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 (total wall area and number of elements), 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. Update your System:

* Update your existing list of packages.

*sudo yum update -y*

* 1. Install Docker
* Install Docker using the yum package manager.

*sudo yum install docker -y*

* 1. Verify Docker Installation:

*docker --version*

* The current docker version will be displayed on screen.

1. Create a Dockerfile

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

* 1. Create a directory for your project:

*mkdir my-docker-project*

*cd my-docker-project*

* 1. Create a Dockerfile:

*touch Dockerfile*

* 1. 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"]*

* 1. 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.
  1. Verify the image was created

*docker images*

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

1. Run the created Docker image
   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.
  1. 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
  1. Verify the container is running:

*docker ps*

* You should see your my-python-app container listed.
  1. 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.

Building and running with Docker-compose

1. Install Docker Compose:

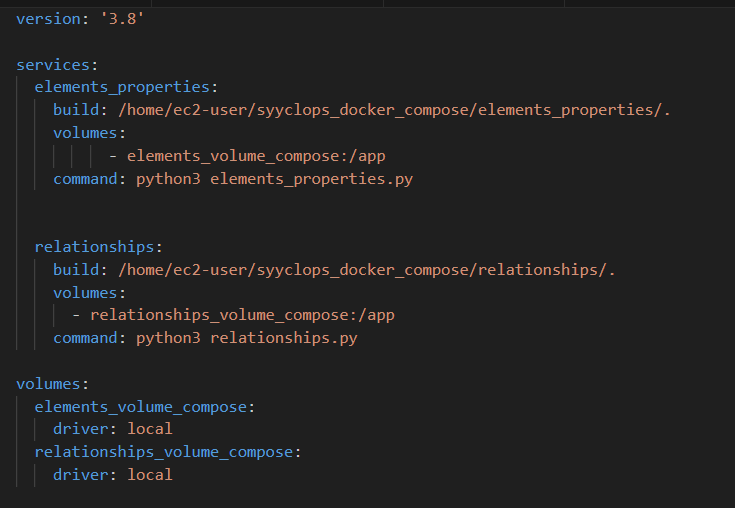
*sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose*

*sudo chmod +x /usr/local/bin/docker-compose*

*docker-compose –version*

1. Create a docker-compose.yml File

* Create a docker-compose.yml file in your project directory. This file defines the services, networks, and volumes for your application.
* Example:



* The build directive tells Docker Compose to build an image using the Dockerfile located in /home/ec2-user/syyclops\_docker\_compose/elements\_properties/.
* The volumes directive mounts the named volume elements\_volume\_compose to the /app directory inside the container.
* The command directive runs python3 elements\_properties.py when the container starts.

1. Build and Run the Application

*docker-compose build*

1. Run the Services:

*docker-compose up -d*

1. Verify the Services:

* Check the running containers:

*docker-compose ps*

1. Access and Manage Volumes:

* The data from your application will be stored in the Docker volume for persistent data retrieval.
* To access the data on the host machine:

*cd /var/lib/docker/volumes*

Challenges

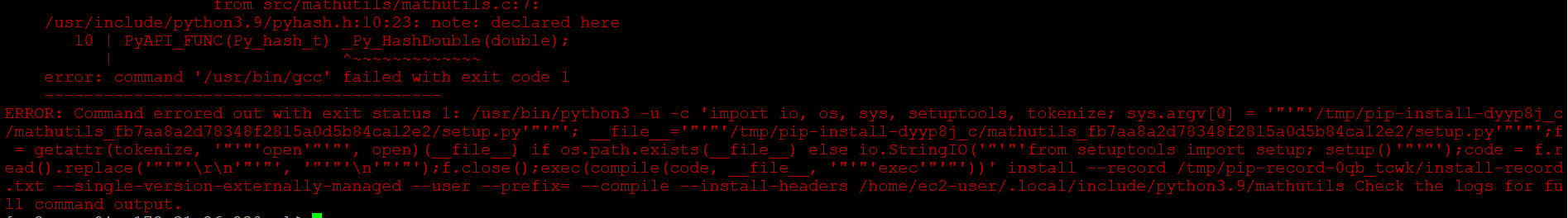
1. Domain knowledge acquisition for proper implementation of the requirements:

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.

1. Ifcopenshell dependency installation: encountered “‘/user/bin/gcc’ failure with exit status 1“ error while installing ifcopenshell using pip.



Solution:

* Download the compatible pre-build package zipfile from:

[https://docs.ifcopenshell.org/ifcopenshell-python/installation.html#](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()*

1. 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.