Georchestra - MapStore Documentation

GeoSolutions S.a.s.

Apr 03, 2020

Contents

CHAPTER 1

MapStore Guide

The official documentation of MapStore is available here.

CHAPTER 2

MapStore Integration in geOrchestra

This section describes how to include MapStore in your geOrchestra installation.

All the steps assumes you are working from the project main directory.

2.1 Edit docker-compose.yml

Add the following snippet to the docker-compose.yml file:

```
mapstore:
image: geosolutionsit/mapstore2-georchestra:latest
depends_on:
  - database
volumes:
  - ./config:/etc/georchestra
environment:
  - JAVA_OPTS=-Xms512m -Xmx512m -Dgeorchestra.datadir=/etc/georchestra
```

2.2 Configure the 'proxy' container to forward requests to MapStore

Add the following to the config/security-proxy/targets-mapping.properties file:

```
analytics=http://analytics:8080/analytics/
atlas=http://atlas:8080/atlas/
console=http://console:8080/console/
extractorapp=http://extractorapp:8080/extractorapp/
geonetwork=http://geonetwork:8080/geonetwork/
geoserver=http://geoserver:8080/geoserver/
header=http://header:8080/header/
mapfishapp=http://mapfishapp:8080/mapfishapp/
mapstore=http://mapstore:8080/mapstore/
```

2.3 Create config directory for MapStore

```
mkdir config/mapstore/
```

2.4 Add geostore override file to user PostgreSQL

Create the config/mapstore/geostore-datasource-ovr.properties with the following content:

```
#Default Postgres Connection values, uncomment for using postgres
geostoreDataSource.driverClassName=org.postgresql.Driver
geostoreDataSource.url=jdbc:postgresql://database:5432/geostore
geostoreDataSource.username=geostore
geostoreDataSource.password=geostore
geostoreVendorAdapter.databasePlatform=org.hibernate.dialect.
↪PostgreSQLDialect
geostoreEntityManagerFactory.jpaPropertyMap[hibernate.hbm2ddl.auto]=validate
geostoreEntityManagerFactory.jpaPropertyMap[hibernate.default_
↪schema]=geostore
geostoreVendorAdapter.generateDdl=true
geostoreVendorAdapter.showSql=false
```

NOTE: update the values to match your environment

2.5 Setup Database

1. Jump into the PostgreSQL container and create GeoStore Role and DB

```
docker exec -it docker_database_1 bash
su postgres
psql -U georchestra
CREATE ROLE geostore WITH LOGIN PASSWORD 'geostore';
CREATE DATABASE geostore WITH OWNER 'geostore';
```

2. Create schemas and tables as per GeoStore documentation

2.6 Recreate MapStore Container to pick up the change

```
docker-compose up -d --force-recreate mapstore
```

2.7 Update “header” application index to link to Mapstore

1. Extract the “header” application war from the docker image

```
docker cp docker_header_1:/var/lib/jetty/webapps/header.war ./header.war
```

2. In the war file, update the WEB-INF/classes/_header/i18n/index_en.properties file adding the following:

```
mapstore=mapstore
```

3. Repeat for all the other translation files
4. In the war file, update the WEB-INF/jsp/index.jsp file adding the following:

```
<c:choose>
    <c:when test='<%= active.equals("mapstore") %>'>
    <li class="active"><a><fmt:message key="mapstore"/></a></li>
    </c:when>
    <c:otherwise>
    <li><a href="/mapstore/"><fmt:message key="mapstore"/></a></li>
    </c:otherwise>
</c:choose>
```

5. In the war file, update the WEB-INF/jsp/index.jsp file with the following:

```
...
Boolean extractorappadmin = false;
Boolean msadmin = false;
String sec_roles = request.getHeader("sec-roles");
if(sec_roles != null) {
    String[] roles = sec_roles.split(";");
    for (int i = 0; i < roles.length; i++) {
        if (roles[i].equals("ROLE_GN_EDITOR") || roles[i].equals("ROLE_GN_
↪REVIEWER") || roles[i].equals("ROLE_GN_ADMIN") || roles[i].equals("ROLE_
↪ADMINISTRATOR") || roles[i].equals("ROLE_USER")) {
            anonymous = false;
        }
        if (roles[i].equals("ROLE_SUPERUSER")) {
            admin = true;
            console = true;
        }
        if (roles[i].equals("ROLE_ORGADMIN")) {
            admin = true;
            console = true;
        }
        if (roles[i].equals("ROLE_GN_ADMIN")) {
            admin = true;
            catadmin = true;
        }
        if (roles[i].equals("ROLE_ADMINISTRATOR")) {
            admin = true;
            extractorappadmin = true;
        }
        if (roles[i].equals("ROLE_MAPSTORE_ADMIN")) {
            admin = true;
            msadmin = true;
        }
    }
}
...
```

6. In the war file, update the WEB-INF/jsp/index.jsp file adding the following:

```
<c:choose>
    <c:when test='<%= msadmin == true %>'>
    <c:choose>
```

(continues on next page)

(continued from previous page)

```
        <c:when test='<%= active.equals("msadmin") %>'>
    <li class="active"><a><fmt:message key="mapstore"/></a></li>
        </c:when>
        <c:otherwise>
    <li><a href="/mapstore/#/admin"><fmt:message key="mapstore"/></a></li>
        </c:otherwise>
    </c:choose>
    </c:when>
</c:choose>
```

7. Finally put the edited war back to the container:

```
docker cp header.war docker_header_1:/var/lib/jetty/webapps/header.war
```

CHAPTER 3

Developer Guide

To start using the MapStore geOrchestra project as a developer you need the following:

- install the needed requirements: * NodeJS (>=8) * JDK (>= 8) * Maven (>= 3.x)
- clone the GitHub repository:

```
git clone --recursive https://github.com/georchestra/mapstore2-georchestra
```

- from the cloned source, install the dependencies from the npm registry:

```
npm install
```

- do a full build using the build script:

```
./build.sh
```

3.1 Configuring the backend

To develop locally you will need to use a proxied backend. To configure your backend of choice you need to properly change the webpack.config.js file, in particular you need to change the following variables:

- DEV_PROTOCOL: http or https
- DEV_HOST: host and port of the backend

```
const DEV_PROTOCOL = "http";
const DEV_HOST = "localhost:8080";
```

You can either:

- use an online backend
- deploy and run your just build backend on a Tomcat instance

To deploy your local backend you will need to:

- copy the mapstore.war from web/target to your Tomcat webapps folder
- create a local geOrchestra datadir anywhere in your PC and copy the following inside it:
- a standard geOrchestra default.properties file with generic configuration (database and LDAP settings for example)
- the datadir/mapstore folder from web/target/geOrchestra with the mapstore specific configuration files
- add the georchestra.datadir environment variable to the Tomcat setenv script to point to your datadir folder

```
-Dgeorchestra.datadir=/etc/georchestra
```

- properly change the configuration files, in particular to set the database and LDAP repository connection settings

If you don't have a local database and LDAP repository properly configured for geOrchestra you can use remote ones. Remember: to use a local backend both a PostgreSQL database and LDAP repository needs to be available and properly populated.

3.2 Developing the frontend

To start the frontend locally, just run:

```
npm start
```

Your application will be available at http://localhost:8081

3.3 Mocking security

When working locally you won't have the security proxy authentication enabled, but you can simulate it using a specific Chrome extension called ModHeader.

Install this extension and configure it to set the following request headers:

- sec-username: the username logged in
- sec-roles: a semicolon delimited list of roles (e.g. ROLE_MAPSTORE_ADMIN)

Remember to disable the extension when you don't need it.

CHAPTER 4

Configuration Guide

geOrchestra MapStore can be configured using the geOrchestra configuration directory.

The configuration directory is enabled through the MapStore datadir functionality. For more information on this, please look at the official MapStore documentation here

geOrchestra MapStore enables the MapStore datadir and in addition, uses the standard geOrchestra default.properties file from the geOrchestra configuration directory to inherit all the shared configuration (e.g. the database connection settings, LDAP connection settings and so on).

More in detail:

- a lot of shared configuration properties are read from the geOrchestra default.properties file
- a MapStore subfolder contains MapStore specific configuration files:
  - geostore.properties: MapStore specific database configuration properties (not contained in default.properties, e.g. the database schema used by MapStore for its persistence layer)
  - proxy.properties: MapStore proxy configuration (see here: https://github.com/geosolutions-it/http-proxy/wiki/Configuring-Http-Proxy) for further details)
  - log4j.properties: logging configuration (see here: https://logging.apache.org/log4j/2.x/manual/configuration.html#ConfigurationSyntax>)
  - localConfig.json: main frontend configuration file, in JSON format (see here: https://mapstore.readthedocs.io/en/latest/developer-guide/local-config/)
  - extensions.json: dynamic registry of currently installed extensions, in JSON format
  - pluginsConfig.json: dynamic registry of available plugins (both standard and extensions) for the context configurator, in JSON format
  - printing: folder with mapfish-print configuration files, config.yaml and related resource (see here: https://github.com/geosolutions-it/mapfish-print/wiki>)
  - dist subfolder: will contain all the dynamically uploaded extensions, one folder for each of them, with all the extension assets (javascript bundle, translations, etc.)

If the datadir is not configured / used in a particular environment, default configurations will be applied.

If the datadir is configured, but some of the above mentioned files are missing from it, a default fallback will be used.

This allows the administrator to find a good compromise between two conflicting needs:

- customizing your geOrchestra MapStore installation
- allow an easy upgrade to newer versions

It is important to understand that if a configuration file is loaded from the datadir, it will not be upgraded when a new version of the application is installed, and any necessary upgrades should be done manually. So, our advice is: put a configuration file in the datadir only if it is dynamically changed from the application or you need to customize it.

Dynamic files are for example:

- extensions.json
- pluginsConfig.json
- the dist subfolder

These files are updated by the extensions upload functionality, so they have to be put in the datadir, or they will be removed at every application update.

All the other files should be put in the datadir only if you need to customize them. A particular attention is needed for localConfig.json: this is where the available MapStore plugins are registered, so, if you copied it in the datadir, you will need to manually add new plugins when you upgrade to a new version.

More details about configuration aspects can be found in the following sections:

4.1 Database Configuration

MapStore uses the geOrchestra PostgreSQL database to store resources saved by the application (maps, contexts, etc.). A specific schema, called geostore, is used for this purpose. The schema can be created / populated using the SQL scripts in the source code database folder here.

The database connection settings are taken from the geOrchestra default.properties configuration file, and mapped to internal configuration variables (e.g. ${pgsqlHost}).

In addition to that a geostore.properties file in datadir/mapstore is used for MapStore specific settings:

- pgsqlGeoStoreSchema: schema used for the MapStore permissions database (Defaults to geostore)

To configure the default.properties location the default georchestra environment variable is used (georchestra.datadir). For local development, this must be configured for the JVM:

```
-Dgeorchestra.datadir=/etc/georchestra
```

4.2 Security Integration

MapStore is integrated with the geOrchestra security infrastructure. This happens thanks to:

- an authentication filter using the geOrchestra security proxy headers to authenticate the user and assign proper MapStore groups and roles
- LDAP enabled DAOs to get available roles from the geOrchestra LDAP repository

4.2.1 Authentication Filter

The authentication filter intercepts every MapStore backend request to extract the headers forwarded by the geOrchestra security-proxy, and use them to properly authenticate and authorize the current user, in particular:

- sec-username is used to authenticate the current user (anonymous access is assigned if the header does not exist)
- sec-roles is used to assign MapStore groups to the current user (groups will be used by the admin to assign permissions for the MapStore resources, e.g. maps)
- a particular role (MAPSTORE_ADMIN) is mapped to the MapStore ADMIN role

The filter is configured in the geostore-security-proxy.xml file:

```
<security:http auto-config="true" create-session="never" >
            ...
    <!-- include filter to capture geOrchestra security proxy headers -->
            <security:custom-filter ref="headersProcessingFilter" before=
↪"FORM_LOGIN_FILTER"/>
            ...
    </security:http>

<!-- GeOrchestra header based Auth Provider -->
<bean id="georchestraAuthenticationProvider"
            class="it.geosolutions.geostore.services.rest.security.
↪PreAuthenticatedAuthenticationProvider">
    </bean>

<!-- GeOrchestra header based Auth Filter -->
<bean class="it.geosolutions.geostore.services.rest.security.
↪HeadersAuthenticationFilter"
    id="headersProcessingFilter">
        <property name="addEveryOneGroup" value="true"/>
        <property name="usernameHeader" value="sec-username"/>
        <property name="groupsHeader" value="sec-roles"/>
        <property name="listDelimiter" value=";"/>
        <property name="authoritiesMapper" ref="rolesMapper"/>
    </bean>

<bean id="rolesMapper" class="it.geosolutions.geostore.core.security.
↪SimpleGrantedAuthoritiesMapper">
    <constructor-arg>
        <map>
            <!-- add more entries to map other roles to MapStore ADMIN -->
            <entry key="ROLE_MAPSTORE_ADMIN" value="ADMIN"/>
        </map>
    </constructor-arg>
</bean>
```

4.2.2 LDAP Integration

MapStore is integrated with the geOrchestra LDAP repository, to be able to fetch users and roles information and use it in the Admin UI, to assign permissions to MapStore resources and functionalities (maps, contexts, etc.).

This ia also configured in the geostore-security-proxy.xml file:

```
<!-- GeOrchestra LDAP DAOs -->
<bean id="ldap-context" class="org.springframework.security.ldap.
↪DefaultSpringSecurityContextSource">
```

(continues on next page)

(continued from previous page)

```
            <constructor-arg value="${ldapScheme}://${ldapHost}:${ldapPort}/$
↪{ldapBaseDn}" />
    <property name="userDn" value="${ldapAdminDn}"/>
    <property name="password" value="${ldapAdminPassword}"/>
    </bean>
<bean id="ldapUserDAO" class="it.geosolutions.geostore.core.dao.ldap.impl.
↪UserDAOImpl">
    <constructor-arg ref="ldap-context"/>
    <property name="searchBase" value="${ldapUsersRdn}"/>
    <!-- membership attribute (member) has the syntax uid=username,ou=users,.
↪.. -->
    <property name="memberPattern" value="^uid=([^,]+).*$"/>
    <property name="attributesMapper">
        <map>
            <!-- optional, LDAP attribute to internal user attribute -->
            <entry key="mail" value="email"/>
            <entry key="givenName" value="fullname"/>
            <entry key="description" value="description"/>
        </map>
    </property>
</bean>
<bean id="ldapUserGroupDAO" class="it.geosolutions.geostore.core.dao.ldap.
↪impl.UserGroupDAOImpl">
    <constructor-arg ref="ldap-context"/>
    <property name="searchBase" value="${ldapRolesRdn}"/>
    <property name="addEveryOneGroup" value="true"/>
</bean>
<alias name="ldapUserGroupDAO" alias="userGroupDAO"/>
<alias name="ldapUserDAO" alias="userDAO"/>
```

LDAP connection settings are taken from the geOrchestra default.properties configuration file, and mapped to internal configuration variables (e.g. ${ldapHost}).

To configure the default.properties location the default georchestra environment variable is used (georchestra.datadir). For local development, this must be configured for the JVM:

```
-Dgeorchestra.datadir=/etc/georchestra
```

Here a diagram of how the various pieces work together:

Here some of the most important MapStore workflows and their relation to the security infrastructure:

4.3 Header Configuration

MapStore includes the geOrchestra header application on top of the map viewer pages.

Some configuration properties for the header are taken from the geOrchestra default.properties configuration file.

In particular the following are used:

- headerHeight: height of the header app (Defaults to 90px)
- headerUrl: url of the header app (Defaults to /header/)

To configure the default.properties location the default georchestra environment variable is used (georchestra.datadir). For local development, this must be configured for the JVM:

```
-Dgeorchestra.datadir=/etc/georchestra
```

CHAPTER 5

Localize this documentation

To localize this documentation install sphinx-intl:

```
sudo  pip install sphinx-intl
```

Every time you have to update the translation files you have to update the .po files running the following commands:

```
cd docs # all commands must run in docs directory
make gettext # generates .pot files
sphinx-intl update -p build/gettext -l fr # generate .po files for fr lang
```

Then you can edit the .po files and commit them

To generate the documentation locally for the you can run (on linux)

```
sphinx-build  -b html -D language=fr source build/html/fr
```

This will generate *mo* files that should be ignored in .gitignore