**MifosLab Installation/Configuration Walkthrough**

**Mojaloop**

Mojaloop is run in a Kubernetes environment provided by Azure. Create an instance of Kubernetes service in your Azure portal. We are running 2 nodes instead of the default of 3. Mojaloop will not run correctly with a single node. We are creating a resource group called Mojaloop and the Kubernetes cluster is called MojaloopCluster.

After creating the Kubernetes service, we need to run a couple of commands in the Azure Cloud shell to set up RBAC for the cluster:

az ad sp create-for-rbac --skip-assignment

az aks get-credentials --resource-group Mojaloop --name MojaloopCluster

We used the Mojaloop deployment guide (<https://mojaloop.io/documentation/deployment-guide/>) to create the Mojaloop cluster. In section 4 (Helm configuration), we need to run a couple of extra commands before installing the Ingress (step 6). In cloud shell, run the following:

kubectl create serviceaccount --namespace kube-system tiller

kubectl create clusterrolebinding tiller-cluster-rule --clusterrole=cluster-admin --serviceaccount=kube-system:tiller

kubectl patch deploy --namespace kube-system tiller-deploy -p '{"spec":{"template":{"spec":{"serviceAccount":"tiller"}}}}'

After these commands have been run, you can install the ingress as directed.

In order to view logs in the Kubernetes containers, you will need to authorize access. Create a yaml file with the following contents:

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

name: containerHealth-log-reader

rules:

- apiGroups: [""]

resources: ["pods/log", "events"]

verbs: ["get", "list"]

---

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: containerHealth-read-logs-global

roleRef:

kind: ClusterRole

name: containerHealth-log-reader

apiGroup: rbac.authorization.k8s.io

subjects:

- kind: User

name: clusterUser

apiGroup: rbac.authorization.k8s.io

Save this yaml file as LogReaderRBAC.yaml and upload the file in the Azure Cloud Shell. From the shell run the following command:

kubectl create -f LogReaderRBAC.yaml

**To upgrade Mojaloop/Helm version**

If Mojaloop has been installed, the following steps will allow you to upgrade to the latest version and components (or to select a specific version to install). Run the following in cloud shell:

helm del --purge moja;

helm repo update

helm search -l mojaloop/mojaloop (will list Mojaloop versions)

helm --namespace demo --name moja install mojaloop/mojaloop --version <version>

**Installing Fineract**

Create VM on Azure – be sure to add static IP. I also created DNS entries for the instance

Clone Fineract repo to get latest code and build.

Update SQL migration scripts to create tenants (tn01 and tn02) – the file is:

fineract-provider/src/main/resources/sql/migrations/list\_db/V1\_\_mifos-platform-shared-tenants.sql

Update database initialization script to create tenant databases (tn01, tn02). The file is:

fineract-db/docker/01-databases.sql

Get Dockerfile and docker-compose.yml from repo – overwrite existing versions

docker-compose build

docker-compose up -d

Note: Make sure that network ports are open on VM – 8443

* Ports 80, 48888, 48889, 58080, 58180, 58181 also need to be open for payment hub
* Ports 62020 and 62021 need to be open for openbanking adapter

Note: To use a CA certificate in the tomcat keystore, you will need to compile it to a .p12 file. Use the following command (use the same password as you use for the keystore):

openssl pkcs12 -export -in mifosio.crt -inkey mifosio.key -out mifosio.p12 -name mifos -CAfile mifosio.cabundle -caname root -chain

**Install Fineract CN**

Copy the deployment scripts from the fincn directory onto the local machine. In the env.sh script set DEP\_PROVISION to true for the first run. After that, you can set it to false.

**Install Payment Hub**

Clone Payment Hub repo (<https://github.com/openMF/payment-hub>)

Get Dockerfile and docker-compose from repo

NOTE: The following section is no longer needed since we have valid certs. But if you are setting up in your own environment with self-signed certs, you will need to do this.

Configure payment hub to allow SSL connection to fineract instances:

1. cd work
2. javac InstallCert.java
3. java InstallCert fineract.mifoslab.org:8443 (or whatever is the appropriate hostname for the first FSP)
4. java InstallCert fincn.mifoslab.org:8443 (or the hostname for the second FSP)

When the docker-compose up command is run, it will copy the cert files to the correct place in the container.

Get application.yml file from repo – copy into /work directory

* Update configuration files point to the Fineract instances and set the correct tenant names

docker-compose build

docker-compose up -d

**Install Account Oracle**

We have created a simple utility to do mock pathfinder lookups for MSISDN. Pull the code from the repository (<https://github.com/openMF/mock-als-oracle>) and install into a separate directory on your server for account lookups:

docker-compose build

docker-compose up -d

\*\* Make sure that any MSISDN accounts that you will be using in your payment flows are registered in the Account Oracle. The postman scripts (below) can be used to do that.

**Configure Accounts in Fineract**

To create accounts, there are several sql scripts that need to be run – this will configure savings products, user accounts, IBAN and MSISDN identifiers. These scripts are in the sample\_data directory in the Fineract repository. They have been customized for the Mifos lab environment and can be found in the mifos-io-configuration repository.

For each Fineract tenant, connect to the correct tenant database and run the interop\_sample\_data.sql script. Then run the sample data script for that tenant. For example, in01tn01 (database tn01) will use tn01\_interop\_sample\_data.sql.

After running the tnXX\_interop\_sample\_data script, select from the m\_clients table and write down the client\_id for the interop client that was just created. This will be needed later for the openbanking configuration.

**Run Postman Scripts**

Get postman scripts and environment from repo:

MifosLab.postman\_collection.json

MifosLab.postman\_environment.json

Run all scripts – environment variables should all be set correctly. The only environment variables that will need to be changed are the settlement account numbers. When you create each DFSP in Mojaloop, it will return a settlement account number. Update the environment variable for each DFSP to use the newly created account number.

The script sections that should be run are:

Initialization

DFSP Onboarding

Create Participants

\*\* Note that the central-ledger callback endpoints will not be overwritten if you send multiple requests – they will simply add new entries. If you have an incorrect callback, you will have to manually delete them in the database. This issue has been reported to Mojaloop and will hopefully be resolved soon \*\*

**Test Installation with Postman**

Once all of the onboarding has been done, you can test a transfer of funds from one account to another. In the ‘Test Transfer’ section of the postman queries, there is a transfer request through the payment hub. Currently it is configured to send $100 from in01tn01 user to in01tn02 user. You can edit the body of this transfer to reflect your test case.

After running the query, it will respond with a transaction ID. Copy this transaction ID and append it to the query transaction (the next request in the postman collection). When you run this request, you should see that the transfer has been completed (COMMITTED) – assuming that there were sufficient funds in the sender’s account.

**OpenBanking WSO2 Configuration**

This installation is based on this documentation: <https://mifos.gitbook.io/docs/>

Create a new virtual machine for each dfsp (our environment is currently configured with api.lion.mifos.io and api.elephant.mifos.io). Make sure that docker and docker-compose are installed on the VMs.

Install the WSO2 adapter and configure according to these instructions: <https://mifos.gitbook.io/docs/wso2/configure-ec2-instance-for-wso2-apigateway/wso2-apigateway-install-from-docker-compose>

Note: the URL for our WSO2 Gateway instance is <http://api.lion.mifos.io:9443> or <http://api.elephant.mifos.io>

Note: I had to increase the healthcheck timeouts in the docker-compose.yml file – otherwise it would time out and indicate that the container is unhealthy

In Azure, ensure that ports 9443, 8280, 8243, 9763 are open in network security manager

The Swagger definitions for the openbanking API’s can be located here:

<https://raw.githubusercontent.com/OpenBankingUK/read-write-api-specs/v3.1.2-RC1/dist/account-info-swagger.yaml>

<https://raw.githubusercontent.com/OpenBankingUK/read-write-api-specs/v3.1.2-RC1/dist/payment-initiation-swagger.yaml>

They are also in the mifos-io-configuration repository.

The production endpoints for these tenants will be:

lion.mifos.io:62021/adapter/ob

elephant.mifos.io:62021/adapter/ob

Register the ACE Fintech user (TPP) on api.lion and api.elephant and LionFintech user (FPP) on api.lion. Create user accounts in api.lion.mifos.io:9443/store and api.elephant.mifos.io:9443/store

Username: acefintech Password: AceFintech1

Username: lionfintech Password: LionFintech1

Create applications for ACE Fintech and Lion Fintech and subscribe to both the accounts API and the subscription API. Register the callback for authorization as:

http://acefintech.mifos.io/netbank/customer/banks/authorize

Generate OAuth keys for each subscription – keep track of the key and secret for each app – they will be needed for configuring the TPP and FPP applications

Now, configure the users for each Fineract tenant in WSO2.

Username: buffalouser1 Password: BuffaloUser1

Username: lionuser1 Password: LionUser1

Username: rhinouser1 Password: RhinoUser1

Username: elephantuser1 Password: ElephantUser1

Have the lion and elephant tenants subscribe to the Accounts and Payments APIs.

**Configure API Gateway Authentication**

On the api.lion.mifos.io and api.elephant.mifos.io virtual machines, clone the repository from <https://github.com/openMF/openbanking-api-gateway>

Edit src/main/webapp/WEB-INF/web.xml and set openbanking.logic.url parameter to <http://lion.mifos.io/accessschema/ob> (or elephant)

Build the gateway with the command: mvn clean package

Copy the war file using the command: docker cp authenticationendpoint.war <WSO2 container>:/home/wso2carbon/wso2am-2.6.0/repository/deployment/server/webapps

It will pick up the new war file and re-deploy it automatically.

**Configure Openbanking adapter**

On each Fineract instance, clone the open banking adapter repository: <https://github.com/openMF/openbanking-adapter.git>

Configure the work/application.yml file for the appropriate Fineract tenants

Configure mysql databases for each instance – populate the url in the application.yml file

From the sources directory, build with this command: mvn clean package

Create the tenant databases in mysql. On Lion, these will be ob\_tn01 and ob\_tn02 and on Elephant, it will be ob\_tn03 and ob\_tn04

Build the docker container using: docker-compose build

Navigate to the fineract directory – we have added the openbanking adapter to the fineract launch (since both use the mysql database):

Docker-compose up -d

Now, create the tables and users for the openbanking adaptor. Navigate to the **src/main/resources/db/changelog/sql** directory. Edit the 01-ob\_tn01\_init.sql file. In the first line, ensure that the psp\_user\_id matches the client id for the Fineract client ID that was created with the interop\_sample\_data scripts. Also ensure that the api\_user\_id matches the user account that was created in WSO2 for each banking instance.

Repeat this process for all tenants.

**Configure Third-party and First Party apps**

On acefintech.mifos.io, clone the backend and frontend repositories for the third party banking provider app:

<https://github.com/openMF/openbanking-netbank-server.git>

<https://github.com/openMF/openbanking-netbank-client.git>

Ensure that ports 8080, 8081, and 8085 are open

