# IBM Transformation Extender SWIFT MT-MX translation integrated with REST API application to track map usage deployed on minikube (Kubernetes)


| | |
|---|---|
| **Authors:** | Mehmet Cambaz |
| **Version:** | 1.0 |
| **Date Created:** | 01.10.2019 |
| **Last Updated:** | 28.11.2019 |

## Document Revision History

| Revision | Date | Description | Author |
|---|---|---|---|
| 1.0 | 01.10.2019 | First Draft | Mehmet Cambaz |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Table of Contents

# 1. Introduction

This document's purpose is to give helpful information on how to leverage IBM Transformation Extender's (ITX) capabilities and have an application deployed on kubernetes to track the usage of the maps so that it can be used to create department/partner payback model as central transformation engine of the business. (Since ITX v10 is dockerized it is possible to deploy it on Kubernetes too.) ITX is Red Hat Openshift certified along with IBM Sterling File Gateway, B2B Integrator, Connect:Direct.

ITX Financial Pack includes samples of MT to MX translations at the install folder "*INSTALLFOLDER*/packs/financial_payments_v9.0.3/swift/mx/examples/mt_mx_translation"

Check out the *readme.txt* for the sample map details at this folder.

The know-how shared on this document:

- Minikube installation, configuration
- Sample Express Node.js application as REST API server to save the ITX usage on MySQL database, generate a monthly usage report by partner whenever needed
- Dockerize the Sample Express Node.js application, deploy the containerized application to minikube (Kubernetes)
- How to call ITX sample map provided by ITX Financial Pack

Some useful links:

- Documentation of minikube
- Using local docker images in minikube
- Sample Express Nodejs REST API project
- MySQL Workbench to connect and manage MySQL
- Sample wordpress and mysql deployment

# 2. Content

## kubectl

Before installing minikube itself, you should install kubectl command line interface (CLI), I had installed minikube to macos via Home Brew: steps

```
brew install kubectl
```

after successful install you can start check kubectl below command:

```
kubectl version
```

**Sample output**

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"14", GitVersion:"v1.14.7",
GitCommit:"8fca2ec50a6133511b771a11559e24191b1aa2b4", GitTreeState:"clean", BuildDate:"2019-09-18T14:47:22Z",
GoVersion:"go1.12.9", Compiler:"gc", Platform:"darwin/amd64"}
```

## minikube

I had installed minikube to macos via Home Brew and used Vmware Fusion. Minikube installation steps:

```
brew install minikube
```

after successful install you can start minikube via below command:

```
minikube start --vm-driver=vmware
```

you can set vm-driver as vmware as always on config via command below and can start with only `minikube start` command after that

```
minikube config set vm-driver vmware
```

**Sample output**

```
minikube start
😄  minikube v1.5.2 on Darwin 10.15
💡  Tip: Use 'minikube start -p <name>' to create a new cluster, or 'minikube delete' to delete this one.
🔄  Starting existing vmware VM for "minikube" ...
⌛  Waiting for the host to be provisioned ...
🐳  Preparing Kubernetes v1.16.2 on Docker '18.09.9' ...
🔄  Relaunching Kubernetes using kubeadm ...
⌛  Waiting for: apiserver
🏄  Done! kubectl is now configured to use "minikube"
⚠️  /usr/local/bin/kubectl is version 1.14.7, and is incompatible with Kubernetes 1.16.2. You will need to update
/usr/local/bin/kubectl or use 'minikube kubectl' to connect with this cluster
```

You can launch the dashboard via below command:

```
minikube dashboard
```

**Sample output**

```
minikube dashboard
🤔  Verifying dashboard health ...
🚀  Launching proxy ...
🤔  Verifying proxy health ...
🎉  Opening http://127.0.0.1:57377/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-
dashboard:/proxy/ in your default browser...
```

It opens http://127.0.0.1:57377/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/#/overview?namespace=default

You can check out the IP for your minikube via command: `minikube ip`

```
minikube ip
192.168.106.133
```

## Deploy MySQL to minikube

You can deploy a MySQL instance to Kubernetes via yaml configuration files easily.

1. Create persistent volume claim: (save below content as file named "persistentVolumeClaim.yaml")

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-data-disk
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Execute the command at terminal `kubectl apply -f persistentVolumeClaim.yaml`

***Check the entity:***

```
kubectl get pvc
NAME              STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS
mysql-data-disk   Bound    pvc-c1e9d76f-27af-4431-bc37-abb3f85528e2   1Gi        RWO            standard
```

2. Create secrets (password): (save below content as file named "secrets.yaml")

```
---
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secrets
type: Opaque
data:
  ROOT_PASSWORD: bXktc3VwZXItc2VjcmV0LXBhc3N3b3JkCg==
```

Execute the command at terminal `kubectl apply -f secrets.yaml`

***Check the entity:***

```
kubectl get secrets
NAME                  TYPE                                  DATA
default-token-f29r8   kubernetes.io/service-account-token   3
ingess-tls-secret     Opaque                                3
mysql-secrets         Opaque                                4
```

*(Password is base64 for my-super-secret-password)*

```
Mehmets-MacBook-Pro:itx-mysql-restapi mehmetcambaz$ echo my-super-secret-password | base64
bXktc3VwZXItc2VjcmV0LXBhc3N3b3JkCg==
```

3. Create deployment: (save below content as file named "deployment.yaml")

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
  labels:
    app: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:5.7
```

```
        ports:
          - containerPort: 3306
        volumeMounts:
          - mountPath: "/var/lib/mysql"
            subPath: "mysql"
            name: mysql-data
        env:
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysql-secrets
                key: ROOT_PASSWORD
    volumes:
      - name: mysql-data
        persistentVolumeClaim:
          claimName: mysql-data-disk
```

Execute the command at terminal `kubectl apply -f deployment.yaml`

***Check the entity:***

```
kubectl get pods
mysql-deployment-5b68bb45bc-dt44b   1/1     Running
```

4. Expose deployment as service:

Execute the command at terminal

```
kubectl expose deployment/mysql-deployment --type="NodePort" --port 3306
```

***Check the entity:***

```
kubectl describe service mysql-service
Name:                     mysql-service
Namespace:                default
Labels:                   <none>
Annotations:              kubectl.kubernetes.io/last-applied-configuration:
                            {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"name"
:"mysql-service","namespace":"default"},"spec":{"ports":[{"port":33...
Selector:                 app=mysql
Type:                     NodePort
IP:                       10.103.236.137
Port:                     <unset>  3306/TCP
TargetPort:               3306/TCP
NodePort:                 <unset>  30062/TCP
Endpoints:                172.17.0.2:3306
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

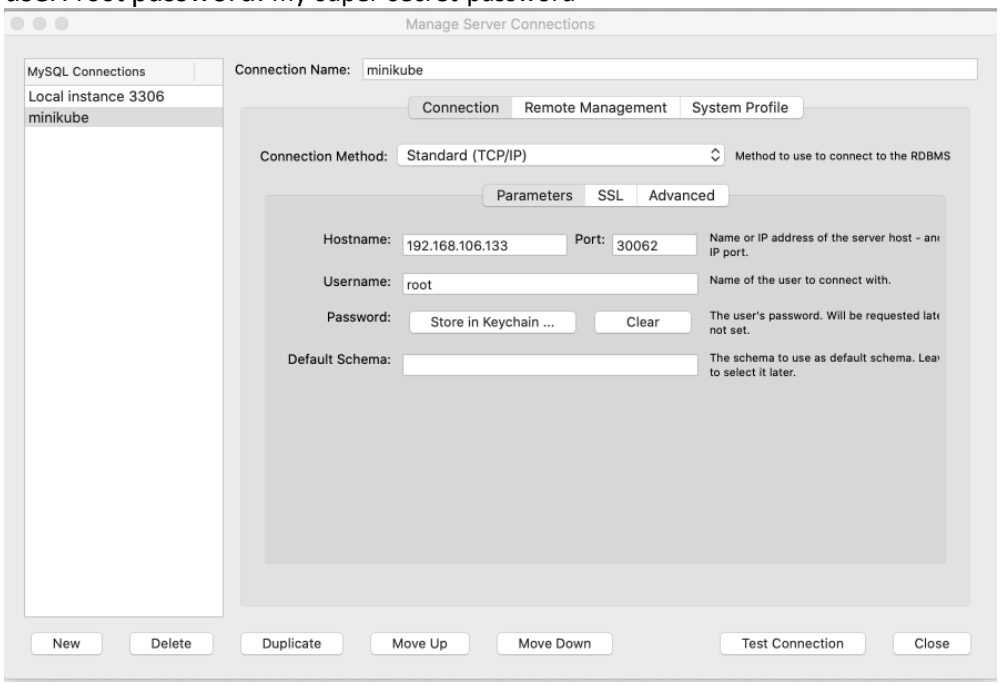
## Node.js application

1. Download the code from github here (*Disclaimer: this is not a production ready code, it is just a proof of the concept*)

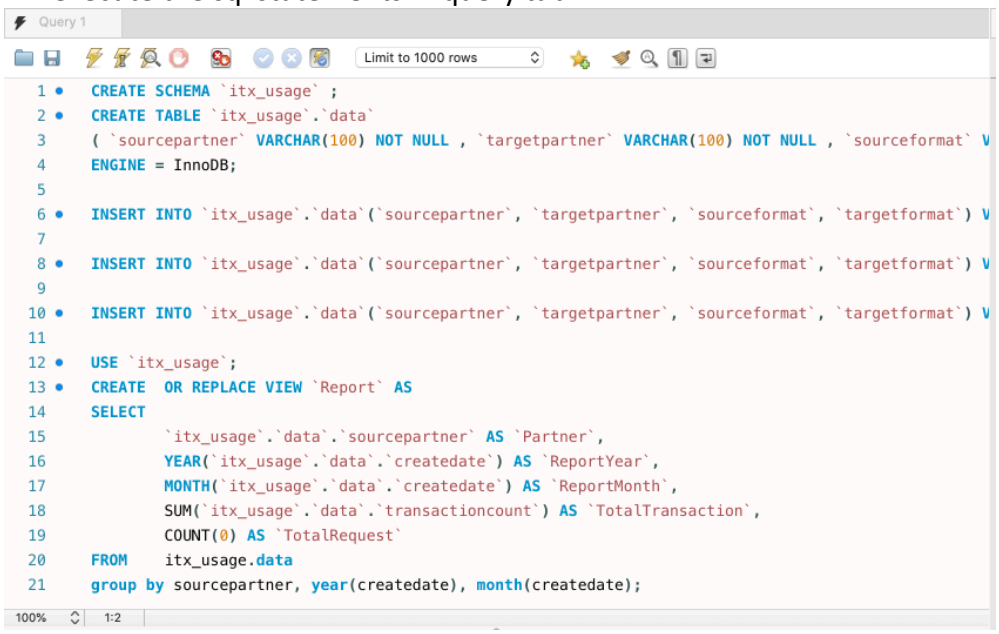
2. Now the mysql service is accessable via minikube IP (mine was `192.168.106.133`) and nodeport (we set as `30062`) provided at command `kubectl describe service mysql-service`

Connect to MySQL via MySQL Workbench

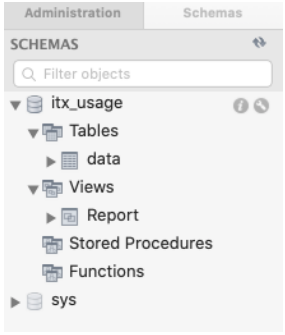
user: root password: my-super-secret-password



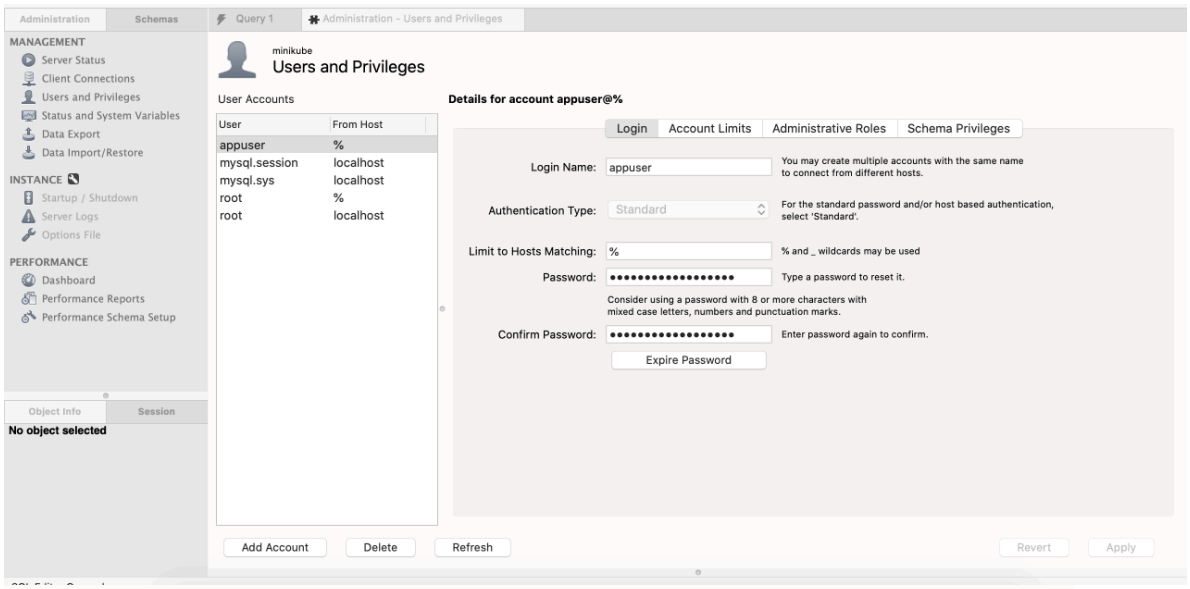

3. After connecting to the instance we will create all needed database objects described in db.sql file here https://github.com/cambazm/NodeExpressJsRestApiToMysql/blob/master/db.sql , execute the sql statements in query tab:

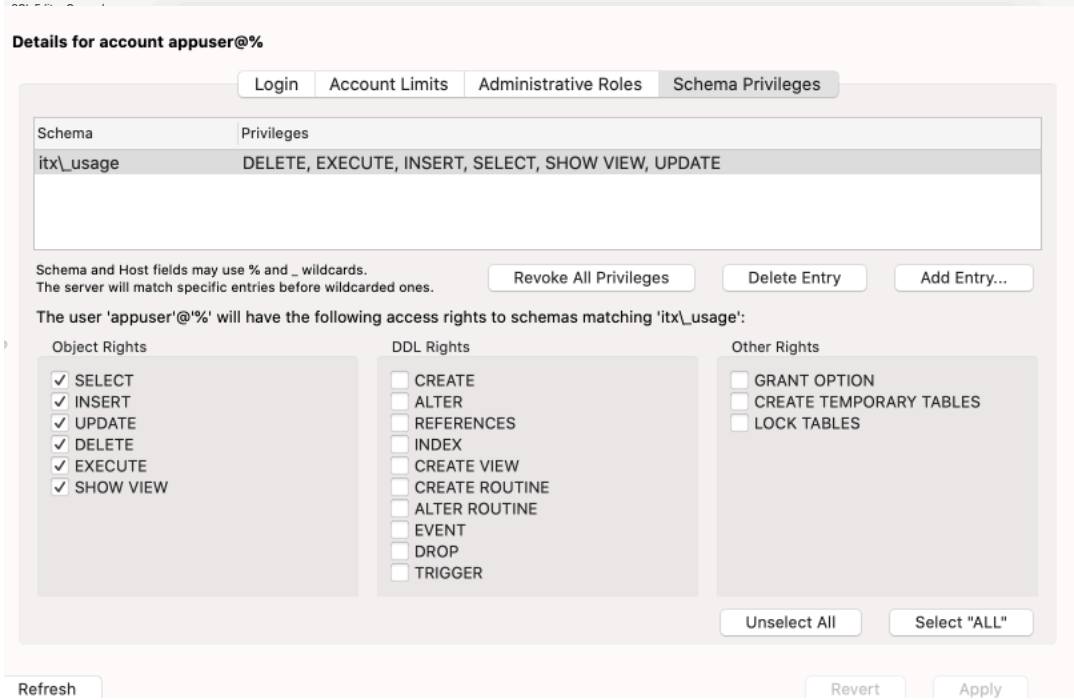


Objects are created:




4. Create an application user called `appuser` to use in the application with less privilege (Administration->Management->Users and Privileges->Add Account



(It is also possible to create the user and schema etc. via initdb sql scripts through Kubernetes ConfigMap yaml execution similar to below sample)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
  labels:
    app: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:5.6
          args: ["--default-authentication-plugin=mysql_native_password", "--ignore-db-
dir=lost+found"]
          ports:
            - containerPort: 3306
          volumeMounts:
            - mountPath: "/var/lib/mysql"
              subPath: "mysql"
              name: mysql-data
          env:
               - name: MYSQL_ROOT_PASSWORD
                 value: "my-super-secret-password"
      volumes:
        - name: mysql-data
          persistentVolumeClaim:
            claimName: mysql-data-disk
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: initdb
data:
  initdb.sql: |-
    CREATE USER 'appuser'@'%' IDENTIFIED BY 'my-super-secret-password' ;
```

5. Enter to the folder of the code downloaded and install node, express, mysql framework and dependencies

Configure the db.js file with your MySQL instance info, sample of my instance:

```
var connection = mysql.createConnection({
    host     : '192.168.106.133',
    port     : 30062,
    user     : 'appuser',
    password : 'my-super-secret-password',
    database : 'itx_usage'
});
```

(sometimes you can have issues related to ex-node versions if you had installed you can delete and reinstall via

```
sudo rm -rf /usr/local/lib/node_modules/npm
brew reinstall node
```
)

```
brew install node
node -v
npm install
```

Start the application for local testing:
```
node server.js
```

You can test via opening http://localhost:3000/data and http://localhost:3000/report You can stop the node via ctrl+z.

Sometimes 3000 port keeps blocked, you can release it via command: `kill -9 $(lsof -t -i:3000)`

If you have challenges installing used packages, you can use below commands
```
npm install -g npm
npm install express
npm install mysql
```

Project TODOs

- Move db.js config information to Kubernetes secrets
- Make it https
- Add authentication mechanism
- Configure for load balancer for multiple pods
- Push the docker image to a registry
- Display the report via HTML5 UI, add filtering capabilities
- Create jenkins project for different environments and configs
- Add indexes to table, add rebuild job of indexes daily
- Create an alternate sample project Java micro service with Eclipse MicroProfile

## Dockerize Node.js application

1. Create a file called **Dockerfile** in the node.js application folder (mine was itx-mysql-restapi) and copy below content into it:

```
FROM node:6.9.2
WORKDIR /usr/src/itx-mysql-restapi
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD [ "node", "server.js" ]
```

2. Check if you can reach docker command via terminal, if it is not installed you can install Docker Desktop easily to add to your environment

```
Mehmets-MacBook-Pro:expressjs mehmetcambaz$ docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config
```

3. Execute below command, it will set your minikube environment to see local docker images:

```
eval $(minikube docker-env)
```

4. Execute below command, it will build your docker image with the Dockerfile contents you had created

```
docker build -t itx-mysql-restapi:v2 .
```

5. Control your docker image you had built

```
docker images itx-mysql-restapi
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
itx-mysql-restapi   v2                  bfacf2652964        3 days ago          662MB
```

6. Execute below command, it will run your docker image you had built

```
kubectl run itx-mysql-restapi --image=itx-mysql-restapi:v2 --port=3000 --
image-pull-policy=Never
```

7. Execute below command, it will expose your docker deployment you had deployed on minikube

```
kubectl expose deployment itx-mysql-restapi --type=NodePort --port=3000 --
target-port=3000
```

8. Execute below command to get more information about the exposed service deployed

```
kubectl describe service itx-mysql-restapi
```

**Sample output:**

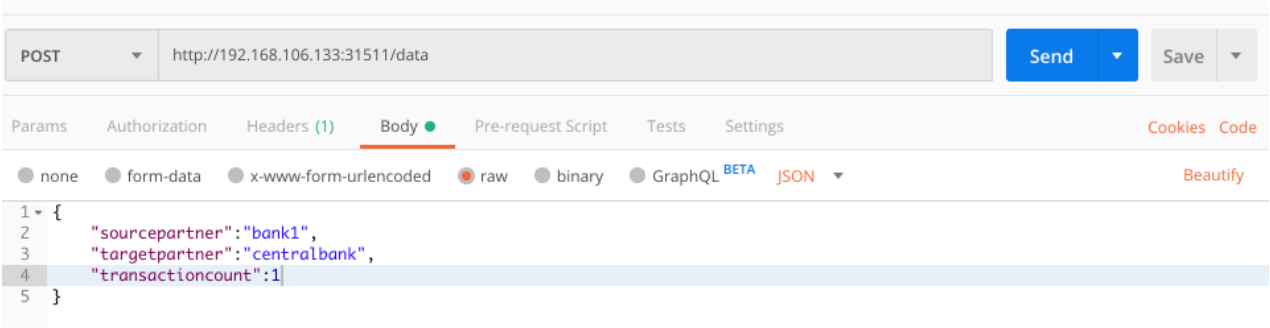
```
Name:                     itx-mysql-restapi
Namespace:                default
Labels:                   run=itx-mysql-restapi
Annotations:              <none>
Selector:                 run=itx-mysql-restapi
Type:                     NodePort
IP:                       10.99.214.15
Port:                     <unset>  3000/TCP
TargetPort:               3000/TCP
NodePort:                 <unset>  31511/TCP
Endpoints:                172.17.0.9:3000
Session Affinity:         None
External Traffic Policy:  Cluster
```

9. My minikube IP was 192.168.106.133
http://192.168.106.133:31511/report shows the report (http get)
http://192.168.106.133:31511/data lists all data (http get)

POST to http://192.168.106.133:31511/data sample json below to insert data to database

```
{
	"sourcepartner":"bank1",
	"targetpartner":"centralbank",
	"transactioncount":1
}
```

10. You can test POST command via Postman application, copy paste the json sample above, set html body to raw, and set Content-Type to application/json




## Add HTTP POST to Sample MT-MX map in ITX

Select one of the sample maps provided with ITX Financial Pack. For example: "mt202_COV_mx_pacs_009.mms → converts MT202COV to MX" MT-MX message information

Add a new type-tree including sourcepartner, targetpartner, transactioncount

Add a new output card to do HTTP POST to the Node.js application created (this could be done via couple of ways

1- REST Client capability which planned to be available ITX v10.0.0.2
2- Call HTTP POST at the RULE via sink output card:

```
POST ( "HTTP",  "-URL http://192.168.106.133:31511/data/" +
    Build application json
+  " -TRACE" )
```

3- Call a command line script does a curl to the REST API, since this is better to do async it might be better)

ITX can be called directly via REST APIs to execute the maps in system. This provides a great way when modern applications interact with ITX.

## 3. Summary

We had installed, configured kubernetes at our local computer. Created a simple REST API application which pushes usage data to mysql, created a simple reporting view for chargeback discussions. We deployed our application to Kubernetes and saw how easy to deploy with yaml descriptive files. We also leveraged the sample MT-MX translation maps provided by IBM Transformation Extender Financial Pack on complex transformation from legacy MT to XML based-MX.

*Special thanks to JP Morgan and Rex Chan from HCL whom assisted me.*