MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

PROCESSAMENTO E ANÁLISES DE DADOS DE PROPRIEDADES ÓPTICAS DE MASSAS DE ÁGUA PARA AVALIAÇÃO DE QUALIDADE DA ÁGUA EM RESERVATÓRIOS E LAGOS DA PLANÍCIE DE INUNDAÇÃO AMAZÔNICA

Ariana Rodrigues Cursino

Relatório de Iniciação Científica do programa PIBIC, orientada pelo Dr. Cláudio Clemente Faria Barbosa.

INPE
São José dos Campos
2021

Resumo

O Laboratório de Instrumentação para Sistemas Aquáticos (LabISA), tem coletado dados in situ e integrado estes dados às imagens de sensores remotos para estudar e monitorar rios e lagos. Este trabalho tem como objetivo principal desenvolver uma aplicação 'web' com uma interface amigável e intuitiva para demonstrar os produtos gerados pelo laboratório aos usuários finais. O processo foi divido em 2 módulos e a linguagem de programação python foi escolhida para realização de todas as etapas. A primeira parte foi o desenvolvimento da aplicação utilizando o framework Dash Plotly e Flask. Os sistemas gerenciadores de banco de dados objeto relacional (SGBD) foi o PostgreSQL e SQLite. Esta interface no momento permite apenas a visualização dos produtos gerados pelo laboratório, porém, já foi iniciado o processo em que o usuário poderá fazer uma busca por data e descarregar os produtos escolhidos. A segunda etapa foi padronizar a localização das imagens que serão inseridas na aplicação, o sistema de catalogação seguiu o padrão Spatio Temporal Asset Catalog (STAC) utilizando as Application Programing Interface (API) do Brazil Data Cube (BDC) e a biblioteca python pystac.

Palavras-chave: Python, Flask, STAC

LISTA DE FIGURAS

Pág.

SUMÁRIO

Pág.

1 – Introdução

A crescente acessibilidade e disponibilidade de imagens de satélite nos últimos anos traz muitos benefícios para as mais diversas áreas, principalmente para as áreas com análise aquática. O Laboratório de Instrumentação para Sistemas Aquáticos (LabISA), para fim de realização de seus estudos sobre as caracterizações e mudanças ambientais em ambientes aquáticos, faz a utilização de diversos equipamentos para coleta de dados, dentre eles os satélites, esses que serão utilizados para desenvolvimento de metodologias para monitoramento por sensoriamento remoto de rios e lagos.

Este projeto de iniciação científica visa a criação de uma aplicação ‘web’ para facilitar aos usuários acessar um sistema que consiga disponibilizar os produtos que o laboratório gera e ainda ofereça uma ‘interface’ simples para a busca de imagens, bem como descarregar as imagens e produtos gerados pelo laboratório.

2 – Atividades Desenvolvidas Durante o Período

Para organização e clareza do projeto o mesmo foi dividido em 3 etapas, descritas abaixo:

2.1 - Contextualização

Ao iniciar as atividades, foi realizado uma contextualização com alguns dos assuntos tratados pelo LabISA, sendo estes o sensoriamento remoto e satélites, OLI/LandSat 8 e MSI/Sentinel-2.

A introdução a cada um destes conceitos, foi feita para que o contato com os programas, processos e estruturas de dados já desenvolvidos no laboratório fossem facilmente compreendidos. Desta forma após a apresentação dos conceitos básicos, houve a interação com os processos do laboratório, estes relacionados ao acesso, e visualização dos dados presentes na servidora de dados.

Após conhecer os processos, foi realizado a familiarização com os programas já existentes no laboratório, que são utilizados para processamento e estudo dos dados.

2.2 - Desenvolvimento da Aplicação Web

Após toda contextualização com as atividades do laboratório, deu-se início aos estudos das estruturas de dados das coleções de imagens dos sensores OLI/LandSat 8 e do MSI/Sentinel-2, utilizando a linguagem Python.

Com o conhecimento das coleções de imagens, iniciou-se o processo de engenharia do programa, para que o projeto atendesse os requisitos dos usuários.

Foi utilizado o framework Flask pois possui uma estrutura leve de aplicativo da web WSGI, que foi projetado agilizar os primeiros passos e com a capacidade de escalar para aplicativos complexos. E o framework Dash-Plotly pois facilita a visualização de dados com interfaces de usuário altamente e utiliza a linguagem Python puro para a realização da interface. Para inserir os mapas, a biblioteca Folium.

Após definição do layout o desenvolvimento da plataforma Web foi iniciada. A Figura 2.1, mostra a tela inicial. Há as opções de navegar no mapa, utilizar algumas ferramentas nativas da biblioteca Folium, trocar o tipo de visualização do mapa. Além das opções de abas "Pesquisa" ou "Resultado".

Figura 2.1 - Protótipo MapaQuali – Tela Inicial

Fonte: Autora

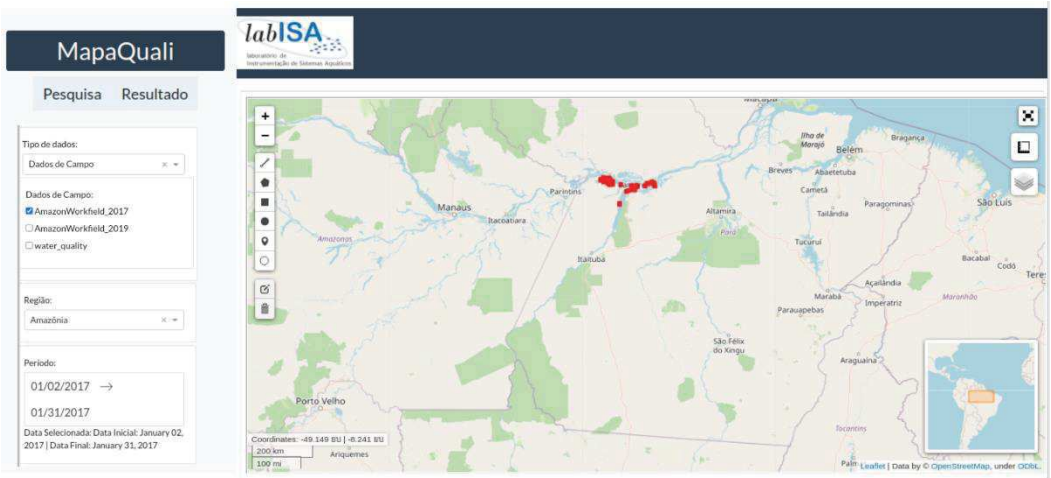
As Figuras 2.2, mostram a aba "Pesquisa", onde é possível escolher o tipo de dado, a região e o período desejado.

Figura 2.2 - Protótipo Mapaquali – Aba Pesquisa

Aba Pesquisa selecionada na opção Tipo de dados: Dados de Campo - water_quality, Região: Amazônia e Período: de 02 de janeiro à 31 de janeiro de 2017.
Fonte: Autora

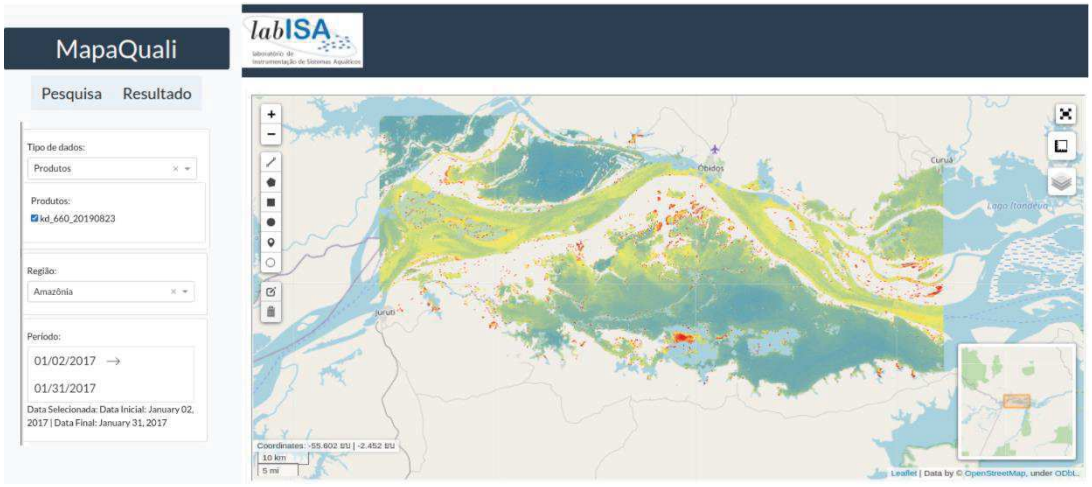
As Figuras 2.3 e 2.4, mostram a aba "Resultado" com os resultados das pesquisas realizadas na aba "Pesquisa".

Figura 2.3 - Protótipo MapaQuali – Aba Resultado A



Aba "Resultado" onde na aba Pesquisa foi selecionada na opção Tipo de dados: Dados de Campo - AmazonWorkfield_2017, Região: Amazônia e Período: de 02 de janeiro à 31 de janeiro de 2017.
Fonte: Autora

Figura 2.4 - Protótipo MapaQuali – Aba Resultado B



Aba "Resultado" onde na aba Pesquisa foi selecionada na opção Tipo de dados: Produto – kd_660, Região: Amazônia e Período: de 02 de janeiro à 31 de janeiro de 2017.
Fonte: Autora

A aplicação web contém apenas a parte da visualização dos dados de acordo com a região e o período escolhido. Para a realização de descarregamento das imagens, outras etapas são necessárias para padronização e o armazenamento adequado das imagens no repositório.

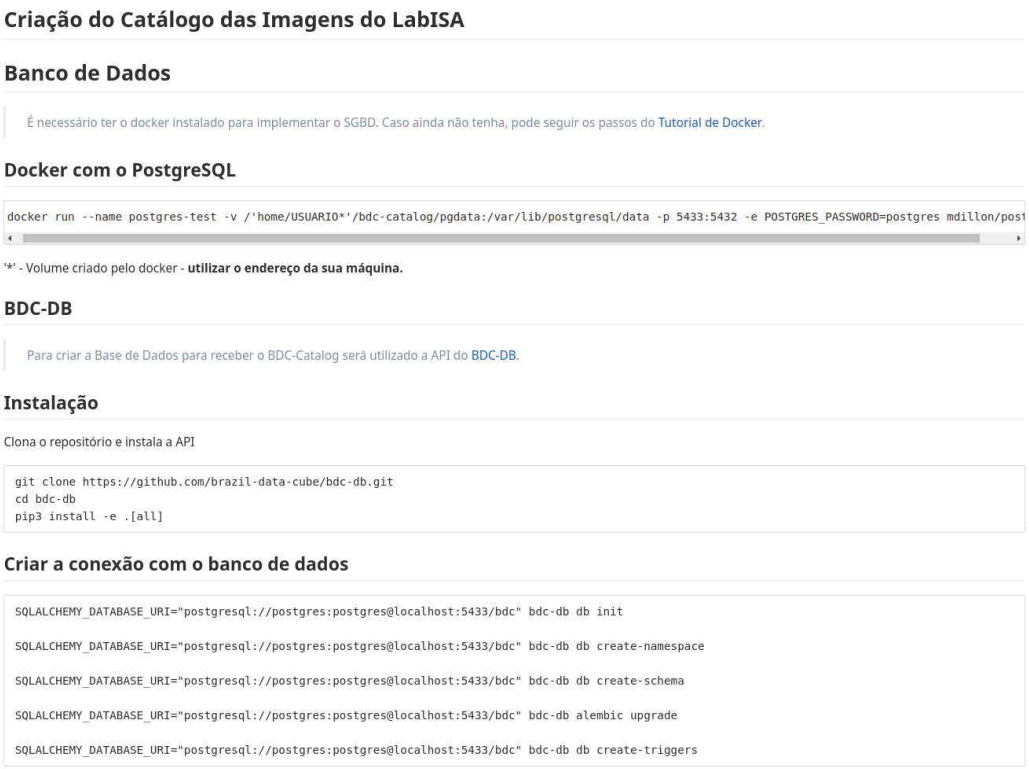
2.3 - Padronização STAC

Para padronização do armazenamento das imagens e produtos utilizados pelo Laboratório a especificação SpatioTemporal Asset Catalog (STAC) foi seguida. A STAC fornece uma linguagem comum para descrever uma variedade de informações geoespaciais, para que possam ser indexadas. Objetivo é que todos os dados e informações dos produtos espaço-temporais (imagens, SAR, cubos de dados, etc.) sejam catalogados para que o código não precise ser escrito sempre que um novo conjunto de dados ou API é lançado (STAC, [s.d.]).

- Item STAC é a unidade atômica central, representando um único ativo espaço-temporal como um recurso GeoJSON com data, hora e links.
- Catálogo STAC é um arquivo JSON simples e flexível de links que fornece uma estrutura para organizar e navegar pelos itens STAC.
- A Coleção STAC é uma extensão do Catálogo STAC com informações adicionais, como extensões, licenças, palavras-chave, provedores, etc. que descrevem os Itens STAC que se enquadram na Coleção.

Para construir a Coleção STAC foi utilizado as Application Programing Interface (API) do Brazil Data Cube (BDC) e a biblioteca python pystac para a catalogação. A primeira etapa foi a preparação do banco de dados (BDC-DB, 2020), foi utilizado o Sistema de Gerenciamento de Banco de Dados (SGBD) PostgreGIS 2.5 e contêiner Docker. A imagem do docker utilizada baseada no conteiner oficial do PostgreSQL (MDILLON 2021). A Figuras 2.5, descreve os passos para a criação do banco de dados.

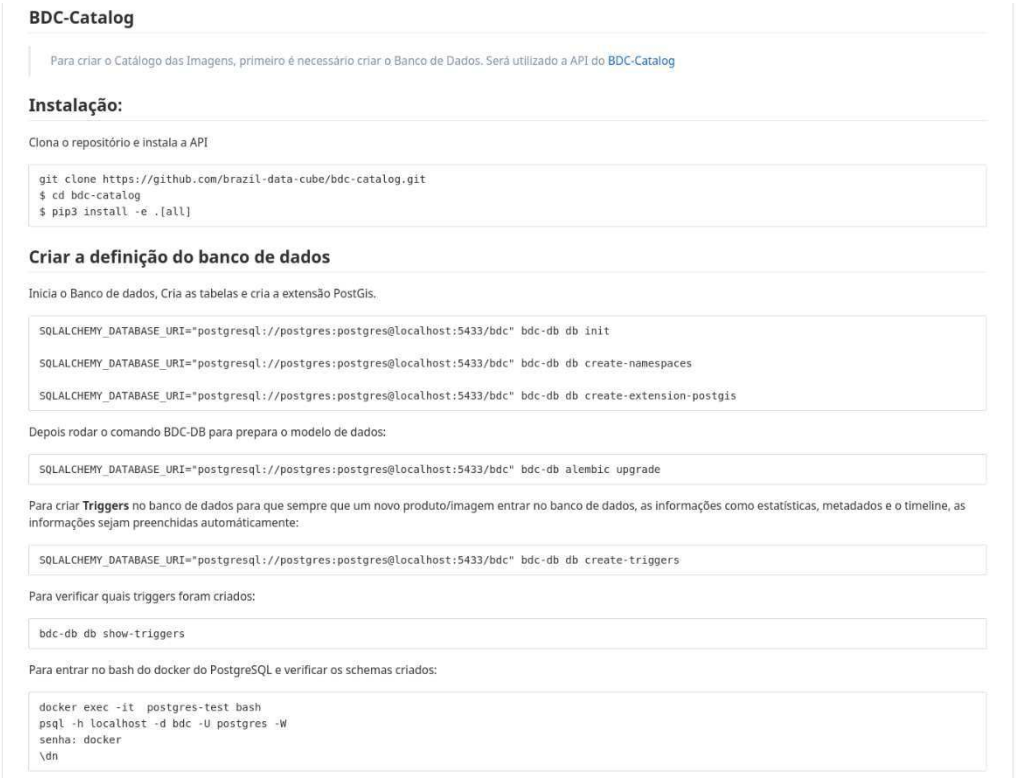
Figura 2.5 - Criação do Banco de dados



Etapas para criar e conectar com o banco de dados utilizando a API BDC-DB.
Fonte: Autora

A Figura 2.6 mostra os passos para a criação do catálogo, que utilizou a API (BDC-CATALOG, 2019).

Figura 2.6 - Criação do Catálogo



BDC-Catalog

Para criar o Catálogo das Imagens, primeiro é necessário criar o Banco de Dados. Será utilizado a API do BDC-Catalog

Instalação:

Clona o repositório e instala a API

```
git clone https://github.com/brazil-data-cube/bdc-catalog.git
$ cd bdc-catalog
$ pip3 install -e .[all]
```

Criar a definição do banco de dados

Inicia o Banco de dados, Cria as tabelas e cria a extensão PostGis.

```
SQLALCHEMY_DATABASE_URI="postgresql://postgres:postgres@localhost:5433/bdc" bdc-db db init

SQLALCHEMY_DATABASE_URI="postgresql://postgres:postgres@localhost:5433/bdc" bdc-db db create-namespaces

SQLALCHEMY_DATABASE_URI="postgresql://postgres:postgres@localhost:5433/bdc" bdc-db db create-extension-postgis
```

Depois rodar o comando BDC-DB para prepara o modelo de dados:

```
SQLALCHEMY_DATABASE_URI="postgresql://postgres:postgres@localhost:5433/bdc" bdc-db alembic upgrade
```

Para criar **Triggers** no banco de dados para que sempre que um novo produto/imagem entrar no banco de dados, as informações como estatísticas, metadados e o timeline, as informações sejam preenchidas automáticamente:

```
SQLALCHEMY_DATABASE_URI="postgresql://postgres:postgres@localhost:5433/bdc" bdc-db db create-triggers
```

Para verificar quais triggers foram criados:

```
bdc-db db show-triggers
```

Para entrar no bash do docker do PostgreSQL e verificar os schemas criados:

```
docker exec -it  postgres-test bash
psql -h localhost -d bdc -U postgres -W
senha: docker
\dn
```

Etapas para iniciar o banco de dados e criar as tabelas para que as informações do catálogo sejam persistidas na base de dados.
Fonte: Autora

A Figura 2.7 mostra os passos para a criar a conexão do banco de dados com o catálogo criados logo nas etapas acima e utilizou a API (BDC-STAC, 2019).

Figura 2.7 - Implementação da API do BDC-CATALOG com o Banco de dados

STAC

Será adaptado a API do BDC-STAC do projeto Brazil Data Cube para elaboração do Catálogo. Mais informações sobre a API.

Instalação:

Clona o repositório e instala a API

```
git clone https://github.com/brazil-data-cube/bdc-stac.git
$ cd bdc-stac
$ pip3 install -e .[all]
```

Implementação:

Cria o network e anexa o docker do banco de dados.

```
docker network create bdc_net
docker network connect bdc_net postgres-test
```

**Observação: postgres-test é o nome do container do banco de dados _

Etapas para criação da conexão da API com o banco de dados.
Fonte: Ariana

Para facilitar e agilizar o processo, o container com a imagem do PostGIS e a conexão com o container com o serviço STAC pode ser acionado apenas com as

opções de comando das Figura 2.8, para rodar em modo de produção ou da Figura 2.9, para rodar em modo de desenvolvimento.

Figura 2.8 - Exemplo de como rodar o contêiner do docker em modo de produção

Rodar o Docker com o Serviço de STAC

```
docker run --detach \
           --name bdc-stac \
           --publish 127.0.0.1:8080:5000 \
           --network=bdc_net \
           --env SQLALCHEMY_DATABASE_URI="postgresql://postgres:postgres@192.168.0.41:5433/bdc" \
           --env BDC_STAC_BASE_URL="http://localhost:8080" \
           --env BDC_STAC_FILE_ROOT="http://localhost:8081" \
           bdc-stac:0.9.0-0
```

Fonte: Autora

Figura 2.9 - Exemplo de como rodar o contêiner do docker em modo de desenvolvimento

Rodando em modo de Desenvolvimento:

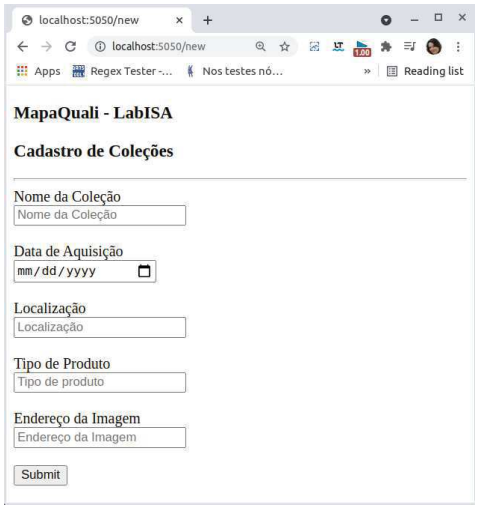
Toda a Base do BDC-Catalog deverá estar preparada

```
FLASK_APP="bdc_stac" \
  FLASK_ENV="development" \
  SQLALCHEMY_DATABASE_URI="postgresql://postgres:postgres@localhost:5433/bdc" \
  BDC_STAC_BASE_URL="http://localhost:5000" \
  BDC_STAC_API_VERSION="0.8.1" \
  BDC_STAC_FILE_ROOT="http://localhost:5001" \
  flask run
```

Fonte: Autora

Para o cadastro das imagens e produtos produzidos e que serão visualizados na plataforma Web, uma aplicação simples utilizando apenas as técnicas de CRUD (Create, Read, Update, Delete - é um acrônimo para as maneiras de se operar em informação de armazenamento persistente), a biblioteca SQLite e o framework Flask foi criada. A Figura 2.10 mostra a página para cadastro e criação de novas coleções com as informações como nome, data de aquisição, a localização, tipo de produto e o endereço onde a imagem se encontra (seja na máquina local ou em algum servidor).

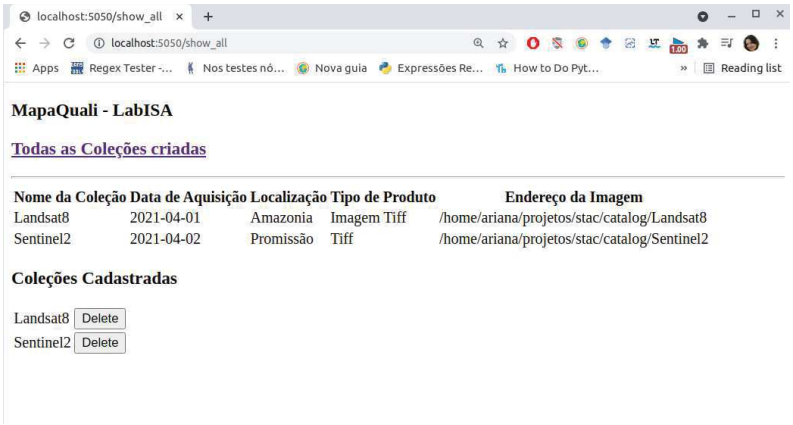
Figura 2.10 - Cadastro de Coleções



Fonte: Autora

A Figura 2.11, mostra todas as coleções cadastradas e a possibilidade de deletar as coleções cadastradas.

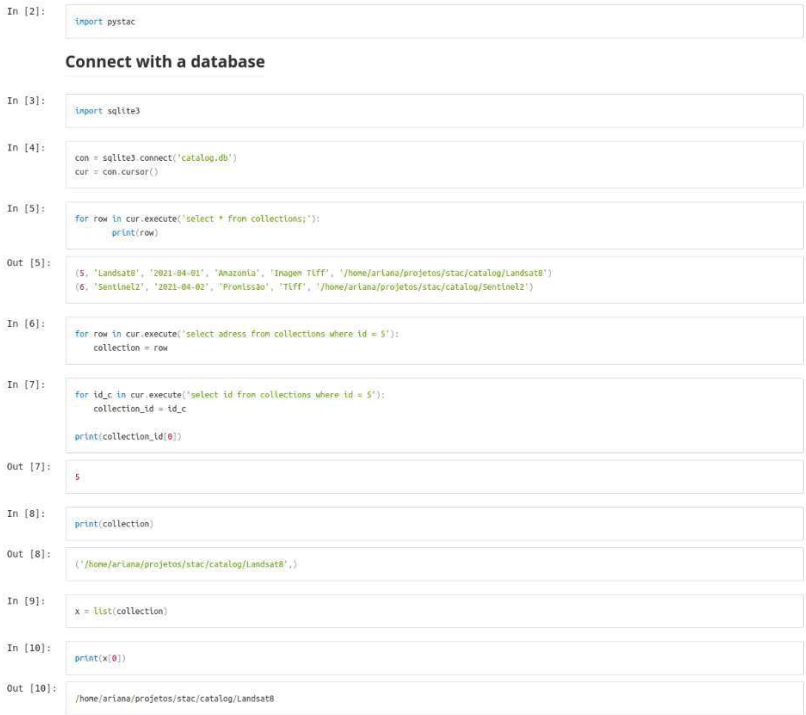
Figura 2.11 - Listagem de Coleções Criadas na plataforma



Fonte: Autora

Após a configuração do banco de dados, conexão do banco com as API's do BDC e as imagens cadastradas na aplicação acima a catalogação das coleções pode ser persistida no banco de dados propriamente dita. Para esta etapa, foi utilizado o Jupyter Notebook, a API BDC-STAC e as bibliotecas pystac e squilite3. A Figura 2.12 mostra como conectar com a base de dados da aplicação de armazenamento do endereço das imagens.

Figura 2.12 - Conexão com a base de dados



Fonte: Autora

Com o apoio da biblioteca rasterio é possível criar o bounding box das imagens para que essa informação seja adicionada nas informações dos itens. A Figura 2.13, mostra as etapas necessárias para verificar a coleção escolhida para a criação do Catálogo, a criação de uma função para extrair as informações de bounding box, bem como a conexão com a API do BDC.

Figura 2.13 - Criação do Catálogo a partir do endereço na base de dados

Creating a catalog from a database

In [11]:
```
import os
imgs = os.listdir(x[0])
```

In [12]:
```
print(imgs)
```

Out [12]:
```
['L1C_T21MXT_A002138_20170803T141045.tif', 'L1C_T21MXT_A006714_20180619T141044.tif', 'L1C_T21MXT_A007114_20161101T141043.tif', 'L1C_T21MXT_A009574_20190105T1
```

In [13]:
```
id_data = dict([img.replace(".tif", ""), {"img": 'file:{}/{}'.format(x[0],img) }] for img in imgs)
id_data = dict(list(id_data.items())[:-1])
id_data
```

Out [13]:
```
{'L1C_T21MXT_A002138_20170803T141045': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A002138_20170803T141045.tif'},
 'L1C_T21MXT_A006714_20180619T141044': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A006714_20180619T141044.tif'},
 'L1C_T21MXT_A007114_20161101T141043': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A007114_20161101T141043.tif'},
 'L1C_T21MXT_A009574_20190105T141044': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A009574_20190105T141044.tif'},
 'L1C_T21MXT_A000822_20150819T141044': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A000822_20150819T141044.tif'},
 'L1C_T21MXT_A004540_20160505T141035': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A004540_20160505T141035.tif'},
 'L1C_T21MXT_A007000_20180709T141044': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A007000_20180709T141044.tif'},
 'L1C_T21MXT_A010003_20190204T141046': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A010003_20190204T141046.tif'},
 'L1C_T21MXT_A002109_20151117T140930': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A002109_20151117T140930.tif'},
 'L1C_T21MXT_A009431_20181226T141042': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A009431_20181226T141042.tif'},
 'L1C_T21MXT_A010575_20190316T141218': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A010575_20190316T141218.tif'},
 'L1C_T21MXT_A009974_20170520T141049': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A009974_20170520T141049.tif'},
 'L1C_T21MXT_A007286_20180729T141043': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A007286_20180729T141043.tif'},
 'L1C_T21MXT_A001852_20170714T141046': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A001852_20170714T141046.tif'},
 'L1C_T21MXT_A006428_20180530T141042': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A006428_20180530T141042.tif'},
 'L1C_T21MXT_A004140_20171221T141033': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A004140_20171221T141033.tif'},
 'L1C_T21MXT_A010260_20170609T141047': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A010260_20170609T141047.tif'},
 'L1C_T21MXT_A004397_20160425T141030': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A004397_20160425T141030.tif'},
 'L1C_T21MXT_A002252_20151127T141059': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A002252_20151127T141059.tif'},
 'L1C_T21MXT_A003568_20171111T141032': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A003568_20171111T141032.tif'},
 'L1C_T21MXT_A001108_20150908T141045': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A001108_20150908T141045.tif'},
 'L1C_T21MXT_A002424_20170823T141042': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A002424_20170823T141042.tif'},
 'L1C_T21MXT_A003282_20171022T141032': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A003282_20171022T141032.tif'},
 'L1C_T21MXT_A006857_20180629T141045': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A006857_20180629T141045.tif'},
 'L1C_T21MXT_A007143_20180719T141044': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A007143_20180719T141044.tif'},
 'L1C_T21MXT_A006571_20180609T141044': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A006571_20180609T141044.tif'},
 'L1C_T21MXT_A005713_20180410T141043': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A005713_20180410T141043.tif'},
 'L1C_T21MXT_A002681_20151227T141143': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A002681_20151227T141143.tif'},
 'L1C_T21MXT_A002710_20170912T141037': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A002710_20170912T141037.tif'},
 'L1C_T21MXT_A004426_20180110T141037': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A004426_20180110T141037.tif'},
 'L1C_T21MXT_A006542_20160922T141041': {'img': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A006542_20160922T141041.tif'}}
```

In [14]:
```
import rasterio
from shapely.geometry import Polygon, mapping

def get_bbox_and_footprint(raster_uri):
    with rasterio.open(raster_uri) as ds:
        bounds = ds.bounds
        bbox = [bounds.left, bounds.bottom, bounds.right, bounds.top]
        footprint = Polygon([
            [bounds.left, bounds.bottom],
            [bounds.left, bounds.top],
            [bounds.right, bounds.top],
            [bounds.right, bounds.bottom]
        ])

        return (bbox, mapping(footprint))
```

In [15]:
```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

from bdc_catalog import BDCCatalog
import bdc_catalog.models as bdc_models
```

In [16]:
```
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://postgres:postgres@localhost:5433/bdc'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

Fonte: Autora

A Figura 2.14 mostra como extrair as informações das imagens. Da linha 1 a 14 são criadas variáveis com as informações para serem persistidas no banco de dados referentes ao id da coleção e o tipo de produto a ser catalogado. Da linha 17 a 72 é possível visualizar a função que extrairá as informações a serem persistidas no banco.

Figura 2.14 - Função para extrair as informações das imagens a serem catalogadas

In [41]:

```
ASSET_TYPES = {
    '.tif': 'application/tif',
    '.png': 'image/png'
}

ASSET_NAME = {
     '.tif': 'asset',
     '.png': 'thumbnail'
}

COLLECTION_ID = 5
PROVIDER_ID = 1
APPLICATION_ID = 1
SRID = 4326


def extract(assets):
    iten = {}
    for id_ in id_data:
        img_url = id_data[id_]['img']
        #print('Processing {}'.format(img_url))
        bbox, footprint = get_bbox_and_footprint(img_url)

        infos = {}

        asset_id = id_
        assetinfos = asset_id.split('_')
        name = assetinfos[0], assetinfos[1], assetinfos[2]

        infos['name'] = '_'.join(name)
        infos['tile_id'] = None

        _date = assetinfos[3][0:8]

        infos['start_date'] = _date
        infos['end_date'] = _date
        infos['cloud_cover'] = None
        infos['metadata'] = {}
        infos['geom'] = footprint
        infos['min_convex_hull'] = footprint

        provider_id = 1
        application_id = 1
        SRID = 4326

        # Assets
        assetlist = {}
        for _asset in assets:
            for i in ASSET_NAME:
                assetname = ASSET_NAME[i] if ASSET_NAME[i] is not None else 1
                #if i in _asset:
                    #assetname = ASSET_NAME[i]

            assetlist[assetname] = {}
            assetlist[assetname]['href'] = img_url

            for i in ASSET_TYPES:
                if i in _asset:
                    assetlist[assetname]['type'] = ASSET_TYPES[i]

        infos['assets'] = assetlist

        infos['collection_id'] = collection_id[0]
        infos['provider_id'] = provider_id #verificar com o Menino
        infos['application_id'] = application_id #verificar com o Menino
        infos['srid'] = SRID
        #infos['created'] = None
        #infos['updated'] = None

        iten[id_] = infos

    return iten
```

Fonte: Autora

Figura 2.15 mostra a variável general_infos foi criada para chamar a função extract utilizando o argumento id_data (In 13 da Figura 2.13).

Figura 2.15 - Variável criada para chamar a função extract com as informações das imagens

In [42]:

```
general_infos = extract(id_data)
```

Fonte: Autora

Ao imprimir a variável general_infos é possível visualizar todos os dezessetes itens presentes no repositório, a Figura 2.16 mostra apenas os três primeiros itens.

Figura 2.16 - Visualização dos três primeiros itens presentes na coleção.

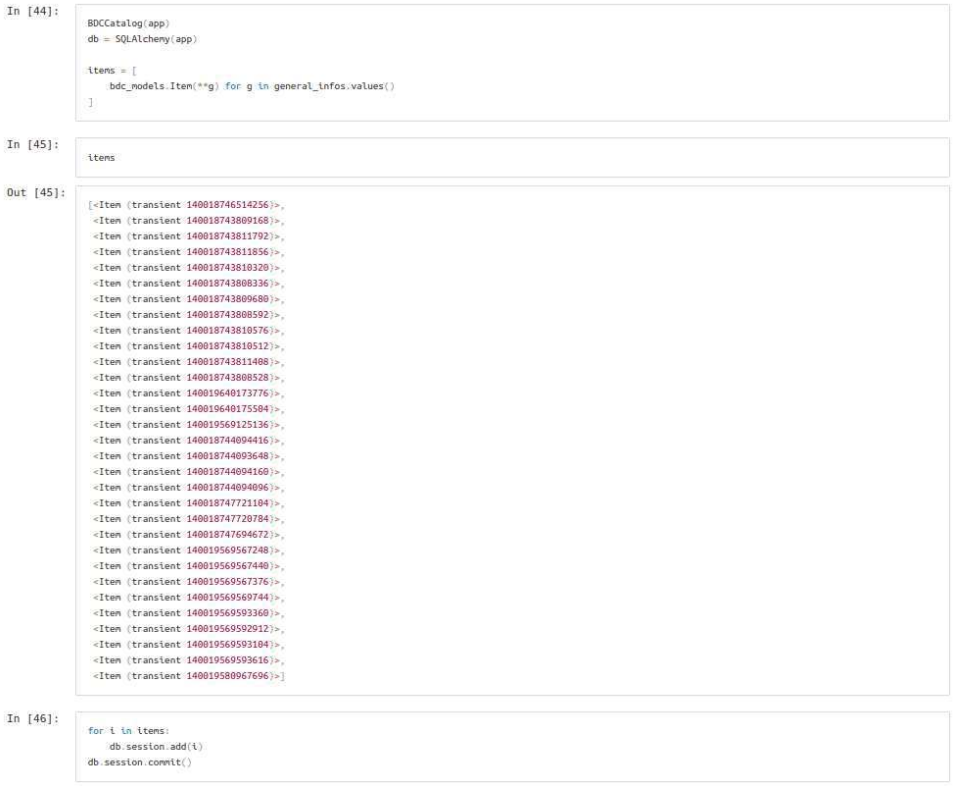
```
In [43]:
general_infos

Out [43]:
{'L1C_T21MXT_A002138_20170803T141045': {'name': 'L1C_T21MXT_A002138',
  'tile_id': None,
  'start_date': '20170803',
  'end_date': '20170803',
  'cloud_cover': None,
  'metadata': {},
  'geom': {'type': 'Polygon',
   'coordinates': (((600000.0, 9744940.0),
     (600000.0, 9800020.0),
     (709800.0, 9800020.0),
     (709800.0, 9744940.0),
     (600000.0, 9744940.0)),)},
  'min_convex_hull': {'type': 'Polygon',
   'coordinates': (((600000.0, 9744940.0),
     (600000.0, 9800020.0),
     (709800.0, 9800020.0),
     (709800.0, 9744940.0),
     (600000.0, 9744940.0)),)},
  'assets': {'thumbnail': {'href': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A002138_20170803T141045.tif'}},
  'collection_id': 5,
  'provider_id': 1,
  'application_id': 1,
  'srid': 4326},
 'L1C_T21MXT_A006714_20180619T141044': {'name': 'L1C_T21MXT_A006714',
  'tile_id': None,
  'start_date': '20180619',
  'end_date': '20180619',
  'cloud_cover': None,
  'metadata': {},
  'geom': {'type': 'Polygon',
   'coordinates': (((600000.0, 9744940.0),
     (600000.0, 9800020.0),
     (709800.0, 9800020.0),
     (709800.0, 9744940.0),
     (600000.0, 9744940.0)),)},
  'min_convex_hull': {'type': 'Polygon',
   'coordinates': (((600000.0, 9744940.0),
     (600000.0, 9800020.0),
     (709800.0, 9800020.0),
     (709800.0, 9744940.0),
     (600000.0, 9744940.0)),)},
  'assets': {'thumbnail': {'href': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A006714_20180619T141044.tif'}},
  'collection_id': 5,
  'provider_id': 1,
  'application_id': 1,
  'srid': 4326},
 'L1C_T21MXT_A007114_20161101T141043': {'name': 'L1C_T21MXT_A007114',
  'tile_id': None,
  'start_date': '20161101',
  'end_date': '20161101',
  'cloud_cover': None,
  'metadata': {},
  'geom': {'type': 'Polygon',
   'coordinates': (((600000.0, 9744940.0),
     (600000.0, 9800020.0),
     (709800.0, 9800020.0),
     (709800.0, 9744940.0),
     (600000.0, 9744940.0)),)},
  'min_convex_hull': {'type': 'Polygon',
   'coordinates': (((600000.0, 9744940.0),
     (600000.0, 9800020.0),
     (709800.0, 9800020.0),
     (709800.0, 9744940.0),
     (600000.0, 9744940.0)),)},
  'assets': {'thumbnail': {'href': 'file:/home/ariana/projetos/stac/catalog/Landsat8/L1C_T21MXT_A007114_20161101T141043.tif'}},
  'collection_id': 5,
  'provider_id': 1,
  'application_id': 1,
  'srid': 4326},
 'L1C_T21MXT_A009574_20190105T141044': {'name': 'L1C_T21MXT_A009574',
  'tile_id': None,
```

Fonte: Autora

Conexão com a API do BDC para persistir os dados no banco de dados, impressão das dos itens da coleção, a persistência e a declaração 'commit' no banco de dados (para atualizar e efetivar os dados), Figura 2.17.

Figura 2.17 - Persistência das informações dos itens no catálogo



Fonte: Autora

3 - Conclusão

Como descrito nos ciclos de trabalho, após a familiarização com os conceitos básicos, as estruturas de dados do laboratório, e dos satélites, e a estruturação do projeto iniciou-se a elaboração da aplicação web para que os usuários visualizassem os produtos desenvolvidos pelo laboratório.

Os frameworks escolhidos utilizaram apenas a linguagem de programação python, o que facilitou a realização da catalogação e também a aplicação web.

Referências Bibliográficas

BDC-CATALOG. GITHUB BDC-CATALOG, c2019. Disponível em: < https://github.com/brazil-data-cube/bdc-catalog >. Acesso em: 20 de maio de 2021.

BDC-DB. GITHUB BDC-DB, c2020. Disponível em: < https://github.com/brazil-data-cube/bdc-db >. Acesso em: 20 de maio de 2021.

BDC-STAC. GITHUB BDC-STAC, c2019. Disponível em: < https://github.com/brazil-data-cube/bdc-stac >. Acesso em: 20 de maio de 2021.

MDILLON. DOCKER HUB POSTGIS, c2021. Disponível em:< https://hub.docker.com/r/mdillon/postgis >. Acesso em 19 de maio de 2021.

STAC Specification. **SpatioTemporal Asset Catalogs**, **[S.l.].** Página inicial. Disponível em: < https://stacspec.org/ >. Acesso em: 19 de maio de 2021.