THE CLOUD STORAGE ENGINE 1.0

Last modified: 6 July 2014

1 Table of Contents

2 Introduction

ClouSE is a high-reliability and high-performance transactional storage engine that uses a cloud storage utility provider (such as Amazon S3) to store user data. Key advantages of ClouSE include:

- It provides transactional (ACID) guarantees to the users, with transactions featuring commit, rollback, and crash-recovery capabilities to protect user data.
- It allows utilizing cloud storage to store user data.
- It supports encryption of user data to guarantee its confidentiality in the cloud.
- It supports direct access to blob content (see Weblobs) to enable scaling out content delivery of large data objects (e.g. pictures, movies).

With ClouSE users can offload storage management to a cloud storage utility provider without granting access to the data, while enjoying full capabilities and guarantees of a transactional SQL database (even though the cloud storage utility provider may have basic capabilities and limited consistency guarantees).

ClouSE supports the following features:

Storage limits	256TB[1]	**Transactions**	Yes	**Locking granularity**	Range
MVCC	No	**Geospatial data type support**	Yes	**Geospatial index support**	No
B-tree indexes	Yes	**Hash indexes**	No	**Full-text search indexes**	No
Clustered indexes	Yes	**Data caches**	Yes	**Index caches**	Yes
Compressed data	Yes	**Encrypted data**	Yes	**Cluster database support**	No
Replication support[2]	Yes	**Foreign key support**	No	**Backup / point-in-time recovery**[3]	Yes
Query cache support	No	**Update statistics for data dictionary**	Yes		

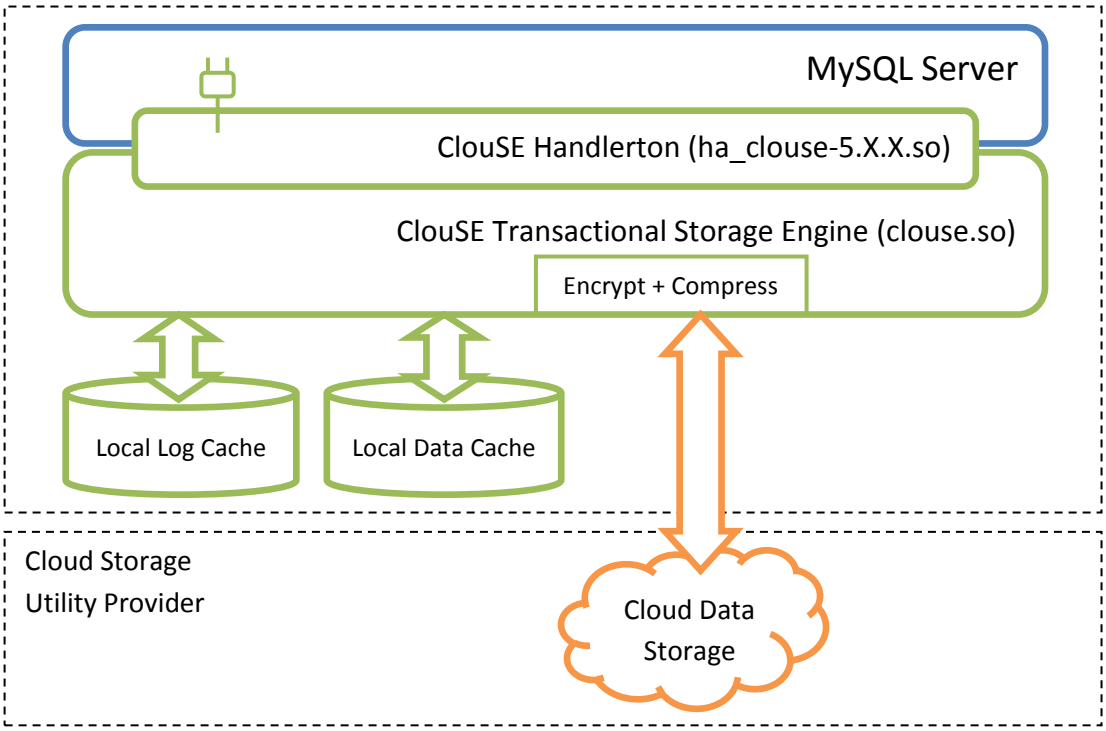
[1] Relational data storage only, weblob storage is not limited.

[2] Implemented in the server rather than in the storage engine

[3] Backup for data protection is not needed, data protection is provided by a combination of exceptional durability of the cloud storage and built-in point-in-time recovery capabilities.

3 Overview of ClouSE Architecture

The high-level architecture of ClouSE is shown here:



The ClouSE transactional storage engine manages the following data:

- a copy of the latest transaction log is stored locally
- a persistent cache of recently accessed relational data is stored locally
- relational and weblob data is stored in the cloud storage

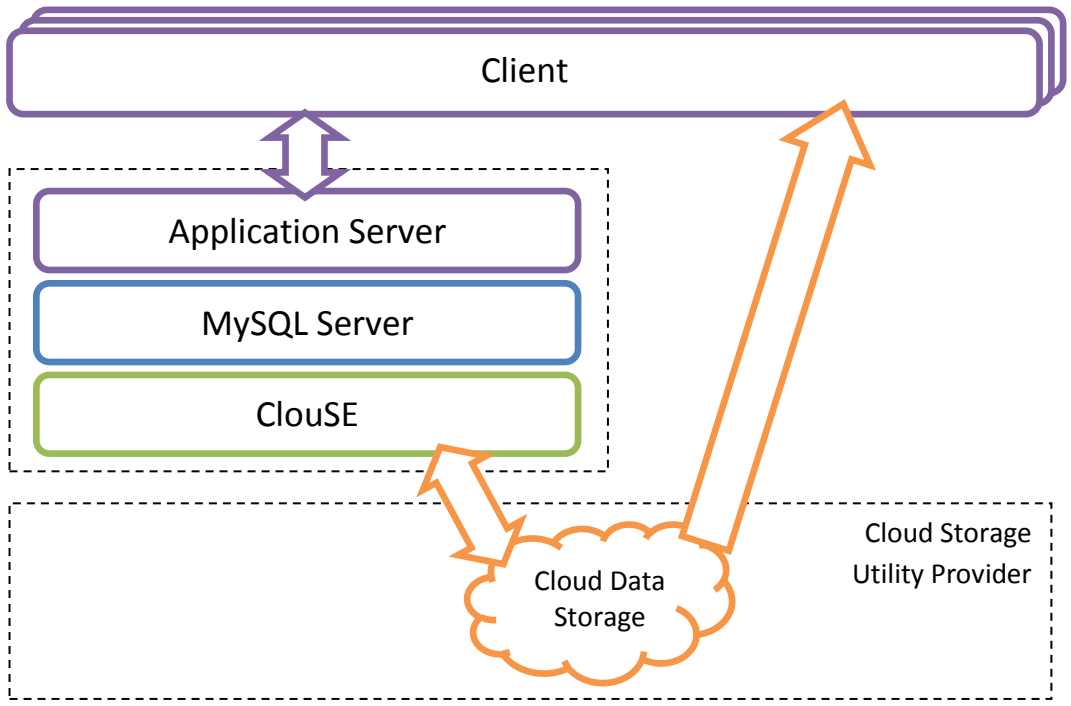
The relational data is encrypted before it's sent to the cloud storage provider to provide protection from accidental or malicious misuse. It's also compressed to save network bandwidth and storage.

The ClouSE transactional storage engine implements functionality for transaction control, record-oriented access methods with fast key and range lookup, direct access to blob content (see Weblobs), key range locking, and in-memory cache of recently accessed relational data for fast access.

The ClouSE handlerton implements the MySQL's storage engine plugin interfaces to expose ClouSE to MySQL. The ClouSE is loaded into MySQL process as a pluggable storage engine.

4 Weblobs

Weblobs[4] enable direct access to blob content. From the database perspective, weblobs are regular blobs and fit into the ACID model; however they also expose URLs that can be used by the client (e.g. a web browser) to download the weblob content directly from the cloud storage utility provider. The concept can be illustrated by the following diagram:



With weblobs the server just needs to provide control data (i.e. the weblob content URLs) to the client, and the client can leverage capacities of the cloud storage utility provider to download the content. For some classes of applications this could lead to significant scale-out, for example applications that deal with large amount of multimedia content (pictures, movies, etc.) will benefit a lot from storing multimedia content in weblobs, as the server would need to serve only a tiny portion of the content and let the cloud utility provider deliver the rest.

4.1 Weblob Syntax and Usage Patterns

In MySQL a weblob is expressed via a pair of BLOB fields that have a special naming convention: *field_name*$wblob and *field_name*$wblob_info. The following example shows how to create the **pictures** table that is keyed by the **id** and has a **picture** weblob:

```
CREATE TABLE pictures (id BIGINT KEY, picture$wblob LONGBLOB,
picture$wblob_info BLOB) ENGINE=CLOUSE;
```

[4] Web-accessible blobs.

The *field_name*$wblob field is the base weblob field can be used to access the weblob content. From the MySQL perspective it can be used as a regular LONGBLOB field. All operations with weblobs happen under transaction control and are fully compliant with the ACID model. The following example shows how to insert and select weblob content[5]:

```
INSERT INTO pictures VALUES (42, 'MYPICTUREDATA', NULL);
SELECT picture$wblob FROM pictures WHERE id=42;
```

The latter command would produce a result like this:

```
+---------------+
| picture$wblob |
+---------------+
| MYPICTUREDATA |
+---------------+
```

The *field_name*$wblob_info is a weblob information field that can be used to get the direct cloud storage utility provider URL of the weblob content (on select), as well as to specify weblob control commands (on insert / update). The following example shows how to select the direct cloud storage provider URL:

```
SELECT picture$wblob_info FROM pictures WHERE id=42;
```

The result would look like the following:

```
+-------------------------------------------------------------------+
| picture$wblob_info                                                |
+-------------------------------------------------------------------+
| https://s3.amazonaws.com/mybucket/db/wblob/15E30<...>QSr2DM/wb  |
+-------------------------------------------------------------------+
```

The result varies widely as the URL string is randomly generated. The resulting URL can then be used to download the content that was previously written into the **picture$wblob** field.

On insert / update the *field_name*$wblob_info should either be NULL / empty or should contain one or more weblob control commands separated by a semicolon. Weblob control commands can be used to:

- implement upload streaming that allows the application to upload large objects (e.g. pictures, movies) piecemeal
- specify a name for the weblob URL

[5] A text string is used here for illustrative purposes; generally a picture contains binary data in an image format such as JPEG, GIF, PNG, etc.

4.1.1 Weblob Streaming command

The streaming command can take one of the following values: first, next and last. All values must be in the lower case. streaming:first indicates that the value specified the *field_name*$wblob field is the first part of the content. streaming:next indicates that the value specified the *field_name*$wblob field is a middle part of the content. streaming:last indicates that the value specified the *field_name*$wblob field is the last part of the content. The following example shows how to insert the **picture** content in four parts:

```
START TRANSACTION;
INSERT INTO pictures VALUES (74, 'ONE', 'streaming:first');
UPDATE pictures SET picture$wblob='TWO', picture$wblob_info='streaming:next'
WHERE id=74;
UPDATE pictures SET picture$wblob='THREE', picture$wblob_info='streaming:next'
WHERE id=74;
UPDATE pictures SET picture$wblob='FOUR', picture$wblob_info='streaming:last'
WHERE id=74;
COMMIT;
```

The following example shows how to update the **picture** content in two parts:

```
START TRANSACTION;
UPDATE pictures SET picture$wblob='ONE', picture$wblob_info='streaming:first'
WHERE id=42;
UPDATE pictures SET picture$wblob='TWO', picture$wblob_info='streaming:last'
WHERE id=42;
COMMIT;
```

The following example illustrates the effect of the abovementioned modifications:

```
SELECT id, picture$wblob FROM pictures WHERE id=42 or id=74;

+----+-----------------+
| id | picture$wblob   |
+----+-----------------+
| 42 | ONETWO          |
| 74 | ONETWOTHREEFOUR |
+----+-----------------+
```

Note: Only one weblob streaming operation can be in progress in a transaction and must be completed by the time transaction commits. If a transaction is committed while a weblob streaming is in progress it's going to be forcibly rolled back. Thus in the autocommit = 1 mode attempts to use weblob streaming without starting an explicit transaction would always fail due to an implicit commit happening while weblob streaming is in progress.

The most powerful usage model for weblobs is when the *field_name*$wblob field is only written into, but never read from; instead the *field_name*$wblob_info is used to get the cloud storage provider URL so that it could be passed to the client for direct download. For example, a web application that provides picture sharing functionality could use the streaming commands to upload pictures into the **picture$wblob** field and then select the **picture$wblob_info** field to

embed the URL in the <img …> tags so that the web browser would download the pictures directly from the cloud storage utility provider.

4.1.2 Weblob Name command

The name command can be used to specify a name for the weblob URL. A name is the last part of the weblob URL after the last slash. If a name is not specified the weblob URL will have "wb" as a name. The following example illustrates specifying the pic.gif name:

```
INSERT INTO pictures VALUES (24, 'ONE', 'name:pic.gif');
SELECT picture$wblob_info from pictures WHERE id=24;

+-------------------------------------------------------------------------+
| picture$wblob_info                                                      |
+-------------------------------------------------------------------------+
| https://s3.amazonaws.com/mybucket/db/wblob/15E30<...>QSr2DM/pic.gif |
+-------------------------------------------------------------------------+
```

The name also determines the content type (a.k.a. MIME type) associated with the weblob content. ClouSE has an extension-to-type map, similar to ones that http servers have. The map associates the extension (the part of the name after the last dot, e.g. pic.gif would have **gif** extension) with the content type. For example, the **gif** extension is associated with the **image/gif** content type. If the extension is not known or there is no extension, the content is going to have the **application/octet-stream** content type.

A name command can be combined with the streaming:first command to specify the name for the streamed weblob's URL. The commands should be separated by a semicolon. The following example illustrates combining name and streaming commands:

```
START TRANSACTION;
INSERT INTO pictures VALUES (88, 'ONE', 'streaming:first;name:img.jpg');
UPDATE pictures SET picture$wblob='TWO', picture$wblob_info='streaming:last'
WHERE id=88;
COMMIT;

SELECT picture$wblob_info FROM pictures WHERE id=88;

+-------------------------------------------------------------------------+
| picture$wblob_info                                                      |
+-------------------------------------------------------------------------+
| https://s3.amazonaws.com/mybucket/db/wblob/15E30<...>QSr2DM/img.jpg |
+-------------------------------------------------------------------------+

SELECT picture$wblob FROM pictures WHERE id=88;

+---------------+
| picture$wblob |
+---------------+
| ONETWO        |
+---------------+
```

4.2 Weblob Performance Considerations

When ClouSE uploads and downloads weblob content each request goes directly to the cloud storage utility provider, it does not go through the buffer pool that caches relational data. This implies that selecting a *field_name*$wblob field could be expensive both in terms of runtime performance and in terms of transfer cost, as it transfers all weblob content from the cloud storage utility provider.

Another thing to consider is that MySQL always allocates a contiguous chunk of memory that would be able to hold the field's value. For regular fields it works very well, but for weblobs it could create memory pressure or even make MySQL run out of memory if content of individual weblob fields is large. On upload the streaming functionality solves the problem and allows uploading large weblob content with constant memory usage. However, on download, memory needs to be allocated to hold the content of all selected weblobs thus selecting a *field_name*$wblob field could consume a lot of memory. To mitigate the problem ClouSE imposes a limit on downloading individual weblob content, so a select query that selects a *field_name*$wblob field would fail if it fetches a record that contains a weblob larger than the limit. The default limit is 16 MB and can be configured to a value between 1 KB and 1 GB using the clouse_max_weblob_download_size configuration option.

Due to the considerations above it is recommended to avoid using *field_name*$wblob in select statements (and in WHERE, ORDER BY, etc. clauses). This recommendation is in line with the most powerful usage model for weblobs: *field_name*$wblob field should only be written into, but never read from; instead the *field_name*$wblob_info should be used to get the cloud storage provider URL so that it could be passed to the client for direct download. Note that the recommendation also implies that SELECT * FROM … statements should also be avoided when the table has weblob fields.

When MySQL executes the update part of the INSERT … ON DUPLICATE KEY UPDATE statement it may read and update all fields of the table, even those that are not needed for the update. If the table contains weblobs, the weblob content may get needlessly copied and exhibit performance problems due to considerations described above. If the record contains weblobs larger than the size limit the statement would fail.

When MySQL executes the ALTER TABLE statement, in many cases it creates a new table, copies all the data from the original table to the new table, and drops the original table. ClouSE detects this case and tries to avoid copying weblobs if the table has them. MySQL passes only partial information to the storage engine handler about the ALTER TABLE statement (namely the old and new structures without the relationship between them); based on that information ClouSE uses the following logic to handle weblob fields:

- weblob fields with the same name are considered the same and moved to the new table

- exactly one dropped weblob field and exactly one added weblob field is considered a rename and weblobs are moved from the old field to the new field
- exactly one dropped weblob field and exactly one relational field is considered a retype and weblobs are copied to the new field by value

Complex cases of ALTER TABLE statements may fail with an error if this logic cannot handle the weblob modifications, if that happens, the ALTER TABLE statement should be simplified.

4.3 Weblob Security Considerations

A weblob URL contains a predictable identifier and a 128-bit cryptographically random salt. A properly configured cloud storage location must not grant any access to public, which in conjunction with SSL enforce that it's impossible to discover valid weblob URLs. In a way, this could be considered an equivalent of protecting each weblob's content with a strong access key. So even though the weblob content is publicly readable, the effort to find a weblob is the same as or harder than to discover the access key for a secured location (i.e. virtually impossible). In fact, it's even more secure, because discovering one weblob's URL does not help to discover others, while discovering the access key gives access to all content.

With this security model the server application is in control of what weblobs are accessible to the client. When weblob content changes its URL changes as well, so the client does not get access to future content of the weblob unless the server lets it so. So even though the server does not serve weblobs and cannot make an access control decision at the time of weblob download, the level of access control is practically the same[6].

Note that weblob content is neither encrypted nor compressed by ClouSE: the purpose of weblobs is to be accessed directly by the client (e.g. a web browser) and the client wouldn't be able to properly interpret weblob content if the content was transformed by ClouSE. Instead weblob content is encrypted using the server-side encryption[7] capabilities of the cloud storage utility provider, which is transparent to the consumers. If client-side weblob encryption is desired it should be implemented on the application level so that the application server and the client (e.g. a web browser) can cooperate on how to properly decrypt weblob content.

4.4 The "Mighty Cloud" Design Pattern

The "mighty cloud" design pattern uses weblobs to scale out caching to the cloud storage utility provider.

[6] Once the client has a weblob URL is can access the weblob content and the server cannot revoke the access without changing the weblob, however, from the security perspective information disclosure cannot really be revoked even if the server could control access on weblob download: once the client has had the content, it has the content.

[7] http://docs.amazonwebservices.com/AmazonS3/latest/dev/UsingServerSideEncryption.html.

Most relational databases are designed to store data in a normalized form, such as in the third normal form or in the Boyce-Codd normal form, to minimize data redundancy and dependencies, eliminate modification anomalies, simplify data processing, etc. However, in most cases the data is exposed in some serialized representation: HTML, XML, JSON, etc. If the data is much more frequently read than updated, the application scalability could be increased if the data is delivered by the cloud storage utility provider directly to the client.

To illustrate the idea, let's consider a movie clip sharing web application. Suppose the web page that shows a movie clip has the following content: the movie clip itself, 50 latest comments, and 25 related movies. The page gets the data for the comments and related movies in the JSON format. The data could be represented in the following tables:

```
CREATE TABLE clips
(
   clip_id BIGINT,                                -- clip unique id
   name VARCHAR(256) NOT NULL,                    -- clip name
   tnail$wblob LONGBLOB, tnail$wblob_info BLOB, -- clip thumbnail (picture)
   movie$wblob LONGBLOB, movie$wblob_info BLOB, -- clip data (movie)
   json$wblob LONGBLOB, json$wblob_info BLOB,   -- cached JSON data
   PRIMARY KEY (clip_id)
)
ENGINE = CLOUSE;

CREATE TABLE clip_comments
(
   clip_id BIGINT,                                -- clip unique id
   comment_id INT,                                -- comment id
   time DATETIME NOT NULL,                        -- time of the comment
   content VARCHAR(1024) NOT NULL,                -- comment content
   PRIMARY KEY (clip_id, comment_id),
   KEY (time)
)
ENGINE = CLOUSE;

CREATE TABLE clip_related
(
   clip_id BIGINT,                                -- clip unique id
   related_clip_id BIGINT,                        -- related clip unique id
   rank INT NOT NULL,                             -- how closely related
   PRIMARY KEY (clip_id, related_clip_id),
   KEY (rank)
)
ENGINE = CLOUSE;
```

Without the "mighty cloud" design pattern, serving a movie clip webpage for a clip with id=12345 would involve getting data using the queries like the following:

```
SELECT name, movie$wblob_info FROM clips WHERE clip_id=12345;

SELECT content FROM clip_comments
  WHERE clip_id=12345 ORDER BY time DESC LIMIT 50;
```

```
SELECT c.name, c.tnail$wblob_info
  FROM clips c JOIN clip_related r ON c.clip_id=r.related_clip_id
  WHERE r.clip_id=12345 ORDER BY r.rank LIMIT 25;
```

and serializing the data into a JSON format to be embedded into the web page.

The "mighty cloud" design pattern helps to scale out most of the query processing and serialization to the cloud. Instead of running on each read, the abovementioned queries would need to run once after each update and serialize the result into the **clips.json$wblob** field[8] (in a transaction). Thus with the "mighty cloud" design pattern serving a movie clip page would be a matter of running a query like this:

```
SELECT json$wblob_info FROM clips WHERE clip_id=12345;
```

and embedding the resulting URL into the `<script>` tag of the webpage. The resulting webpage served from the application is going to be small and the data is going to be fetched by the web browser directly from the cloud storage provider.

So the "mighty cloud" design pattern helps to elegantly combine fully transactional ACID model for data processing and the cloud storage power for content delivery.

5 ClouSE Setup and Configuration

ClouSE can be downloaded from https://www.oblaksoft.com/downloads. The distribution contains README file with brief deployment instructions and my-clouse.cnf file with annotated ClouSE configuration examples that can be included into the [mysqld] section of the my.cnf file.

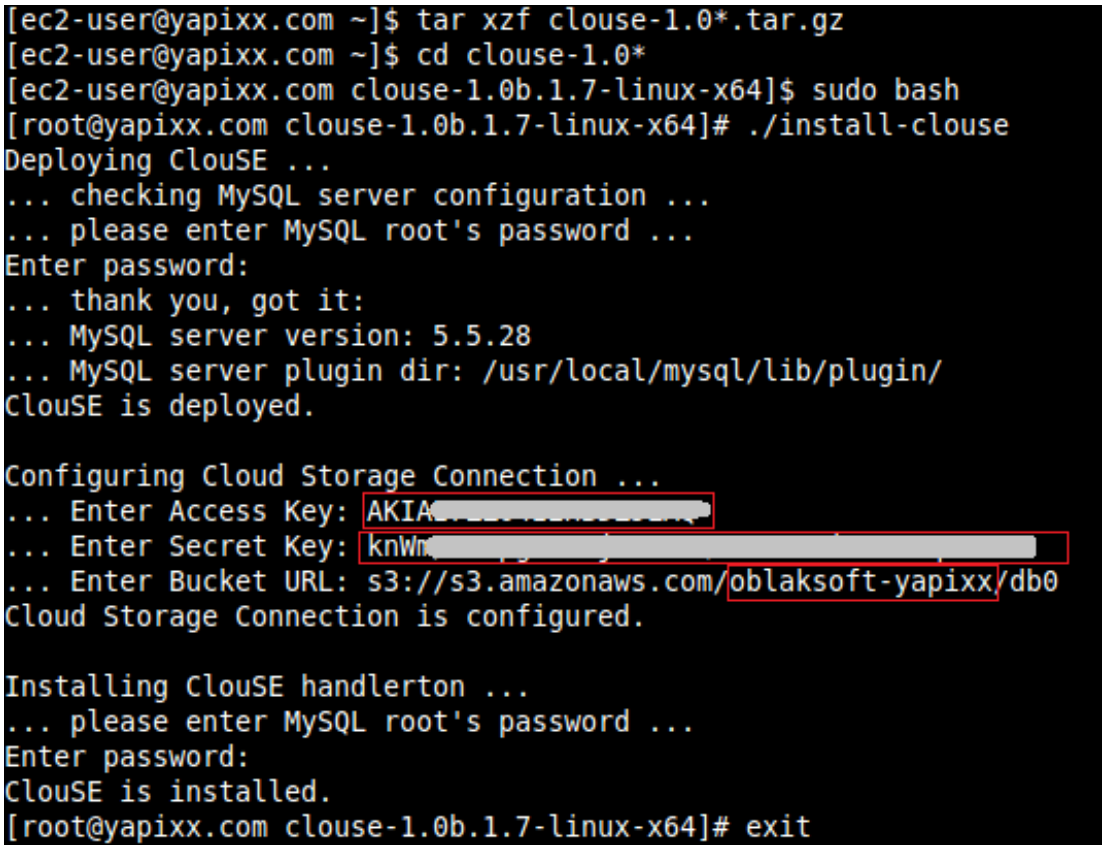
The distribution contains the install-clouse script that can be used to install ClouSE. The script needs to be run with root credentials and it prompts the MySQL's root password. Here is an example of running the script from the ClouSE distribution directory:

```
./install-clouse
```

The install-clouse script is going to deploy ClouSE binaries and configure connection to the cloud storage. The script is going to prompt for the Access Key, the Secret Key and the Bucket URL. For instructions on how to obtain the cloud connection information, please refer to the corresponding cloud utility provider configuration: Amazon S3 Configuration, Google Cloud Storage Configuration, Eucalyptus Walrus Configuration.

Here is a screenshot of a successful run of the install-clouse script that configures ClouSE to access Amazon S3 bucket oblaksoft-yapixx:

[8] Alternatively, on update the **clips.json$wblob** field can be set to NULL to indicate that the first reader would have to serialize the relational data into **clips.json$wblob** field.



To create a table in ClouSE the ENGINE=CLOUSE option can be used in the CREATE TABLE statement, for example:

```
CREATE TABLE t1 (id INT KEY, data VARCHAR(64) NOT NULL) ENGINE=CLOUSE;
```

To create all new tables in ClouSE by default the default-storage-engine configuration option can be used on the command line or in the [mysqld] section of the my.cnf file. Here is an example of setting the default-storage-engine configuration option in the my.cnf file:

```
default-storage-engine=ClouSE
```

To move an existing table to ClouSE the ENGINE=CLOUSE option can be used in the ALTER TABLE statement, for example:

```
ALTER TABLE t2 ENGINE=CLOUSE;
```

Note that if the table uses features that are not supported by ClouSE the alter operation will fail. See Using ClouSE Tables section for information about supported features. The ALTER TABLE statement copies all data stored in the table, so it may take a long time.

To move all existing tables to ClouSE the table names can be retrieved from the INFORMATION_SCHEMA.TABLES table and then the ALTER TABLE statement can be used for each table. The following example shows how to retrieve all eligible table names:

```
SELECT table_schema, table_name, engine
  FROM INFORMATION_SCHEMA.TABLES
  WHERE table_type='BASE TABLE'
    AND table_schema<>'INFORMATION_SCHEMA'
    AND table_schema<>'PERFORMANCE_SCHEMA'
    AND table_schema<>'mysql'
    AND engine<>'ClouSE';

+--------------+------------+--------+
| table_schema | table_name | engine |
+--------------+------------+--------+
| test         | t3         | InnoDB |
| test         | t4         | InnoDB |
+--------------+------------+--------+
```

Note that according to the MySQL documentation the MySQL system tables cannot use a different storage engine. For more information about MySQL restrictions on storage engine usage please refer to the MySQL documentation.

5.1 Cloud Data Configuration

The only mandatory configuration option for ClouSE is the clouse_cloud_data_url option that can be used to specify the cloud data storage location root; if it's not set ClouSE will fail to initialize. Each ClouSE instance must use a unique data storage location root.

The clouse_cloud_data_url option has the following format: *sheme*://*host*[:*port*]/*bucketName*[/*prefix*]. The *scheme* part corresponds to the cloud storage utility provider. Currently, the following schemes are supported: **s3**, **gs**, and **walrus** that correspond to Amazon S3, Google Cloud Storage and Eucalyptus Walrus. The *host* part defines the cloud storage service endpoint to connect to; private cloud storage utility providers may also support different ports, in which case an optional port part may need to be specified. The *bucketName* part defines the bucket that the data is going to be stored in; bucket is a container that may determine where the data is stored and how it's accessed. An optional *prefix* may be specified if it's desired to use only part of the bucket namespace for a ClouSE instance. For details and provider specifics please refer to the corresponding cloud utility provider configurations: Amazon S3 Configuration, Google Cloud Storage Configuration, Eucalyptus Walrus Configuration.

For security reasons it's strongly recommended that the cloud data storage is only accessible to ClouSE, so the clouse_cloud_auth_key option should specify the corresponding authentication key in the form of *AccessKeyId*:*SecretAccessKey*. The authentication key must allow read / write access to the cloud storage location.

To protect the data from accidental or malicious misuse the data can be encrypted. The clouse_cloud_data_encrypt_key option can be used to specify the encryption key data in the form *Algorithm:Passphrase*. aes256 is the only algorithm that is currently supported. The passphrase can be any phrase, case-sensitive, leading and trailing whitespace is trimmed, but

spaces between the words are significant. The actual encryption key used in the encryption algorithm is deterministically generated from the passphrase, so the passphrase doesn't need to be random, but it should be long enough to generate a good encryption key.

The clouse_cloud_proxy option can be used to specify a web proxy for accessing the Internet. If specified the proxy is used both for relational data storage and weblob storage.

The clouse_cloud_ssl_cert_file option can be used to specify the name of a file that contains a list of Root certificates in the PEM format. If the option is not specified a hardcoded list of Root certificates is used. You may need to set the option to provide an updated list of the Root certificates if the hardcoded list gets out of date. If the clouse_cloud_ssl_cert_file option is set to the literal 'none' value then the SSL verification is disabled (not recommended in production).

All options must be either set in the my.cnf file or specified on the command line of the MySQL server. In the my.cnf file the options should be specified in the [mysqld] section. Here is an example configuration in the my.cnf file:

```
clouse_cloud_data_url=s3://s3.amazonaws.com/mybucket/db
clouse_cloud_auth_key=MYACCESSKEYID:MySeCRetKeY
clouse_cloud_data_encrypt_key=aes256:2beer or not 2beer? 6beer.
clouse_cloud_proxy=myproxy:8080
```

For more information about MySQL server configuration please refer to the MySQL documentation.

Caution: if the data was encrypted and then the original value of the clouse_cloud_data_encrypt_key option is lost, the data is completely useless! **It is extremely important to have a strategy to recover the encryption key in the case of the loss of the configuration data**: the actual user data is safe with the cloud storage utility provider, but without the key it's as good as lost. In our opinion the best strategy is to pick a phrase that can be reliably recovered from personal memory or from an available source (e.g. a book). If that is not possible, the encryption key should be backed up.

Caution: once the encryption key is set it cannot be changed as the data that has been encrypted with the key cannot be read with a different key. It is recommended to set the encryption key at the very beginning (unless the data is supposed to stay unencrypted).

5.2 Local Transaction Log Configuration

ClouSE stores a copy of the latest transaction log on the local disk. The transaction log is represented by a set of *.xnl[9] files. By default the transaction log is stored in the MySQL data directory in files that have names starting with the clse_log (e.g. clse_log_hdr.xnl, clse_log_42.xnl), but it can be specified explicitly using the clouse_local_log_path option that

[9] xnl stands for XactioN Log (transaction log).

can be set in the my.cnf file or specified on the command line of the MySQL server. In the my.cnf file the options should be specified in the [mysqld] section. Here is an example configuration in the my.cnf file:

```
clouse_local_log_path=/data/clouse/mylog
```

With the example configuration the /data/clouse directory would contain files that have names starting with the mylog (e.g. mylog_hdr.xnl, mylog_42.xnl). If the /data/clouse path doesn't exist ClouSE will attempt to create it.

To get the best performance, the local transaction log should be placed to a dedicated physical disk. The only times when the transaction log is read from are transaction rollback and system restart, which are relatively rare events compared to normal processing when the transaction log only written to, thus from the performance perspective fast write access is most important.

The disk that the local transaction log resides on must have enough free disk space for the ClouSE to run. The transaction log is automatically truncated so it should reach a stable disk footprint that generally depends on the machine configuration (does not depend on the length of transactions or database size). A couple GB of disk space should suffice for most configurations.

Caution: ClouSE is a transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data. **However, it cannot do so** if the local storage stack (i.e. all components from the OS system call interface to the physical durable storage) delays or reorders writes across the fsync() system call! Data loss (including corruption of the whole database) may occur if the fsync() system call returns before all data that was written before it reaches the physical durable storage.

Caution: It's not recommended to store the local transaction log files on NFS volumes. Potential problems vary according to OS and version of NFS, and include such issues as lack of protection from conflicting writes, and limitations on maximum file sizes.

5.3 Local Data Cache Configuration

ClouSE stores a copy of recently accessed relational data on the local disk. The on-disk cache data persists across server restarts and machine reboots, it is crash-tolerant and self-healing. The on-disk cache drastically reduces the amount of data that needs to be retrieved from the cloud storage.

By default, the on-disk cache data is located in the MySQL data directory, in files that start with the clse_cache prefix. The cache is represented by a set of *.dat files and *.xnl files.

The on-disk cache location can be configured using the clouse_local_data_path option that can be set in the my.cnf file or specified on the command line of the MySQL server. In the my.cnf

file the options should be specified in the [mysqld] section. Here is an example configuration in the my.cnf file:

```
clouse_local_data_path=/data/clouse/mydata
```

With the example configuration the /data/clouse directory would contain files that have names starting with the mydata_cache (e.g. mydata_cache_data_0.dat, mydata_cache_log_42.xnl). If the /data/clouse path doesn't exist ClouSE will attempt to create it.

The on-disk cache size is capped by the maximum disk cache size limit. By default, the maximum disk cache size is automatically calculated based on the available disk space, but it can be configured using the clouse_local_data_size option. Here is an example of setting the maximum on-disk cache size to 8 GB:

```
clouse_local_data_size=8GB
```

Setting clouse_local_data_size to 0 disables the on-disk cache.

5.4 History Retention for Point-in-time Recovery

ClouSE supports built-in point-in-time recovery. It automatically retains history, such that it's possible to restore data to any point in time within the retention period with a second granularity. For more information on point-in-time recovery see Point-in-time Recovery.

The default history retention period is 24 hours. This should be enough to discover a disaster (e.g. admin accidentally dropping the database) and restore the data to a point before the disaster happened.

The history retention period can be configured to a value between 0 and 744 hours (31 days) using the clouse_retention_period configuration option. The value is in hours. Here is an example of setting the history retention period to one week (168 hours):

```
clouse_retention_period=168
```

5.5 System Configuration

ClouSE uses a buffer pool cache to cache table and index data. All relational data access goes thru the buffer pool which is essential for good performance.

The default (and maximum) buffer pool size is set to the 75% of the physical memory available on the machine. This assumes that ClouSE is the primary memory consumer on the machine which is a recommended configuration when high performance is required.

In some cases, however, it may be beneficial to limit the amount of memory ClouSE uses for the buffer pool. The can be accomplished using the clouse_buffer_pool_size configuration option. The buffer pool size cannot be less than 256 MB or greater than 75% of the physical memory. Here is an example of specifying the buffer pool size to be limited to 512 MB:

```
clouse_buffer_pool_size=512M
```

ClouSE imposes a size limit on downloading individual weblob content via select statements. A select statement that selects a *field_name*$wblob field would fail if it fetches a record that contains a weblob larger than the limit. The default limit is 16 MB, but can be configured to a value between 1KB and 1GB using the clouse_max_weblob_download_size configuration option.

Here is an example of setting the weblob download size limit to 64 MB:

```
clouse_max_weblob_download_size=64M
```

Note that the limit does not affect weblob content that is downloaded directly from the cloud storage using the weblob URL (e.g. by the web browser).

5.6 Moving ClouSE to Another Machine

To move ClouSE to another machine move all local transaction log files to the new location and set the ClouSE configuration options to specify the corresponding cloud and local data locations.

In the most typical case the entire MySQL directory is moved as well (and in the case of the default local transaction log location that's all that needs to be moved). For more information about moving MySQL databases to another machine please refer to the MySQL documentation.

If for some reason the MySQL directory cannot be moved (e.g. because it's lost), the data can be recovered from the local transaction log on the other machine using the disaster recovery procedures (see Disaster Recovery from Local Transaction Log).

Caution: Only one ClouSE at a time should access the cloud storage location! **The data may get corrupted beyond repair** if two or more ClouSE instances access the same cloud storage location. When moving ClouSE to another machine, please make sure that the original ClouSE instance is completely shut down before starting the new ClouSE instance.

5.7 Upgrading ClouSE

The update-clouse script can be used to upgrade ClouSE. The script needs to be run with root credentials. Here is an example of running the script from the ClouSE distribution directory:

```
./update-clouse
```

Newer ClouSE versions transparently upgrade the data to newer formats[10]. To support transparent upgrades and prevent downgrades, the data is stamped with the data format version.

[10] Transparent data upgrades are not supported for data created by ClouSE Beta (versions 1.0b.1.x). The update-clouse script offers automatic data migration for upgrades from ClouSE Beta.

The version format is the following: <major>.<minor>.<data>.<patch>. The first two numbers are the feature set version; it can have a "b" suffix (stands for Beta) or an "rc" suffix (stands for Release Candidate) or no suffix. The third number is the data format version; it's incremented every time the data format changes. The patch number is updated to make the version unique for each released build.

If any of the first three version numbers (major, minor, or data) are different, the new version of ClouSE is going to change data format and will make downgrades impossible.

To retrieve the ClouSE version the following command could be used:

```
strings clouse.so | grep @VERSION@
```

The result is going to be similar to the following:

```
@VERSION@: 1.0b.5.42
```

To see the version from MySQL the following statement could be used:

```
SHOW VARIABLES LIKE 'clouse_version';

+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| clouse_version | 1.0b.5.42 |
+-----------------+-----------+
```

6 ClouSE Licensing

When ClouSE is installed and configured it is automatically licensed for free trial. The trial license gives access to the full feature set and is valid for 60 days. A production license must be obtained at http://www.oblaksoft.com/licensing/ to continue using ClouSE beyond the trial term. Should you have any questions about licensing, feel free to contact info@oblaksoft.com.

6.1 Viewing ClouSE Licensing Information

To get the current ClouSE licensing information, the following MySQL statement can be used:

```
SHOW STATUS LIKE 'clouse_license_%';
```

The result is going to look like the following:

```
+--------------------+----------------------------+
| Variable_name      | Value                      |
+--------------------+----------------------------+
| clouse_license_err |                            |
| clouse_license_exp | 2014-08-31                 |
| clouse_license_sku | TRY                        |
| clouse_license_url | s3://db.yapixx.com/shard0  |
+--------------------+----------------------------+
```

The clouse_license_exp status variable shows the license expiration date, in this example it's August 31st, 2014.

The clouse_license_sku status variable shows the licensed feature set (edition). It can have the following values:

- TRY – Trial license
- BAS – Basic license
- STD – Standard license

Please see http://www.oblaksoft.com/documentation/clouse-features/ for more information about ClouSE editions.

The clouse_license_url status variable shows the licensed storage location. That's the value that should be used in the Bucket URL edit box at http://www.oblaksoft.com/licensing/.

Note that clouse_license_url may be different from the clouse_cloud_data_url configuration option. Make sure you use the clouse_license_url status variable's value when ordering your license.

6.2 Installing a New ClouSE License

The new ClouSE license is going to be sent in email as the license.sql file attachment. The content of the license.sql file is going to look like this:

```
SET SESSION clouse_license_url='s3://db.yapixx.com/shard0';
SET SESSION clouse_license_exp='2015-07-01';
SET SESSION clouse_license_sku='BAS';
SET SESSION clouse_license_key='XXXXXXX.................';
```

The license can be installed by executing the license.sql script using the mysql command line utility like this:

```
mysql < license.sql
```

Alternatively, the license.sql script can be executed by any tool that can execute MySQL scripts, e.g. phpMyAdmin, Webmin, etc. Users of the WordPress to Cloud solution can also install the license from the WordPress to Cloud Settings page.

7 Public Cloud Configuration for ClouSE

7.1 Amazon S3 Configuration

To get started with Amazon S3 please refer to http://docs.amazonwebservices.com/AmazonS3/latest/gsg/. The guide introduces the concepts

of AWS account and buckets, and shows how to sign up for an account and create a bucket. At the time of this writing signing up for an AWS account was free, and there was a free usage tier that could be used to try S3 free of charge as described at http://aws.amazon.com/s3/pricing/. For more information about Amazon S3 please refer to http://aws.amazon.com/s3/.

To use Amazon S3 as the cloud storage utility provider ClouSE needs three pieces of information:

- Cloud storage location root
- Access Key ID
- Secret Access Key

The Amazon S3 storage location root has the following format: s3://*AmazonS3RegionEndpoint*/*bucketName* or s3://*AmazonS3RegionEndpoint*/*bucketName*/*prefix*

The *AmazonS3RegionEndpoint* is a region-specific REST API endpoint, for example s3.amazonaws.com or s3-eu-west-1.amazonaws.com. For a list of the REST API endpoints and their corresponding regions please refer to http://docs.amazonwebservices.com/general/latest/gr/rande.html#s3_region.

The *bucketName* is the name of the bucket that is going to contain the ClouSE data and an optional *prefix* can be used if it's desired to use only part of the bucket namespace for ClouSE. All ClouSE data is going to be contained within the specified prefix. The AWS management console can be used to view all buckets for an AWS account.

Here is an example of a cloud storage location root for an Amazon S3 in the US Standard region that uses the db.yapixx.com bucket and shard0 prefix:

```
clouse_cloud_data_url=s3://s3.amazonaws.com/db.yapixx.com/shard0
```

If the db.yapixx.com bucket was created in the US West (Northern California) region, the cloud storage location root would be specified as the following:

```
clouse_cloud_data_url=s3://s3-us-west-1.amazonaws.com/db.yapixx.com/shard0
```

Note that the bucket must be created before ClouSE can use it, but there is no need to manually create any objects with the specified prefix.

By default ClouSE uses SSL to access Amazon S3; this guarantees data security over the wire. If the channel between ClouSE and Amazon S3 is secured by other means, for example if ClouSE runs on Amazon EC2, using SSL may not be necessary. To use plain HTTP for ClouSE / Amazon S3 communication the s3:http protocol qualifier can be used, for example:

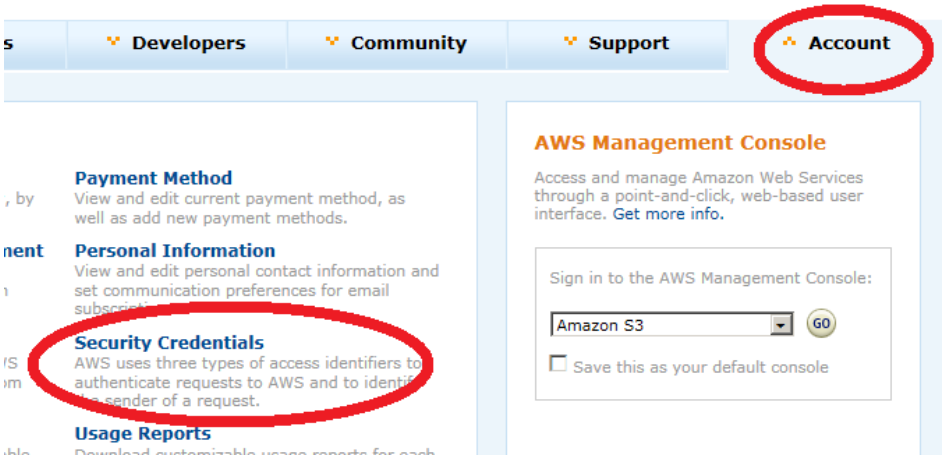
```
clouse_cloud_data_url=s3:http://s3.amazonaws.com/db.yapixx.com/shard0
```

It is possible to also use the s3:https protocol qualifier, which is a synonym for s3 as HTTPS is used by default. The protocol configuration can be changed at any time.

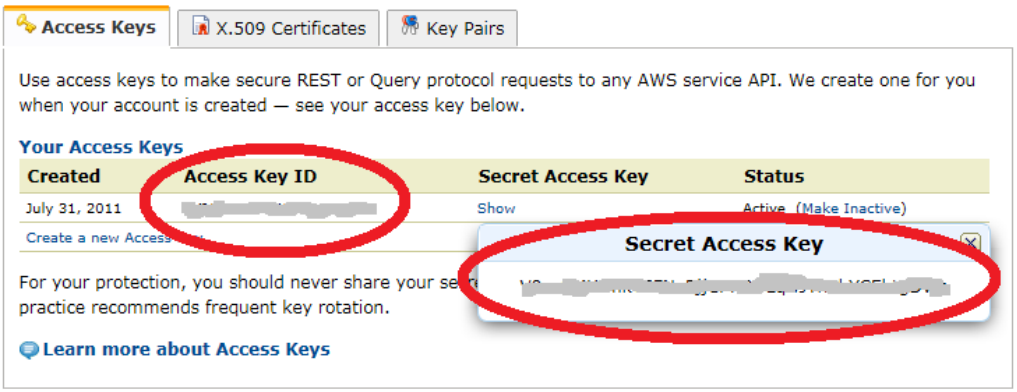
Caution: Even if ClouSE uses data encryption and data confidentiality is not a concern, for data integrity it is still necessary to use SSL when the channel between ClouSE and Amazon S3 may not be secure to guarantee that ClouSE is indeed connected to Amazon S3! Even though the attacker wouldn't be able to get to the user data, they still could corrupt the data going in either direction which **may lead to severe corruption and total data loss**.

To quickly get started with S3 the AWS account access credentials can be used, however in production it's strongly recommended to create a dedicated user for ClouSE access and restrict all other access to the bucket as described later in this section.

To get the AWS account access credentials, please sign up to your account and go to the security credentials page:



and locate the access keys for your AWS account:

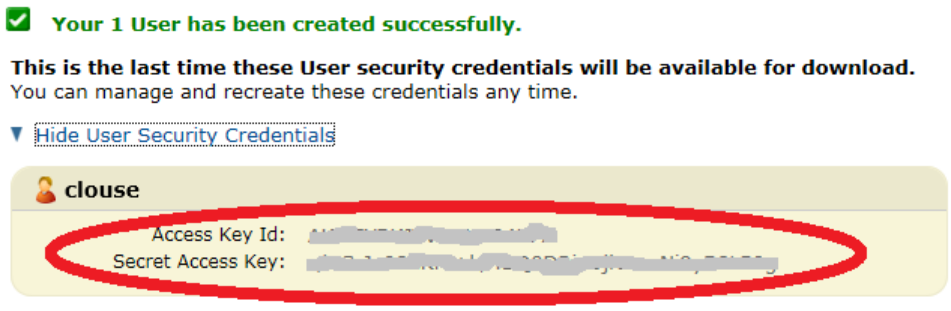


That's all the information needed from Amazon for ClouSE to work. Here is an example of specifying the Access Key ID and the Secret Access Key in the ClouSE configuration options:

```
clouse_cloud_auth_key=MYACCESSKEYID:MySeCRetKeY
```

Caution: Only ClouSE should modify the data in the cloud storage location! Direct modification of any data in the cloud storage location **may lead to severe corruption and total data loss**. To minimize possibility of accidental direct data modification it is strongly recommended to create a dedicated user for ClouSE access and restrict all other access to the bucket as described below.

The AWS management console https://console.aws.amazon.com/iam/home can be used to create a new user named "clouse". When the new user is created, it's possible to view the user's Access Key ID and Secret Access Key:



That's the values to use in the clouse_cloud_auth_key option.

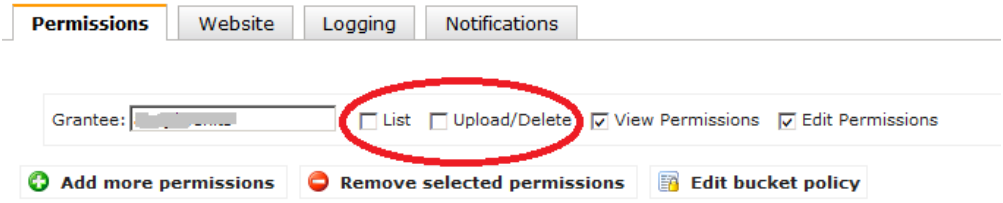
In order to be used for ClouSE, the clouse user needs to get full S3 access to the bucket. When the user is selected in the IAM console, the permissions tab can be used to attach a user policy. Here is an example of a user policy that grants full S3 access to the db.yapixx.com bucket:

```
{
    "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::db.yapixx.com",
        "arn:aws:s3:::db.yapixx.com/*"
        ]
      }
    ]
}
```

Note that the bucket name is used twice: once to allow operations on the bucket itself and once to allow operations on the objects in the bucket.

Once testing is done that ClouSE can successfully access the data with the clouse user's credentials, the bucket owner's access should be restricted in the S3 management console.

When a bucket is selected in the S3 management console, its properties have a permissions tab that can be used to manage the permissions. It is recommended to uncheck both List and Upload/Delete permissions for the bucket owner:



7.2 Google Cloud Storage Configuration

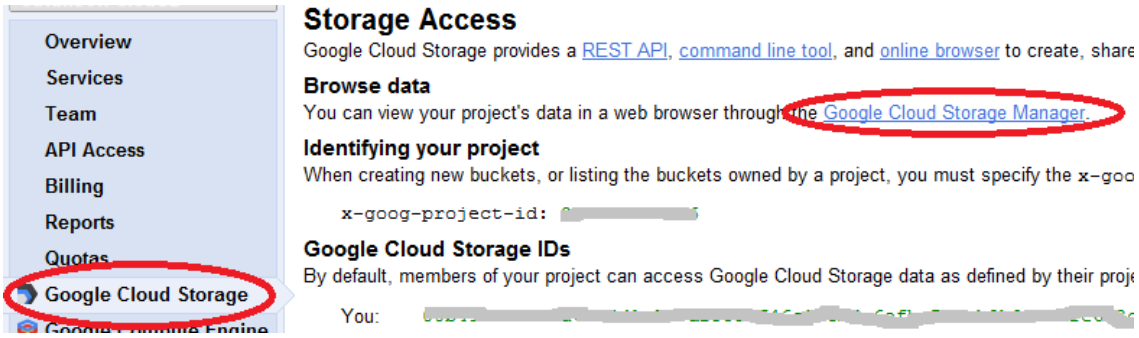
To get started with Google Cloud Storage please refer to http://developers.google.com/storage/docs/signup. The page contains instructions on how to get started with Google Cloud Storage.

To use Google Cloud Storage as the cloud storage utility provider ClouSE needs three pieces of information:

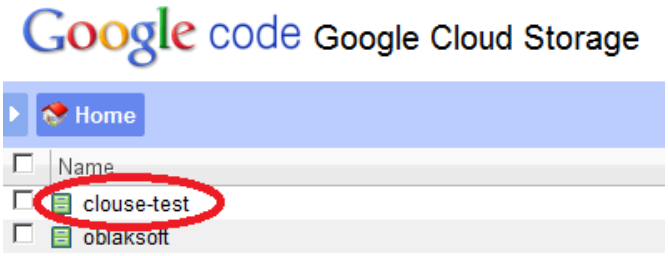
- Cloud storage location root
- Access Key ID
- Secret Access Key

The Google Cloud Storage location root has the following format:
gs://commondatastorage.googleapis.com/*bucketName* or
gs://commondatastorage.googleapis.com/*bucketName/prefix*

To get the name of an existing bucket or create a bucket, please go to https://code.google.com/apis/console, select Google Cloud Storage channel, and go to the Google Cloud Storage Manager.



The Cloud Storage Manager presents a list of existing buckets and can be used to create a new bucket. For example:



Here is an example of a cloud storage location root for Google Cloud Storage that uses the clouse-test bucket and yapixx0 prefix:

```
clouse_cloud_data_url=gs://commondatastorage.googleapis.com/clouse-test/yapixx0
```

Note that the bucket must be created before ClouSE can use it, but there is no need to manually create any objects with the specified prefix.

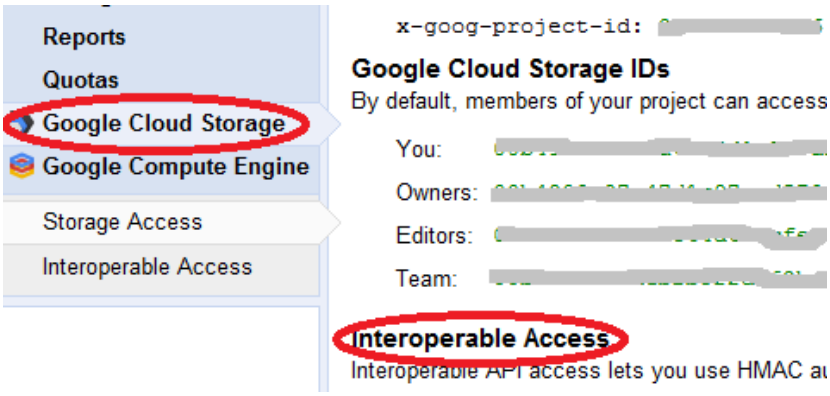
By default ClouSE uses SSL to access Google Cloud Storage; this guarantees data security over the wire. If the channel between ClouSE and Google Cloud Storage is secured by other means, for example if ClouSE runs on Google Compute Engine, using SSL may not be necessary. To use plain HTTP for ClouSE / Google Cloud Storage communication the gs:http protocol qualifier can be used, for example:

```
clouse_cloud_data_url=gs:http://commondatastorage.googleapis.com/clouse-
test/yapixx0
```

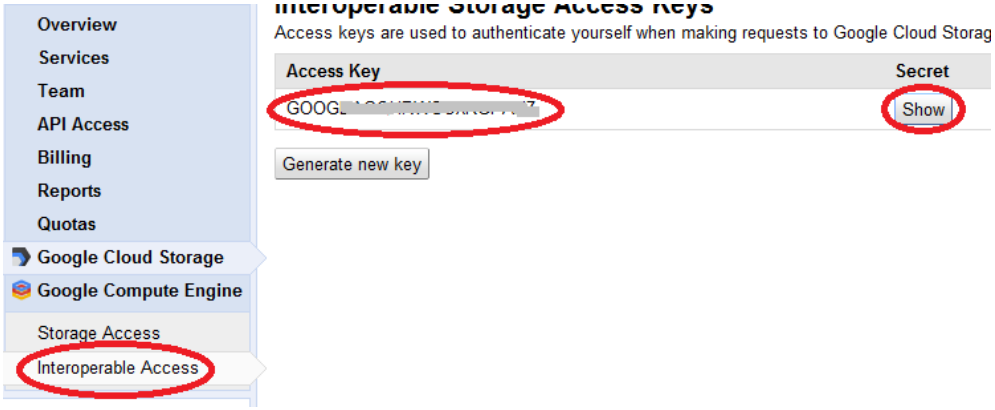
It is possible to also use the gs:https protocol qualifier, which is a synonym for gs as HTTPS is used by default. The protocol configuration can be changed at any time.

Caution: Even if ClouSE uses data encryption and data confidentiality is not a concern, for data integrity it is still necessary to use SSL when the channel between ClouSE and Google Cloud Storage may not be secure to guarantee that ClouSE is indeed connected to Google Cloud Storage! Even though the attacker wouldn't be able to get to the user data, they still could corrupt the data going in either direction which **may lead to severe corruption and total data loss**.

To authenticate with Google Cloud Storage, ClouSE needs Interoperable Storage Access enabled. To enable Interoperable Access please go to https://code.google.com/apis/console, select Google Cloud Storage channel, and turn on Interoperable Storage Access:



Once the Interoperable Access has been enabled, select the Interoperable Access channel and get the Access and Secret keys. The Access key is visible immediately; click the Show button to get the Secret key.



The Access and Secret keys should be specified in the clouse_cloud_auth_key configuration option, separated by a colon. Here is an example of specifying the Access and Secret keys in the ClouSE configuration options:

```
clouse_cloud_auth_key=MYACCESSKEYID:MySeCRetKeY
```

Caution: Only ClouSE should modify the data in the cloud storage location! Direct modification of any data in the cloud storage location **may lead to severe corruption and total data loss**.

8 Private Cloud Configuration for ClouSE

8.1 Eucalyptus Walrus Configuration

ClouSE can be used with Eucalyptus cloud storage utility provider – Walrus http://www.eucalyptus.com. The Eucalyptus Walrus can be used as a private cloud storage utility provider or as a pre-production testing environment prior to deploying to Amazon S3.

To install Eucalyptus Walrus on a Debian / Ubuntu Linux machine the following commands can be run from a shell prompt:

```
sudo apt-get install eucalyptus-cloud
sudo apt-get install eucalyptus-walrus
```

For more information on the sudo and apt-get commands please refer to the Linux documentation. For other Linux distributions please refer to the Eucalyptus and/or corresponding Linux distribution's documentation.

If the commands complete successfully the Eucalyptus should be up and running. However, in order to use Walrus it needs to be registered as described below.

Eucalyptus adds a user named eucalyptus, but the user doesn't have a password. The password for this user is needed for the Walrus registration and can be changed with the following command:

```
sudo passwd eucalyptus
```

To register Walrus the **sudo euca_conf --register-walrus *IPaddr*** command can be used where *IPaddr* is an external IP address of the machine (for some reason specifying a loopback interface doesn't work). To find an external IP address the **ifconfig** command can be used, for example:

```
ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:83:cc:36
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe83:cc36/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

...
```

In this example the command lists the IP address 10.0.2.15. The Walrus registration command would look like the following:

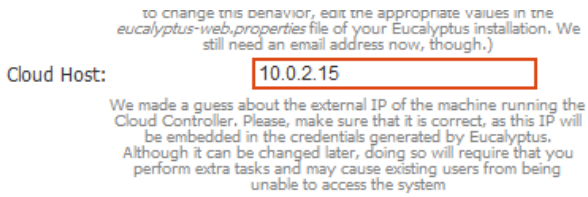
```
sudo euca_conf --register-walrus 10.0.2.15
```

Note that in the case when the IP address is allocated via DHCP, the registration may need to be amended to reflect the correct IP address. For that reason it may beneficial to use a static IP address. To get more information about Linux networking configuration please refer to the Linux documentation.

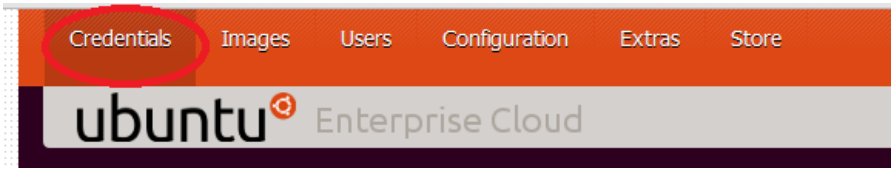
To get the security credentials for ClouSE to work with the Eucalyptus admin console can be used. The Eucalyptus admin console can be accessed via a Web Browser at https://*hostName*:8443 where hostName is the name of the host running Eucalyptus. As Eucalyptus uses a self-signed certificate the Web Browser may not be able to validate the certificate and show a warning, so the option to continue browsing must be chosen in that case.

The first time login uses the user name “admin” and the password “admin”. The password must be changed at the first login.

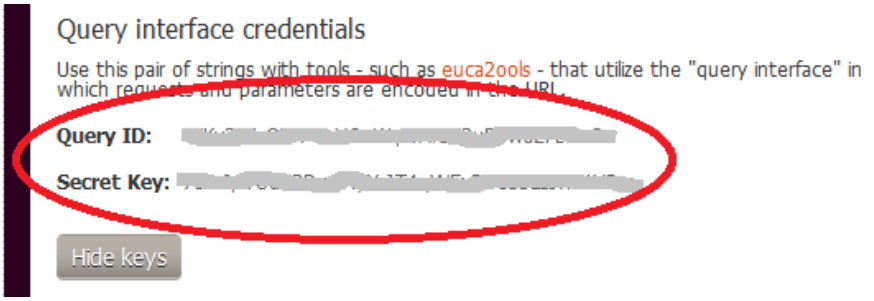
During the first login, Eucalyptus will ask to confirm Cloud Host’s IP address. Please make sure that it matches the one reported by the **ifconfig** command. For example:



To get the security credentials please go to the credentials tab in the admin console:



and locate the Query Interface Credentials:



The Query ID and the Secret Key in Eucalyptus correspond to the Access Key ID and Secret Access Key in Amazon AWS.

Caution using the admin credentials is ok in a test environment, however for a production environment it is strongly recommended to create a dedicated user for ClouSE access.

The **wscmd** utility can be used to create a bucket in the Walrus service. Here is an example command that creates a database bucket in a Walrus service that runs on the mybox host with the MYQUERYID Query ID and MySeCRetKeY Secret Key:

```
wscmd -i MYQUERYID -s MySeCRetKeY -H mybox -P 8773 -U -W -a create -n database
```

For more information about wscmd run the **wscmd -?** command. Other tools that can be used to operate and configure the Eucalyptus Walrus service are referenced at http://open.eucalyptus.com/wiki/s3-compatible-tools.

Once the bucket is created ClouSE can be configured to use the bucket. The Query ID and the Secret Key in Eucalyptus correspond to the Access Key ID and Secret Access Key in Amazon AWS. Here is an example of specifying the Query ID and the Secret Key in the ClouSE configuration options:

```
clouse_cloud_auth_key=MYQUERYID:MySeCRetKeY
```

The Eucalyptus Walrus location root has the following format: walrus://*hostname*:8773/*bucketName* or walrus://*hostname*:8773/*bucketName/prefix* where *hostName* is the host name of the machine that run the Walrus services, *bucketName* is the name of the bucket that is going to contain the ClouSE data and an optional *prefix* can be used if it's desired to use only part of the bucket namespace for ClouSE. All ClouSE data is going to be contained within the specified prefix. Here is an example of a cloud storage location root for an Eucalyptus Walrus that uses the database bucket and shard0 prefix:

```
clouse_cloud_data_url= walrus://mybox:8773/database/shard0
```

Note that the bucket must be created before ClouSE can use it, but there is no need to manually create any objects with the specified prefix.

Caution: ClouSE uses plain HTTP to communicate with Eucalyptus Walrus. Even if ClouSE uses data encryption and data confidentiality is not a concern, the channel between ClouSE and Eucalyptus Walrus must be secure enough to guarantee that ClouSE is indeed connected to Eucalyptus Walrus! Even though the attacker wouldn't be able to get to the user data, they still could corrupt the data going in either direction which **may lead to severe corruption and total data loss**.

9 Using ClouSE with Cloud Compute

9.1 Using ClouSE with Amazon EC2

To get started with Amazon EC2 please refer to http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/concepts.html. The guide introduces the concepts of AMI, instance, ephemeral storage (a.k.a. instance store) and EBS.

It is strongly recommended to use Elastic Block Storage (EBS) for the ClouSE local transaction log copy, as it's more reliable than the ephemeral storage. The transaction log is automatically truncated so it should reach a stable disk footprint that generally depends on the machine configuration (does not depend on the length of transactions or database size). A couple GB of disk space should suffice for most configurations.

It is also recommended to keep MySQL configuration and MySQL metadata on EBS: even though MySQL metadata for ClouSE can be easily recovered from the local transaction log copy, it's

easier to avoid it. The size of MySQL metadata depends on the number of tables but it is typically tiny compared to the user data.

A very simple and straightforward way to satisfy this recommendation is to use EBS-backed instances with default MySQL configuration (i.e. MySQL server and its data are stored on the root partition). For more information about EBS-backed AMIs and instances please refer to http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/Concepts_BootFromEBS.html.

Besides these specifics, ClouSE configuration on Amazon EC2 is not much different from any other environment.

As a convenient starting point OblakSoft provides an EBS-backed AMI with Amazon Linux, MySQL server community edition and ClouSE preinstalled. For more information about OblakSoft AMI please go to https://www.oblaksoft.com/downloads.

When launching an instance using OblakSoft AMI please make sure that the security group[11] that is used for the instance includes SSH access to the instance. SSH access is required to finish ClouSE configuration to point it to the appropriate Amazon S3 storage location.

For more information about launching instances please refer to http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/LaunchingAndUsingInstances.html.

After the instance is up and running, you need to connect to the instance to finish ClouSE configuration. This can be done using the **ssh** command or PuTTY software. For more information about connecting to instances please refer to http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/AccessingInstances.html.

If ClouSE is not configured (which is always the case on the first connect), the following information is printed:

```
****************************************
*** ClouSE is currently NOT configured ***

To enable ClouSE please set the clouse_cloud_data_url and clouse_cloud_auth_key
configuration options in /etc/my.cnf and restart MySQL using:
sudo /etc/init.d/mysql restart
For more information please see /etc/my.cnf, README and clouse.pdf.
```

To modify /etc/my.cnf any text editor can be used, for example the following command shows how to use the nano text editor:

```
sudo nano /etc/my.cnf
```

[11] The security group in this context is effectively firewall settings that define which network traffic is allowed.

After the configuration is modified, MySQL server should be restarted using the following command:

```
sudo /etc/init.d/mysql restart
```

To verify that the ClouSE started successfully the following command can be used:

```
echo 'SHOW ENGINE CLOUSE STATUS;' | mysql -u root
```

In the case of success, the result is going to look like the following:

```
Type   Name   Status
ClouSE        Online
```

If ClouSE failed to start, looking into the MySQL error log may help to find the problem. The following command can be used to list the tail of the MySQL error log:

```
sudo tail /usr/local/mysql/data/mysqld.err
```

For more information about troubleshooting please refer to Troubleshooting Configuration and Connection Issues. If a solution still cannot be found, please refer to How to File Bug Reports.

Once ClouSE is configured to access the appropriate Amazon S3 storage, the instance is ready to provide cloud-based relational data access. The default storage engine is set to ClouSE, so tables are created in ClouSE unless a different storage engine is explicitly specified. The MySQL root does not have a password and network access is disabled.

Caution: if MySQL configuration is changed to enable network access, the MySQL **root password must be changed**! In any case it is a best practice to change the MySQL root password as soon as possible.

The instance can be used as a standalone database server (e.g. as a data tier in a multi-tier web application) or can be configured further by installing additional software. For example, one could install Apache, PHP and WordPress to create a cloud-based multi-functional web site. After the additional configurations are made, it is also possible to create a new AMI based on the new configuration. For more information about creating Amazon EBS-backed AMIs please refer to http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/creating-an-ami-ebs.html.

When creating AMI from a configured EC2 instance, care should be taken to make sure that instances launched from the new AMI do not point to the same Amazon S3 location. The AMI that is provided by OblakSoft has a script that detects when an instance is cloned and cleans up ClouSE configuration, so AMIs that are derived from the OblakSoft AMI require no special action. If you decide to build your own AMI from scratch, please refer to the /etc/rc.d/init.d/clouse-cleanup script in the OblakSoft AMI for the information on how to properly clean up ClouSE configuration on clone.

10 ClouSE Diagnostics and Troubleshooting

10.1 Tracing Support

To capture what ClouSE sends to or receives from the cloud storage utility provider HTTP request tracing can be used. Tracing can be configured via the clouse_trace_filter global variable which takes a string in the following format: *category*[; *category* …][;destination:*fileName*] where the *category* is one of http or http_verbose (the latter dumps more data) and the *fileName* is the file name of the trace file (the /tmp/clouse.trace file is used if no destination is specified). Setting the variable to an empty value disables tracing.

The clouse_trace_filter configuration option can be also set in the [mysqld] section of the my.cnf file. This can be used to enable tracing during ClouSE plugin initialization.

Here are some examples of setting the clouse_trace_filter configuration option in the my.cnf file:

```
# enable basic http tracing (headers only) to default file (/tmp/clouse.trace)
clouse_trace_filter=http

# enable verbose http tracing and log it to a custom location
clouse_trace_filter=http_verbose;destination:/tmp/mytrace.trace
```

Here are some examples of configuring tracing from MySQL:

```
-- enable verbose http tracing into a given file
SET GLOBAL clouse_trace_filter='http_verbose;destination:/tmp/mytrace.trace';

-- show the current trace filter value
SHOW VARIABLES LIKE 'clouse_trace_filter';

-- disable all tracing
SET GLOBAL clouse_trace_filter='';
```

Note: tracing is buffered so the traced data may not immediately appear in the log. To flush the log re-set the clouse_trace_filter variable to the same value.

10.2 Troubleshooting Configuration and Connection Issues

If installation of the ClouSE plugin fails or the plugin fails to initialize or goes offline when MySQL starts the MySQL error log often contains data that can help to identify the root cause of the problem. For more information about MySQL error log please refer to the MySQL documentation.

The following table can be useful to find a possible solution from error messages can be could be found in the MySQL error log.

Error message	**Problem and/or solution**
[ERROR] /usr/local/mysql/bin/mysqld: unknown variable	The ClouSE plugin is not installed and

'clouse_cloud_data_url' Or unknown variable 'clouse_local_log_path'	MySQL does not recognize the clouse_XXX variables in the my.cnf file. Remove (or comment out) clouse_XXX variables from the my.cnf first, restart mysql, add the variables back and install ClouSE plugin. When mysql installs a plugin, it automatically re-loads my.cnf and passes plugin-specific variables to the corresponding plugin.
ClouSE: lockfile ./clse_log_hdr.xnl: Resource temporarily unavailable.	Another instance of MySQL server is running with the same my.cnf settings.
ClouSE: The cloud data URL must be specified in the clouse_cloud_data_url variable.	The clouse_cloud_data_url variable is not specified in the [mysqld] section of the my.cnf file.
ClouSE: Invalid storage class 'https' in 'https://s3.amazonaws.com/mybucket/prefix'.	The clouse_cloud_data_url value has invalid format. Make sure that value starts with the correct scheme (s3, gs, etc.) prefix.
ClouSE: Bucket name is missing in cloud storage URL s3.amazonaws.com/	The clouse_cloud_data_url value has invalid format.
ClouSE: The 'listObjects' operation for 'mybucket' failed. 404 Not Found: http://wrong-host/mybucket/ (Code=NotFound, RequestId=)	Invalid host name is specified in the clouse_cloud_data_url variable or the host is currently unreachable.
ClouSE: The 'listObjects' operation for 'mybucket2' failed. The specified bucket does not exist (Code=NoSuchBucket, RequestId=1234)	Invalid bucket name is specified in the clouse_cloud_data_url variable.
ClouSE: The 'listObjects' operation for 'mybucket-in-west-1-region' failed. The bucket you are attempting to access must be addressed using the specified endpoint. Please send all future requests to this endpoint. (Code=PermanentRedirect, RequestId=1234)	Invalid end point is specified in the clouse_cloud_data_url variable. Use region-specific endpoint name. For a list of region specific endpoints please refer to http://docs.amazonwebservices.com/general/latest/gr/rande.html#s3_region.
ClouSE: The 'listObjects' operation for 'mybucket' failed. Recv failure: Connection was reset. Or Couldn't connect to s3.amazonaws.com:999 Or routines:SSL23_GET_SERVER_HELLO:unknown protocol.	Invalid port number is specified in the clouse_cloud_data_url variable or the service is currently unavailable.
ClouSE: The 'listObjects' operation for 'mybucket' failed.	Invalid host name is specified in the

Could not resolve host: 127.0.0.10; Host not found.	clouse_cloud_data_url variable or the host is currently unreachable.
ClouSE: The 'put' operation for prefix/data/1669CA300D2924/2A/1669CA300D2924/00000000000.dat failed. Access Denied (Code=AccessDenied, RequestId=A754895B91DAEB2C)	The user identified by the AccessKeyId specified in the clouse_cloud_auth_key variable (the first part of the value before the ':') doesn't have full access to the S3 bucket specified in the clouse_cloud_data_url variable. Check both variables and / or S3 configuration.
ClouSE: The 'listObjects' operation for 'mybucket' failed. The AWS Access Key Id you provided does not exist in our records. (Code=InvalidAccessKeyId, RequestId=1234)	Invalid AccessKeyId is specified in the clouse_cloud_data_url variable. Verify and change the first part of the value before ':' as needed.
ClouSE: The 'listObjects' operation for 'mybucket' failed. The request signature we calculated does not match the signature you provided. Check your key and signing method. (Code=SignatureDoesNotMatch, RequestId=1234)	Invalid SecretAccessKey is specified in the clouse_cloud_data_url variable. Verify and change the second part of the value after ':' as needed. Make sure that you use path-style bucket name, i.e. s3.amazonaws.com/***bucket*** not *bucket*.s3.amazonaws.com.
ClouSE: The 'listObjects' operation for 'mybucket' failed. Could not resolve proxy: 127.0.0.300; Host not found.	Invalid proxy host name is specified in the clouse_cloud_proxy variable or the proxy host is currently unreachable.
ClouSE: The 'listObjects' operation for 'mybucket' failed. Couldn't connect to s3.amazonaws.com:10.	Invalid proxy port number is specified in the clouse_cloud_proxy variable or the proxy service is currently unavailable.
ClouSE: Could not parse traceFilter.	Invalid value is specified in the clouse_trace_filter variable.
ClouSE: The 'listObjects' operation for 'mybucket' failed. The difference between the request time and the current time is too large. (Code=RequestTimeTooSkewed, RequestId=1234)	The time of the machine that ClouSE runs on is different from the AWS time by more than 15 minutes. Check the machine time and update it as needed.
ClouSE: SSL certificate problem, verify that the CA cert is OK. Details:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate verify failed	• Invalid root certificate due to changes in the AWS certificate chain. The clouse_cloud_ssl_cert variable can be used to specify an updated list of root certificates. • https traffic is compromised. • Connecting to a wrong SSL server.
ClouSE: The 'listObjects' operation for 'mybucket' failed.	Invalid value is specified in the

error setting certificate verify locations: CAfile: /tmp/cacert.pem	clouse_cloud_ssl_cert_file variable.
ClouSE: Log file /usr/local/mysql/data/clse_log_hdr.xnl version 1.0.5.x is not supported, supported version is 1.0.2.x.	The installed ClouSE version (1.0.2.x in this example) doesn't support the data format and cannot work with the data. Update ClouSE to a version that supports the data format (in this example that would be 1.0.5.x).
ClouSE: Local log fork 52D0B903BB65FA is outdated by cloud log fork 52D476652CF179.	ClouSE detected that there is another instance of ClouSE that is writing into the same cloud storage location. There must be only one ClouSE instance writing to the same cloud storage location.

11 Using ClouSE Tables

To create a table in ClouSE the ENGINE=CLOUSE option can be used in the CREATE TABLE statement, for example:

```
CREATE TABLE t1 (id INT KEY, data VARCHAR(64)) ENGINE=CLOUSE;
```

Unique indexes cannot contain NULL.

The NO_AUTO_VALUE_ON_ZERO SQL mode is not respected: inserted values that contain 0 or NULL always get a unique generated value.

To create all new tables in ClouSE by default the default-storage-engine configuration option can be used on the command line or in the [mysqld] section of the my.cnf file. Here is an example of setting the default-storage-engine configuration option in the my.cnf file:

```
default-storage-engine=ClouSE
```

To move an existing table to ClouSE the ENGINE=CLOUSE option can be used in the ALTER TABLE statement, for example:

```
ALTER TABLE t2 ENGINE=CLOUSE;
```

Note that if the table uses features that are not supported by ClouSE the alter operation will fail. The ALTER TABLE statement copies all data stored in the table, so it may take a long time.

ClouSE supports transactions. The START TRANSACTION, COMMIT, and ROLLBACK statements can be used to group data access operations into ACID transactions.

To provide isolation between transactions ClouSE implements automatic key range locking. Transactions take shared locks before selecting data and take exclusive locks before inserting, updating, and deleting data. ClouSE also supports update locks that are taken when ClouSE infers the intent of the transaction to modify the data later (e.g. SELECT with FOR UPDATE, UPDATE, and DELETE statements). If a transaction requests a lock that is not compatible with a lock that another transaction holds, it waits until the lock is granted. The compatibility matrix is the following:

	Shared	**Update**	**Exclusive**
Shared	Compatible	Compatible	Conflict
Update	Compatible	Conflict	Conflict
Exclusive	Conflict	Conflict	Conflict

Exclusive locks are always taken before the data is modified and held until the end of the transaction. The policy for shared locks is determined by the transaction isolation level. The transaction isolation levels have the following meaning for ClouSE:

- READ UNCOMMITTED means that ClouSE doesn't take shared locks and the transaction can read data even if a concurrent transaction holds an exclusive lock on it
- READ COMMITTED means that ClouSE takes shared locks, but releases them at the end of each statement
- REPEATABLE READ means that ClouSE takes shared locks and holds them until the end of the transaction; currently this is the same as SERIALIZABLE
- SERIALIZABLE is currently the same as REPEATABLE READ

The default transaction isolation level for ClouSE is specified by MySQL server and is REPEATABLE READ.

It is possible that a lock request cannot be granted because the transaction waits on another transaction that waits on the first transaction. This situation is called deadlock. ClouSE can detect deadlocks that involve ClouSE tables. Here is an example that results in a deadlock:

On client A:

```
CREATE TABLE t1 (id INT KEY) ENGINE=CLOUSE;

START TRANSACTION;
SELECT * FROM t1 WHERE id=1;
```

On client B:

```
START TRANSACTION;
SELECT * FROM t1 WHERE id=2;
INSERT INTO t1 VALUES (1);
```

On client A:

```
INSERT INTO t1 VALUES (2);
```

Now both transactions are waiting for each other until the deadlock detector runs and one of the transactions is going to fail with an error like this:

```
ERROR 1105 (HY000): Waiting for lock X:10:1 aborted due to deadlock.
Deadlock detected:
2->X:10:1==>S:10:1
1->X:10:2==>S:10:2
```

The deadlock information is a list of lock waits (it's a cycle so the first line logically follows the last); each lock wait has the following format: *ConnId->WaitInfo==>ConflictInfo*; each lock info has the following format: *LockMode*:*ObjectId*:*KeyInfo*. The deadlock information above then means the following: the connection 1 is waiting to claim the lock X:10:1 that conflicts with the lock S:10:1, which is held by the connection 2 (because the next line is about the connection 2); the connection 2 is waiting to claim the lock X:10:2 that conflicts with the lock S:10:2, which is held by the connection 1 (because the first line is about the connection 1).

The *ConnId* is the connection identifier for the connections that is running the commands that are involved in the deadlock. To get more information about the connection the SHOW PROCESSLIST statement or INFORMATION_SCHEMA.PROCESSLIST can be used, for example[12]:

```
SELECT id, info FROM INFORMATION_SCHEMA.PROCESSLIST WHERE id=1 OR id=2;

+----+-----------------------------------------------------------------------+
| id | info                                                                  |
+----+-----------------------------------------------------------------------+
|  2 | SELECT id, info FROM INFORMATION_SCHEMA.PROCESSLIST WHERE id=1 OR id=2 |
|  1 | INSERT INTO t1 VALUES (2)                                             |
+----+-----------------------------------------------------------------------+
```

The *LockMode* is one of the S, U or X letters that stand for shared, update or exclusive lock mode correspondingly.

The *ObjectId* is the ClouSE internal object identifier for the table / index. To get more information the INFORMATION_SCHEMA.CLOUSE_TABLES can be used, for example:

```
SELECT * FROM INFORMATION_SCHEMA.CLOUSE_TABLES WHERE object_id=10;

+-----------+-------------+-----------+--------------+------------+
| parent_id | object_name | object_id | table_schema | table_name |
+-----------+-------------+-----------+--------------+------------+
|         0 | ./test/t1   |        10 | test         | t1         |
+-----------+-------------+-----------+--------------+------------+
```

[12] In this example the command is run from the client that got the deadlock error and not running the original command any more.

For more information about the INFORMATION_SCHEMA.CLOUSE_TABLES plugin see INFORMATION_SCHEMA tables for ClouSE.

The *KeyInfo* is just the primary key / index key information. It may contain a key or a range, like [7..30); it may also contain the special value INF, which means 'infinity' (the whole table or everything below / above a certain key is covered by the lock).

Note that ClouSE does not support distributed transactions, so if tables from other engines are mixed with ClouSE tables in the same transaction, the modifications made to tables from other engines may succeed or fail independently.

Currently ClouSE doesn't implement savepoints, so it cannot roll back a single statement and the entire transaction needs to be rolled back.

For more information about transaction control statements and semantics please refer to the MySQL documentation.

ClouSE stores tables and indexes in the B-tree data structure that enables fast key and range lookup. The records are ordered in the key order and are stored inline on the page. The following table has the information about the data stored in ClouSE:

Feature	**Information**
B-tree page size	256 KB.
Max number of columns	About 32000, but generally the max record size is reached before this limit.
Max record size	About 128 KB.
Max key size	About 128 KB. Key + data size must be less than the max record size.

Note, that for some features MySQL imposes stricter limits in which case the MySQL limits are used. For more information about MySQL limits please refer to the MySQL documentation.

12 INFORMATION_SCHEMA tables for ClouSE

ClouSE provides an INFORMATION_SCHEMA plugin that can be used to view ClouSE tables and indexes. The INFORMATION_SCHEMA.CLOUSE_TABLES view can be used to get information about ClouSE tables and indexes. For example:

```
SELECT * FROM INFORMATION_SCHEMA.CLOUSE_TABLES;

+-----------+---------------------+-----------+--------------+---------------+
| parent_id | object_name         | object_id | table_schema | table_name    |
+-----------+---------------------+-----------+--------------+---------------+
|         0 | ./test/clip_comments |        10 | test         | clip_comments |
|         0 | ./test/clip_related  |        12 | test         | clip_related  |
|         0 | ./test/clips         |         9 | test         | clips         |
|        10 | time                 |        11 |              |               |
```

```
|         12 | rank                  |          7 |              |                |
+------------+-----------------------+------------+--------------+----------------+
```

The semantics of the CLOUSE_TABLES fields are the following:

Field	Semantics
parent_id	For indexes, the object_id of the table that the index is created for. For tables it's 0.
object_name	The name that is used by ClouSE as a key to identify the table / index. This corresponds to the name that MySQL passes down to ClouSE.
object_id	The ClouSE internal object identifier of the table / index.
table_schema	For tables, the database name, as parsed out of the object name using MySQL rules. If the object name doesn't correspond to a valid MySQL path this field contains an empty string.
table_name	For tables, the table name, as parsed out of the object name using MySQL rules. If the object name doesn't correspond to a valid MySQL path this field contains an empty string.

13 Disaster Recovery with ClouSE

13.1 Disaster Recovery from Local Transaction Log

If MySQL data is lost but the ClouSE local transaction log is accessible, the data stored in ClouSE can be completely recovered.

To recover ClouSE data, follow the steps described in ClouSE Setup and Configuration to deploy ClouSE binaries and configure access to the cloud storage and the local transaction log. Then execute the SHOW DATABASES statement to let MySQL discover database information from ClouSE.

Caution: Only one ClouSE at a time should access the cloud storage location! **The data may get corrupted beyond repair** if two or more ClouSE instances access the same cloud storage location. When recovering ClouSE data please make sure that the new ClouSE instance is the only one accessing the cloud storage location.

The recovery happens on the fly via callbacks that MySQL uses to discover metadata from the storage engine: if MySQL doesn't find the .frm file on disk it asks all storage engines if they know how to recover it. If the server instance has other tables then it might result in a situation when another table with the same name prevents a ClouSE table to be discovered.

ClouSE provides an INFORMATION_SCHEMA plugin that can be used to view ClouSE tables for example:

```
SELECT * FROM INFORMATION_SCHEMA.CLOUSE_TABLES WHERE parent_id=0;
```

```
+-----------+----------------------+-----------+--------------+---------------+
| parent_id | object_name          | object_id | table_schema | table_name    |
+-----------+----------------------+-----------+--------------+---------------+
|         0 | ./test/clip_comments |        10 | test         | clip_comments |
|         0 | ./test/clip_related  |        12 | test         | clip_related  |
|         0 | ./test/clips         |         9 | test         | clips         |
+-----------+----------------------+-----------+--------------+---------------+
```

For more information about the INFORMATION_SCHEMA.CLOUSE_TABLES plugin see INFORMATION_SCHEMA tables for ClouSE.

Generally recovery doesn't need any special actions as the tables in ClouSE are automatically discovered by MySQL, but in complex cases the ClouSE tables can be renamed directly in ClouSE using the **renameobj** ClouSE admin command. The command takes two arguments that specify the source and target names separated by space. For example:

```
SET GLOBAL clouse_admin_command='renameobj ./test/clips ./test/clips2';
```

The command would rename the ClouSE object ./test/clips into ./test/clips2. Backticks can be used to quote unusual names; a double-backtick can be used to use a backtick in a name. Any arbitrary object name can be specified, but if the name doesn't conform to MySQL conventions for a path to a table then the object cannot be accessed from MySQL. The name convention is <root_dir>/<database>/<table> where the <root_dir> is typically ".". The database and table names can be encoded according to the MySQL conventions that can be found in the MySQL documentation, but for practical purposes it's generally sufficient to rename an object into some ASCII alphanumeric name, let MySQL discover the table and then use the RENAME statement to rename it further as needed.

13.2 Disaster Recovery from Cloud Storage

If the ClouSE local transaction log is lost or corrupted, it is possible to recover the local transaction log from the cloud storage with potential loss of seconds of transactions. To minimize the possibility of the loss it is recommended to place the local transaction log to reliable storage to reduce chance of losing the local transaction log.

To recover ClouSE data, follow the steps described in ClouSE Setup and Configuration to deploy ClouSE binaries and configure access to the cloud storage and the local transaction log.

Then execute the SHOW DATABASES statement to let MySQL discover database information from ClouSE.

In most cases, no further action is required for recovery. See Disaster Recovery from Local Transaction Log for further discussion on the functionality that can be used to interrogate and control ClouSE data.

13.3 Point-in-time Recovery

The database is protected from corruption and data loss by the cloud storage; however, cloud storage cannot protect from operational mistakes. The user may accidentally drop the database or update all records in a table to a wrong value. ClouSE will make sure that the result of the user's operation is made durable in the cloud storage; it cannot know that it's a mistake.

To protect from operational mistakes, ClouSE supports built-in point-in-time recovery. It automatically retains history, such that it's possible to restore data to any point in time within the retention period with a second granularity. The default history retention period is 24 hours, but can be configured to a different value (see History Retention for Point-in-time Recovery).

A point-in-time recovery operation is initiated using the **restoreto** ClouSE admin command. The command has the following format: **restoreto *YYYY-MM-DD HH:mm:ss+tztz***. The time zone offset must be explicitly specified to avoid confusion between the user's and server's time zones.

Here is an example of a **restoreto** command that restores the data to Jan 15, 2014 at 5:03:34pm PST (GMT-08:00):

```
SET GLOBAL clouse_admin_command='restoreto 2014-01-15 17:03:34-0800';
```

After the point-in-time recovery operation finishes, the database is restored to the state it was at the specified time.

A point-in-time recovery operation is non-destructive. After a point-in-time recovery operation is done, the user can do another point-in-time recovery to recover to an earlier or a later point in time (e.g. because the original point in time was wrong), or even undo the original point-in-time operation by restoring the database to the point in time just before the operation started.

As an example, let's consider the following sequence of events:

Operation	**State**	**Time**
Insert value 42	Value 42	17:00:00
Insert value 77	Values 42 and 77	17:05:00
Insert value 153	Values 42, 77 and 153	17:10:00
Restore to 17:09:00	Value 42 and 77	17:15:00
Restore to 17:04:00	Value 42	17:20:00
Restore to 17:14:00	Values 42, 77 and 153	17:25:00

The state represents what the database contains after the operation (assuming it was initially empty) and the time is the time the operation happened. From the example it's easy to see that point-in-time recovery operations restore the database state to what it was at that point in time.

14 How to File Bug Reports

Before you file a bug report, please try to verify that it is a bug and that it hasn't been reported already:

- See Troubleshooting Configuration and Connection Issues
- Check the list of ClouSE known issues and bugs: http://www.oblaksoft.com/known-issues
- Try the latest ClouSE version from https://www.oblaksoft.com/downloads

If you aren't able to find a resolution, we would like you to fill out the bug report form below and send it to bugs@oblaksoft.com. We will reply with a tracking number for the reported issue. Please include a concise bug summary into the email subject.

14.1 Bug Report Form

MySQL version From MySQL client run: SELECT @@version;	
ClouSE version From MySQL client run: SELECT @@clouse_version; From shell (go to plugin directory) run: strings clouse.so | grep @VERSION@	
Operating system information From shell run: uname -a	
Bug Type (choose one: code defect, feature/change request, suggestion, documentation, security)	
Repro steps (what did you do?) We would appreciate a short list of steps, the shorter the better☺ Please use the exact list of commands that led to the problem (every little detail may matter) or provide a MySQL / shell / perl / etc. script that can be used to reproduce the problem.	
Expected result (what did you expect to happen?)	
Actual result (what has actually happened?) Please describe the results as precisely as possible. If the results contain	

error or other messages please copy and paste the messages with all the details. If MySQL error log contains errors from ClouSE, please provide those as well. In the case of a crash please provide the stack backtrace from MySQL error log. In the case of a hang please see How to Capture a Backtrace for a Running Process.	
Notes (any other details that can help identify the problem).	

14.2 Howtos

14.2.1 How to Generate a Core Dump when mysqld Crashes

1. Remove any limits on core dump size. To do so include the **ulimit -c unlimited** command to the scripts that are used to execute the mysqld program prior to mysqld invocation. If running mysqld manually from shell, run the **ulimit -c unlimited** command before running mysqld.

2. Add the core-file option to my.cnf in [mysqld] section (http://dev.mysql.com/doc/refman/5.5/en/mysql-cluster-program-options-common.html#option_ndb_common_core-file)

3. You may need to enable the following settings as well and optionally set up naming pattern for dumps being generated:

   ```
   echo 2 > /proc/sys/fs/suid_dumpable
   mkdir /tmp/coredumps
   chmod 770 /tmp/coredumps
   echo /tmp/coredumps/core > /proc/sys/kernel/core_pattern
   echo 1 > /proc/sys/kernel/core_uses_pid
   ```

14.2.2 How to Get a Core Dump for a Running Process

From shell run: gcore *processId*

Or alternatively from shell run: gdb -p *processId* --batch -ex gcore

14.2.3 How to Capture a Backtrace for a Running Process

From shell run: gdb -p *processId* --batch -ex "thread apply all backtrace full" > backtrace.txt

14.2.4 How to Get a Backtrace From a Core Dump

From shell run: gdb --core=*coreFile* --se=*procFile* --batch -ex "thread apply all backtrace full" > backtrace.txt

For example:

```
gdb --core=~/temp/core.4723 --se=/opt/mysql/server-5.5/bin/mysqld --batch -ex
"thread apply all backtrace full" > backtrace.txt
```