Adding netCDF to Geoserver

GeoServer is an open source server for sharing geospatial data. It is often used to publish shapefiles and PostGIS tables (Vector data), ArcGrid and GeoTiff (Raster data). Moreover, it also allows for configuration of (cascading) remote WMS. Deltares maintains its own instance of GeoServer here. Data layers from GeoServer can be exposed in via WMS/WFS in many standard formats for subsequent public use. As NetCDF data format is widely used by the scientific community, ingesting that data through GeoServer tools represents a valid and useful alternative of data dissemination. Additional information on NetCDF data format can be found here.

This tutorial refers to GeoServer <stable> version 2.8.2 on Windows. Since the stable version of GeoServer does not allow ingestion of NetCDF as input Coverage Format, Geoserver Extension can be installed from http://ares.boundlessgeo.com/geoserver/ (Select corresponding extension version to Geoserver version downloaded on machine). Once the extension has been downloaded, the .jar files have to be copied to `GeoServer 2.8.2 \webapps\geoserver\WEB-INF\lib\`. For any of the following steps, the easiest way to work with NetCDF Coverage input is to use the user interface provided by GeoServer installation. However, for reasons of efficiency and process automation a client might prefer to use the REST API GeoServer provides. The REST configuration allows you to work with bindings from various programming languages or simply with cURL command line tool. Python bindings are also available from gsconfig.py, but there is no implementation of working with NetCDF files yet.

Adding a workspace

This is the first step before adding any resource, so that your project is kept tidy and structured. That has to be done prior to the store creation for our NetCDF resources.

GeoServer User interface

First of all, create and add a workspace.

RESTful interface

Add a workspace called `ws`.

Python | gsconfig.py

```
import os
from geoserver.catalog import Catalog

geoserverRestURL = "http://localhost:8090/geoserver/rest/"
username = "admin"
password = "geoserver"
workspace = 'ws'

cat = Catalog(geoserverRestURL, username, password)
ws = cat.create_workspace(workspace, workspace)

# # delete workspace
# ws = cat.get_workspace(workspace)
# cat.delete(ws)
```

cmd | cURL

```
curl -u admin:geoserver -v -POST -H "Content-type: text/xml" -d "<workspace><name>ws</name></workspace>"
"http://localhost:8090/geoserver/rest/workspaces"
```

Python | requests

```
import os
import requests

workspace = 'ws'

headers_xml = {'content-type': 'text/xml'}
headers_zip = {'content-type': 'application/zip'}
headers_sld = {'content-type': 'application/vnd.ogc.sld+xml'}

r_create_workspace = requests.post("http://localhost:8090/geoserver/rest/workspaces",
    auth=('admin', 'geoserver'),
    data='<workspace><name>' + workspace + '</name></workspace>',
    headers=headers_xml)
```

Adding a NetCDF coverage store

At the moment of writing this tutorial, the NetCDF plugin supports datasets where each variable's axis is identified by an independent coordinate variable, therefore two dimensional non-independent latitude-longitude coordinate variables aren't currently supported.

GeoServer User interface

Once the workspace is created, you can add a NetCDF data store by following instructions and requirements on this page.

RESTful interface

Add a store called `st`. If you have a GeoTIFF, using Python bindings, `cat.create_coveragestore()` is sufficient. As Python bindings do not support adding NetCDF data stores yet, cURL commands and Python requests are shown only.

cmd | cURL

```
REM Coordinate reference system CRS can be added as well, although not compulsory, under nativeCRS tags.
Example: ...<nativeCRS>GEOGCS...</nativeCRS><srs>EPSG:4326</srs>...
curl -u admin:geoserver -v -POST -H "Content-type: text/xml" -d "<coverageStore><name>st</name><workspace>ws<
/workspace><enabled>true</enable><type>NetCDF</type><url>d:\\YOURPATH\\data.nc</url></coverageStore>"
"http://localhost:8090/geoserver/rest/workspaces/ws/coveragestores?configure=all"
```

Python | requests

```
coveragestore = 'st'

r_create_coveragestore = requests.post('http://localhost:8090/geoserver/rest/workspaces/ws/coveragestores?
configure=all',
    auth=('admin', 'geoserver'),
    data='<coverageStore><name>' + coveragestore + '</name><workspace>' + workspace + "<
/workspace><enabled>true</enabled><type>NetCDF</type><url>" + "d:\\YOURPATH\\data.nc" + '</url><
/coverageStore>',
    headers=headers_xml)
```

Adding a layer

GeoServer User interface

Adding a layer with GeoServer interface does not differ form adding any other type of layer. You can find a tutorial here.

RESTful interface

Before creating a layer, the `.nc` file has to be zipped so that it can be transferred to a Web server via `application/zip` content type. The zip file will be opened in a folder inside the GeoServer data folder. Ancillary files will be created in that exact folder. Later on, the zip file can be deleted.

```
import zipfile

# zip the nc file into a zip
zfile = 'd:\\YOURPATH\\data.zip'
output = zipfile.ZipFile(zfile, 'w')
output.write('d:\\YOURPATH\\data.nc', coveragestore + '.nc', zipfile.ZIP_DEFLATED )
output.close()
```

cmd | cURL

```
curl -u admin:geoserver -XPUT -H "Content-type:application/zip" --data-binary @d:\\YOURPATH\\data.zip
http://localhost:8090/geoserver/rest/workspaces/ws/coveragestores/st/file.netcdf
```

Python | requests

```
with open(output.filename, 'rb') as zip_file:
    r_create_layer = requests.put("http://localhost:8090/geoserver/rest/workspaces/" + workspace  + "
/coveragestores/" + coveragestore  + "/file.netcdf",
        auth=('admin', 'geoserver'),
        data=zip_file,
        headers=headers_zip)
```

At the end, remember to delete the zip file created for Web transfer.

```
import zipfile

# deletes the zip created
os.remove(zfile)
```

Performing actions using RESTful API

If you are using GeoServer interface, the official manual describes very well how to modify layers, change names, or assign styles. As Restful interface might result harder to use, here are some examples.

Change layer name

Changing name to a layer can become convenient in case you have many NetCDF files with different names, though NetCDF variables have the exact same name, e.g. `Band1`. Say you want to change `Band1` into `newBand`.

cmd | cURL

```
curl -v -u admin:geoserver -XPUT -H "Content-type: text/xml" -d "<coverage><name>newBand</name><enabled>true<
/enabled></coverage>" "http://localhost:8090/geoserver/rest/workspaces/ws/coveragestores/st/coverages/Band1"
```

Python | requests

```
old_name_layer = "Band1"
new_name_layer = "newBand"

r_change_name = requests.put('http://localhost:8090/geoserver/rest/workspaces/' + workspace + "/coveragestores
/" + coveragestore + "/coverages/" + old_name_layer,
    auth=('admin', 'geoserver'),
    data='<coverage><name>' + new_name_layer + '</name><enabled>true</enabled></coverage>',
    headers=headers_xml)
```

Create and assign a style to a layer

Create a new style, upload it to GeoServer and assign to a `workspace:layer` definition.

Python | requests

```
stylename = 'test'
stylefilename = stylename + '.sld'
styleallname = 'd:\\ROURPATH\\styles\\' + stylefilename

# creates new style
r_create_new_style = requests.post("http://localhost:8090/geoserver/rest/styles",
    auth=('admin', 'geoserver'),
    data='<style><name>' + stylename + '</name><filename>' + stylefilename + '</filename></style>',
    headers=headers_xml)

# upload new style
with open(styleallname, 'rb') as sld_file:
    r_upload_new_style = requests.put("http://localhost:8090/geoserver/rest/styles/" + stylename,
        auth=('admin', 'geoserver'),
        data=sld_file,
        headers=headers_sld)

# assign it to a layer
r_assign_new_style = requests.put("http://localhost:8090/geoserver/rest/layers/"+ workspace + ':' +
new_name_layer,
    auth=('admin', 'geoserver'),
    data='<layer><defaultStyle><name>' + stylename + '</name></defaultStyle></layer>',
    headers=headers_xml)
```