NEC TM Technical Description

Version 2.03

This report is the result of the Connecting Europe Facility (CEF) programme

Action No: 2017-EU-IA-0149

Agreement number: INEA/CEF/ICT/A2017/1565776

0. Revision Control **4**

1. INTRODUCTION **4**
1.1 Purpose 4
1.2 Scope 4
1.3 Overview 5
1.4 Definitions and Acronyms 5
1.5 Dependencies 5

2. SYSTEM OVERVIEW **5**

3. SYSTEM ARCHITECTURE **6**
3.1 Architectural Design 6
3.3 Design Rationale 9

4. DATA DESIGN **10**
4.1 Data Description 10
4.1.1 Monolingual index 10
4.1.2 Bilingual index (Map DB) 11
4.1.3 Users & Scopes 12

5. Prerequisites of NEC TM **12**

6. Maintenance **13**
6.1 Get NEC TM Docker image (latest version) 13
6.2 Run NEC TM for the first time 13
6.3 Stop NEC TM 13
6.4 Backup NEC TM data 13
6.4.1 Backup ElasticSearch data 14
6.4.2 Backup PostgreSQL data 15
6.5 Restoring NEC TM data 15
6.5.1 Restoring ElasticSearch data 15
6.5.2 Restoring PostgreSQL data 16
6.6 Restarting NEC TM 16
6.7 Updating NEC TM 16
6.7.1 Full Update 16
6.7.2 Partial Update 17
6.8 NEC TM logs 17
6.9 Checking the status 17

6.10 Troubleshooting 18

7. Performance test 18

8. Functionality test 18

9. Permissions test 19

10. Metadata test 19

11. Security of NEC TM 19

12. Integration with eTranslation 20

13. Integration with ELRC 20

14. TMX Anonymizer 20

14.1 Requirements 20

14.2 Run TMX Anonymizer 20

14.3 Anonymize TMX 21

0. Revision Control

Author	Description
Alex Helle	Initial Version of Specification (Sept 25th 2018)
Alex Helle	Draft version with installation instructions and system management instructions (Apr 26th 2019)
Amando Estela	Revision (Apr 26th 2019)
Alex Helle	Update the index (May 22th 2019)
Alex Helle	Change in the process of Backup/Restore of ElasticSearch (May 22th 2019)
Alex Helle/Carmen Herranz	Revision (May 30th 2019)
Alex Helle/Alex Raginsky	Changes in the parameters for new functions (metadata) (February 24th 2020)

1. INTRODUCTION

1.1 Purpose

This document describes the architecture and system design of NEC TM based on ActivaTM, cloud-based Translation Memory tool. The target audience are developers, system administrators and advanced users of the tool.

1.2 Scope

ActivaTM is a fast, highly-scalable cloud-based Translation Memory tool developed by

Pangeanic as a part of the EXPERT EU project. The goal of ActivaTM is to seamlessly integrate all available TMs to serve vendor-independent tools in the standard translation flow. ActivaTM integrates (via plugins) with multiple manual and machine translation software (SDL Studio, PangeaMT, NeuralMT, Bing Translate).

NEC TM Data project will provide the centralised infrastructure for efficient data sharing, TM matching, TM retrieval, and domain categorisation of resources generated by Member States/ EEA. This will enable the development of NEC TM, an open source software developed from Pangeanic's translation memory database ActivaTM.

1.3 Overview

The document describes design and architecture of ActivaTM and it subcomponents

1.4 Definitions and Acronyms

TM - Translation Memory

ES - ElasticSearch

CAT - Computer-Assisted Translation

Translation unit - data structure, containing two (or more) text segments of different language along with additional properties. Sometimes used interchangeably with segment

Segment - single-language text string, part of translation unit

TMX - Translation Memory standard eXchange file format

1.5 Dependencies

ElasticSearch - 6.*

Apache Spark - 1.6.4

PostgreSQL - 11*

Celery (Redis) - 2.*

During the scope of the project some of these dependencies maybe upgraded.

2. SYSTEM OVERVIEW

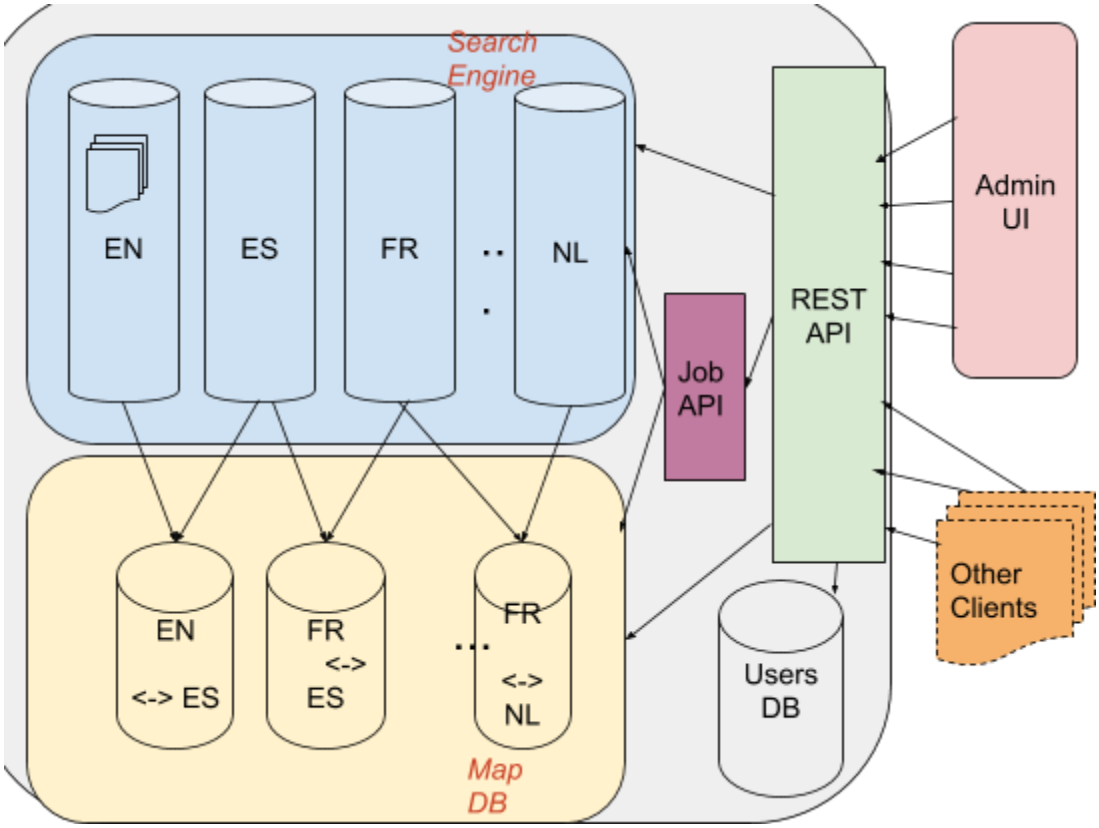
A Translation Memory (TM) is a collection of segments helping to improve the quality of

translation processes in CAT tools (such as Trados Studio) by reusing translation of phrases, sentences or even paragraphs (commonly known as segments) from previously known translations. Usually, automatic machine translation tools use translation memory in-memory (loaded from standard TMX format) or by using web service. The number of segments can be huge if all possible European language combination pairs are taken into account. Current non-commercial solutions appear to be either non-scalable (such as dealing with TMX files) or in early stages of development, such as TinyTM database. This document describes the design of a fast and scalable cloud-based TM database with the goal of overcoming the limitations of current solutions for TM storage & query.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

This diagram shows the main components of ActivaTM:



Search Engine (*based on ElasticSearch*[1]**)** holds monolingual segment text in a separate index (aka monolingual index), e.g. for example, all English segments are stored separately no matter what language they have been mapped with in the source TM data. This enables a significant conservation of memory, as each and every unique segment is stored only once. The segments are stored along with their corresponding properties, such as industry, type, organization etc. Furthermore each segment is allocated a unique id (UUID, calculated by hashing a segment string) once added to the system. Sample EN segment looks similar to the following snippet (JSON representation):


```
{
        "text" : "Connect the pipe to the female end of the T.",
        "id" : "1b45648d-5b94-525a-ba75-0b16d3ce3d7e",
        "industry" : ["Automotive Manufacturing"],
        "type" : ["Instructions for Use"],
        "organization" : ["pangeanic"],
        "language" : ["en-GB"]

}
```


Sample corresponding ES segment is stored in a corresponding index (JSON representation):


```
{
        "text" : "Conecte la tubería al extremo hembra de la T.",
        "id" : "b9049718-157b-5a7b-9c6a-f2a5ac8265cb",
        "industry" : ["Automotive Manufacturing"],
        "type" : ["Instructions for Use"],
        "organization" : ["pangeanic"],
        "language" : ["es-ES"]

}
```


Once the search engine is queried for the source language segment, it will look first for an exact match, if an exact match is not found, the system will widen the search conditions to support fuzzy matching and regular expressions. The selected search engine should support all necessary query combinations: exact match, fuzzy match and regular expressions.

Map DB (*based on ElasticSearch***)** holds collections of segment mappings for each pair of languages. Each language pair is stored in a separate ES index (aka bilingual index). This way when we find a match in a source language segment DB (indexed by search engine), we will quickly retrieve corresponding the UUID of the target language segment. The text of a target segment will be retrieved by querying a corresponding search engine index.

[1] "Elastic." https://www.elastic.co/. Accessed 26 April. 2019.

Map DB supports a quick bulk update operation enabling future updates of a segment, for example, modifications such as data changes. Sample mapping of the previously mentioned segments are stored in Map DB in a following way as a part of "EN-ES" database:


```
{
        "source_id" : "1b45648d-5b94-525a-ba75-0b16d3ce3d7e",
        "target_id" : "b9049718-157b-5a7b-9c6a-f2a5ac8265cb",
        "creation_date" : "20090914T114346Z",
        "change_date" : "20090914T114346Z",
}
```


Each language pair is held in a separate ES index named after a language pair, e.g. "EN <-> ES" language mappings are stored in the"EN-ES" collection, thus identifying the source and target language of stored segments.

Users DB (*based on PostgreSQL*[2]*)* - contains users and their permissions (scope). For each user defined in the system, the UsersDB stores a username, encrypted password, list of scopes (permissions) and user settings. The scopes are associated with a user and stored in a separate table containing a list of permitted language pairs, domains, usage count (number of times the user has utilised a scope), limit (maximum number of times the scope can be used) along with start and end date of the scope.

Job API (*based on Celery*[3] *and Apache Spark*[4]*)* is a mechanism for parallel background processing of lengthy client requests (such as adding, deleting and cleaning segments) preventing the blocking of the REST API. The mechanism is triggered by REST API requests, queuing job to Celery and executed by Apache Spark which processes the data in the background

REST API (*based on Flask*[5]*)* is a programming interface for interacting with ActivaTM content(adding, deleting, updating, cleaning translation units etc.), managing users and managing background jobs. The user is authorized by accessing authorization endpoint (with username and password). Access to other endpoints is granted based on JWT token return by authorization endpoint and depending on user role (admin or user). More information in the "NEC TM - Admin UI description" manual.

[2] "PostgreSQL." https://www.postgresql.org/. Accessed 26 April. 2019.
[3] "Homepage | Celery: Distributed Task Queue." http://www.celeryproject.org/. Accessed 26 April. 2019.
[4] "Apache Spark™ - Unified Analytics Engine for Big Data." https://spark.apache.org/. Accessed 26 April. 2019.
[5] "Flask." http://flask.pocoo.org/. Accessed 26 April. 2019.

3.3 Design Rationale

For Search Engine, we looked mainly on Lucene-based search engines which dominate the open-source search engine market : Solr and ElasticSearch both satisfy our basic requirements, supporting out-of-the box exact matching, fuzzy matching and regular expressions. Moreover, both engines are very popular in industry and in the academy. ElasticSearch has a couple of major advantages which makes it our top pick:

- ElasticSearch is scalable almost transparently when Solr requires more effort and additional software (ZooKeeper) to set up clusters and shards
- ElasticSearch has a powerful Query DSL (structured JSON language to build complex queries easily)

Several alternatives were considered to serve as a Map DB. We looked primarily at NoSQL solutions such as:

- ElasticSearch - full-text search engine but can be used as key-value storage
- MongoDB - a document-oriented database
- CouchDB - another document-oriented database
- Redis - fast in-memory key-value database, used mainly as a cache

All of the databases provide key-value mapping interface via HTTP and suitable for purposes of ActivaTM and scale horizontally rather easily due to their nature of being NoSQL solutions. Consequently, the major differentiation should be their performance of bulk adding and one-by-one key querying as well as memory consumption. The benchmark experiments were conducted on English-Spanish TMX files having overall 440K translation memories. The following table demonstrates the recorded execution times and memory consumption:

Metric/Engine	**ElasticSearch**	**MongoDB**	**CouchDB**	**Redis**
Bulk adding (47K segments)	83s	432s	***67s***	458s
Bulk adding (440K segments)	858s	6112s	644s	***621s***
Querying (1K segments)	***11s***	23s	52s	51s
Querying (10K	***51s***	187s	458s	72s

segments)				
Querying (440K segments)	1400s	6451	19647	***1210s***
Process size (440K segments)	4.9G [1]	549M	771M	***148M***

[1] *ElasticSearch process maintains indexes of monolingual texts too and thus can't be directly compared in terms of memory consumption of MapDB only*

4. DATA DESIGN

4.1 Data Description

Mainly, ActivaTM deals with **translation units**, a data structure containing source segment (text in source language), target segment (text in target language), multiple timestamps (creation date, modification date) and various properties (industry, type, organization). Translation units are currently parsed from TMX file and translated into internal ActivaTM data. This section describes the internal data structures of ActivaTM.

ActivaTM has the following databases:

- Monolingual indexes (Search Engine, stored in ElasticSearch)
- Bilingual indexes (Map DB, stored in ElasticSearch)
- Users & scopes (stored in PostgreSQL)

4.1.1 Monolingual index

Monolingual index stores text segments of a single language along with few additional auxiliary properties (list of target languages, token count). ElasticSearch index is named as "tm_<language code>", for example **tm_en** and has the following schema:

- ***_id*** - UUID-based unique id, allocated by hashing the text[6]. Same-text segments will have the same id (though, there is a minor chance of collision, currently ignored)
- ***text*** - contains raw text, the field is analyzed (e.g. tokenized and searchable)
- ***target_language*** - contains codes of target languages for which this segment has corresponding translation unit. For example, if translation unit is EN-ES ("hello" ⟵ ⟶ "hola"), then English segments will have 'es' as a target language. Similarly,

[6] "22.20. uuid — UUID objects according to RFC 4122 — Python 3.7.0 ...." https://docs.python.org/3/library/uuid.html. Accessed 26 April. 2019.

Spanish segments such as "hola" will have 'en' as a target language. If there is an additional EN-FR translation unit ("hello" ← → "bonjour"), then "hello" segment's target_language field will contain the list: *["en", "fr"]*. This field is mainly used to Generate algorithms

- ***token_cnt*** **-** number of tokens in text. Used for internal matching algorithm

4.1.2 Bilingual index (Map DB)

Bilingual index stores all fields of translation units needed to recreate TMX file properly. Additionally, it contains IDs of its source and target texts so that search algorithms can map the found source segment text to actual translation units stored in bilingual index. The ElasticSearch index is named as "map_<lang1>_<lang2>", for example "map_en_es". The same index serves for EN->ES and ES->EN queries, imports, exports etc. The index has the following schema:

- ***_id*** - UUID-based unique id, allocated by hashing the concatenation of source and target text segments[7]. Same-text translation units will have the same id (though, there is a minor chance of collision, currently ignored)
- ***source_id, target_id*** **-** IDs of correspondent text segments in monolingual indexes
- ***source_text, target_text*** **-** source and target segment texts, stored here as not-analyzed, to speed up TMX generation and to avoid accessing monolingual indexes
- ***source_metadata, target_metadata*** **-** source and target segment metadata, stored here as not-analyzed, allows to match pre- and post-segments to support 101% match during search.
- ***source_language, target_language*** **-** source and target languages, including locale (en-GB, es-AR etc)
- ***metadata*** **-** translation unit-level metadata
- ***domain*** **-** list of domains (tags) assigned to the translation unit
- ***filename*** **-** list of filename from where the translation unit comes
- ***Other*** **-** other optional fields (industry, type and organization) can be added during import if they are specified in the TMX file.

To emphasize the difference, bilingual index data is usually accessed by ID and not by free-text search, for that monolingual index is used.

[7] "22.20. uuid — UUID objects according to RFC 4122 — Python 3.7.0 ...." https://docs.python.org/3/library/uuid.html. Accessed 26 April. 2019.

4.1.3 Users & Scopes

Users and their permission scopes (in short, scopes) are stored in PostgreSQL in the following tables:

- Table: **users**
  - *username*
  - *password*
  - *role* - either “user” or “admin”. Admin has access to all ActivaTM interfaces, User is limited to querying and limited statistics
  - *token_expires* - whether JWT expires within 24 hours or stays valid indefinitely (useful for automatic API clients such as crawler)
  - *is_active* - whether user is active. Inactive user’s access will be denied
  - *created* - timestamp of user creation date
- Table: **user_scopes**
  - *id*
  - *username* - user to whom this scope belongs
  - *lang_pairs* - list of language pairs, for them the scope is applicable
  - *domains* - list of domains, for them the scope is applicable
  - *usage_limit* - maximal number of user queries applicable to this scope. If the user exceeds this number, access will be denied
  - *usage_count* - current number of user queries matching this scope
  - *start_date* - beginning of this scope validity
  - *end_date* - end of this scope validity
  - *can_import* - if import of TMX files is allowed in this scope
  - *can_export* - if export to TMX files is allowed in this scope
  - *can_update* - if update of translation units is allowed in this scope

5. Prerequisites of NEC TM

A small summarization of the hardware and software requirements:

- Hardware:
  - RAM: 64GB recommended, 16GB minimum
  - CPU: Not important
  - Disk: SSD of 1TB recommended, 256GB minimum
- Software:
  - SO: Ubuntu 16.04 recommended or later

6. Maintenance

6.1 Get NEC TM Docker image (latest version)

Download the last version from the Docker repository:

$ sudo docker pull nectm/activatm:latest

6.2 Run NEC TM for the first time

NEC TM requires 3 ports to be opened on the host machine. In this example the (mapped) ports are 27979, 27878 and 19200 but they could be changed if needed.

The host directory requires NEC TM data to be stored on a disk (in this example the directory is /ssd/elasticsearch_data):

$ sudo mkdir -p /ssd/elasticsearch_data
$ sudo chmod -R aog+w /ssd/elasticsearch_data

Run NEC TM:

$ sudo docker run -p 27979:7979 -p 27878:7878 -p 19200:9200 -v /ssd/elasticsearch_data:/elasticsearch_data --name nectm -t nectm/activatm:latest

6.3 Stop NEC TM

Stop the NEC TM docker instance:

$ sudo docker stop nectm

6.4 Backup NEC TM data

The data of NEC TM is separated in the ElasticSearch data and the PostgreSQL data:

6.4.1 Backup ElasticSearch data

The backups are done through snapshots stored in a repository[8]:

- Check if the repository of snapshots is registered
  $ sudo docker exec -it nectm curl -X GET "localhost:9200/_snapshot/_all"

- If the repository doesn't exist, register it
  $ sudo docker exec -it nectm curl -X PUT "localhost:9200/_snapshot/my_backup" -H 'Content-Type: application/json' -d' {"type": "fs","settings": {"location": "/elasticsearch_data/backup"}}'

- Finally create the snapshot using an associated name (posterior snapshots should have different names)
  $ sudo docker exec -it nectm curl -X PUT "localhost:9200/_snapshot/my_backup/snapshot_1?wait_for_completion=true"

- List snapshots done:
  $ sudo docker exec -it nectm curl -X GET "localhost:9200/_snapshot/my_backup/_all"
  {"snapshots":[{"snapshot":"snapshot_1","uuid":"3HpVRYiPR1CBLajTBlAp0w","version_id":6070099,"version":"6.7.0","indices":["tm_en","jobs","map_en_es","tm_es","tm_sv","map_en_sv","query_log-2019.05"],"include_global_state":true,"state":"SUCCESS","start_time":"2019-05-20T15:34:05.849Z","start_time_in_millis":1558366445849,"end_time":"2019-05-20T15:34:06.934Z","end_time_in_millis":1558366446934,"duration_in_millis":1085,"failures":[],"shards":{"total":35,"failed":0,"successful":35}}]}

  $ ls -lhtr /ssd/elasticsearch_data/backup/
  total 56K
  -rw-r--r-- 1 systemd-resolve systemd-network 31K may 20 17:34 meta-3HpVRYiPR1CBLajTBlAp0w.dat
  drwxr-xr-x 9 systemd-resolve systemd-network 4,0K may 20 17:34 indices
  -rw-r--r-- 1 systemd-resolve systemd-network 301 may 20 17:34 snap-3HpVRYiPR1CBLajTBlAp0w.dat
  -rw-r--r-- 1 systemd-resolve systemd-network 29 may 20 17:34 incompatible-snapshots
  -rw-r--r-- 1 systemd-resolve systemd-network 902 may 20 17:47 index-3
  -rw-r--r-- 1 systemd-resolve systemd-network 663 may 20 17:49 index-4
  -rw-r--r-- 1 systemd-resolve systemd-network 8 may 20 17:49 index.latest

- Delete snapshot:
  $ sudo docker exec -it nectm curl -X DELETE "localhost:9200/_snapshot/my_backup/snapshot_1"

- Copy the repository of snapshots:

[8] More information: https://www.elastic.co/guide/en/elasticsearch/reference/6.7/modules-snapshots.html. Accessed 22 May. 2019.

$ tar -zcvf /path/to/backup/elasticsearch-backup.tar.gz /ssd/elasticsearch_data/backup/

6.4.2 Backup PostgreSQL data

$ sudo docker exec -it nectm su postgres -c "pg_dump activatm" | gzip -c > psql_backup.sql.gz

6.5 Restoring NEC TM data

The data of NEC TM is separated in the ElasticSearch data and the PostgreSQL data:

6.5.1 Restoring ElasticSearch data

- Restore backup for the snapshot repository, and move the directory to the same location where it was originally located:
  $ tar -zxvf /path/to/backup/elasticsearch-backup.tar.gz
  $ mv /path/to/backup/ssd/elasticsearch_data/backup/ /ssd/elasticsearch_data/backup/*

- Input the right permissions:
  $ sudo docker exec -it nectm chown -R elasticsearch:elasticsearch /elasticsearch_data/backup/

- Check if the snapshot repository is registered:
  $ sudo docker exec -it nectm curl -X GET "localhost:9200/_snapshot/_all"

- If the repository doesn't exist, register it:
  $ sudo docker exec -it nectm curl -X PUT "localhost:9200/_snapshot/my_backup" -H 'Content-Type: application/json' -d' {"type": "fs","settings": {"location": "/elasticsearch_data/backup"}}'

- List snapshots found in the restored repository:
  $ sudo docker exec -it nectm curl -X GET "localhost:9200/_snapshot/my_backup/_all"
  {"snapshots":[{"snapshot":"snapshot_1","uuid":"3HpVRYiPR1CBLajTBlAp0w","version_id":6070099,"version":"6.7.0","indices":["tm_en","jobs","map_en_es","tm_es","tm_sv","map_en_sv","query_log-2019.05"],"include_global_state":true,"state":"SUCCESS","start_time":"2019-05-20T15:34:05.849Z","start_time_in_millis":1558366445849,"end_time":"2019-05-20T15:34:06.934Z","end_time_in_millis":1558366446934,"duration_in_millis":1085,"failures":[],"shards":{"total":35,"failed":0,"successful":35}}]}

- Finally restore a snapshot using the associated name
  sudo docker exec -it nectm curl -X POST "localhost:9200/_snapshot/my_backup/snapshot_1/_restore"

- If it gives an error related to open index, close all the indices:
  sudo docker exec -it nectm curl -X POST "localhost:9200//_close"*

- Check the status of the restore:
  sudo docker exec -it nectm curl -X GET "localhost:9200/_snapshot/my_backup/snapshot_1/"

6.5.2 Restoring PostgreSQL data

$ sudo docker exec -i nectm /etc/init.d/postgresql restart; sudo docker exec -i nectm su postgres -c "dropdb activatm"; sudo docker exec -i nectm su postgres -c "createdb activatm"; zcat psql_backup.sql.gz | sudo docker exec -i nectm su postgres -c "psql -d activatm"

6.6 Restarting NEC TM

Restart the NEC TM docker instance:

$ sudo docker stop nectm
$ sudo docker start nectm

6.7 Updating NEC TM

There is two types of updates: full update (requires backup) and partial update (not requires backup).

6.7.1 Full Update

Prior to updating NEC TM, a backup shall be done (refer to section 6.4).

Stop NEC TM:

$ sudo docker stop nectm

Make sure that you have NEC TM's latest version:

$ sudo docker pull nectm/activatm:latest

Remove the old NEC TM docker:

$ sudo docker rm nectm

Run new NEC TM docker:

$ sudo docker run -p 27979:7979 -p 27878:7878 -p 19200:9200 -v /ssd/elasticsearch_data:/elasticsearch_data --name nectm -t nectm/activatm:latest

After updating, it is necessary to restore backup.

6.7.2 Partial Update

Update the content of the docker:

$ sudo docker exec -it nectm su root -c "git pull"

If you see an error executing the previous step, please execute this:

$ sudo docker exec -it nectm su root -c "git stash"
$ sudo docker exec -it nectm su root -c "git pull"

Restart NEC TM:

$ sudo docker exec -it nectm su root -c "supervisorctl restart all"

6.8 NEC TM logs

It is possible to see the different logs with:

sudo docker exec -it nectm su root -c "tail -f /elastictm/log/elastictm/gunicorn.log"

6.9 Checking the status

It is possible to see the status of the processes related to NEC TM:

sudo docker exec -it nectm su root -c "supervisorctl status"

6.10 Troubleshooting

- If 502 error is shown during the running/starting of NEC TM:

  sudo docker exec -it nectm su root -c "supervisorctl restart all"

  If the problem persists, check that the /ssd/elasticsearch_data has the right permissions, if not, apply them again:

  $ sudo chmod -R aog+w /ssd/elasticsearch_data

7. Performance test

The performance test has been run with a computer with much lower specifications than the ones required for NEC TM. The computer was a 4-Core Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz, 8G Memory and 120GB SSD:

- Import: 170 TUs/second
- Export: 1350 TUs/second
- Delete: 1720 TUs/second
- Query: 0.2 second/query

All measurements were carried out on TM data with 480K translation units.

8. Functionality test

There is a basic test which tests the basic functionalities of NEC TM. This basic test is run from inside the docker and the path is relative to the root of NEC TM docker. The login and the password are NEC TM's admin user's (default user/pass is admin/admin). Port is an internal docker port which ActivaTM is listening to (default is 7979):

sudo docker exec -t nectm test/basic_test.py [--login <login>] [--pwd <password>] [--port <port>]

9. Permissions test

There is a permission test which tests different types and combinations of permissions in NEC TM. This permissions test is run from inside the docker and the path is relative to the root of NEC TM docker. The login and the password are NEC TM's admin user's (default user/pass is admin/admin). Port is an internal docker port which ActivaTM is listening to (default is 7979):

> *sudo docker exec -t nectm test/permission_test.py [--login <login>] [--pwd <password>] [--port <port>]*

10. Metadata test

There is a metadata test which tests metadata functionality of NEC TM. This metadata test is run from inside the docker and the path is relative to the root of NEC TM docker. The login and the password are NEC TM's admin user's (default user/pass is admin/admin). Port is an internal docker port which ActivaTM is listening to (default is 7979):

> *sudo docker exec -t nectm test/metadata_test.py [--login <login>] [--pwd <password>] [--port <port>]*

11. Security of NEC TM

Internally, NEC TM uses the following libraries for security:

- *flask-jwt* for handling jwt-based authorization
- *werkzeug* for hashing the passwords

Passwords are stored hashed in PostgreSQL.

All the rest of security should be handled by the hosting machine via securing NEC TM port access using web server and https.

12. Integration with eTranslation

Different Machine Translation (MT) systems can be configured to perform backup response after failing to get a positive match. One of the systems that can be configured is EU DGT eTranslation services.
The connection to eTranslation services is provided thru MT-HUB.

13. Integration with ELRC

TMX exported files use V1.4.b format version, and this accepted for the ELRC upload procedures.

14. TMX Anonymizer

The TMX Anonymizer will be made available during the scope of the NEC TM data project. The TMX Anonymizer is a independent toolkit for bilingual TMX files for the most widely used european languages and it will be a preliminary version of the anonymization algorithm.

14.1 Requirements

A small summarization of the software requirements:

- Ubuntu 18.04 or higher (recommended) or any Linux distribution
- Docker

14.2 Run TMX Anonymizer

Open a new terminal and type this command.

sudo docker run -p 8080:8080 -t pangeamt/mclehm_tmx_anonymizer:1

The first time this command will be executed. The docker container will be downloaded. It could take a while.

14.3 Anonymize TMX

Open a new terminal and navigate to the directory containing the TMX files you would like to anonymize. Type this command replacing file.tmx with your TMX file.

curl -F 'tmx=@file.tmx' http://0.0.0.0:8080 -J -O

Notice that file.tmx has to be changed to the actual name of the file to anonimize and 0.0.0.0:8080 will have to be changed to the actual ipaddress and port the anonymizer is running.

This will download the anonymized TMX file (called file.anon.tmx).