**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

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**

The account setup has not yet been automated, so you will need to create a client in each Fineract instance and then assign an account (either savings or loan account) to that client. The external account ID for the account must match the values that are in the Postman environment. The list of external account ids for each user (one per tenant) is:

In01tn01 user – 12345

In01tn02 user - 54321

In02tn03 user – 23456

In02tn04 user - 65432

Be sure to add funds to the payer accounts.

**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.