Esta herramienta permite que a partir de cualquier modelo genérico en ASN1 se pueda implementar un modelo en Cassandra , así como interaccionar con telemetrías y telecomandos mediante ficheros CSV.

Se han preparado dos escenarios:

- Linux (Ubuntu 22.04)
- Docker

Pasos instalación en Linux:

Será necesario tener instalado el compilador ASN1SCC y cassandra.

Instalación del compilador:

ejecutar el script ***install.sh***(*) como sudo
ejecutar la siguiente sentencia:
export PATH=/compiler/asn1scc:$PATH

(*)Incluso entre distintas versiones de la misma distribución (Ubuntu), da problemas.

Aquí se muestra un ejemplo con la version 22.04 pero podrían ocurrir otros errores en otras distribuciones de Linux .

Comprobar que está correctamente instalado ejecutando:

asn1scc –help

Instalación de Cassandra: ejecutar el script ***install_cassandra.sh*** como sudo

Comprobar que está correctamente instalado y arrancado ejecutando:

service cassandra status

Pasos instalación en Docker:

Es la opción preferente para hacer pruebas por dos razones:

- No hay problemas de compatibilidad del S.O. en la instalación(esto es debido al compilador)
- Es más sencillo crear un escenario de varios nodos de un cluster

El único requisito es tener Docker instalado(descargar docker).

En el caso de Windows es necesario hacer unas modificaciones adicionales(a partir de Windows 10). Visitar la guia de docker para windows

Supondremos que nos encontramos en la carpeta raíz del proyecto /dmt .

Primero creamos una imagen con los requerimientos que necesitamos para el compilador.

Para ello debemos ejecutar el siguiente comando:

docker build -t tfgjasle -f .

Ejecutamos el fichero docker-compose.yaml con el siguiente comando para lanzar Cassandra:

docker-compose up -d

Luego creamos el contenedor a partir de dicha imagen , con la ruta local del volumen a donde se encuentra la carpeta /dmt que contiene el código:

docker run -itd --name main --network cassandra -v "C:\Users\jasle.carrasco\Desktop\TFG\docker\dmt:/dmt" tfgjasle

donde hay que cambiar la ruta primera del volumen por la que corresponda con la carpeta /dmt que contiene el código.

Comprobar que están levantados los nodos con el comando :

docker exec cassandra-1 nodetool status

Para detener los contenedores y eliminarlos:

docker-compose down(*)

(*)El uso de un volumen en Docker para la base de datos hace que vaya bastante más lento,y más al haber más de un nodo en el cluster. En el docker-compose, se encuentra comentado por esa razón. Sin embargo, si se quiere mantener los cambios introducidos en la base datos debe estar definido el volumen o se perderán los datos cuando se ejecuta el comando.

USO DE LA HERRAMIENTA

Se tienen tres funcionalidades:

- Creación del modelo a partir de ficheros ASN1
- Inserción de telemetrías y telecomandos con validación de los campos
- Creación de telecomandos

Para ser usada correctamente se debe tener en cuenta lo siguiente:

Se creará una tabla por módulo ASN1 definidos en los ficheros pasados como parámetro, con las tablas de nombre igual al nombre del módulo

Se asociará un fichero CSV por tabla , tanto para telemetrías como para telecomandos .

- **Diferencias en la ejecución entre Linux y Docker**

  La ejecución con Docker es prácticamente igual a la de linux , y puede hacerse de dos maneras muy similares:

1.) Conectándonos al contenedor que ejecuta la lógica y desde allí usar los mismos comandos que con linux.

docker exec -it main bash

Todo el contenido tendría que estar dentro del volumen claro. Es decir en mi caso al crear el contenedor definí así el volumen:

-v "C:\Users\jasle.carrasco\Desktop\TFG\docker\dmt:/dmt"

Por lo que tendría que tener eso en cuenta para las ficheros obtenidos y las rutas que hay que pasar como parámetro

2.) Desde fuera del contenedor a través del host , se ejecutarían los comandos copiando primero lo que tengamos que pasar al contenedor y a la inversa. Tiene la ventaja de que no tenemos la ruta limitada al volumen que hemos definido, y habría que definir un script intermedio para copiar los archivos.

A continuación se detalla cómo utilizar la herramienta y los resultados esperados para cada una de las tres funciones:

1) Creación del modelo

Debemos llamar al módulo asn2dataModel.py de la siguiente manera:

Path/To/asn2dataModel.py <params> <DirALLfiles> <selectedFile1 selectedFile2 …>

Donde los argumentos:

<DirALLfiles> : Es el directorio que contiene todos los ficheros ASN1 que necesitamos compilar.

<selectedFile1 selectedFile2 …> : Es una lista de ficheros ASN1 (que debe encontrar en el directorio del comando anterior) separados por espacios. Son los ficheros de los que realmente queremos construir el modelo. El resto son para campos importados.

Donde <params> :

-help : muestra instrucciones de como debe ser el comando

-modulesTelecommand: <moduleTelecommand1, moduleTelecommand2 …>.Indica que módulos de los que se encuentran en los ficheros pasados como argumento (selectedFile1, selectedFile2,..), van a ser para telecomandos. Es opcional , por defecto null

Debe estar contenido en un único parámetro , por lo que debe tener el formato(“module1,module2,..”)

-keyspace ; el keyspace donde crear las tablas.Si no existe lo crea

-*contact_points*: string que contiene separado por comas la dirección IP o el nombre identificativo de los nodos del cluster con los que nos queramos conectar(en el caso de Docker , es suficiente con el nombre del nodo principal)

-clusterPort: el puerto con el que conectar al cluster.

De modo que ejecutaría lo siguiente, estando situado dentro dentro de /dmt:

python3 ***/dmt/src/asn2dataModel.py*** -modulesTelecommand "***DataTypes-Telecommands"*** -keyspace ***tfg*** -contact_points ***cassandra*** -clusterPort ***9042*** ***./filesASN1*** ***DataTypes-Telecommands.asn DataTypes-Telemetries.asn***

Se crearía en el keyspace una tabla por cada módulo definido en los ficheros:

./filesASN1/DataTypes-Telecommands.asn ./filesASN1/DataTypes-Telemetries.asn

Y en la carpeta /templatesCSV , dentro de /dmt , crearía una plantilla con un csv por tabla con solamente las cabeceras. Tanto las telemetrías que se van a recibir , como los telecomandos que se van a enviar deben tener exactamente ese formato.

2.) Inserción de telemetrías/telecomandos

Debemos llamar al módulo readCSV.py de la siguiente manera:

Path/To/readCSV.py <DirfilesCSV> <params>

Donde los argumentos:

<DirFilesCSV> : Es el directorio que contiene los ficheros CSV , donde cada uno se corresponde con una tabla.

Donde <params> :

-help : muestra instrucciones de como debe ser el comando

-keyspace ; el keyspace donde crear las tablas.Si no existe lanza un error

-*contact_points*: string que contiene separado por comas la dirección IP o el nombre identificativo de los nodos del cluster con los que nos queramos conectar(en el caso de Docker , es suficiente con el nombre del nodo principal)

-clusterPort: el puerto con el que conectar al cluster.

-filesTelecommands <fileCSV1 fileCSV2 …>. Indica que ficheros de los que se encuentran en los ficheros pasados como argumento (DirfilesCSV), van a ser para telecomandos. Es opcional , por defecto Null

De modo que , por ejemplo ejecutaría lo siguiente, estando situado dentro dentro de /dmt:

python3 ***/dmt/src/ReadWriteTMTC/readCSV.py*** ***./filesCSV*** -keyspace ***tfg*** -contact_points ***cassandra*** -clusterPort ***9042*** -filesTelecommands ***datatypes_telecommands.csv***

Así se insertarían los datos a las tablas, donde se validarían los campos.

Hay una distinción entre telemetrías y telecomandos por dos razones:

- Si es una telemetría y hay un error en un valor que no corresponde con lo esperado , se lanza un warning con el campo , su valor y los valores en los que debería estar.Ya que es el contenido de la información recibida y debe almacenarse en cualquier caso. Si se trata de un telecomando debe impedir que se inserte en la base de datos esa fila , para que no se envíe, lanzando un error mostrando lo mismo que para las telemetrías.

- En el caso de los telecomandos debe distinguirse los que están pendientes de enviar de los enviados. Por ello aunque en los ficheros CSV no va a aparecer , en la tabla internamente hay un campo boolean que valdrá True para todos los pendientes de enviar. Cuando se insertan datos procedentes de telecomandos , es porque quiere validar que su contenido sea correcto , así como registrarlos en la base de datos.

3) Creación de telecomandos

Debemos llamar al módulo createCSV.py de la siguiente manera:

Path/To/createCSV.py <outputDir> <tables> <params>

Donde los argumentos:

<outputDir> : Es el directorio donde se van a crear los ficheros CSV , donde cada uno se corresponde con una tabla.Si no existe lo crea

<tables>: Son las tablas a partir de las cuales se generará un CSV para ser enviado como telecomando. Debe estar contenido en un único parámetro , por lo que debe tener el formato(“table1,table2,..”)

Donde <params> :

-help : muestra instrucciones de cómo debe ser el comando

-keyspace ; el keyspace donde crear las tablas.Si no existe lanza un error

-*contact_points*: string que contiene separado por comas la dirección IP o el nombre identificativo de los nodos del cluster con los que nos queramos conectar(en el caso de Docker , es suficiente con el nombre del nodo principal)

-clusterPort: el puerto con el que conectar al cluster.

-sendTelecommands: boolean Indica que las tablas pasadas como parámetro van a enviarse y por lo tanto además de generar un CSV cambiará su estado de pendientes de enviar a False en la BBDD. Es opcional , por defecto False

De modo que , por ejemplo ejecutaría lo siguiente, estando situado dentro dentro de /dmt:

python3 ***/dmt/src/ReadWriteTMTC/createCSV.py ./filesTelecommand “datatypes_telecommands”*** -keyspace ***tfg*** -contact_points ***cassandra*** -clusterPort ***9042*** -sendTelecommands ***True***

FORMATO DE LAS TABLAS:

Para que la herramienta sea general , no puede haber ambigüedad entre las columnas de las tablas que se definan en un modelo, ya que se aplanan al pasar a formato CSV.De modo que se va concatenando cada valor que está anidado hasta llegar al valor final añadiendo '_' .Esto es así porque los nombres de las tablas y las columnas en Cassandra solo pueden contener valores alfanuméricos y "_".

Para evitar nombres excesivamente largos pero que si permitan diferenciar un campo de otro en cualquier caso , cuando se llega al último valor de un nodo(básico o enumerado) , se crea un array donde cada elemento es el que está separado por la "_" . Se eliminan los repetidos manteniendo el orden.

En el caso de la tabla de validación también se crea aquí al crear el modelo ,solamente que en este caso no nos interesa tener una fila por campo , ya que muchos campos comparten un tipo común , que es el que determina la condición en la que debe insertarse un dato.

Por ello al crear las columnas , por cuestiones de eficiencia , se añade "__" justo antes del identificador del campo y tipo y esa es la parte que compararemos l insertar los datos.

También es obligatorio definir al menos una primary-key en un módulo , y solo puede hacer en un tipo básico (excepto boolean). También existe la opción de declarar la clustering key para ese módulo. Por ejemplo:

```
BASIC-A-PRIMARY-KEY ::= INTEGER (0 .. 255)

BASIC-B-CLUSTERING-KEY-ASC  ::= Epoch-Type

PRIMARY-KEY ::=  SEQUENCE {

    BASIC-C-PRIMARY-KEY  Epoch-Type,

     BASIC-D-CLUSTERING-KEY-DESC  Heater-Power-Type

}
```