Launching Virtual Machine in OpenNebula and Ensuring Accountability in Cloud Data

Midhu Babu[1], A. M. J Muthu Kumaran[2]

[1]Department of Computer Science and Engineering, SRM University, Chennai, India
midhumtech13@gmail.com

[2]Assistant Professor, Dept of Computer Science and Engineering, SRM University, Chennai, India
muthu.a.@ktr.srmuniv.ac.in

Abstract : ***The paper aims to launch a virtual machine in the OpenNebula cloud environment and ensuring accountability in cloud data. OpenNebula interoperability makes cloud an evolution by leveraging existing IT infrastructure. Cloud computing enables highly scalable services to be easily consumed over the internet on an as-needed basis. A major feature of the cloud services is that user's data are usually processed remotely in unknown machines that users do not own or operate. While user fears of losing control of their own data can become a significant barrier to the wide adoption of cloud services. To address this problem, we propose a novel highly decentralized information accountability framework to keep track of the actual usage of the user's data in the cloud. In particular, propose an object-centered approach that enables enclosing our logging mechanism together with user's data and policies. It leverage the JAR programmable capabilities to both create a dynamic and traveling object, and to ensure that any access to user's data will trigger authentication and automated logging local to the JARs. Creation of instance into that openNebula infrastructure would be more flexible to the organization.***

Keywords: *Accountability, cloud computing, data sharing, openNebula, VMware*

1. Introduction

Cloud computing has raised a wide range of important privacy and security issues. Such issues are due to the fact that in the cloud, user's data and applications reside at least for a certain amount of time on the cloud cluster which is owned and maintained by a third party. In previous works the cloud is deployed by using Eucalyptus or XCP (Xen Cloud Platform). XCP does not provide the overall cloud architecture, but rather focuses on configuration and maintenance of clouds. Eucalyptus was designed to provide services compatible with Amazon's EC2 cloud only. Interestingly all communication is made through Web Services standards. The solution requires third-party services to complete the monitoring and focuses on lower level monitoring of system resources. This paper focus on to create a virtual windows instance using openNebula and then provide an effective mechanism for users to monitor the usage of their data in the cloud using a novel approach namely Cloud Information Accountability (CIA) framework.

1.1 Characteristics of cloud computing

Cloud security has summarized five essential characteristics that illustrate the relation to and differences from traditional computing paradigm.

On-demand self-service: A cloud customer may unilaterally obtain computing capabilities, like the usage of various servers and network storage, as on demand, without interacting with the cloud provider.

Broad network access: Services are delivered across the Internet via a standard mechanism that allows customers to access the services through heterogeneous thin or thick client tools (e.g., PCs, mobile phones, and PDAs).

Resource pooling: The cloud provider employs a multitenant model to serve multiple customers by pooling computing resources, which are different physical and virtual resources dynamically assigned or reassigned according to customer demand. Examples of resources include storage, processing, memory, network bandwidth, and virtual machines.

Rapid elasticity: Capabilities may be rapidly and elastically provisioned in order to quickly scale out or rapidly released to quickly scale in. From customer's point of view, the available capabilities should appear to be unlimited and have the ability to be purchased in any quantity at any time.

Measured service: The service purchased by customers can be quantified and measured. For both the provider and customers, resource usage will be monitored, controlled, metered, and reported.

Cloud computing becomes a successful and popular business model due to its charming features. In addition to the benefits at hand, the former features also result in serious cloud-specific security issues. The people whose concern is the cloud security continue to hesitate to transfer their business to cloud. Security issues have been the dominate barrier of the development and widespread use of cloud computing. There are three main challenges for building a secure and trustworthy cloud system:

Outsourcing: Outsourcing brings down both capital expenditure (CapEx) and operational expenditure for cloud customers. However, outsourcing also means that customers physically lose control on their data and tasks. The loss of control problem has become one of the root causes of cloud insecurity. To address outsourcing security issues, first, the cloud provider shall be trustworthy by

providing trust and secure computing and data storage; second, outsourced data and computation shall be verifiable to customers in terms of confidentiality, integrity, and other security services. In addition, outsourcing wills potentially incur privacy. Violations, due to the fact that sensitive classified data is out of the owner's control.

Multi-tenancy: Multi-tenancy means that the cloud platform is shared and utilized by multiple customers. Moreover, in a virtualized environment, data belonging to different customers may be placed on the same physical machine by certain resource allocation policy.

2. OpenNebula Architecture

OpenNebula provides a powerful, scalable and secure multi-tenant cloud platform for fast delivery and elasticity of virtual resources. The storage system allows storing disk images in data stores, which can be then used to define VMs or shared with other users. The images can be OS installations, or data blocks. The Template Repository system allows registering Virtual Machine definitions in the system, to be instantiated later as Virtual Machine instances. Virtual Networking is provided to interconnect Virtual Machines; they can be defined as fixed or ranged network. Once a Template is instantiated to a Virtual Machine, there are a number of operations that can be performed to control their lifecycle, such as migration (live and cold), stop, resume, cancel, etc. These operations are available both from the CLI and the Sunstone GUI.

OpenNebula has been designed to be modular in order to allow its integration with as many different hypervisors and environments as possible. It assumes that the physical infrastructure adopts a classical cluster-like architecture with a front-end, and a set of host nodes where VMs will execute. There is at least one physical network joining all the cluster nodes with the front-end. The front-end executes the main Open Nebula processes while the cluster nodes are hypervisor enabled hosts that provide the resources needed by the VMs. Open Nebula is designed with three layers in mind: Tools, Core and Drivers, as depicted in Fig(1). The Tools layer contains modules providing functionalities for administrators and clients. One component is the Command Line Interface (CLI) that can be used by administrators to manipulate the infrastructure through intuitive commands. The Scheduler module, responsible for VM placement, is implemented in this layer. Other tools can be created using the OpenNebula cloud API which is based on a XML-RPC interface. Similarly to Eucalytpus, OpenNebula works with administrative and client accounts. Administrators access OpenNebula through CLI, while clients launch and manage VMs using web services interfaces. OpenNebula implements an interface compatible with the EC2 Query API from Amazon and another one compatible with the Open Cloud Computing Interface from the Open Grid Forum.

The Core layer consists of components responsible for handling client requests and control resources. The main component of this layer is the Request Manager, which handles client requests through an XML-RPC interface calling internal components according to the invoked method. Hosts and VMs are managed and monitored by the Host Manager and the VM Manager, respectively. The Virtual Network Manager (VN Manager) manages virtual networks by keeping track of IP and MAC addresses and their association with VMs. The SQL Database stores internal data structures. Finally, the third layer is formed by modules called Drivers that supports different underlying platforms. These Drivers run on separated processes that communicate with the Core module through a simple text messaging protocol. There are drivers to deal with file transfers that are implemented by network protocols like NFS and SSH. Also, there are drivers to manage VMs that are dependent on each hypervisor running on the host. Finally, there are drivers to request services from external clouds like Amazon EC2 or Elastic Hosts.

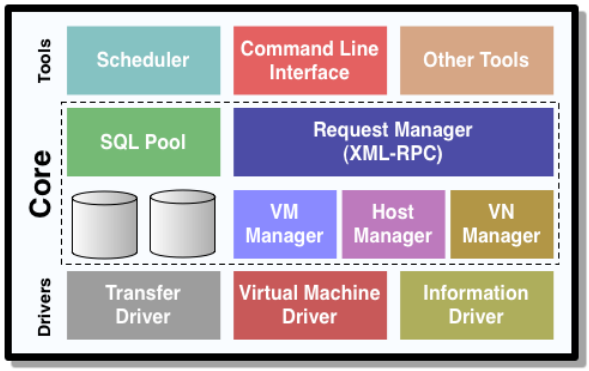

Figure 1: OpenNebula Architecture Diagram

Components of OpenNebula

OpenNebula is the open-source industry standard for data center virtualization, offering the most feature-rich, flexible solution for the comprehensive, complete management of virtualized data centers to enable on-premise IaaS clouds in existing infrastructures. OpenNebula interoperability makes cloud an evolution by leveraging existing IT assets, protecting your investments, and avoiding vendor lock-in. An OpenNebula Private Cloud provides infrastructure users with an elastic platform for fast delivery and scalability of services to meet dynamic demands of service end-users. Services are hosted in VMs, and then submitted, monitored and controlled in the Cloud by using Sunstone or any of the OpenNebula interfaces

Interfaces & APIs: OpenNebula provides many different interfaces that can be used to interact with the functionality offered to manage physical and virtual resources. There are two main ways to manage OpenNebula instances: command line interface and the Sunstone GUI. There are also several cloud interfaces that can be used to create public clouds: OCCI and EC2 Query, and a simple self-service portal for cloud consumers. In addition, OpenNebula features powerful integration APIs to enable easy development of new components (new virtualization drivers for hypervisor support, new information probes, etc). Users and Groups: OpenNebula supports user accounts and groups, as well as various authentications and authorized. This feature can be used to create isolated compartments within the same cloud, implementing multi-tenancy. Moreover, a powerful Access Control List

mechanism is in place to allow different role management, allowing a fine grain permission granting.

Hosts: Various hypervisors are supported in the virtualization manager, with the ability to control the lifecycle of Virtual Machines, as well as monitor them. This monitoring also applies to the physical hosts. The main hypervisors are supported, Xen, KVM, and VMware.

Networking: An easily adaptable and customizable network subsystem is present in OpenNebula in order to better integrate with the specific network requirements of existing datacenters. Support for VLANs and Open vSwitch are also featured.

Storage: OpenNebula is flexible enough to support as many different image storage configurations as possible. The support for multiple data stores in the Storage subsystem provides extreme flexibility in planning the storage backend and important performance benefits. The main storage configurations are supported, file system datastore, to store disk images in a file form and with image transferring using ssh or shared file systems (NFS, GlusterFS, Lustre), iSCSI/LVM to store disk images in a block device form, and VMware data store specialized for the VMware hypervisor that handle the vmdk format.

Clusters: Clusters are pools of hosts that share datastores and virtual networks. Clusters are used for load balancing, high availability, and high performance computing.

OpenNebula provides a powerful, scalable and secure multi-tenant cloud platform for fast delivery and elasticity of virtual resources. The Storage system allows storing disk images in datastores that can be then used to define VMs or shared with other users. The images can be OS1 installations, or data blocks. The Template Repository system allows registering Virtual Machine definitions in the system, to be instantiated later as Virtual Machine instances. Virtual Networking is provided to interconnect Virtual Machines, they can be defined as fixed or ranged networks .Once a Template is instantiated to a Virtual Machine, there are a number of operations that can be performed to control their lifecycle, such as migration (live and cold), stop, resume, cancel, etc. These operations are available both from the CLI and the Sunstone GUI.

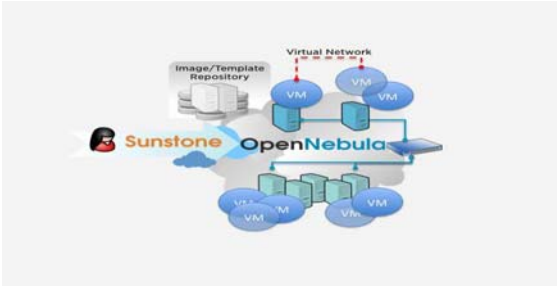

Figure 2: OpenNebula Interface

2.2 Networking

Open Nebula manages IP and MAC addresses of VMs and the virtual networks between them. There are two types of virtual networks: the Fixed network (Public) that uses fixed set of IP and associated MAC addresses and the Ranged network (Red LAN) defined over a range of network addresses. VMs must pertain to one Red LAN and can, optionally, pertain to the fixed network.

2.3 VM Placement

As aforementioned, VM placement [1] decisions are made by the Scheduler. The scheduler accesses all requests received by Open Nebula and, based on them, it keeps track of allocations and sends appropriate deployment or enforcement commands to the Core. The default Open Nebula scheduler provides a scheduling policy that places VMs according to a ranking algorithm, which relies on performance data from both the running VMs and physical resources. The basic operation of this scheduling algorithm is showed at Fig(3) The algorithm needs two inputs sent in the VM request: VM Requirements and VM Rank. The former consists of simple rules indicating the minimum resource requirements (as CPU or Memory), and the later consists of a policy for resource allocation. Thus, when a VM request arrives, the Scheduler, based on the VM requirements, filters those hosts that do not meet the requirements. After that, the remaining hosts are sorted using the Rank policy.

Algorithm 1: Scheduling algorithm.

1 **Inputs:** requirements, rank, hostsList;

2 **Outputs:** selectedHost;

3 **Begin**

4 **for each** host **in** hostsList **do**

5 **if** (host meets requirements) **then**

6 candidates.new(host);

7 **end**

8 **end**

9 sorted = sortByRank(candidates, rank);

10 selectedHost = sorted(1);

11 **end**

Figure 3:.Default scheduling Algorithm

3. Cloud Information Accountability

In this section, we present an overview of the Cloud Information Accountability framework and discuss how the CIA [2] framework meets the design requirements met. The Cloud Information Accountability framework proposed in this work conducts automated logging and distributed auditing of relevant access performed by any entity, carried out at any point of time at any cloud service provider. It has two major components: logger and log harmonizer.

Major Components

There are two major components of the CIA, the first being the logger, and the second being the log harmonizer. The logger is the component which is strongly coupled with the user's data, so that it is downloaded when the data are

accessed, and is copied whenever the data are copied. It handles a particular instance or copy of the user's data and is responsible for logging access to that instance or copy. The log harmonizer forms the central component which allows the user access to the log files. The logger is strongly coupled with user's data (either single or multiple data items). Its main tasks include automatically logging access to data items that it contains, encrypting the log record using the public key of the content owner, and periodically sending them to the log harmonizer. It may also be configured to ensure that access and usage control policies associated with the data are honored. For example, a data owner can specify that user X is only allowed to view but not to modify the data. The logger will control the data access even after it is downloaded by user X.

The logger requires only minimal support from the server (e.g., a valid Java virtual machine installed) in order to be deployed. The tight coupling between data and logger, results in a highly distributed logging system, therefore meeting our first design requirement. Furthermore, since the logger does not need to be installed on any system or require any special support from the server, it is not very intrusive in its actions, thus satisfying our fifth requirement. Finally, the logger is also responsible for generating the error correction information for each log record and sends the same to the log harmonizer. The error correction information combined with the encryption and authentication mechanism provides a robust and reliable recovery mechanism, therefore meeting the third requirement.

The log harmonizer is responsible for auditing. Being the trusted component, the log harmonizer generates the master key. It holds on to the decryption key for the IBE key pair, as it is responsible for decrypting the logs. Alternatively, the decryption can be carried out on the client end if the path between the log harmonizer and the client is not trusted. In this case, the harmonizer sends the key to the client in a secure key exchange. It supports two auditing strategies: push and pull. Under the push strategy, the log file is pushed back to the data owner periodically in an automated fashion. The pull mode is an on-demand approach, whereby the log file is obtained by the data owner as often as requested. These two modes allow us to satisfy the aforementioned fourth design requirement. In case there exist multiple loggers for the same set of data items, the log harmonizer will merge log records from them before sending back to the data owner. The log harmonizer is also responsible for handling log file corruption. In addition, the log harmonizer can itself carry out logging in addition to auditing. Separating the logging and auditing functions improves the performance. The logger and the log harmonizer are both implemented as lightweight and portable JAR files. The JAR file implementation provides automatic logging functions, which meets the second design requirement.

3.1 Data Flow

The overall CIA framework, combining data, users, logger and harmonizer is sketched in Fig.(4). At the beginning, each user creates a pair of public and private keys based on Identity-Based Encryption [3]. The IBE scheme is a Weil-pairing-based IBE scheme, which protects us against one of the most prevalent attacks to our architecture. Using the generated key, the user will create a logger component which is a JAR file, to store its data items.

The JAR file includes a set of simple access control rules specifying whether and how the cloud servers and possibly other data stakeholders (users, companies) are authorized to access the content itself. Then, he sends the JAR file to the cloud service provider that he subscribes to. Once the authentication succeeds, the service provider (or the user) will be allowed to access the data enclosed in the JAR.

Depending on the configuration settings defined at the time of creation, the JAR will provide usage control associated with logging, or will provide only logging functionality. As for the logging, each time there is an access to the data; the JAR will automatically generate a log record, encrypt it using the public key distributed by the data owner, and store it along with the data. The encryption of the log file prevents unauthorized changes to the file by attackers. The data owner could opt to reuse the same key pair for all JARs or create different key pairs for separate JARs. In addition, some error correction information will be sent to the log harmonizer to handle possible log file corruption.

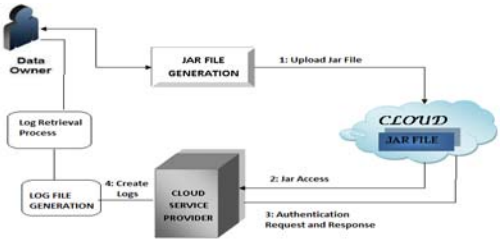

Figure 4: System Architecture

To ensure trustworthiness of the logs, each record is signed by the entity accessing the content. Further, individual records are hashed together to create a chain structure, able to quickly detect possible errors or missing records. The encrypted log files can later be decrypted and their integrity verified. They can be accessed by the data owner or other authorized stakeholders at any time for auditing purposes with the aid of the log harmonizer. Our proposed framework prevents various attacks such as detecting illegal copies of users' data. Note that our work is different from traditional logging methods which use encryption to protect log files. With only encryption, their logging mechanisms are neither automatic nor distributed. They require the data to stay within the boundaries of the centralized system for the logging to be possible, which is however not suitable in the cloud.

Each log harmonizer is in charge of copies of logger components containing the same set of data items. The harmonizer is implemented as a JAR file. It does not contain the user's data items being audited, but consists of class files for both a server and a client processes to allow it to communicate with its logger components. The harmonizer stores error correction information sent from its logger components, as well as the user's IBE decryption key, to decrypt the log records and handle any duplicate records. Duplicate records result from copies of the user's

data JARs. Since user's data are strongly coupled with the logger component in a data JAR file, the logger will be copied together with the user's data. Consequently, the new copy of the logger contains the old log records with respect to the usage of data in the original data JAR file. Such old log records are redundant and irrelevant to the new copy of the data. To present the data owner an integrated view, the harmonizer will merge log records from all copies of the data JARs by eliminating redundancy. The logger components always ping the harmonizer before they grant any access right. If the harmonizer is not reachable, the logger components will deny all access. In this way, the harmonizer helps prevent attacks which attempt to keep the data JARs offline for unnoticed usage. If the attacker took the data JAR offline after the harmonizer was pinged, the harmonizer still has the error correction information about this access and will quickly notice the missing record.

3.2 Technique Used For Auditing Mechanism

To allow users to be timely and accurately informed about their data usage, our distributed logging mechanism is complemented by an innovative auditing mechanism. We support two complementary auditing modes: push mode and pull mode

Push mode. In this mode, the logs are periodically pushed to the data owner (or auditor) by the harmonizer. The push action will be triggered by either type of the following two events: one is that the time elapses for a certain period according to the temporal timer inserted as part of the JAR file; the other is that the JAR file exceeds the size stipulated by the content owner at the time of creation.

After the logs are sent to the data owner, the log files will be dumped, so as to free the space for future access logs. Along with the log files, the error correcting information for those logs is also dumped. This push mode is the basic mode which can be adopted by both the PureLog and the AccessLog, regardless of whether there is a request from the data owner for the log files. This mode serves two essential functions in the logging architecture: first it ensures that the size of the log files does not explode and secondly it enables timely detection and correction of any loss or damage to the log files. Concerning the latter function, we notice that the auditor, upon receiving the log file, will verify its cryptographic guarantees, by checking the records' integrity and authenticity. By construction of the records, the auditor will be able to quickly detect forgery of entries, using the checksum added to each and every record.

Pull mode. This mode allows auditors to retrieve the logs anytime when they want to check the recent access to their own data. The pull message consists simply of an FTP pull command, which can be issues from the command line. For naive users, a wizard comprising a batch file can be easily built. The request will be sent to the harmonizer, and the user will be informed of the data's locations and obtain an integrated copy of the authentic and sealed log file.

Pushing or pulling strategies have interesting tradeoffs. The pushing strategy is beneficial when there are a large number of accesses to the data within a short period of time. In this case, if the data are not pushed out frequently enough, the log file may become very large, which may increase cost of operations like copying data. The pushing mode may be preferred by data owners who are organizations and need to keep track of the data usage consistently over time. For such data owners, receiving the logs automatically can lighten the load of the data analyzers. The maximum size at which logs are pushed out is a parameter which can be easily configured while creating the logger component. The pull strategy is most needed when the data owner suspects some misuse of his data; the pull mode allows him to monitor the usage of his content immediately. A hybrid strategy can actually be implemented to benefit of the consistent information offered by pushing mode and the convenience of the pull mode.

The logging and synchronization steps with the harmonizer in case of PureLog. First,checks whether the size of the JAR has exceeded a stipulated size or the normal time between two consecutive dumps has elapsed. The size and time threshold for a dump are specified by the data owner at the time of creation of the JAR. The algorithm also checks whether the data owner has requested a dump of the log files. If none of these events has occurred, it proceeds to encrypt the record and write the error-correction information to the harmonizer.

The communication with the harmonizer begins with a simple handshake. If no response is received, the log file records an error. The data owner is then alerted through e-mails, if the JAR is configured to send error notifications. Once the handshake is completed, the communication with the harmonizer proceeds, using a TCP/IP protocol. If any of the aforementioned events (i.e., there is request of the log file, or the size or time exceeds the threshold) has occurred, the JAR simply dumps the log files and resets all the variables, to make space for new records. In case of AccessLog, the algorithm is modified by adding an additional check is needed. Precisely, the AccessLog checks whether the CSP accessing the log satisfies all the conditions specified in the policies pertaining to it. If the conditions are satisfied, access is granted; otherwise, access is denied. Irrespective of the access control outcome, the attempted access to the data in the JAR file will be logged .Our auditing mechanism has two main advantages. First, it guarantees a high level of availability of the logs. Second, the use of the harmonizer minimizes the amount of workload for human users in going through long log files sent by different copies of JAR files.

4. Conclusion

A noble work of implementing openNebula based cloud computing infrastructure has been presented here. OpenNebula seems to be more suited for research and experimental studies. OpenNebula provides a modular architecture intended to be more flexible. An interesting algorithm that places VMs depending on their requirements. The client communication is also managed by modules that offer interfaces based on web services. It is perhaps the only open management platform that has invested into a tailor able VM placement algorithm. As such it may provide a nice environment for those

researchers seeking to compare and develop different resource allocation strategies. A limitation found with the OpenNebula is that, like XCP, their infrastructure assumes a classical cluster-like architecture with a front-end and without any redundant services. We also proposed a solution to provide an effective mechanism for users to monitor the usage of their data in the cloud using a novel approach namely Cloud Information Accountability (CIA) framework and provide innovative approaches for automatically logging any access to the data in the cloud together with an auditing mechanism. Our approach allows the data owner to not only audit his content but also enforce strong back-end protection if needed. Moreover, one of the main features of our work is that it enables the data owner to audit even those copies of its data that were made without his knowledge.

References

[1] Thiago Cordeiro, Douglas Damalio, Nadilma Pereira,Patricia Endo, André Palhares, Glauco Gonçalves, Djamel Sadok, Judith Kelner Open Source Cloud Computing Platforms, 2010 Ninth International Conference on Grid and Cloud Computing.

[2] Smitha Sundareswaran, Anna C. Squicciarini, Member, IEEE, and Dan Lin,"Distributed Accountability for Data Sharing in the cloud 2012 IEEE Transactions".

[3] D. Boneh and M.K. Franklin" Identity-Based Encryption from the Weil Pairing,2001".

[4] OpenNebulaProjectOpenNebula1.4Documentation. Available:http://www.opennebula.org/documentation: documentation. Visited on May, 2010.

[5] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L.Youseff, and D. Zagorodnov. Eucalyptus: A Technical Report on an"Elastic Utility Computing Archietcture Linking Your Programs to Useful Systems. University of California, Santa Barbara, October 2008".

[6] G.Ateniese, R.Burns,R.Curtmola, J. Herring, L. Kissner, Z.Peterson, and D. Song, "Provable Data Possession at Untrusted Stores," Proc. ACM Conf. Computer and Comm. Security, pp. 598-609, 2007.

[7] R. Corin, S. Etalle, J.I. den Hartog, G. Lenzini, and I. Staicu,"A Logic for Auditing Accountability in Decentralized Systems", Proc. IFIP TC1 WG1.7 Workshop Formal Aspects in Security and Trust, pp. 187-201, 2005."

Author Profile

Midhu Babu was born in 1989. She received her BTech degree in computer Science from Sree Narayana Gurukulam College Of Engineering, Mahatma Gandhi University, Kerala, India in 2011.Now pursuing M Tech degree in Computer Science and Engineering from SRM University, Chennai, India. She is currently doing training on cloud computing from JPA Solutions, Chennai. Her research interests are cloud computing, web services and networking.

A. M. J Muthu Kumaran received M Tech degree in Computer Science & Engineering from Dr. MGR University in 2007 and he did BE in Computer Science from Banglore University in 1996.Currently working as Assistant Professor in SRM University. He had a good experience in teaching.His research area is in image processing.