UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

# Exploring Reference Architectures for Software as a Service (SaaS)

CS5722 Term Paper

Date: 13-05-20

Group Members

Assad Nawaz - 19085842

Reezvee Sikder - 15140997

Shravan Chandrashekharaiah - 18043038

# Table of Contents

## Abstract

This paper researches and describes three components and their reference architectures in relation to software as a service (SaaS). Research in logging & monitoring, caching and multi-tenancy has been carried out to describe the various different reference architectures in these areas. The paper aims to give an overview of requirements, methods and descriptions of these reference architectures. This paper aims to give an in-depth analysis of various database architecture to support multi-tenancy. It also aims at discussing the cost that incurs in each architecture. Furthermore, it explains in depth about the design pattern on the security concerns that a potential client may have on using a shared database. The topic of caching is also covered in relation to SaaS applications and how they make the user experience great by reducing delay in time latency and better search results. In this paper different architectures and techniques of caches are explained in depth.

## I. Introduction

Software as a service is a common software distribution model where a provider hosts an application on cloud infrastructure and makes it available to customers around the globe over the internet. SaaS is one of the three categories of cloud computing, the other two being infrastructure as a service (IaaS) and platform as a service (PaaS). SaaS has become increasingly popular over the last years, with its rise gives the need to carefully design the various components in SaaS such that they are architecturally sound and scalable. The reference architectural patterns of these components will be researched and discussed in this paper. Three of the most important components in SaaS have been chosen to be discussed in this paper, these include logging and monitoring, caching architectures and multi-tenant architectures. This paper discusses the fundamental technology(multi-tenancy) that cloud uses to share IT resources and securely and the isolated and shared approach and its pros and cons of each approach. The use of design patterns to enhance security is also described. Caching plays an important role in SAAS application as it helps in providing a better experience to the user by improving time latency. For performance improvement in small scale platforms, co-operation between caches can be used, but for large scale platforms, these divided caches are combined together along with some principles and properties to form architecture. This paper explains the two architectures available for caching i.e. Hierarchical and Distributed. Caching has a variety of techniques available to its disposal to achieve its target. In this paper, we are focusing on SQUID, which is used for web proxy caching to improve the delay in time latency and Memcached caching, which is used for reducing database load as they are used widely and prominently in SAAS applications.

## II. Logging and Monitoring Architectures in SaaS

### *A. Logging & Monitoring*

Logging and monitoring are key components in typical SaaS applications. When building these components, it is vital that they are maintainable and scalable in nature. Logging is the process of tracking and storing errors that occur in software or related data. Logging usually leads to log files being created. These files contain recording of events that occurred in an application or system. These events include failures, errors or changes in state in software systems. Logging is one of the many tools that system administrators use to view system status and operational status on a high-level.

Monitoring is where data collected from logging or system statues is aggregated and then analysed. This data is then plotted in some way to give an overview of system status. Monitoring is used for alerting developers or admins for when there are changes in system behaviours or core parts of systems have malfunctioned. Monitoring relies on incoming data from the system to be aggregated and visualised [1].

### *B. Logging Architecture & Requirements*

As modern cloud-based architectures move towards a more distributed architectures, managing of these systems also increases in further complexity. As server clusters are spun up/down to handle change in demand, it is important to handle the logging information of these servers. Centralised logging strategies are proposed. This method ensures that once cloud resources are put to sleep due to change changed demand, the logging information will still be available. Two major requirements are outlined in [2] to architect a centralised logging strategy, store all logs to a redundant, isolated storage area and the standardisation of log formats.

The first requirement outlines that all logs from the relevant servers are sent to a syslog before being written directly to disk. The syslog acts as a controller who directs the logs to a logging server. These logging servers are also required to be distributed in nature so that they are redundant. The log data can then be manipulated into a form so that it can be used by various actors such as system administrators and developers. To ensure that logs provide good value to the engineers and administrators, it is critical for them to be standardised such that there is a common API to interact with these logs, this includes having various log formats, log error codes and severity codes. Error descriptions should contain common attributes such as date, time, server timestamp, software module codes etc. so that developers can quickly search for these logs.

While logging and monitoring can be used to assess a systems live performance, it can also be used as a way to evaluate the performance of a cloud SaaS. PRESENCE is a framework developed to monitor, model and evaluate performance of SaaS web services. The framework contains five components [3]. The monitoring and modelling module is used to collect data from the system, a set of agents which are responsible for the specific scalability and performance metrics, a stealth module which ensures load balancing of the workload, the virtual QoS aggregator and the PRESENCE client. PRESENCE uses a set of common performance metrics to measure the behaviour and performance of the cloud application. An example of a PRESENCE agent is the Yahoo Cloud Serving Benchmark (YCSB). This agent measures the throughput (operations/sec) of the different noSQL databases (Redis, MongoDB, Memcached) in a cloud system. Metrics that were measured for this agent are throughput, latency read and latency update. This experimental works objective was to understand system performance behaviour under various different types of load (ops/second).

## *C. Monitoring Architectural Framework*

Frameworks for monitoring and logging can provide way to ensure performance, security and robustness. Kun Ma et al [4] describes a lightweight monitoring framework which can be used to monitor cloud applications such SaaS. The framework consists of four components which are deployed on two layers, the SaaS layer and the IaaS layer. The four components include software monitoring, service monitoring (SaaS), hypervisor monitoring and resource monitoring (IaaS). Figure x describes the manager-agent architecture below. Agents (guest, hypervisor etc.) are responsible for gathering data from the various services in the cloud. These agents then send data via UDP packets to the manager who is responsible for storing and monitoring of this data.

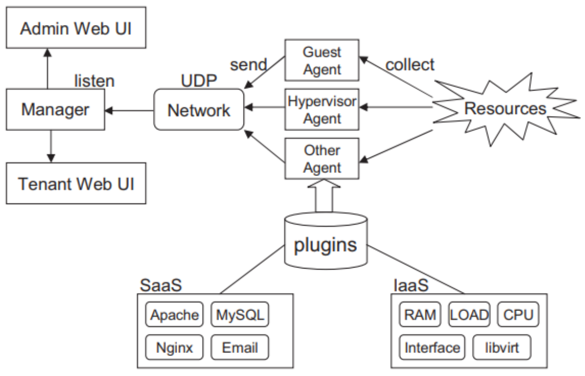


*Figure 1. Manager-agent architecture [4]*

The monitoring agent is a daemon that periodically collects system information. This information is collected by agents which are then sent to the manager. The framework provides default basic monitoring metrics but plugins can be added to monitor additional information. The framework also provides a module centric architecture. These modules/plugins carry out individual monitoring. This allows for reusability of modules in different monitoring applications. Modules include, virtualisation monitoring module, SNMP performance monitoring module, application monitoring module and a few other modules. The virtualisation monitoring module gathers statistics on virtualised users on the cloud system using a libvirt plugin. Libvirt supports various hypervisor drivers for monitoring these drivers include Hyper-V, KVM, VMware and VirtualBox. The libvirt plugin is essential as it can directly retrieve driver stats from the hypervisor. Metrics such as CPU usage, memory and networking can be monitored using libvirt.

The application monitoring module is used to monitor the software layer in the cloud. It monitors server throughput of various different servers such as the Apache web server and the Nginx web server. This modular approach of having various different monitoring modules allows web administrators to be able to seamlessly add/remove monitoring modules to a cloud application.

## *D. An Industry Example of a Logging Stack*

It is very important that software companies implement logging and monitoring correctly. One of the reasons being that logging and monitoring can help to find issues or bugs before the customer finds them. This was one of the main problems at Grab, to find out when a service stops before the customer. Using external a cloud-based storage service was not sufficient due to external issues such as time taken to retrieve logs or the system was not scalable. Requirements for this logging stack were formed by answering questions such as *how much data was produced each day, what was the responsible response time to wait for?* Offering visualisation tools which showed errors and alerts in real time was requested the most by Grab customers. Elasticsearch, a service which allows real time analysation of data was used as it supported horizontal scaling. Initial designs included a 5 node closer but this was changed so that Elasticnodes would be increased as Grab enrolled more customers. In addition to Elasticsearch, a tool called Datadog was used as a monitoring tool. This tool allowed for visualisation of heap utilisation which was essential for Grab. Knowing a nodes JVM usage was key to their logging stack, as this was one of the key monitoring data that helped to predict system crashes/ failures. Elasticsearch's ELK (Elasticsearch, Logstash & Kibana) stack was an ideal solution for Grab. Logstash which is a server-side data processing pipeline along with Kibana which allows for easy visualisation of Elasticsearch data proved to be a power monitoring solution with lots of extensibility and scalability. [5]

## III. Caching Architectures & Techniques in SaaS

In order to be scalable, configurable, and multi-tenant efficient, SAAS infrastructure needs a scale-out infrastructure without any data contamination. One of the factors which contribute to this is a shorter latency of time. As the SAAS is hosted on the web it follows web caching protocols to provide a better result and improved time latency [6].

Web caching is similar to system caching, it anticipates future requests which may come to it and stores the web resources [7]. It differs from the system caching in terms of object size and retrieval costs. For object size, web cache operators track the percentage of requests that it serves, generally called as Object Hit Rate and the percentage of bytes that it serves, called as the Byte Hit Rate. Traditional replacement algorithms used in the system caches often assume a fixed object size and retrieval cost, thus the variable object size of web caches affects the performance sometimes [7].

The World Wide Web provides a lot of information and various services to the world, but these services are not backed up by proper networks, thus causing time latency while accessing these resources. To keep these latencies in the limit, multiple copies of these resources are stored in dispersed web caches [6, 7].

The percentage of requests that can be fetched from previously cached documents is known as Hit Rate [7]. This Hit Rate can be increased drastically by ensuring that a large number of people are accessing the same cache. This will increase the chances of finding the document in that cache [7]. Several caches can cooperate with each other to increase the number of people using that cache. Two common approaches to implement this cooperation between the caches are Hierarchical and Distributed caching [7].

*A. Caching Architecture: Hierarchical Caching*

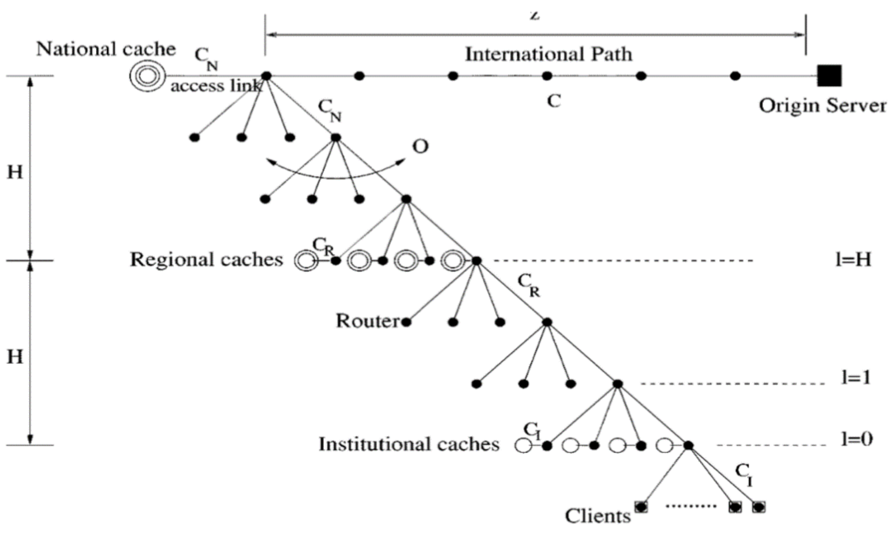


*Figure 2: Tree Node, showing cache placement [8]*

In Hierarchical caching, caches are positioned at levels shown in Fig 2. Client caches are present at the lowest level of the hierarchy. Whenever a request is not satisfied by the client's cache, it gets passed onto the institutional caches. If the required document is not present at this level, that request then gets passed on to the regional caches, which in return pass on the unsatisfied request to the cache above it i.e. national cache. If the requested document is not available in any of the caches as described in Fig 2 then the national cache contacts the origin server directly. Ultimately if the document is found at the origin server or in any cache level as shown in Fig 2, the document gets passed down the hierarchy and leaves a version of itself at each of the cache levels. Any more requests for the same document which is not present at any of the level, travels up in level until the document is found and saved [8].

From fig 2, 'O' is the nodal outdegree of the tree [8]. 'H' is the number of network links between the root nodes of the regional network and national network and it is also considered to be the number of links between root nodes of the regional and institutional network [8]. 'l' is considered to be the level of the tree where l = 0 is institutional cache and l = 2 is origin server. Also, 'C' is the bottleneck rate on the international path, where C1, CR, and CN are the transmission rates at the institutional, regional, and national networks respectively [8].

Caches are normally positioned between two separate networks at the access points to minimize travel costs through a new network, as shown in fig 2. Caches are placed at heights 0, H, and 2H of the tree i.e. in the cache hierarchy at levels 1, 2, and 3. Caches are connected to the Internet Service Provider through the access links. The capacity of the access link at every level is assumed to be equal to the capacity of the network link at C1, CR, CN, and C [8].

*B. Caching Architecture: Distributed Caching*

In Distributed caching, there are only institutional caches at the edge of the network and no intermediate caches are setup. As there are no intermediate caches available to centralize and store all the documents required by lower-level caches, institutional caches make use of other mechanisms to share the documents they contain. Some of these mechanisms are as follows [8]:

- Institutional caches can work with other institutional caches that are cooperating for the given document which are responsible for misses that occur locally, this query is usually carried out by making use of ICP (Inter Cache Protocol).
- To avoid queries or poll, institutional caches can keep a summary of the content of the other institutional caches which are cooperating. These content summaries are exchanged among the institutional caches periodically. A hierarchical structure of intermediate nodes can be used to distribute the content summary in a more scalable and efficient way. This structure only gives away the location of the document and will not retain any print of the document.
- Institutional caches can also use the Hash function for cooperating with each other. This Hash function maps a request from the client with an appropriate cache. This approach makes sure that no duplicate copies of identical document is available in other caches and make sure that caches should not know about other cache's content [8].

In the distributed cache framework, institutional caches make sure to keep a copy of every document which is requested locally. Institutional caches make sure to exchange metadata information about the documents they have, in a periodic manner. This allows the institutional caches to share local document copies among different institutional caches [8]. Every time a new document is collected in any cache, the metadata information is updated immediately at each of the institutional caches.

Programs like Napster or Scour use distributed caching architecture to provide a different kind of information stored at individual client caches. These applications make use of a central repository to keep a tab on the document cached by each client. Clients query this repository to obtain the information, such as the location of the cooperating client with the required copy of the document and the connectivity of the cooperative client. Depending upon the geographically closer location of a cooperative client or high connectivity bandwidth, a client manually selects a cooperating client. After this, the client cache makes sure to save the information locally and provide the same to other co-operative clients. Due to rapid replication of the content from one to another client, the chances of that document being available in nearby client increases [8].

### *C. Caching Techniques*

Overall, there are different types of caching techniques available and each has its own purpose and benefits. In this paper, we discuss in detail the two techniques of caching which are being used widely i.e. SQUID and Memcached.

### *D: Caching Techniques: SQUID*

SQUID is a type of web cache proxy which can support HTTP and HTTPS [9]. The main function of the SQUID is to decrease the usage of bandwidth and to refine the response time by periodically caching the web page which is requested and also reusing it. SQUID supports a lot of operating system like Windows and Linux [9].

SQUID uses an LRU (Least Recently Used) Algorithm for cache replacement and management. LRU is one of the popular algorithms for handling cache replacement and management [9]. The working of this algorithm is such that it will replace a page that has been not used for the longest time with a new page. An overview of the LRU algorithm is:

| Frame 1 | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 1 | 3 | 2 | 3 | 6 | 0 | 4 | 1 | 4 | 2 | 4 | 7 | 0 | 5 | 1 | 5 | 2 | 5 | 8 | 0 | 3 | 1 | 3 |
| **Frame 2** | | | | | | | | | | | | | | | | | | | | | | | | |
| 0 | 3 | 1 | 3 | 2 | 3 | 6 | 0 | 4 | 1 | 4 | 2 | 4 | 7 | 0 | 5 | 1 | 5 | 2 | 5 | 8 | 0 | 3 | 1 | 3 |
| | 0 | 3 | 1 | 3 | 2 | 3 | 6 | 0 | 4 | 1 | 4 | 2 | 4 | 7 | 0 | 5 | 1 | 5 | 2 | 5 | 8 | 0 | 3 | 1 |
| | | 0 | 0 | 1 | 1 | 2 | 3 | 6 | 0 | 0 | 1 | 1 | 2 | 4 | 7 | 0 | 0 | 1 | 1 | 2 | 5 | 8 | 0 | 0 |

*Figure 3. LRU Algorithm [9]*

From Fig 3 we can see how an LRU algorithm works. There are 25 string references in frame 1. In Frame 2, the strings are entered in the form of 3 in frame 2. 0 is the first string which goes to the first frame under it. The next string is 3 which goes to the first frame and replaces 0, which in return shifts down to the frame which is below it. The third string shifts the number 3 which in return replaces the number 0, and number 0 moves down to the lower most frame. The string then replaces all the string below it, in return removing the string value that has not been in the use for the longest time [9].

SQUID caching stores data from the web server i.e. from origin. This is done so that whenever a client revisits or new client requests for the data it will be available to them in an easier and faster way. SQUID send requests to the proxy server and f the requested data is present on the proxy server it serves the client's request, which is called a HIT. On the other hand, if it's not available on the proxy server, which is called as MISS, then the SQUID will move the request to the original server. The original server goes through the proxy server and gives the request to the client, where it will be stores in the cache and the forward the request to the client's

## *E: Caching Techniques: Memcached*

Many web services such as emails, social network websites/apps, maps, OTT platforms, etc. must be able to store a large amount of data and also should be able to retrieve it very quickly whenever requested. The total amount of data to be retrieved is so large that it is impractical to store an entire copy of it on each web server [10]. Instead of this, a distributed storage scheme is used, wherein web servers share multiple storage servers. Memcached is an example of such a distributed cache storage scheme.

Memcached is an open-source package that exposes data in RAM to the clients over the network [10]. When the size of the data increases more servers or RAM are added to the server. Clients use hashing to select a unique server per key. By using this technique, the entire data is presented as a unified hash table which enables the servers to be completely independent and makes sure that scaling is also achievable.

Memcached provides all the basic facility which a hash table provides, along with some extra complex operations which are built on top of it. Two basic cooperation are GET, to fetch the value of a given key and SET, to cache value for a given key [10]. Another common operation is DELETE which deletes the key-value pair. Memcached uses the LRU algorithm to add new files to the cache [10].

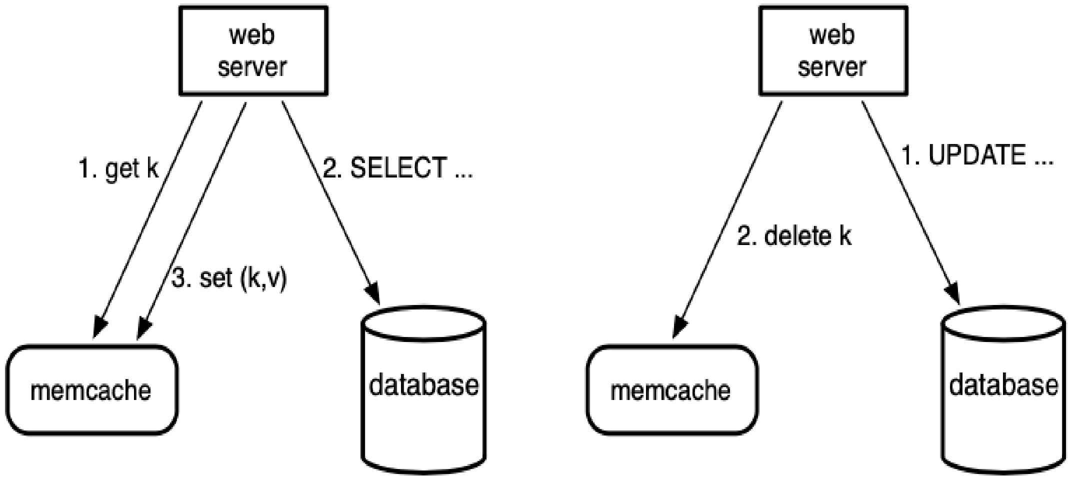


*Figure 4. Memcached as a demand-filled look aside cache [11]*

Memcached can be used as a demand-filled look aside cache as shown in Fig 4 to lighten the read load on the databases. A web server requests the value from Memcached by providing a string key [11]. If the data addressed by the key is not available in the cache then the webserver fetches the data from the original database and stores the key-value pair in the cache. For write requests, the web server uses SQL queries to handle the database and then sends a delete request to the Memcached to invalidate any previous stale data [11].

## IV. Multi-Tenant Architecture in SaaS

Multi-tenancy is an architectural pattern in which one instance of the software runs on the infrastructure of the service provider, and multiple tenants access the same instance [13]. Unlike the multi-user model, multi-tenancy requires the configuration of the single instance according to many tenants. This makes maximum use of the economies of scale, as multiple customers – "tenants" – share the same application and

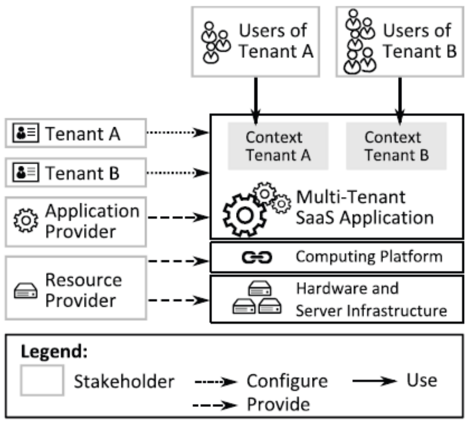


*Figure 5: Multi-Tenant Architecture [12]*

database instance. All the while, the tenants enjoy a highly configurable application, making it appear that the application is deployed on a dedicated server. [13]. “Tenants are interest groups or companies that rent a tailored application from the SaaS application provider. A tenant provides a tailored application to his own group of potential users” [13]. “For the individual users of a certain tenant, service and resource sharing is transparent (i.e., they are unaware of the fact that other users—even from other tenants or other applications—may use the same resources and applications in parallel)” [13].

In this paper, we will discuss the data architecture that can be done at the database level.

### *A. Multi-Tenant Data Architecture in SaaS*

Data architecture is an environment where the optimum degree of isolation will differ significantly based on technological and business requirements for a SaaS application. Three ways a database can be implemented considering the isolation and sharing of the metadata

*Three Approaches to Managing Multi-Tenant Data*

#### *A. I. Separate Databases*

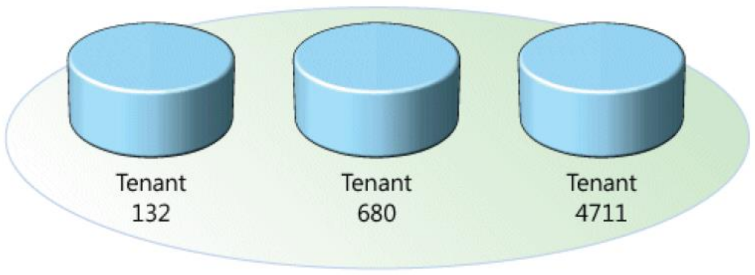


*Figure 6: Separate Database [14]*

In this approach, each tenant gets its own database. That is every table that represents the tenant particular data is created in a separate database altogether. This type of separation of data will make sure that always the right data is fetched to the request. Data won't be amiss to read from another tenant even by accident. And in terms of restoring the data back in the case of failure is also secure. Backup of the database will be taken at regular intervals and when it has to be restored the entire database is replaced and another tenant will continue to serve as before. There is no impact on another tenant.

*A. II. Separate Databases, Separate Schema*

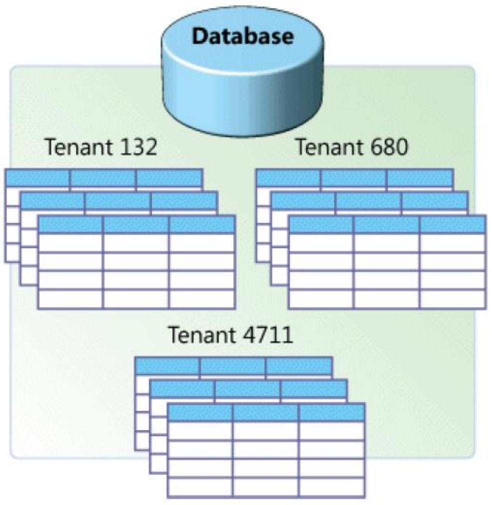
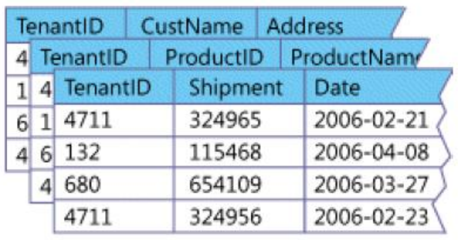


*Figure 7: Shared Database, Shared Schemas [14]*

In this method, whenever a new tenant is created in the system, all the tables that are required for the tenant are created under the new schema created to represent the tenant data. This is an example of shared implementation and it as easy as that of an isolated approach and in terms of expanding as well it is easier. As this is using the same database but a different schema, this has lesser security as compared to the isolated approach for the tenant. This approach can support a large number of tenants for a database.

The drawback of this approach is, when the failure occurs, the Whole database has to be replaced. This will mean that all tenants that are present in the database will be refreshed with the new data. Even though there is no damage that has occurred for all the tenant but has to go under such restoration.

*A. III. Shared Databases, Shared Schema*



| TenantID | CustName | Address |
|---|---|---|
| 4 | | |
| 1 | | |
| 6 | | |
| 4 | | |

| TenantID | ProductID | ProductNam |
|---|---|---|
| 4 | | |
| 1 | | |
| 6 | | |
| 4 | | |

| TenantID | Shipment | Date |
|---|---|---|
| 4711 | 324965 | 2006-02-21 |
| 132 | 115468 | 2006-04-08 |
| 680 | 654109 | 2006-03-27 |
| 4711 | 324956 | 2006-02-23 |

*Figure 8: Shared Database, Shared Schema [14]*

The third solution for data architecting a multi-tenant application is by using the same database and the same schema. Here in this approach, all the metadata will be shared across the tenants. But there will be an identifier in each table as to say which tenant particular metadata belongs to. Tenant id will be there as an identifier of the metadata. This approach requires less hardware and the cost involved in the backup is also less compared to the above-discussed approach. This can also support a large number of tenants.

As is the case of the shared schema, restoring the database replaces the data for all tenants, the same applies to shared database and shared schema approach. An alternate approach is to restore the data into a separate server and pick the required tenant data and then restore it on the production server

[14,15].

*B. Choosing an Approach*

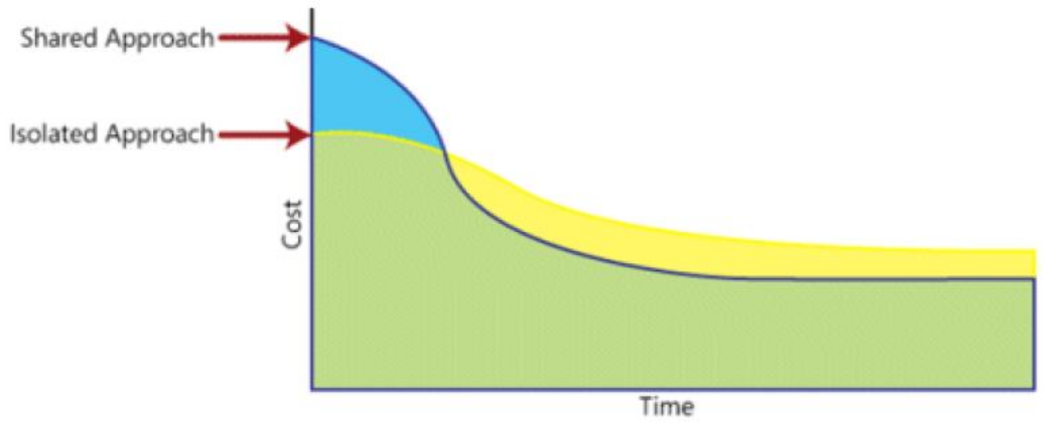


Figure 9: Choosing an approach, Yellow being the cost estimation of the isolated approach and blue being the cost estimation of the shared approach. The green colour is the overlap of colour.

As shown in the diagram, the shared database, and shared schema require development work to be more at the beginning of the project. This is mainly because of the design that involves securing the shared data architecture, which will cost more initially. Ultimately the shared approach will save money on a long duration. If the initial cost can be handled then revenue generation will happen at the later part of the workflow cycle (i.e. Once the initial design phase is over). [14,12].

*C. Realising Multi-Tenant Data Architecture*
The main concern in a shared architecture is its measure of guarantee that it provides for the security of each tenant's data. We will discuss how a security design pattern can help achieve this security concern by making sure there is no data leakage or threat for each tenant at any point in time. This can be achieved by techniques such as granting permission, SQL views, and encryption.

*D. Security Patterns*
For a SaaS application, each component of the application needs high security, and achieving this is a very important job. Sensitive information such as financial reports, trade information, transaction history, and salary information of the employees are not supposed to get leaked or put under the wrong hands, this could lead to more problems. Securing SaaS frameworks requires one to make sure high security is ensured using many layers of security that are dependent on each other in many different ways possible is a must in order to protect threats from both external and interval.

Three patterns that can ensure high security are as below:

- **Filtering:** This pattern will make sure that it sets up an environment for a request that it appears there is only one tenant particular data is present in the database.

- **Permissions:** Access control lists ACLS is used to grant access and only that particular user will be able to access the data.

**Encryption:** Encrypting the critical information of every tenant in order to make it harder to understand the data though somehow the information is obtained from the server.

Two methods typically used by the architects in a multi-tiered application environment are trusted subsystem and impersonation [16, 17]. With impersonation and database grants, the database can be accessed by individual users and all the views, procedures, tables, and other components of the database. If the user does an operation in the system which requires to make a query on the database, the application will interact with the database on behalf of the user, by impersonating the call from the user to fetch the data from the database.

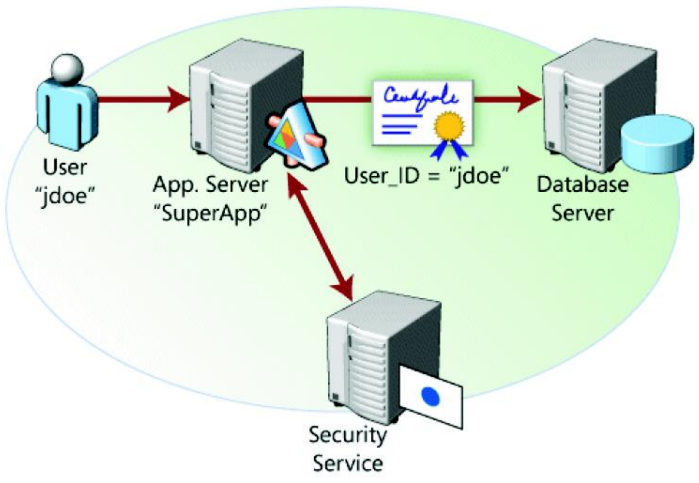


*Figure 10: Impersonation method [14]*

On the contrary to impersonation, a trusted subsystems method apart from the user's identity, there will be a separate grant that happens for the application to access the metadata for the user request and if possible, manipulate the data. In order to provide an extra layer of security, the application has to provide methods or rules that are met and then only do access to the metadata. In this approach, there is no per-user configuration of access on the database but this will enable management of security.

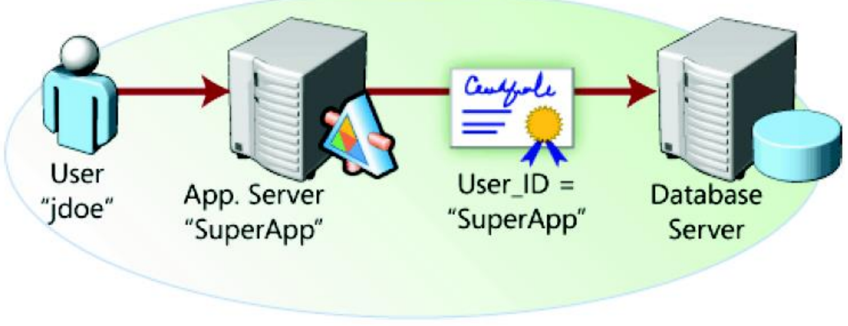


*Figure 11: Trusted subsystem method [14]*

When a SaaS application is discussed it is often the case that the word "user" is misunderstood as to referring it as an end-user and tenant. The tenant is nothing but a company/organization that uses the application in order to access/manipulate its data, which in turn is again isolated virtually from the data of the other tenant in the platform. End-user is nothing but the actual user of the application that a tenant provides grant to. End-user can leverage the features that a tenant has provided in the application. Tenant, in turn, has the complete authority to store the user data safe and secure.

In such a case, hybrid approach to access the data in the database can be employed. Where in both the impersonation is taken into consideration and also the trusted subsystems. The advantage of this is the database security is achieved to the fullest and logically it is possible to isolate the tenant's particular data.

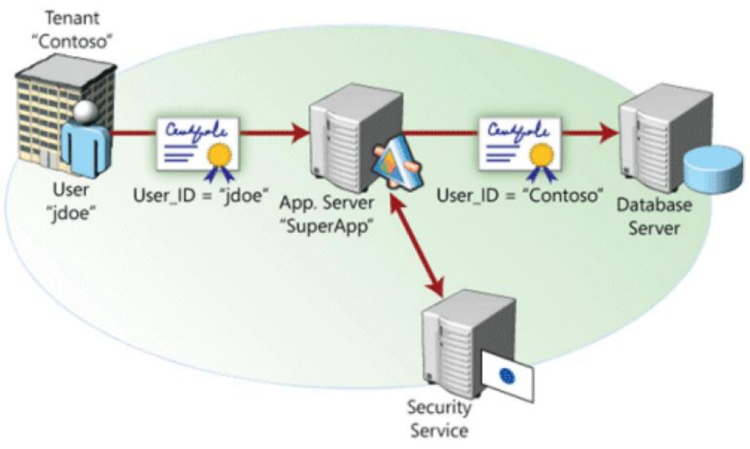


*Figure 12: Hybrid Method [14]*

The hybrid approach will ensure access control is created for all the tenants that are present in the database and by using ACLs database access is controlled by allowing grant separate permission to all the tenants. In this case, when a request is received by the server, the tenant is identified and to access the database that particular tenant credentials are used rather based on the user-specific credential. The application will not bother on identifying each user and grant the access but access to tenants' particular credentials are used and all the request is then allowed to leverage the data to be fetched or manipulated.

*E. Security Database Tables*

To secure a database on the table level, use SQL's GRANT command to grant a tenant user account access to a table or other database object:

**Query [14]:**

```
GRANT SELECT, UPDATE, INSERT, DELETE ON [TableName] FOR [UserName]
```

This command puts the user account to the ACL for that table. This has to done only once, as said in the previous section, whenever a tenant is created a separate schema is created. While creating this schema grant has to provide for the tenant to access all the tables under it. This will make sure for tenant-specific requests only the data that that tenant can access only be allowed to fetch or manipulate the data in the database.

Perfect usage for this pattern is when a separate database with a separate schema approach is used. But whereas for an isolated database approach, this need may not come, data can be isolated by restricting access at the database level. An extra layer of security can be added by granting permission at the table level.

**Tenant View Filter**

To restrict access at rows level of a table, SQL views will serve as the best possible way. This restriction can be implied for each tenant on certain rows.

SQL view is nothing but a table that exists virtually by using the SELECT query. The views then can be used to query and it appears as if it is being query on an actual database. Example for this is below, which will create a view of the Employee table which will display only rows of a single tenant:

**Query [14]:**

```
CREATE VIEW TenantEmployees AS

SELECT * FROM Employees WHERE TenantID = SUSER_SID()
```

*F. Tenant Data Encryption*
Encrypting the data inside the database will further enhance the security of the tenant data. Even the database somehow obtained it cannot be read or understood.

Two types of Cryptographic methods are as below:

**Symmetric Cryptography:** Encryption and decryption happen with a secret key. Secret key is generated initially and used to encrypt or decrypt the data in the database.

**Asymmetric cryptography (also called public-key cryptography):** Here there is two separate keys, one public key and the other is called the private key. Both the private key and its public key are related to each other. Data that is encrypted with a public key will only be decrypted with the related private key. Public keys are the ones that will be given to the client-side of the applications and private keys are generally stored in the server.

In terms of computational complexity, asymmetric stand above when compared to symmetric cryptography. This may take several thousands of times to encrypt as well and decrypting the data. As this is the case, SaaS applications which are common to have all the data to be encrypted, the performance will drastically go down. The alternate solution is the use of the central wrapping method which takes advantage from both the cryptographic method discussed above.

With this method, at the beginning of the tenant creation, there will be creation of three keys for each tenant. One asymmetric key and another key is asymmetric key pair which is consists of both public and private keys. A very strong symmetric key is created for encryption of a very sensitive data that the tenant will store in the database. And then public and a private key is used to further encrypt and decrypt the symmetric key, this will ensure the two layers of encryption.

In the process of user login, the application impersonates to get access of the tenant data using the tenant's specific security code. This application can then access the tenant's private key. And finally, the application will use the tenant's private key to decrypt the and metadata is fetched or updated based on the request [14].

### *G. Personal Experience with Multi-Tenancy*

In the projects that I have worked on, it employed a hybrid approach that was used (Impersonation and Trusted subsystem method) [20]. Each tenant is supposed to have a unique subdomain that represents the schema that the metadata will be stored. When each request to the application is received subdomain is processed, and then the application will connect the database to tenant particular schema, and then the application will impersonate itself to access the metadata required for the request. There are many different approaches where it could be implemented in order to connect tenant particular schema.

Ruby on Rails [18] web development framework has an excellent library called apartment [19] that will actually perform the processing of subdomain and connecting to the schema at the middleware level. With this, it is possible to achieve multi-tenancy. But the application lacked the encryption part that is mentioned in the above section. With the above-mentioned security pattern, any framework can implement multitenancy with a shared database and shared schema with excellent security to guarantee.

## V. Discussion & Conclusion

In this report, topics such as logging, caching and multi-tenancy were discussed in terms of their architecture and usage in real world SaaS systems. Research presented discussed the importance of developing and maintaining an architecturally sound logging and monitoring framework. Developing these frameworks also include careful consideration of the various requirements for logging and monitoring. Modularity in these monitoring frameworks outlines the reusability and flexibility in these frameworks. In this discussion it is mentioned on how to encash SaaS further with multi-tenant architecture. It discusses in detail about data architecture that helps multi-tenancy and also details about the cost that it takes in choosing each approach. In the end it discusses the design pattern that can be used for the database architecture to enhance the security and personal experience is also stated.

## Acknowledgements

| Name | Section |
|---|---|
| Reezvee Sikder | **II.** Logging & Monitoring, Review |
| Assad Nawaz | **III.** Caching, Review |
| Shravan Chandrashekharaiah | **IV.** Multi-Tenancy in SaaS, Review |
| All members | Introduction, Discussion & Conclusion, Abstract |

## References

[1] BMC, *Monitoring, Logging, Tracing*, (2019). [Online] Available at: https://www.bmc.com/blogs/monitoring-logging-tracing/ [Accessed 04 May 2020]

[2] Michael J. Kavis, (2014) *Architecting the Cloud: Design Decisions for Cloud Computing Service Models (SaaS, PaaS and IaaS,* Wiley.

[3] A. A. Z. A. Ibrahim, "PRESEnCE: A Framework for Monitoring, Modelling and Evaluating the Performance of Cloud SaaS Web Services," 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), Luxembourg City, 2018, pp. 83-86.

[4] K. Ma, R. Sun and A. Abraham, "Toward a lightweight framework for monitoring public clouds," 2012 Fourth International Conference on Computational Aspects of Social Networks (CASoN), Sao Carlos, 2012, pp. 361-365.

[5] Daniel Kasen. (2019), *How we built a logging stack at Grab,* engineering.grab.com, [Online] Available: https://engineering.grab.com/how-built-logging-stack [Accessed 05 May 2020]

[6] Chen, H. and Tan, G, (2018). "A Q-learning-based network content caching method". *EURASIP Journal on Wireless Communications and Networking*, *2018*(1), p.268.

[7] Davison, B.D., (2001) "A web caching primer". *IEEE internet computing*, *5*(4), pp.38-45.

[8] Rodriguez, P., Spanner, C. and Biersack, E.W., (2001) "Analysis of web caching architectures: hierarchical and distributed caching". *IEEE/ACM Transactions On Networking*, *9*(4), pp.404-418.

[9] Ghofir, A. and Ginanjar, R., (2017) "Distributed Cache with Utilizing Squid Proxy Server and LRU Algorithm". *Indonesian Journal of Electrical Engineering and Computer Science*, *7*(2), pp.474-482.

[10] Xu, Y., Frachtenberg, E., Jiang, S. and Paleczny, M., (2013) "Characterizing facebook's Memcached workload". *IEEE Internet Computing*, *18*(2), pp.41-49.

[11] Nishtala, R., Fugal, H., Grimm, S., Kwiatkowski, M., Lee, H., Li, H.C., McElroy, R., Paleczny, M., Peek, D., Saab, P. and Stafford, D., (2013) "Scaling Memcache at Facebook". *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*(pp. 385-398).

[12] J. Schroeter, S. Cech, S. Götz, C. Wilke, U. Aßmann, Towards modeling a variable architecture for multi-tenant SaaS-applications, in Proceedings of the Sixth International Workshop on Variability Modelling of Software-Intensive Systems, Leipzig, Germany, January 25–27, 2012, 2012

[13] Bezemer CP, Zaidman A (2010) Multi-tenant saas applications: maintenance dream or nightmare? In: Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE). ACM.

[14] Chong F, Carraro G, Wolter R: Multi-tenant data architecture. 2006. Online. Available: Accessed: 05-Jun-2011 http://msdn.microsoft.com/en-us/library/aa479086.aspx

[15] Dean Jacobs and Stefan Aulbach. 2007. Ruminations on Multi-Tenant Databases. BTW Proceedings 103 (2007)

[16] Trusted Subsystem, 2017. Online. Available: https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/trusted-subsystem

[17] Multi-tenant SaaS database tenancy patterns, 2019, Online. Available: https://docs.microsoft.com/en-us/azure/sql-database/saas-tenancy-app-design-patterns

[18] Ruby on Rails, 2005, Online. Available: https://guides.rubyonrails.org/

[19] Apartment Gem, 2014, Online. Available: https://github.com/influitive/apartment

[20] Three Database Architectures for a Multi-Tenant Rails-Based SaaS App, 2020, Online. Available: https://rubygarage.org/blog/three-database-architectures-for-a-multi-tenant-rails-based-saas-app

## Peer Based Review

### Review 1. Reezvee Sikder

*Review of Caching*

Caching was explained and described with relation to SaaS in the first section. This gave the reader a general idea of how caching was corelated to an efficient and scalable SaaS. The sections were split into two sections, caching architecture and the different types of caching techniques. Both sections were explained thoroughly with sufficient detail for the reader. In terms of critiques, I believe that the caching architectures should have been compared and contrasted giving and overview of which architectures should be used and why. The same goes for the caching techniques, the various scenarios showing where each caching technique is issued should have been given.

*Review of Multi-tenancy in SaaS*

Multi-tenancy was introduced and described along with a diagram explaining its architecture. An overview of data architecture was given in relation to multi-tenancy. Different approaches of storing and saving data in multi-tenancy systems were described. Code snippets were given to aid with describing the database approaches, this helped with the understanding of the topic in detail. Similarly, to the previous critiques, I believe that a comparison of the different methods explained and an aid to help decide which method should be used should have been given.

### Review 2. Assad Nawaz

*Review of Logging & Monitoring*

This section describes the logging and monitoring structure in the SAAS application. In this section, a general description of what logging and monitoring is explained, and then the section moves on to explain how the overall structure of the same work in SAAS. The logging structure section explains the need for Syslog and how Syslog is used as a controller, which gives a general idea about the logging procedure in SAAS. The section also introduces us to the PRESENCE framework, which is used to evaluate the performance of SAAS. The monitoring architecture section explains the relationship between manager-agent architecture which plays an important role in monitoring. In the final section, a real-time scenario from the company Grab explains how logging and monitoring can help identify the issue before it is encountered. For a critique, I felt like the PRESENCE framework should have been explained in depth which would have given a justification of how logging works and what are the actual results of logging.

*Review of Multi-tenancy in SaaS*

This section explains how multi-tenancy works in SAAS applications. The first part explains the overall structure of Multi-Tenant and three different approaches to manage multi-tenant data. Each approach of storing and saving data in the multi-tenant database were explained in depth using diagrams. This section also explains the difference between shared and isolated approaches of managing databases and how and when to choose one of these approaches. Code snippets presented in this section helps to understand the different database approach in a proper way. This section also touches a very important topic of security and explains different kinds of pattern SAAS uses. For a critique, I think a little more information should have been given on how to choose between multi-tenant database approaches under different scenarios.

## Review 3. Shravan Chandrashekharaiah

*Review of Logging & Monitoring*

Logging and Monitoring are covered adequately. In logging architecture, it is explained how Syslog works as a controller and then what is the importance of logging format and for whom and how it is important. Further PRESENCE framework is introduced to measure the performance of the SaaS application, by also mentioning the industry that is using the framework showing its importance in the SaaS application. And monitoring framework is explained with the functionality of each component giving a clear insight into the framework and how the software layer in the cloud is monitored. In the final section, the real scenario is introduced as to how a company called Grab and says how logging and monitoring can help stop or identify the issue before it is encountered. I did get the importance of logging and monitoring in SaaS applications from the entire section along the frameworks to help the same.

*Review of Caching*

In the first section under caching it is nicely described about web caching and system caching. It has given enough details by stating the differences between the same giving clear information about both concepts. In the second section Hierarchical caching architecture, I find it very easy to understand with the help of diagram and diagram is explained in detail as well. In the next section Distributed caching architecture is detailed and again following the similar pattern of comparing two architectures is followed making it easy to understand and clearing the point as to what are the pros and cons of each architecture. Further it covers two different techniques for caching. Both the techniques are covered in details with diagrams backing it. It explains the LRU algorithms used and gives use cases as when it is used.