Ref. Ares(2022)4721725 - 28/06/2022

ReMAP

Real-time Condition-based Maintenance for Adaptive Aircraft Maintenance Planning

D2.5 IT platform packaging

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 769288

Document History

Revision Nr	Description	Author	Review	Date
v0.0	ToC	ATOS		24/07/2019
v0.1	Deployment guides & developer guides	ATOS		06/05/2022
v0.2	GitLab runner & CI/CD strategy	ATOS		10/05/2022
v0.3	Repository structure	ATOS		26/05/2022
v1.0	First draft	ATOS		31/05/2022
V1.1	Review draft	UPAT		17/06/2022

Index

1. Introduction ... 6

1.1. Scope of this deliverable ... 6

1.2. Goals ... 6

1.3. Structure of this deliverable ... 6

2. CI/CD Common Infrastructure ... 7

3. IT Platform Components Code Repository ... 11

3.1. Core ... 12

3.1.1. https-proxy ... 13

3.1.2. mng-tools-gui ... 15

3.1.3. mng-tools-api ... 16

3.1.1. connector ... 18

3.1.2. keycloak ... 19

3.1.3. keycloak-initializer ... 19

3.1.4. pypi-server ... 20

3.2. Node ... 21

3.2.1. https-proxy ... 22

3.2.2. mng-tools-gui ... 24

3.2.3. mng-tools-api ... 25

3.2.4. data-acquisition-api ... 28

3.2.5. data-adaptor-api ... 29

3.2.6. local-repository ... 30

3.2.7. data-adaptor-cron ... 31

3.2.8. federation ... 33

3.2.9. connector ... 34

3.2.10. keycloak ... 35

3.2.11. keycloak-initializer ... 36

3.2.12. pypi-server ... 37

4. Deployment guide (node) ... 38

4.1. Prerequisites ... 38

4.2. Installing – production ... 38

4.3. Installing – development environment ... 39
4.3.1. Automatic setup ... 40
4.3.2. Manual setup ... 40
4.3.3. Local domain ... 40
4.3.4. Starting the node ... 41
4.4. Building (manual) ... 41
5. Deployment guide (core) ... 42
5.1. Prerequisites ... 42
5.2. Installing – production ... 42
5.3. Installing – development environment ... 43
5.3.1. Automatic setup ... 44
5.3.2. Manual setup ... 44
5.3.3. Local domain ... 44
5.3.4. Starting the core ... 45
5.4. Building (manual) ... 45
6. Developer guides ... 45
6.1. Debugging ... 45
6.2. Adding a new service ... 46
6.2.1. Declare the service in the docker-compose files ... 47
6.2.2. Route the new service through the nginx reverse proxy (for production deployment) ... 47
6.2.3. Update .env files ... 47
6.2.4. Keycloak authentication ... 47
6.2.5. Protecting an API ... 49
7. Conclusions ... 51
8. References ... 52

Figures

Figure 1 GitLab runner variables 8
Figure 2 GitLab CI/CD pipelines 8
Figure 3 Successful CI/CD job 9
Figure 4 Core Docker repository (https://hub.docker.com/u/remapcore) 9
Figure 5 Node Docker repository (https://hub.docker.com/u/remapnode) 10
Figure 6 ReMAP repository 11
Figure 7 Core repository structure 13
Figure 8 Node repository structure 22

1. Introduction

1.1. Scope of this deliverable

Following the architecture described in D2.3 ReMAP platform architecture [1], expanded with the technical documentation in D2.4 IT platform code & technical documentation [2], this document closes the platform loop with an insight in the packaging, distribution and deployment tools and techniques.

1.2. Goals

The goal of this document is to describe the CI/CD pipeline that enables each component composing the ReMAP IT platform to be packaged as a Docker image to facilitate its distribution, as well as a one-line deployment strategy for the whole core (or node) with the help docker-compose.

1.3. Structure of this deliverable

This document starts with a brief overview on the common CI/CD infrastructure that manages automatic build and distribution of the components after a code update to the project repository. Then, the different components of the core and node are described, this time describing how each of them is bundled in a docker image as well as the deployment configuration details as they are reflected in the docker-compose script.

After each component details have been described, the document presents deployment guides for the core and node, both for a production environment or a development setup. Finally, some developer guidelines are included at the end of the document, showcasing how relatively easy it would be to include a new component in the ReMAP ecosystem, showcasing once more the modular approach of the ReMAP platform.

2. CI/CD Common Infrastructure

The complete ReMAP platform code and CI/CD configuration is hosted in private GitLab repository hosted at Atos premises [3]. Following an agile approach, as each development sprint is closed, the development team merges the changes to the master branch of the repository, and this stable version is tagged as 'sprint-XX'. Using GitLab's built-in CI/CD tools [4], docker images for these stable versions are automatically built and published in the project Docker repositories.

This is achieved with the following .gitlab-cy.yml script, which can be found at the root of the repository, following GitLab guidelines.

```
build-master:
  stage: build
  script:
    - cd node
    - TAG=latest
    - echo "Building $TAG"
    - echo "TAG=$TAG" > .env
    - docker-compose -f build.yml build --pull
    - echo $DOCKER_REGISTRY_TOKEN | docker login -u $DOCKER_REGISTRY_USER --password-stdin
    - docker-compose -f build.yml push
    - cd ../core
    - TAG=latest
    - echo "Building $TAG"
    - echo "TAG=$TAG" > .env
    - docker-compose -f build.yml build --pull
    - echo $DOCKER_REGISTRY_TOKEN | docker login -u $DOCKER_REGISTRY_USER --password-stdin
    - docker-compose -f build.yml push
  tags:
    - build
  only:
    - master

build-tags:
  stage: build
  script:
    - cd node
    - TAG=$CI_COMMIT_REF_SLUG
    - echo "Building $TAG"
    - echo "TAG=$TAG" > .env
    - docker-compose -f build.yml build --pull
    - echo $DOCKER_REGISTRY_TOKEN | docker login -u $DOCKER_REGISTRY_USER --password-stdin
    - docker-compose -f build.yml push
    - cd ../core
    - TAG=$CI_COMMIT_REF_SLUG
    - echo "Building $TAG"
    - echo "TAG=$TAG" > .env
    - docker-compose -f build.yml build --pull
    - echo $DOCKER_REGISTRY_TOKEN | docker login -u $DOCKER_REGISTRY_USER --password-stdin
    - docker-compose -f build.yml push
  tags:
    - build
  only:
    - tags
```

These build jobs are run by a GitLab runner also hosted on Atos infrastructure. The necessary sensitive variables to publish the images to the project's Docker Hub are set through the project's configuration in GitLab, so only approved Atos developers can edit.

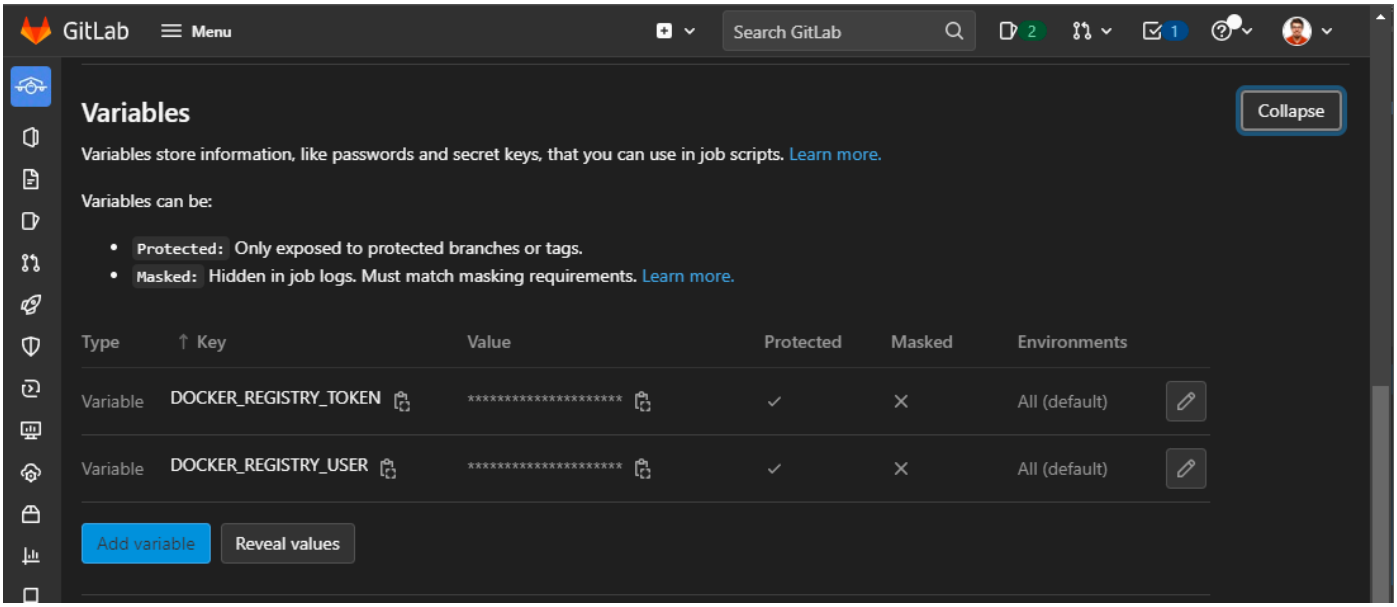

Figure 1 GitLab runner variables

As discussed, after each commit to the master branch and each tag created, a new CI/CD pipeline will be started by GitLab.

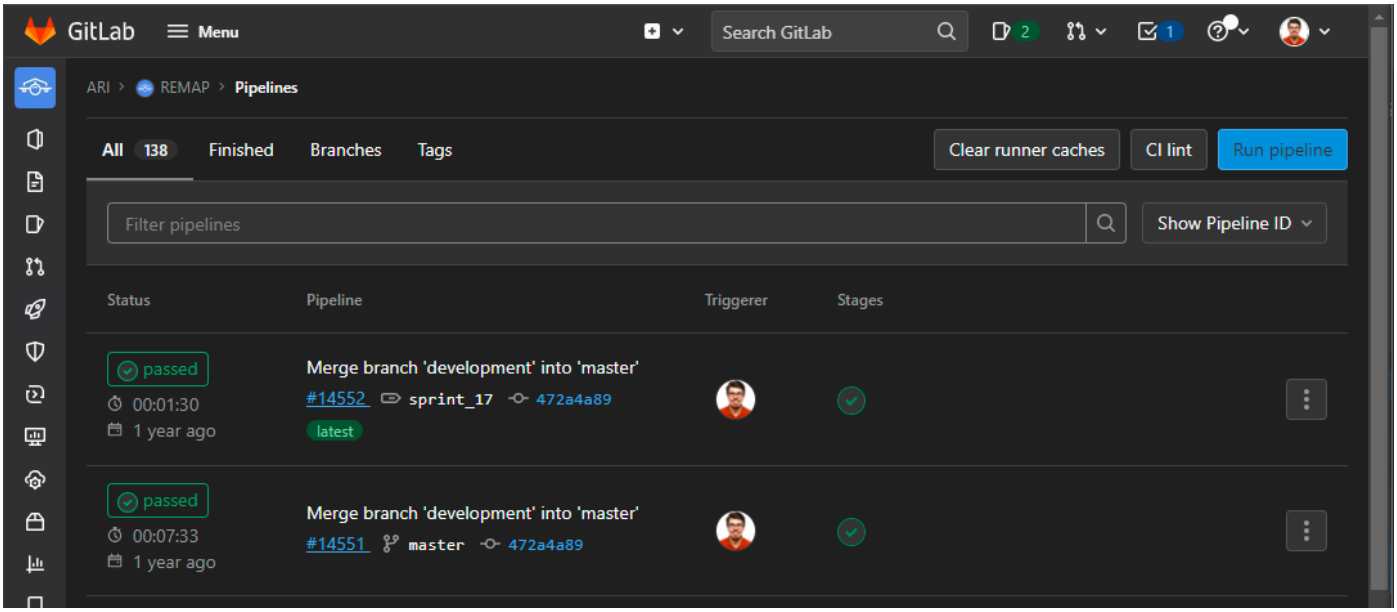

Figure 2 GitLab CI/CD pipelines

If the process was successful, the new images will be found in the Docker Hub repositories of the core [5] and the node [6].

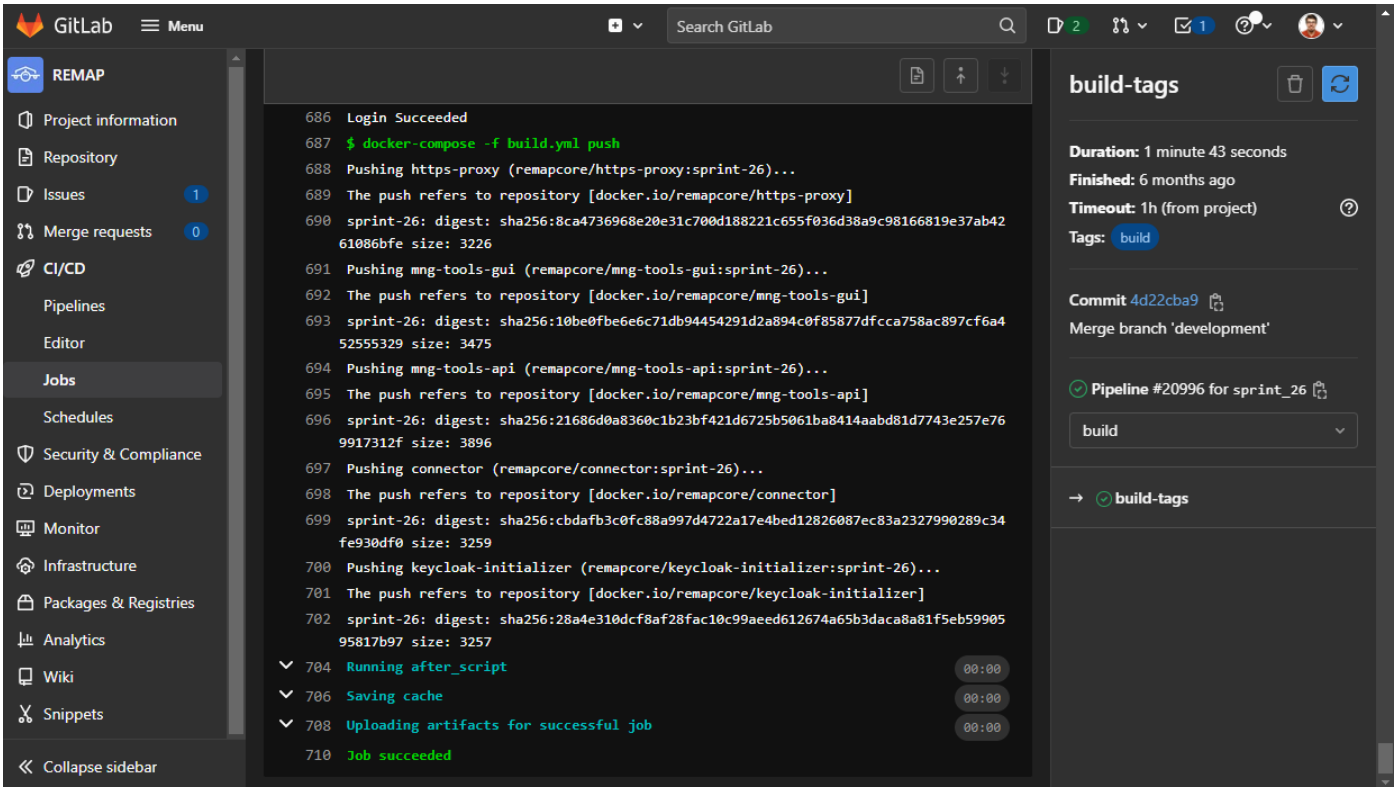

Figure 3 Successful CI/CD job

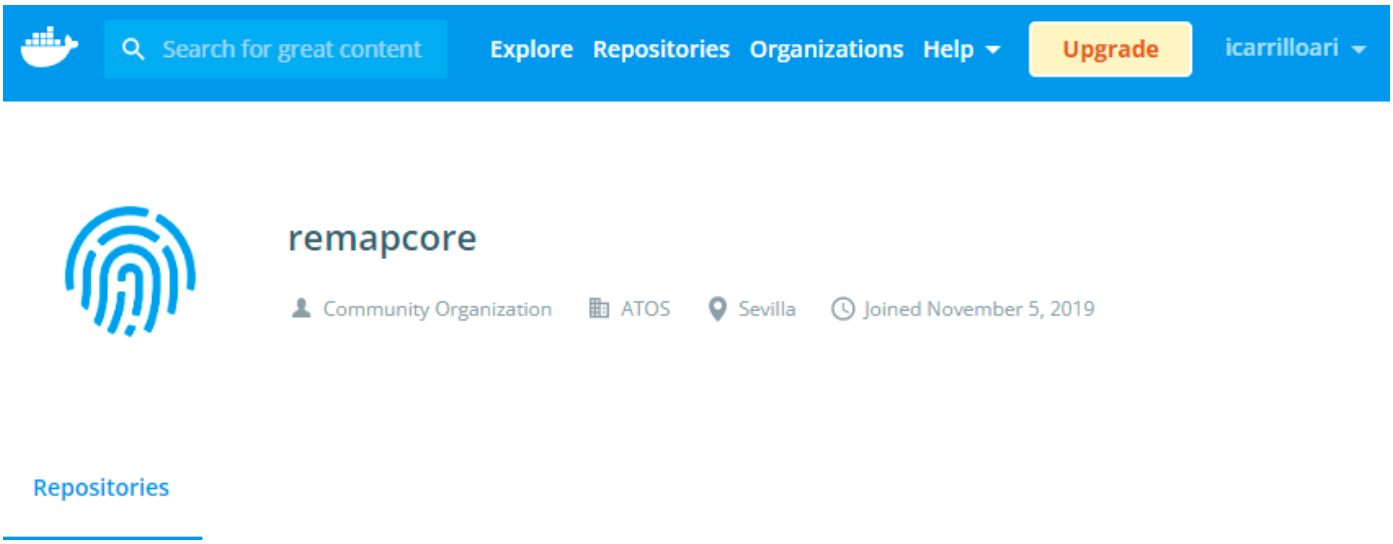

Figure 4 Core Docker repository (https://hub.docker.com/u/remapcore)

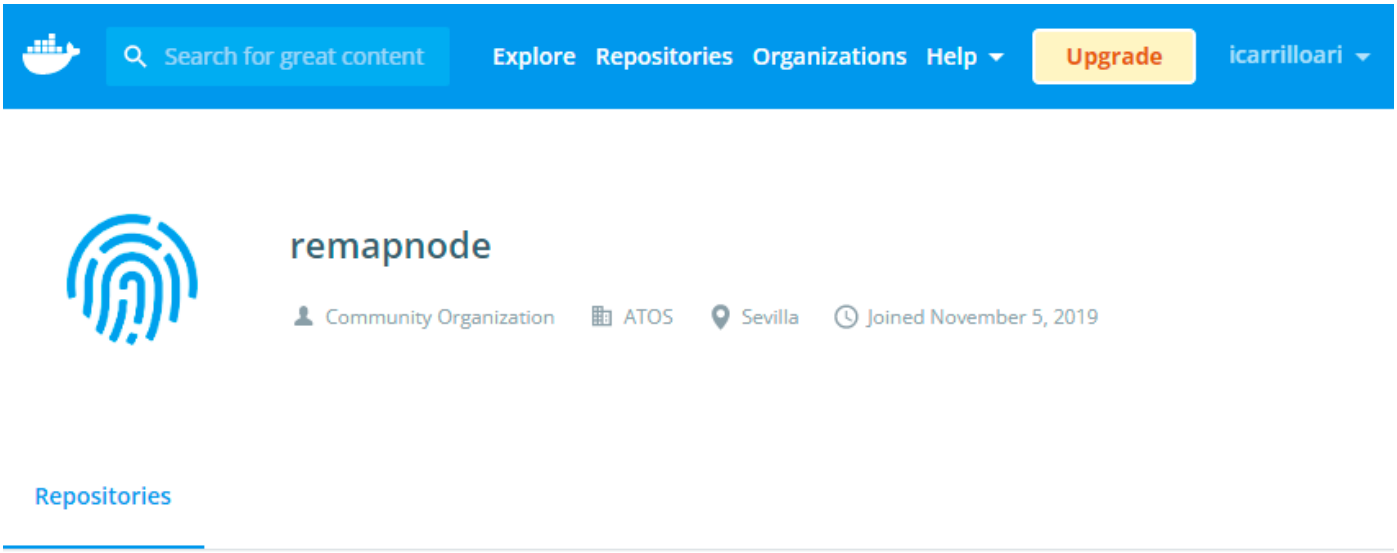

Figure 5 Node Docker repository (https://hub.docker.com/u/remapnode)

While a continuous deployment strategy is technically feasible with relatively low effort (given the pipeline in place already), during the project phase it was decided to stay at the continuous delivery stage instead for two main reasons:

- For security reasons, it might have been hard to get the approval of the airlines IT teams to host a GitLab runner to automatically deploy external code unsupervised
- During the development stage (especially during the first sprints), it was common to gradually include new modules, configuration options and in some rare occasions even updates to the data model that might break backwards compatibility. Thus, it made sense to have a member of the development team supervise the deployment

Anyhow, as it can be seen from the deployment guides in the next sections, the (re)deployment of either the core or the node is as simple as updating the TAG variable in the .env file to the latest version and re-running the one-liner “docker-compose up -d”, so the pros of a hybrid continuous delivery + manual deployment outweigh the cons of a fully automatized continuous deployment strategy for our case.

3. IT Platform Components Code Repository

As discussed, the repository is hosted in Atos private GitLab. The main folders that host the node and core code and deployment files are the ‘node’ and ‘core’ folders respectively:

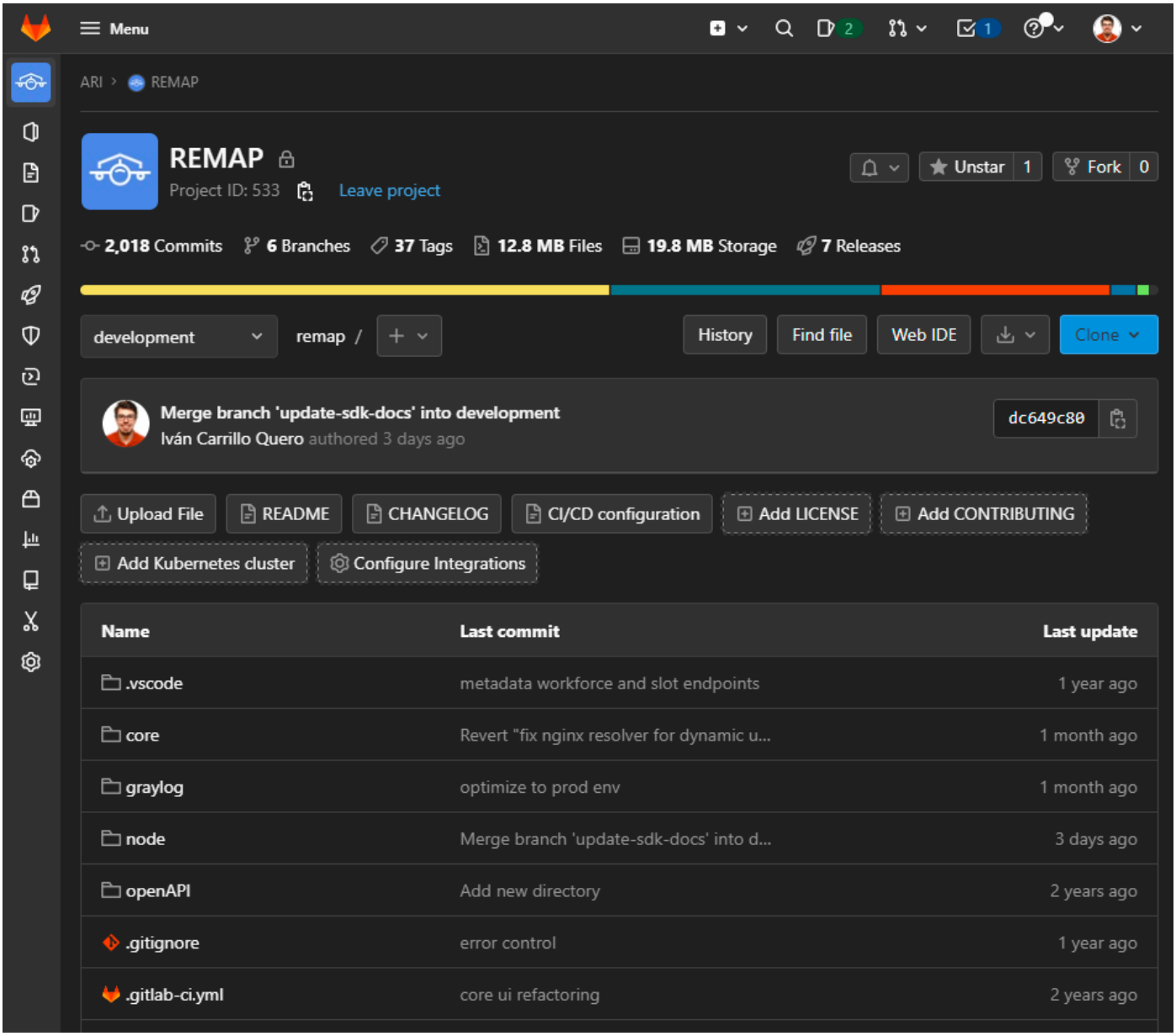

Figure 6 ReMAP repository

While the reader might be familiar with D2.3 and D2.4 definitions on the architecture of the platform from a logical and technical point of view of each of the modules, this section will go through the packaging and deployment considerations for each of the containers that compose the ReMAP node and core respectively.

All the images have been built with parametrization and reusability in mind, that's why all the configurations can be done through environment variables or in some specific cases, mounting some files as docker volumes. This ensures that in a development environment or in a future scenario where a component might be hosted in a different server, no code modifications will be required, instead the administrator can simply edit the configuration file and re-deploy the container in a matter of minutes.

While not mentioned in previous documentation as it technically is not part of the ReMAP ecosystem, the core and node do ship with an additional nginx-based reverse proxy container to facilitate a secure deployment of the system in a production environment.

3.1. Core

The 'core' folder of the repository contains all the necessary files to develop, build and deploy the core. Different docker-compose templates for building, deploying locally or in a production environment are included in the repository, along with the necessary parametrization templates (.env files), which the deployment guide section will discuss. This section will instead focus on the actual services and how they are configured in the docker-compose scripts, focusing on the production deployment setup where the images are directly pulled from the Docker Hub repositories already mentioned.

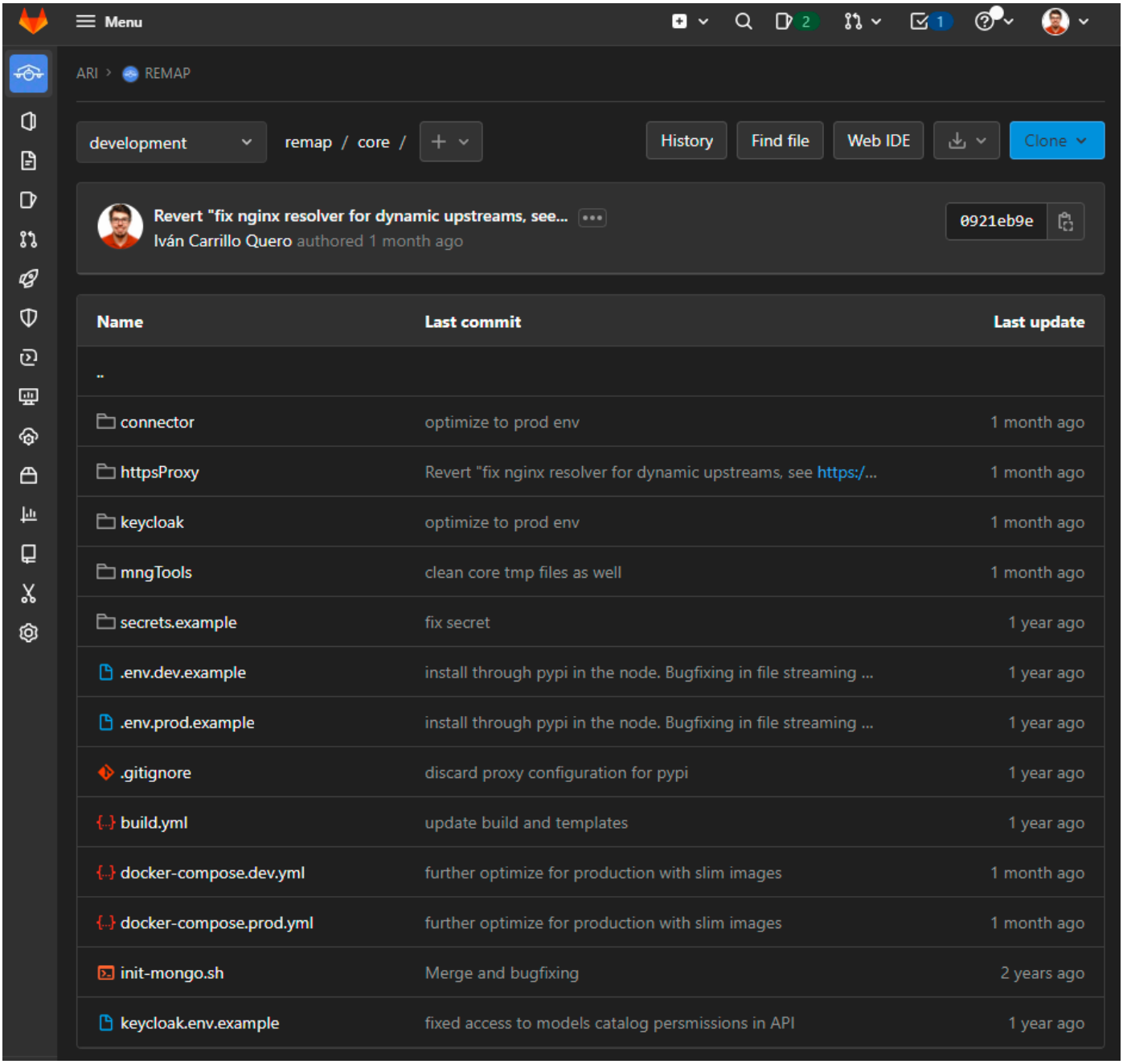

Figure 7 Core repository structure

3.1.1. https-proxy

To securely deploy the core in a production environment, a pre-configured nginx-based reverse proxy is bundled with the core, which will centralize all incoming traffic to the core, manage the HTTPS connections and forward the traffic to the appropriate service. The httpsProxy folder in the core repository already contains the required configurations to route the traffic to each of the exposed components of the core depending on the path in the request:

```
# Management Tools API
location /mngTools/api {
    proxy_pass  http://mngToolsApi;
    include     /etc/nginx/proxy.conf;
}

# Keycloak
location /connector {
    proxy_pass  http://connector;
    include     /etc/nginx/proxy.conf;
}

# Keycloak
location /auth {
    proxy_pass  http://keycloak;
    include     /etc/nginx/proxy.conf;
}

# Management Tools GUI
location / {
    proxy_pass  http://mngToolsGui;
    include     /etc/nginx/proxy.conf;
}
```

The Dockerfile for this component uses the nginx:stable image:

```
FROM nginx:stable

EXPOSE 81 443

ADD proxy.conf /etc/nginx/
ADD securityHeaders.conf /etc/nginx/
ADD sslConfig.conf /etc/nginx/
ADD remap.conf /etc/nginx/
ADD conf.d/* /etc/nginx/conf.d/
ADD templates/* /confTemplates/
ADD entrypoint.sh .

RUN chmod +x entrypoint.sh

ENTRYPOINT [ "./entrypoint.sh" ]
```

In the docker-compose script, this service is configured as follows:

```
  https-proxy:
    image: remapcore/https-proxy:${TAG:-latest}
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - HTTPS_ENABLED=${HTTPS_ENABLED:?err}
    ports:
      - 81:81/tcp
      - 443:443/tcp
    volumes:
      - ${FULLCHAIN_PEM:-/dev/null}:/certs/fullchain.pem:ro
      - ${PRIVKEY_PEM:-/dev/null}:/certs/privkey.pem:ro
      - ${DHPARAM_DH:-/dev/null}:/certs/dhparam.pem:ro
    networks:
      proxy:
      default:
```

Environment variable HTTPS_ENABLED configures whether the proxy should enforce HTTPS connections (certificates must be supplied) or only HTTP

- If HTTPS is enabled, the environment variables FULLCHAIN_PEM, PRIVKEY_PEM and DHPARAM_DH must be set as well to point to the appropriate certificate files
- Note that this is the only container with “ports” configured, as this is the only service directly open to the internet

3.1.2. mng-tools-gui

The code for the graphical user interface (GUI) AngularJS application can be found under mngTools/gui/. The Dockerfile for this component uses the node:12.9.1-slim base image:

```
FROM node:12.9.1-slim AS builder

WORKDIR /usr/src/app

# First install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm i

# Copy and build sources only after installing dependencies in order to
# fully take advantage of caching
COPY . .

RUN $(npm bin)/ng build --prod

RUN chmod +x entrypoint.sh

CMD [ "./entrypoint.sh" ]

EXPOSE 8080
```

In the docker-compose script, this service is configured as follows:

```
  mng-tools-gui:
    image: remapcore/mng-tools-gui:${TAG:-latest}
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - MNG_TOOLS_API_HOST=${PUBLIC_ADDR:?err}
      - MNG_TOOLS_API_PORT=${PUBLIC_PORT:?err}
      - KEYCLOAK_HOST=${PUBLIC_ADDR:?err}
      - KEYCLOAK_PORT=${PUBLIC_PORT:?err}
    env_file:
      - keycloak.env
    networks:
      - proxy
      - default
```

3.1.3. mng-tools-api

The code for the management tools API can be found under mngTools/api/. While this component uses node:12.9.1 base image, it uses a multi-stage build to also install the required python dependencies to internally communicate with the private pypi-server and manage models:

```
FROM node:12.9.1 AS python-build

# Install python dependencies
RUN wget https://www.python.org/ftp/python/3.8.5/Python-3.8.5.tgz && \
    tar zxf Python-3.8.5.tgz && \
    cd Python-3.8.5/ && \
    ./configure --prefix=/usr/local && \
    make && \
    make install && \
    rm -f Python-3.8.5.tgz && \
    rm -rf Python-3.8.5

RUN python3 -m pip install -U pip
RUN python3 -m pip install twine

RUN python3 --version
RUN python3 -m pip --version
RUN pip3 --version

FROM node:12.9.1-slim

# Copy python installation from previous stage
COPY --from=python-build /usr/local/bin/python* /usr/local/bin/pip* /usr/local/bin/pkginfo
/usr/local/bin/pydoc* /usr/local/bin/
COPY --from=python-build /usr/local/lib /usr/local/lib

RUN python3 --version
RUN python3 -m pip --version
RUN pip3 --version

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm i

# Bundle app source
COPY . .

EXPOSE 3200

RUN chmod +x entrypoint.sh

CMD [ "./entrypoint.sh" ]
```

In the docker-compose script, this service is configured as follows:

```
  mng-tools-api:
    image: remapcore/mng-tools-api:${TAG:-latest}
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - PUBLIC_ADDR=${PUBLIC_ADDR:?err}
      - PUBLIC_PORT=${PUBLIC_PORT:?err}
      - CONNECTOR_HOST=${PUBLIC_ADDR:?err}
      - CONNECTOR_PORT=${PUBLIC_PORT:?err}
      - PYPI_HOST=${PYPI_INTERNAL_HOST:?err}
      - PYPI_PORT=${PYPI_INTERNAL_PORT:?err}
      - KEYCLOAK_HOST=${PUBLIC_ADDR:?err}
      - KEYCLOAK_PORT=${PUBLIC_PORT:?err}
      - KEYCLOAK_USER_FILE=/run/secrets/keycloak-username
      - KEYCLOAK_PASSWORD_FILE=/run/secrets/keycloak-password
      - MONGO_HOST=${MNG_TOOLS_DB_HOST:?err}
      - MONGO_PORT=${MNG_TOOLS_DB_PORT:?err}
      - MONGO_INITDB_USERNAME_FILE=/run/secrets/mng-tools-db-username
      - MONGO_INITDB_PASSWORD_FILE=/run/secrets/mng-tools-db-password
      - MONGO_INITDB_DATABASE_FILE=/run/secrets/mng-tools-db-database
      - LOG_LEVEL=${LOG_LEVEL:-info}
    env_file:
      - keycloak.env
    secrets:
      - keycloak-username
      - keycloak-password
      - pypirc
      - mng-tools-db-username
      - mng-tools-db-password
      - mng-tools-db-database
    networks:
      - proxy
      - mng-tools-db
      - pypi
      - default
    volumes:
      - keycloak-mng-tools-api-config:/keycloak-config:ro
      - python-data-models:/data/packages
```

Several environment variables, secrets and volumes are passed to this component, so that it can reach the different services as well as credentials to manage the model repository or Keycloak for example.

This component connects to a mongo:4.0.10 database, as described in the docker-compose script:

```
  mng-tools-db:
    image: mongo:4.0.10
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - MONGO_INITDB_ROOT_USERNAME_FILE=/run/secrets/mng-tools-db-root-username
      - MONGO_INITDB_ROOT_PASSWORD_FILE=/run/secrets/mng-tools-db-root-password
      - MONGO_INITDB_USERNAME_FILE=/run/secrets/mng-tools-db-username
      - MONGO_INITDB_PASSWORD_FILE=/run/secrets/mng-tools-db-password
      - MONGO_INITDB_DATABASE_FILE=/run/secrets/mng-tools-db-database
    secrets:
      - mng-tools-db-root-username
      - mng-tools-db-root-password
      - mng-tools-db-username
      - mng-tools-db-password
```

```
      - mng-tools-db-database
    networks:
      - mng-tools-db
    volumes:
      - mng-tools-db:/data/db
      - ./init-mongo.sh:/docker-entrypoint-initdb.d/init-mongo.sh:ro
```

3.1.1. connector

The code for the connector application can be found under connector/. The Dockerfile for this component uses the node:12.9.1-slim base image:

```
FROM node:12.9.1-slim

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm i

# Bundle app source
COPY . .

EXPOSE 3200

RUN chmod +x entrypoint.sh

CMD [ "./entrypoint.sh" ]
```

In the docker-compose script, this service is configured as follows:

```
  connector:
    image: remapcore/connector:${TAG:-latest}
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - MNG_TOOLS_API_HOST=${PUBLIC_ADDR:?err}
      - MNG_TOOLS_API_PORT=${PUBLIC_PORT:?err}
      - KEYCLOAK_HOST=${PUBLIC_ADDR:?err}
      - KEYCLOAK_PORT=${PUBLIC_PORT:?err}
      - PUBLIC_ADDR=${PUBLIC_ADDR:?err}
      - PUBLIC_PORT=${PUBLIC_PORT:?err}
      - LOG_LEVEL=${LOG_LEVEL:?err}
    env_file:
      - keycloak.env
    volumes:
      - keycloak-connector-config:/keycloak-config:ro
    networks:
      - proxy
      - default
```

3.1.2. keycloak

Keycloak is deployed using the official jboss/keycloak:7.0.1 image and mariadb:10.4.6 for its database, as described in the docker-compose script:

```
 keycloak:
    image: jboss/keycloak:7.0.1
    environment:
      - TZ=${TZ}
      - KEYCLOAK_USER_FILE=/run/secrets/keycloak-username
      - KEYCLOAK_PASSWORD_FILE=/run/secrets/keycloak-password
      - DB_VENDOR=${KEYCLOAK_DB_VENDOR:?err}
      - DB_ADDR=${KEYCLOAK_DB_ADDR:?err}
      - DB_PORT=${KEYCLOAK_DB_PORT:?err}
      - DB_DATABASE=${KEYCLOAK_DB_DATABASE:?err}
      - DB_USER_FILE=/run/secrets/keycloak-db-user
      - DB_PASSWORD_FILE=/run/secrets/keycloak-db-password
      - PROXY_ADDRESS_FORWARDING=true
      - JAVA_TOOL_OPTIONS=-Dkeycloak.profile.feature.token_exchange=enabled
    secrets:
      - keycloak-username
      - keycloak-password
      - keycloak-db-user
      - keycloak-db-password
    restart: unless-stopped
    networks:
      - keycloak
      - keycloak-db
      - proxy
      - default

  keycloak-db:
    image: mariadb:10.4.6
    environment:
      - TZ=${TZ}
      - MYSQL_DATABASE=${KEYCLOAK_DB_DATABASE:?err}
      - MYSQL_ROOT_PASSWORD_FILE=/run/secrets/keycloak-db-root-password
      - MYSQL_USER_FILE=/run/secrets/keycloak-db-user
      - MYSQL_PASSWORD_FILE=/run/secrets/keycloak-db-password
    secrets:
      - keycloak-db-root-password
      - keycloak-db-user
      - keycloak-db-password
    restart: unless-stopped
    volumes:
      - keycloak-db:/var/lib/mysql
    networks:
      - keycloak-db
```

3.1.3. keycloak-initializer

A NodeJS script to initialize the Keycloak and distribute the client credentials along the different services can be found under keycloak/.

The Dockerfile for this component uses the node:12.9.1-slim base image:

```
FROM node:12.9.1-slim

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm i

# Bundle app source
COPY . .

RUN chmod +x entrypoint.sh

CMD [ "./entrypoint.sh" ]
```

In the docker-compose script, this service is configured as follows:

```
  keycloak-initializer:
    image: remapcore/keycloak-initializer:${TAG:-latest}
    environment:
      - TZ=${TZ}
      - KEYCLOAK_USER_FILE=/run/secrets/keycloak-username
      - KEYCLOAK_PASSWORD_FILE=/run/secrets/keycloak-password
      - KEYCLOAK_HOST=${KEYCLOAK_HOST_INTERNAL:?err}
      - KEYCLOAK_PORT=${KEYCLOAK_PORT_INTERNAL:?err}
      - PUBLIC_ADDR=${PUBLIC_ADDR:?err}
      - PUBLIC_PORT=${PUBLIC_PORT:?err}
      - MNG_TOOLS_API_PORT=${PUBLIC_PORT:?err}
      - CONNECTOR_API_PORT=${PUBLIC_PORT:?err}
      - LOG_LEVEL=${LOG_LEVEL:?err}
    env_file:
      - keycloak.env
    secrets:
      - keycloak-username
      - keycloak-password
    restart: on-failure
    volumes:
      - keycloak-mng-tools-api-config:/keycloak-mng-tools-api-config
      - keycloak-connector-config:/keycloak-connector-config
    networks:
      - keycloak
```

3.1.4. pypi-server

The private model repository runs a pypi-server:v1.4.2, secured by a .htpasswd file passed as a secret:

```
  pypi-server:
    image: pypiserver/pypiserver:v1.4.2
    restart: unless-stopped
    environment:
      - TZ=${TZ}
    secrets:
```

```
      - .htpasswd
    volumes:
      - python-data-models:/data/packages
    networks:
      - pypi
      - default
    command: -P /run/secrets/.htpasswd -a update -v /data/packages
```

3.2. Node

The ‘node folder of the repository contains all the necessary files to develop, build and deploy the node. Different docker-compose templates for building, deploying locally or in a production environment are included in the repository, along with the necessary parametrization templates (.env files), which the deployment guide section will discuss. This section will instead focus on the actual services and how they are configured in the docker-compose scripts, focusing on the production deployment setup where the images are directly pulled from the Docker Hub repositories already mentioned.

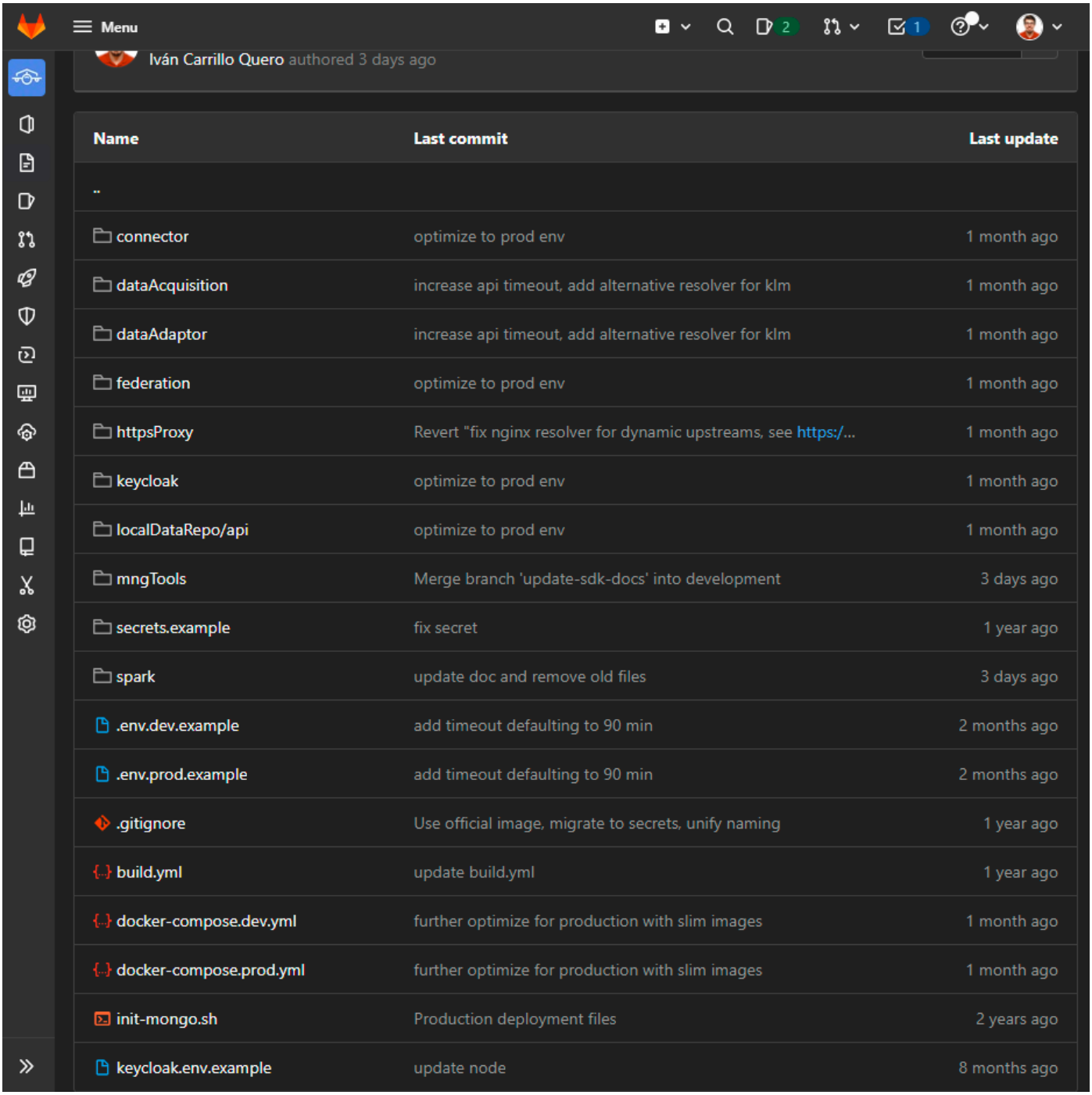

Figure 8 Node repository structure

3.2.1. https-proxy

To securely deploy the core in a production environment, a pre-configured nginx-based reverse proxy is bundled with the core, which will centralize all incoming traffic to the core, manage the HTTPS connections and forward the traffic to the appropriate service. The

httpsProxy folder in the core repository already contains the required configurations to route the traffic to each of the exposed components of the core depending on the path in the request:

```
# Management Tools API
location /mngTools/api {
    proxy_pass  http://mngToolsApi;
    include     /etc/nginx/proxy.conf;
}

# Data Adaptor API
location /dataAdaptor/api {
    proxy_pass  http://dataAdaptorApi;
    include     /etc/nginx/proxy.conf;
}

# Data Acquisition API
location /dataAcquisition {
    proxy_pass  http://dataAcquisition;
    include     /etc/nginx/proxy.conf;
}

# Local Repository API
location /localDataRepo {
    proxy_pass  http://localRepository;
    include     /etc/nginx/proxy.conf;
}

# Federation API
location /federation {
    proxy_pass  http://federation;
    include     /etc/nginx/proxy.conf;
}

# Federation API
location /connector {
    proxy_pass  http://connector;
    include     /etc/nginx/proxy.conf;
}

# Keycloak
location /auth {
    proxy_pass  http://keycloak;
    include     /etc/nginx/proxy.conf;
}

# Management Tools GUI
location / {
    proxy_pass  http://mngToolsGui;
    include     /etc/nginx/proxy.conf;
}
```

In the docker-compose script, this service is configured as follows:

```
  https-proxy:
    image: remapnode/https-proxy:${TAG:-latest}
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - HTTPS_ENABLED=${HTTPS_ENABLED:?err}
```

```
    ports:
      - 80:80/tcp
      - 443:443/tcp
    volumes:
      - ${FULLCHAIN_PEM:-/dev/null}:/certs/fullchain.pem:ro
      - ${PRIVKEY_PEM:-/dev/null}:/certs/privkey.pem:ro
      - ${DHPARAM_DH:-/dev/null}:/certs/dhparam.pem:ro
    networks:
      proxy:
      default:
```

- Environment variable HTTPS_ENABLED configures whether the proxy should enforce HTTPS connections (certificates must be supplied) or only HTTP
- If HTTPS is enabled, the environment variables FULLCHAIN_PEM, PRIVKEY_PEM and DHPARAM_DH must be set as well to point to the appropriate certificate files
- Note that this is the only container with “ports” configured, as this is the only service directly open to the internet

3.2.2. mng-tools-gui

The code for the graphical user interface (GUI) AngularJS application can be found under mngTools/gui/. The Dockerfile for this component uses the node:12.9.1-slim base image:

```
FROM node:12.9.1-slim AS builder

WORKDIR /usr/src/app

# First install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm i

# Copy and build sources only after installing dependencies in order to
# fully take advantage of caching
COPY . .

RUN $(npm bin)/ng build --prod

RUN chmod +x entrypoint.sh

CMD [ "./entrypoint.sh" ]

EXPOSE 8080
```

In the docker-compose script, this service is configured as follows:

```
  mng-tools-gui:
    image: remapnode/mng-tools-gui:${TAG:-latest}
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - MNG_TOOLS_API_HOST=${PUBLIC_ADDR:?err}
      - MNG_TOOLS_API_PORT=${PUBLIC_PORT:?err}
      - FEDERATION_API_HOST=${PUBLIC_ADDR:?err}
```

```
      - FEDERATION_API_PORT=${PUBLIC_PORT:?err}
      - DATA_ADAPTOR_API_HOST=${PUBLIC_ADDR:?err}
      - DATA_ADAPTOR_API_PORT=${PUBLIC_PORT:?err}
      - KEYCLOAK_HOST=${PUBLIC_ADDR:?err}
      - KEYCLOAK_PORT=${PUBLIC_PORT:?err}
    env_file:
      - keycloak.env
    networks:
      - proxy
      - default
```

3.2.3. mng-tools-api

The code for the management tools API can be found under mngTools/api/. While this component uses node:12.9.1 base image, it uses a multi-stage build to also install the required python dependencies to internally communicate with the private pypi-server to manage models as well as build and run the model-runner docker containers:

```
FROM node:12.9.1 as python-build

# Install python dependencies
RUN wget https://www.python.org/ftp/python/3.8.5/Python-3.8.5.tgz && \
    tar zxf Python-3.8.5.tgz && \
    cd Python-3.8.5/ && \
    ./configure --prefix=/usr/local && \
    make && \
    make install && \
    rm -f Python-3.8.5.tgz && \
    rm -rf Python-3.8.5

RUN python3 -m pip install -U pip
RUN python3 -m pip install twine

RUN python3 --version
RUN python3 -m pip --version
RUN pip3 --version

FROM node:12.9.1-slim as docker-install

# Install docker for model runner containers management
RUN apt-get update && apt install -y \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common

RUN curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add - && \
    add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb_release -cs)
stable" && \
    apt update && apt install -y docker-ce

FROM node:12.9.1-slim

# Copy docker binary
COPY --from=docker-install /usr/bin/docker /usr/bin/docker

# Copy python installation from previous stage
```

```
COPY --from=python-build /usr/local/bin/python* /usr/local/bin/pip* /usr/local/bin/pkginfo
/usr/local/bin/pydoc* /usr/local/bin/
COPY --from=python-build /usr/local/lib /usr/local/lib

RUN python3 --version
RUN python3 -m pip --version
RUN pip3 --version

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm i

# Copy model templates
COPY modelTemplates /modelTemplates

RUN chmod +x /modelTemplates/buildDocker.sh
RUN chmod +x /modelTemplates/runDocker.sh
RUN chmod +x /modelTemplates/cleanDocker.sh

# Bundle app source
COPY . .

EXPOSE 3200

RUN chmod +x entrypoint.sh

CMD [ "./entrypoint.sh" ]
```

In the docker-compose script, this service is configured as follows:

```
  mng-tools-api:
    image: remapnode/mng-tools-api:${TAG:-latest}
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - MNG_TOOLS_API_HOST=${PUBLIC_ADDR:?err}
      - MNG_TOOLS_API_PORT=${PUBLIC_PORT:?err}
      - DATA_ACQUISITION_API_HOST=${PUBLIC_ADDR:?err}
      - DATA_ACQUISITION_API_PORT=${PUBLIC_PORT:?err}
      - DATA_ADAPTOR_API_HOST=${PUBLIC_ADDR:?err}
      - DATA_ADAPTOR_API_PORT=${PUBLIC_PORT:?err}
      - CONNECTOR_API_HOST=${PUBLIC_ADDR:?err}
      - CONNECTOR_API_PORT=${PUBLIC_PORT:?err}
      - PYPI_HOST=${PYPI_INTERNAL_HOST:?err}
      - PYPI_PORT=${PYPI_INTERNAL_PORT:?err}
      - KEYCLOAK_HOST=${PUBLIC_ADDR:?err}
      - KEYCLOAK_PORT=${PUBLIC_PORT:?err}
      - LOCAL_REPOSITORY_API_HOST=${PUBLIC_ADDR:?err}
      - LOCAL_REPOSITORY_API_PORT=${PUBLIC_PORT:?err}
      - MONGO_HOST=${MNG_TOOLS_DB_HOST:?err}
      - MONGO_PORT=${MNG_TOOLS_DB_PORT:?err}
      - MONGO_INITDB_USERNAME_FILE=/run/secrets/mng-tools-db-username
      - MONGO_INITDB_PASSWORD_FILE=/run/secrets/mng-tools-db-password
      - MONGO_INITDB_DATABASE_FILE=/run/secrets/mng-tools-db-database
```

```
      - NODE_NETWORK=${NODE_NETWORK:?err}
      - EXEC_TIMEOUT_SECONDS=${EXEC_TIMEOUT_SECONDS:-5400}
      - LOG_LEVEL=${LOG_LEVEL:-info}
    env_file:
      - keycloak.env
    secrets:
      - pypirc
      - mng-tools-db-username
      - mng-tools-db-password
      - mng-tools-db-database
    networks:
      - proxy
      - mng-tools-db
      - default
    volumes:
      - keycloak-mng-tools-api-config:/keycloak-config:ro
      - keycloak-rul-sdk-config:/keycloak-rul-sdk-config:ro
      - local-repo-shared:/local-repo-shared
      - python-data-models:/data/packages
      - python-model-local-storage:/models/local-storage
      - ${DOCKER_SOCK:-/var/run/docker.sock}:/var/run/docker.sock
```

Several environment variables, secrets and volumes are passed to this component, so that it can reach the different services as well as credentials to manage the model repository, Keycloak, and even the docker socket.

Volume ‘local-repo-shared’ has been configured as an option to optimize memory usage when exporting datasets. Thus, when this component queries the local-repository for a large sensor dataset, the local-repository can use the disk to export the results to a temporary file for the mng-tools-api to read from, instead of loading the full result to memory which might highly impact performance.

This component connects to a mongo:4.0.10 database, as described in the docker-compose script:

```
  mng-tools-db:
    image: mongo:4.0.10
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - MONGO_INITDB_ROOT_USERNAME_FILE=/run/secrets/mng-tools-db-root-username
      - MONGO_INITDB_ROOT_PASSWORD_FILE=/run/secrets/mng-tools-db-root-password
      - MONGO_INITDB_USERNAME_FILE=/run/secrets/mng-tools-db-username
      - MONGO_INITDB_PASSWORD_FILE=/run/secrets/mng-tools-db-password
      - MONGO_INITDB_DATABASE_FILE=/run/secrets/mng-tools-db-database
    secrets:
      - mng-tools-db-root-username
      - mng-tools-db-root-password
      - mng-tools-db-username
      - mng-tools-db-password
      - mng-tools-db-database
    networks:
      - mng-tools-db
    volumes:
      - mng-tools-db:/data/db
      - ./init-mongo.sh:/docker-entrypoint-initdb.d/init-mongo.sh:ro
```

3.2.4. data-acquisition-api

The code for the data acquisition application can be found under dataAcquisition/. The Dockerfile for this component uses the node:12.9.1-slim base image:

```
FROM node:12.9.1-slim

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm i

# Bundle app source
COPY . .

EXPOSE 3100

RUN chmod +x entrypoint.sh

CMD [ "./entrypoint.sh" ]
```

In the docker-compose script, this service is configured as follows:

```
data-acquisition-api:
    image: remapnode/data-acquisition-api:${TAG:-latest}
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - MNG_TOOLS_API_HOST=${PUBLIC_ADDR:?err}
      - MNG_TOOLS_API_PORT=${PUBLIC_PORT:?err}
      - KEYCLOAK_HOST=${PUBLIC_ADDR:?err}
      - KEYCLOAK_PORT=${PUBLIC_PORT:?err}
      - CONNECTOR_API_HOST=${PUBLIC_ADDR:?err}
      - CONNECTOR_API_PORT=${PUBLIC_PORT:?err}
      - DATA_ADAPTOR_API_HOST=${PUBLIC_ADDR:?err}
      - DATA_ADAPTOR_API_PORT=${PUBLIC_PORT:?err}
      - MONGO_HOST=${DATA_ACQ_API_DB_HOST:?err}
      - MONGO_PORT=${DATA_ACQ_API_DB_PORT:?err}
      - MONGO_INITDB_USERNAME_FILE=/run/secrets/data-acquisition-api-db-username
      - MONGO_INITDB_PASSWORD_FILE=/run/secrets/data-acquisition-api-db-password
      - MONGO_INITDB_DATABASE_FILE=/run/secrets/data-acquisition-api-db-database
      - LOG_LEVEL=${LOG_LEVEL:-info}
      - MAX_RETRIES=${MAX_RETRIES:?err}
    env_file:
      - keycloak.env
    secrets:
      - data-acquisition-api-db-username
      - data-acquisition-api-db-password
      - data-acquisition-api-db-database
    networks:
      - proxy
      - default
      - data-acquisition-api-db
```

```
    volumes:
      - keycloak-data-acquisition-api-config:/keycloak-config:ro
```

This component connects to a mongo:4.0.10 database, as described in the docker-compose script:

```
  data-acquisition-api-db:
    image: mongo:4.0.10
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - MONGO_INITDB_ROOT_USERNAME_FILE=/run/secrets/data-acquisition-api-db-root-username
      - MONGO_INITDB_ROOT_PASSWORD_FILE=/run/secrets/data-acquisition-api-db-root-password
      - MONGO_INITDB_USERNAME_FILE=/run/secrets/data-acquisition-api-db-username
      - MONGO_INITDB_PASSWORD_FILE=/run/secrets/data-acquisition-api-db-password
      - MONGO_INITDB_DATABASE_FILE=/run/secrets/data-acquisition-api-db-database
    secrets:
      - data-acquisition-api-db-root-username
      - data-acquisition-api-db-root-password
      - data-acquisition-api-db-username
      - data-acquisition-api-db-password
      - data-acquisition-api-db-database
    networks:
      - data-acquisition-api-db
    volumes:
      - data-acquisition-api-db:/data/db
      - ./init-mongo.sh:/docker-entrypoint-initdb.d/init-mongo.sh:ro
```

3.2.5. data-adaptor-api

The code for the data adaptor application can be found under dataAdaptor/api/. The Dockerfile for this component was upgraded to node:16.13.1-slim base image instead of the standard 12.9.1 used throughout the project to make use of some recent NodeJS functionalities to optimize the data processing pipeline:

```
FROM node:16.13.1-slim

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm i

# Bundle app source
COPY . .

EXPOSE 3200

RUN chmod +x entrypoint.sh

CMD [ "./entrypoint.sh" ]
```

In the docker-compose script, this service is configured as follows:

```
  data-adaptor-api:
    image: remapnode/data-adaptor-api:${TAG:-latest}
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - MNG_TOOLS_API_HOST=${PUBLIC_ADDR:?err}
      - MNG_TOOLS_API_PORT=${PUBLIC_PORT:?err}
      - LOCAL_REPOSITORY_API_HOST=${LOCAL_REPOSITORY_API_HOST:?err}
      - LOCAL_REPOSITORY_API_PORT=${LOCAL_REPOSITORY_API_PORT:?err}
      - DATA_ACQUISITION_API_HOST=${PUBLIC_ADDR:?err}
      - DATA_ACQUISITION_API_PORT=${PUBLIC_PORT:?err}
      - KEYCLOAK_HOST=${PUBLIC_ADDR:?err}
      - KEYCLOAK_PORT=${PUBLIC_PORT:?err}
      - LOG_LEVEL=${LOG_LEVEL:-info}
    env_file:
      - keycloak.env
    networks:
      - proxy
      - default
    volumes:
      - keycloak-data-adaptor-api-config:/keycloak-config:ro
```

3.2.6. local-repository

The code for the local data repository application can be found under localDataRepo/api/. The Dockerfile for this component uses the node:12.9.1-slim base image:

```
FROM node:12.9.1-slim

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm i

# Bundle app source
COPY . .

EXPOSE 3000

RUN chmod +x entrypoint.sh

CMD [ "./entrypoint.sh" ]
```

In the docker-compose script, this service is configured as follows:

```
  local-repository:
    image: remapnode/local-repository:${TAG:-latest}
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - LOG_LEVEL=${LOG_LEVEL:-info}
```

```
      - MONGO_HOST=${LOCAL_REPOSITORY_DB_HOST:?err}
      - MONGO_PORT=${LOCAL_REPOSITORY_DB_PORT:?err}
      - MONGO_INITDB_USERNAME_FILE=/run/secrets/local-repo-db-username
      - MONGO_INITDB_PASSWORD_FILE=/run/secrets/local-repo-db-password
      - MONGO_INITDB_DATABASE_FILE=/run/secrets/local-repo-db-database
      - KEYCLOAK_HOST=${PUBLIC_ADDR:?err}
      - KEYCLOAK_PORT=${PUBLIC_PORT:?err}
    env_file:
      - keycloak.env
    secrets:
      - local-repo-db-username
      - local-repo-db-password
      - local-repo-db-database
    networks:
      - local-repository-db
      - default
    volumes:
      - keycloak-local-repository-config:/keycloak-config
      - local-repo-shared:/local-repo-shared
```

This component connects to a mongo:5.0.5 database, as described in the docker-compose script. This mongo instance was upgraded to a newer version compared to the standard 4.0.10 used throughout the project. The reason for this upgrade was to make use of the latest wiredTiger options [7] to apply Facebook's Zstandard compression algorithm [8], which has proven to reduce disk usage while staying efficient in terms of CPU load.

```
  local-repository-db:
    image: mongo:5.0.5
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - MONGO_INITDB_ROOT_USERNAME_FILE=/run/secrets/local-repo-db-root-username
      - MONGO_INITDB_ROOT_PASSWORD_FILE=/run/secrets/local-repo-db-root-password
      - MONGO_INITDB_USERNAME_FILE=/run/secrets/local-repo-db-username
      - MONGO_INITDB_PASSWORD_FILE=/run/secrets/local-repo-db-password
      - MONGO_INITDB_DATABASE_FILE=/run/secrets/local-repo-db-database
    secrets:
      - local-repo-db-root-username
      - local-repo-db-root-password
      - local-repo-db-username
      - local-repo-db-password
      - local-repo-db-database
    volumes:
      - local-repository-db:/data/db
      - ./init-mongo.sh:/docker-entrypoint-initdb.d/init-mongo.sh:ro
    networks:
      - local-repository-db
    command: [ "--wiredTigerCollectionBlockCompressor", "zstd" ]
```

3.2.7. data-adaptor-cron

The code for the automatic job scheduler can be found under dataAdaptor/cron/. While this component uses a node:12.9.1-slim base image, the entrypoint for this image is not really a NodeJS process but rather fcron [9], an implementation of the standard Vixie Cron [10] that ships with most Unix systems. As it can be seen from the Dockerfile, this image includes several NodeJS applications that will be launched by the automatic job scheduler depending on the triggers defined in the platform:

- dbPoller is set to run each minute to check the datasources and model executions configured from the GUI, and add new entries for fcron to start the dataGetter or executor jobs periodically as configured
- dataGetter receives a datasource id as input to start the data pulling process by sending the appropriate request to the data acquisition API
- executor receives a model execution id as input to request the management tools API to spawn the appropriate model runner
- catalogSync is set to run daily to synchronize the node catalog to the latest version in the core

```
FROM node:12.9.1-slim

#####################
# Instal fcron
#####################
RUN apt-get update && apt-get -y install \
    gcc \
    make \
    ssmtp \
    vim-tiny
WORKDIR /tmp
ADD http://fcron.free.fr/archives/fcron-3.2.1.src.tar.gz fcron-3.2.1.tar.gz
RUN tar -xf fcron-3.2.1.tar.gz \
    && cd fcron-3.2.1 \
    && ./configure \
    && make \
    && make install
#####################
# End install fcron
#####################

RUN mkdir -p /usr/src/app \
    &&  mkdir -p /usr/src/app/dataGetter \
    &&  mkdir -p /usr/src/app/executor \
    &&  mkdir -p /usr/src/app/dbPoller \
    &&  mkdir -p /usr/src/app/catalogSync

#####################
# NODE APPs
#####################
# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
WORKDIR /usr/src/app/dataGetter
COPY dataGetter/package*.json ./
RUN npm i

WORKDIR /usr/src/app/dbPoller
COPY dbPoller/package*.json ./
RUN npm i

WORKDIR /usr/src/app/catalogSync
COPY catalogSync/package*.json ./
RUN npm i

WORKDIR /usr/src/app/executor
COPY executor/package*.json ./
RUN npm i
#####################
```

```
# END NODE APPs
####################

WORKDIR /usr/src/app

COPY db-poll.sh .
COPY catalog-sync.sh .

COPY init-fcron /etc/fcron.d/init-fcron
RUN cp /etc/fcron.d/init-fcron /etc/fcron.d/fcron
RUN fcrontab /etc/fcron.d/fcron

COPY run-task.sh .

COPY entrypoint.sh .

# Bundle app source
COPY . .

RUN chmod +x db-poll.sh
RUN chmod +x catalog-sync.sh
RUN chmod +x run-task.sh
RUN chmod +x entrypoint.sh

ENTRYPOINT [ "./entrypoint.sh" ]
```

In the docker-compose script, this service is configured as follows:

```
  data-adaptor-cron:
    image: remapnode/data-adaptor-cron:${TAG:-latest}
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - KEYCLOAK_HOST=${PUBLIC_ADDR:?err}
      - KEYCLOAK_PORT=${PUBLIC_PORT:?err}
      - CONNECTOR_API_HOST=${PUBLIC_ADDR:?err}
      - CONNECTOR_API_PORT=${PUBLIC_PORT:?err}
      - DATA_ACQUISITION_API_HOST=${PUBLIC_ADDR:?err}
      - DATA_ACQUISITION_API_PORT=${PUBLIC_PORT:?err}
      - MNG_TOOLS_API_HOST=${PUBLIC_ADDR:?err}
      - MNG_TOOLS_API_PORT=${PUBLIC_PORT:?err}
      - LOG_LEVEL=${LOG_LEVEL:?err}
    env_file:
      - keycloak.env
    volumes:
      - keycloak-data-adaptor-cron-config:/keycloak-config:ro
```

3.2.8. federation

The code for the federation application can be found under federation/. The Dockerfile for this component uses the node:12.9.1-slim base image:

```
FROM node:12.9.1-slim

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app
```

```
# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm i

# Bundle app source
COPY . .

EXPOSE 3500

RUN chmod +x entrypoint.sh

CMD [ "./entrypoint.sh" ]
```

In the docker-compose script, this service is configured as follows:

```
  federation:
    image: remapnode/federation:${TAG:-latest}
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - KEYCLOAK_HOST=${PUBLIC_ADDR:?err}
      - KEYCLOAK_PORT=${PUBLIC_PORT:?err}
      - PUBLIC_ADDR=${PUBLIC_ADDR:?err}
      - PUBLIC_PORT=${PUBLIC_PORT:?err}
      - CONNECTOR_API_HOST=${PUBLIC_ADDR:?err}
      - CONNECTOR_API_PORT=${PUBLIC_PORT:?err}
      - LOG_LEVEL=${LOG_LEVEL:?err}
      - KEYCLOAK_USER_FILE=/run/secrets/keycloak-username
      - KEYCLOAK_PASSWORD_FILE=/run/secrets/keycloak-password
    volumes:
      - federation-log:/federation-log
      - keycloak-federation-config:/keycloak-config:ro
    env_file:
      - keycloak.env
    secrets:
      - keycloak-username
      - keycloak-password
    networks:
      - proxy
      - default
```

This component shares the ‘federation-log’ volume with the connector component, where some key federation information, such as the URL of the core connector can be found.

3.2.9. connector

The code for the connector application can be found under connector/. The Dockerfile for this component uses the node:12.9.1-slim base image:

```
FROM node:12.9.1-slim

RUN mkdir -p /usr/src/app
```

```
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm i

# Bundle app source
COPY . .

EXPOSE 3200

RUN chmod +x entrypoint.sh

CMD [ "./entrypoint.sh" ]
```

In the docker-compose script, this service is configured as follows:

```
  connector:
    image: remapnode/connector:${TAG:-latest}
    restart: unless-stopped
    environment:
      - TZ=${TZ}
      - KEYCLOAK_HOST=${PUBLIC_ADDR:?err}
      - KEYCLOAK_PORT=${PUBLIC_PORT:?err}
      - LOG_LEVEL=${LOG_LEVEL:?err}
    env_file:
      - keycloak.env
    volumes:
      - federation-log:/federation-log:ro
      - keycloak-connector-config:/keycloak-config:ro
    networks:
      - proxy
      - default
```

3.2.10. keycloak

Keycloak is deployed using the official jboss/keycloak:7.0.1 image and mariadb:10.4.6 for its database, as described in the docker-compose script:

```
 keycloak:
    image: jboss/keycloak:7.0.1
    environment:
      - TZ=${TZ}
      - KEYCLOAK_USER_FILE=/run/secrets/keycloak-username
      - KEYCLOAK_PASSWORD_FILE=/run/secrets/keycloak-password
      - DB_VENDOR=${KEYCLOAK_DB_VENDOR:?err}
      - DB_ADDR=${KEYCLOAK_DB_ADDR:?err}
      - DB_PORT=${KEYCLOAK_DB_PORT:?err}
      - DB_DATABASE=${KEYCLOAK_DB_DATABASE:?err}
      - DB_USER_FILE=/run/secrets/keycloak-db-user
      - DB_PASSWORD_FILE=/run/secrets/keycloak-db-password
      - PROXY_ADDRESS_FORWARDING=true
      - JAVA_TOOL_OPTIONS=-Dkeycloak.profile.feature.token_exchange=enabled
```

```
    secrets:
      - keycloak-username
      - keycloak-password
      - keycloak-db-user
      - keycloak-db-password
    restart: unless-stopped
    networks:
      - keycloak
      - keycloak-db
      - proxy
      - default

  keycloak-db:
    image: mariadb:10.4.6
    environment:
      - TZ=${TZ}
      - MYSQL_DATABASE=${KEYCLOAK_DB_DATABASE:?err}
      - MYSQL_ROOT_PASSWORD_FILE=/run/secrets/keycloak-db-root-password
      - MYSQL_USER_FILE=/run/secrets/keycloak-db-user
      - MYSQL_PASSWORD_FILE=/run/secrets/keycloak-db-password
    secrets:
      - keycloak-db-root-password
      - keycloak-db-user
      - keycloak-db-password
    restart: unless-stopped
    volumes:
      - keycloak-db:/var/lib/mysql
    networks:
      - keycloak-db
```

3.2.11. keycloak-initializer

A NodeJS script to initialize the Keycloak and distribute the client credentials along the different services can be found under keycloak/.

The Dockerfile for this component uses the node:12.9.1-slim base image:

```
FROM node:12.9.1-slim

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm i

# Bundle app source
COPY . .

RUN chmod +x entrypoint.sh

CMD [ "./entrypoint.sh" ]
```

In the docker-compose script, this service is configured as follows:

```
  keycloak-initializer:
    image: remapnode/keycloak-initializer:${TAG:-latest}
    environment:
      - TZ=${TZ}
      - KEYCLOAK_USER_FILE=/run/secrets/keycloak-username
      - KEYCLOAK_PASSWORD_FILE=/run/secrets/keycloak-password
      - KEYCLOAK_HOST=${KEYCLOAK_HOST_INTERNAL:?err}
      - KEYCLOAK_PORT=${KEYCLOAK_PORT_INTERNAL:?err}
      - PUBLIC_ADDR=${PUBLIC_ADDR:?err}
      - PUBLIC_PORT=${PUBLIC_PORT:?err}
      - MNG_TOOLS_API_PORT=${PUBLIC_PORT:?err}
      - DATA_ACQ_API_PORT=${PUBLIC_PORT:?err}
      - DATA_ADAPTOR_API_PORT=${PUBLIC_PORT:?err}
      - LOCAL_REPOSITORY_API_PORT=${PUBLIC_PORT:?err}
      - CONNECTOR_API_PORT=${PUBLIC_PORT:?err}
      - FEDERATION_API_PORT=${PUBLIC_PORT:?err}
      - LOG_LEVEL=${LOG_LEVEL:?err}
    env_file:
      - keycloak.env
    secrets:
      - keycloak-username
      - keycloak-password
    restart: on-failure
    volumes:
      - keycloak-mng-tools-api-config:/keycloak-mng-tools-api-config
      - keycloak-data-adaptor-cron-config:/keycloak-data-adaptor-cron-config
      - keycloak-data-adaptor-api-config:/keycloak-data-adaptor-api-config
      - keycloak-data-acquisition-api-config:/keycloak-data-acquisition-api-config
      - keycloak-local-repository-config:/keycloak-local-repository-config
      - keycloak-connector-config:/keycloak-connector-config
      - keycloak-federation-config:/keycloak-federation-config
      - keycloak-rul-sdk-config:/keycloak-rul-sdk-config
    networks:
      - keycloak
```

3.2.12. pypi-server

The private model repository runs a pypi-server:v1.4.2, secured by a .htpasswd file passed as a secret:

```
  pypi-server:
    image: pypiserver/pypiserver:v1.4.2
    restart: unless-stopped
    environment:
      - TZ=${TZ}
    secrets:
      - .htpasswd
    volumes:
      - python-data-models:/data/packages
    networks:
      - pypi
      - default
    command: -P /run/secrets/.htpasswd -a update -v /data/packages
```

4. Deployment guide (node)

4.1. Prerequisites

You will need git, docker and docker-compose (version 3.1+) installed in your system.

4.2. Installing – production

First of all, clone the repository and open the remap/node folder:

```
git clone https://scm.atosresearch.eu/ari/remap.git
cd remap/node
```

In order to run the node infrastructure, you must create a .env file to set certain environment variables. We recommend you copy the env.prod.example template we provide.

```
cp .env.prod.example .env
```

Keycloak roles, clients and realm must be defined in a keycloak.env file. We recommend you copy the keycloak.env.example template we provide. Most probably you will not need to modify any of these fields.

```
cp keycloak.env.example keycloak.env
```

Sensitive parameters are passed to the container as docker secrets. We recommend you copy the secrets.example template folder we provide and just edit each file's contents to change the default values. Each secret is defined as a separate file. **Please make sure that your editor is not inserting a new line at the end of the file to prevent the different services from interpreting the new line character as part of your secret.**

Secret files .pypirc and .htpasswd contain the credentials for the (internal) pypi server. The .htpasswd file requires the password field to be encoded and can be created as described in this section of the pypiserver documentation [11].

The different exposed services all sit behind an nginx reverse proxy, which handles any incoming connection to the node services. In order to enable HTTPS (which you should), you must set the values of the following variables in your .env file:

```
...

FULLCHAIN_PEM=/path/to/fullchain.pem
PRIVKEY_PEM=/path/to/key.pem
DHPARAM_DH=/path/to/dhparam.pem

PUBLIC_HOSTNAME=your.domain

HTTPS_ENABLED=true
PUBLIC_ADDR=https://your.domain
PUBLIC_PORT=443

...
```

While deprecated, HTTP is also supported in case you absolutely cannot provide signed certificates:

```
...

FULLCHAIN_PEM=
PRIVKEY_PEM=
DHPARAM_DH=

PUBLIC_HOSTNAME=your.domain

HTTPS_ENABLED=false
PUBLIC_ADDR=http://your.domain
PUBLIC_PORT=<PORT>

...
```

Copy the production docker-compose template:

```
cp docker-compose.prod.yml docker-compose.yml
```

On some configurations, iptables might default to dropping connections from docker containers to the host. In order to enable ReMAP services to access each other and especially the identity manager or an SFTP hosted on the same node by its public domain name, add the following rule to iptables:

```
iptables -I INPUT 1 -s 192.168.32.0/20 -j ACCEPT
```

Where 192.168.32.0/20 is the subnet specified for the default network in the docker.compose.prod.yml template.

Finally, run the following command, which will pull and set up the different services as docker containers:

```
docker-compose -p node up -d
```

Specifying parameter -p node configures the name of the docker-compose project to node, such that the default network configured earlier will be named as node_default. If you wish to change the name of the project to something else, please make sure to reflect the appropriate network name in the .env file NODE_NETWORK variable (which defaults to node_default).

You might need to wait a couple of minutes before Keycloak initializes the database. At this point you should be able to access the GUI at:

```
<PUBLIC_ADDR>:<PUBLIC_PORT>
```

The GUI should ask you to log in via Keycloak. If this is your first time running the project, you will need to access the Keycloak administration console to create a new user under the realm <KEYCLOAK_REALM> and give it 'Airline IT Staff' permisions at:

```
<PUBLIC_ADDR>:<PUBLIC_PORT>/auth
```

4.3. Installing – development environment

First of all, clone the repository and open the remap/node folder:

```
git clone https://scm.atosresearch.eu/ari/remap.git
cd remap/node
```

4.3.1. Automatic setup

Warning: All your existing configuration, running containers and volumes will be erased, both for node and core, use with caution.

If you wish to clean your local development environment and erase all your existing data, reset all configuration to the default values, and update everything with the latest templates, just run the cleanEnv.sh script.

4.3.2. Manual setup

In order to run the node infrastructure, you must create a .env file to set certain environment variables.

We recommend you copy the .env.dev.example template we provide.

```
cp .env.dev.example .env
```

Keycloak roles, clients and realm must be defined in a keycloak.env file. We recommend you copy the keycloak.env.example template we provide. Most probably you will not need to modify any of these fields.

```
cp keycloak.env.example keycloak.env
```

Sensitive parameters are passed to the container as docker secrets. We recommend you copy the secrets.example template folder we provide and just edit each file's contents to change the default values. Each secret is defined as a separate file. **Please make sure that your editor is not inserting a new line at the end of the file to prevent the different services from interpreting the new line character as part of your secret.**

Secret files .pypirc and .htpasswd contain the credentials for the (internal) pypi server. The .htpasswd file requires the password field to be encoded and can be created as described in this section of the pypiserver documentation [11].

Copy the development docker-compose template:

```
cp docker-compose.dev.yml docker-compose.yml
```

4.3.3. Local domain

In order to enable the different services to share a common hostname (so that a unique token issuer is seen by keycloak throughout the whole flow), you must edit the following variables in the .env file.

```
...

PUBLIC_HOSTNAME=<YOUR_MACHINE_HOSTNAME>
```

```
PUBLIC_ADDR=http://<YOUR_MACHINE_HOSTNAME>
PUBLIC_IP=<HOST_IP_IN_DOCKER_INTERFACE>

...
```

You can define a custom hostname in your /etc/hosts file. You can find your host's IP in the docker interface by running ifconfig and searching for the docker interface, tipically named docker0.

4.3.4. Starting the node

Finally, run the following command, which will build and set up the different services as docker containers:

```
docker-compose up -d --build
```

You might need to wait a couple of minutes before Keycloak initializes the database. At this point you should be able to access the GUI at:

```
<PUBLIC_ADDR>:<MNG_TOOLS_GUI_PORT>
```

The GUI should ask you to log in via Keycloak. If this is your first time running the project, you will need to access the Keycloak administration console to create some users under the realm <KEYCLOAK_REALM> with the appropriate roles at:

```
<PUBLIC_ADDR>:<KEYCLOAK_PORT>
```

4.4. Building (manual)

First of all, clone the repository and open the remap/node folder:

```
git clone https://scm.atosresearch.eu/ari/remap.git
cd remap/node
```

Set the tag for the images in your .env file (tipically sprint-XX, if no tag is defined, latest will be used) and build them using the build.yml file provided:

```
docker-compose -f build.yml build --pull
```

You can also push the images to Docker Hub in a similar manner:

```
docker-compose -f build.yml push
```

5. Deployment guide (core)

5.1. Prerequisites

You will need git, docker and docker-compose (version 3.1+) installed in your system.

5.2. Installing – production

First of all, clone the repository and open the remap/core folder:

```
git clone https://scm.atosresearch.eu/ari/remap.git
cd remap/core
```

In order to run the core infrastructure, you must create a .env file to set certain environment variables. We recommend you copy the env.prod.example template we provide.

```
cp .env.prod.example .env
```

Keycloak roles, clients and realm must be defined in a keycloak.env file. We recommend you copy the keycloak.env.example template we provide. Most probably you will not need to modify any of these fields.

```
cp keycloak.env.example keycloak.env
```

Sensitive parameters are passed to the container as docker secrets. We recommend you copy the secrets.example template folder we provide and just edit each file's contents to change the default values. Each secret is defined as a separate file. **Please make sure that your editor is not inserting a new line at the end of the file to prevent the different services from interpreting the new line character as part of your secret.**

Secret files .pypirc and .htpasswd contain the credentials for the (internal) pypi server. The .htpasswd file requires the password field to be encoded and can be created as described in this section of the pypiserver documentation [11].

The different exposed services all sit behind an nginx reverse proxy, which handles any incoming connection to the core services. In order to enable HTTPS (which you should), you must set the values of the following variables in your .env file:

```
...

FULLCHAIN_PEM=/path/to/fullchain.pem
PRIVKEY_PEM=/path/to/key.pem
DHPARAM_DH=/path/to/dhparam.pem

PUBLIC_HOSTNAME=your.domain

HTTPS_ENABLED=true
PUBLIC_ADDR=https://your.domain
PUBLIC_PORT=443

...
```

While deprecated, HTTP is also supported in case you absolutely cannot provide signed certificates:

```
...

FULLCHAIN_PEM=
PRIVKEY_PEM=
DHPARAM_DH=

PUBLIC_HOSTNAME=your.domain

HTTPS_ENABLED=false
PUBLIC_ADDR=http://your.domain
PUBLIC_PORT=<PORT>

...
```

Copy the production docker-compose template:

```
cp docker-compose.prod.yml docker-compose.yml
```

On some configurations, iptables might default to dropping connections from docker containers to the host. In order to enable ReMAP services to access each other and especially the identity by its public domain name, add the following rule to iptables:

```
iptables -I INPUT 1 -s 192.168.32.0/20 -j ACCEPT
```

Where 192.168.32.0/20 is the subnet specified for the default network in the docker.compose.prod.yml template.

Finally, run the following command, which will pull and set up the different services as docker containers:

```
docker-compose -p core up -d
```

You might need to wait a couple of minutes before Keycloak initializes the database. At this point you should be able to access the GUI at:

```
<PUBLIC_ADDR>:<PUBLIC_PORT>
```

The GUI should ask you to log in via Keycloak. If this is your first time running the project, you will need to access the Keycloak administration console to create a new user under the realm <KEYCLOAK_REALM> and give it ReMAP IT Staff' permisions at:

```
<PUBLIC_ADDR>:<PUBLIC_PORT>/auth
```

5.3. Installing – development environment

First of all, clone the repository and open the remap/core folder:

```
git clone https://scm.atosresearch.eu/ari/remap.git
cd remap/core
```

5.3.1. Automatic setup

Warning: All your existing configuration, running containers and volumes will be erased, both for node and core, use with caution.

If you wish to clean your local development environment and erase all your existing data, reset all configuration to the default values, and update everything with the latest templates, just run the cleanEnv.sh script.

5.3.2. Manual setup

In order to run the core infrastructure, you must create a .env file to set certain environment variables.

We recommend you copy the .env.dev.example template we provide.

```
cp .env.dev.example .env
```

Keycloak roles, clients and realm must be defined in a keycloak.env file. We recommend you copy the keycloak.env.example template we provide. Most probably you will not need to modify any of these fields.

```
cp keycloak.env.example keycloak.env
```

Sensitive parameters are passed to the container as docker secrets. We recommend you copy the secrets.example template folder we provide and just edit each file's contents to change the default values. Each secret is defined as a separate file. **Please make sure that your editor is not inserting a new line at the end of the file to prevent the different services from interpreting the new line character as part of your secret.**

Secret files .pypirc and .htpasswd contain the credentials for the (internal) pypi server. The .htpasswd file requires the password field to be encoded and can be created as described in this section of the pypiserver documentation [11].

Copy the development docker-compose template:

```
cp docker-compose.dev.yml docker-compose.yml
```

5.3.3. Local domain

In order to enable the different services to share a common hostname (so that a unique token issuer is seen by keycloak throughout the whole flow), you must edit the following variables in the .env file.

```
...

PUBLIC_HOSTNAME=<YOUR_MACHINE_HOSTNAME>
PUBLIC_ADDR=http://<YOUR_MACHINE_HOSTNAME>
PUBLIC_IP=<HOST_IP_IN_DOCKER_INTERFACE>

...
```

You can define a custom hostname in your /etc/hosts file. You can find your host's IP in the docker interface by running ifconfig and searching for the docker interface, tipically named docker0.

5.3.4. Starting the core

Finally, run the following command, which will build and set up the different services as docker containers:

```
docker-compose up -d --build
```

You might need to wait a couple of minutes before Keycloak initializes the database. At this point you should be able to access the GUI at:

```
<PUBLIC_ADDR>:<MNG_TOOLS_GUI_PORT>
```

The GUI should ask you to log in via Keycloak. If this is your first time running the project, you will need to access the Keycloak administration console to create some users under the realm <KEYCLOAK_REALM> with the appropriate roles at:

```
<PUBLIC_ADDR>:<KEYCLOAK_PORT>
```

5.4. Building (manual)

First of all, clone the repository and open the remap/core folder:

```
git clone https://scm.atosresearch.eu/ari/remap.git
cd remap/core
```

Set the tag for the images in your .env file (tipically sprint-XX, if no tag is defined, latest will be used) and build them using the build.yml file provided:

```
docker-compose -f build.yml build --pull
```

You can also push the images to Docker Hub in a similar manner:

```
docker-compose -f build.yml push
```

6. Developer guides

6.1. Debugging

In order to enable debugging of the different node apps when run packaged in a docker container, a .vscode folder with the appropriate configuration to connect VSCode's debugger to the different apps has been added to the root folder of the repository.

The docker-compose.dev.yml template contains the appropriate (commented) lines so that each service is run with the appropriate debug flags and exposes a debug port. By default, the process will pause in the first line and will wait until a debugger is attached, change --inspect-brk to --inspect if you don't want it to wait.

As an example, to debug the mng-tools-api service, one would have to uncomment the 9229 port mapping as well as the command section:

```
...

mng-tools-api:
  build:
    context: ./mngTools/api
  image: remapnode/mng-tools-api:${TAG:-latest}
  restart: unless-stopped
  ports:
    - ${MNG_TOOLS_API_PORT:?err}:3200/tcp
    # Uncomment this line for debugging
    # - 9230:9229
  environment:
    - TZ=${TZ}
    - KEYCLOAK_HOST=${PUBLIC_ADDR:?err}
    - KEYCLOAK_PORT=${KEYCLOAK_PORT:?err}
    - MONGO_HOST=${PUBLIC_HOSTNAME:?err}
    - MONGO_PORT=${MNG_TOOLS_DB_PORT:?err}
    - MONGO_INITDB_USERNAME_FILE=/run/secrets/mng-tools-db-username
    - MONGO_INITDB_PASSWORD_FILE=/run/secrets/mng-tools-db-password
    - MONGO_INITDB_DATABASE_FILE=/run/secrets/mng-tools-db-database
    - LOG_LEVEL=${LOG_LEVEL:-info}
  env_file:
    - keycloak.env
  secrets:
    - mng-tools-db-username
    - mng-tools-db-password
    - mng-tools-db-database
  volumes:
    - keycloak-mng-tools-api-config:/keycloak-config:ro
  extra_hosts:
    - ${PUBLIC_HOSTNAME:?err}:${PUBLIC_IP:?err}
  # Uncomment these 4 lines for debugging
  # command:
  #   - node
  #   - "--inspect-brk=0.0.0.0"
  #   - "."

...
```

6.2. Adding a new service

In order to add an additional service to the existing architecture, please refer to the following guidelines. While this example is focused on the node, everything can be extrapolated to the core.

6.2.1. Declare the service in the docker-compose files

At the moment we are using 3 different .yml files, where you must declare the new service:

- docker-compose.prod.yml: A file describing the production setup, where the different containers are exposed to the internet via a reverse proxy
- docker-compose.dev.yml: A file describing the development setup, where each container directly exposes a port
- build.yml: A file describing how to build and tag each of the images

6.2.2. Route the new service through the nginx reverse proxy (for production deployment)

If your service needs to expose any endpoint to the public internet, then you must update the nginx reverse proxy configuration so that a specific path is mapped to your backend service in the following files:

- In upstreams.conf, you must define an upstream to your backend service, specifying the service_name declared in the docker-compose files, and the port exposed by your container.

```
upstream <service_upstream> {
    server  <service_name>:<port>;
}
```

- In reamp.conf, you must add the following location block:

```
location /path/to/your/service {
    proxy_pass  http://<service_upstream>;
    include     /etc/nginx/proxy.conf;
}
```

6.2.3. Update .env files

In case your service needs new environment variables, these should be define in a file rather than directly in the docker-compose files directly, for the sake of maintainability. Try to reuse any existing variables instead of defining new ones where possible, in order to avoid redundancy.

6.2.4. Keycloak authentication

If your service needs to protect an endpoint or communicate with another service's protected endpoint, then you will need to modify the keycloak-initializer service so that it creates a client and credentials for your service, as well as configure the token exchange policies.

The strategy followed to share the client secret generated with each service is to define a docker volume keycloak-<service>-config to be shared between the keycloak-initializer and the corresponding service.

If your service must obtain a token for itself to access any protected endpoints, then you must set the serviceAccountsEnabled field to true while creating the client. Otherwise you can let it default to false (see mng-tools-api client creation). GUI clients, which are supposed to obtain user tokens, must set the publicClient field to true (see mng-tools-gui client creation).

Summing up, you should add an adjusted version of this code to the promise chain in keycloak/app.js:

```
...
    .then(() => {
        log.debug(`Creating ${DATA_ADAPTOR_API} client`);
        return kcAdmin.clients.create({
            clientId: DATA_ADAPTOR_API,
            rootUrl: getUrl(),
            adminUrl: getUrl(),
            redirectUris: [`${getUrl()}/` + '*'],
            webOrigins: [getUrl()],
            directAccessGrantsEnabled: true,
            fullScopeAllowed: true,
            publicClient: false,
            serviceAccountsEnabled: true
        });
    })
    .then(client => {
        log.debug(`Retrieving ${DATA_ADAPTOR_API} client`);
        return kcAdmin.clients.findOne({ id: client.id });
    })
    .then(client => {
        log.debug(JSON.stringify(client, null, 2));
        dataAdaptorApiClient = client;
        log.debug(JSON.stringify(dataAdaptorApiClient, null, 2));
    })
    .then(() => {
        log.debug(`Retrieving secret for ${DATA_ADAPTOR_API}`);
        return kcAdmin.clients.getClientSecret({ id: dataAdaptorApiClient.id });
    })
    .then(secret => {
        log.debug(`Storing secret for ${DATA_ADAPTOR_API}`);
        return storeSecret({
            folder: DATA_ADAPTOR_API_FOLDER,
            secret: secret.value
        });
    })
...
```

As well as modify one of these existing calls to configure token exchange, or add a new one, depending on how will your service interact with the rest:

```
...
        realmManagementClient.authorizationSettings = configureTokenExchange({
            authorizationSettings: realmManagementClient.authorizationSettings,
            starting: [dataAdaptorCronClient],
            target: dataAdaptorApiClient
        });

        realmManagementClient.authorizationSettings = configureTokenExchange({
```

```
            authorizationSettings: realmManagementClient.authorizationSettings,
            starting: [dataAdaptorApiClient],
            target: localRepositoryClient
        });
...
```

6.2.5. Protecting an API

In case you need to add and protect a new API, we encourage you to have a look at some of the exiting APIs for reference. mng-tools-api module might be a good place to start. As you can see in app.js, securizing the API is pretty straight-forward.

This section is not intended to be an in-depth guide on how to implement a protected API, but rather some guidelines on how to adapt it to your project.

The files required via folder-mapper are meant to be reused in your project with little to no modifications (mostly the file config/keycloak.js, where you will need to edit environment variable defining the Keycloak client to use, from keycloak.env). These files can be found in the app and config folders.

- Required modules

const express = require('express'); // Express Web Server

```
const bodyParser = require('body-parser');  // Express body parser
const map = require('folder-mapper');       // Folder mapping
const Keycloak = require('keycloak-connect');
const session = require('express-session');

// Secrets reader service
const secrets = require(map.getService('secrets'));

// Keycloak config class
const kcConfig = require(map.getConfig('keycloak'));

// Controllers
const cors = require(map.getController('cors'));
const apiConnection = require(map.getController('api/connection'));
...
```

- Create the express server and configure session & CORS

```
...

// Read the client secret
const SECRET = secrets.get('/keycloak-config/secret');

// Initialize the express web server
const app = express();

// Session
const memoryStore = new session.MemoryStore();
app.use(session({
  secret: SECRET,
  resave: false,
```

```
  saveUninitialized: true,
  store: memoryStore
}));

// Keycloak middleware
const keycloak = new Keycloak({ store: memoryStore }, kcConfig.getKcOpts());

// Parse body as json
app.use(bodyParser.json());

// CORS
app.use(cors);

...
```

- Finally define your API path and add the required middlewares required for keycloak and token exchange. You can define your own protection filter as shown below:

```
...

/**
 * Create route /connection
 */
app.use(
  '/mngTools/api/connection',
  keycloak.middleware(),
  keycloak.protect(token => {
    let authorized = false;

    if (token.hasRealmRole(process.env.KC_REMAP_ROLE_AIRLINE_IT_STAFF)) {
      authorized = true;
    } else if (token.content.azp === process.env.DATA_ADAPTOR_CRON_KEYCLOAK_CLIENT) {
      authorized = true;
    }

    return authorized;
  }),
  apiConnection
);

...
```

7. Conclusions

This document has explained the CI/CD and deployment strategy followed by the ReMAP team, and together with D2.3 & D2.4 it should give the reader the necessary tools to understand how to install and even integrate, modify and adapt the platform to accommodate new modules or changes to the existing ones, reinforcing the idea that ReMAP has been built with modularity and extensibility in mind at all times.

The deployment guides showcase how much the docker-compose and configuration templates provided have simplified the actual deployment process to just copy the necessary files to the server, edit a minimal set of parameters for the machine (such as address or path to certificates for HTTPS), and just run the docker-compose script for everything to be pulled from the repo and deployed for you. This straight-forward approach has reduced the required time to install the platform significantly from the inital goal, to the point where 1 day of work should be more than enough in the worst case.

8. References

[1] Atos, “D2.3 IT platform architecture,” H2020 EU project ReMAP, 2021.

[2] Atos, “D2.4 IT platform code & technical documentation,” H2020 EU project ReMAP, 2022.

[3] Atos, “ReMAP GitLab repository,” [Online].

[4] GitLab, “GitLab CI/CD,” [Online]. Available: https://docs.gitlab.com/ee/ci/.

[5] Atos, “ReMAP Core Docker Hub repository,” [Online]. Available: https://hub.docker.com/u/remapcore.

[6] Atos, “ReMAP Node Docker Hub repository,” [Online]. Available: https://hub.docker.com/u/remapnode.

[7] WiredTiger, “WiredTiger: Compressors,” [Online]. Available: https://source.wiredtiger.com/3.1.0/compression.html.

[8] Facebook, “Zstandard - Real-time data compression algorithm,” [Online]. Available: https://facebook.github.io/zstd/.

[9] Fcron, “Fcron: a periodical command scheduler for Unix and Linux systems,” [Online]. Available: http://fcron.free.fr/.

[10] V. Cron, “Vixie Cron Github repository,” [Online]. Available: https://github.com/vixie/cron.

[11] pypiserver, “Pypiserver documentation - Apache-Like Authentication (htpasswd),” [Online]. Available: https://github.com/pypiserver/pypiserver#apache-like-authentication-htpasswd.