CHAPTER 4

THE PROPOSED MODEL (CMaaS)
The desired features discussed in chapter 3 can be achieved to a great extent by realizing a centralized monitoring architecture in cloud computing. The concept of centralization (see figure 4.1) in monitoring is in fact about having mutual, correlated set of activities between the cloud providers (data centers) and user (as an organization). What is centralized? The aspects of access and audit logs are maintained in synchronous to each side. Will it hamper the cloud usage? In some way, the activities might affect the usage at least in terms of cost and performance. What are we going to gain from it? Trustable logs, verifiable logs, long term repositories for auditing and high efficiency in terms of employee’s usage as end users. Can it be integrated to other monitoring frameworks? Yes, with little variations we can achieve high degree of integration. The following sections will present the CMaaS, starting with an overview model. The overview model is the bird’s eye view of CMaaS where by one can have an idea of CMaaS whereabouts of various parts in the cloud framework. Here the user role and perspectives are discussed. The interaction model discusses a closer view where we classify on various interactions between various servers or entities. The access management part in the whole overview model is not being depicted extensively as it requires a separate attention which has been given in the auditing and monitoring of access using ABAC and BABAC section. Attribute based access control (ABAC) and bitwise attribute based access control (BABAC) are discussed. In the same section a comparison analysis of generic storage implications is also presented. Lastly, the framework of CMaaS is presented with subtle detail using Nagios monitoring system as a base. The framework is realized in the chapter 6 where a prototype is presented to demonstrate the main aspects of the framework.
4.1 The CMaaS Model: An Overview

The term centralized access management and monitoring as a service (CMaaS) contains basically 4 aspects namely:

a. Centralized access management,
b. Monitoring,
c. Log keeping and,
d. As a service.

Figure 4.1: The centralized access management in cloud computing.
4.1.1 At organization (user)

Figure 4.2a shows the client side or in other words user side of the model. Here, users are seen as an organization where the end users are employees who login to a centralized access server to gain access to a service or resource leased or subscribed at the cloud. The organizational monitor server (OM) is the one which receives request from the end users and authenticates it. Once authenticated the organizational monitor register logs and stores them. Prior to logging the OM forwards the request to the cloud monitor server (CM) with its own credentials attached. The OM also attaches the logs created so as to get synchronized at the cloud end. In this process the role of two repositories (at the organization end) is importantly depicted in figure 4.2a. The roles and policies repository maintains the roles and policies of each end user along with authentication information. The precise way of maintaining access roles and policies will be discussed later in section 4.3, where access mechanisms which storage aspects are also discussed and proposed.

Both the repositories are to be secured using certain security mechanisms which are out of scope of the thesis.
Figure 4.2a: Overview of Organizational side in CMaaS.

Figure 4.2b: Overview of Cloud Provider side in CMaaS.
The general workflow can be:

1. End user opens a browser and request for cloud service.

2. The monitor server pops up.

3. End user enters credentials.

4. Organization monitor performs checks using access repository.

5. Organizational monitor logs into the log repository.

6. Organizational monitor attaches
   
   User information + organization credentials + log (at time)

7. Sends to cloud monitor.

8. Waits for confirmation.

9. Receives confirmation from cloud monitor (+ log)

10. Sends link + session to the end user browser.

11. Synchronizes the logs.

12. User can now use the service.

13. In case the user wants to stop. He or she will logout.

14. The OM will send terminate to CM + log

15. CM will send confirmation + log in to OM.

16. OM synchronizes the logs.

17. Send link expires to users.
In other cases of terminations and or service failure etc. will be monitored at both ends at CM and OM from time to time (time step fashion) and will be logged accordingly as well as logs will be synchronized.

4.1.2 At cloud

The figure 4.2b depicts the model at cloud side. At the cloud side the monitor might be present on a cluster with separate VM allocated for monitoring alone or it can reside on a service cluster with a VM which can be shared for other services as well by the user. These cases are well elaborated in section 4.5 (evaluating CMaaS). However, if the model is specified in a ‘as a service’ perspective it is obvious that the cloud monitor occupies an independent place so as to be managed centrally as well as a scalable to include any number of users organizations. In fact, the performance aspects as well as cost aspects are fully supportive of this case as will be presented later in this chapter (section 4.5).

In the figure 4.2b it is clear that the major entities in the CMaaS at cloud are, cloud monitor (CM), cloud service cluster, organizational credentials repository and log repository. The organization credentials repository is responsible to hold the credentials of various organizations which have subscribed to cloud as well as CMaaS. This in fact is necessary so as to authenticate and synchronize organizational monitor server (OM) once the organization subscribes for subtitles for cloud. The cloud monitors verify the user organization through the credentials sent by the organizational monitor (OM) using the organizational credential repository. This is linked to the steps 6 - 9 in the previous part.

The logs once synchronized are kept in the logs repository with proper segregation and security. The logs stored at cloud/log repository must remain intact till the audit verification of logs at the organization is done. The cloud monitor will also activate/notify the services cluster
to allow a particular client request. This can be formalized through a series of ticket exchange. Internally the ways of doing this might vary from cloud to cloud. However the basic scenario can be formalized to some extent by the following steps:

1. The client the cloud monitor receives organization monitor request with authentication and user credentials information.
2. Verifies and sends confirmation to OM.
3. Forwards signal to service cluster.
4. The service cluster starts service or assigns thread, sends the handle the cloud monitor.
5. The cloud monitor logs and sends the handle and logs or monitor.
6. The organizational monitor synchronizers logs and forwards the handle to the end user.

Needless to mention that the overall communication will be secure and all data exchange will be properly encrypted. The dialogues at organization side and cloud side can be better understood by going through the interaction diagrams discussed in the next section. Also, few other aspects like synchronization, verification in flushing of logs are also presented along with basic dialogues.

4.2 The Interaction Model

At organizational level

The need for a centralized monitoring can reduce the manual intervention so as to bring more transparency in the approach. As shown in the use case in figure 4.3 the traditional approach in monitoring is facilitated by introducing a user as organizational monitor. This human intervention is responsible to verify the logs and maintain repository by viewing logs as well as generate reports (figure 4.3). The major part which is automated is the access mechanism using access strings that is configured by the admin of the organization. The access strings will be configured based on the access policy and roles of the users. To introduce automated
and centralized monitoring the use case is revised as shown in figure 4.4. Thus, the user intervention in monitoring is minimized to lowest level. The monitoring and logging will be based on the customizable monitoring strings and alert strings similar to that of access strings. Thus, the whole access and monitoring management will be based on access strings (user policy and roles), monitoring strings (log event types) and alert strings (event types for which alerts will be generated). The configuring of these strings will be done by the organizational admin. The role of the human monitor will be limited to that of viewing logs and generating reports only.

Figure 4.3: Traditional monitoring use case.
At the cloud provider's level

In the centralized approach, modelling organizational usage would be inappropriate without considering the broker entity as marked by Calheiros, R. N. et al (2011) into the purview. Datacenter broker or cloud broker carries out negotiations between the application providers and the cloud service providers based on the application QoS given by Ferretti, S. et al. (2010) requirements. As shown in the use case at service provider’s level and Data centre (figure 4.5) the monitoring is shown to be done by the Data centre only rather than the broker. As shown in figure 4.5, the broker’s will be mainly responsible in configuring and performing access as
well as configuring the monitoring. Whereas, the DCMonitor will be responsible in managing logs repositories, monitoring, assigning VMs to process logs, calculating monitoring cost, support verification of logs and notifying the user level monitor for logs and usages.

Figure 4.5: CMaaS use case at providers.

To understand the CMaaS it would be necessary to carry out an interaction model whereby the object/entity activation and deactivation as well as a sequence of method calls or procedure
calls by communication can be depicted in a more precise manner. The main idea of presenting the interaction model via set of sequence diagrams is to have a better perspective. However, the sequence diagrams hereby are not meant for design restrictions. In other words the interaction diagrams are for perspectives and are not meant for implementation guidelines. The motive of the whole exercise is to generate and understand the motive, the implementation aspects can be realized in many different ways found suitable to various conditions. Sequence diagrams are the best way to depict dialogues and their sequence, especially in a classical peer-peer models as our CMaaS is. There are mainly 6 scenarios that need to be understood, namely:

a) User request and getting services/resources (access).

b) Event of change of user state

c) Login failure

d) Service failure

e) Verification and flushing of logs

The scenario a) is being talked about in previous section whereby the sequence of steps were discussed. However, the other scenarios were not discussed in previous sections, which we can have an idea by looking in the following subsections.

### 4.2.1 Getting services and resources access

The cloud monitor or the DCmonitor shown in figure 4.6 works in tandem with the cloud service broker. The ORGmonitor and access monitor also work together to achieve access. However, the access part is minimally active compared to throughout active ORGmonitor. The access request is made at the user end to begin with. The request might be a login page of the organization served through its ORGmonitor and access servers. The request along with access string is received and logged at ORGmonitor. The access string will be the state, role,
resource or service request, authentication etc. bundled in one string. Once logged, the ORGmonitor will forward the request to access server to get it verified. The access server verifies the authentication with proper access mechanism as discussed in the next section. It returns ‘ok’ to the ORGmonitor which then reverts back to the end user. The end user then makes a service request of a particular type (fine grained). The ORGmonitor is now in full connection oriented mode (via threads) with the end user and it logs the service request and forwards it to the broker. The forwarded package will be holding timestamps as well as monitoring string. The main part of the monitoring string can hold the organizational credentials and user credentials with service-specifics. It is to be noted that timestamps at both organization and cloud sites have to be principally maintained and synchronized before final logging. Broker does few things in sequence. Firstly, it forwards the request to the DCmonitor to log. Secondly, it’s verifies the credentials sent and waits for the DCmonitor to send acknowledgement of log recorded. The broker then forwards service activation command to the service cluster and once acknowledged records the service start timestamp. The service start can be a thread assignment only. This vitally proves the most desired aspect of monitoring. That is, fine grained access monitoring as discussed in previous chapter (3.4). A service can be even of an emailing; even this needs to be logged as desired. Once the user thread is formed the start and end of that thread can be monitored and logged. In the further track of the sequence diagram one can understand it more closely. The cluster manager sends the handle of the allocated resource (or thread) to the broker which forwards the service timestamps to the DCmonitor, which is logged at once by the monitor. The DC monitor sends the log to the ORGmonitor to be logged in synchronous fashion. Meanwhile, the service handle is send to the end user by the broker and the service is assumed to be started. Once the user ends his work, job completion message (log off) is sent to the broker at the datacenter and the broker forwards it to the monitor to be logged and also to the cluster manager to free the resource (or thread). Thus the service stops and ends. As discussed in section 3.4, we have a desired feature of long term logs. In many cases storage seems simple. However, storage logs at long term at DCmonitor might be an overburden to the datacenters as well as costly. It might sometimes be useful to get digitally signed logs from the datacenters and verify with its own and then instruct the DCmonitor to flush its logs. Logs kept outside the boundary of
organization sometimes can be very unsafe and they may not be acceptable to organizations. The next section depicts the verification and flushing of log sequence diagrams.

### 4.2.2 Verification and flushing of logs

Figure 4.4 depict the flushing of logs. The organizational admin decides to verify the logs. The verification of logs can be initiated at the top management level. The organizational admin (ORGAdmin) has rights to log into the access dashboard (explained in the section 4.4) and generate verify event. The logs stored so far will be bundled and hashed and sent to the datacenter monitor by the organizational monitor. The datacenter monitor matches the hash received by its own hash of logs and sends a confirmation along with its own hash back to the DCmonitor, which sends the confirmation back to the admin. The admin can send a follow up event to flush all logs at datacenter. DCmonitor will be saving the logs so far with confirmation hash at secondary or tertiary storage and sends a flush signal to the DCmonitor. The DCmonitor deletes its copy of logs and frees the space. However, the DCmonitor keeps the hash for later verification of history logs. New log files are initiated and times synchronized at both ends before completion. The major aspects of verification and flushing are the secure mode of verification which increases the trust among the organization as a user and the service provider owing to the desired feature of trust in the previous chapter (chapter 3). To better describe the monitoring process of the access management, one has to understand what happens when certain failure occurs. The following sub section 4.2.3 and 4.2.4 discusses the failure interaction model for login and service. The login failure is an instance when the user enters wrong credentials and service failures are instances when the service cluster manager detects a failure at the datacenter and notifies the DCmonitor. Apart from these two failures, another kind of failure which needs to be mentioned here is the failure of authentication if the state of users gets changed. For example, role of a user gets changed and this might disallow him to use certain service or resources. These instances are depicted in the section 4.2.5 (change in state).


4.2.3 Login failure (end user)

Figure 4.7 depicts a case where a user tries to login for a service or resource with wrong credentials. It can be either wrong credentials or unauthorized entry. The sequence diagram shows that even a wrong credentials login will be logged and synched with the DCmonitor. Figure 4.8 depicts the scenario of a service failure. The service cluster manager notifies of a service which logged and synched at both sides monitors. The DCmonitor can later issue a service restart command to the service cluster manager or take evasive/corrective actions.

4.2.4 Role or State change

The role/state of an employee can be changed via access dashboard. On role/state change the access will send a notice to ORGmonitor to log. The access will re-verify the user for the current access he/she is having. On failure the fail event will be notified to ORGmonitor which then will synchronise with the DCmonitor for logging as well as stopping the service to the particular user and notifying him/her for his/her role/state changed (see figure 4.9).

The interaction model, following use cases shows the sequence of interactions in form of sequence diagrams. The sequence diagrams reveal the flow of dialogues and major activities at the users and provider’s level. The sequence diagrams of few activities like connecting and using service (figure 4.6), verifying and flush logs (figure 4.10) and log event from broker to DCMonitor (figure 4.11) are important to be mentioned to give a clear picture of the whole process of monitoring. The end users connecting to a service provider will pass through the access and monitors at both users level monitor through broker to DCMonitors as shown in figure 4a. The service is rendered by the user cloudlets as marked by Calheiros, R. N. et al (2011) received by the broker and processed at datacenter by holding one or more Virtual Machines (VM) depending upon the workloads, number of end users, and SLA [SLA INFORMATION ZONE, url: http://www.sla-zone.co.uk/]. The processing of logs will be done by the monitors at data centres or separate monitor VMs reserved for monitoring or shared VMs held by the brokers. Later in the cost analysis we will look into the cost.
implications with regard to both these cases. Logs that are generated synchronously at the data center and, at the organization needs to be verified for consistency from time to time. The logs can be verified and can be flushed later on as shown in figure 4.10. The repositories of logs can be maintained for a long time, but needs to be flushed so as to minimize storage implications. The log event will be generated by the broker based on the monitor string as shown in figure 4.11. However, the processing a log and forwarding it to the user monitor for recording will also take place with consistent time stamping as discussed in the book by George Coulouris.

Figure 4.6: Sequence diagram for user request and service in CMaaS.
Figure 4.7: Login failure in CMaaS.
Figure 4.8: Service failure in CMaaS.

Figure 4.9: Service stopped: Role changed
Figure 4.10: Sequence diagram for verify and flush logs.

Figure 4.11: Sequence diagram for generation of log events.
4.3 Auditing and Monitoring of Access Using ABAC and BABAC

As discussed in chapter 3 section 3 (Audit and Access Management), if access logs are maintained through the access management then the job of monitor becomes lesser burdened as well as the need for monitoring agents is minimized. This is realized only when we have a centralized approach of access and monitoring. However, maintaining long term logs certainly has storage implications, especially with larger organizations. The storage implications need to be studied.

4.3.1 ABAC Based Audit Log Maintenance

The ABAC model can be used to have a structured logs maintained for audit purposes and having fine grained policy frameworks. The aspect of maintaining long term logs can be realized without losing too much space.

Policies $\rightarrow$ access $\rightarrow$ policy based audit logs

A typical scenario of maintaining ABAC based audit logs is shown in figure 4.12.

Thus, audit logs for each request can be maintained in a structured and precise fashion. Also, we can have automated analytics for the auditors using the ABAC audit logs. The audit log can be formalized based on the ABAC model as follows:

$\text{Attr}(e); \text{Attr}(s); \text{Attr}(r); \text{Access}(s, r, e); \text{Matched\ Violation}$
The time stamp is assumed to be integral part of access. Matched violation, is the factor or measure of violation (if any) that has occurred due the access. For example, in most of the cases the user tries to access the resource off time or during the time the resource is under secret editing. Although, the access may not be granted to the resource but, the move by an employee to access it is in fact a violation. The measurement of violation factor can be established by the one to one mapping of all possible violations with number sequences with attached weightage.

\[
\text{Violation}_\text{Set} = \{V_1, V_2, V_3, \ldots, V_n\}
\]

\[
\text{Offence}_\text{weightage} = \{W_1, W_2, W_3, \ldots, W_n\}
\]

The Table I shows an example set of violation types and corresponding offence weightage based on the seriousness of the violation.

---

Figure 4.12: Attribute Based Access Control.
Table 4.1: Example resource violations.

<table>
<thead>
<tr>
<th>Violation</th>
<th>Offence Weightage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessing Resource/service not ready</td>
<td>0.3</td>
</tr>
<tr>
<td>Login Failure on R</td>
<td>0.01</td>
</tr>
<tr>
<td>Resource/service not ready (non-confidential)</td>
<td>0.1</td>
</tr>
<tr>
<td>Accessing Resource/service not allowed</td>
<td>0.2</td>
</tr>
<tr>
<td>Editing Resource/service finalized (edit offline)</td>
<td>0.8</td>
</tr>
</tbody>
</table>

### 4.3.2 ABAC Based Audit Log Maintenance

We propose another access model i.e. Bitwise Attribute Based Access Control (BABAC), where, the methodology of maintaining fine grained policies and roles is in bitwise fashion. Also, the mapping of policies and roles onto access is bitwise, which is more compact form than the ABAC model. This can lead to save more space in log maintenance. The user roles are maintained as states and the mapping is just done by ‘AND’ing the resulting policies and obtaining the access to a set of fine grained services/resources. The figure 4.13 depicts the access in BABAC model of mapping.

The process of mapping is simple, however, maintaining these kind of policies can be little complex, however, one can make automated system consoles which can form these Bitwise policies with very less effort as in case of ABAC. In case of ABAC the Granularity and scaling can be done resource-wise whereas, in case of BABAC the scaling is restricted to an allowable bit size. In fact, methods can be devised to come out of these constraints as well. The audit log can be formalized based on the BABAC model as follows:

\[
States_word(s, r, e); Access_word(s, r, e); Matched_Violation
\]
4.3.3 Analyzing the Storage Implications of CMaaS audit logs

A simulation in cloud environment was made to study the storage requirements of various methods discussed above. The simulation was carried out for 3 aspects of (long term) log maintenance while monitoring:

a. Generic monitoring.
b. Generic monitoring with ABAC logs;

c. Generic monitoring with BABAC logs;

Cloudsim tool [Calheiros, R. N. et al (2011)] was used to simulate for the above factors. The simulation was made for 10 to 20 users with subjects and resources scaling from 10 each to 20 each. Although this might not be a real scenario but its more than enough to get a good idea about the storage implications of the long term log keeping for auditing purposes.

To understand the study of the logs we have to go along the three most important part of the ABAC logs and BABAC logs, namely:

a. Subject Pattern

b. Resource Pattern

c. Environment Pattern

*In case of ABAC:*

Typically, subject pattern involves login_name, general information and state/role. The criteria for the state/role can be in a range (from no role to all roles). An organization can have as large as 50 roles on a particular application. Studies can’t be made on all aspects of role. However, a typical 10-20 role subject pattern is studied in our simulation. That is, a user might not have any roles as well as at the same time a user can have 20 roles. The subject pattern is shown in figure 4.14a.

Resource pattern can depend upon number of atomically usable resources leased in cloud by the organization. It can be a database, it can be a file, it can be an application. It can be parts of application, it can be an OS or other platforms. Every resource can have access lists(typically ranging from 10-20) is classified by resource type, name and access list. Access list can be the set of roles that are allowed access on a particular resource. Thus, a resource pattern can be as shown in figure 4.14b.
However, mostly the resource pattern will be owing to access list in the same manner as roles in subject pattern. The environment pattern can mostly contain time ranges. In some cases they can be holding specifics but, in our case we will be filling it as constant. Hence, the log space can be calculated as:

\[
Log_l = P_{s_i} + P_{r_i} + P_e + S_v
\]

Where, \(P_{s_i}\) = Size of subject pattern for ith log, \(P_{r_i}\) = Size of resource Pattern for ith log, \(P_e\) = Size of environment pattern (constant) and \(S_v\) = Size of violation information.

In case of BABAC:

In BABAC model, the roles are made in bitwise fashion. Hence, scaling factors may arise. However, for 0-20 roles 4 bit is required which can be scaled to 128 roles on 8 bit. However, the user attributes remains the same. The 128 roles bit can contain environmental constraints.
also. As shown in figure 4.15a) the login name and general information parts are same as in case of ABAC. However, the roles are now bitwise represented and given a space of 2 bytes (a word).

![State Pattern (BABAC)](image)

Figure 4.15a) State Pattern (BABAC).

![Resource Pattern (BABAC)](image)

Figure 4.15b) Resource Pattern (BABAC).

The resource pattern will hold the resource name and resource type as in ABAC but the access list can be fixed to 16 bits (2 bytes) (see figure 4.15b). However with 0-20 accesses, which will make us add 1 more word (after 16 accesses). In BABAC the environment pattern is part of the states in the subject pattern only. Thus, the overall log space can be calculated as:

\[
Log_i = P_{s_i} + P_{r_i} + S_v
\]

Where, \(P_{s_i}\) = Size of subject pattern, \(P_{r_i}\) = Size of resource Pattern for ith log, \(S_v\) = Size of violation information.
4.4 Proposed Framework of CMaaS

Section 4.1 and 4.3 discusses the overview model and the structural building blocks of the CMaaS. However, to realize the overall model into the implement-ability aspects, it is very essential to view the model in perspective of currently used monitoring systems. Nagios [http://www.nagios.com] is one such system which we have kept as base. The implementable CMaaS model presented hereby constitutes the interface model with the Nagios. The major chunks of the model can be very easily formulated to interface with other open source monitoring systems like Ganglia[http://ganglia.sourceforge.net] etc. The architectural and functional aspects of Nagios and Ganglia are discussed widely in the previous chapters (ch 2.5). The model presented here is mainly aimed at giving a perspective of implementing the basis of CMaaS while having very little changes in the ongoing monitoring system. However, the best way to implement CMaaS is to extend the logs of the current access management at organization to the monitoring at cloud. The Model is not a complete design and hence the reader / implementer will have to make it as a basis to implement CMaaS. The prototype presented in the chapter 6 will give further idea about implementation aspects. The model is presented as modular overview, then architecture and flow of certain important activities and synchronization.
4.4.1 CMaaS framework (with Nagios)

We propose a framework and first present an overview of the working of the CMaaS. The figure 4.16 gives an overviewed idea about its working. Each host in the cluster will hold the Nagios plugin. Similarly, on every host access plugin will be there to capture access data of specific and overall fine grained services on that host. The infrastructural/QoS specific logs gathered by the Nagios plugin will be sent to the core running at monitor server along with the access logs.

More formally the access logs will be sent in an event oriented fashion whereas the QoS metrics can be sent in a time step fashion. That is, for an instance, a user logs in and starts service A the log will be formed then which is based on the event of access. The availability of that server can be checked periodically by an agent or plugin and can be logged. The monitor will consolidate the logs and keep it for recording.
The CMaaS model was first introduced in section 4.1 with only an overviewed idea. One can establish the overview diagram in figure 4.1 with that presented in figure 4.16. In the figure 4.16 closer aspects of Nagios or Ganglia’s roles are depicted. Majorly the monitor servers at both cloud and organizational boundaries are to be holding Nagios or Ganglia cores, which will be continuously listening to the agents or plugins at hosts or access servers.

The monitors can be configured for either time step or event step fashion as shown in figure 4.17. The access logs generated at the access server at the organization will be rendered in event step fashion to the monitor server at organization.

The organizational monitor and the cloud monitor will synchronize in automated time step fashion or in event oriented fashion generated by the organizational admin. Organizational admin can set the metrics at both sites through the audit dashboard (figure 4.18). The repository module will be responsible to compact the logs and secure it as well as provide synchronization and verify services through the audit dashboard events.

The framework is designed to integrate with the Nagios. However, it can very well be set to integrate with other monitoring frameworks like Ganglia etc. The interface model is shown in figure 4.19. The Nagios monitor agent (plugin) will read from metrics manager module interfaced with the CMaaS agent to send logs. Similarly, the access agent reads from metrics manager and interface with CMaaS to send logs. At the organization site the access server interface with CMaaS server and both are interfaced with the dashboard, sync and verify modules. The networking layer for carrying out sync, verify etc are present as part of cloud sync and organization sync modules at every host server in hosts cluster as well as monitor server at organization.
Figure 4.17a) Monitoring Flow at cloud data center core (Event Step).

Figure 4.17b) Monitoring Flow at cloud data center core (Time Step).
Figure 4.17 c) Time Step Sync and verify for audit (Black rectangle is the line to mark communication channel).

Figure 4.18. CMaaS Architecture.
Call to various methods while the functioning of the overall monitoring in CMaaS is shown in figure 4.17 a) b) and c). The general methods are:

1. `get_fm_Nplugin()`
2. `get_fm_Aplugin()`
3. `record_log()`
4. `record_log()`
5. `get_recorded_log()`
6. `recieve_org_log()`
7. `send_record_to_cloud()`
8. `recieve_cloud_log()`
9. `Store_log()`

Figure 1.19: Interface Modules in CMaaS.
4.4.2 Secure Logs Synchronization

The exchange of recorded logs will be done in a full SSL model in the dialogues shown hereby:

For a time period from $i$ to $j$:

1. At cloud at $t_{j+1}$: $Hash\_Encrypt(T_p + \sum_{t_i}^{t_j} CD + \sum_{t_i}^{t_j} CA) \rightarrow C_{dig}$ (digital signature)

2. At organization at $t_{j+1}$: $Hash\_Encrypt(T_u + \sum_{t_i}^{t_j} UD + \sum_{t_i}^{t_j} UA) \rightarrow U_{dig}$ (digital signature)

3. At organization at $t_{j+2}$: $Compare(C_{dig}, U_{dig}) \rightarrow Verification$

Where,

$T_p \rightarrow$ Task metric at provider.

$CD_{t_j} \rightarrow$ Data Center monitored log.

$CA_{t_i \rightarrow t_j} \rightarrow$ Cloud Access monitored logs.

$T_u \rightarrow$ Task metric at user organization.

$UA_{t_i \rightarrow t_j} \rightarrow$ Access monitored logs at organization.

$UD_{t_i \rightarrow t_j} \rightarrow$ Data center monitored logs synched with User organization.