IfcOpenShell-Based BIM Application

Overview

This project uses the IfcOpenShell library to process Building Information Model (BIM) files in the IFC format. The application reads an IFC file, extracts relevant data about building elements, their properties, relationships and performs simple analysis and saves the processed data to two separate JSON files. This project is containerized using Docker to ensure a consistent runtime environment.

Approach

1. IfcOpenShell Library: Utilized to read and process IFC files.
2. Data Extraction: Extracted building elements, their properties and relationships from the IFC file using python scripts.
3. Analysis: Performed a simple analysis.
4. Data Output: Saved the processed data and analysis results to a JSON file.
5. Docker: Created a Dockerfile to set up the environment, install dependencies, and run the application in the container.

Pre-requisites

1. Basic domain knowledge of Building Information Modelling or BIM.
2. The IFC file provided is well-formed and follows standard IFC specifications.
3. The Docker environment has network access to fetch and install these dependencies during the build process.
4. The Docker runtime environment of minimum 16 GB memory and 8 cores CPU to handle the processing of the IFC file, especially if the file is large.

Instructions to build and run the application using Docker

1. Install Docker

1.1.Update your System:

- Update your existing list of packages.

```
sudo yum update -y
```

1.2.Install Docker

- Install Docker using the yum package manager.

```
sudo yum install docker -y
```

1.3. Verify Docker Installation:

```
docker --version
```

- The current docker version will be displayed on screen.

2. Create a Dockerfile

   A Dockerfile is a text-based file with a series of commands to create an image.

2.1. Create a directory for your project:

```
mkdir my-docker-project
cd my-docker-project
```

2.2. Create a Dockerfile:

```
touch Dockerfile
```

2.3. Edit the Dockerfile

- Open the Dockerfile in your preferred text editor and add the necessary commands. Here’s an example Dockerfile for a simple Python based application:
- Dockerfile:

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim

# Set the working directory in the container
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install IfcOpenShell and other necessary dependencies
RUN apt-get update && apt-get install -y \
    libboost-all-dev \
    cmake \
    build-essential \
    python3-dev \
    libpython3-dev \
    libxml2-dev \
    libgeos-dev \
    libgdal-dev \
    && rm -rf /var/lib/apt/lists/*

# Install Python dependencies
RUN pip install --upgrade pip
# RUN pip install --no-cache-dir -r requirements.txt

# copy ifcopenshell dir to python path - site packages
RUN cp -r /app/ifcopenshell /usr/local/lib/python3.9/site-packages/.

# install lark dependency
RUN pip install lark-parser

# Define environment variable
ENV NAME IfcOpenShellApp

# Run the application
CMD ["python", "process_ifc.py"]
```

2.4.Build the Docker Image.

- Use the docker build command to create a Docker image from your Dockerfile.

  ```
  docker build -t my-python-app .
  ```

- '-t my-node-app': Tags the image with a name (my-python-app).
- ' . ' Specifies the current directory as the build context.

2.5.Verify the image was created

```
docker images
```

- You should see your my-python-app image listed.

3. Run the created Docker image

3.1.Create docker volume to map container volume

- Create a docker volume on the host machine to map the container volume for persistent data retrieval.

  docker volume create container_volume

- Docker volume 'container_volume' will be created on the host machine.

3.2. Run the container:

  docker run -d -v container_volume:/app my-python-app

- '-d' : Runs the container in detached mode.
- '-v' : Maps the volume 'container_volume' to the '/app' directory volume on container
- 'my-python-app' : Name of the image to run

3.3. Verify the container is running:

  docker ps

- You should see your my-python-app container listed.

3.4. Access the created volume on host machine

- Access the created 'container_volume' on host machine to retrieve the output.

  cd /var/lib/docker/volumes/container_volume/_data/

- '_data' stores the output by default.

Challenges

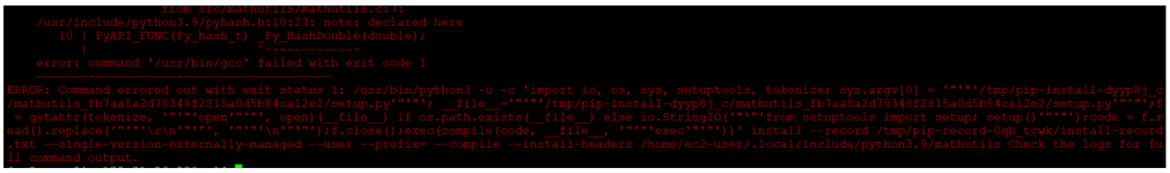
1. Domain knowledge acquisition for proper implementation of the requirements:

   IFC files are complex and contain a wealth of information about building structures, including geometry, spatial relationships, and properties of building elements. To work effectively with IFC files, you need to have a basic understanding of both the technical aspects of the IFC format and the domain-specific knowledge related to building construction and architecture.

   Solution:

Utilize online tutorials, guides, and example projects to see how others have approached similar challenges. Websites like BIMForum, buildingSMART, and YouTube channels dedicated to BIM maybe utilized.

2. Ifcopenshell dependency installation: encountered "'/user/bin/gcc' failure with exit status 1" error while installing ifcopenshell using pip.



```
                     from src/mathutils/mathutils.c:7:
    /usr/include/python3.9/pyhash.h:10:23: note: declared here
       10 | PyAPI_FUNC(Py_hash_t) _Py_HashDouble(double);
          |                       ^~~~~~~~~~~~~~
    error: command '/usr/bin/gcc' failed with exit code 1
    ----------------------------------------
ERROR: Command errored out with exit status 1: /usr/bin/python3 -u -c 'import io, os, sys, setuptools, tokenize; sys.argv[0] = '"'"'/tmp/pip-install-dyyp8j_c
/mathutils_fb7aa8a2d78348f2815a0d5b84ca12e2/setup.py'"'"'; __file__='"'"'/tmp/pip-install-dyyp8j_c/mathutils_fb7aa8a2d78348f2815a0d5b84ca12e2/setup.py'"'"';f
 = getattr(tokenize, '"'"'open'"'"', open)(__file__) if os.path.exists(__file__) else io.StringIO('"'"'from setuptools import setup; setup()'"'"');code = f.r
ead().replace('"'"'\r\n'"'"', '"'"'\n'"'"');f.close();exec(compile(code, __file__, '"'"'exec'"'"'))' install --record /tmp/pip-record-0qb_tcwk/install-record
.txt --single-version-externally-managed --user --prefix= --compile --install-headers /home/ec2-user/.local/include/python3.9/mathutils Check the logs for fu
ll command output.
```

Solution:

- Download the compatible pre-build package zipfile from: https://docs.ifcopenshell.org/ifcopenshell-python/installation.html#

- Unzip the downloaded file and copy the ifcopenshell directory into your Python path. To find the Python path, run the following code in Python:

  import sys
  print(sys.path)

  This will give a list of possible directories that can install the IfcOpenShell module into. Copy the ifcopenshell directory into one of these called site-packages.

- Test importing the module in a Python session or script to make sure it works.

  import ifcopenshell
  print(ifcopenshell.version)
  model = ifcopenshell.file()

3. Output data retrieval
The container created after running the image is destroyed after the execution of the python scripts within and hence the output data retrieval becomes a challenge.

Solution: Docker volume is explicitly created on the host machine and hence mapped to the container volume using the following command:

'docker volume create container_volume'   #To create volume on host
'docker run -d -v container_volume:/app image_ID'   #Volume mapping

The main reason for the creation of the volume is for persistent data of the output on the host machine i.e., since the output was getting destroyed along with the destruction of the container.