OGC Earth Observation Applications Pilot

Pixalytics Engineering Report

Publication Date: 2020-10-22

Approval Date: 2020-09-23

Submission Date: 2020-08-11

Reference number of this document: OGC 20-037

Reference URL for this document: http://www.opengis.net/doc/PER/EOAppsPilot-Pixalytics

Category: OGC Public Engineering Report

Editor: Samantha Lavender

Title: OGC Earth Observation Applications Pilot: Pixalytics Engineering Report

OGC Public Engineering Report

Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Table of Contents

Chapter 1. Subject

This is an individual Engineering Report (ER) created by Pixalytics Ltd as part of the Earth Observation Applications Pilot. Pixalytics' role was that of an App developer, testing deployment to the OGC Earth Observation Applications Pilot architecture.

Chapter 2. Executive Summary

Pixalytics are increasingly involved in developing applications (Apps) that need to be run in near-real-time as part of automated operational services. Therefore, we're developing cloud-deployed Apps that are currently running in a cloud environment we've setup but in the longer term this will expand to:

- Additional cloud platforms, so we can elastically expand on-demand as more resources are needed;
- Deploying our Apps to other platforms where they are sold as part of a workflow.

Pixalytics chose its Mosaic App for containerization and deployment to the OGC Earth Observation Applications Pilot architecture. This engineering report defines 'dockerization' as the containerization of an app for deployment in or as a Docker [https://www.docker.com] container. To support the dockerization of the App, we first concentrated on Dockerization itself and created a very simple application that we could deploy and run. This enabled us to make the distinction between parameters passed into the container for the workflow versus the platform itself to, for example, define where to write output files.

Testing started on the Spacebel platform as that had a GUI as an interface. The use of a private repository was found to cause deployment problems and so Spacebel provided a temporary private Google Cloud Platform (GCP) [https://cloud.google.com/gcp] repository that we could use for the duration of this project.

The standard approach of providing satellite imagery to application containers developed in the OGC pilot project is to use an online repository service, such as Sentinel Hub to pull in the required resources. Pixalytics' Mosaic App instead relies on data provided from a manually curated source. It was decided to provide data to the container using an Amazon Web Services (AWS) S3 [https://aws.amazon.com/s3/] bucket. Once an approach for providing the authorization credentials was solved, this approach worked well but resulted in some data charges from Amazon for data transfer.

There was some confusion over defining a Common Workflow Language (CWL) [https://www.commonwl.org] file for deployment as it was possible to create a file that worked locally, but didn't run remotely as what was implemented on the platforms didn't include all CWL options. On the Spacebel platform it wasn't possible to use environment variables instead of inputs. So we worked around this by having the inputs, which are ignored, including the setup of environment variables.

Testing was also undertaken on the CRIM platform. They were provided access to the Pixalytics Docker repository and downloaded it so that when the App ran it found the container rather than trying to download it as for this platform. Also, private Docker containers were not supported. We modified the CWL file so that the App could be deployed, but errors were generated when it ran and these were not solved in advance of testing concluding.

It would be good if the platforms could accept Docker containers within private repositories as there are business models for both private and public containers.

For those new to OGC, the wealth of knowledge built up over time can be confusing to navigate. It would be useful to have more extensive guidance from each platform on App deployment, including a testing tool (like cwltool) so the approach can be tested locally before deployment.

2.1. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Contacts

Name	Organization	Role
Samantha Lavender	Pixalytics Ltd	Editor/Contributor
Dean Thomas	Pixalytics Ltd	Contributor

2.2. Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

Chapter 3. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

- **Container**

  A standardized unit of software (Docker [https://www.docker.com/resources/what-container]).

- **OpenAPI Definition Document**

  A document (or set of documents) that defines or describes an API. An OpenAPI definition uses and conforms to the OpenAPI Specification [OpenAPI [https://www.openapis.org]]

- **OpenSearch**

  Draft specification for web search syndication, originating from Amazon's A9 project and given a corresponding interface binding by the OASIS Search Web Services working group.

- **Service interface**

  Shared boundary between an automated system or human being and another automated system or human being

- **Workflow**

  Automation of a process, in whole or part, during which electronic documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules (source ISO 12651-2:2014)

3.1. Abbreviated terms

- ADES Application Deployment and Execution Service
- AOI Area Of Interest
- AP Application Package
- AWS Amazon Web Service
- BPEL Business Process Execution Language
- CFP Call For Participation
- CWL Common Workflow Language
- DWG Domain Working Group
- EMS Execution Management Service
- EO Earth Observation
- EP Exploitation Platform
- ER Engineering Report
- ESA European Space Agency
- FUSE Filesystem in Userspace

- GCP Google Cloud Platform
- GUI Graphical User Interface
- JSON JavaScript Object Notation
- MEP Mission Exploitation Platform
- OGC Open Geospatial Consortium
- OWC OWS Context
- REST REpresentational State Transfer
- TEP Thematic Exploitation Platform
- TIE Technology Integration Experiments
- TOI Time Of Interest
- UI User Interface
- URI Uniform Resource Identifier
- URL Uniform Resource Locator
- VM Virtual Machine
- WKT Well-Known Text
- WCS Web Coverage Service
- WFS Web Feature Service
- WPS Web Processing Service
- WPS-T Transactional Web Processing Service

Chapter 4. Overview

Section 5 introduces the Earth Observation (EO) App. It describes the situation prior to the testbed and discusses the requirements set by the sponsors.

Section 6 describes the findings from testing the EO App in two different platforms - Spacebel (TIE 3001 and 3100) and CRIM (TIE 3000 and 3101).

Section 7 provides a summary of the main findings, and discusses recommendations and future work.

Chapter 5. EO Application Description

5.1. Overview

The existing Mosaic application was configured to run as a long-running RESTful web service, which is handled by using Flask to define an Application Programming Interface (API) designed to be called by external websites.

To fit in with the execution model defined by the OGC pilot project, the containerized application is designed in a style more comparable to traditional command-line applications. Common Workflow Language [https://www.commonwl.org/v1.1/] (CWL) is used to specify inputs and outputs to the application, along with other relevant metadata, to enable the Mosaic application to be run and exit immediately upon completion.

To help understand the requirements placed upon the container to fit in with the OGC application/platform architecture, a lightweight test container was developed. The test container allowed us to more easily explore how run-time parameters, for example, string or integer values are passed into the container as environment variables. It also enabled us to make the distinction between parameters passed into the container as defined by the workflow and those defined by the platform itself (such as where to write output files). It also allowed us to generate some initial outputs to validate that the application could take a set of run-time parameters and use this to generate output derived from these.

5.2. Inputs

The standard approach of providing satellite imagery to application containers developed in the OGC pilot project is to use an online repository service, such as Sentinel Hub to pull in the required resources. Pixalytics' Mosaic application instead relies on data provided from a manually curated source; in its existing Pixalytics hosted implementation this is handled by mounting a network share in the execution environment.

This approach does not directly fit with the "application at data source" architecture defined by the OGC project. Hence, alternative approaches were investigated as to how to provide the curated data into the platform. An initial approach was to attempt to mount the shared data within the container itself, using the Secure Shell Filesystem (SSHFS) protocol. However, this proved to be problematic as it would require modified deployment parameters in the Platform to handle mounting the volume using FUSE (Filesystem in Userspace). To aid portability, it was decided that the approach would be to provide data to the container using an Amazon Web Service (AWS) S3 bucket. This is an approach that Pixalytics are familiar with, as it has been used in other projects.

5.3. Containerization

To enable the application to be deployed to an external EO platform the program is provided as a command line application wrapped in a docker container. This is a slight departure from the approach used on the API server to host the API using a Flask RESTful API.

To handle the different deployment strategies, the following architecture of containers is used:

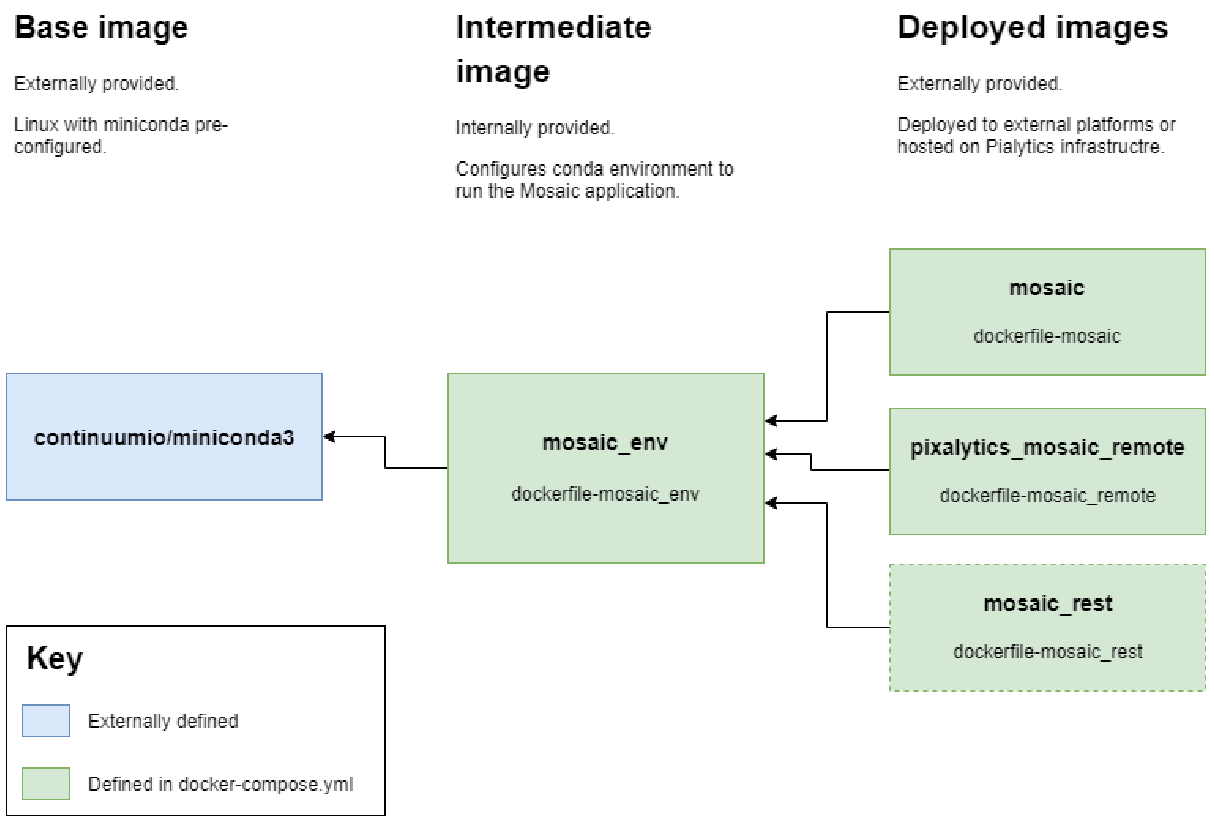


Figure 1. Docker container deployment architecture

Using this tiered approach enables us to take advantage of Docker's ability to compose images from increasingly focused output images. The **miniconda3** image is provided externally by the creators of miniconda [https://docs.conda.io/en/latest/miniconda.html#]. The **mosaic_env** image is the most complex and time-consuming level of the architecture to construct as it constructs the conda environment required to run the application. Providing it as a standalone intermediate image enables rapid development and debugging of the deployable images, as the intermediate image does not require frequent rebuilding.

There are currently three deployment strategies defined:

- **mosaic** wraps the raw command-line mosaic application to be deployed locally with mounted volumes.
- **pixalytics_mosaic_remote** wraps the raw command-line mosaic application to be deployed to OGC platform providers, which accesses data from AWS S3.
- **mosaic_rest** (not currently developed) will be used to provide a container around the Flask API

Building **pixalytics_mosaic_remote** varied according to the platform the EO App was being on. To do this we setup a different build file for each platform, with the one below being the example used for Spacebel:

Code Build Spacebel container

```
FROM api_mosaic_mosaic_env

#   copy all files into this container, including data files
COPY ./dist $APP_HOME/dist
COPY ./wrs2_descending $APP_HOME/wrs2_descending

#   set environment defaults
ENV DEFAULT_OUTDIR=./
ENV DEFAULT_INDIR=$APP_HOME/mtl
ENV DEFAULT_SWLAT=50.36
ENV DEFAULT_SWLON=-4.19
ENV DEFAULT_NELAT=50.42
ENV DEFAULT_NELON=-4.07

ENV INDIR=$DEFAULT_INDIR \
    OUTDIR=$DEFAULT_OUTDIR \
    USERDIR=$DEFAULT_INDIR \
    SWLAT=$DEFAULT_SWLAT \
    SWLON=$DEFAULT_SWLON \
    NELAT=$DEFAULT_NELAT \
    NELON=$DEFAULT_NELON \
    FLAGS="--stretch "

#   for debugging we'll dump the location of the 'mosaic_env' environment using the C
ONDA_PREFIX variable
RUN source activate mosaic_env && echo $CONDA_PREFIX

#   run the code in the correct environment
ENTRYPOINT ["bash", "-c", "source activate mosaic_env &&
/app/dist/landsat_mosaic/landsat_mosaic --in=${INDIR} \
                                          --out=${OUTDIR} \
                                          --user=${USERDIR} \
                                          --swlat=${SWLAT} \
                                          --swlon=${SWLON} \
                                          --nelat=${NELAT} \
                                          --nelon=${NELON} \
                                          ${FLAGS}"]

#   set AWS credentials
COPY .aws /home/.aws
ENV AWS_SHARED_CREDENTIALS_FILE=/home/.aws/credentials

#   copy in files that are needed - MTL files
COPY ./mtl $APP_HOME/mtl

#   create OUTDIR as not mounted as volume
CMD mkdir $APP_HOME/out

# Set Spacebel environment variable
ENV WPS_myOutput=RGB-Stretched.TIF
```

Then the container is built using a yml using the command:

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

Code docker-compose.yml

```
version: '3.7'

services:
  # build the base environment for running the other services from.  This should be
built as little as possible as it
  # has a long running 'conda create env' operation
  mosaic_env:
    build:
      context: .
      dockerfile: dockerfile-mosaic_env
    container_name: mosaic_env

  # run mosaic as a one time execution process with remote deployment
  mosaic_remote_spb:
    build:
      context: .
      dockerfile: dockerfile-mosaic_spb
    depends_on:
      - mosaic_env
    image:
      pixalytics_mosaic_remote
```

5.4. Processing

The processing approach within this EO App is simple as the aim was not to include an App with a complex processing chain as the first step. It includes the following processing steps, which are implemented using Python code:

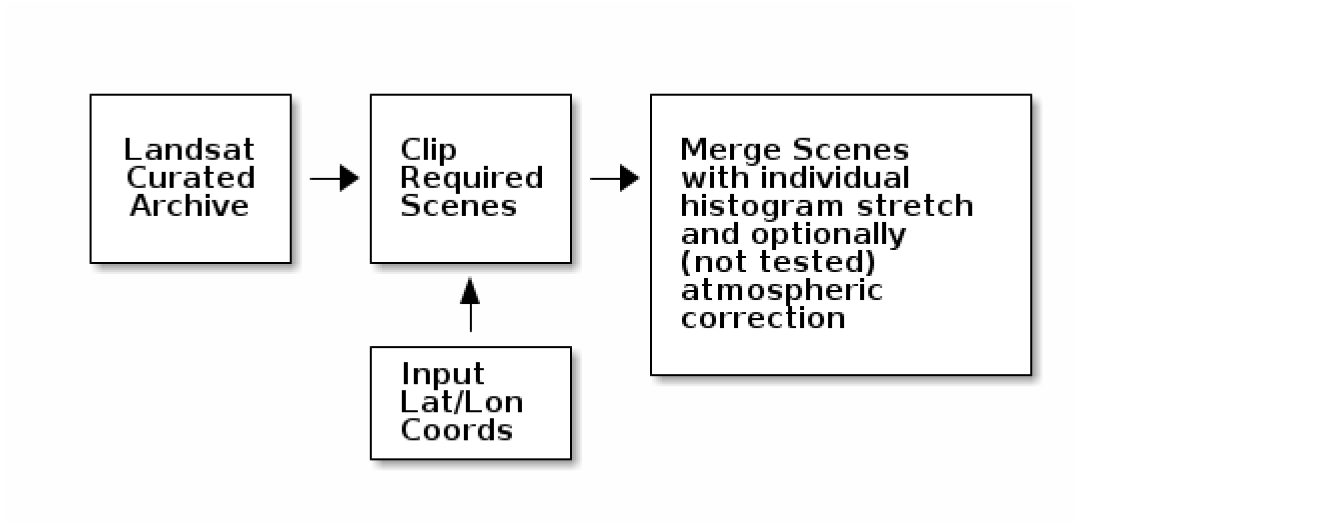

Landsat Curated Archive
Clip Required Scenes
Merge Scenes with individual histogram stretch and optionally (not tested) atmospheric correction
Input Lat/Lon Coords

Figure 2. Landsat Mosaic Workflow

The chosen Latitude/Longitude range for the testing was chosen in a way that the Area Of Interest

(AOI) lay within a single Landsat-8 scene but, more generally, when the code is run it can accept a defined AOI of any size. Therefore, when being run, it is used to merge more than one Landsat-8 scene as part of the mosaic creation process.

The reason for this restriction was to reduce the needed computer resources in terms of the number of scenes to be downloaded, memory and processing requirements. Use of a larger dataset, alongside the application of an atmospheric correction, would be the next steps in testing for this App.

5.5. Outputs

The output is a single GeoTIFF named according to the processing applied, with the Latitude/Longitude Box specified at the start of the filename and any processing specified at the end:

```
50.36--4.19-50.42--4.07-RGB-Stretched.tiff
```

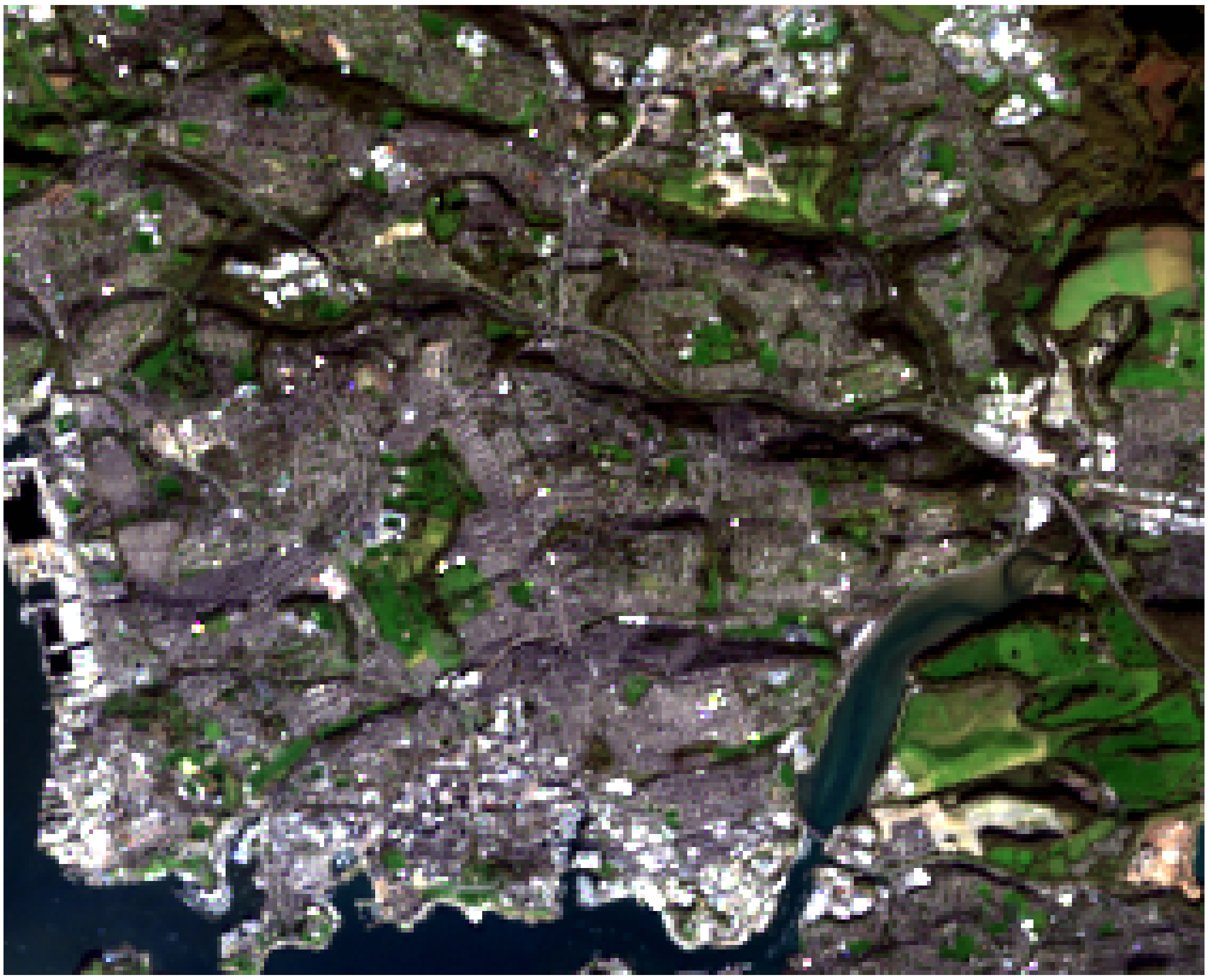


Figure 3. Final Output

Chapter 6. Findings and discussion during the application integration in the platforms

6.1. Packaging

For Docker image storage Pixalytics currently uses private Docker Hub repositories. The Docker containerization is described in Section 5.

6.2. Interface with Spacebel platform

Two applications were testing on the Spacebel platform:

- TIE 3100: A simple Docker container, python_hello_docker, to test the setting up and deployment of container that just prints a simple text string
- TIE 3001: The full EO app called pixaltics_mosaic_remote that is described in Section 5

6.2.1. Application description

There was some confusion over defining a CWL file deployment as it was possible to create a file that worked locally, run and validated using cwltool, but that didn't run remotely as what was implemented on the platforms didn't include all CWL options.

To find a CWL that worked we consulted the Wiki written by Spacebel for this activity and documentation from Testbed-16. The way we'd setup the container was that the inputs were specified by environment variables and when run it immediately executed the application, so we didn't need to provide a command line of inputs for the default processing setup. For the first test we just supplied a single input, which wasn't needed for the code to execute as the command line is not used, and so it used the default inputs:

Code Spacebel Default CWL

```
cwlVersion: v1.0
class: CommandLineTool
id: pixalytics_mosaic_spb
label: Pixalytics API application
hints:
  DockerRequirement:
    dockerPull: eu.gcr.io/spb-gep-ogc/pixalytics_mosaic_remote:latest
inputs:
  WPS_myInput:
    label: No Input
    default: none
    inputBinding:
      position: 1
      eo:envvar: WPS_myInput
    type: string
outputs:
  WPS_myOutput:
    type: File
    outputBinding:
        glob: '50.36--4.19-50.42--4.07-RGB-Stretched.TIF'
    eo:envvar: WPS_myOutput
$namespaces:
  eo: http://www.opengis.net/eo/cwl/extension
```

As it wasn't possible to use environment variables instead of inputs, we worked around this by having the inputs include the setup of environment variables, e.g. SWLAT sets up `eo:envvar: SWLAT`.

Code Spacebel Full CWL

```
cwlVersion: v1.0
class: CommandLineTool
id: pixalytics_mosaic_spb
label: Pixalytics API application
hints:
  DockerRequirement:
    dockerPull: eu.gcr.io/spb-gep-ogc/pixalytics_mosaic_remote:latest
inputs:
  WPS_myInput:
    label: Atmospheric Correction flag
    default: none
    inputBinding:
      position: 1
      eo:envvar: WPS_myInput
    type: String
  SWLAT:
    label: South West Latitude
    inputBinding:
      position: 2
      eo:envvar: SWLAT
    type: String
  SWLON:
    label: South West Longitude
    inputBinding:
      position: 3
      eo:envvar: SWLON
    type: String
  NELAT:
    label: North East Latitude
    inputBinding:
      position: 4
      eo:envvar: NELAT
    type: String
  NELON:
    label: North East Longitude
    inputBinding:
      position: 5
      eo:envvar: NELON
    type: String
outputs:
  WPS_myOutput:
    type: File
    outputBinding:
        glob: '*RGB-Stretched.TIF'
    eo:envvar: WPS_myOutput
$namespaces:
  eo: http://www.opengis.net/eo/cwl/extension
```

6.2.2. Metadata

This EO App didn’t need to supply the platform with metadata information as the data is retrieved internally.

6.2.3. Deployment

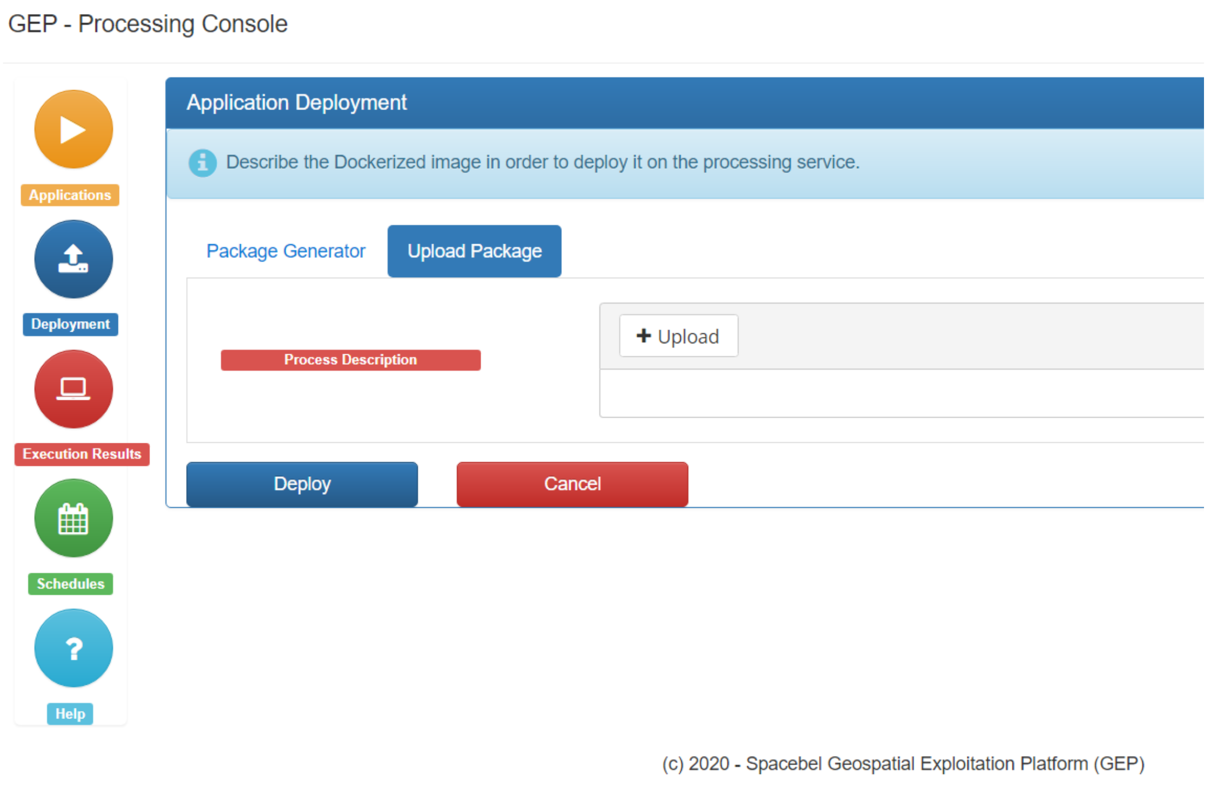
Using a private Docker Hub repository was found to cause deployment problems on the Spacebel platform, which in its initial implementation required containers to exist in public repositories. To address this, Spacebel agreed to provide a temporary private repository that we could use for the duration of this project. This is hosted on Google Cloud Platform (GCP) and required a moderate adjustment to the existing deployment process to target the new platform:

```
gcloud auth login

docker tag pixalytics_mosaic_remote:latest eu.gcr.io/spb-gep-
ogc/pixalytics_mosaic_remote:latest

docker push eu.gcr.io/spb-gep-ogc/pixalytics_mosaic_remote:latest
```

Deployment then involved uploading the CWL file as the Package, and then clicking Deploy:



Describe the Dockerized image in order to deploy it on the processing service.

(c) 2020 - Spacebel Geospatial Exploitation Platform (GEP)

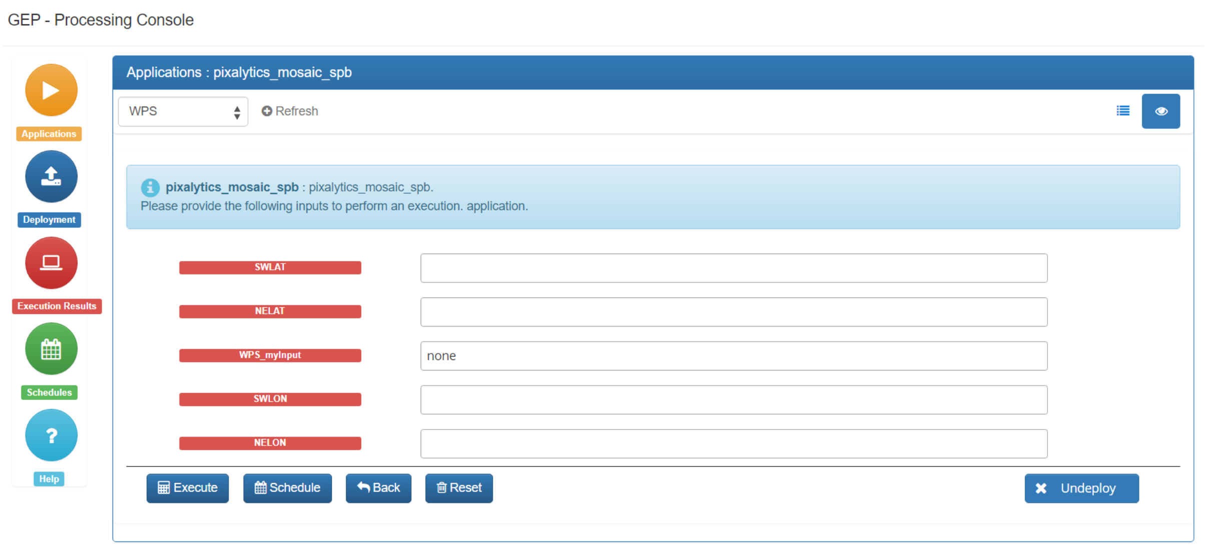
Figure 4. Spacebel deployment

6.2.4. Hardware Requirements

The EO App doesn’t have strong hardware requirements in the way it was configured to run for the testing and so this wasn’t focused on.

6.2.5. Execution

Execution on the platform required specifying the input variables and clicking the 'Execute' button (see Figure 4).



Applications : pixalytics_mosaic_spb

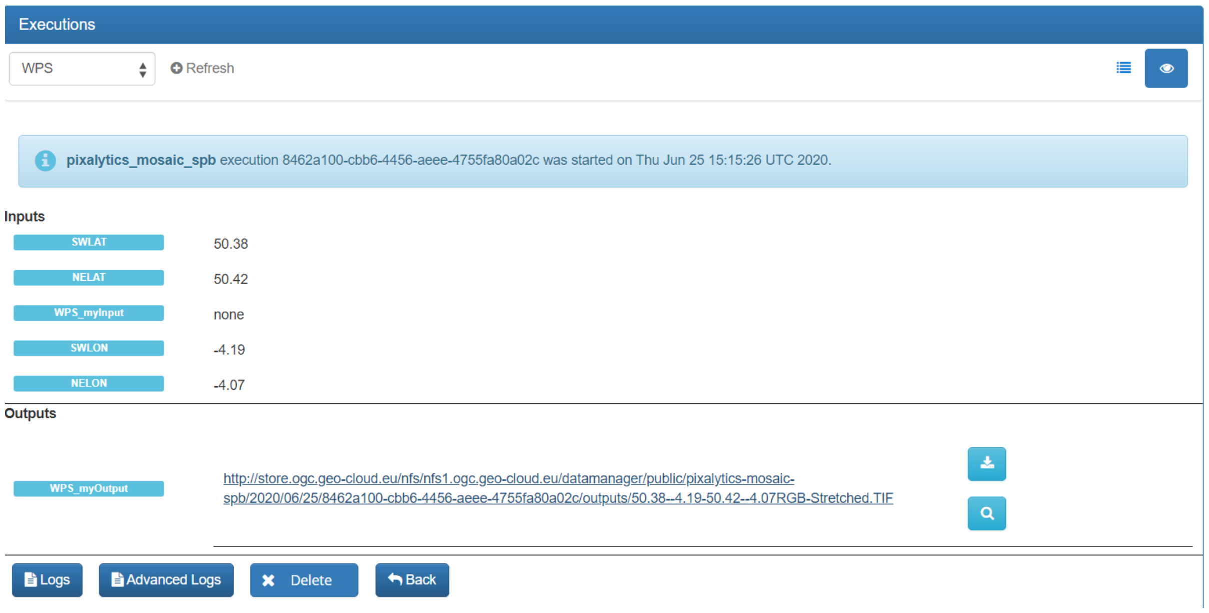
pixalytics_mosaic_spb : pixalytics_mosaic_spb.
Please provide the following inputs to perform an execution. application.

SWLAT	
NELAT	
WPS_myInput	none
SWLON	
NELON	

Figure 5. Spacebel execution

6.2.6. Communication

Executing could be monitored on the platform, and once the process was completed logs were available (see Figure 5).



Executions

pixalytics_mosaic_spb execution 8462a100-cbb6-4456-aeee-4755fa80a02c was started on Thu Jun 25 15:15:26 UTC 2020.

Inputs

SWLAT	50.38
NELAT	50.42
WPS_myInput	none
SWLON	-4.19
NELON	-4.07

Outputs

WPS_myOutput	http://store.ogc.geo-cloud.eu/nfs/nfs1.ogc.geo-cloud.eu/datamanager/public/pixalytics-mosaic-spb/2020/06/25/8462a100-cbb6-4456-aeee-4755fa80a02c/outputs/50.38--4.19-50.42--4.07RGB-Stretched.TIF

Figure 6. Spacebel results

The logs were very useful for tracking down issues when versions of the CWL were deployed but then didn’t run correctly.

6.3. Interface with CRIM platform

One application was tested on the platform:

- TIE 3000: The full EO app called pixaltics_mosaic_remote that is described in Section 5

Unfortunately, we didn’t get to the point where the EO App ran correctly but learnt a lot about deployment to the CRIM platform during the back and forth email conversations. In order to not manually type in the commands, we setup a Makefile containing all the commands with a .env file holding sensitive information (such as usernames and passwords) so they were not pushed to the Git repository holding the EO App.

6.3.1. Application description

We modified the default CWL used for the Spacebel platform to be able to deploy to the CRIM platform. The CWL needed to be placed into an open GitHub repository so it could be read from a remote location when the JSON file was sent as part of the push command:

Code CRIM Default CWL

```
{
    "cwlVersion": "v1.0",
    "class": "CommandLineTool",
    "hints": {
        "DockerRequirement": {
            "dockerPull": "pixalytics/pixalytics_mosaic_remote:latest"
        }
    },
    "inputs": {
        "WPS_myInput": {
            "default": "none",
            "inputBinding": {
                "position": 1
            },
            "type": "string"
        }
    },
    "outputs": {
        "output": {
            "outputBinding": {
                "glob": "*-RGB-Stretched.TIF"
            },
            "type": "File"
        }
    }
}
```

6.3.2. Deployment

Deployment involved logging into the CRIM platform and saving a cookie as the first step:

```
curl -X POST https://ogc-ems.crim.ca/magpie/signin -d
'{"user_name":"$(USER)","password":"$(PASSWORD)"}' -H "Content-Type: application/json"
--silent -c cookie-jar.txt
```

Then, deployment was through a push command with a JSON file:

```
curl -k -d "@deploy-crim.json" -X POST "https://ogc-ems.crim.ca/EMS/processes" -H
"Content-Type: application/json" -b cookie-jar.txt
```

Code CRIM Push JSON

```
{
    "processDescription": {
        "process": {
            "id": "pixalytics_mosaic_crim",
            "title": "Pixalytics Cloud Free Mosaics",
            "abstract": "Pixalytics Docker for Landsat-8 cloud free mosaics",
            "keywords": ["mosaic","landsat"],
            "owsContext": {
                "offering": {
                    "code": "http://www.opengis.net/eoc/applicationContext/cwl",
                    "content": {
                        "href": "https://raw.githubusercontent.com/pixalytics-ltd/ogc-
eoapps/master/pixalytics_mosaic_crim.cwl"
                    }
                }
            },
            "inputs": [
                {
                    "id": "WPS_myInput",
                    "title": "No Input",
                    "type": "string"
                }

            ],
            "outputs": [
                {
                    "id": "output",
                    "title": "Mosaic Image",
                    "formats": [
                        {
                            "mimeType": "application/geotiff",
                            "default": true
                        }
                    ]
                }
            ]
        },
        "processVersion": "1.0.0",
        "jobControlOptions": [
            "async-execute"
        ],
        "outputTransmission": [
            "reference"
        ]
    },
    "immediateDeployment": "true",
    "executionUnit": [
        {
            "href": "pixalytics/pixalytics_mosaic_remote:latest"
        }
    ],
    "deploymentProfileName":
"http://www.opengis.net/profiles/eoc/dockerizedApplication"
}
```

6.3.3. Execution

Before a deployed application could be seen, it needed to be made visible:

```
curl -k -H "Accept: application/json" -H "Content-Type: application/json" -X PUT -d
'{"value":"public"}' "https://ogc-
ems.crim.ca/EMS/processes/pixalytics_mosaic_crim/visibility" -b cookie-jar.txt

curl "https://ogc-ems.crim.ca/EMS/processes/pixalytics_mosaic_crim" -b cookie-jar.txt
```

Then the execution command was sent:

```
curl -L -k -d "@run-crim.json" -X POST "https://ogc-
ems.crim.ca/EMS/processes/pixalytics_mosaic_crim/jobs" -H "Content-Type:
application/json" -b cookie-jar.txt
```

using the following JSON file:

Code CRIM Run JSON

```
{
  "mode": "async",
  "response": "document",
  "inputs": [
    {
      "id": "WPS_myInput",
      "data": "none"
    }
  ],
  "outputs": [
    {
      "id": "output",
      "format": {
        "mimeType": "application/geotiff"
      },
      "transmissionMode": "reference"
    }
  ]
}
```

6.3.4. Communication

To check the execution, the following commands were used to check the log and exceptions with the JOB_ID having been returned from the execution command:

```
curl "https://ogc-
ems.crim.ca/EMS/processes/pixalytics_mosaic_crim/jobs/${JOB_ID}/logs" -b cookie-
jar.txt

curl "https://ogc-
ems.crim.ca/EMS/processes/pixalytics_mosaic_crim/jobs/${JOB_ID}/exceptions" -b cookie-
jar.txt
```

Unfortunately, although the code deployed the execution resulted in the following error messages:

```
curl "https://ogc-ems.crim.ca/EMS/processes/pixalytics_mosaic_crim/jobs/584c50a0-450b-
4f38-90b7-c69d7b280480/logs" -b cookie-jar.txt

["[2020-06-22 14:19:29] INFO     [weaver.datatype.Job] 0:00:00   1% accepted   Job
task setup completed.","[2020-06-22 14:19:29] INFO     [weaver.datatype.Job] 0:00:00
2% accepted   Execute WPS request for process [pixalytics_mosaic_crim]","[2020-06-22
14:19:29] ERROR    [weaver.datatype.Job] 0:00:00  85% failed     Failed to run Job
<584c50a0-450b-4f38-90b7-c69d7b280480>.","[2020-06-22 14:19:29] INFO
[weaver.datatype.Job] 0:00:00  90% failed     Job failed.","[2020-06-22 14:19:29] INFO
[weaver.datatype.Job] 0:00:00 100% failed     Job task complete."]

curl "https://ogc-ems.crim.ca/EMS/processes/pixalytics_mosaic_crim/jobs/584c50a0-450b-
4f38-90b7-c69d7b280480/exceptions" -b cookie-jar.txt

["weaver.owsexceptions.OWSNoApplicableCode: Failed to retrieve WPS capabilities.
Error: [<?xml version=\"1.0\" encoding=\"utf-8\"?>\n<ExceptionReport
version=\"1.0.0\"\n    xmlns=\"http://www.opengis.net/ows/1.1\"\n
xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"\n
xsi:schemaLocation=\"http://www.opengis.net/ows/1.1
http://schemas.opengis.net/ows/1.1.0/owsExceptionReport.xsd\">\n    <Exception
exceptionCode=\"NoApplicableCode\" locator=\"AccessForbidden\">\n
<ExceptionText>Not authorized to access this resource. User does not meet required
permissions.</ExceptionText>\n    </Exception>\n</ExceptionReport>]."]
```

We agreed with CRIM to stop the testing at this point as the Technology Integration Experiment (TIE) testing period had concluded.

Chapter 7. Conclusions/Recommendations/Future Work

7.1. Conclusions

The conclusions are divided according to project meetings, as this was the first time we had taken part in an OGC activity, and the testing activity itself.

7.1.1. Project kick-off meeting (January 2020)

From the perspective of the other members of the working group, Pixalytics joined the process rather late. Many of the existing platform providers and application developers had been working in the group for several, if not all, of the previous stages. Talking to other developers at the meeting highlighted this; therefore, it was identified that our involvement might not just extend to the development and integration of an application for the platforms. Pixalytics were also able to give a unique insight into how an application developer with minimal existing experience of the project may integrate their products.

One noticeable aspect of the ongoing nature of the project was that a large body of terminology had amassed in terms of how to describe roles and the technologies used. Initially, from a relative newcomers point-of-view, this made some of the discussions challenging to understand. Through conversations with other members of the working group, we were directed towards the relevant areas of the project documents to help understand these terms. Another interesting observation from the presentations was that the technology stacks had already been agreed upon; each App developer and platform provider had already settled on the use of Docker containers and use of Kubernetes (or similar) based solutions. Pixalytics already had a basic understanding of how these solutions worked; however, the use of workflow descriptive languages, including CWL, was a new development for us.

When it came to discussing with providers about the services that their platforms could offer, the unique way that the data was provided to our platform (curated, rather than cloud service sourced) played an important role in who it was felt necessary we could work with. Spacebel were keen to work with us as our relatively simple application would easily integrate with their platform in its current state. Their platform also had the advantage that it was already relatively close to being deployable for testing. Spacebel also said they were keen to closely work with CRIM to coordinate their efforts to produce a similar system for hosting and deploying apps. While the CRIM platform was further from being ready for deployment, we felt that we could integrate well into their collective efforts.

7.1.2. Testing the EO App on the Platforms

To support the docker-based containerization of the App, we first concentrated on containerization through Docker itself and created a very simple application that we could deploy and run. This simple App helped us to understand how run-time parameters, for example, string or integer values, are passed into the container as environment variables. It also enabled us to make the distinction between parameters passed into the container as defined by the workflow and those defined by the platform itself (such as where to write output files).

Testing started on the Spacebel platform as that had a GUI as an interface. The test container was deployed and run after some iteration with Spacebel, which gave us the confidence to deploy the more complex EO App.

For container image storage, Pixalytics currently uses private Docker Hub repositories. This use of a private repository was found to cause deployment problems on both the CRIM and Spacebel platforms, which expected containers to exist in public repositories. To address this:

- CRIM was provided access to the Pixalytics Docker repository and downloaded it so when the App ran it found the container rather than trying to download it.
- Spacebel agreed to provide a temporary private repository that we could use for the duration of this project. This repository was hosted on the GCP and required a moderate adjustment to the existing deployment process to target the new platform. It also enabled us to make a comparison in the deployment process between GCP and DockerHub; while this was relatively trivial, it required an adjustment to the command line toolset used to push the containers into the cloud.

The standard approach of providing satellite imagery to application containers developed in the OGC pilot project is to use an online repository service, such as Sentinel Hub to pull in the required resources. Pixalytics' Mosaic App instead relies on data provided from a manually curated source; when run on Pixalytics infrastructure this is handled by mounting a network share in the execution environment. To aid portability, it was decided that the approach would be to provide data to the container using an AWS S3 bucket. This is an approach that Pixalytics are familiar with, as it has been used in other projects. This resulted in some data charges from Amazon, to the S3 bucket owner for data transfer, so would need to be taken into consideration for larger-scale deployments.

There was some confusion over defining a CWL file for deployment as it was possible to create a file that worked locally, but didn’t run remotely as what was implemented on the platforms didn’t include all CWL options. To find a CWL that worked, we consulted the Wiki written by Spacebel for this activity and documentation from Testbed-16. It took several days to find a solution for the Spacebel deployment, and we didn’t solve this for the CRIM platform.

The way we’d setup the container was that environment variables specified the inputs and, when run, it immediately executed the application, so we didn’t need to provide a command-line of inputs for the default processing setup. However, for the Spacebel platform, it wasn’t possible to use environment variables instead of inputs. So we worked around this by having the inputs, which are ignored, including the setup of environment variables.

Another element that took some effort to solve was providing security credentials into the Docker container for downloading data from AWS. There were several suggestions on the internet, and our chosen approach was to copy the AWS credentials from the container builder’s home folder into the image. This approach avoided saving the credentials in files that would be synced as part of the EO Apps Github repository.

7.2. Recommendations

It would be good if the platforms could accept Docker containers within private repositories. We feel this is important as commercial platforms are now building business models where containerized Apps are deployed within workflows and the owner of the App receives payment for their usage. So, there are business models for both private and public containers.

For those new to OGC, the wealth of knowledge built up over time can be confusing to navigate. We found it useful to read through the documents from previous testbeds to try and understand what others had a much better knowledge of. The participants were also helpful and so asking questions also allowed us to focus in on where our deployment issues were.

As an App developer, it would be useful to have more extensive guidance from each platform on App deployment. We found examples of working CWL files to be particularly helpful, and Spacebel providing a Wiki for explanation further helped. However, it would be useful to also have a testing tool (like cwltool) so the approach can be tested locally before deployment.

7.3. Future Work

The chosen Latitude/Longitude range for the Mosaic App testing was such that only a single Landsat-8 scene was needed, and we didn’t run tests using the atmospheric correction. Therefore, use of a larger AOI alongside more complex processing should be included in future testing. This would enable us to begin to investigate the specification and allocation of platform resources.

We’d also look at deploying more complex Apps, to further test what is possible. As part of this, we’d investigate further how to build the Docker containers as this was a new activity for us and we’re sure there is more to learn about how best to do this. Specifying the command line within the image and setting the inputs via environment values makes the container more secure as it’s not straightforward to run other commands to look inside the container.

To avoid having to remember the different commands, we setup a Makefile. Overall this worked well, but we didn’t finish the CRIM interface where values such as the JOB_ID need to be retrieved so they can be used for the next command. This should be straightforward, but we ran out of time with the CRIM TIE testing.

Appendix A: Revision History

NOTE *Example History (Delete this note).*

replace below entries as needed

Table 1. Revision History

Date	Name	Version	Sections	Notes
July 05, 2020	S. Lavender	1.0	all	first draft of Pixalytics text
August 04, 2020	Ingo Simonis	1.1	all	full review
October 14, 2020	G. Hobona	1.2	all	final staff review