automatically. Predictable and controlled application deployment is a central issue for cost effective and efficient deployments in the cloud, and even more so for the meta cloud. Several application provisioning solutions exist, enabling developers and administrators to declaratively specify deployment artifacts and dependencies to allow for repeatable and managed resource
provisioning.

At runtime, an important aspect of the meta cloud is application monitoring, which enables the meta cloud to decide whether it's necessary to provision new instances of the application or migrate parts of it. Various vendors provide tools for cloud monitoring, ranging from systemlevel monitoring (such as CPU and bandwith) to application-level monitoring (Amazon's CloudWatch; http://aws.amazon.com/cloudwatch/) to SLA monitoring (as with monitis; http://portal.monitis.com/index.
php/cloudmonitoring). However, the meta cloud requires more sophisticated monitoring techniques and, in particular, approaches for making automated provisioning decisions at runtime based on current application users' context and location.

Meta Cloud API

The meta cloud API provides a unified programming interface to abstract from the differences among provider API
implementations. For customers, using this API specific cloud service offering.



The meta cloud API can build on available cloud provider abstraction APIs, as previously mentioned. Although these deal mostly with keyvalue stores and compute services, in principle, all services can be covered that are abstract enough for more than one provider to offer and whose specific APIs don't differ too much, conceptually.resource templates in different projects. Using the DSL, developers model their application components and their basic runtime requirements, such as (providerindependently normalized) CPU, memory, and I/O capacities, as well as dependencies and weighted communication relations between these components. The provisioning strategy uses the weighted component relations to determine the application's optimal

deployment configuration. Moreover, resource templates allow developers to define constraints based on costs, component proximity, and geographical distribution.

Migration and Deployment RecipesDeployment recipes are an important ingredient for automation in the meta cloud infrastructure. Such recipes applications using meta cloud development components. The meta cloud runtime abstracts from provider specifics using proxy objects, and automates application life-cycle management. service providers and the particular services they provide, including response time, availability, and more service-specific quality statements.

Provisioning Strategy

The provisioning strategy component primarily matches an application's cloud service requirements to actual cloud service providers. It finds and ranks cloud services based on data in the knowledge base. The initial deployment decision is based on the resource templates, specifying the resource requirements of an application, together with QoS and pricing information about service providers. The result is a list of possible cloud service combinations ranked according to expected QoS and costs. At runtime, the component can reason about whether migrating a resource to another resource provider is beneficial based on new insights into the application's behavior and updated cloud provider QoS or pricing data. Reasoning about migrating also involves calculating migration costs. Decisions about the provisioning strategy result in the component executing customer-defined deployment or migration scripts.

Knowledge Base

The knowledge base stores data about cloud provider services, their pricing and QoS, and information necessary to estimate migration costs. It also stores customerprovided resource templates and migration or deployment recipes. The knowledge base indicates which cloud providers are eligible for a certain customer. These usually comprise all providers the allow for controlled deployment of the application, including installing packages, starting required services, managing package and application parameters, and establishing links between related components. Automation tools such as Opscode Chef provide an extensive set of functionalities that are directly integrated into the meta cloud environment. Migration recipes go one step further and describe how to migrate an application during runtime — for example, migrate storage functionality from one service provider to another. Recipes only describe initial deployment and migration; the provisioning strategy and the meta cloud proxy execute the actual process using the aforementioned automation tools.

Meta Cloud Proxy

The meta cloud provides proxy objects, which are deployed with the application and run on the provisioned cloud resources. They serve as mediators between the application and the cloud provider. These proxies expose the meta cloud API to the application, transform application requests into cloud-providerspecific requests, and forward them to the respective cloud services. Proxies provide a way to execute deployment and migration recipes triggered by the meta cloud's provisioning strategy.

Moreover, proxy objects send QoS statistics to the resource monitoring component running within the meta cloud. The meta cloud obtains the data by intercepting the application's calls to the underlying cloud services and measuring their processing time, or by executing short benchmark programs.
Applications can also define and monitor custom QoS metrics that the proxy objects send to the resource monitoring component to enable advanced, application-specific management strategies. To avoid high load and computational bottlenecks, communication between proxies and the meta cloud is kept at a minimum. Proxies don't run inside the meta cloud, and regular service calls from the application to the proxy aren't routed through the meta cloud, either.

Resource Monitoring

On an application's request, the resource monitoring component receives data collected by meta cloud proxies about the resources they're using. The component filters and processes these data and then stores them on the knowledge base for further processing. This helps generate comprehensive QoS information about cloud customer has an account with and providers that offer possibilities for creating (sub)accounts on the fly. Several information sources contribute to the knowledge base: meta cloud proxies regularly send data about application behavior and cloud service QoS.

For initial deployment, the developer submits the application's resource template to the meta cloud. It specifies not only the three types of cloud services needed to run the sports application, but also their necessary properties and how they depend on each other. For compute resources, for instance, the developer can specify CPU, RAM, and disk space according to terminology defined by the meta cloud resource template DSL. Each resource can be named in the template, which allows for referencing during deployment, runtime, and migration. The resource template specification should also contain interdependencies, such as the direct connection between the Web service compute instances and the message queue service.

The rich information that resource templates provide helps the provisioning strategy component make profound decisions about cloud service ranking. We can explain the working principle for initial deployment with a Web search analogy, in which resource templates are queries and cloud service provider QoS and pricing information represent indexed documents. Algorithmic aspects of the actual ranking are beyond this article's scope. If some resources in the resource graph are only loosely coupled, then the meta cloud will be more likely to select resources from different cloud providers for a single application. In our use case, however, we assume that the provisioning strategy ranks the respective Amazon cloud services first, and that the customer follows this recommendation.

Cloud-centric migration makes the meta cloud infrastructure responsible for most migration aspects, leading to issues with application specific intricacies, whereas in application-centric migration, the meta cloud only triggers the migration process, leaving its execution mostly to the application. We argue that the meta cloud should control the migration process but offer many interception points for applications to influence the process at all stages

Conclusion

The meta cloud can help mitigate vendor lock-in and promises transparent use of cloud computing services. Most of the basic technologies necessary to realize the meta cloud already exist, yet lack integration. Thus, integrating these stateofthe-art tools promises a huge leap toward the meta cloud. To avoid meta cloud lockin, the community must drive the ideas and create a truly open meta cloud with added value for all customers and broad support for different providers and implementation technologies.

References

1.M. Armbrust et al., "A View of Cloud Computing," Comm. ACM, vol. 53, no. 4, 2010, pp. 50–58.
2.B.P. Rimal, E. Choi, and I. Lumb,
"A Taxonomy and
3.Survey of Cloud Computing Systems," Proc. Int'l Conf. Networked Computing and Advanced Information Management, IEEE CS Press, 2009, pp. 44–51.
4.J. Skene, D.D. Lamanna, and W. Emmerich, "Precise

5.Service Level Agreements," Proc. 26th Int'l Conf. Software Eng. (ICSE 04), IEEE CS Press, 2004, pp. 179– 188.
6.Q. Zhang, L. Cheng, and R. Boutaba, "Cloud Computing:Stateof-the-Art and Research
Challenges," J. Internet Services and Applications, vol. 1, no. 1, 2010, pp. 7–18.
7.M.D. Dikaiakos, A. Katsifodimos, and G. Pallis,
8."Minersoft: Software Retrieval in
Grid and Cloud
9.Computing Infrastructures," ACM Trans. Internet Technology, vol. 12, no. 1, 2012, pp. 2:1–2:34.

N. SRINATH
REDDY ,VICE PRINCIPAL,
Department of CSE,
PBR VISVODAYA INSTITUTE OF
TECHNOLOGY &SCIENCE,
KAVALI.

G SUDHAKAR,
M.TECH (CS) STUDENT,
Department of CSE, PBR VISVODAYA INSTITUTE OF TECHNOLOGY &SCIENCE,
KAVALI.