# Starting Okapi

Go to okapi root then start it

java -Dstorage=postgres -jar okapi-core/okapi-core/target/okapi-core-fat.jar dev

Or run that to increase number of ports

java -jar -Dstorage=postgres -Dport\_start=9131 -Dport\_end=9144 okapi-core/target/okapi-core-fat.jar dev

In the end you should have Okapi running in a console window, and listening on port 9130.

# Creating tenants

Let's create two test tenants for modules that will added later

## Create first tenant

cat > /tmp/okapi-tenant1.json <<END

{

  "id": "tenant1",

"name": "first tenant",

  "description": "our first tenant"

}

END

curl -w '\n' -X POST -D - \

  -H "Content-type: application/json" \

  -d @/tmp/okapi-tenant1.json \

  http://localhost:9130/\_/proxy/tenants

HTTP/1.1 201 Created

Content-Type: application/json

Location: /\_/proxy/tenants/tenant1

X-Okapi-Trace: POST okapi-2.32.0-SNAPSHOT /\_/proxy/tenants : 201 8204us

content-length: 87

{

"id" : "tenant1",

"name" : "first tenant",

"description" : "our first tenant"

}

## Create second tenant

The same steps as created first tenant

cat > /tmp/okapi-tenant2.json <<END

{

  "id": "tenant2",

  "name": "second tenant",

  "description": "our second tenant"

}

END

curl -w '\n' -X POST -D - \

  -H "Content-type: application/json" \

  -d @/tmp/okapi-tenant2.json \

  http://localhost:9130/\_/proxy/tenants

# Required modules

## Order of operations

Basically we just need to declare, deploy, and enable the modules.

Declaring and deploying the modules can be done in any order, but we have to be careful with the order of enabling them. Specifically, we may not enable mod-authtoken until the very end, when we have all the other modules in place and loaded with data, as mod-authtoken will not let us finish the process, locking us out of our own system.

# mod-permissions

## Introduction

This module stores permissions and associations between permissions and users. It also maintains a hierarchy of permissions and sub-permissions, allowing for permissions to act as roles, rather than simple bits. It is used primarily by the Authtoken module, though it is possible that some Authentication implementations may have reason to make calls to the Permissions module as well.

The following commands fetch and compile module:

git clone --recursive https://github.com/folio-org/mod-permissions

cd mod-permissions

git checkout master

mvn clean install

The install rule also runs tests. Tests should not fail. If they do then please report it, and in the meantime fall back to:

mvn install –DskipTests

## After building and run module by okapi without docker by default its use docker, in Module Descriptor change the Launch Descriptor from launch by docker to launch by exec command for okapi.

## Declaring the mod-permissions module

The module descriptors are generated from a template during the build process.

We just need to POST module to Okapi:

curl -w '\n' -D - -X POST  \

-H "Content-type: application/json" \

  -d @mod-permissions/target/ModuleDescriptor.json \

  http://localhost:9130/\_/proxy/modules

You can verify we have all modules properly declared:

curl -w '\n' -D -  http://localhost:9130/\_/proxy/modules

This should list mod-permissions module you declare and Okapi's internal module.

## Deploying the mod-permissions module

Now we need to deploy mod-permissions module. Here we simply use the default deployment descriptor that is provided by each module for its development purposes, so there is some tweaking required.

cat mod-permissions/target/DeploymentDescriptor.json \

  | sed 's/..\///' | sed 's/embed\_postgres=true//' > /tmp/deploy-permissions.json

curl -w '\n' -D - -X POST  \

  -H "Content-type: application/json" \

  -d @/tmp/deploy-permissions.json \

  http://localhost:9130/\_/discovery/modules

You can see all modules deployed with:

curl -w '\n' -D - http://localhost:9130/\_/discovery/modules

## Enabling the mod-permissions module for our tenants

Enable module for our tenants. We do not need to specify the version number here, Okapi will choose the latest (and only) version we have declared.

curl -w '\n' -D - -X POST  \

  -H "Content-type: application/json" \

   -d'{"id":"mod-permissions"}' \

http://localhost:9130/\_/proxy/tenants/<tenant\_id>/modules

# mod-users

## Introduction

Module to provide central user management for FOLIO systems.

Fetch and compile module:

git clone --recursive https://github.com/folio-org/mod-users

cd mod-users

git checkout master

mvn clean install

## Declaring the mod-users module

POST module to Okapi:

curl -w '\n' -D - -X POST  \

-H "Content-type: application/json" \

  -d @mod-users/target/ModuleDescriptor.json \

  http://localhost:9130/\_/proxy/modules

## Deploying the mod-users module

cat mod-users/target/DeploymentDescriptor.json \

  | sed 's/..\///' | sed 's/embed\_postgres=true//' > /tmp/deploy-users.json

curl -w '\n' -D - -X POST  \

  -H "Content-type: application/json" \

  -d @/tmp/deploy-users.json \

  http://localhost:9130/\_/discovery/modules

## Enabling the mod-users module for our tenants

curl -w '\n' -D - -X POST  \

  -H "Content-type: application/json" \

   -d'{"id":"mod-users"}' \

<http://localhost:9130/_/proxy/tenants/><tenant\_id>/modules

# mod-login

## Introduction

This module is responsible for verifying the user's identity and issuing a valid JWT that can be used for system access. The implementation of this module may vary (username/password, SAML, OAuth, etc.), and it is possible for more than one Authentication module to exist in a running system. The default implementation uses a simple username and password for authentication.

Fetch and compile module:

git clone --recursive https://github.com/folio-org/mod-login

cd mod-login

git checkout master

mvn clean install

## Declaring the mod-login module

POST module to Okapi:

curl -w '\n' -D - -X POST  \

-H "Content-type: application/json" \

  -d @mod-login/target/ModuleDescriptor.json \

  http://localhost:9130/\_/proxy/modules

## Deploying the mod-login module

cat mod-login/target/DeploymentDescriptor.json \

  | sed 's/..\///' | sed 's/embed\_postgres=true//' > /tmp/deploy-login.json

curl -w '\n' -D - -X POST  \

  -H "Content-type: application/json" \

  -d @/tmp/deploy-login.json \

  http://localhost:9130/\_/discovery/modules

## Enabling the mod-login module for our tenants

curl -w '\n' -D - -X POST  \

  -H "Content-type: application/json" \

   -d'{"id":"mod-login"}' \

<http://localhost:9130/_/proxy/tenants/><tenant\_id>/modules

# mod-authtoken

## Introduction

This module is responsible for filtering all proxy traffic and checking for a valid token. In addition, it is responsible for retrieving the permissions for a given user and making decisions regarding access based on user permissions and defined requirements for a given path. It provides a token creation endpoint that privileged modules (such as Authentication) may make use of.

Fetch and compile module:

git clone --recursive https://github.com/folio-org/mod-authtoken

cd mod-authtoken

git checkout master

mvn clean install

## Declaring the mod-authtoken module

POST module to Okapi:

curl -w '\n' -D - -X POST  \

-H "Content-type: application/json" \

  -d @mod-authtoken/target/ModuleDescriptor.json \

  http://localhost:9130/\_/proxy/modules

## Deploying the mod-authtoken module

cat mod-authtoken/target/DeploymentDescriptor.json \

  | sed 's/..\///' | sed 's/embed\_postgres=true//' > /tmp/deploy-authtoken.json

curl -w '\n' -D - -X POST  \

  -H "Content-type: application/json" \

  -d @/tmp/deploy-authtoken.json \

  http://localhost:9130/\_/discovery/modules

## Enabling the mod-authtoken module for our tenants

curl -w '\n' -D - -X POST  \

  -H "Content-type: application/json" \

   -d'{"id":"mod-authtoken"}' \

<http://localhost:9130/_/proxy/tenants/><tenant\_id>/modules

# mod-circulation-storage

## Introduction

FOLIO compatible persistent storage of loans, loan policies, circulation rules, requests, and fixed due date schedules.

Fetch and compile module:

git clone --recursive https://github.com/folio-org/mod-circulation-storage

cd mod-circulation-storage

git checkout master

mvn clean install

## Declaring the mod-circulation-storage module

POST module to Okapi:

curl -w '\n' -D - -X POST  \

-H "Content-type: application/json" \

  -d @mod-circulation-storage/target/ModuleDescriptor.json \

  http://localhost:9130/\_/proxy/modules

## Deploying the mod-circulation-storage module

cat mod-authtoken/target/DeploymentDescriptor.json \

  | sed 's/..\///' | sed 's/embed\_postgres=true//' > /tmp/deploy-circulation-storage.json

curl -w '\n' -D - -X POST  \

  -H "Content-type: application/json" \

  -d @/tmp/deploy-circulation-storage.json \

  http://localhost:9130/\_/discovery/modules

## Enabling the mod-circulation-storage module for our tenants

curl -w '\n' -D - -X POST  \

  -H "Content-type: application/json" \

   -d'{"id":"mod-circulation-storage "}'

http://localhost:9130/\_/proxy/tenants/<tenant\_id>/modules

# mod-data-import

## Introduction

FOLIO data import module.

To use this module we will need some module first:

* Mod-source-record-storage
* Mod-source-record-manager

Fetch and compile module:

git clone --recursive https://github.com/folio-org/mod-data-import

cd mod-data-import

git checkout master

mvn clean install

## Declaring the mod-data-import module

POST module to Okapi:

curl -w '\n' -D - -X POST  \

-H "Content-type: application/json" \

  -d @mod-data-import/target/ModuleDescriptor.json \

  http://localhost:9130/\_/proxy/modules

## Deploying the mod-data-import module

cat mod-data-import/target/DeploymentDescriptor.json \

  | sed 's/..\///' | sed 's/embed\_postgres=true//' > /tmp/deploy-data-import.json

curl -w '\n' -D - -X POST  \

  -H "Content-type: application/json" \

  -d @/tmp/deploy-data-import.json \

  http://localhost:9130/\_/discovery/modules

## Enabling the mod-data-import module for our tenants

curl -w '\n' -D - -X POST  \

  -H "Content-type: application/json" \

   -d'{"id":"mod-data-import"}'

http://localhost:9130/\_/proxy/tenants/<tenant\_id>/modules

# mod-agreements

## Introduction

Mod-Agreements is a FOLIO module to manage agreements which control an institutions access to resources. Normally, this will be subscribed access to packages of electronic resources, although the aim is to make the affordances offered by mod-agreements as general as possible.

Mod-Agreements can create agreements that control access to content bundled into packages and defined in knowledgebase systems, it can identify specific electronic or print resources, and act as a bridge between those resources and associated licenses and purchase documents.

## Compiling and Running

The build requirements are:

* Apache Groovy.
* Java 8 JDK

We can install Apache Groovy from this link

<https://computingforgeeks.com/how-to-install-apache-groovy-on-ubuntu-18-04-ubuntu-16-04/>

git clone --recursive https://github.com/folio-org/mod-agreements

cd mod-agreements

Build and run mod-agreements stand alone

cd service

grails war

../scripts/run\_external\_reg.sh

Register the module and load some test data

cd scripts ./register\_and\_enable.sh ./dev\_submit\_pkg.sh ./dev\_trigger\_kb\_sync.sh

## Enabling the mod-agreements module for our tenants

curl -w '\n' -D - -X POST  \

  -H "Content-type: application/json" \

   -d'{"id":"mod-agreements"}'

http://localhost:9130/\_/proxy/tenants/<tenant\_id>/modules

# Conclusion

In the end I can create tenants and finish operations (declare, deploy and enable) on some modules:

* mod-login
* mod-users
* mod-circulation-storage

And some modules have problem in enable operation like:

* mod-permissions
* mod-authtoken
* mod-agreements
* mod-data-import