Georchestra - MapStore Documentation

GeoSolutions S.a.s.

Feb 09, 2022

Contents

This documentation provides an overview of how to use and configure MapStore inside geOrchestra.

CHAPTER 1

MapStore Guide

MapStore is a framework to develop WebGIS applications that can provide advanced viewing tools for maps and geo-spatial data. geOrchestra includes MapStore as the main map viewer for data visualization and sharing.

The official documentation of MapStore is available here. The documentation offers general information about the tools and the pages available in the main MapStore product. Since geOrchestra uses MapStore to build its own MapViewer and MapViewer administration interface, excluding all the other parts of the original product (home page, dashboards, stories...) some of the information in the official documentation may differ from what you will find in geOrchestra.

Anyway you can find useful hints looking at it about some of the tools available in geOrchestra mapstore viewer.

Here the links to the interesting sections for user or for administration UI:

- The main map viewer (see the Exploring Maps section of the official mapstore documentation for an overview of the main functionalities.
- The *context editor* as administration interface: (see Managing Contexts Section for a general overview of the functionalities of this UI.

CHAPTER 2

Configuration Guide

geOrchestra MapStore can be configured using the geOrchestra configuration directory.

All basic configurations for the database are already configured and loaded from the `default.properties` file inside the root of the configuration directory.

The configuration directory sub-folder `mapstore`, related to MapStore, contains several files dedicated to basic settings, integration with geOrchestra, application configuration and advanced setup.

In this documentation you will learn how to configure in detail the `default.properties` entries and the files in the `mapstore` folder.

2.1 General

MapStore uses the default geOrchestra environment variable `georchestra.datadir` to identify the `default.properties` file location and uses it for it's basic configurations (e.g. the database connection settings, LDAP connection settings and so on).

Moreover, MapStore enables it's own datadir inside the `georchestra.datadir` sub-folder called `mapstore`, to handle it's own configuration files.

2.2 Database Configuration

MapStore uses the geOrchestra PostgreSQL database to store resources saved by the application (maps, contexts, etc.). A specific schema, called `mapstore`, 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 (using for instance `${pgsqlHost}`, etc...).

2.2.1 Overriding Database configuration for MapStore

In `mapstore/geostore.properties` there is a dedicated configuration file that contains the settings that can not be loaded from `default.properties`. It contains only one setting that indicates the database schema to use for MapStore.

- `pgsqlGeoStoreSchema`: schema used for the MapStore database (Defaults to `mapstore`)

This file can be used to override connection parameters defined in `default.properties`, to configure MapStore in a different way from the rest of the infrastructure.

2.3 Application Configuration

The `mapstore` folder inside the geOrchestra configuration directory contains several files dedicated to the configuration of various parts of MapStore. This sections contain information about the configuration files contained and how they are managed by MapStore.

2.3.1 Back End services

MapStore includes some back-end services that provides some special functionalities (printing, proxy). They can be configured by editing some special files the the `mapstore` directory.

2.3.1.1 Printing

MapStore provides a printing module, that is an extension of mapfish print (v2) with many customizations, included in geOrchestra by default. The directory `mapstore/printing` is dedicated to the setup of this module. More information about how to configure these files is available here.

2.3.1.2 Proxy

MapStore brings an internal secure proxy to give support request to external services (WMS, WMTS ...). The file `proxy.properties` contains all the configuration for this file. More information about configuring this file is available here.

2.3.2 Map Viewer

To configure the map viewer(plugins, context editor...), the administrator can edit different files dedicated to the different parts of the application. If some of these files are not present, the application will take automatically the files from the original *mapstore* webapp.

The files are:

2.3.2.1 `localConfig.json`

This is the main frontend configuration file. It contains the main settings of the whole MapStore viewer, including the list of plugins shown in the main viewer, and the tools of the administration page. You can edit the `plugins` section of this file to customize the plugins inside the main viewer, as well as you do from the context-editor UI for other contexts.

See here for more detail about the configuration of this file.

2.3.2.2 config.json

This configuration file contains the settings of the default map. The administrator can edit this file to change the default layers (e.g. backgrounds) or initial map position.

See here to learn more about the format of this file.

2.3.2.3 new.json

This is the initial map configuration used for new contexts in the context editor.

See here to learn more about the format of this file.

2.3.2.4 pluginsConfig.json

This is the registry of the available plugins in JSON format. The administrator can edit this file to change the default list of plugins in the context editor plugin selection section. See here to learn more about the format of this file).

2.3.3 Advanced Map Viewer Configuration

Next sections describe some advanced functionalities that can be applied **only** to the Map Viewer settings (`json` files) and some information about the extension system and its relation with the configuration directory of MapStore.

2.3.3.1 Patch Files

Configuration files with plugins like `localConfig.json` and `pluginsConfig.json` may change a lot from one update to another (introducing new plugins, deprecating other, adding or removing old settings). To maintain changes (manually applied by administrator or automatically applied by the application) across different versions of the application, MapStore provides a *patch system* that allows to preserve configuration files and keep modification in a separate one.

For this reason in the data directory you may add or find some `.patch` files (generated by the application, but administrators can edit them on their own) containing only the differences between the original file and the final configuration we want to obtain (see here for more information about patching system in MapStore). The final version of the file is provided directly by the application via `http` when the configuration file is required.

2.3.3.2 Dynamic Files

MapStore allows to install extensions from the UI (administration section, when editing a new context). When a new plugin is installed, several files will be written in the data-directory.

- `extensions.json`: dynamic registry of currently installed extensions, in JSON format
- `dist` subfolder: will contain all the dynamically uploaded extensions, one folder for each of them, with all the extension assets (javascript bundles, translations, etc.)
- `pluginsConfig.json.patch`: a **patch** file for new plugins installed from the UI.

These files are all updated and managed by the extensions upload functionality.

To set a different folder for these files, you have to set the `georchestra.extensions` JVM option to the desired path. If not set, also dynamic files will be stored in the standard configuration directory.

2.3.3.3 Read/Write configuration directory

Since MapStore needs to write in the data directory and the administrator may want to edit configuration files on its own, MapStore provides also a system of **multiple configuration directories**, that makes possible to separate the manual changes from the automatic ones.

In particular an administrator can configure more then one `georchestra.datadir` values, separated by comma (see here for specific MapStore implementation details about this part).

The configuration files read/write rules are the following:

- Reading the configuration, MapStore will search for the first file found, looking in order in every directory provided. If the file is not found in any directory, MapStore will take the one present in the webapp.
- Writing operation will be applied **only** in the **first** directory of the list.

geOrchestra can be configured to have a **write** and a **read-only** configuration directory simply by giving 2 directories in the `georchestra.datadir` value. MapStore will write only in the first, the second one will be a read-only configuration dir, that can be edited manually by the administrator.

2.4 Other Configurations

This section contains some other miscellaneous information and configurations that are available for MapStore.

2.4.1 General information about integration between MapStore and geOrchestra

geOrchestra MapStore can be configured using the geOrchestra configuration directory.

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

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. The fallback will look for files in the web application root folder (webapps/mapstore), so the configurations from the original MapStore war file 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 you need to customize it. 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.

2.4.2 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 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 Integration

The MapStore integration inside geOrchestra SDI involves the following parts:

- Header (and header localization)
- Database Creation scripts
- Docker
- Data Dir
- Github Workflows

3.1.1 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

3.1.1.1 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="MAPSTORE_ADMIN" value="ADMIN"/>
        </map>
    </constructor-arg>
</bean>
```

3.1.1.2 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">
            <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:

3.2 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.

3.3 Developing the frontend

To start the frontend locally, just run:

```
npm start
```

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

3.4 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

Documentation Guide

This documentation uses Sphinx. In this section you will find out how to build and localize this documentation.

It is configured to be deployed in multi-language environment using read the docs.

4.1 Requirements

To build the documentation you need:

- `python` and `pip` installed
- `make` installed. You can install it using *sudo apt-get install build-essential* on linux.
- You will need also to install these extensions, using *pip*

```
pip install sphinx
pip install sphinx_rdt_theme
pip install recommonmark
```

4.2 Building

The main build, multi-language, is made by readthedocs.org. You can build the documentation **in english**, locally for testing, you can run:

```
cd docs # change directory in docs folder
make html # builds html version of the documentation
```

See the next section to see how to build the documentation localized.

4.3 Localizing

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