Provider Independent Model For Developing Cloud Applications

Suja P. Mathews, MCA, Mphil
Associate Professor
Jyoti Nivas College, Bangalore

Sunu George, MCA, Mphil
Department of Computer Science
Assumption HSS Varandarappilly, Thrissur

Abstract

Enterprises are increasingly looking at reducing IT capital and operating expenses and the Cloud Computing paradigm is an ideal platform for achieving this. Cloud Computing is a paradigm shift from the traditional in-house infrastructure setup to a shared and dynamically provisioned computing infrastructure, which also provides on-demand scaling. Cloud provides a pay-as-you-go model that offers computing resources as a service which significantly cuts on IT capital expenses and enables control the operating expenses effectively. A major issue which is a deterrent to this move is that the current application architectures does not have the necessary elements to address elasticity, virtualization and payment. The cloud applications should be designed considering these elements. Further, there is no generic cloud software architecture for designing and building applications utilizing the capabilities of the cloud. To top it all, each cloud service provider follows different standards which dictate how the applications should be written for each platform/provider. This essentially binds the cloud applications and users to a particular provider, since switching becomes very expensive without the software being designed to be portable. This paper will focus on defining a model for developing applications that are provider agnostic, and also presents the main cloud design elements. It also shows the set of configuration rules and the semantic interpretation. It provides an abstract architecture of the system which is important to tackle platform specific issues later. This separation of concerns allows for better maintainability, and facilitates applications portability.

1. Introduction

Cloud computing is a general term for anything that involves delivering hosted services over the Internet. These services are broadly divided into three categories: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). Cloud Services are sold on demand, its elastic, and fully managed by the provider. Innovations in virtualization and distributed computing, and ever-improving access to high speed internet have accelerated interest in cloud computing.

The economics of cloud computing has several driving factors like the pricing models of the service providers, the fluctuating business demands and the high cost of switching between providers. The pricing models and provider switchability depends on the service model. IaaS provides a simple virtual server instance, PaaS provides a set of software and product development tools or APIs on the provider's infrastructure and SaaS provides the server infrastructure, the software products and APIs, and also a front end portal for the end user. SaaS hosted services can be anything from web-based email (Google Mail for enterprises) to inventory control and database processing.

When a end customer uses a PaaS or SaaS cloud infrastructure provided by a service provider, switching to another provider involves re-implementing several layers of the software to delink from the current providers APIs and move on to using the new provider's APIs and services. This may even impact the workflows defined within the companies, and the cost of switching becomes very high.

While there are several compelling use cases – like elastic infrastructure, pay as you use, load spikes handling, extremely low upfront capital expenditure and risk mitigation of underutilization or under provisioning - that favour cloud computing, the above mentioned factor is a major block in migration to cloud by several companies.

This paper addresses the issue of vendor lock-in by proposing a model to standardize cloud

platform (provider) independent application development and inter-operability between different cloud platforms. Some of the cloud development environments like Microsoft Azure and Google Cloud do share some common components which can also be leveraged to ensure the inter-operability. The aim of this paper is to define a model, a high level architecture and the related design pattern. This should enable cloud users to design "cloud platform independent" applications without sacrificing the benefits of cloud infrastructure.

3. Need for Provider Independent Model

In this section we will investigate the need for this model, and identify the differences between this model and the Service-oriented-architecture (SOA) reference model.

SOA is an umbrella that describes any kind of service. A cloud application is a service. A cloud application reference model is a SOA model that conforms to the SOA meta-model. This makes cloud applications SOA applications. A cloud application is a SOA application that runs under a specific environment, which is the cloud computing environment (platform). This environment is characterized by horizontal scalability, rapid provisioning, ease of access, and flexible prices. While SOA is a business model that addresses the business process management, cloud architecture addresses many technical details that are environment specific, which makes it more a technical model.

Cloud platforms are complex environments, which need to be refined at different levels of granularity. The cloud hierarchical view (i.e. SaaS, PaaS, IaaS) is an example of a refinement that uses SOA to describe the high level services provided over the internet (the cloud). There is a need to create a modeling language that is tailored to build efficient, elastic and autonomous applications from tasks and services provided by the cloud environment, and to define patterns that can result in the efficient optimization of money and resources.

4. Model for Provider independent cloud applications

The core of the architecture is formed using a composable CloudJob unit, which consists of a set of actions. These actions utilize services to provide functionalities to meet a requirement. The CloudJob is mutable and can be replicated to multiple Virtual Machines to enable scalability. The Jobs should be stateless, have very little coupling, should be modular and should have semantic inter-operability. The Jobs have semantic connections to other jobs in the cloud through the Roles they play to meet a functionality/requirement bounded by Responsibilities.

CloudJobs can be uniquely identified using a DNS name provided by a global Dynamic Name Service (DDNS) service at run time. Such names assigned to the corresponding Virtual Machines makes the *Job* highly available and fault tolerant. This also enables the cloud application to be upgraded dynamically without interruptions.

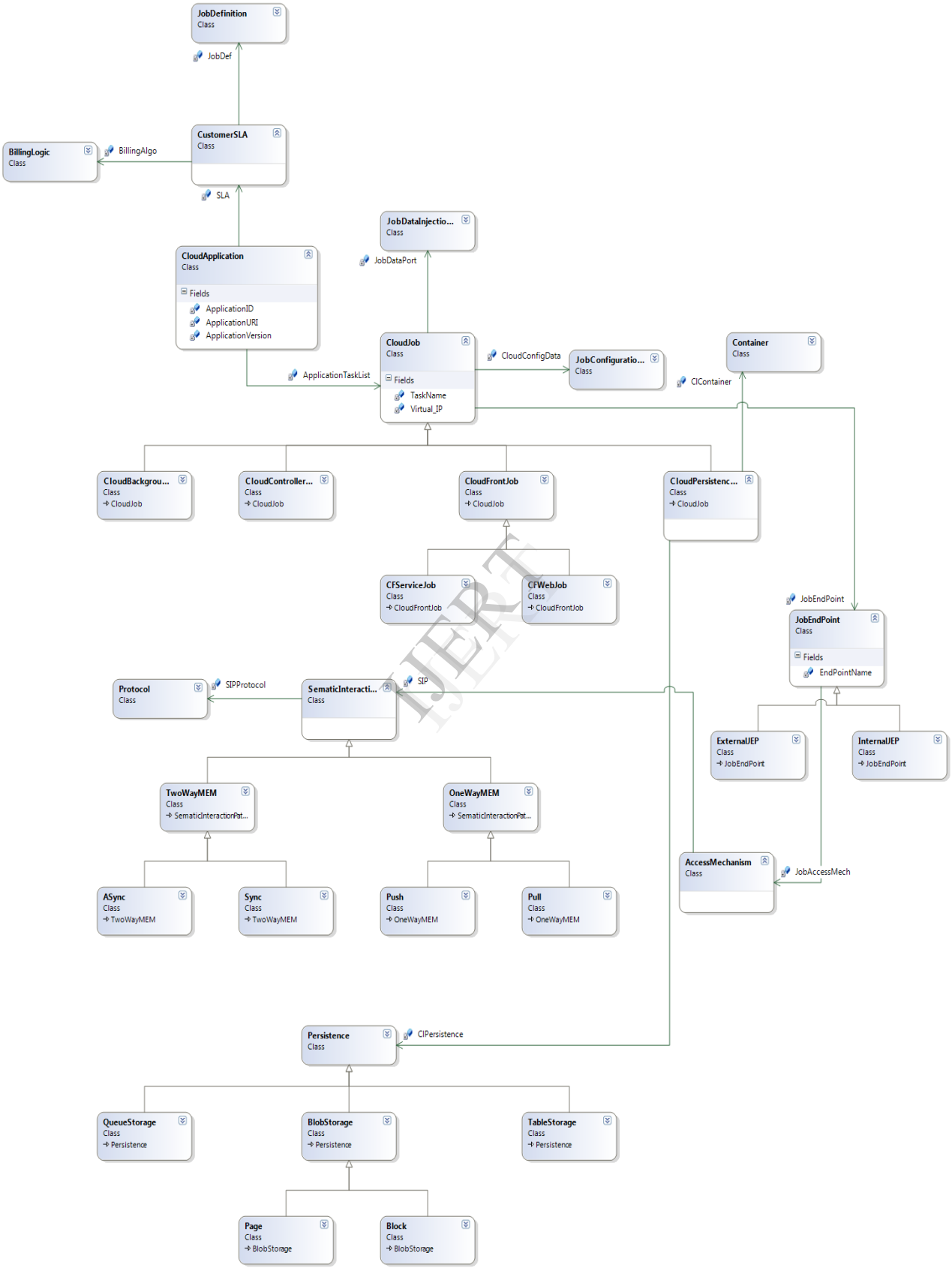
JobDefinition
Class
JobDef
CustomerSLA
Class
BillingLogic
Class
BillingAlgo
SLA
JobDataInjectio...
Class
JobDataPort
CloudApplication
Class
Fields
ApplicationID
ApplicationURI
ApplicationVersion
CloudJob
Class
Fields
TaskName
Virtual_IP
ApplicationTaskList
CloudConfigData
JobConfiguratio...
Class
Container
Class
CIContainer
CloudBackgrou...
Class
CloudJob
CloudController...
Class
CloudJob
CloudFrontJob
Class
CloudJob
CloudPersistenc...
Class
CloudJob
CFServiceJob
Class
CloudFrontJob
CFWebJob
Class
CloudFrontJob
JobEndPoint
JobEndPoint
Class
Fields
EndPointName
Protocol
Class
SIPProtocol
SematicInteracti...
Class
SIP
ExternalJEP
Class
JobEndPoint
InternalJEP
Class
JobEndPoint
TwoWayMEM
Class
SematicInteractionPat...
OneWayMEM
Class
SematicInteractionPat...
AccessMechanism
Class
JobAccessMech
ASync
Class
TwoWayMEM
Sync
Class
TwoWayMEM
Push
Class
OneWayMEM
Pull
Class
OneWayMEM
Persistence
Class
CIPersistence
QueueStorage
Class
Persistence
BlobStorage
Class
Persistence
TableStorage
Class
Persistence
Page
Class
BlobStorage
Block
Class
BlobStorage

Fig 1. Architecture for the provider independent model

Every *CloudJob has a* definition file. This definition file – *JobDefinition* – contains information about jobs that the cloud application provides. A *JobDefinition* contains information about the jobs in the cloud application, which are determined at design time. A *JobDefinition* provides the structure of the cloud application, in terms of the provided jobs, their types and relationships between jobs, in addition to a set of job interfaces and their contracts.

Elasticity is a key differentiator between a normal application and a cloud application. Cloud applications must be able to scale up and down seamlessly. This is typically achieved by replicating jobs to several virtual machines at runtime depending on the dynamics of work demand. *JobConfigurationData* is where dynamic aspects of cloud application are determined at runtime. The running application need not be stopped or redeployed to modify the information in the *JobConfigurationData* file.

JobConfigurationData contains information such as the size of the virtual machine (*VM_Size*), number of instances of virtual machines (*VM_InstanceCount*), the database size (*VM_DBSize)* and internet *BandWidth.* It could also contain the location (*LocationProximity)* where the job instances are to be executed and whether they belong to the same affinity group or not.

Another key parameter is the pricing models. Every cloud platform provider have their own unique pricing models. A typical model is pay-as-you-use based model or variations of this. In some variations, the way in which resources are allocated varies based on the amount of money (or slabs). It is also possible for a cloud application developer to allocate resources explicitly. The cloud user can set such parameters and values in the *JobConfigurationData* file based on need and budget. Another variant is where the providers use algorithms *(BillingLogic)* to dynamically allocate resources based on the cloud user's budget. Instead of setting the load parameters in the configuration file, the user sets the budget and usage guidelines and the provider automatically sets the values in the *JobConfigurationData* file to achieve the best configuration. This *JobConfigurationData* configuration file becomes a contract with the cloud user (application developer), and this is represented in the form of a Service Level Agreement (SLA).

Job properties can also be modified at runtime and this is achieved through a *JobDataInjectionPort.* A *JobDataInjectionPort* modifies tasks crosscutting properties such as those related to quality of service (QoS). Cloud platform providers vary in the way they support job modification. This is risky and a source of security breaches, and hence the level of support varies across providers.

4.1 Classification of *CloudJobs*

CloudJobs can be classified as *CloudFrontJobs, CloudBackgroundJobs, CloudControllerJobs and CloudPersistenceJobs.* Each of the job types are explained in details below.

4.1.1 *CloudFrontJob*

CloudFrontJob is an an entry point to the cloud application that can handle user requests and distributed by a load balancer. A *CloudFrontJob* should support interactive request-response pattern. It is typically a web application (CFWebJob) hosted on the cloud data centre where a web-server is running. It can also be a web-service (CFServiceJob) provided by the service provider or third party. A *ServiceJob* uses the Enterprise Service Bus (EBS) to discover and access remote or enterprise services.

4.1.2 ***CloudBackgroundJob***

CloudBackgroundJob is a background job of the CloudFrontJob on the cloud data centre. It is not directly accessible from outside the cloud data centre and does general development work and supports other jobs by performing a particular functionality. The CloudBackgroundJob should support event driven messaging and communication patterns. A typical example of CloudBackgroundJob is Grid computing.

4.1.3 CloudControllerJob

This task manages aspects cutting across the cloud, such as those related to monitoring cloud resources, which includes computing and storage instances and a load balancer to ensure resource utilization and performance. It also provides for logging, maintaining QoS of the cloud application, deployments of application, dynamically add/remove instances based on metrics, launching instances, login to instances, and job properties changes through the *JobDataInjectionPort. CloudControllerJobs* can be accessed directly through a web portal or a specific

API (i.e. REST, SOAP). Communication with *CloudControllerJobs* should be secure by using certificate based SSL over HTTP or public key algorithms.

4.1.4 CloudPersistenceJob

Managing storage accounts is the main role of *CloudPersistenceJobs.* They *manage* the access control and login to cloud storages. A cloud storage (e.g., blob, table, queue) does not have any access control mechanism. The persistence job is responsible for providing the authorization and authentication services. *CloudPersistenceJobs* create containers, which are similar to folders but with no nesting (multiple level hierarchy). Containers can be accessed through a unique Uniform Resource Identifier (URI). *CloudPersistenceJobs* assign persistency to containers and give them a unique URI that is either privately or publicly accessible. The *CloudPersistenceJob* supports three main types of cloud storages that are reliable, scalable, simple, inexpensive and have better performance under the cloud environment. These types are: unstructured data (BlobStorage), structured data (TableStorage) and asynchronous messaging (QueueStorage).

BlobStorage: Blobs are unstructured large data files and their meta-data. It can be stored as a sequence of blocks or pages. The BlobStorage is the simplest and largest cloud storage unit. Cloud drive storages are blobs.

TableStorage: Tables are structured data files, that are more complex than blobs, but different than relational database tables. Cloud tables are simpler and make them suitable for huge scalability to support any number of simultaneous tasks. A cloud table consists of a set of entities and its associated properties. The cloud table uses two types of keys: partition keys and row keys. They do not support SQL queries, have no schema and use optimistic concurrency for updates and deletions. Cloud tables are mostly similar to data sheet tables.

QueueStorage: This is a scalable messages storage, which supports the polling based model used in message passing between tasks. A message can be stored for long periods (i.e. days) before it is read and then removed from the queue. Cloud queues are different from conventional queuing systems and the major differences are given below. It must support fault tolerance. When a message read is read from the queue, it does not delete the message from the queue, which is unlike conventional queues. The message will be in hidden mode until it is successfully processed. The processing job must delete the message after successful processing. The QueueStorage is the main communication mechanism between *CloudFrontJobs* and *CloudBackgroundJobs,* and this makes it one of the most frequently used design patterns in the cloud. The main advantages of this pattern is that the end-user need not wait a long time for the job to process the message, and also makes the scalability much easier.

Relationships between jobs are determined by *JobEndPoints. JobEndPoints* are ports through which a *CloudJob* connects to other jobs or to the environment. Each CloudJob has one or more *JobEndPoints.* An *JobEndPoint* can be classified based on several criteria like - whether it is publicly visible (external) or only accessible within the Cloud Application (internal), load balanced at the network level or not, or whether it allows inbound or outbound communication. Each *JobEndPoint* uses an access mechanism, which uses a semantic

interaction pattern for the coordination of message exchange. These patterns are built using specific protocols that determine the syntax and semantics of the messages that are exchanged between the two communication parties.

4.2 Message Exchange Mechanisms

Message Exchange Mechanisms (MEM) can be classified into two main categories, one-way or two-way. The one-way MEM is referred to as the event driven MEM, or publish subscribe (pub/sub), in which the participating parties are not fully aware of each other. A temporary storage in the form of a queue is used to accomplish this. One party will push a message, and the second will pull it from the queue. This is one of the common communication mechanisms between *CloudFrontJobs* and *CloudBackgroundJobs*. On the other hand, the two-way MEM is usually referred to as request/response MEM. It can be either synchronous (blocking) or asynchronous (non-blocking). This is an interactive communication that is usually needed when there is a direct interaction with the user. *CloudFrontJobs* must support this type of interaction with the application user.

5. Conclusion

This paper presented a cloud platform provider independent model for building cloud applications. Cloud computing is a new paradigm for developing elastic and flexible applications with less time to market. The motivation is to reduce the overhead of developing, configuring, deploying, and maintaining cloud applications. Currently, there is no common vocabulary, development methodologies, or best practices that distinguish the cloud development paradigm from the existing ones. There are practically no standardization and common terminologies to enable portability and migration between different cloud platforms. The lack of software architectural models and patterns makes cloud application development an ad-hock approach, which is almost entirely driven by the APIs and design of services provided by a particular vendor. Switching to another provider essentially means rebuilding a major portion of the cloud application.

In this paper we defined a model that is capable of capturing the syntax and some of the semantics of cloud applications. This model can be used by developers to better understand cloud applications independent of any specific cloud development environment. This model is expected to serve as a first step towards building a cloud modelling language.

Future directions include refining the syntax and defining semantics of the proposed model, mapping the proposed model to different cloud platforms, and creating a modelling language for building service provider independent cloud applications.

6. Acknowledgement

If words are considered as symbols of approval and tokens of Acknowledgement, then let the words play the heralding role of expressing our gratitude.

First and foremost, we are thankful to the Almighty, for his blessings in the successful completion of this paper.

We would like to express our sincere and hearty thanks to Mr. M J Samuel, M.S BITS Pilani, Engineering Manager, Celstream Technologies, for his splendid help for the successful completion of our paper.

REFERENCES

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. (2009). Above the clouds: A berkeley view of cloud computing. *EECS Department, Univer- sity of California, Berkeley, Tech. Rep. UCB/EECS- 2009-28.*

CA Labs (2009). Cloud computing Web-Services offering and IT management aspects. In *OOPSLA09, 14th con- ference companion on Object Oriented Programming Systems Languages and Applications*, pages 27–39.

Charlton, S. (2009). Model driven design and operations for the cloud. In *OOPSLA09, 14th conference companion on Object Oriented Programming Systems Languages and Applications*, pages 17–26.

Frey, S. and Hasselbring, W. (2010). Model-Based migra- tion of legacy software systems into the cloud: The CloudMIG approach. In *WSR2010, 12th Workshop Software-Reengineering*, pages 1–2.

Google (2010). Google app engine. Retrieved: December 2010, from http://code.google.com/appengine/.

Matthews, C., Neville, S., Coady, Y., McAffer, J., and Bull, I. (2009). Overcast: Eclipsing high profile open source cloud initiatives. In *OOPSLA09, 14th con- ference companion on Object Oriented Programming Systems Languages and Applications*, pages 7–15.

Maximilien, E. M., Ranabahu, A., Engehausen, R., and An- derson, L. C. (2009). Toward cloud-agnostic middle- wares. In *OOPSLA09, 14th conference companion on Object Oriented Programming Systems Languages and Applications*, pages 619–626.

Microsoft (2010). Windows azure microsoft's cloud ser- vice platform. Retrieved: December 2010, from http://www.microsoft.com/windowsazure/.

Sirtl, H. (2008). Software plus Services: New IT-and Business Opportunities by Uniting SaaS, SOA and Web 2.0. In *IEEE EDOC'08, 12th International En- terprise Distributed Object Computing Conference*, pages 1541–7719.

Tsai, W., Sun, X., and Balasooriya, J. (2010). Service- Oriented Cloud Computing Architecture. In *ITNG10, 7th International Conference on Information Technol- ogy: New Generations*, pages 684–689.

Zhang, L. J. and Zhang, J. (2009). Architecture-Driven vari- ation analysis for designing cloud applications. In *IEEE CLOUD09, 2nd International Conference on Cloud Computing*, pages 125–134.

Zhang, W., Berre, A. J., Roman, D., and Huru, H. A. (2009). Migrating legacy applications to the service cloud. In *OOPSLA09, 14th conference companion on Object Oriented Programming Systems Languages and Ap- plications*, pages 59–68.