Building GeoServices for CloudEO Store Using ENVI®

Alberto Meroni
Harris Geospatial Solutions

Thomas Benz
CloudEO
Introduction
Developing a GeoService in a cloud environment may appear like a complex task because it requires competencies in different technologies, e.g. JavaScript, for generating the web interface that gathers user inputs. Or, one must develop a technology to query and retrieve the Electro-Optical (EO) data needed to perform the service and furthermore, one needs the development environment used for the construction of the analytic function that is the core of the service.

This document is intended to illustrate this process. It is straightforward when the set of tools provided by CloudEO and Harris Geospatial Solutions are used. In particular, it illustrates how the development of a GeoService is realized through an ENVI Task with some small precautions. This is done without the necessity of other technologies or the development of other components.

The figure below shows a diagram with all the components that may be needed to carry out a GeoService. The diagram mirrors the CloudEO infrastructure, however it is useful to remember that all functional blocks, although in a different form, must also be available on other platforms and therefore need to be customized to host a GeoService.

Figure 1
CloudEO GeoService

In summary, all that is needed to deploy a new GeoService in the CloudEO Platform is to create the analytic processing routine, indicated by the green box above, as an ENVI Task. All the other components are already available.
**Thin Client**

The traditional development of a GeoService includes the development of an interface for a thin client such as a web browser, which provides mechanisms for selecting and/or setting the parameters in a format consistent with the GeoService exposed and hence running the service. In the clear majority of cases where Earth Observation data is needed, all the query and retrieve mechanisms of the data are completely borne by the developer.

Using the CloudEO infrastructure, this is dramatically simplified by customizing a generic client, illustrated in Figure 2 below, which allows for the selection of common parameters such as the Region of Interest (ROI), the time interval, and up to 50 additional customizable parameters.

The figure below shows a diagram with all the components that may be needed to carry out a GeoService. The diagram mirrors the CloudEO infrastructure, however it is useful to remember that all functional blocks, although in a different form, must also be available on other platforms and therefore need to be customized to host a GeoService.

---

**Figure 2**

*Thin Client Generic UI*
The combination of Client UI and Order Orchestration fulfils three indispensable tasks:

- Formats a json file containing all the parameters needed to run the GeoService
- Performs the query to the EO database, selects and retrieves the data, and then generates a working folder containing the data.
- Launches the GeoService and publishes the results to the required device.

For the example case, there are the following requirements:

- **Parameters to request in UI:**
  - Index
  - Start Month (StartMonth)
  - End Month (EndMonth)
  - Year
  - ROI

- **Query and Retrieve from Sentinel-2 database**
- **Launch GeoService**
- **Publish results**

![Thin Client Example Product](image-url)

*Figure 3*

*Thin Client Example Product*
After the data recovery process, the CloudEO job orchestrator engine generates an order file, as reported below:

```json
{
    "order": {
        "OrderID": "9999",
        "OrderNumber": "9999-01-08",
        "OrderDateUTC": "2017-08-01",
        "OrderTimeUTC": "10:41:05"
    },
    "line_item": {
        "SKU": "5-9999-999",
        "AOI": "POLYGON((10.552 45.215, 10.857 45.215,
                       10.857 45.430, 10.552 45.430, 10.552 45.215 ))",
        "BBox": "POLYGON((10.552 45.215, 10.857 45.215,
                        10.857 45.430, 10.552 45.430, 10.552 45.215 ))",
        "SizeOfAOI": "9999.99",
        "StartMonth": "February",
        "EndMonth": "March",
        "Index": "NDVI",
        "output_folder": "//tmp//storejobs_out/",
        "LineItem": "9999"
    },
    "scene_paths": {
        "scene_1": "//AppData//S2A_MSIL1C_20170216T102101//MTD_MSIL1C.xml",
        "scene_2": "//AppData//S2A_MSIL1C_20170328T102021//MTD_MSIL1C.xml"
    }
}
```

The order request is then sent to the Geospatial Service Framework (GSF) for the execution of the job via an ENVI Task. The ENVI Task invoked from the orchestrator engine is a generic task, named ENVI_ProcessCloudEOOrder, that has the duty to format and pass the parameters to the specific analytic processing function, which itself is also an ENVI Task.

The analytic ENVI Task is identified by the SKU parameter, which is an ID that should be available in the GSF configuration file named CloudEO_catalog.json as reported below.

```json
{
    "apps_catalog": {
        "app_1": {
            "CloudEO": "5-0131-110",
            "Harris": "app_uhi_doit"
        },
        "app_2": {
            "CloudEO": "5-9999-999",
            "Harris": "app_s2spectralindexdifference_doit"
        }
    }
}
```
**ENVI Task**
The development and installation of the analytic function in the CloudEO platform does not differ much from the creation of an ENVI Task for the desktop environment. More comprehensive documentation is available in the ENVI online help. Analytical ENVI Tasks developed for the CloudEO system will have a minimum set of required input/output parameters.

These parameters are:
- **Input scene**: A URI, or array of URIs, representing the full file path of the EO products
- **Roi**: URI of the XML file describing the ROI
- **OrderID**: String representing the OrderID generated by the job orchestrator.
- **Output folder**: URI representing the output directory
- **Output raster**: This is the reference to the output raster

The general steps for writing an ENVI task are:
1. Write an IDL procedure that contains the data-processing step. It should have keywords that define inputs and outputs. (See the Task Code section for an example)
2. Create a task template that maps the keywords in your procedure to the task parameters. Save the task file with a .task extension (See the Task Description section).
3. Deploy the task files to the CloudEO staff for deployment.
**Task Code**

```fortran
PRO app_s2spectralspectralindex_difference_doit, $  
  input_scenes=input_scenes, $  
  Roi=Roi_File, $  
  Index=Index, $  
  OrderID=OrderID, $  
  output_folder=output_folder, $  
  output_raster=output_raster
  compile_opt idl2

  Catch, errorStatus
  if (errorStatus ne 0) then begin
    App_UtilMessage,"...S2_SID: Process terminated by an anomaly !!!"
    App_UtilMessage,"...Reason: '+!ERROR_STATE.msg
    Message, /RESET
    return
  endif

; Sanity and Parameters checking
outDir = N_ELEMENTS(output_folder) NE 0 ? output_folder : ''
OrderID = N_ELEMENTS(OrderID) NE 0 ? OrderID : ''
Index = N_ELEMENTS(Index) NE 0 ? Index : 'NDVI'

IF output_folder EQ '' THEN BEGIN
  prefs = envi.preferences
  outDir = prefs['directories and files:temporary directory'].value
ENDIF

IF N_ELEMENTS(output_raster) EQ 0 THEN BEGIN
  output_raster = outDir + 'S2_SID_order_+orderID+.dat'
ENDIF

; Check input data
IF ~ISA(input_scenes) THEN BEGIN
  App_UtilMessage,"...S2_SID: Input Scenes do not exist !!!"
  RETURN
ENDIF ELSE BEGIN
  App_UtilMessage,"...S2_SID: Input Scene are: '+input_scenes[0]
  App_UtilMessage,"...S2_SID: Input Scene are: '+input_scenes[1]
ENDELSE

; Validate input products
Valid_URI = BYTARR(input_scenes.LENGTH)
FOR j=0,input_scenes.LENGTH-1 DO $
  Valid_URI[j] = FILE_TEST(input_scenes[j])

pGood = WHERE(Valid_URI EQ 1, /NULL)
IF ~ISA(pGood) THEN BEGIN
  App_UtilMessage,"...S2_SID: Input data doesn't contain valid files !"
  RETURN
ENDIF
input_scenes = input_scenes[pGood]

; Generate ROI from Array
IF ~FILE_TEST(Roi_File) THEN BEGIN
  App_UtilMessage,"...S2_SID: Input ROI does not exist !"
  RETURN
ENDIF

; Start the Processing
; Generate ENVI Rasters
```
Ras1 = envi.OpenRaster(Input_ Scenes[0])
Ras2 = envi.OpenRaster(Input_ Scenes[1])

; Verify the oldest product
Time1 = Ras1[0].Time
Time2 = Ras2[0].Time

IF (Time2.unix_seconds-Time1.unix_seconds) LT 0 THEN BEGIN ; Swap Rasters
    TmpRas = Ras2
    Ras2 = Ras1
    Ras1 = TmpRas
ENDIF

; Select only Rasters for 10 meters resolution
Ras110m = Ras1[0]
Ras210m = Ras2[0]

; ROI
Boundaries = app_UtilLoadRoi(Roi_File)

; Subset Products by ROI
Ras110m_Subset = app_UtilSubSetByRoi(Ras110m,Boundaries)
Ras210m_Subset = app_UtilSubSetByRoi(Ras210m,Boundaries)

; Calibrate S-2 data
Ras110m_Rad = ENVI CalibrateRaster(Ras110m_Subset,Calibration='Radiance')
Ras210m_Rad = ENVI CalibrateRaster(Ras210m_Subset,Calibration='Radiance')

; QUICK Atmospheric Correction (QUAC)
; Use ENVIQuacRaster()

; Spectral Index
Ras110m_Ind = ENVI SpectralIndexRaster(Ras110m_Rad,Index)
Ras210m_Ind = ENVI SpectralIndexRaster(Ras210m_Rad,Index)

; Perform the Image difference
Task = ENVI Task('ImageBandDifference')
Task.INPUT RASTER1 = Ras110m_Ind
Task.INPUT RASTER2 = Ras210m_Ind
; Define outputs
Task.OUTPUT RASTER_URI = envi.GetTemporaryFilename()
; Run the Task
Task.Execute

; ColorSliceClassification range [-1,1] 20 slices Red to Green
; Get the task from the catalog of ENVI Tasks
Task1 = ENVI Task('ColorSliceClassification')
Task1.Input Raster = Task.OUTPUT RASTER
Task1.Color Table Name = 'CB-RdYlGn'
Task1.DATA_MINIMUM = -1.0
Task1.DATA_MAXIMUM = 1.0
Task1.NUMBER_OF_RANGES = 20
Task1.OUTPUT_RASTER_URI = output_raster
Task1.Execute

; Output TIFF file
oRas = Task1.OUTPUT RASTER
output_raster = output_raster.replace('.dat','.tif')
oRas.Export(output_raster,'TIFF')

App_UtilMessage,'...S2_SID::Process completed with success'
Task Description

```
{
    "name": "app_s2spectralindexdifference_doit",
    "base_class": "ENVITaskFromProcedure",
    "routine": "app_s2spectralindexdifference_doit",
    "display_name": "Difference between a Spectral Index computes from two images",
    "description": "Difference between a Spectral Index computes from two images",
    "schema": "envitask_3.0",
    "parameters": [
        {
            "name": "INPUT_SCENES",
            "display_name": "Select two S-2 images",
            "type": "ENVIURIArray",
            "direction": "input",
            "dimensions": "[*]",
            "required": true,
            "description": "Select two S-2 images"
        },
        {
            "name": "ROI",
            "display_name": "ROI",
```

Non-Export Controlled Information
"type": "ENVIURI",
"direction": "input",
"required": true,
"description": "The Region of Interest."
},
{
"name": "INDEX",
"display_name": "Index",
"type": "ENVISPECTRALINDEX",
"direction": "INPUT",
"required": true,
"description": "String representing the pre-defined spectral index."
},
{
"name": "OrderID",
"display_name": "Order Name",
"type": "string",
"direction": "input",
"required": false,
"description": "........"
},
{
"name": "OUTPUT_FOLDER",
"keyword": "OUTPUT_FOLDER",
"display_name": "Output Folder",
"type": "ENVIURI",
"direction": "input",
"required": false,
"description": "........"
},
{
"name": "OUTPUT_RASTER",
"keyword": "OUTPUT_RASTER",
"display_name": "Output Raster",
"type": "ENVIURI",
"direction": "OUTPUT",
"required": false,
"description": "This is a reference to the output raster of filetype PNG."
}