Documento de diseño - Draco

Estructura de la librería

Para el desarrollo de la librería se tuvo en cuenta el lenguaje de programación usado y las tecnologías disponibles para generar los archivos binarios. Para estos retos el equipo decidió usar Python con ayuda de librerías como GDAL y rasterio. Con esto en cuenta, la estructura de archivos de la librería es la siguiente:

```
Draco/
├─ draco/
|  ├─ __init__.py
|  ├─ transform.py
|  ├─ compress.py
├─ tests/
|  ├─ test_draco.py
|  ├─ __init__.py
|  ├─ data/
|  |  ├─ rgb.tif
├─ venv/
├─ setup.py
├─ LICENSE
├─ README.md
```

Dentro de la carpeta “Draco/draco”, en los archivos “transform.py” y ”compress.py” se encuentra el código fuente que ejecuta las funcionalidades de la librería. En “transform.py” está la función que permite hacer la traslación y rotación de imágenes geográficas. Mientras que en “compress.py” se encuentra el programa para comprimir imágenes para previsualización.

Por otro lado, en la carpeta “Draco/tests/” se presenta el código realizado para hacer pruebas de las funcionalidades de las librerías. Estas pruebas son importantes para la continua mejora de la librería. Lo anterior permite hacer una librería Open Source con más estabilidad.

También se encuentra la carpeta “Draco/venv/” que es la encargada de guardar la información para la construcción de la librería. Y al mismo nivel de esa carpeta, se encuentran los archivos LICENSE, README.md y setup.py, archivos vitales para una librería abierta al público.

Infraestructura

Para el desarrollo de este reto se utilizó la infraestructura que se presenta a continuación, que fue recomendada por Microsoft® en su plataforma Microsoft Azure®.

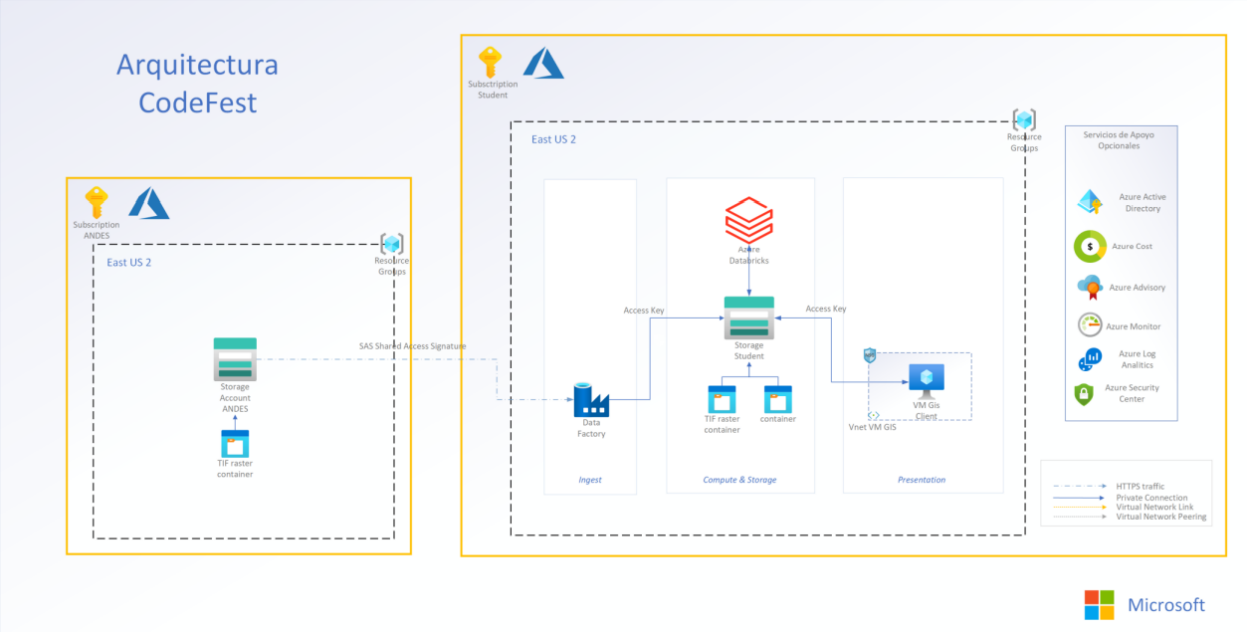


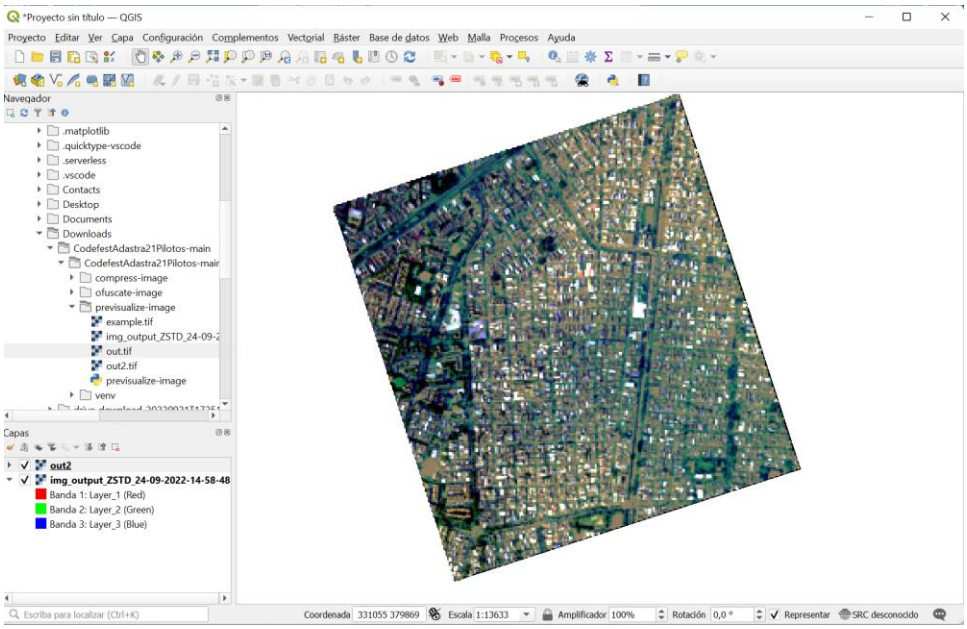
Ilustración 1. Arquitectura CODEFEST

Esta arquitectura consta de una ingesta de datos a partir de un repositorio en el que se tenían colgadas las imágenes a utilizar para el desarrollo de los retos, que eran posteriormente almacenadas en un storage del equipo en la nube a partir del cual se consultaban los datos a través de Databricks.

Las imágenes generadas se abrieron en una máquina virtual Windows con el softwar QGIS para verificar su visualización.

Análisis de imágenes

Para el análisis de imágenes a través de los retos hicimos uso intensivo de qGIS, que es un programa de visualización de imágenes georeferenciadas. Estuvimos visualizando las imágenes originales y sus respectivos procesamientos dependiendo del reto. En varias ocasiones nos enfrentamos al escenario de que el procesamiento realizado resultaba en imágenes con problemas de visualización y el estudio y uso de la herramiento nos ayudó a comprender mucho mejor el tratamiento de imágenes satelitales.



Solución de los retos

1. Alteración de coordenadas:

Para este reto se realizó un acercamiento con la librería rasterio. Se comprendió que para hacer un cambio de ubicación de la imagen se puede editar la transformación que tiene la imagen en los metadatos de la información de georeferencia. Se procedió a cambiar únicamente esta transformación con translación y rotación y escribir una imagen nueva con los nuevos metadatos pero idéntica en contenido.

2. Lossly Compression

Para el desarrollo de los retos identificamos que algunos nuevos métodos de compresión como el ZSTD era los que mejores resultados arrojaban, tanto en tasa de compresión como en tiempo de escritura y lectura. Se trata de hacer un balance entre todos estos algoritmos, identificando que el ZSTD era el mejor para las solución de los retos de compresión.

3. Lossless Compression

Method	Predictor	Size	Write Time	Read Time
Uncompressed	-	865 Mb	5.9 s	7.2 s
Packbits	-	593 Mb	11.0 s	5.6 s
LZW	1	723 Mb	21.3 s	45.3 s
LZW	2	585 Mb	22.7 s	47.3 s
DEFLATE	1	527 Mb	46.2 s	33.8 s
DEFLATE	2	471 Mb	55.9 s	39.6 s

Según comparaciones de expertos, los resultados que vemos en la tabla superior, muestran que uno de los algoritmos con mejor tasa de compresión, es el DEFLATE. Lo utilizamos para la solución del reto3, aunque no cumplimos la tasa de compresión esperada de 1/10.