Building dynamic tiles server with Python

Vincent Sarago - Developmentseed

developmentSEED

Geospatial Engineer @ Developmentseed

COG Tzar

Self-Taught Python dev

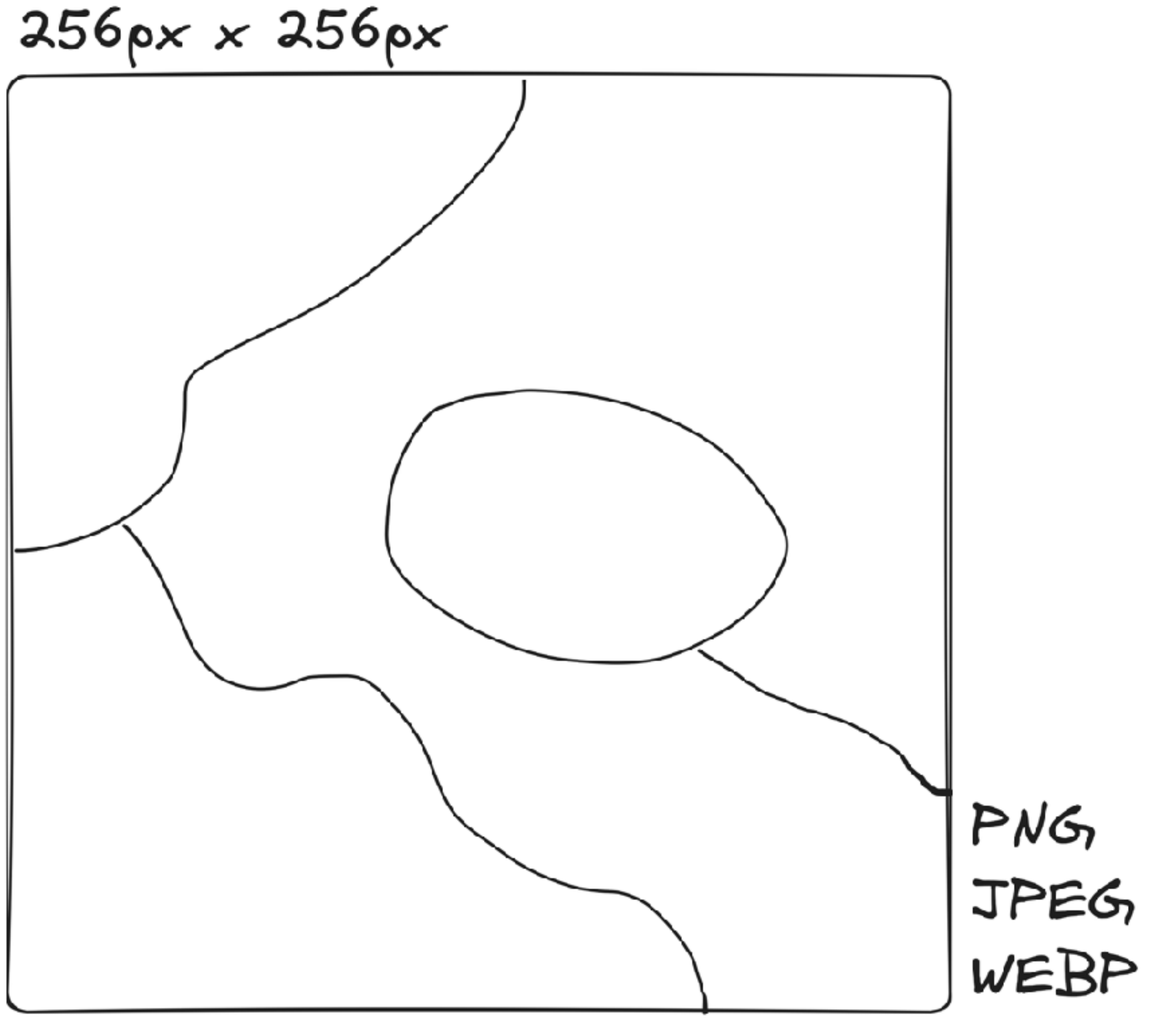
Creator of @RemotePixel

MSc in Earth Sciences

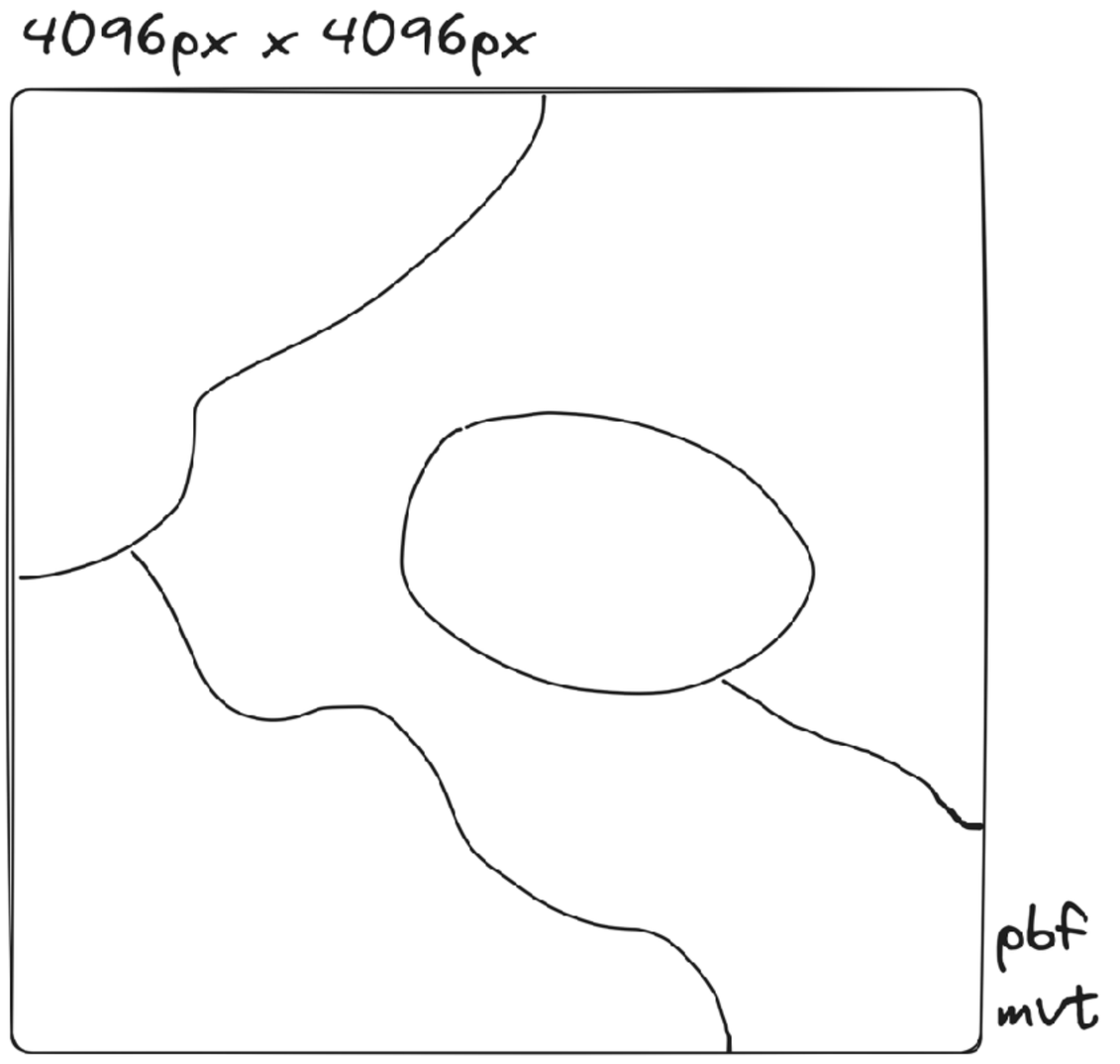
Bike & Coffee

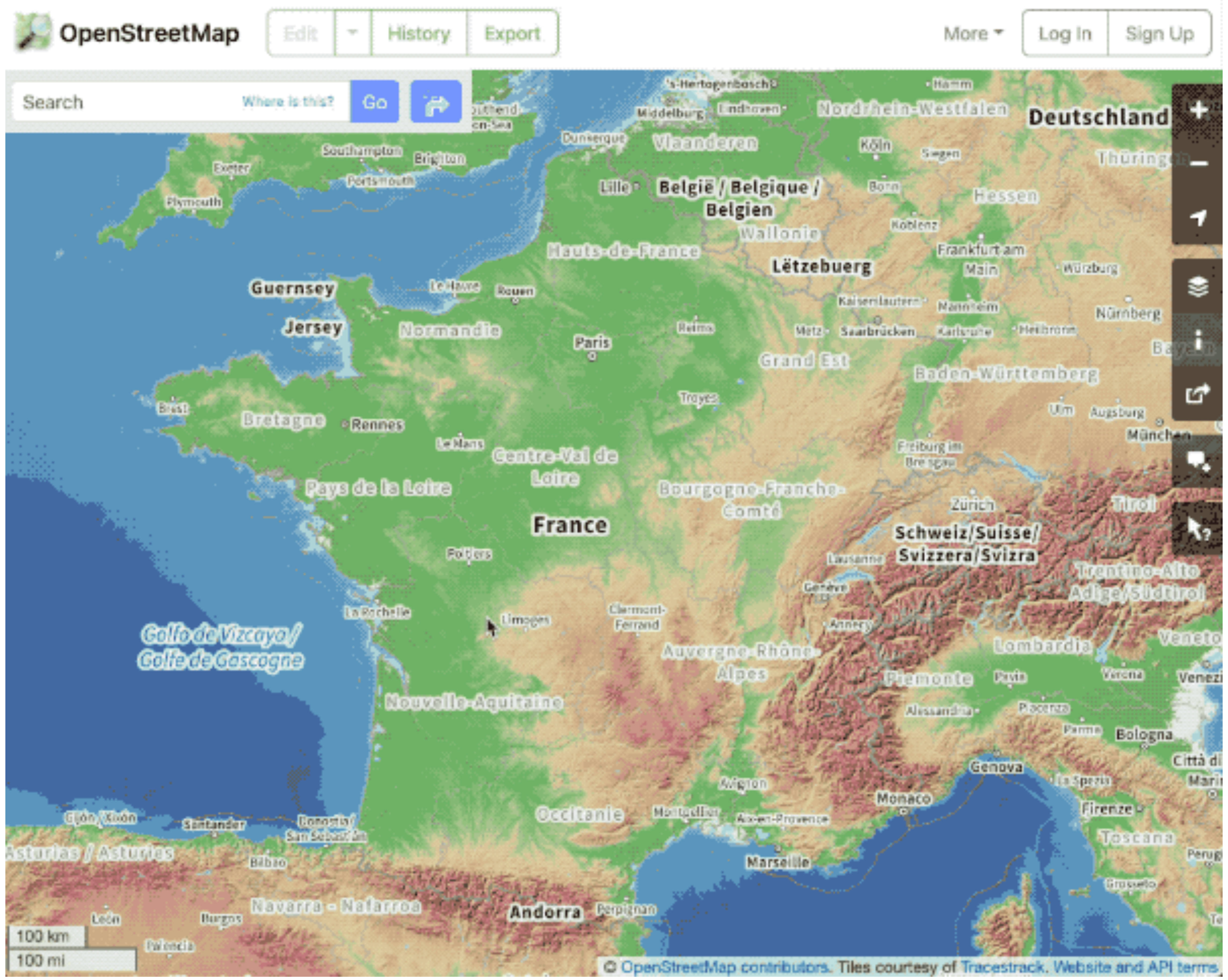
Who knows what's is a dynamic tiler?

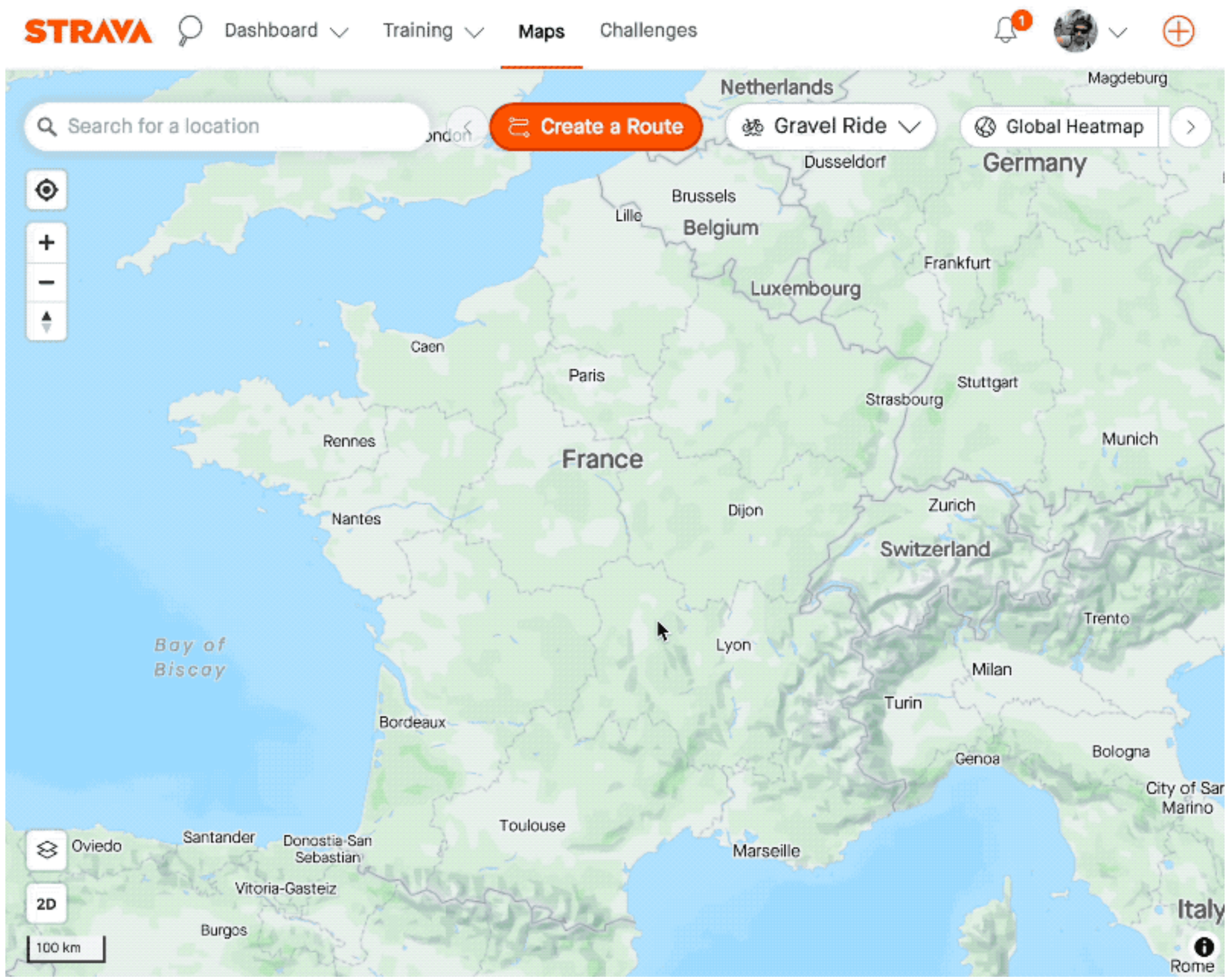
Map Tile (raster)



Map Tile (vector)

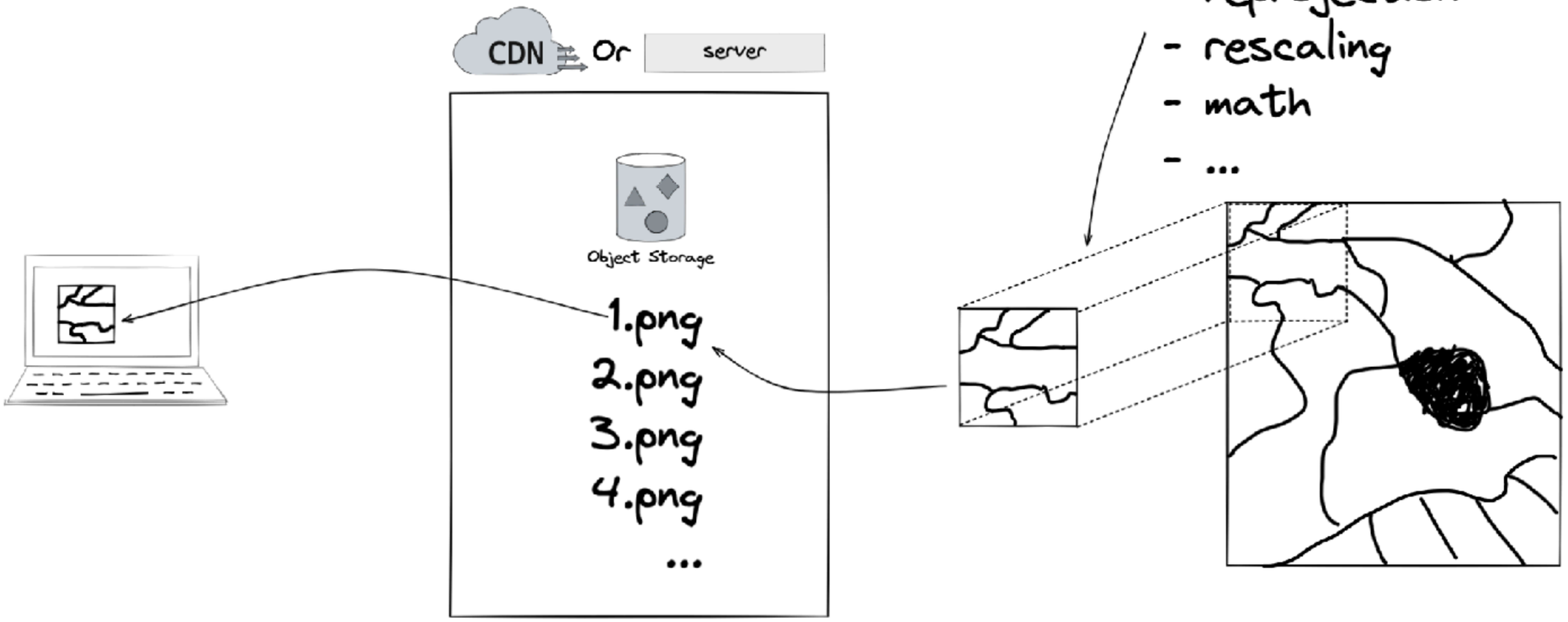



OpenStreetMap
Edit
History
Export
More
Log In
Sign Up
Search
Where is this?
Go
Deutschland
België / Belgique / Belgien
Lëtzebuerg
Guernsey
Jersey
Paris
France
Schweiz/Suisse/Svizzera/Svizra
Golfo de Vizcaya / Golfe de Gascogne
Andorra
Monaco
100 km
100 mi
© OpenStreetMap contributors. Tiles courtesy of Tracestrack. Website and API terms


STRAVA
Dashboard
Training
Maps
Challenges
Search for a location
Create a Route
Gravel Ride
Global Heatmap
Netherlands
Magdeburg
Dusseldorf
Germany
Brussels
Lille
Belgium
Frankfurt
Luxembourg
Caen
Paris
Stuttgart
Strasbourg
Munich
Rennes
France
Nantes
Dijon
Zurich
Switzerland
Bay of Biscay
Lyon
Trento
Milan
Turin
Bordeaux
Genoa
Bologna
City of San Marino
Toulouse
Marseille
Oviedo
Santander
Donostia-San Sebastian
Vitoria-Gasteiz
Burgos
Italy
Rome
2D
100 km

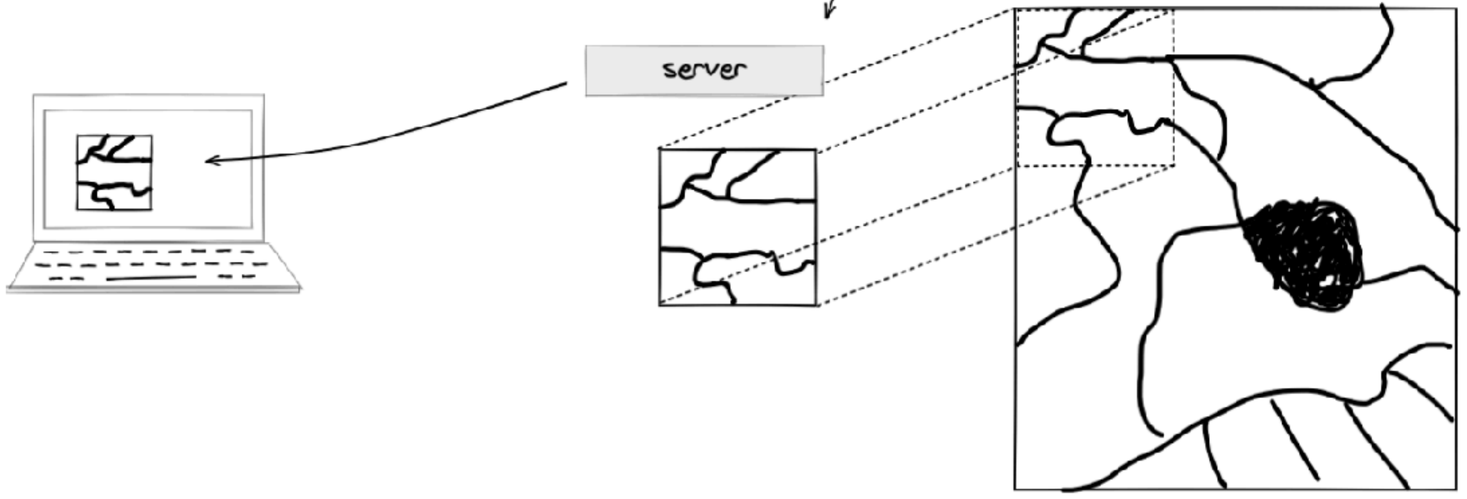
Static vs Dynamic

Static (pre-rendered)



Dynamic
(rendering on-demand)

Processing:
- reprojection
- rescaling
- math
- ...



Ti-* Family

TiTiler
Raster Services

https://developmentseed.org/titiler/

TiPg
Vector / Features Services

https://github.com/developmentseed/tipg

TiTiler
Raster Services

https://developmentseed.org/titiler/

Mercantile -> MOREcantile

Construct and use map tile grids (a.k.a TileMatrixSet / TMS).

morecantile

Construct and use map tile grids (a.k.a TileMatrixSet / TMS).

Open Geospatial Consortium

OGC Two Dimensional Tile Matrix Set and Tile Set Metadata

Pydantic Models

morecantile

Construct and use map tile grids (a.k.a TileMatrixSet / TMS).

```python
import morecantile
from morecantile.defaults import tms
```

```python
for id in tms.list():
    print("-", id)
```

```
- CDB1GlobalGrid
- CanadianNAD83_LCC
- EuropeanETRS89_LAEAQuad
- GNOSISGlobalGrid
- LINZAntarticaMapTilegrid
- NZTM2000Quad
- UPSAntarcticWGS84Quad
- UPSArcticWGS84Quad
- UTM31WGS84Quad
- WGS1984Quad
- WebMercatorQuad
- WorldCRS84Quad
- WorldMercatorWGS84Quad
```

```python
webmercator = tms.get("WebMercatorQuad")
print(webmercator.xy_bounds(0, 0, 0))
print(webmercator.bounds(0, 0, 0))
```

```
BoundingBox(left=-20037508.342789244, bottom=-20037508.34278925, right=20037508.34278925, top=20037508.342789244)
BoundingBox(left=-180.0, bottom=-85.0511287798066, right=180.00000000000009, top=85.0511287798066)
```

```python
print(webmercator.xy_bounds(2, 2, 4))
print(webmercator.bounds(2, 2, 4))
```

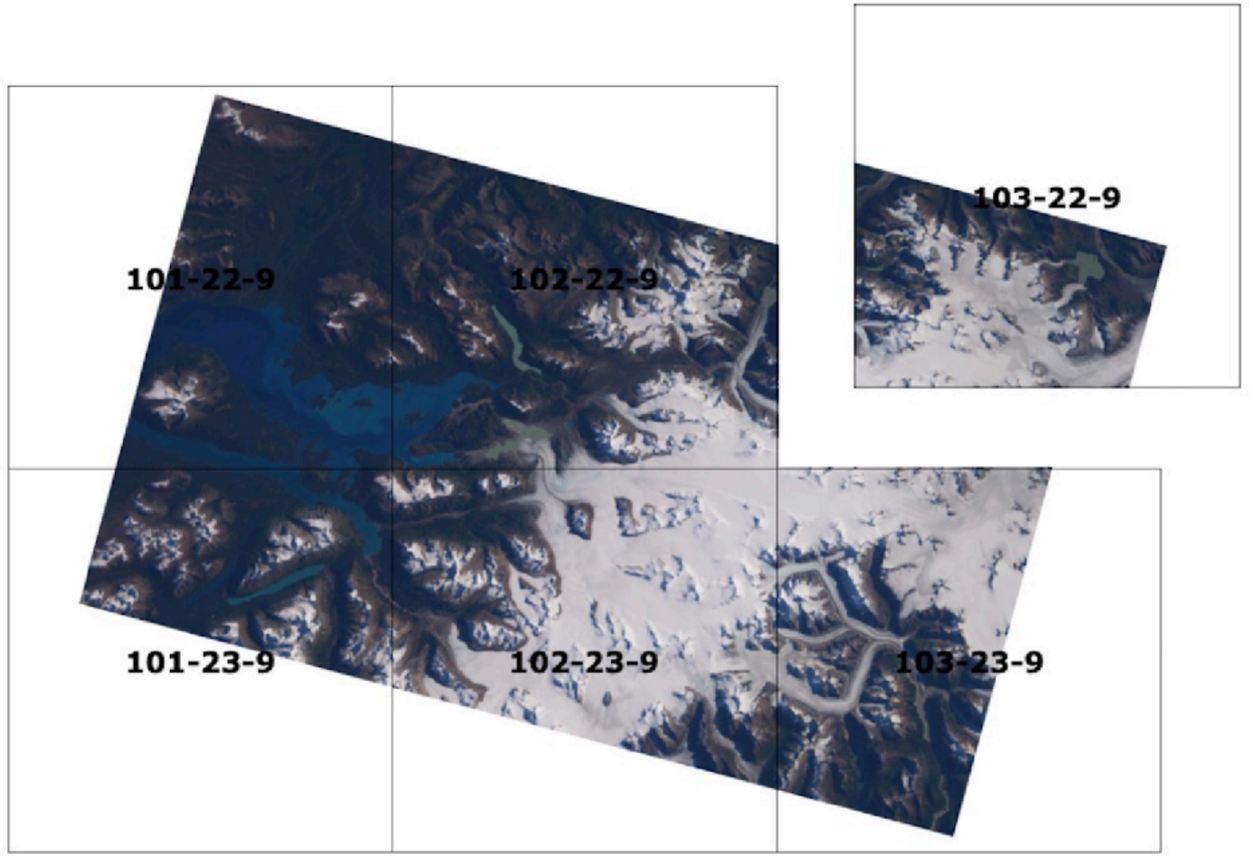
```
BoundingBox(left=-15028131.257091932, bottom=12523442.714243278, right=-12523442.714243278, top=15028131.257091932)
BoundingBox(left=-135.0, bottom=74.01954331150226, right=-112.5, top=79.17133464081945)
```

```python
print(webmercator.crs)
```

```
root='http://www.opengis.net/def/crs/EPSG/0/3857'
```

Data access: rasterio/GDAL

rio-tiler



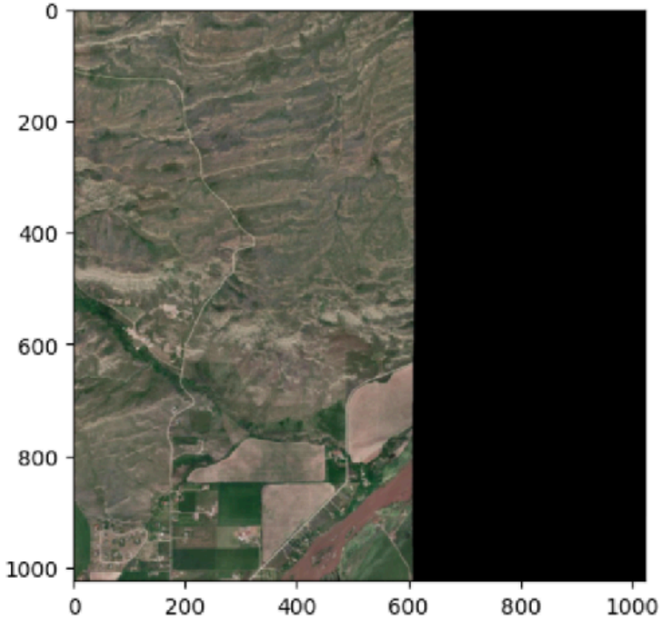
User friendly Rasterio plugin to read raster datasets.

Raster

```
from rio_tiler.io import Reader

with Reader(
    "https://maxar-opendata.s3.amazonaws.com/events/yellowstone-flooding22/ard/12/120000002133/2022-06-18/10300100D51B8C00-visual.tif",
) as src:
    print(src.info().model_dump())

>> {
    'bounds': (-110.55249247234418, 45.6909821481078, -110.48382520510808, 45.73908503122822),
    'minzoom': 11,
    'maxzoom': 18,
    'band_metadata': [('b1', {}), ('b2', {}), ('b3', {})],
    'band_descriptions': [('b1', ''), ('b2', ''), ('b3', '')],
    'dtype': 'uint8',
    'nodata_type': 'Mask',
    'colorinterp': ['red', 'green', 'blue'],
    'scales': [1.0, 1.0, 1.0],
    'offsets': [0.0, 0.0, 0.0],
    'colormap': None,
    'driver': 'GTiff',
    'count': 3,
    'width': 17408,
    'height': 17408,
    'overviews': [2, 4, 8, 16, 32, 64]
}
```

Raster

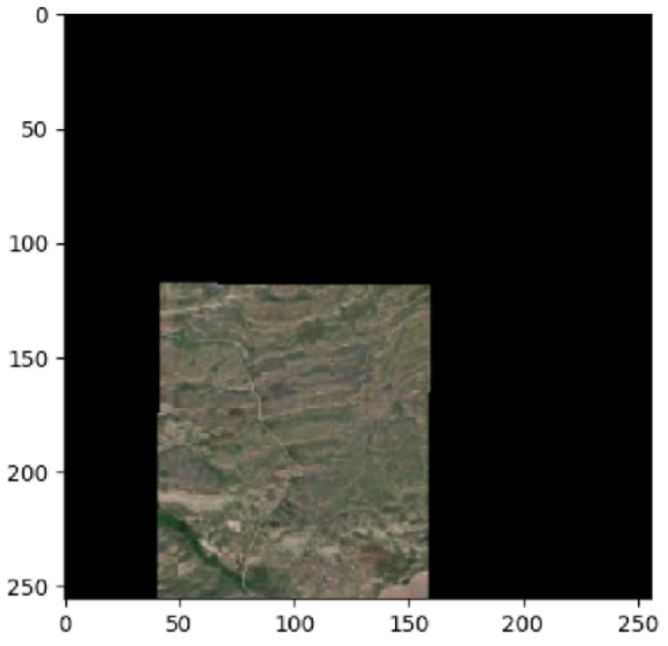
```
from rio_tiler.io import Reader
from matplotlib.pyplot import imshow

with Reader(
    "https://maxar-opendata.s3.amazonaws.com/events/yellowstone-flooding22/ard/12/120000002133/2022-06-18/10300100D51B8C00-visual.tif",
) as src:
    img = src.preview()
    imshow(img.data_as_image())
```



Raster

```
from rio_tiler.io import Reader
from matplotlib.pyplot import imshow

with Reader(
    "https://maxar-opendata.s3.amazonaws.com/events/yellowstone-flooding22/ard/12/120000002133/2022-06-18/10300100D51B8C00-visual.tif",
) as src:
    bounds = src.geographic_bounds
    tile = src.tms.tile(bounds[0], bounds[3], src.minzoom + 1)
    print(tile)

    img = src.tile(*tile)
    imshow(img.data_as_image())
```

```
Tile(x=790, y=1461, z=12)
```



Raster

```python
from rio_tiler.io import Reader, STACReader

# rio_tiler.io.rasterio.Reader
with Reader(
    "https://maxar-opendata.s3.amazonaws.com/events/yellowstone-flooding22/ard/12/120000002133/2022-06-18/10300100D51B8C00-visual.tif",
) as src:
    print(src.info().model_dump())

>> {'bounds': (-110.55249247234418, 45.6909821481078, -110.48382520510808, 45.73908503122822), 'minzoom': 11, 'maxzoom': 18, ...}


# rio_tiler.io.stac.STACReader
with STACReader("https://stac.eoapi.dev/collections/MAXAR_yellowstone_flooding22/items/12_120000002133_10300100D51B8C00") as stac:
    print(stac.info(assets="visual")["visual"].model_dump())

>> {'bounds': (-110.55249247234418, 45.6909821481078, -110.48382520510808, 45.73908503122822), 'minzoom': 11, 'maxzoom': 18, ...}


from rio_viz.io import MultiFilesBandsReader
src_path = "https://sentinel-cogs.s3.us-west-2.amazonaws.com/sentinel-s2-l2a-cogs/34/S/GA/2020/3/S2A_34SGA_20200318_0_L2A/B{04,03,02}.tif"

# rio_viz.io.MultiFilesBandsReader
with MultiFilesBandsReader(src_path) as multi:
    print(multi.info().model_dump())


>> {'bounds': (23.106076243528157, 31.505173744374172, 24.296464503939948, 32.519334871696195), 'minzoom': 8, 'maxzoom': 14, ...}
```

Raster

TiTiler = rio-tiler (readers) + morecantile (TMS) + FastAPI

Abstract Base class

rio-tiler's Reader
+tile(x, y, z)
+preview()
+part()
+info()
+statistics()
+point(lon, lat)

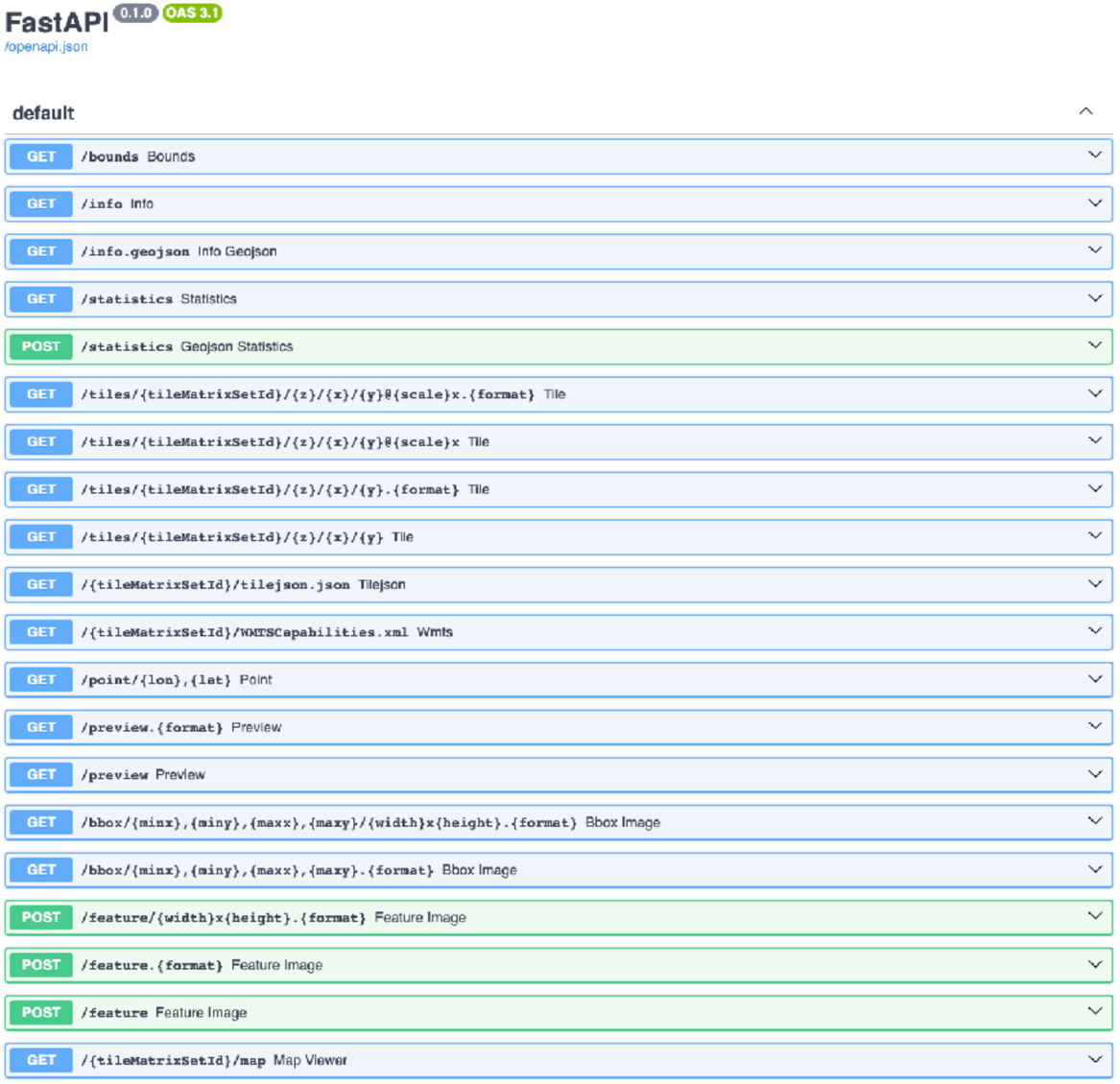
TiTiler's Endpoints
+/tiles/{z}/{x}/{y}.{format}
+/info
+/bbox/{minx},{miny},{maxx},{maxy}.{format}
+/point/{lon},{lat}
+/statistics
+/preview
+/WMTSCapabilities.xml

Endpoints Factories

FastAPI

Raster

```python
from fastapi import FastAPI

from rio_tiler.io import Reader

from titiler.core.factory import TilerFactory

# Create FastAPI application
app = FastAPI()

# Create router and register set of endpoints
cog = TilerFactory(
    reader=Reader,
    # enable optional endpoints
    add_preview=True,
    add_part=True,
    add_viewer=True,
)

# add router endpoint to the main application
app.include_router(cog.router)
```



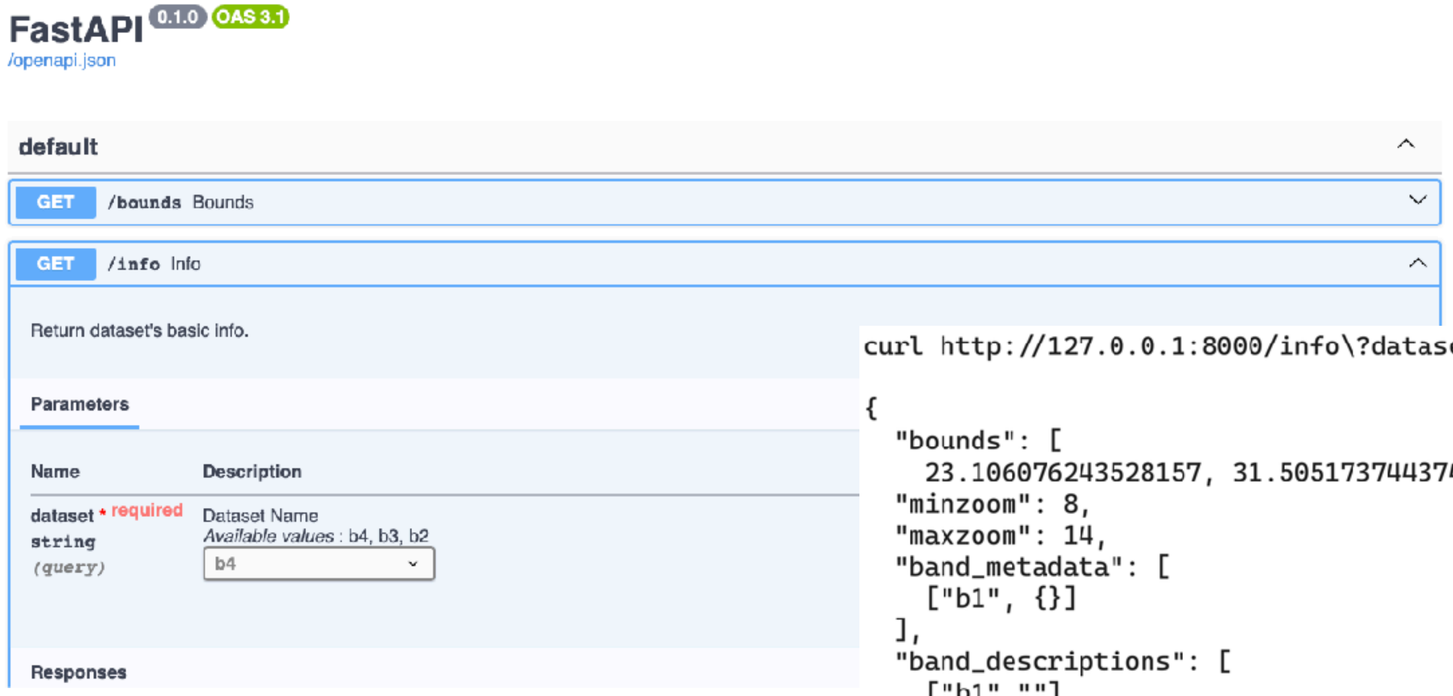
Factories + Dependency injection

```python
@get("/statistics")
def statistics(
    src_path=Depends(self.path_dependency),                          # the input for the reader (e.g COG URL) | reader init arg
    layer_params=Depends(self.layer_dependency),                     # indexes, ...                           | reader.preview() arg
    dataset_params=Depends(self.dataset_dependency),                 # nodata, rescaling, resampling, ...     | reader.preview() arg
    image_params=Depends(self.img_preview_dependency),               # max_size, ...                          | reader.preview() arg
    post_process=Depends(self.process_dependency),                   # custom algorithm
    stats_params=Depends(self.stats_dependency),                     # percentiles, categorical, ...          | ImageData.statistics() arg
    histogram_params=Depends(self.histogram_dependency),             # bin, ...                               | numpy.histogram() arg
    reader_params=Depends(self.reader_dependency),                   # Reader option defined at Factory level
    env=Depends(self.environment_dependency),                        # GDAL env (dynamic or static)
):
    with rasterio.Env(**env):
        with self.reader(src_path, **reader_params) as src_dst:      # rio-tiler BaseReader
            image = src_dst.preview(                                 # BaseReader.preview, returns a `rio_tiler.models.ImageData` object
                **layer_params,
                **image_params,
                **dataset_params,
            )

            if post_process:
                image = post_process(image)

            return image.statistics(                                 # rio_tiler.models.ImageData.statistics
                **stats_params,
                hist_options={**histogram_params},
            )
```

Raster

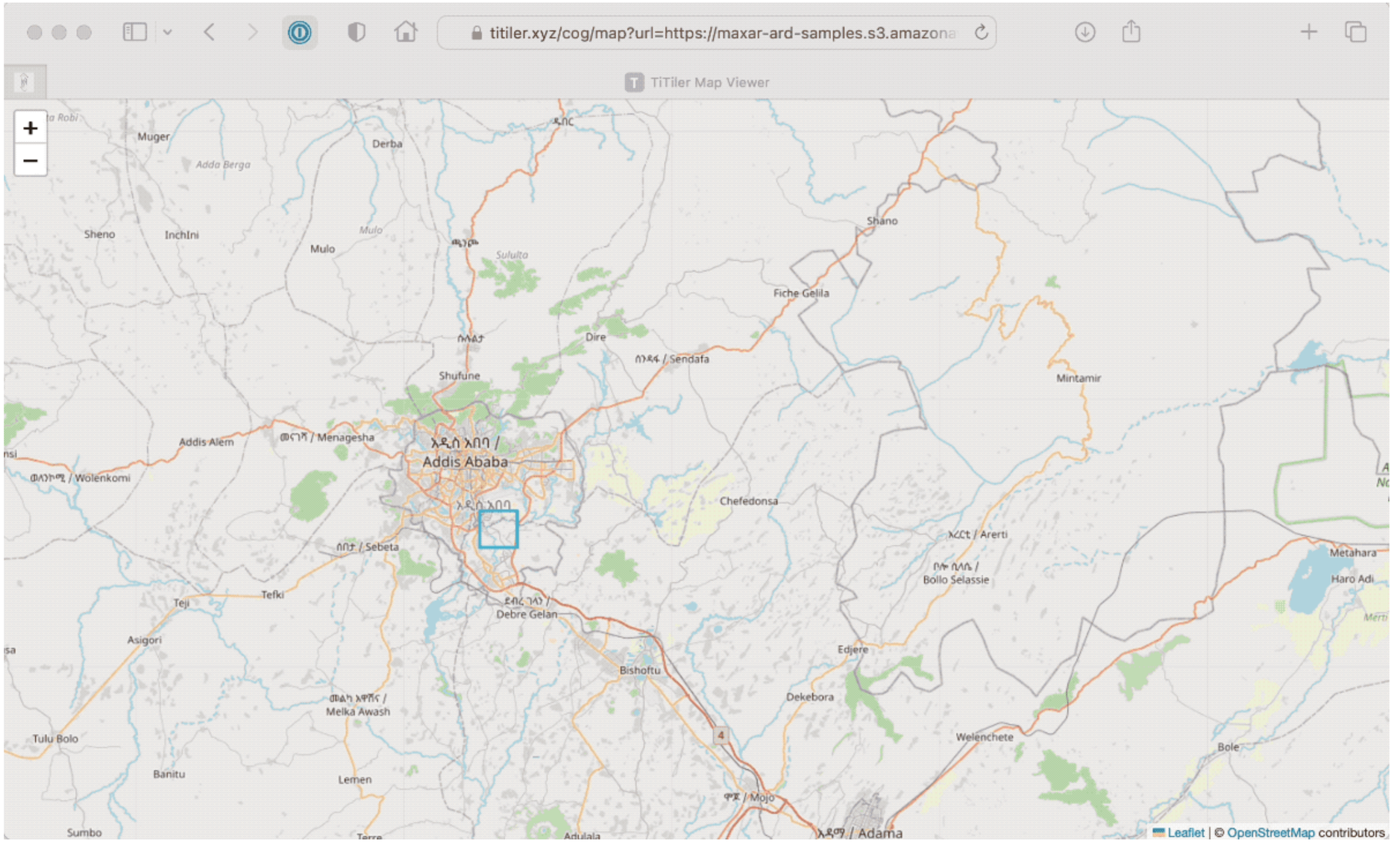
```python
from typing import Literal
from fastapi import FastAPI, Query
from titiler.core.factory import TilerFactory
from typing_extensions import Annotated


def DatasetPathParams(
    dataset: Annotated[
        Literal["b4", "b3", "b2"], Query(description="Dataset Name")]
) -> str:
    """Create dataset path from args"""
    datasets = {
        "b4": "https://sentinel-cogs.s3.us-west-2.amazonaws.com/sentinel-s2-l2a-cogs/34/S/GA/2020/3/S2A_34SGA_20200318_0_L2A/B04.tif",
        "b3": "https://sentinel-cogs.s3.us-west-2.amazonaws.com/sentinel-s2-l2a-cogs/34/S/GA/2020/3/S2A_34SGA_20200318_0_L2A/B03.tif",
        "b2": "https://sentinel-cogs.s3.us-west-2.amazonaws.com/sentinel-s2-l2a-cogs/34/S/GA/2020/3/S2A_34SGA_20200318_0_L2A/B02.tif",
    }
    return datasets[dataset]


app = FastAPI()
cog = TilerFactory(
    dataset_dependency=DatasetPathParams,
)
app.include_router(cog.router)
```

Raster

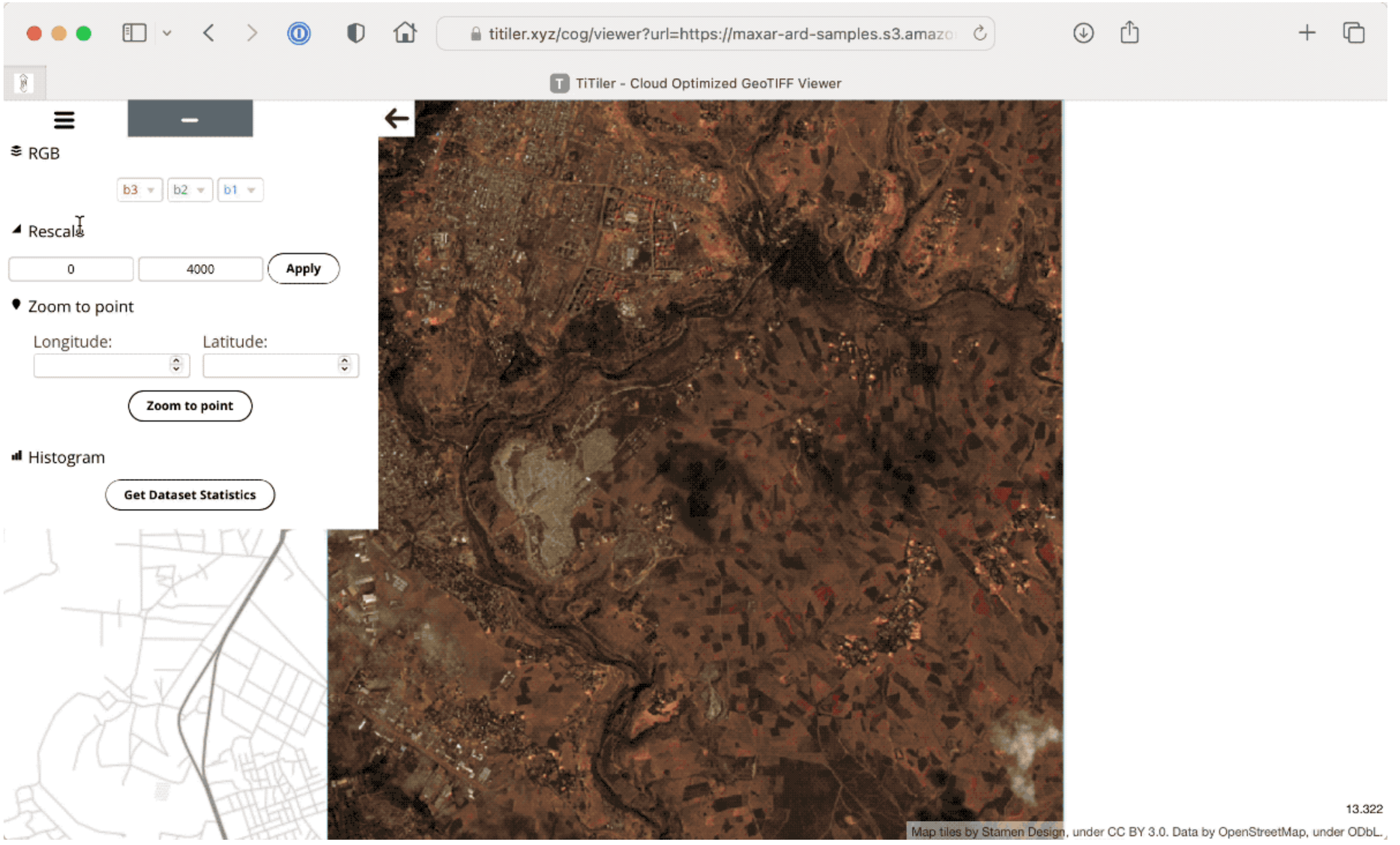



```
curl http://127.0.0.1:8000/info\?dataset\=b2 | jq

{
  "bounds": [
    23.106076243528157, 31.505173744374172, 24.296464503939948, 32.519334871696195],
  "minzoom": 8,
  "maxzoom": 14,
  "band_metadata": [
    ["b1", {}]
  ],
  "band_descriptions": [
    ["b1",""]
  ],
  "dtype": "uint16",
  "nodata_type": "Nodata",
  "colorinterp": ["gray"],
  "scales": [1.0],
  "offsets": [0.0],
  "driver": "GTiff",
  "count": 1,
  "width": 10980,
  "height": 10980,
  "overviews": [2, 4, 8, 16],
  "nodata_value": 0.0
}
```

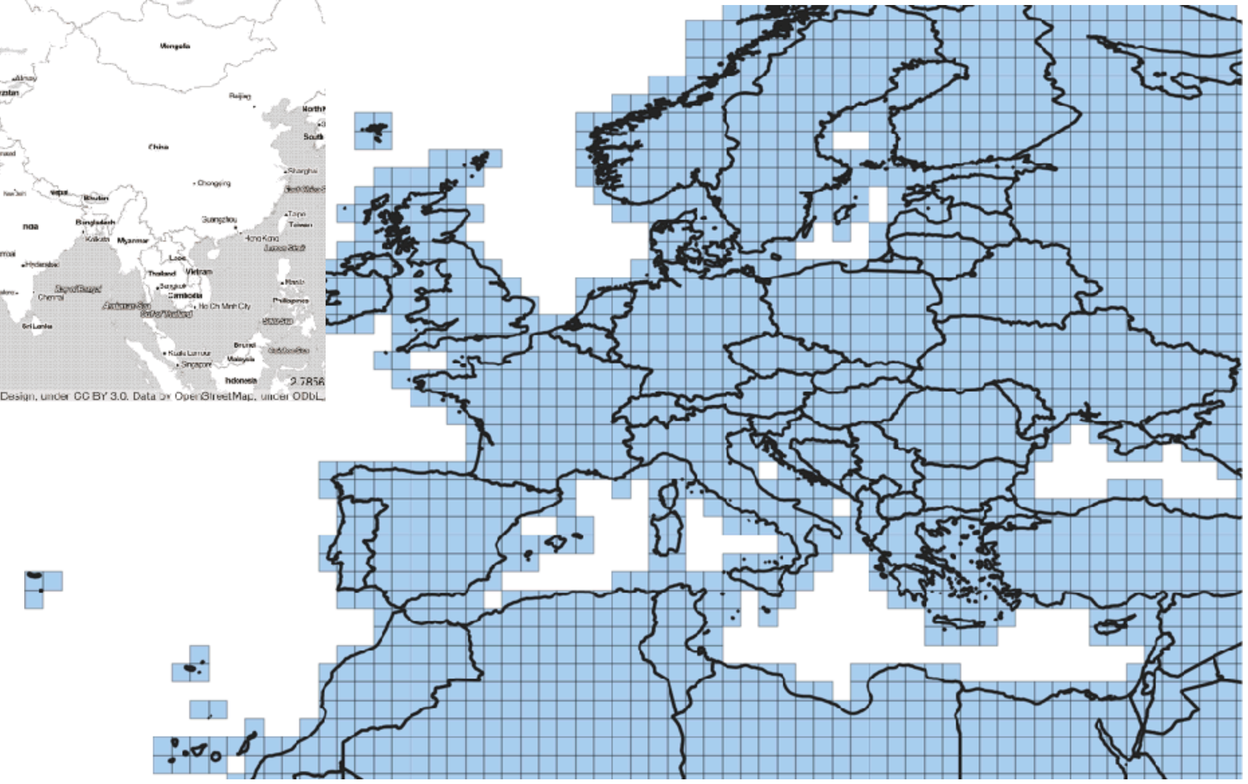
Raster


titiler.xyz/cog/map?url=https://maxar-ard-samples.s3.amazona
TiTiler Map Viewer
Muger
Derba
Adda Berga
Sheno
InchIni
Mulo
Shano
Fiche Gelila
Dire
Shufune
Addis Alem
መናገሻ / Menagesha
አዲስ አበባ / Addis Ababa
ወለንኮሚ / Wolenkomi
Chefedonsa
አረርቲ / Arerti
ሰበታ / Sebeta
Metahara
Haro Adi
Tefki
Teji
ደብረ ገላን / Debre Gelan
Asigori
Bishoftu
Edjere
Dekebora
መልካ አዋሽና / Melka Awash
Welenchete
Bole
Tulu Bolo
Banitu
Lemen
ሞጆ / Mojo
Sumbo
Adulala
አዳማ / Adama
Leaflet | © OpenStreetMap contributors

Raster

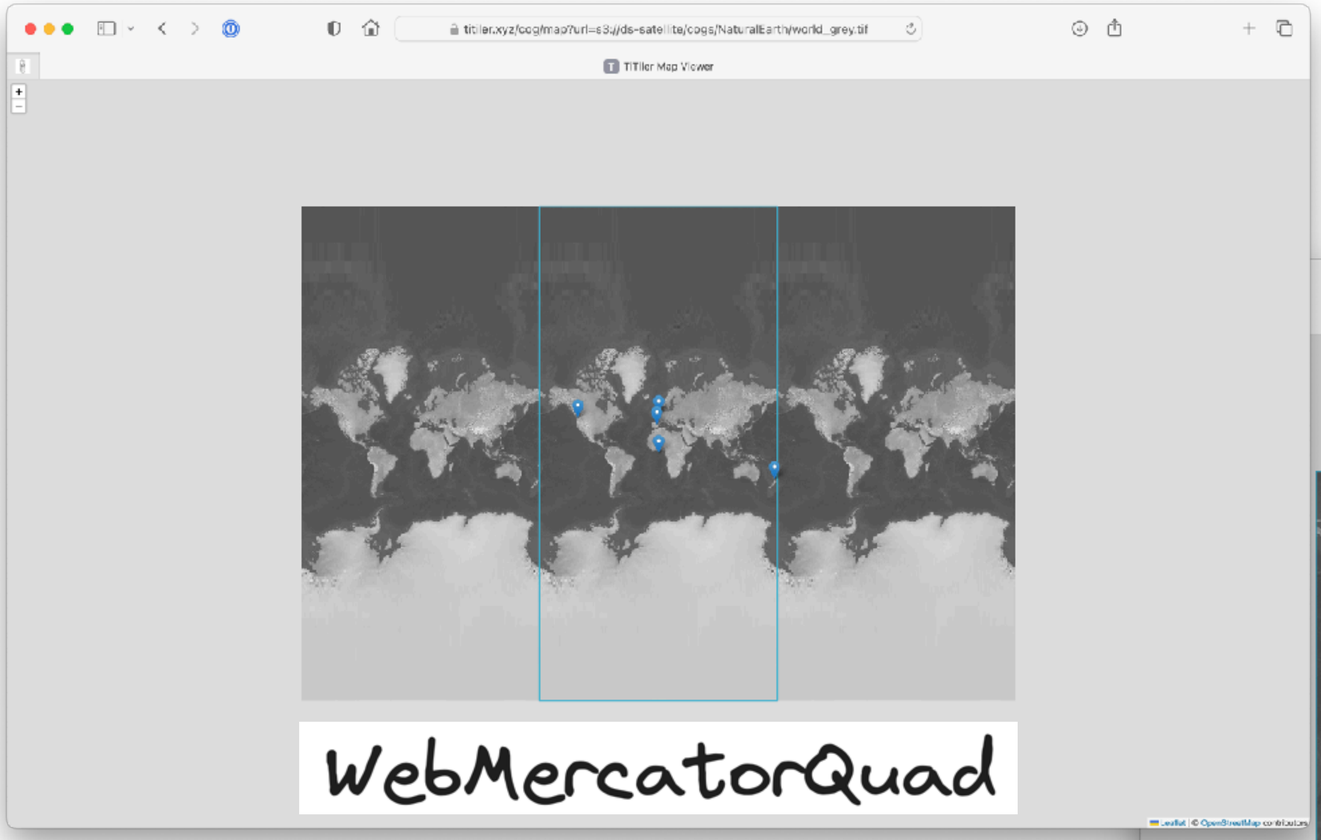

titiler.xyz/cog/viewer?url=https://maxar-ard-samples.s3.amazo
TiTiler - Cloud Optimized GeoTIFF Viewer
RGB
b3
b2
b1
Rescale
0
4000
Apply
Zoom to point
Longitude:
Latitude:
Zoom to point
Histogram
Get Dataset Statistics
13.322
Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.

Raster

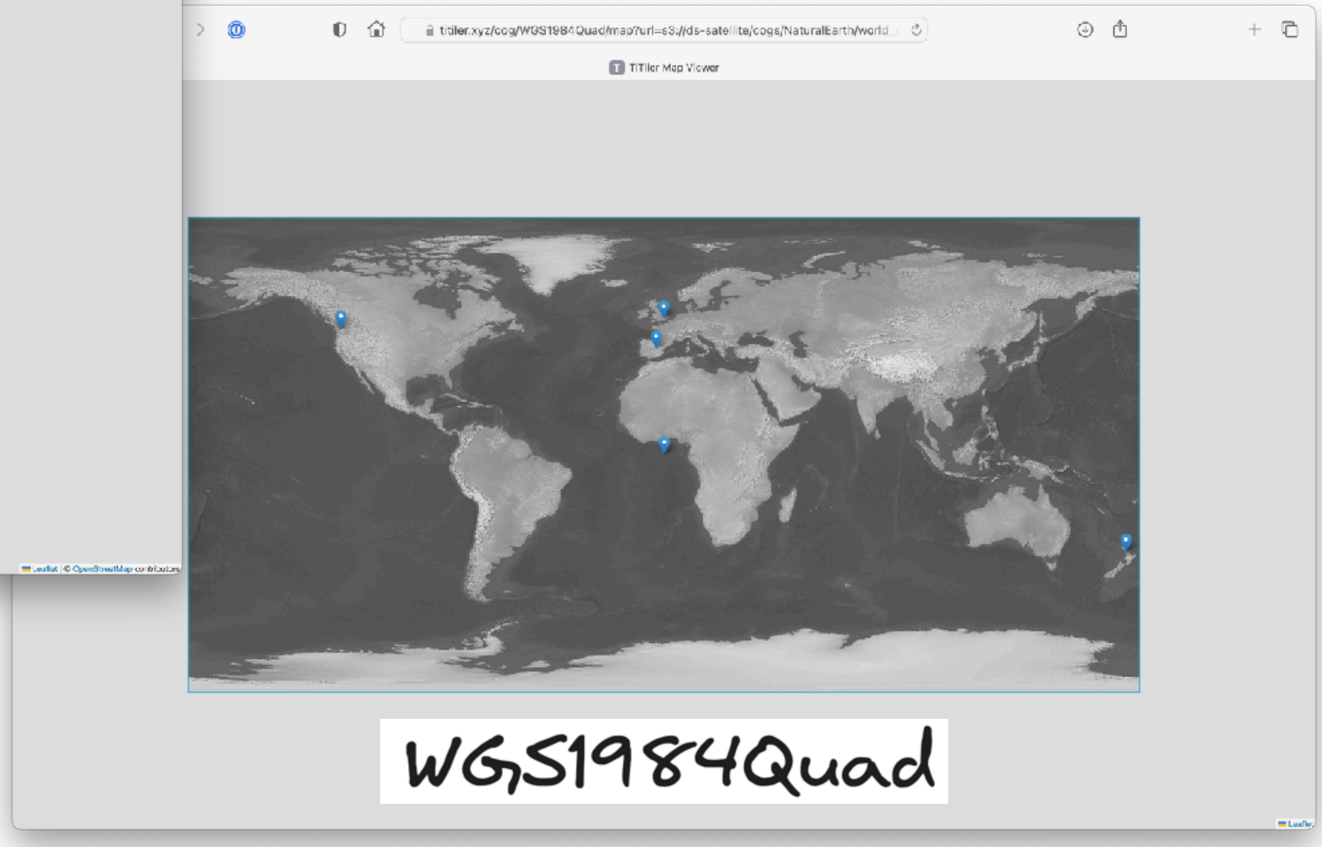


https://registry.opendata.aws/copernicus-dem/

Raster



WebMercatorQuad



WGS1984Quad

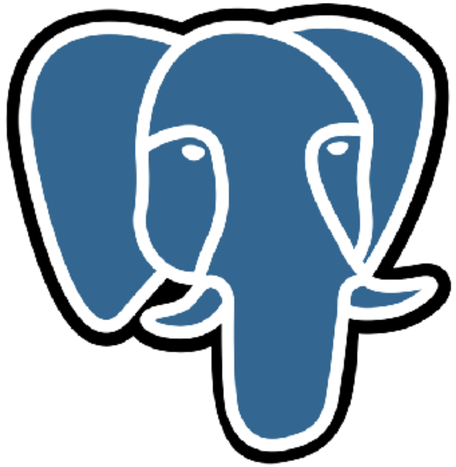
Raster

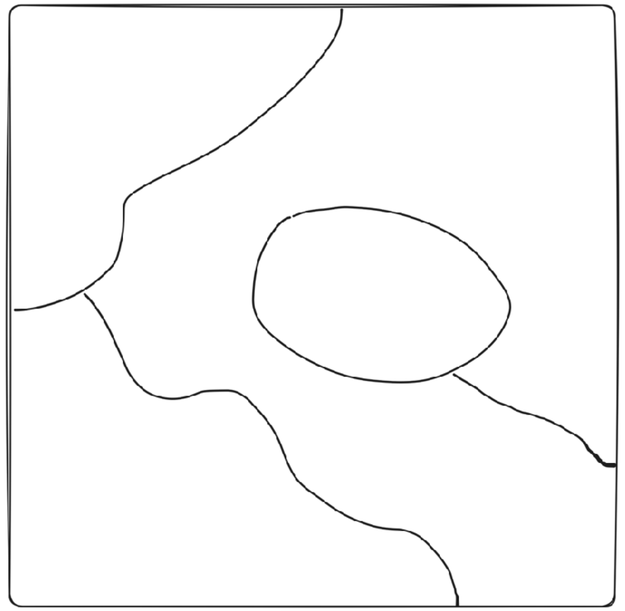
```
python -m pip install titiler.core titiler.application uvicorn

uvicorn titiler.application.main:app
```

TiPg
Vector / Features Services

https://github.com/developmentseed/tipg

Vector





Vector

Open Geospatial Consortium.

APIs for the Web

Features

Approved Standard

OGC API - Features - Part 1: Core and Part 2: Coordinate Reference Systems by Reference are both publicly available.

More Info | GitHub repo

Common

Approved Standard

OGC API - Common specifies those building blocks that are shared by most or all OGC API Standards to ensure consistency across the family.

More Info | GitHub repo

EDR

Approved Standard

Environmental Data Retrieval (EDR) API provides a family of lightweight interfaces to access Environmental Data resources. Each resource addressed by an EDR API maps to a defined query pattern.

More Info | GitHub repo

Tiles

Approved Standard

OGC API - Tiles provides extended functionality to other OGC API Standards to deliver vector tiles, map tiles, and other tiled data.

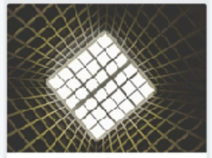
More Info | GitHub repo

Processes

Approved Standard

OGC API - Processes allows for processing tools to be called and combined from many sources and applied to data in other OGC API resources though a simple API.

More Info | GitHub repo



Coverages

OGC API - Coverages allows discovery, visualization and query of complex raster stacks and data cubes.

More Info | GitHub repo

Records

OGC API - Records updates OGC's Catalog Services for the Web by building on the simple access to content in OGC API - Features.

More Info | GitHub repo

Styles

The OGC API - Styles defines a Web API that enables map servers, clients as well as visual style editors, to manage and fetch styles.

More Info | GitHub repo

Maps

OGC API - Maps offers a modern approach to the OGC Web Map Service (WMS) standard for provision map and raster content.

More Info | GitHub repo

DGGS

Enables applications to organise and access data arranged according to a Discrete Global Grid System (DGGS).

More Info | GitHub repo

Routes

Enables applications to request routes in a manner independent of the underlying routing data set, routing engine or algorithm.

More Info | GitHub repo

Joins

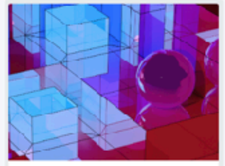
OGC API - Joins supports the joining of data, from multiple sources, with feature collections or directly with other input files.

More Info | GitHub repo

Moving Features

OGC API - Moving Features defines an API that provides access to data representing features that move as rigid bodies.

More Info | GitHub repo



3D GeoVolumes

OGC API - 3D GeoVolumes facilitates efficient discovery of and access to 3D content in multiple formats based on a space-centric perspective.

More Info | GitHub repo

Connected Systems

OGC API - Connected Systems act as a bridge between static and dynamic data collected by sensors.

More Info | GitHub repo

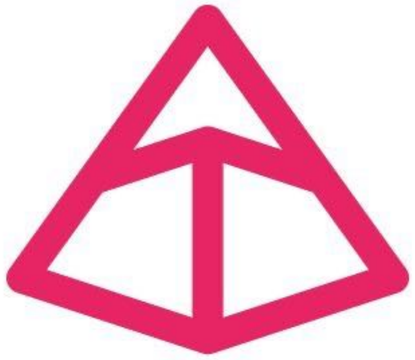
APIs for the IoT

SensorThings

The OGC SensorThings API provides an open, geospatial-enabled and unified way to interconnect Internet of Things (IoT) devices, data, and applications over the Web.

More Info | GitHub repo

Vector



Pydantic Models

Vector

https://github.com/koxudaxi/datamodel-code-generator/

```yaml
type: object
required:
  - id
  - links
properties:
  id:
    description: identifier of the collection used, for example, in URIs
    type: string
    example: address
  title:
    description: human readable title of the collection
    type: string
    example: address
  description:
    description: a description of the features in the collection
    type: string
    example: An address.
  links:
    type: array
    items:
      $ref: link.yaml
    example:
      - href: http://data.example.com/buildings
        rel: item
      - href: http://example.com/concepts/buildings.html
        rel: describedby
        type: text/html
  linkTemplates:
    type: array
    items:
      $ref: linkTemplate.yaml
  extent:
    $ref: extent.yaml
  itemType:
    description: indicator about the type of the items in the collection (the default value is 'feature').
    type: string
    default: feature
  crs:
    description: the list of coordinate reference systems supported by the service
    type: array
    items:
      type: string
    default:
      - http://www.opengis.net/def/crs/OGC/1.3/CRS84
    example:
      - http://www.opengis.net/def/crs/OGC/1.3/CRS84
      - http://www.opengis.net/def/crs/EPSG/0/4326
```

```python
class Collection(BaseModel):
    id: str
    title: Optional[str] = None
    description: Optional[str] = None
    links: List[Link]
    extent: Optional[Extent] = None
    itemType: str = "feature"
    crs: List[str] = ["http://www.opengis.net/def/crs/OGC/1.3/CRS84"]

    model_config = {"extra": "ignore"}
```

Vector

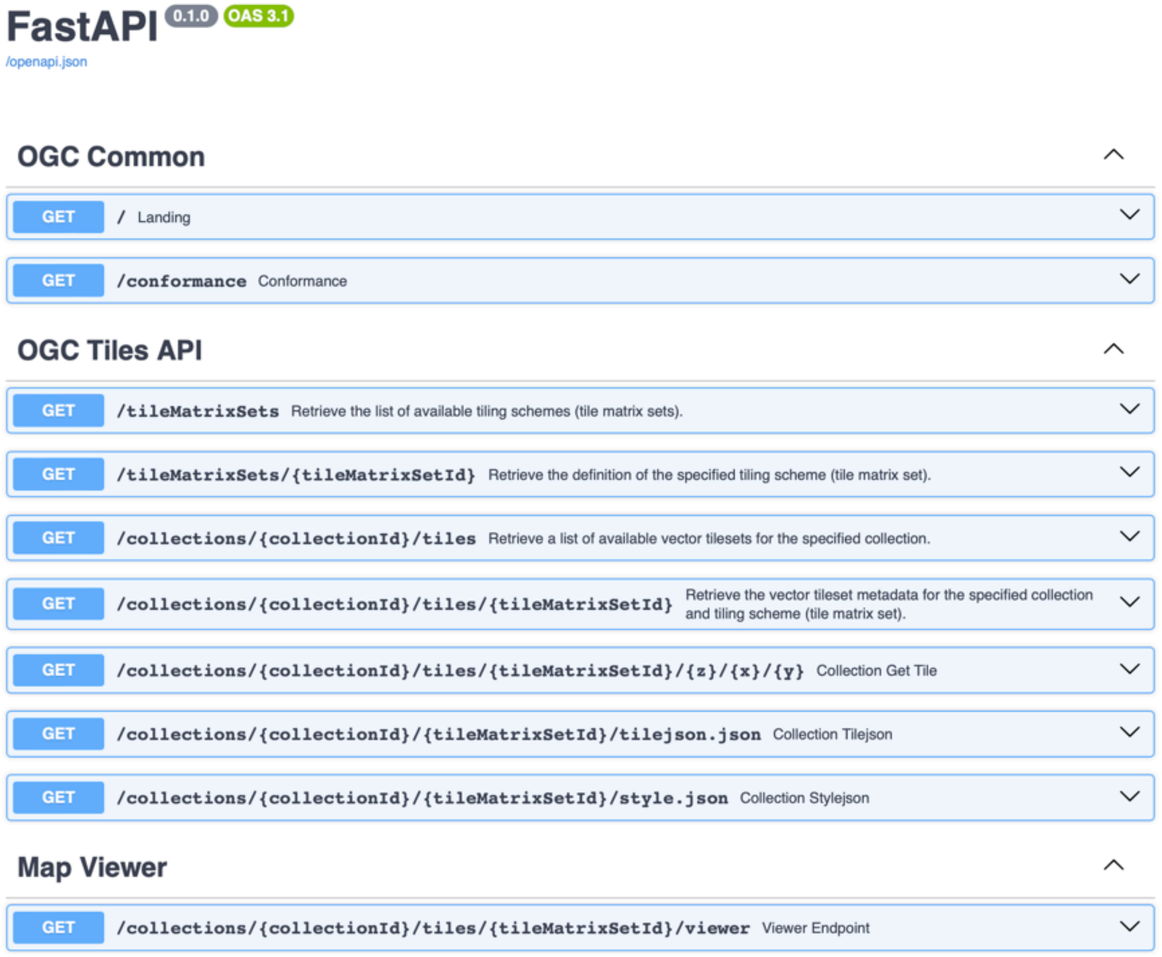
```python
class Collection(BaseModel):

    type: str
    id: str
    table: str
    dbschema: str = Field(alias="schema")

    async def get_features(self, *args, **kwargs):
        ...

    async def get_tile(self, *args, **kwargs):
        ...
```

Vector

```python
# pseudo code
class Factory:

    collection_dependency: Callable[..., Collection]

    def __init__(self, collection_dependency: Callable):
        self.collection_dependency = collection_dependency
        self.router = APIRouter()

        self.register_routes()

    def register_routes(self):

        @self.router.get("/collections/{collectionId}/tiles/{tileMatrixSetId}/{z}/{x}/{y}")
        async def tiles(
            request: Request,
            collection=Depends(self.collection_dependency),
            tileMatrixSetId=Annodated[
                Literal[tuple(self.supported_tms.list())],
                Path(description="Identifier selecting one of the TileMatrixSetId supported ")
            ],
            z: Annotated[int, Path(description="Zoom Level.")],
            x: Annotated[int, Path(description="X Level.")],
            y: Annotated[int, Path(description="Y Level.")],
        ):
            ...
            return await collection.get_tile(tms, x, y, z)
```

Vector

```python
from contextlib import asynccontextmanager

from fastapi import FastAPI
from tipg.collections import register_collection_catalog
from tipg.database import close_db_connection, connect_to_db
from tipg.factory import OGCTilesFactory


@asynccontextmanager
async def lifespan(app: FastAPI):
    await connect_to_db(app)
    await register_collection_catalog(app)
    yield
    await close_db_connection(app)

app = FastAPI()
endpoints = OGCTilesFactory(with_common=True)
app.include_router(endpoints.router)
```



Vector

tipg.factory.OGCFeaturesFactory ← OGC Features API

tipg.factory.OGCTilesFactory ← OGC Tiles API

tipg.factory.Endpoints ← OGC Features & Tiles API

Vector

```
python -m pip install tipg uvicorn

export DATABASE_URL=postgresql://username:password@0.0.0.0:5432/postgis
uvicorn tipg.main:app
```

Ping Me!

@_VincentS_
@cogeotiff
@developmentseed

Join the team & make a better planet.
https://developmentseed.org/careers