SECURED DATA DELETION IN CLOUD BASED MULTI-TENANT DATABASE ARCHITECTURE

Vanitha Muthusamy[1], Kavitha C.[2]

[1]Research Scholar, Anna University of Technology, Coimbatore, India
[2]Proffessor, Department of Computer Science, K.S.R College of Technology, Thiruchengode, India
Email:[1]vanitha_amara@yahoo.co.in

Abstract

Cloud Computing is evolving from a mere "storage" technology to a new vehicle for Business Information Systems (BIS) to manage, organize and provide added-value strategies to current business models. Cloud platform server cluster is running in the network environment and it may contain multiple users' / tenants data and the data may be scattered in different virtual data centers. In a multi-user shared cloud computing platform users are only logically isolated, but data of different users may be stored in same physical equipment. These equipments can be rapidly provisioned, implemented, scaled up or down and decommissioned. Current cloud providers do not provide the control or at least the knowledge over the provided resources to their customers. When the SLA between the customer and the cloud provider ends, today in no way it is assured that the particular customers' data is completely destroyed or destructed from the cloud provider's storage. In this paper we explore the key implementation patterns of data storage and methods to identify individual customer data and securely delete / destruct it.

Key words: Multi-tenant Database, Cloud Computing, Data Lifecycle, Encryption, Cryptography, Meta-data

I. INTRODUCTON

The data in cloud is encrypted during rest, transit and back-up in multi tenant storage. The encryption keys are managed per customer. There are different stages of data life cycle Create, Store, Use, Share, Archive and Destruct. In multi tenant shared database architecture, the final stage is overlooked most of the times, but which is the complex stage of data in cloud. Data retention assurance may be easier for the cloud provider to demonstrate while the data destruction is extremely difficult. If the data is not properly deleted, because of the cloud characteristics of pooling and elasticity the resources allocated to one user will be reallocated to a different user at a later time. For memory or storage resources, it might therefore be possible to recover data written by a previous user. The physical data destruction at the end of SLA is also not possible since the disk / data storage device is possibly used by other users. The proposed method identifies way to track individual customers' data and their encryption keys and provides solution to completely delete the data from the cloud provider's multi-tenant shared storage architecture. This method will also help customer to effectively locate and isolate a particular customer's data through proper identity management (as there is a possibility of data commingling) and completely remove or render it. It also ensures deletion of data copies as there are always possibilities of more than one copy of data being maintained for back-up purposes. The data destruction proof shall also be provided to customer making sure that the owner's data is completely removed.

II. SERVICE ORIENTED CLOUD COMPUTING DATABASE ARCHITECTURE

In a multi-user shared cloud computing platform users are only logically isolated, but data of different users may be stored in same physical equipment. These equipments can be rapidly provisioned, implemented, scaled up or down and decommissioned. There are varying degrees of data isolation for a multi-tenant application database architecture that ranges from an isolated environment to a totally shared environment. Implementation patterns along this spectrum include three models

A. Dedicated database

Each tenant owns a separate database

B. Dedicated table/schema

Multiple tenants shares the same database, but each tenant owns their separate tables/schema

C. *Shared table/schema*

Multiple tenants shares same database, tables and schema. In this pattern, records of all tenants are stored in a single shared table sets mixed in any order in which a tenant ID column is inserted in each table to associate the data records with the corresponding tenants.

III. DATA STORAGE PATTERN IN SHARED TABLE /SCHEMA STRUCTURE

In case of dedicated database or table/schema the tenants have their separated schemas. The changes of the data model of one tenant can be made directly to its specific database/tables without impact to other tenants. But in case of share table/schema structure because of the sharing of schema, it can only support data field extension in which flexibility degree is usually measured by the maximal number of extension fields. In this kind of storage pattern because of the sharing of schema, it can only support data field extension. The flexibility degree is usually measured by the maximum number of extension fields.

We are going to discuss about different data storage pattern in a multi-user shared database environment for a pre-sales application

Reserved / Fixed Columns pattern: The following figure shows a single table that supports multiple tenants. Here the number of columns is fixed. Hence the flexibility is the very less. Most cloud providers don't use this kind of storage. This kind of storage can fit for simple sales applications that run with in the same country where the customer as well as the service provider resides. Table 1. Shows example of this pattern

Extension / Sub-Table pattern: But when the complexity grows and when there is a need for supporting multiple countries, tenants may need to support customers from different countries. In order to support additional tables / columns and also increased performance, data can be stored across different sub-tables. The main table and sub tables can be related with key columns. Table 2 and Table 3 shows one such sample implementation.

Table 1. QUOTE_WITH_DETAILS

ROW_ID	TENANT_ ID	CUSTOMER_ ID	QUOTE ID	QUOTE_NAME	QUOTE_AMT	QUOTE_START _DATE	QUOTE_COUNTRY
1ASFWF	Tenant_1	Customer_1	1-AB167DG	Quote	1234.00	12-10-2011	INDIA
1ASFAF	Tenant_1	Customer_2	1-AB167DL	Quote_2	null		AUSTRALIA
1AVWF	Tenant_1	Customer_3	1-AB167CV	null	5674	null	ARGENTINA
1ASFLF	Tenant_2	……...	……...	……...	……...	……...	SINGAPORE
1ASFQF	Tenant_2	……...	…….....	……...	……...	……...	MALAYSIA
1ASLWF	Tenant_2	Customer_n	1-AB167XX	String1	1111.23	12-11-2011	CANADA

Table 2. QUOTE_MAIN

ROW_ID	TENANT_ ID	CUSTOMER_ ID	QUOTE_ ID	QUOTE_DETAILS_ROW_ID
1ASFWF	Tenant_1	Customer_1	1-AB167DG	1LPOMI
1ASFAF	Tenant_1	Customer_2	1-AB167DL	1HPOMI
1AVWF	Tenant_1	Customer_3	1-AB167CV	1KPOMI
1ASFLF	Tenant_2	……...	……...	1LPNMI
1ASFQF	Tenant_2	……...	……...	1RPOMI
1ASLWF	Tenant_2	Customer_n	1-AB167XX	1LPNMI

Table 3. QUOTE_DETAILS

ROW _ID	TENANT_ ID	QUOTE_ID	QUOTE_NAME	QUOTE_TOTAL_AMT	QUOTE_START _DATE	QUOTE_DISCOUNT
1LPOMI	Tenant_1	1-AB167DG			12-10-2011	
1HPOMI	Tenant_1	1-AB167DL				
1KPOMI	Tenant_1	1-AB167CV			null	
1LPNMI	Tenant_2	……...			……...	
1RPOMI	Tenant_2	……...			……...	
1LPNMI	Tenant_2	1-AB167XX			12-11-2011	

IV. UNIQUE CHANLLENGES OF MULTITENANT DATABASE

Possibilities of one tenant accessing other tenants' data through a malicious request One tenant customizing various schema objects in real time can affect the functionality or availability of the system for all other tenants Possibilities of data updating (insert, modify or delete) of one tenant affecting the data of other tenant. Scaling up of systems response time when more tenants start to use the database in a multi-tenant environment.

Identifying and deleting a single tenant data when closing the account (data, meta-data and crypt keys).

V. MULTI-TENANT DATABASE SETUP

It's difficult to create a statically compiled database engine executable that can meet the unique challenges of multitenancy. Instead, a multitenant cloud-oriented database system must be dynamic in nature, or polymorphic, to fulfill the individual expectations of various tenants and their users. In order to accomplish this, the multi-tenant storage model manages virtual database structures using a set of metadata, data, data encryption and pivot tables

A. MULTI-TENANT META-DATA

Any cloud database model will have two core internal tables that it uses to manage metadata that corresponds to a tenant's schema objects one for storing Multi-tenant objects and other for storing Multi-tenant fields.

The MultiTenant_Objects system table stores metadata about the tables that an tenant defines for an application, including a unique identifier for a object (ObjID), the tenant (TenantID) that owns the object, and the name given to the object (ObjName).

The MultiTenant_Fields system table stores metadata about the fields (columns) that an tenant defines for each object, including a unique identifier for a field (FieldID), the tenant (TenantID) that owns the encompassing object, the object that contains the field (ObjID), the name of the field (FieldName), the field's datatype, a Boolean value to indicate if the field requires indexing (IsIndexed), and the position of the field in the object relative to other fields (FieldNum).

B. MULTI-TENANT DATA

The multi tenant system table stores the application-accessible data that maps to all tenant specific tables and their fields, as defined by metadata in MultiTenant Objects and MultiTenant Field tables. Each row includes the Tenant that owns the row (Tenant ID), and the encompassing object identifier (ObjID). Each row in the MultiTenant_Data table also has a Name field that stores a "natural name" for corresponding records; for example, an Customer record might use "Customer Name," a Case record might use "Case Number," and so on.

MultiTenant_Fields can use any one of a number of standard structured datatypes such as text, number, date, and date/time as well as special-use, rich structured datatypes such as pick list (enumerated field), auto-number (auto-incremented, system-generated sequence number), formula (read-only derived value), master-detail relationship (foreign key), checkbox (Boolean), email, URL, and others. MultiTenant_Fields can also be required (not null) and have custom validation rules (for example, one field must be greater than another field), both of which Database.com enforces.

C. MULTI-TENANT DATA ENCRYPTION

A way to further protect tenant data is by encrypting it within the database, so that data will remain secure even if it falls into the wrong hands.

Cryptographic methods are categorized as either symmetric or asymmetric. In symmetric cryptography, a key is generated that is used to encrypt and decrypt data. Data encrypted with a symmetric key can be decrypted with the same key. In asymmetric cryptography (also called public-key cryptography), two keys are used, designated the public key and the private key. Data that is encrypted with a given public key can only be decrypted with the corresponding private key, and vice versa. Generally, public keys are distributed to any and all parties interested in communicating with the key holder, while private keys are held secure. For example, if Alice wishes to send an encrypted message to Bob, she obtains Bob's public key through some agreed-upon means, and uses it to encrypt the message. The resulting encrypted message, or cyphertext, can only be decrypted by someone in possession of Bob's private key (in practice, this should only be Bob). This way, Bob never

has to share his private key with Alice. To send a message to Bob using symmetric encryption, Alice would have to send the symmetric key separately—which runs the risk that the key might be intercepted by a third party during transmission.

Public-key cryptography requires significantly more computing power than symmetric cryptography; a strong key pair can take hundreds or even thousands of times as long to encrypt and decrypt data as a symmetric key of similar quality. For SaaS applications in which every piece of stored data is encrypted, the resulting processing overhead can render public-key cryptography infeasible as an overall solution. A better approach is to use a key wrapping system that combines the advantages of both systems.

With this approach, three keys are created for each tenant as part of the provisioning process: a symmetric key and an asymmetric key pair consisting of a public key and a private key. The more-efficient symmetric key is used to encrypt the tenant's critical data for storage. To add another layer of security, a public/private key pair is used to encrypt and decrypt the symmetric key, to keep it secure from any potential interlopers.

When an end user logs on, the application uses impersonation to access the database using the tenant's security context, which grants the application process access to the tenant's private key. The application (still impersonating the tenant, of course) can then use the tenant's private key to decrypt the tenant's symmetric key and use it to read and write data

VI. SECURE DELETION OF TENANT DATA

When a contract between the tenant and the provider ends, the provider must ensure that he deletes all tenant related data such as Meta-data, Tenant data and Encryption keys specific to tenant. Since all the table has reference to tenant with Tenant_ID, deletion should be carried over in all the affected table based on the Tenant_ID.

ACKNOWLEDGEMENT

This work is supported by Center for Research, Anna University of Coimbatore

REFERENCES

[1] Cloud_Computing_Use_Cases_Whitepaper-4_0, pp

[2] Seny Kamara, Microsoft Research, Cryptographic Cloud Storage

[3] J. Brodkin. (2008, Jun.). "Gartner: Seven cloud-computing security risks." Infoworld, Available: http://www.infoworld.com/d/security-central/gartner-seven-cloudcomputingsecurity-risks-853?page = 0, 1 > [Mar. 13, 2009].

[4] R. K. Balachandra, P. V. Ramakrishna and A. Rakshit. "Cloud Security Issues." In PROC '09 IEEE International Conference on Services Computing, 2009, pp 517-520.