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 (Calculated total wall area and total number of elements present).
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

Do pointwise:

Install docker and version verification, how to write a dockerfile with prerequisites, generation of docker image from a dockerfile, how to create the volume for the container, how to create and run a container using the docker volume and image

1. How to install Docker

For Windows and MacOS:

* 1. Download Docker Desktop:
* Go to the [Docker Desktop download page](https://docs.docker.com/desktop/).
* Download the installer for your operating system (Windows or macOS).
  1. Run the installer:
* Open the downloaded installer and follow the on-screen instructions to install Docker Desktop.
  1. Start Docker Desktop:
* Once the installation is complete, start Docker Desktop from the Start menu (Windows) or the Applications folder (macOS).
  1. Verify the Installation:
* Open a terminal (Command Prompt, PowerShell, or Terminal) and run the following command:
* Command:

``` sh

docker –version

* You should see the Docker version information, confirming that Docker is installed correctly.

For Linux:

* 1. Update your System:
* Start by updating your existing list of packages.
* Command:

``` sh

sudo apt-get update

* 1. Install Necessary Packages:
* Install a few prerequisite packages which let apt use packages over HTTPS.
* Command:

``` sh

sudo apt-get install apt-transport-https ca-certificates curl software-properties-common

* 1. Add Docker’s Official GPG Key:
* Add the GPG key for Docker’s official repository to your system.
* Command:

``` sh

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –

* 1. Add Docker’s Official APT Repository:
* Command:

``` sh

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

* 1. Update Your Package Database Again:
* Update your package database with the Docker packages from the newly added repository.
* Command:

``` sh

sudo apt-get update

* 1. Install Docker CE (Community Edition):
* Command:

``` sh

sudo apt-get install docker-ce

* 1. Verify Docker Installation:
* Command:

``` sh

docker –version

* The current docker version will be displayed on screen.

1. How to Create a Dockerfile (FOR LINUX ONLY????)

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

* 1. Create a directory for your project:
* Command:

``` sh

mkdir my-docker-project

cd my-docker-project

* 1. Create a Dockerfile:
* Command:

``` sh

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"]

CMD ["sleep", "500"]

* 1. Build the Docker Image.
* Use the docker build command to create a Docker image from your Dockerfile.
* Command:

``` sh

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
* Command:

``` sh

docker images

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

1. How to Run a Dockerfile

Use the docker run command to create and start a container from your 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.
* Command:

``` sh

docker volume create container\_volume

* Docker volume ‘container\_volume’ will be created on the host machine.
  1. Run the container:
* Command:

``` sh

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:
* Command:

``` sh

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.
* Command:

``` sh

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.

1. Dependency Management: Ensuring all necessary dependencies and pre-built packages are included and compatible.

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. If you’re not sure where your Python path is, run the following code in Python:

``` sh

import sys

print(sys.path)

```

This will give you a list of possible directories that you can install the IfcOpenShell module into. Most commonly, you will want to 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.

``` sh

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.