**Azure Batch Application for TOPAS MC Simulations Documentation**

**Table of Contents**

1. Introduction
2. Prerequisites
3. Application Architecture
4. Setup and Configuration
5. Running Simulations
6. Monitoring and Management
7. Appendixes
8. **Introduction**

**1.1 Purpose**

#### Why Cloud over Traditional Clusters?

Access to high-performance computing clusters can be a limiting factor for many research institutions and individual researchers. These limitations arise from various reasons:

* **High Initial Investment**: Establishing a private cluster demands significant capital for hardware, infrastructure, and software licenses.
* **Maintenance Overheads**: A dedicated IT team is often needed for maintenance, updates, and troubleshooting, resulting in recurring costs.
* **Scalability Issues**: Traditional clusters may not scale seamlessly. Expanding computational capabilities can be cumbersome, time-consuming, and expensive.
* **Underutilization**: After the initial setup, these resources might be underutilized during off-peak times, leading to wasted resources and costs.
* **Accessibility and Mobility**: Accessing private clusters may require VPNs or other remote access methods, which can be less than ideal for researchers on the move or those collaborating from different geographical locations.

With the above challenges in mind, the solution presented in this document leverages Azure Batch to provide an alternative to private clusters. By transitioning to a cloud-based solution, researchers/institutions benefit from:

* **Pay-as-you-go Model**: Only pay for the computational resources used, optimizing cost.
* **Auto-scaling**: Easily scale resources up or down based on the demand, ensuring optimal resource utilization.
* **Maintenance-Free**: Azure handles maintenance, updates, and security, letting researchers focus on their work.
* **Global Accessibility**: Access the cloud from anywhere, facilitating seamless collaborations and remote working.
* **Integration with Modern Technologies**: Easy integration with technologies like Docker ensures consistent and reproducible simulation environments.

By offering this Azure Batch application for TOPAS MC simulations, our goal is to empower both researchers and institutions, providing them with an efficient, scalable, and cost-effective solution, bypassing the traditional barriers associated with private clusters.

**1.2 Key Features**

This application has been developed with customizability as its cornerstone. Although initially designed to run TOPAS MC simulations, its architecture is flexible enough to handle any program that can be distributed across computing nodes.

* **Flexible VM Configuration**: Users have the freedom to specify the size and characteristics of the virtual machine pool, tailoring it perfectly to the demands of their specific simulations. This ensures efficient resource utilization and cost-effectiveness.
* **Custom Docker Integration**: Users can supply their personalized Docker images pre-configured with their specific version of TOPAS, desired extensions, Python code, and other required tools. This ensures consistent, reproducible, and customized simulation environments.
* **GitHub Integration**: Enhance the workflow by incorporating custom Python scripts for pre-processing and post-processing directly from the user's GitHub repositories. This feature ensures that users can maintain and update their codes seamlessly, benefiting from version control and collaboration features provided by GitHub.

**2. Prerequisites**

Before diving into the application setup and usage, there are a few foundational elements that users need to be familiar with. While a basic understanding of Azure Batch, Docker, and GitHub is recommended, this guide aims to be comprehensive:

* Azure Subscription Creation: This guide will walk you through the step-by-step process of creating an Azure subscription, followed by setting up the necessary Azure Batch and Storage accounts.
* Docker Environment Configuration: To run your custom simulations, you'll need a Docker image of your desired environment. If you're unsure about how to create or customize Docker images, this guide will provide comprehensive instructions to help you create Docker images tailored to your needs.

**3. Application Architecture**

### **3.1 Overview**

The application architecture (Figure 1) revolves around Azure Batch to distribute and manage the execution of TOPAS MC simulations. Here's an in-depth look:

1. **Local simulation directory**: This houses the TOPAS simulation parameter file and any other requisite support files for the intended simulation.
2. **Azure Storage**: This cloud storage solution acts as an intermediary, storing simulation files and results. **batch\_simulation\_runner.py** uploads the contents of the user's local directory to Azure Storage. After simulation completion, the results are uploaded here and can be retrieved to the user's local computer.
3. **Azure Batch's Pool of VMs**: Once the simulation files are in Azure Storage, the **batch\_simulation\_runner.py** script initiates the creation of a pool of virtual machines (VMs) in Azure Batch. Each VM in this pool is designed to:
   1. Download the simulation files from Azure Storage.
   2. Run the simulation within a user-provided **Docker container** that contains the necessary environment.
   3. Upload the resultant output back to Azure Storage.
4. **Data Reduction**: If data post-processing is needed, the **batch\_data\_reducer.py** script can be utilized. This script operates similarly to **batch\_simulation\_runner.py**, but its primary function is to process, merge, or reduce simulation results. This operation happens on a single VM that uses a Docker image configured for running Python code. The Python reduction tasks/scripts are downloaded from the user-specified GitHub repository.

A diagram of a cloud storage

Description automatically generated

**Figure 1**: Architectural Overview of the Cloud-based TOPAS MC Simulation Framework. This diagram illustrates the integration of Azure Batch services, Docker containers, and GitHub repositories to provide a flexible and scalable solution for running and managing simulations.

#### **3.2 Azure Batch**

Azure Batch is a cloud-based parallel and high-performance computing solution offered by Microsoft Azure. It's designed to manage and execute applications across many computers simultaneously—referred to as 'compute nodes'. The Azure Batch service provides scalable compute resources as a service, without the overhead of managing infrastructure, making it ideal for parallel compute workloads.

#### Key Concepts:

* **Pools**: A pool is a collection of compute nodes (virtual machines). Users can define the number and size of these nodes based on the computational needs. In our application, the size and characteristics of the virtual machines pool can be tailored according to specific simulation requirements.
* **Jobs**: Jobs are collections of tasks. A job is essentially the "what" you want to run, like a project or a simulation. In the context of our application, running a simulation set might be considered as a single job.
* **Tasks**: Tasks are the individual units of work, or the "how", that the job breaks down into. Each task runs on a single node. In our application, a task might correspond to a single TOPAS MC simulation or a particular set of data being processed.

In our architecture:

* The **batch\_simulation\_runner** script manages the entire lifecycle of the simulation in Azure Batch. Once initiated, the script sets up the appropriate pools, uploads the necessary files from the local directory to Azure Storage, and creates the jobs and tasks to execute the simulations across the VMs. Each VM in the pool then processes its assigned tasks using Docker containers.
* Post-simulation, the **batch\_data\_reducer** script can be invoked. This script also leverages Azure Batch but often uses a single VM to consolidate and process the results generated by the individual tasks. This data reduction is achieved by executing Python scripts obtained from a designated GitHub repository.

The interaction between Azure Storage and the VMs in the Azure Batch pool is seamless. Once tasks are completed, results are stored back in Azure Storage, allowing users to conveniently download them to their local computers.

#### **3.3 Docker Containers**

Docker is a platform used to develop, ship, and run applications inside containers. Containers are lightweight, standalone, and executable software packages that contain everything needed to run a piece of software, including the code, runtime, system tools, libraries, and settings. Containers are isolated from each other and the host system, ensuring consistency across different environments. This technology allows developers to package an application with all its dependencies and configurations, ensuring it will run identically everywhere, from a developer's local machine to different cloud environments.

In the context of our application:

Docker aids in encapsulating the TOPAS MC simulation environment, ensuring consistency, reproducibility, and isolation. Here's how Docker is woven into the application:

* **Environment Isolation**: Docker containers wrap up the software, libraries, and environments into a standardized unit. This ensures that TOPAS MC simulations run in the same environment, regardless of where the Docker container is deployed, thus avoiding the infamous "it works on my machine" problem.
* **User-defined Images**: Rather than constraining users to a specific setup, they can provide their custom Docker images. This image should be set up with their preferred version of TOPAS, extensions, and any other necessary software or configurations.
* **Execution**: When a VM in the Azure Batch pool is ready to run a simulation, it pulls the user-defined Docker image, ensuring that the simulation operates within the specified environment. Once completed, the output is outside the container, allowing Azure Batch to upload it to Azure Storage.

#### **3.4 GitHub Integration**

While the integration of GitHub isn't a core component, it serves as an advantageous supplement. Instead of sourcing Python scripts for data reduction from the local directory, they are fetched directly from a user-specified GitHub repository. This approach promotes:

* **Version Control**: By using a GitHub repository, users can leverage the versioning capabilities of Git, ensuring they always have a history of changes and can roll back if necessary.
* **Collaboration**: GitHub allows multiple users to collaborate on the same scripts, making it simpler for teams to develop and improve reduction scripts over time.

With this architectural layout, the application ensures flexibility, scalability, and efficiency in running and managing TOPAS MC simulations on the Azure platform.

**4. Setup and Configuration**

### **4.1 Azure Batch Setup**

Before leveraging Azure Batch for running simulations, it's essential to set up the required Azure resources. In this section, the process of creating an Azure subscription will be outlined, followed by instructions on setting up an Azure Batch account and Azure Batch storage.

**4.1.1 Creating an Azure Subscription**

1. **Navigate to Azure Portal:** Go to the [Azure Portal](https://portal.azure.com/).
2. **Sign in or Create an Account:** If you have an existing Microsoft account, sign in. If not, you'll need to create one. Follow the on-screen instructions.
3. **Start the Subscription:** Once logged in, search for 'Subscriptions' in the Azure services search bar. Click on the **+ Add** button to create a new subscription. Choose your desired subscription plan and follow the prompts.

**4.1.2 Setting up Azure Batch Account**

1. **Access Azure Batch Services:** From the Azure Portal dashboard, click on 'Create a resource'. Then, search for 'Batch Service' and select it.
2. **Create New:** Click on the **+ Create** button.
3. **Configure the Batch Account:**
   * **Subscription:** Choose the subscription you created earlier.
   * **Resource Group:** Create a new resource group or use an existing one. This will help you manage related Azure resources.
   * **Account Name:** Assign a unique name for your Azure Batch account.
   * **Location:** Choose a region that's close to your users or data sources.
4. **Review + create:** After filling in the necessary details, review your settings and click on the **Create** button.

⚠️ **Important**: After setting up your Azure Batch account, you might find the default number of VMs available for your pool to be insufficient for your needs. In such cases, you can request a quota increase. However, be aware that this process can sometimes be complex, as Microsoft may require users to undergo a credit check verification process. Ensure you have all necessary documentation and understand the implications before requesting a quota increase.

**4.1.3 Setting up Azure Batch Storage**

For efficient storage and retrieval of job data in Azure Batch, you need to set up a linked Azure Storage account.

1. **Navigate to the Azure Batch account:** From the Azure Portal, access your Azure Batch account that you created in the previous step.
2. **Access 'Storage account' section:** In the left-hand pane of your Batch account page, find and click on the 'Storage account' option.
3. **Linking Storage:**
   * If you already have an Azure Storage account you'd like to use, you can select it here.
   * If you need to create a new Azure Storage account, click on the **+ Create** button. Follow the on-screen instructions, ensuring you select 'Blob storage' as the account kind.
4. **Finalize the Setup:** After setting up or selecting your storage account, save any changes. Ensure that the storage account is correctly linked to your Azure Batch account.

**Note:** Ensure to monitor your resource consumption in the Azure Portal to avoid unexpected charges. Azure provides cost management tools that help you monitor and control your spending.

### **4.1 Docker configuration**

This section provides a step-by-step guide on setting up a Docker environment tailored for the application's needs:

1. **Install Docker**:
   * If Docker isn't already installed, download and install Docker Desktop from [Docker's official website](https://www.docker.com/products/docker-desktop).
2. **Verify Docker Installation**:
   * Open a terminal or command prompt.
   * Use the command **docker --version** to ensure Docker was installed. The Docker version should display in the terminal.
3. **Create Your Dockerfile**:
   * Navigate to the directory where you want to store your Dockerfile.
   * Using a text editor, create a file named **Dockerfile** (with no file extension).
   * Copy and paste the provided Dockerfile example into this file, making modifications as needed for your application.
4. **Build the Docker Image**:
   * Navigate to the directory containing your Dockerfile.
   * Use the command:

**docker build -t topas\_simulation\_image .**

This builds a Docker image named 'topas\_simulation\_image' based on the Dockerfile. Ensure no errors occur during this build.

1. **Push Docker Image to Docker Hub**:
   * If you don't have a Docker Hub account, create one at [Docker Hub's official website](https://hub.docker.com/).
   * Log into Docker Hub via terminal using **docker login**.
   * Tag your image for the push. If your Docker Hub username is 'username', the command is:

**docker tag topas\_simulation\_image username/topas\_simulation\_image:latest**

* + Push the image with: **docker push username/topas\_simulation\_image:latest**.

1. **Ensure Image Accessibility**:
   * The Docker image should be 'public' on Docker Hub so Azure Batch service can access it. Adjust the visibility settings on your Docker Hub repository accordingly.
2. **Test Docker Image Locally** (Optional, but recommended):
   * Before Azure deployment, consider testing the Docker image locally. This ensures the TOPAS MC simulations work correctly.
   * Use the command: **docker run -it topas\_simulation\_image:latest** to begin a container from your image. Run a brief test simulation within the container for verification.

**4.2.1 Dockerfile**

A Dockerfile is a script made up of commands that Docker uses to automatically build (or construct) a Docker image. When you run a **docker build** command, Docker reads the Dockerfile, executes the commands in the order they are written, and then returns a Docker image as the output. Each instruction in the Dockerfile creates a new layer in the image, which can be seen as a series of changes to the filesystem. This layering approach makes Docker images highly efficient and portable.

Common Dockerfile Commands:

1. **FROM**: This command specifies the base image. Every Docker image starts from a base image, whether it's a base Ubuntu, Alpine, or any other distribution.
   * Example: **FROM ubuntu:20.04**
2. **WORKDIR**: Changes the working directory of the container, any subsequent command that uses a relative path will be executed from this directory.
   * Example: **WORKDIR /app**
3. **RUN**: Executes commands during the image build process. Useful for installing software packages and other setup tasks.
   * Example: **RUN apt update && apt install -y python3**
4. **COPY**: Copies files or directories from the local machine into the Docker image.
   * Example: **COPY . .** (Copies all files from the current directory on your machine to the current location in the image.)
5. **ENV**: Sets environment variables inside the container.
   * Example: **ENV DEBIAN\_FRONTEND=noninteractive**
6. **CMD**: Specifies what command to run when a container starts from the image. Unlike RUN, this command does not execute during the build process, but when the container is instantiated.
   * Example: **CMD ["/bin/bash"]**

In Appendix A, we provide the Dockerfiles used in this solution to serve as a reference for users who wish to create their own images. Appendix A.I corresponds to the image containing the TOPAS installation, while the Dockerfile provided in Appendix A.II is simply an image that includes the latest official version of Python 3 as well as some Azure libraries necessary to access the storage.

**5. Running Simulations**

### **5.1 Prerequisites and setup**

Before initiating a TOPAS simulation, there are several preparatory steps and configurations the user must complete to ensure seamless integration and execution. The following outlines the necessary starting point for users aiming to run simulations using our application:

1. **Local Directory Setup:** Create a local directory on your system. This directory should encapsulate:
   * All files essential for the TOPAS simulation. This primarily includes parameter files, but may also encompass other supporting files tailored to your specific simulation requirements.
   * A shell script designed to invoke the TOPAS simulation. This script can also manage any pre-simulation tasks, ensuring the simulation runs with the intended settings and conditions.
2. **Docker Container Images:** Ensure you have pushed your container images to Docker Hub, one image for the TOPAS worker container and another for the reducer container. This images should have:
   * A properly installed version of TOPAS with all its dependencies.
   * Any specific TOPAS extensions required to execute your simulations.
   * Python installation and python packages necessary to run you reduce code (reducer container).
3. **Python Scripts and GitHub Repository:** Your specific Python scripts, designed to merge or reduce the simulation output files, must be housed in a GitHub repository. This repository should also feature:
   * A primary bash shell script. This script acts as the orchestrator, invoking the necessary Python scripts for data reduction or other operations.
4. **TOPAS Azure Batch Application Setup:**
   * Download the TOPAS Azure Batch application from the GitHub repository: <https://github.com/HectorMiras/topas_docker_azure_batch.git>.
   * Once downloaded to a local directory, you must populate the application's configuration files with the relevant details, ensuring alignment with your specific simulation needs and resources.

### **5.2 Configuration files**

**5.2.1 appconfig.json**

This file contains configuration settings related to Azure Batch, Docker images, and GitHub repository details. Here's a breakdown:

* **Azure Batch Details:**
  + **BATCH\_ACCOUNT\_NAME**: The name of your Azure Batch account.
  + **BATCH\_ACCOUNT\_KEY**: A secret key associated with the Azure Batch account.
  + **BATCH\_ACCOUNT\_URL**: The endpoint URL for your Azure Batch service.
* **Azure Storage Account Information:**
  + **STORAGE\_ACCOUNT\_NAME**: Name of the storage account linked to the Azure Batch account.
  + **STORAGE\_ACCOUNT\_KEY**: The secret key for accessing the storage account.
  + **STORAGE\_ACCOUNT\_DOMAIN**: Domain endpoint for accessing blobs in the storage account.
* **Docker Configuration:**
  + **WORKER\_DOCKER\_IMAGE**: Docker image used for running the TOPAS simulations.
  + **REDUCER\_DOCKER\_IMAGE**: Docker image used for data reduction (post-processing).
  + **DOCKER\_USER**: Docker username or DockerHub account name.
  + **DOCKER\_TOKEN**: Token or password to authenticate the Docker user.
* **GitHub Integration Details:**
  + **GIT\_USER**: GitHub username.
  + **GIT\_TOKEN**: GitHub token for API access.
  + **GIT\_REPO**: The name of the GitHub repository where Python scripts are stored.

**5.2.2 simconfig.json**

This file focuses on the specific details of a simulation run:

* **Simulation Identification:**
  + **SIM\_ID**: A unique identifier for the simulation.
* **Azure Batch Pool Configuration:**
  + **POOL\_NODE\_COUNT**: Number of nodes/compute VMs to be utilized for the simulation.
  + **POOL\_VM\_SIZE**: The size/type of virtual machine used for the nodes.
* **Simulation Local Path:**
  + **LOCAL\_SIM\_PATH**: Local path to the simulation files and data.
* **Scripting:**
  + **RUN\_SCRIPT**: Name of the main shell script that initiates the simulation.
  + **REDUCER\_SCRIPT**: Script that will be used for data reduction after the simulation concludes.
* **Output Details:**
  + **OUTPUT\_FILE\_PATTERNS**: Patterns or names of the files that will be generated as output from the simulation. These patterns help in collecting and organizing results post-simulation.

**5.3 Running a Simulation**

To successfully run a TOPAS MC simulation on the Azure platform, follow the steps outlined below:

1. **Configure Simulation Parameters:**
   * Modify the **simconfig.json** file:
     + Set a unique **SimulationID**.
     + Determine the desired number and size of computing nodes.
     + Define the local simulation directory.
     + Highlight the specific output patterns relevant to the simulation.
2. **Start the Simulation:**
   * Execute **batch\_simulation\_runner.py**
3. **Real-time Monitoring:**
   * As the simulation progresses, the script will monitor the creation and state of the pool and VMs, relaying relevant information to the console.
   * In addition to the provided script, it's recommended to monitor the simulation's progress using the Azure Portal for more detailed insights.
4. **Resource Management:**
   * Post-simulation, the script will pose "Yes/No" questions to determine which resources should be retained or removed, including options related to the pool, job, and storage. It's crucial to ensure resource optimization, especially if running multiple simulations or managing budgetary constraints.
5. **Data Reduction (Optional):**
   * If there's a need to condense or reduce the simulation output data from the worker pool VMs, initiate the **batch\_data\_reducer.py** script.
   * Upon executing this script, revisit steps 3 through 5 to monitor the reduction process, manage resources, and download the condensed outputs.
6. **Retrieving Simulation Outputs:**
   * Once the simulation is complete, the output files are saved in Azure Storage. If the user chooses "yes" in response to the "Download results?" prompt, the outputs will be saved to the local simulation directory. If not, there are various software tools available that facilitate easy access and management of Storage. Some recommended tools include:
     + Azure Storage Explorer
     + CyberDuck
     + CloudBerry Explorer

**6. Monitoring and Management**

Efficiently monitoring and managing the simulation process is crucial for timely results and optimizing resource usage. This section provides an overview of the monitoring and management features available within the Azure environment, especially concerning the TOPAS Azure Batch application.

**6.1 Azure Portal Monitoring**

Azure Portal is Microsoft's official web-based unified console that provides a built-in suite of monitoring tools. Users can:

* **View Pool and Node Status:** Real-time status of the computation pool and individual nodes can be checked. This includes metrics like CPU usage, memory consumption, and task status.
* **Remote access to computing nodes** Within Azure Portal, users can select a specific compute node under the "Pools" section and choose the "Connect to Computing Node" option. This feature facilitates an SSH connection to the node, granting users the ability to access logs, providing in-depth information about tasks executed on that node.
* **View Metrics:** Azure Portal offers graphical visualization of metrics, helping to diagnose issues, and ensuring optimal performance.

**6.2 Scripts Monitoring**

The TOPAS Azure Batch application provides scripts that output important monitoring information:

* **Batch scripts:** When executing **batch\_simulation\_runner.py** or **batch\_data\_reducer.py**, the scripts not only initiates the simulation but also keeps users updated about the pool's creation status, VM states, and other relevant events. Moreover, the scripts generate standard output files for each task, making it convenient to review the logs and understand task-specific operations and any potential issues.
* **Cleanup Prompt:** At the end of simulations, scripts ask users about resource cleanup, ensuring that users only keep necessary resources and avoid unnecessary costs.

**6.3 Recommendations**

1. **Regular Checks:** Especially for longer simulations, it's advisable to regularly check on the simulation's status, ensuring that there aren't any unexpected failures or stalled tasks.
2. **Log Reviews:** Periodically review logs to ensure the simulation's smooth operation and to catch any potential issues early.
3. **Cost Management:** Be proactive about stopping or deleting unnecessary resources. While Azure offers a robust set of tools for large-scale simulations, unused resources can lead to unexpected costs.

**Appendix A.I: Workers DockerFile**

# Use Debian 10 (Buster) as the base image

FROM ubuntu:20.04

# Set non-interactive mode for apt (to avoid prompts)

ENV DEBIAN\_FRONTEND=noninteractive

# 1) Install Data Files:

WORKDIR /G4Data

RUN apt update && apt install -y wget tar

RUN wget -4 https://geant4-data.web.cern.ch/geant4-data/datasets/G4NDL.4.6.tar.gz && \

wget -4 https://geant4-data.web.cern.ch/geant4-data/datasets/G4EMLOW.7.13.tar.gz && \

wget -4 https://geant4-data.web.cern.ch/geant4-data/datasets/G4PhotonEvaporation.5.7.tar.gz && \

wget -4 https://geant4-data.web.cern.ch/geant4-data/datasets/G4RadioactiveDecay.5.6.tar.gz && \

wget -4 https://geant4-data.web.cern.ch/geant4-data/datasets/G4PARTICLEXS.3.1.1.tar.gz && \

wget -4 https://geant4-data.web.cern.ch/geant4-data/datasets/G4SAIDDATA.2.0.tar.gz && \

wget -4 https://geant4-data.web.cern.ch/geant4-data/datasets/G4ABLA.3.1.tar.gz && \

wget -4 https://geant4-data.web.cern.ch/geant4-data/datasets/G4INCL.1.0.tar.gz && \

wget -4 https://geant4-data.web.cern.ch/geant4-data/datasets/G4PII.1.3.tar.gz && \

wget -4 https://geant4-data.web.cern.ch/geant4-data/datasets/G4ENSDFSTATE.2.3.tar.gz && \

wget -4 https://geant4-data.web.cern.ch/geant4-data/datasets/G4RealSurface.2.2.tar.gz && \

wget -4 https://geant4-data.web.cern.ch/geant4-data/datasets/G4TENDL.1.3.2.tar.gz && \

tar -zxf G4NDL.4.6.tar.gz && \

tar -zxf G4EMLOW.7.13.tar.gz && \

tar -zxf G4PhotonEvaporation.5.7.tar.gz && \

tar -zxf G4RadioactiveDecay.5.6.tar.gz && \

tar -zxf G4PARTICLEXS.3.1.1.tar.gz && \

tar -zxf G4SAIDDATA.2.0.tar.gz && \

tar -zxf G4ABLA.3.1.tar.gz && \

tar -xzf G4INCL.1.0.tar.gz && \

tar -zxf G4PII.1.3.tar.gz && \

tar -zxf G4ENSDFSTATE.2.3.tar.gz && \

tar -zxf G4RealSurface.2.2.tar.gz && \

tar -zxf G4TENDL.1.3.2.tar.gz && \

rm \*.tar.gz # Delete all tar.gz files after extraction

# 2) Set up the environment:

ENV TOPAS\_G4\_DATA\_DIR /G4Data

# 3) Pre-Requisites:

RUN apt update && apt install -y \

libexpat1-dev \

libgl1-mesa-dev \

libglu1-mesa-dev \

libxt-dev \

xorg-dev \

build-essential \

libharfbuzz-dev \

cmake \

bash \

unzip \

nano \

# Additional packages for Python3

python3 \

python3-pip \

# install git

git && \

rm -rf /var/lib/apt/lists/\*

# 4) Install TOPAS:

# Assuming the TOPAS files are available publicly. You might need to adjust this for your use.

# Set working directory

WORKDIR /tmp

COPY . .

RUN cat topas\_3\_9\_debian9.tar.gz.part\_\* > topas\_3\_9\_debian9.tar.gz && \

tar -zxvf topas\_3\_9\_debian9.tar.gz && \

mv topas /topas && \

rm -rf ./\*

# 5) To add User Extensions (assuming the user does this part manually once in the container):

# Copy the topas\_extensions directory to the container

COPY topas\_extensions /topas\_extensions

# Recompile with extensions

WORKDIR /topas

RUN unzip Geant4Headers.zip && \

cmake -DTOPAS\_EXTENSIONS\_DIR=/topas\_extensions && \

make

# Entry command, here we simply start a bash shell so you can interact with the container

WORKDIR /topas/mytopassimulations

CMD ["/bin/bash"]

**Appendix A.II: Reducer DockerFile**

FROM python:3

WORKDIR /usr/src/app

COPY . .

RUN pip install numpy \

pandas \

matplotlib \

scipy \

azure-storage-blob \

azure-batch \

azure-core \

azure-common

# Set a default command to keep the container running with bash interaction

CMD ["bash"]