
Documentation for Docker Installation and Usage

Release 1.0

Advanced Computational Electromagnetic Model (ACEM) group

Jun 17, 2024

CONTENTS

Docker is a platform for developing, shipping, and running applications. It uses containerization technology to package software and its dependencies into standardized units called containers. These containers are lightweight, portable, and isolated, allowing applications to run consistently across different environments, from development to production.

With Docker, developers can build, ship, and deploy applications faster and more efficiently by eliminating compatibility issues between different systems. Docker simplifies the process of managing and scaling applications, making it easier to deploy and update software in any environment, whether it's on-premises, in the cloud, or hybrid.

Overall, Docker revolutionizes the way applications are developed, deployed, and managed, enabling teams to innovate more rapidly and reliably.

MOTIVATION FOR CONTAINERS

To see why containers are useful, let us look at the following 2 scenarios.

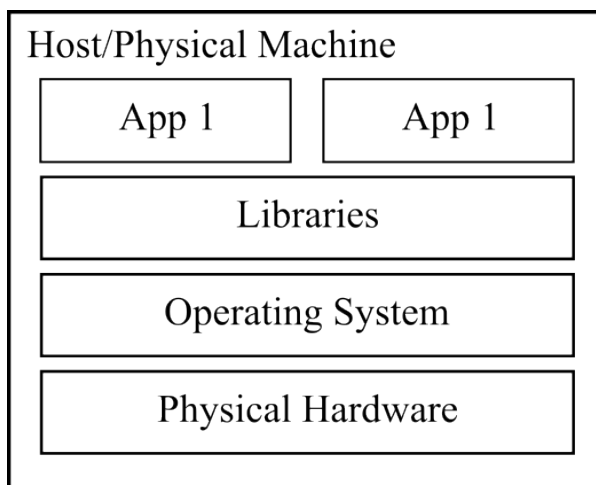
Scenario	Traditional Instructions	With Docker
Development	To set up the development environment, install these libraries and run these scripts	“Run docker compose up”
Deployment	To run the software, prepare Ubuntu of version 22.02. Install these dependencies and run the software	“Run container image with these options”

EVOLUTION OF VIRTUALIZATION

In modern computing, various technologies cater to different needs and scenarios. Bare metal, virtual machines, and containers represent three fundamental approaches to deploying and managing applications, each with its own set of advantages and use cases.

2.1 1. Bare Metal

Bare metal refers to the traditional method of running applications directly on physical servers without any virtualization layer. In this setup, the operating system interacts directly with the underlying hardware. Bare metal environments offer maximum performance and control since all resources are dedicated to the application. They are ideal for high-performance workloads where every bit of computational power counts.



2.2 2. Virtual Machines (VMs)

Virtual machines emulate physical hardware within a host server, allowing multiple operating systems and applications to run independently on the same physical hardware. Each VM runs its own guest operating system, providing strong isolation from other VMs. VMs offer flexibility, enabling different operating systems and configurations to coexist on the same hardware. They are commonly used for server consolidation, testing, and development environments.

When using VM, a hypervisor is usually needed. It is a software or firmware layer that enables the creation and management of virtual machines (VMs) on physical hardware. It sits between the hardware and the operating systems (OS) running on the VMs, facilitating the sharing of physical resources among multiple virtual environments.

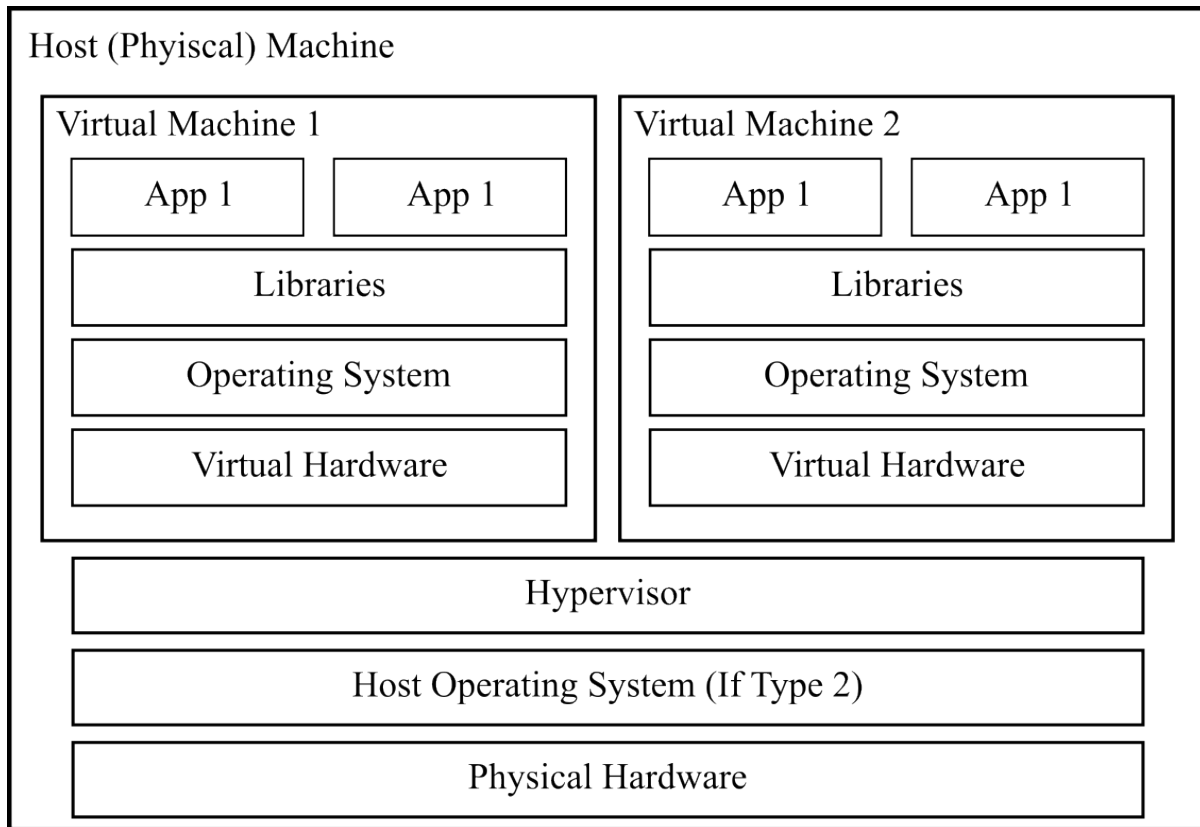
There are two main types of hypervisors:

a. Type 1 Hypervisor (Bare Metal Hypervisor):

Type 1 hypervisors run directly on the physical hardware without the need for a host operating system. They are often referred to as “bare metal” hypervisors because they have direct access to the underlying hardware. Examples include AWS Nitro System, VMware vSphere, and Microsoft Hyper-V. Type 1 hypervisors provide efficient and high-performance virtualization, making them suitable for enterprise-level deployments and cloud computing environments.

b. Type 2 Hypervisor (Hosted Hypervisor):

Type 2 hypervisors run on top of a host operating system, which means they rely on the underlying OS for hardware access. These hypervisors are commonly used on desktop or workstation environments for development, testing, and running multiple operating systems simultaneously. Examples include Oracle VirtualBox, VMware Workstation, and Parallels Desktop. While Type 2 hypervisors offer ease of use and flexibility, they may introduce some performance overhead due to the additional layer of the host OS.



2.3 3. Containers

Containers are lightweight, portable, and isolated environments for running applications and their dependencies. Unlike VMs, containers share the host operating system's kernel, which reduces overhead and improves efficiency. Containers provide fast deployment times and scalability, making them ideal for microservices architectures, continuous integration and deployment (CI/CD) pipelines, and DevOps practices.

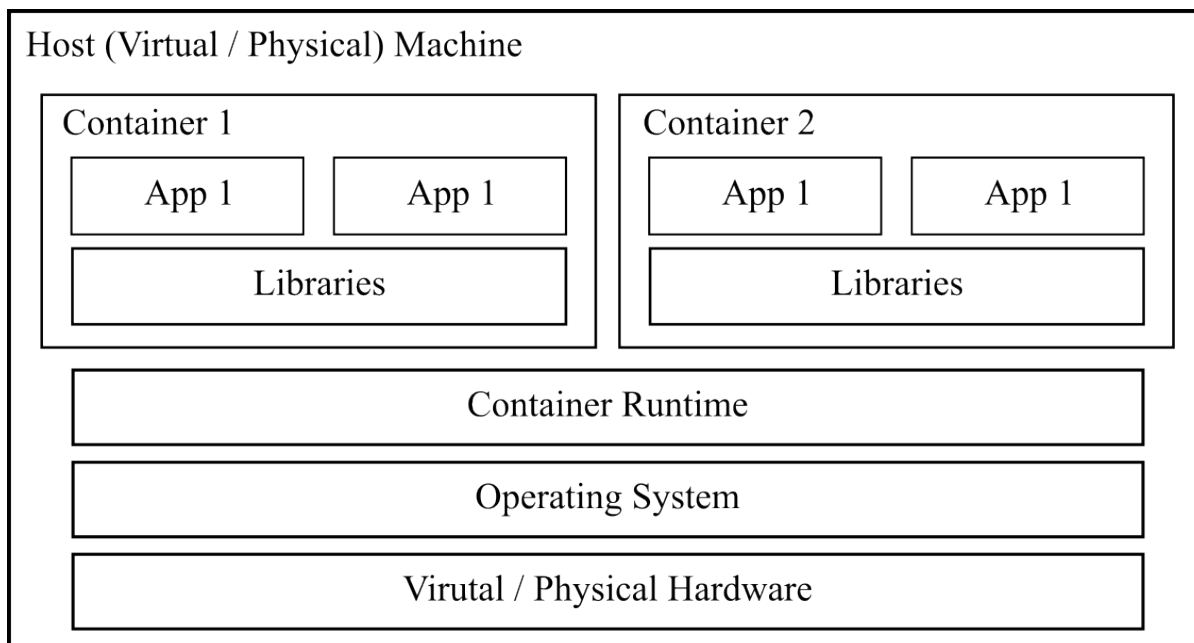


Table 1: Comparison Between Bare Metal, Virtual Machines, and Containers

Aspect	Bare Metal	Virtual Machines	Containers
Isolation	No isolation between applications.	Strong isolation between VMs.	Isolated containers with shared OS kernel.
Resource Utilization	Utilizes all resources of the physical server.	Resources are divided among VMs.	Lightweight, shares OS resources, minimal overhead.
Performance	Offers highest performance as there's no overhead.	Slightly reduced performance due to virtualization layer.	Near-native performance, minimal overhead.
Scalability	Scalability is limited by physical hardware constraints.	Scalable, but constrained by host's resources.	Highly scalable, can run numerous containers on a host.
Deployment Time	Typically longer deployment times due to hardware setup.	Longer deployment times due to virtualization setup.	Very fast deployment times due to lightweight nature.
Flexibility	Less flexible as hardware configurations are fixed.	More flexible due to virtual hardware configurations.	Highly flexible, can package and run various apps.
Resource Overhead	No overhead as resources are dedicated to the system.	Overhead from virtualization layer and guest OS.	Minimal overhead as containers share host resources.
Use Cases	Suitable for high-performance applications.	Used for testing, development, and server consolidation.	Ideal for microservices, CI/CD pipelines, and DevOps practices.

continues on next page

Table 1 – continued from previous page

Aspect	Bare Metal	Virtual Machines	Containers
Dependency management	Bad	Good	Good

2.3.1 Contents

Docker Desktop

Docker Desktop is a powerful tool that simplifies the development, deployment, and management of applications using containers. Here's an introduction to Docker Desktop:

What is Docker Desktop?

Docker Desktop is an application for both Mac and Windows operating systems that enables developers to create, deploy, and manage applications using Docker containers. It provides an easy-to-use interface for working with Docker containers, images, volumes, and networks.

Key Features

1. **Containerization:** Docker Desktop utilizes containerization technology to package applications and their dependencies into lightweight, portable containers. This allows developers to isolate applications from their environment and ensures consistent behavior across different computing environments.
2. **Graphical User Interface (GUI):** Docker Desktop comes with a user-friendly GUI that allows developers to perform container-related tasks such as creating, managing, and monitoring containers using a visual interface. This makes it easy for developers to interact with Docker without needing to use the command line.
3. **Integrated Development Environment (IDE) Integration:** Docker Desktop seamlessly integrates with popular IDEs such as Visual Studio Code, allowing developers to build, debug, and test containerized applications directly from their development environment.
4. **Multi-Platform Support:** Docker Desktop supports both Mac and Windows operating systems, allowing developers to build and run containerized applications on their preferred platform without needing to worry about compatibility issues.
5. **Local Development Environment:** Docker Desktop provides a local development environment that closely mirrors production environments, allowing developers to build and test applications in an environment that closely resembles where they will eventually be deployed.
6. **Automatic Updates:** Docker Desktop regularly receives updates and new features, ensuring that developers have access to the latest Docker capabilities and improvements.

Docker Desktop Architecture

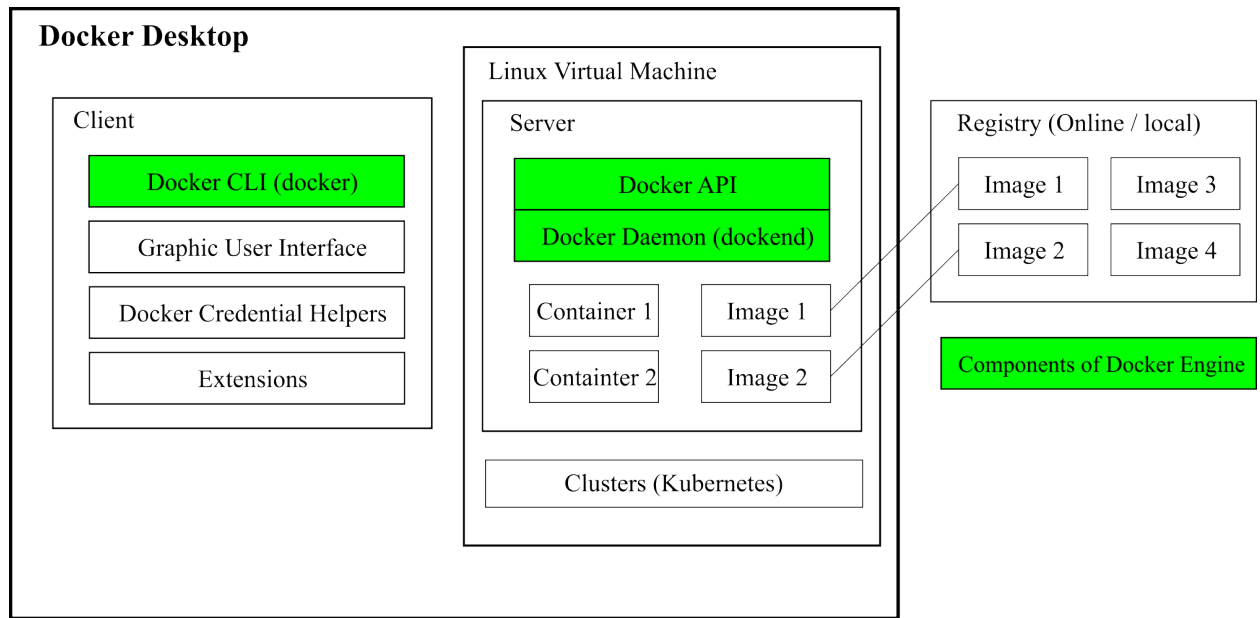
Docker Desktop is a comprehensive package offered by Docker, providing both a graphical user interface (GUI) and a command-line interface (CLI). It serves as a unified platform for developers to manage their Docker containers and workflows seamlessly. One of its key features is its inclusion of a Linux virtual machine, which acts as the runtime environment for Docker containers on non-Linux systems such as Windows and macOS.

Note: For Windows, it is required that the user install Windows Subsystem for Linux (WSL) separately from the installation of Docker Desktop.

This virtual machine wraps around Docker’s core components, including the CLI and the Docker API at the server side, facilitating the creation, management, and deployment of containers. Additionally, Docker Desktop includes the Docker Daemon, which is responsible for managing container lifecycle and interacting with the operating system’s kernel.

While in a Linux environment, working directly with the Docker Engine, a lighter version of Docker, is often sufficient. However, for Windows users, especially those on Windows 11, Docker Desktop is recommended. This recommendation is primarily due to compatibility reasons, especially for users requiring NVIDIA GPU support for containerized applications.

In summary, Docker Desktop provides a unified solution for Docker container management, combining a user-friendly interface with essential tools and runtime environments to streamline the development and deployment of containerized applications across different operating systems.



Difference between images and containers

Container

- **Execution Instance:** A container is a runtime instance of an image. It's a lightweight, standalone, and executable package that encapsulates all the dependencies required to run a piece of software.
- **Isolation:** Containers provide process isolation, ensuring that applications running within them are isolated from one another and from the host system.
- **Dynamic:** Containers can be started, stopped, moved, and deleted dynamically. They are ephemeral by nature, meaning they can be created, destroyed, and replaced rapidly.
- **Runtime Environment:** Each container runs in its own isolated environment, including its own filesystem, networking, and process space.

Image

- **Blueprint:** An image is a static, immutable blueprint used to create containers. It's a snapshot of a container that includes the application code, runtime, libraries, dependencies, and other configuration needed to run the software.
- **Build Artifact:** Images are created either manually or through automated build processes. They are stored in a registry and serve as the foundation for creating containers.
- **Reusable:** Images can be shared, reused, and distributed across different environments and systems. They provide a consistent environment for running applications regardless of the underlying infrastructure.
- **Layered Structure:** Docker images are composed of multiple layers, with each layer representing a filesystem change. This layered structure enables efficient storage, distribution, and reuse of images.

Key Differences

1. **Statefulness:** Containers are dynamic and stateful, while images are static and stateless. Containers can be modified and retain changes during runtime, whereas images remain immutable.
2. **Lifecycle:** Containers have a lifecycle—they can be created, started, stopped, paused, and deleted. Images, on the other hand, do not have a lifecycle; they are static artifacts used to create containers.
3. **Composition:** Images serve as the building blocks for containers. Multiple containers can be instantiated from the same image, each running independently with its own isolated environment.
4. **Persistence:** Containers are ephemeral, meaning any changes made inside a container only persist as long as the container is running. Images, being immutable, preserve the state of the filesystem at the time of creation.

In essence, containers are the runtime instances of images, while images are the blueprints used to create containers. They work together to enable the efficient deployment, management, and execution of applications in a containerized environment.

Installation in Windows

Generally, enabling virtualization from the BIOS and installing Windows Subsystem for Linux (WSL) are crucial steps for setting up Docker Desktop on Windows machines. Here's an updated summary of the general steps for Windows installation, including these prerequisites and the links you provided:

1. **Enable Virtualization from BIOS:** Before installing Docker Desktop, ensure that virtualization is enabled in your system's BIOS settings. This step is crucial for Docker Desktop to function properly on Windows machines. Refer to this link for instructions on how to enable virtualization: [Hardware-Assisted Virtualization and Data Execution Protection](<https://forums.docker.com/t/hardware-assisted-virtualization-and-data-execution-protection-must-be-enabled-in-the-bios/109073>).

2. **Install Windows Subsystem for Linux (WSL):** Docker Desktop relies on WSL to run Linux containers on Windows. Install WSL by following the instructions provided in this link: [Install Windows Subsystem for Linux (WSL)](<https://learn.microsoft.com/en-us/windows/wsl/install>).
3. **Download Docker Desktop Installer:** Visit the official Docker website and navigate to the Windows installation page. From there, download the Docker Desktop installer executable.
4. **Run the Installer:** After downloading the installer, double-click on it to launch the installation wizard.
5. **Follow Installation Instructions:** Follow the on-screen instructions provided by the installation wizard. This includes accepting the license agreement, choosing installation options, and specifying the installation location.
6. **System Requirements Check:** During the installation process, Docker Desktop may perform a system requirements check to ensure your system meets the necessary prerequisites for installation.
7. **Enable Hyper-V (if required):** Docker Desktop for Windows may require Hyper-V to be enabled. (It can also work with WSL) If it's not already enabled, the installer will prompt you to enable it. This step might require a system reboot.
8. **Configuration:** Once the installation is complete, Docker Desktop may require additional configuration, such as configuring shared drives or network settings.
9. **Launch Docker Desktop:** After successful installation and configuration, Docker Desktop can be launched from the Start menu or desktop shortcut.
10. **Sign in (if required):** Depending on your Docker configuration, you may need to sign in to your Docker account during the initial setup. Sign-in is usually optional.
11. **Verify Installation:** To ensure Docker Desktop is installed correctly, you can open a command prompt or PowerShell window and run `docker --version` to verify the Docker CLI version.
12. **Start Using Docker:** With Docker Desktop installed and running, you're ready to start building, running, and managing containers on your Windows system. To use Docker CLI, users will have to start Docker Desktop.

By following these steps, including enabling virtualization from the BIOS and installing WSL, you can successfully install Docker Desktop on your Windows machine and leverage the benefits of containerization for your development projects.

Enable Virtualization for Windows Machine

Accessing the BIOS (Basic Input/Output System) varies slightly depending on your computer's manufacturer and model. Here's a general guide on how to access the BIOS on most Windows computers:

1. **Shut Down Your Computer:** Ensure that your computer is fully powered off, not just in sleep or hibernate mode.
2. **Start Your Computer:** Press the power button to turn on your computer.
3. **Start Tapping the BIOS Key:** Immediately after pressing the power button, start tapping the appropriate key to enter the BIOS. The BIOS key varies depending on your computer's manufacturer. Common keys include: - **F2** - **F10** - **F12** - **Esc** - **Delete** - **Enter**
4. **Check for BIOS Key:** If you're unsure which key to press, check your computer's manual or the manufacturer's website. Sometimes, the BIOS key is displayed briefly on the screen during startup.
5. **Enter BIOS Setup:** Continuously tapping the BIOS key should bring you to the BIOS setup screen. This screen typically has a blue or black background with various options and settings.
6. **Navigate BIOS Menu:** Once inside the BIOS setup, you can navigate using the arrow keys on your keyboard. Be cautious when making changes in the BIOS, as incorrect settings can affect the stability and performance of your computer.

7. **Enable Virtualization:** Look for settings related to virtualization (e.g., Intel Virtualization Technology, AMD-V) in the BIOS setup. Depending on your BIOS version and layout, virtualization settings may be located under different menus such as “Advanced,” “Security,” or “CPU Configuration.” Enable virtualization if it’s disabled.
8. **Save and Exit:** After making changes, save your settings and exit the BIOS setup. This process is usually done by pressing the appropriate key (often labeled as “Save & Exit” or similar) and confirming your choice.
9. **Restart Your Computer:** Once you’ve exited the BIOS setup, your computer will restart.
10. **Verify Virtualization:** After restarting, you can verify if virtualization is enabled by checking your computer’s system information or BIOS settings.

Remember, accessing the BIOS can vary depending on your computer’s manufacturer and model, so if you’re unsure, refer to your computer’s manual or the manufacturer’s website for specific instructions.

Installing WSL

To install Windows Subsystem for Linux (WSL) and manage its settings, follow these steps:

1. **Install WSL:** Open PowerShell or Windows Command Prompt as an administrator and run the command “wsl –install”. This command enables the necessary features to run WSL and installs the default Ubuntu distribution of Linux. After running the command, restart your machine.
2. **Check WSL Version:** To check which version of WSL you are running and view your installed Linux distributions, use the command “wsl -l -v” in PowerShell or Command Prompt.
3. **Set Default WSL Version:** If you want to set the default version to WSL 1 or WSL 2, use the command “wsl –set-default-version <Version#>”, replacing <Version#> with either 1 or 2.
4. **Set Default Linux Distribution:** To set the default Linux distribution used with the “wsl” command, enter “wsl -s <DistributionName>” or “wsl –set-default <DistributionName>”, replacing “<DistributionName>” with the name of the Linux distribution you want to use.

Following these steps allows you to easily install and manage Windows Subsystem for Linux (WSL) on your Windows machine, enabling you to run Linux commands and utilities alongside your Windows environment.

Installation in Linux

In general, Docker Engine can be installed on Ubuntu Linux using the instructions provided in the official Docker documentation at <https://docs.docker.com/engine/install/ubuntu/>. This documentation outlines the steps necessary to install Docker Engine on an Ubuntu system, ensuring that you have the latest version of Docker available for your use.

1. Set up Docker’s apt repository.

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.
↵asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://
↵/download.docker.com/linux/ubuntu \
```

(continues on next page)

(continued from previous page)

```
$(. /etc/os-release && echo "$VERSION_CODENAME" stable" | \  
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null  
sudo apt-get update
```

2. To install the latest version, run:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-  
↪compose-plugin
```

3. Verify that the Docker Engine installation is successful by running the hello-world image.

```
sudo docker run hello-world
```

Supported Operating Systems

Docker Desktop is available for installation on the following operating systems:

1. **Windows 10/11:** Docker Desktop is compatible with Windows 10 and Windows 11 operating systems. It leverages the native Hyper-V virtualization technology on Windows to run containers.

Warning: To ensure NVIDIA GPU compatibility, please use the latest updated Windows 10 or Windows 11. When we tested it with Windows 10 on our machine (not with the latest patches), the code ran into memory errors. However, after updating to Windows 11, the software ran without issues.

More information can be found in the following links:

- [NVIDIA Container Toolkit Issue #226](#)
- [NVIDIA Developer Forum Thread](#)

2. **Mac:** Docker Desktop is compatible with macOS, allowing Mac users to build and manage containers seamlessly using the Docker Desktop application.
3. **Linux (Ubuntu Distro 22 and above):** Docker Desktop is supported on Linux systems running Ubuntu distributions version 22 and above. It provides a convenient way for Linux users to utilize Docker containers within their development workflow.

Note: For Linux Ubuntu Distro 20, we can only install Docker Engine, which is sufficient for most cases.

Docker Engine

Docker Engine is a powerful tool that simplifies the process of creating, deploying, and managing applications using containers. Here's an introduction to Docker Engine and its basic functionalities:

What is Docker Engine?

Docker Engine is the core component of Docker, a platform that enables developers to package applications and their dependencies into lightweight containers. These containers can then be deployed consistently across different environments, whether it's a developer's laptop, a testing server, or a production system.

Basic Functionalities of Docker Engine

1. **Containerization:** Docker Engine allows you to create and manage containers, which are isolated environments that package an application and its dependencies. Containers ensure consistency in runtime environments across different platforms.
2. **Image Management:** Docker uses images as templates to create containers. Docker Engine allows you to build, push, and pull images from Docker registries (like Docker Hub or private registries). Images are typically defined using a Dockerfile, which specifies the environment and setup instructions for the application.
3. **Container Lifecycle Management:** Docker Engine provides commands to start, stop, restart, and remove containers. It also manages the lifecycle of containers, including monitoring their status and resource usage.
4. **Networking:** Docker Engine facilitates networking between containers and between containers and the outside world. It provides mechanisms for containers to communicate with each other and with external networks, as well as configuring networking options like ports and IP addresses.
5. **Storage Management:** Docker Engine manages storage volumes that persist data generated by containers. It supports various storage drivers and allows you to attach volumes to containers, enabling data persistence and sharing data between containers and the host system.
6. **Resource Isolation and Utilization:** Docker Engine uses Linux kernel features (such as namespaces and control groups) to provide lightweight isolation and resource utilization for containers. This ensures that containers run efficiently without interfering with each other or with the host system.
7. **Integration with Orchestration Tools:** Docker Engine can be integrated with orchestration tools like Docker Swarm and Kubernetes for managing containerized applications at scale. Orchestration tools automate container deployment, scaling, and load balancing across multiple hosts.

Key Benefits of Docker Engine

- **Consistency:** Docker ensures consistency between development, testing, and production environments by encapsulating applications and dependencies into containers.
- **Efficiency:** Containers are lightweight and share the host system's kernel, reducing overhead and improving performance compared to traditional virtual machines.
- **Portability:** Docker containers can run on any platform that supports Docker, making it easy to move applications between different environments.
- **Isolation:** Containers provide a level of isolation that enhances security and stability, as each container operates independently of others on the same host.

Basic Steps to build Docker images

1. Create a **Dockerfile** that provides instructions to build Docker image.

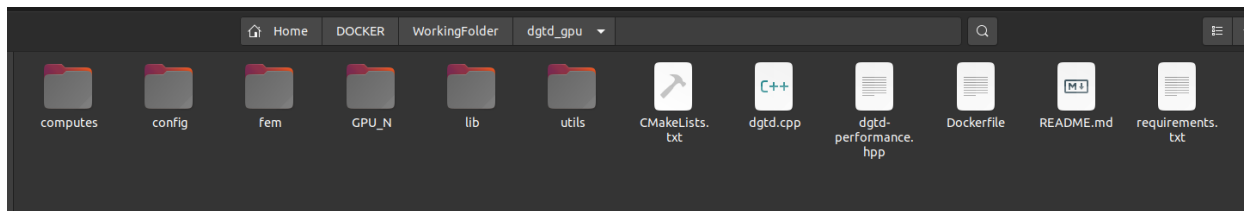


Fig. 1: Placing Dockerfile in the directory where we want to create Docker image

We can examine the contents of the Dockerfile.

We're creating a Docker image by starting from an existing Docker image that includes a UNIX environment with CUDA runtime. Initially, we pull the base image from Docker's official repository. Specifically, we can locate suitable base images by searching on [Docker Hub](https://hub.docker.com/search?q=nvidia%2Fcuda) (<https://hub.docker.com/search?q=nvidia%2Fcuda>).

```
# -----
↪ --
# Step 1 : Import a base image from online repository
# FROM nvidia/cuda:12.5.0-devel-ubuntu22.04
FROM nvidia/cuda:12.4.0-devel-ubuntu22.04
```

In addition, setting `ENV DEBIAN_FRONTEND=noninteractive` in a Dockerfile is a directive that adjusts the environment variable `DEBIAN_FRONTEND` within the Docker container during the image build process.

- **Non-interactive Environment**

Debian-based Linux distributions, including many Docker base images, use `DEBIAN_FRONTEND` to determine how certain package management tools (like `apt-get`) interact with users. Setting `DEBIAN_FRONTEND=noninteractive` tells these tools to run in a non-interactive mode. In this mode, the tools assume default behavior for prompts that would normally require user input, such as during package installation or configuration.

- **Avoiding User Prompts**

During Docker image builds, it's crucial to automate as much as possible to ensure consistency and reproducibility. Without setting `DEBIAN_FRONTEND=noninteractive`, package installations might prompt for user input (e.g., to confirm installation, choose configuration options). This interaction halts the build process unless explicitly handled in advance.

- **Common Usage in Dockerfiles**

In Dockerfiles, especially those designed for automated builds (CI/CD pipelines, batch processes), it's typical to include `ENV DEBIAN_FRONTEND=noninteractive` early on. This ensures that subsequent commands relying on package management tools proceed without waiting for user input.

```
# -----
↪ --
# Step 2 : Suppress Interactive Prompts from Debian
ENV DEBIAN_FRONTEND=noninteractive
```

Next, we will need to install packages, libraries and set the environment variables that we need to compile or run Maxwell-TD.

his Dockerfile snippet outlines steps for setting up a Docker image with various libraries and tools typically required for scientific computing and development environments. Let's break down each part:

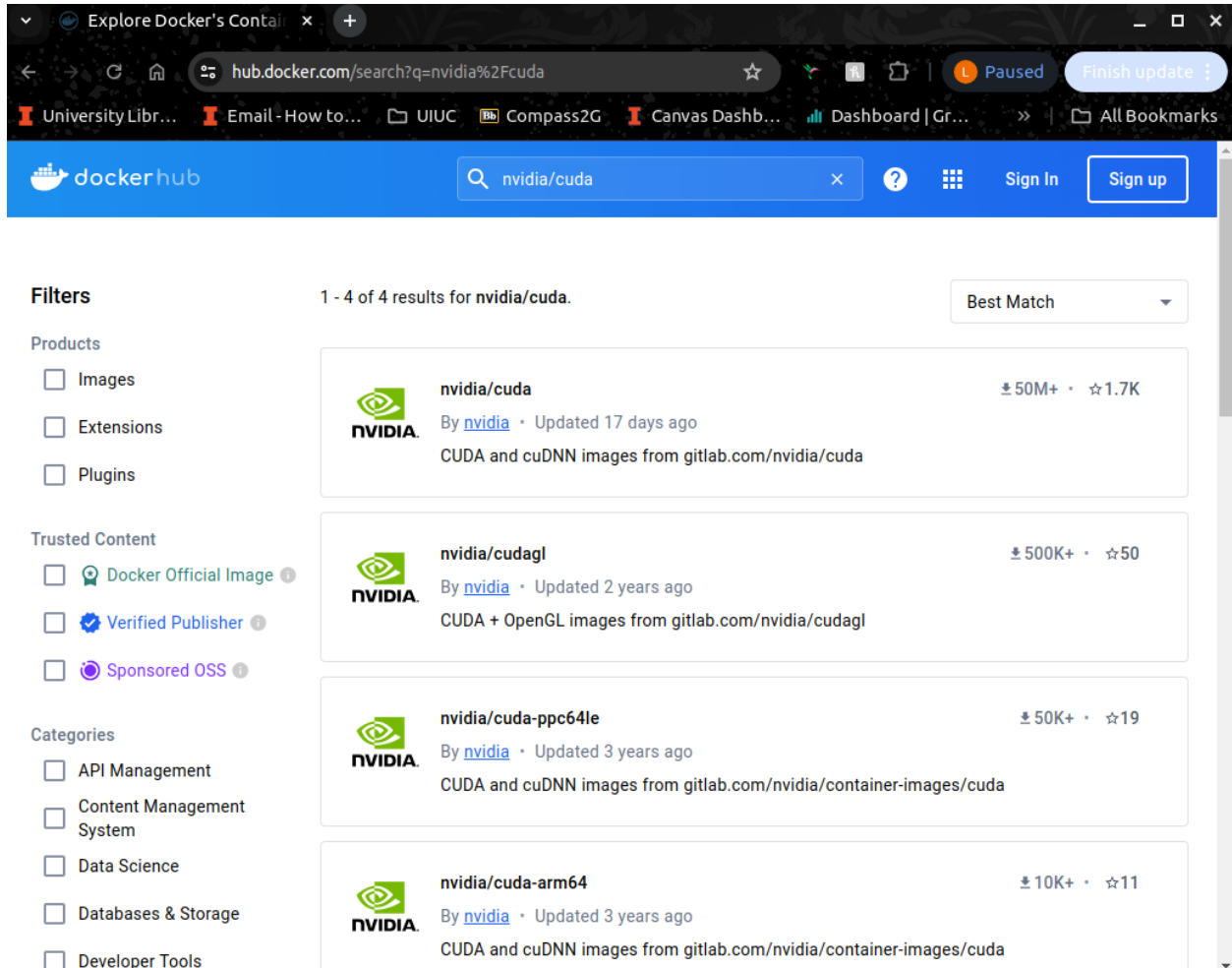


Fig. 2: Find base images online via *Docker Hub* <<https://hub.docker.com/search?q=nvidia%2Fcuda>>

- **Update system and install libraries**

- **Purpose:** Updates the package list and installs a set of essential libraries and tools required for compiling and building various applications.

- **Packages Installed:**

- * **build-essential, g++, gcc:** Compiler tools and libraries.

- * **cmake, gfortran:** Build and Fortran compiler.

- * Various development libraries (libopenblas-dev, liblapack-dev, libfftw3-dev, etc.) for numerical computations, linear algebra, and scientific computing.

- * **libvtk7-dev:** Libraries for 3D computer graphics, visualization, and image processing.

- * **libgomp1, libomp-dev, libpthread-stubs0-dev:** Libraries for multi-threading support.

- **Install Compilers**

- **Purpose:** Ensures that g++ and gcc are installed. These are essential compilers for C++ and C programming languages, often needed for compiling native code.

- **Install Python and pip**

- **Purpose:** Installs Python 3 and pip (Python package installer), which are essential for Python-based applications and managing Python dependencies.

- **Copy current directory to docker image**

- **Purpose:** Sets the working directory inside the Docker image to /dgttd and copies all files from the current directory (presumably where the Dockerfile resides) into the /dgttd directory inside the Docker image.

- **Install Python dependencies**

- **Purpose:** Installs Python dependencies listed in requirements.txt file located in the /dgttd directory. The --no-cache-dir flag ensures that no cached packages are used during installation, which can be important for Docker images to maintain consistency and avoid unexpected behavior.

- **Set Path for libraries and CUDA**

- **Purpose:** Sets environment variables related to CUDA (a parallel computing platform and programming model) if CUDA is used in the project. These variables define paths to CUDA libraries, binaries, headers, and compiler (nvcc).

```
# -----
↪ --
# Step 3 : Installing required packages and setting up environment variables

# Update system and install libraries
RUN apt-get update && apt-get install -y \
    build-essential \
    cmake \
    gfortran \
    libopenblas-dev \
    liblapack-dev \
    libfftw3-dev \
    libmetis-dev \
    libvtk7-dev \
    libgomp1 \
    libomp-dev \
    libblas-dev \
    libpthread-stubs0-dev \
    && rm -rf /var/lib/apt/lists/*
```

(continues on next page)

(continued from previous page)

```
# Install Compilers
RUN apt-get update && apt-get install -y g++ gcc

# Install Python and pip
RUN apt-get install -y python3 python3-pip

# Copy current directory to docker image
WORKDIR dgtd
COPY . .

# Install Python dependencies
RUN pip install --no-cache-dir -v -r /dgtd/requirements.txt

# Set Path for libraries and CUDA
ENV CUDA_HOME /usr/local/cuda
ENV LD_LIBRARY_PATH ${CUDA_HOME}/lib64
ENV PATH ${CUDA_HOME}/bin:${PATH}
ENV CPATH ${CUDA_HOME}/include:${CPATH}
ENV CUDACXX ${CUDA_HOME}/bin/nvcc
```

Finally, we can compile a program using `cmake` and `make`, and then sets up the Docker container to start a Bash shell upon running. `CMD ["bash"]` sets the default command to run inside the container. When the container is started without specifying a command, it will automatically launch a Bash shell.

```
# -----
↪ --
# Step 4 : Compiling Program
RUN ls -l && mkdir build && cd build && cmake .. && make -j 4
CMD ["bash"]
```

2. Use the `docker build` command to build the Docker image from your Dockerfile.

Once you have created your Dockerfile and saved it in your project directory, you can build a Docker image using the `docker build` command. Here's how you would do it:

```
docker build -t maxwell_td_image .
```

- `docker build`: This command tells Docker to build an image from a Dockerfile.
- `-t maxwell_td_image`: The `-t` flag is used to tag the image with a name (`maxwell_td_image` in this case). This name can be whatever you choose and is used to refer to this specific image later on.
- `.`: This specifies the build context. The dot indicates that the Dockerfile and any other files needed for building the image are located in the current directory.

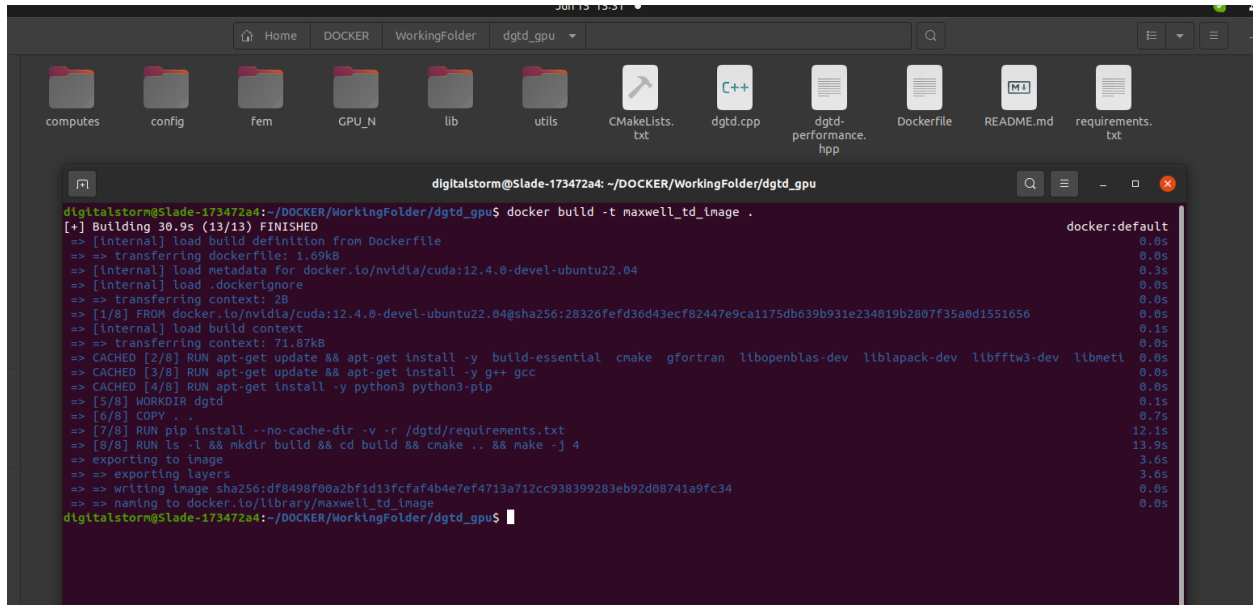


Fig. 3: Run docker build to build Docker image

How to check images

To verify if a Docker image has been successfully built on your local system, you can use the `docker images` command. Here's how you can do it:

- Open your terminal (Command Prompt on Windows or Terminal on macOS/Linux).
- Run the following command

```
docker images
docker images ls
```

This command will list all Docker images that are currently present on your local system. Each image listed will have columns showing its repository, tag, image ID, creation date, and size.

- Finding your image

Look through the list for the image you just built. If it was successfully built, it should appear in the list. Check the repository and tag names to identify your specific image. The repository name will likely be the name you assigned to it in your Dockerfile, and the tag will be `latest` or another tag you specified.

- Confirming successful build

If your image appears in the list with the correct details (repository name, tag, etc.), it indicates that Docker successfully built and stored the image on your local machine.

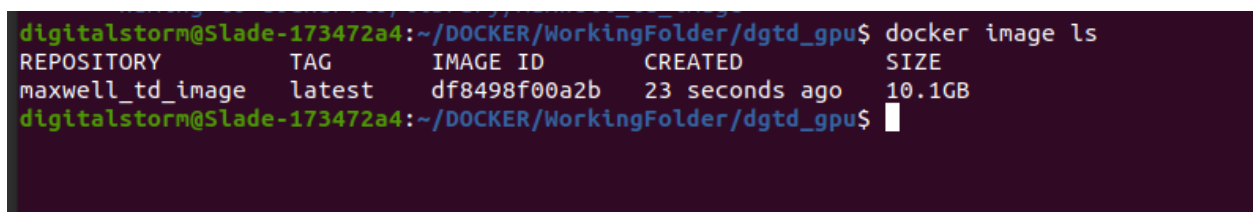


Fig. 4: Check Docker image

How to save built image locally

To save a Docker image locally as a tar archive, you'll use the `docker save` command. This command packages the Docker image into a tarball archive that can be transferred to other machines or stored for backup purposes. Here's how you can do it:

- Open your terminal (Command Prompt on Windows or Terminal on macOS/Linux).
- Run the following command

```
docker save -o <output-file-name>.zip <image-name>
```

Replace `<output-file-name>.zip` with the desired name for your zip archive file.

`<image-name>`: This specifies the Docker image you want to save.

- Confirmation:

After running the command, Docker will package the specified image into zip named `<output-file-name>.zip`. You should see the zipfile (`<output-file-name>.zip`) in your current directory unless you specified a different path for the output.

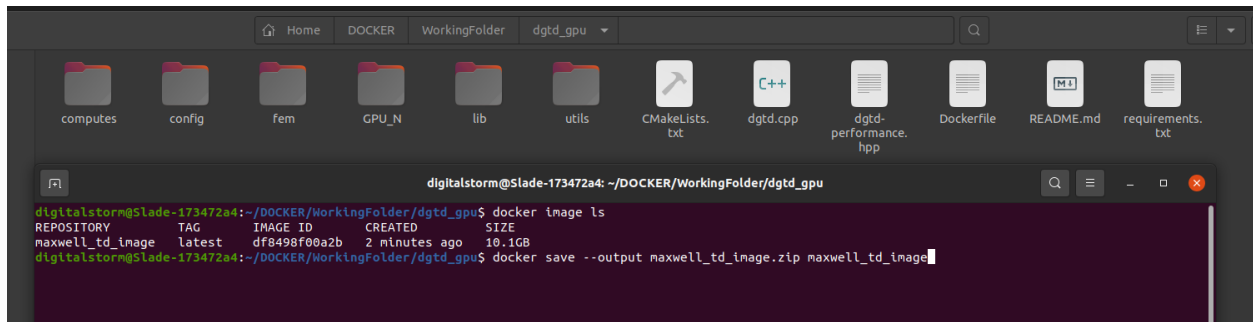


Fig. 5: Save Docker image

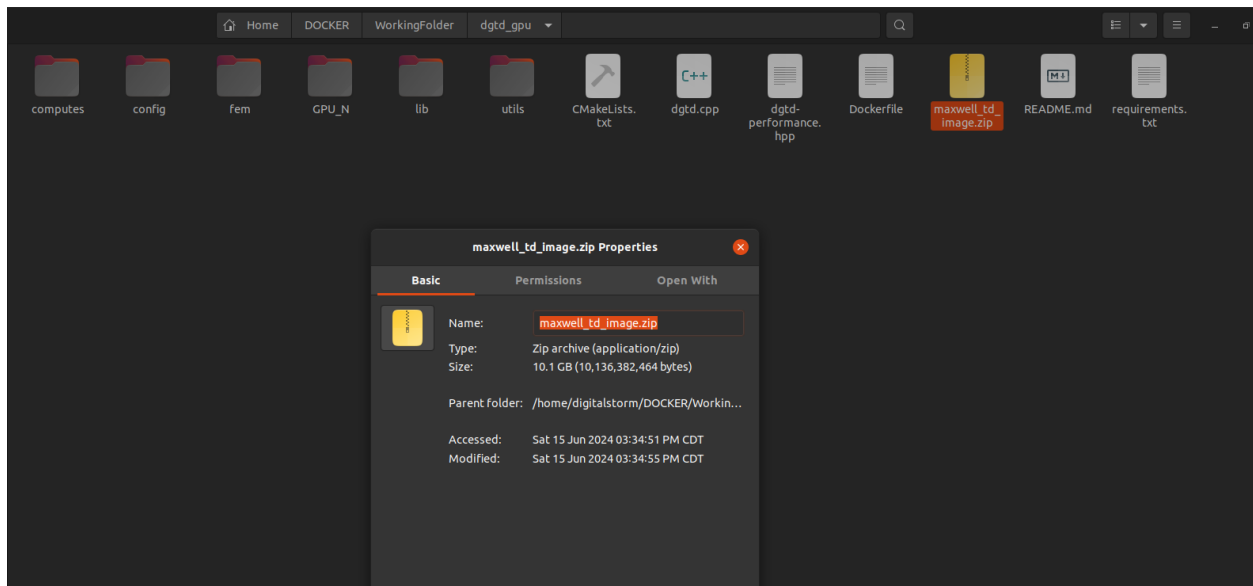


Fig. 6: Resulting zipped Docker image

Note:

- **Transportability:** The generated zip can be transferred to another machine or stored for future use. This is useful for deploying Docker images across different environments without needing to rebuild them.
- **File Size:** Depending on the size of your Docker image, the resulting zip archive can be quite large. Ensure you have enough disk space and consider compression techniques if transferring over networks with limited bandwidth.
- **Loading the Image:** To use the saved zip file on another machine, you'll need to load it into Docker using the docker load command. Here's how you can do that:

```
docker load -i <path/to/your/image.tar>
```

Replace <path/to/your/image.tar> with the actual path to your saved tar archive file.

Managing Docker Resources

Cleaning Up Unused Resources

To clean up any dangling resources (images, containers, volumes, networks), use the following command:

```
docker system prune
```

To remove stopped containers and all unused images (not just dangling), add the *-a* flag:

```
docker system prune -a
```

Removing Docker Images

To remove specific images, list them using `docker images -a` and then delete them with `docker rmi`:

```
docker images -a           # List all images
docker rmi Image Image    # Remove specific images by ID or tag
```

To remove dangling images, use:

```
docker image prune
```

Removing Docker Containers

To remove specific containers, list them using `docker ps -a` and then delete them with `docker rm`:

```
docker ps -a              # List all containers
docker rm ID_or_Name ID_or_Name # Remove specific containers by ID or name
```

To remove all exited containers, use:

```
docker rm $(docker ps -a -f status=exited -q)
```

Removing Docker Volumes

To remove specific volumes, list them using `docker volume ls` and then delete them with `docker volume rm`:

```
docker volume ls          # List all volumes
docker volume rm volume_name volume_name # Remove specific volumes by name
```

To remove dangling volumes, use:

```
docker volume prune
```

User Manual (Windows System)

Step-by-Step installation in Windows

Welcome to the user manual for setting up and using Docker on a Windows system. This guide will provide comprehensive instructions to help you install and configure Docker Desktop on Windows, activate virtualization in BIOS, install WSL (Windows Subsystem for Linux), import Docker images, mount user data, and run Docker containers. Additionally, it includes pointers and tips to address common questions and issues that users may encounter.

Activate Virtualization in BIOS

To enable virtualization, you'll need to access the BIOS settings during startup. Here's how:

1. Power on your computer.
2. Keep an eye on the initial boot screen.
3. When you see the screen, press the F2 / F10 / F12 / Esc / Delete / Enter key to enter the BIOS settings.

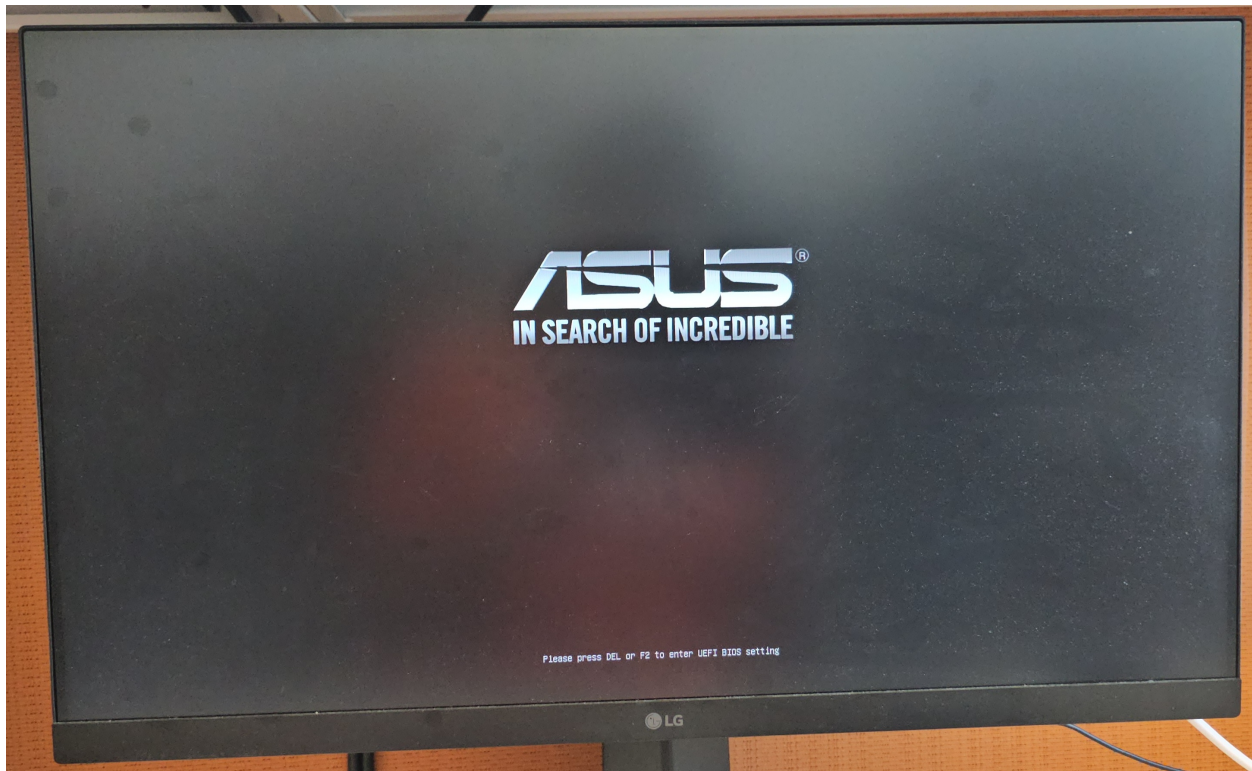


Fig. 7: Screen when we boot up the machine. Press F2 / F10 / F12 / Esc / Delete / Enter

Once inside the BIOS settings:

- Navigate through the BIOS menus using the arrow keys. Look for the section typically labeled Advanced, Advanced Settings or CPU Configuration.

or

- Alternatively, some BIOS versions allow you to search for settings. Look for a search bar or a similar feature where you can type virtualization or VT-x (for Intel processors) / AMD-V (for AMD processors).

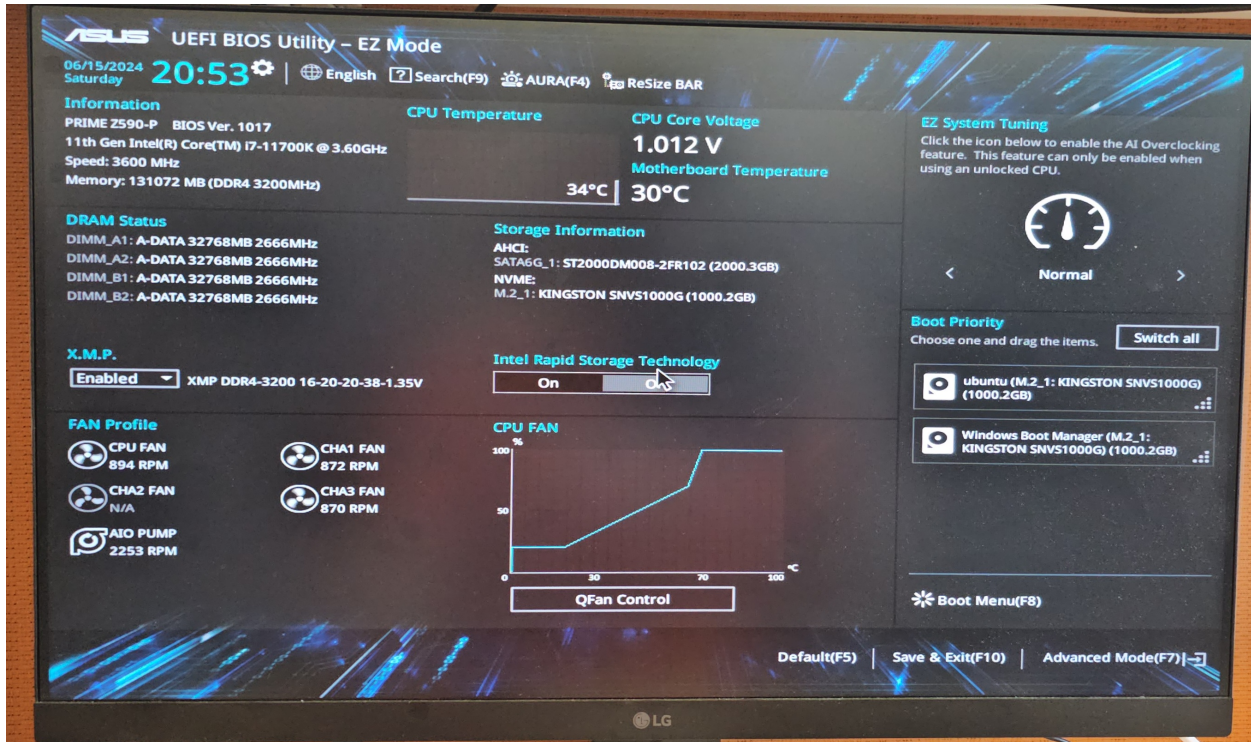


Fig. 8: BIOS settings screen.

Once you locate the virtualization options within these sections, you can proceed to enable them. This step is crucial for utilizing virtualization features such as running virtual machines or other virtualization-based applications on your computer. After making the necessary changes, remember to save your settings and exit the BIOS.

- Change the setting from Disabled to Enabled using the appropriate key (usually Enter or the + key).
- Save changes and exit BIOS

Your computer will restart with virtualization support enabled.

Install WSL

Here are the simplified instructions for installing Windows Subsystem for Linux (WSL) and verifying its installation:

Installing Windows Subsystem for Linux (WSL)

- Open PowerShell as Administrator**
- Install WSL:** Run the following command to install WSL: `wsl --install`. Follow any prompts or confirmations that appear during the installation process.
- Wait for Installation to Complete:** Allow the installation process to finish. This may take some time depending on your internet speed.

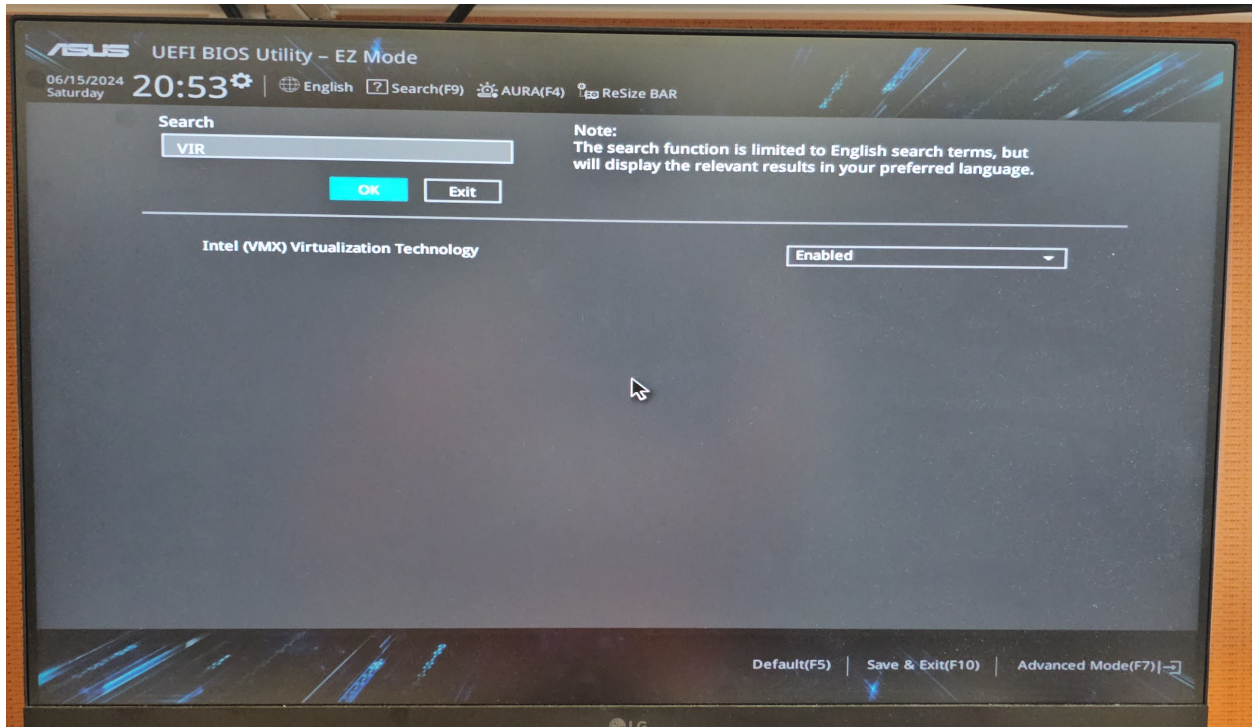


Fig. 9: Turn on Virtualization.

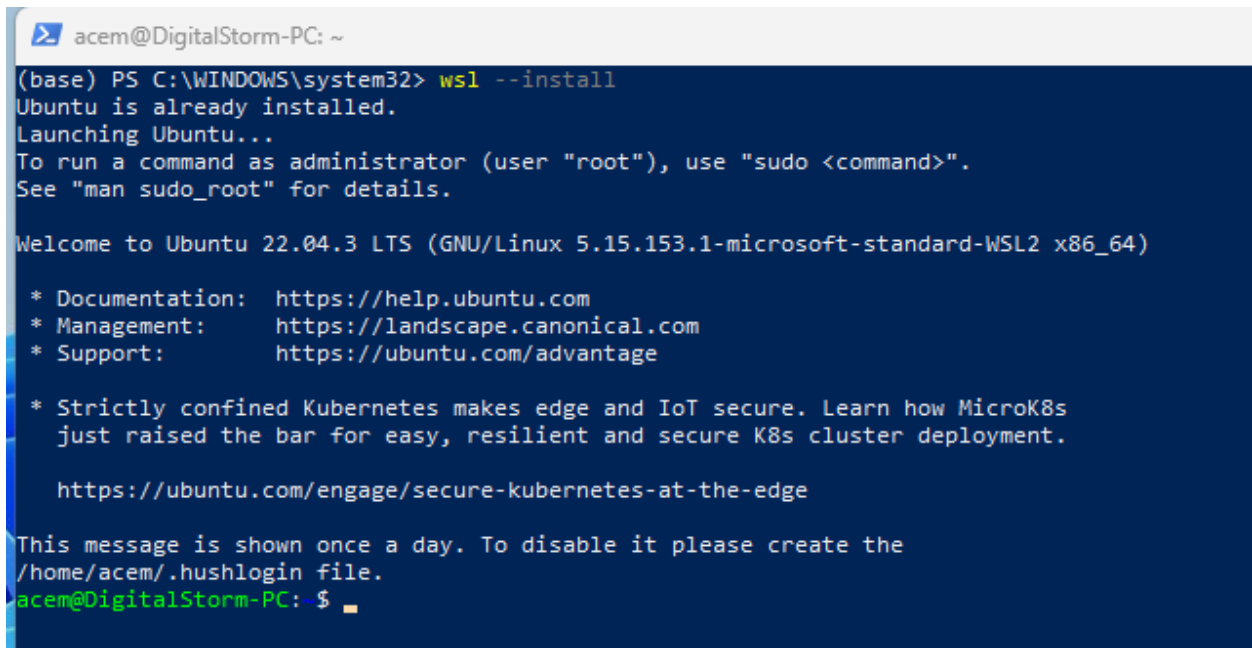
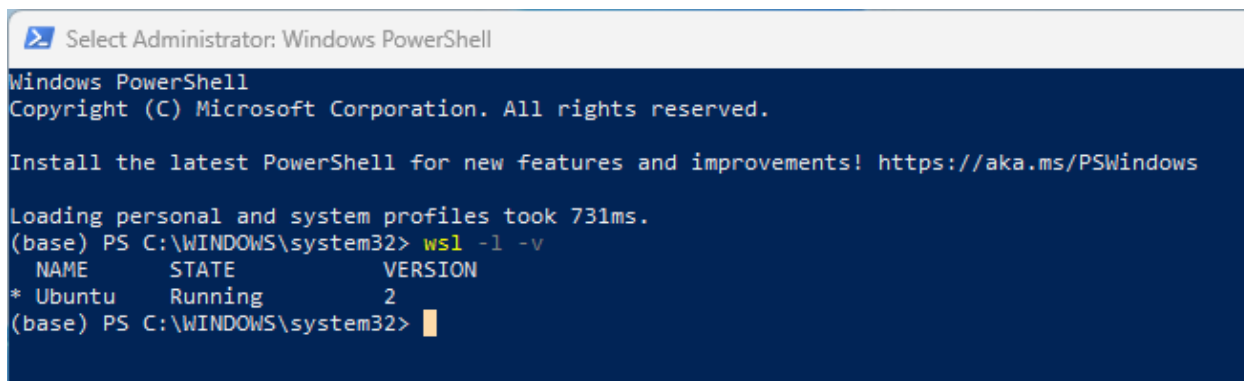


Fig. 10: Install Windows Subsystem for Linux (WSL)

4. **Check WSL Version:** Run the following command to check the installed WSL versions and their respective distributions: `wsl -l -v` This command lists all installed Linux distributions and their associated WSL versions (1 or 2).



```

Select Administrator: Windows PowerShell

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

Loading personal and system profiles took 731ms.
(base) PS C:\WINDOWS\system32> wsl -l -v
  NAME      STATE      VERSION
* Ubuntu    Running    2
(base) PS C:\WINDOWS\system32>

```

Fig. 11: Check if WSL is properly installed

Following these steps ensures that you install Windows Subsystem for Linux (WSL) and confirm its proper installation on your Windows system.

Install Docker Desktop in Windows

1. Download Docker Desktop from Docker's official website (<https://www.docker.com/get-started/>).

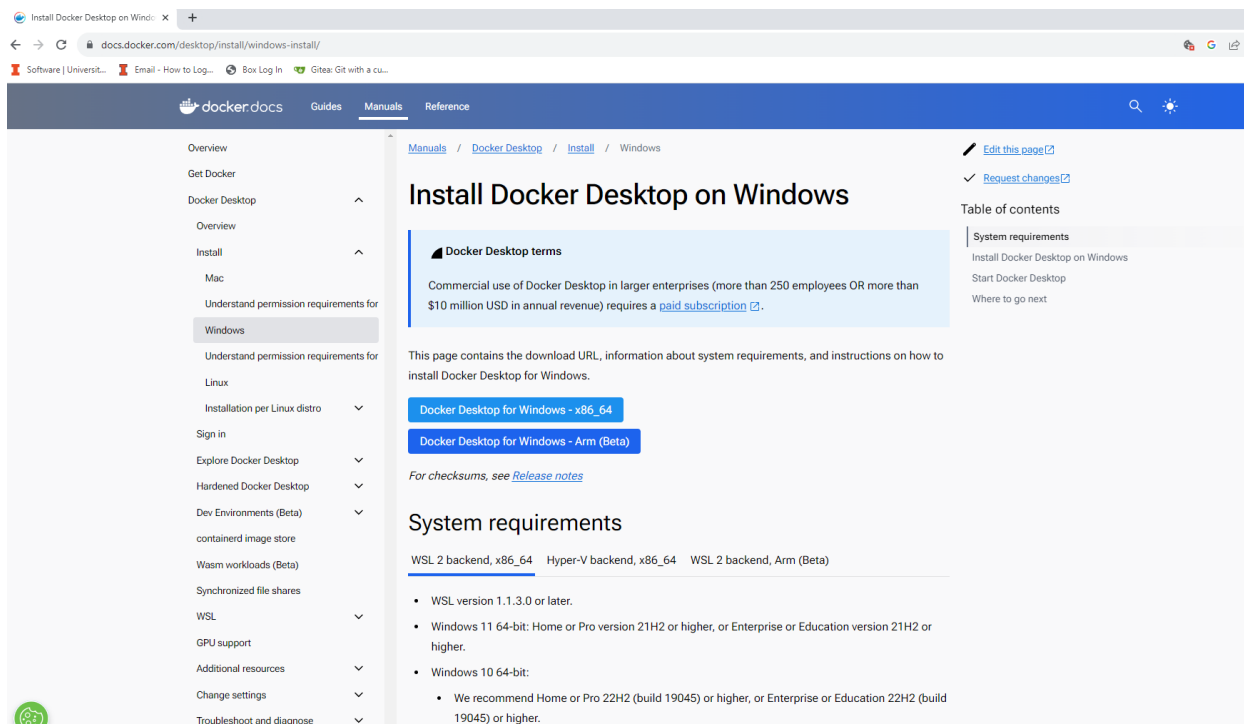


Fig. 12: Download Docker Desktop online

2. Once downloaded, locate the installer file (e.g., `DockerDesktopInstaller.exe`) and double-click to launch it.

3. Follow the prompts provided by the Docker Desktop installer to complete the installation process. This may involve accepting terms and conditions and choosing installation preferences.


Name	Date modified	Type	Size
▼ Today			
 Docker Desktop Installer (1)	6/15/2024 9:05 PM	Application	487,594 KB

Fig. 13: Docker Desktop Installer

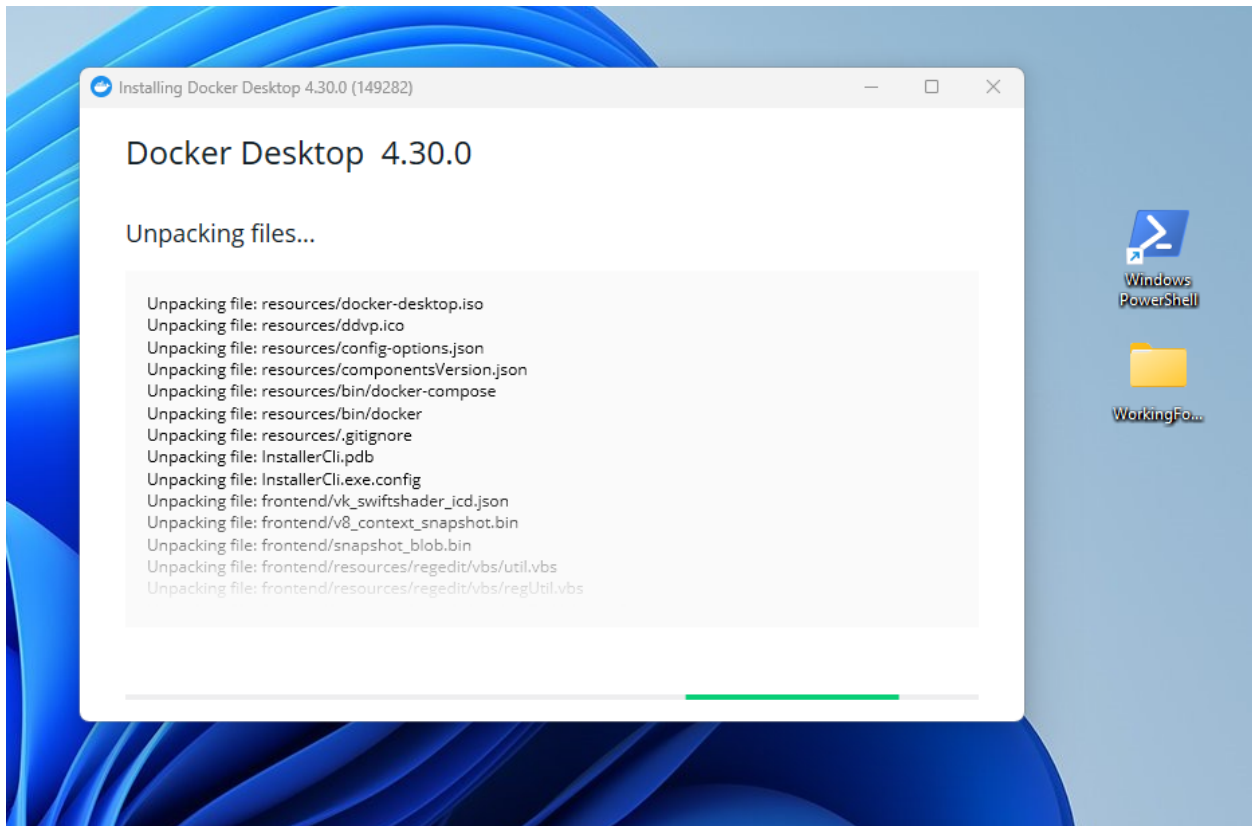


Fig. 14: Installing Docker Desktop

4. After installation, a Docker Desktop shortcut will appear on your desktop. The necessary components to run Maxwell-TD Docker images include Docker Desktop itself, Windows PowerShell for running CLI commands, and a designated working folder containing Docker images and user data (e.g., geometry information).
5. Launch Docker Desktop by double-clicking the Docker Desktop shortcut on your desktop.
6. Upon first launch, Docker Desktop may prompt you to accept terms and conditions. Agree to proceed with using the application.
7. You can continue without logging into Docker Desktop if prompted, and optionally skip any survey presented during initial setup.

Currently, the local repository in Docker Desktop is empty. To proceed, we must upload the compiled image for Maxwell-TD. This requires using the PowerShell console, as the GUI does not support managing local Docker images directly and defaults to searching online repositories for Docker images.

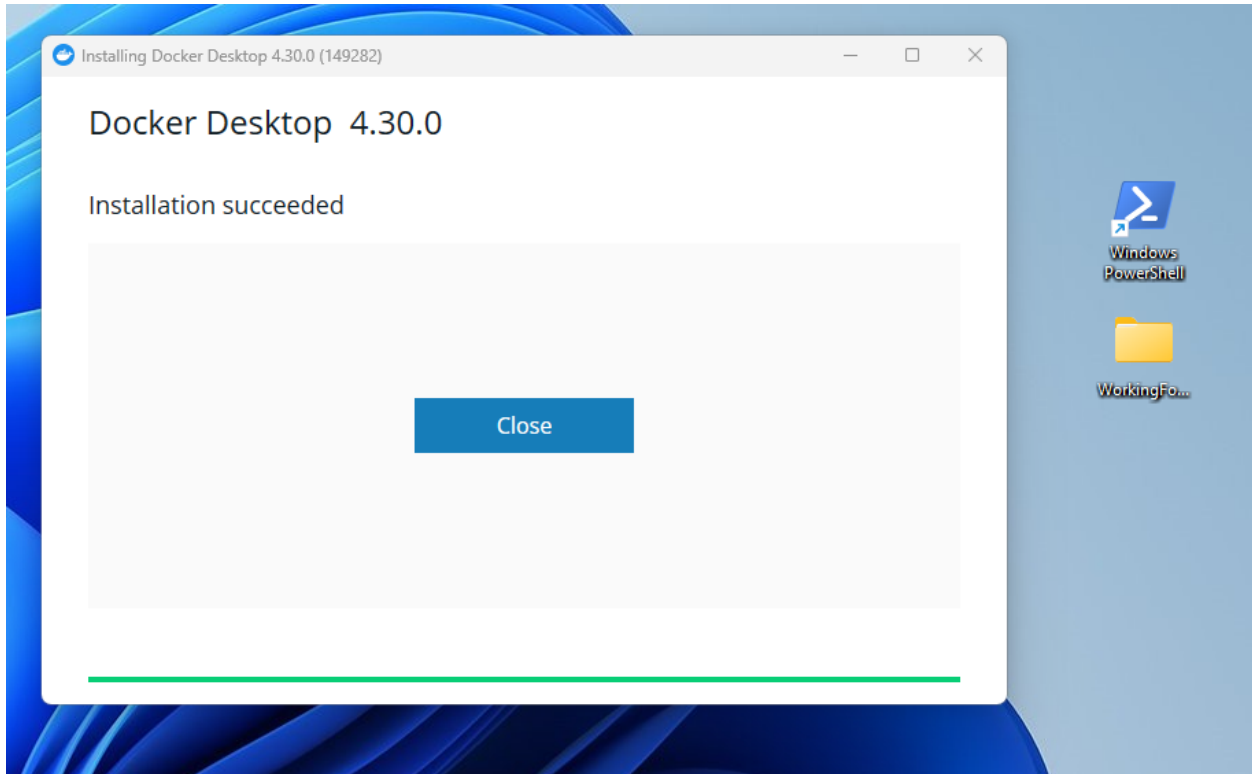


Fig. 15: Finish installing Docker Desktop

Import Docker Image

Next, we need to mount the zipped Docker image onto the Windows system. Since this action cannot be performed using the Docker Desktop GUI, we'll need to use the console terminal instead.

In the designated working folder, you will find a zipped Docker image file and an "examples" folder containing user-defined data.

To load the Docker image, execute the command `docker load --input <PATH to zipped docker image>`. This process will take some time, and you can verify its completion by using `docker images` in the console.

The loaded Docker image will also be visible both in Docker Desktop.

Mounting User data and running Docker image

Finally, we are ready to run the Docker image. You can use a PowerShell script with administrative privileges to execute the Docker image. We have prepared a script file named `runDocker.ps1` that can be executed in the console terminal.

To run the script and start the Docker image, follow these steps:

1. **Open PowerShell as Administrator**
2. **Navigate to Script Directory:** Use the `cd` command to navigate to the directory where `runDocker.ps1` is located.
3. **Execute the Script:**
 - Run the script by typing `.\runDocker.ps1` and press `Enter`.

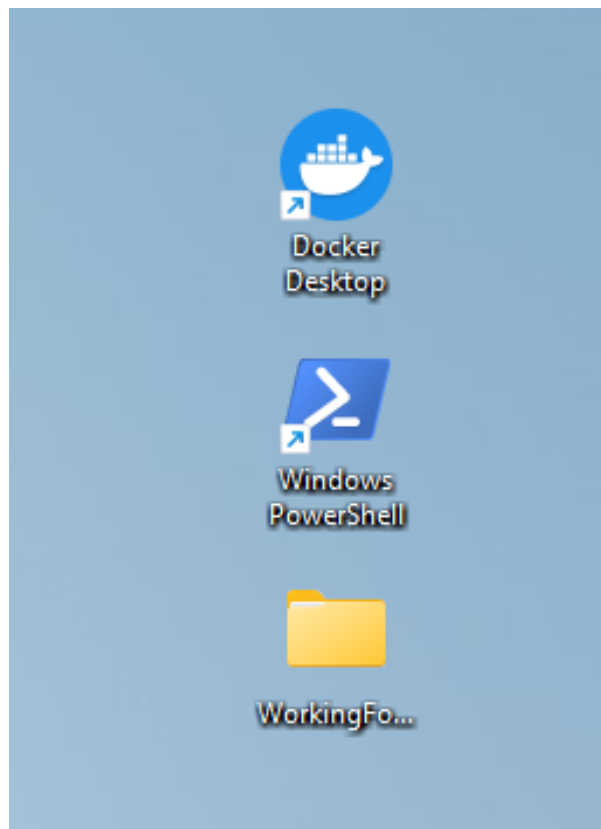


Fig. 16: All the data/folders/apps needed to run Maxwell-TD Docker image

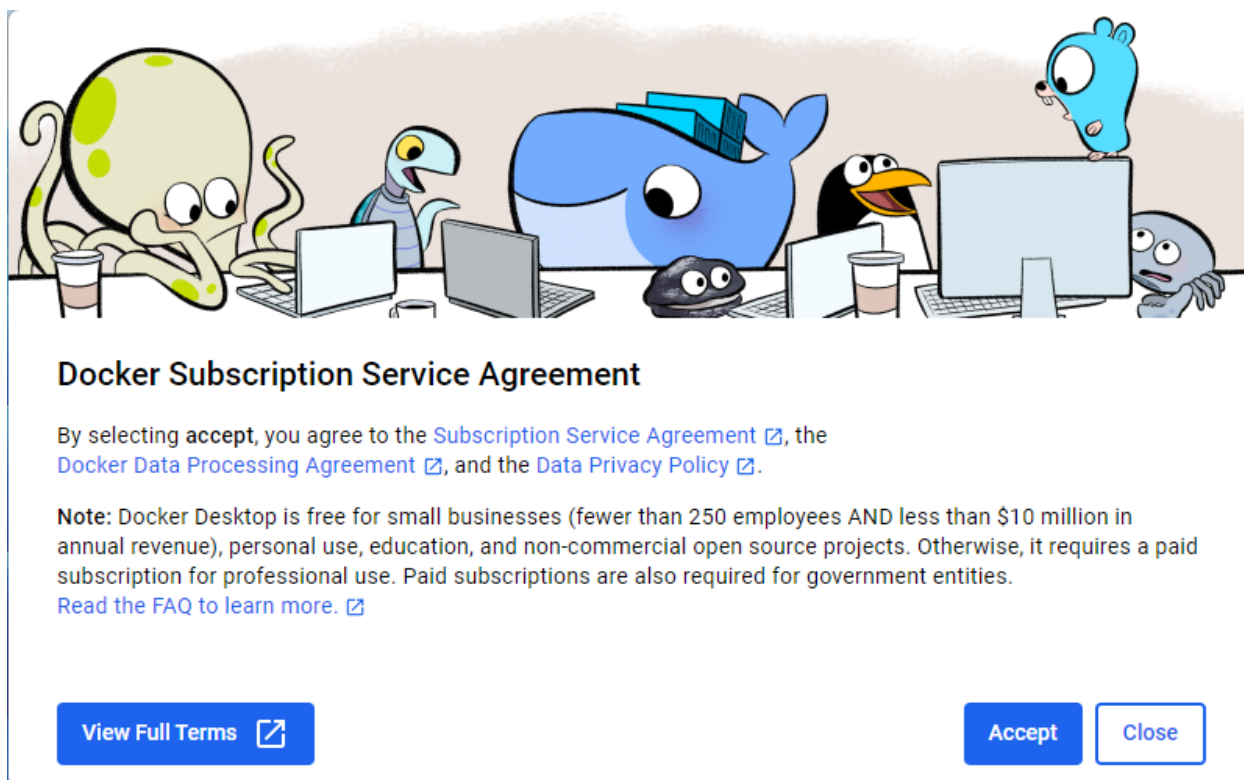


Fig. 17: Accept the terms and conditions

- The script will mount the folder containing user-defined data into the Docker Server environment.

Note:

- When running the Docker image, any data saved onto the mounted filesystem will persist. However, other modifications, such as custom installations of packages, will be temporary and will disappear after you exit the Docker session.
- Treat the Docker image as a saved state of a customized environment.

Let's break down the lines in the PowerShell script to understand the necessary steps for running the Docker image. This will clarify what is needed for execution.

- **\$imageName:**

This variable defines the name of the Docker image that you want to run. In this case, it is set to `maxwell_td_image`. Users should not change this variable if they intend to run Maxwell-TD.

- **\$containerName:**

This variable specifies the name of the container that will run the Docker image. You can run multiple containers from the same image concurrently, so each running instance needs a unique name. Users can customize this variable to suit their naming conventions.

- **\$hostVolumePath:**

This variable holds the path to the working folder on the Windows system. This folder contains essential files such as geometry files for simulation models and scripts for setting simulation parameters. Users must update this path to point to their specific working folder.

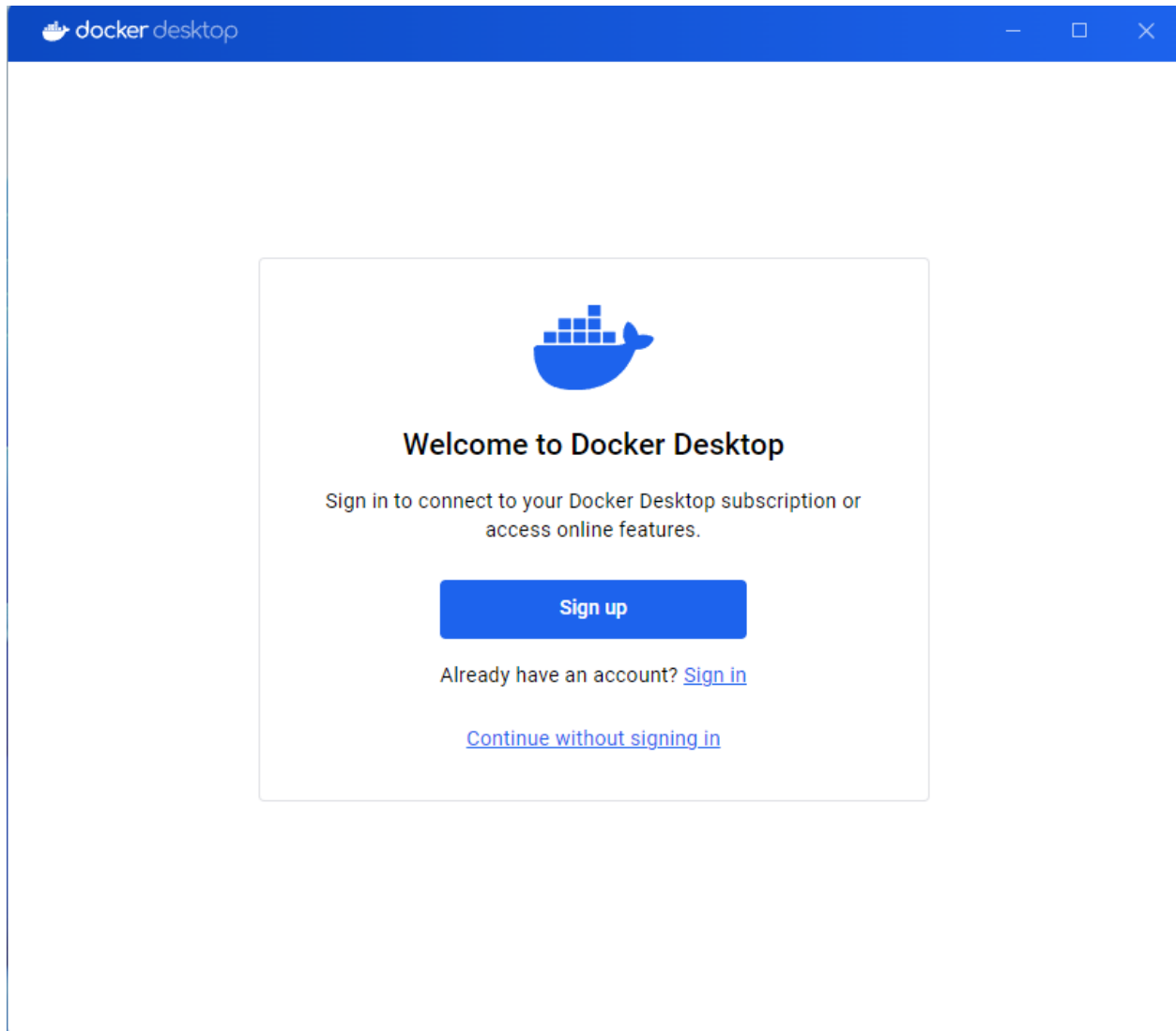


Fig. 18: Continue without logging in

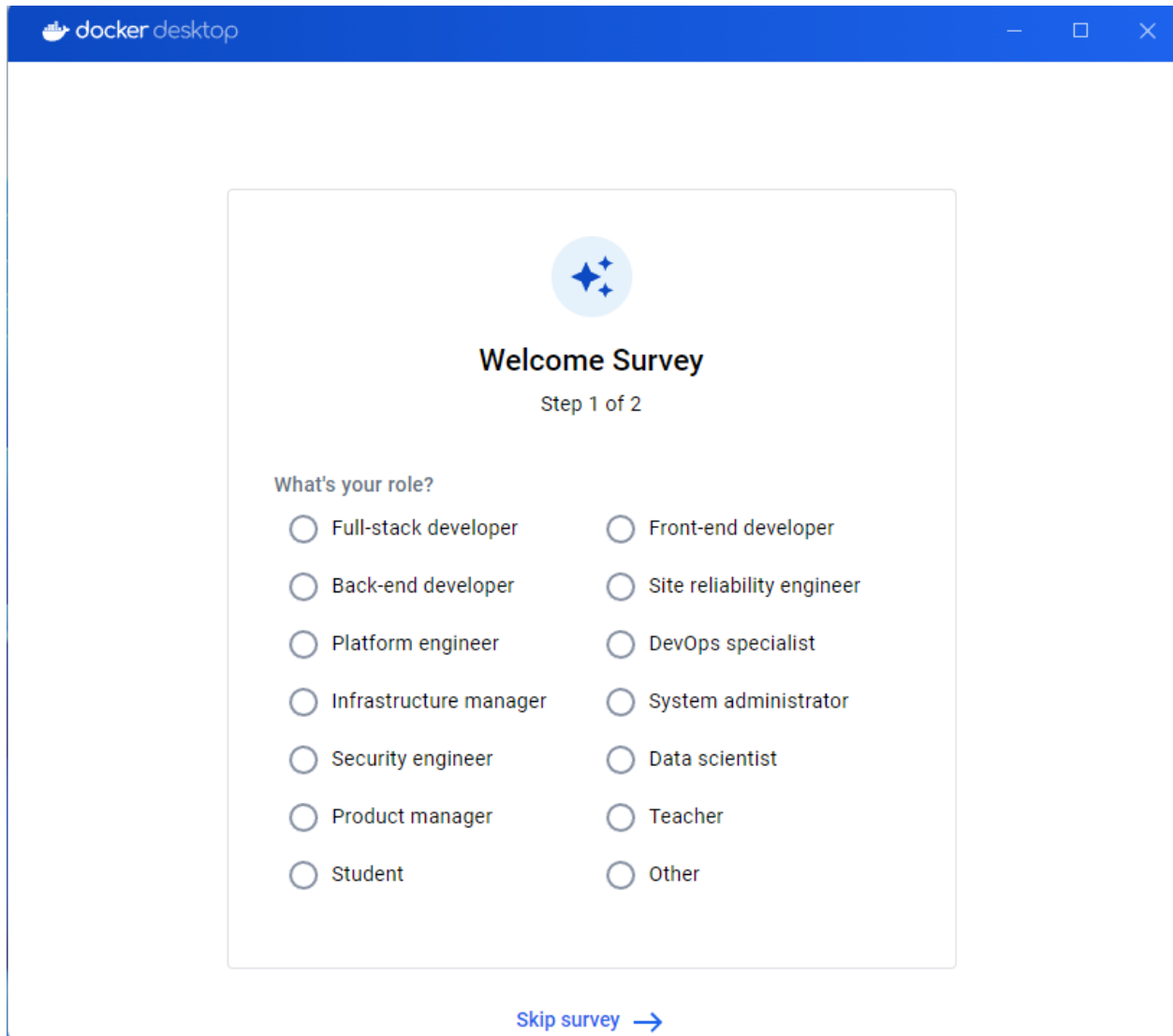


Fig. 19: Skip Survey

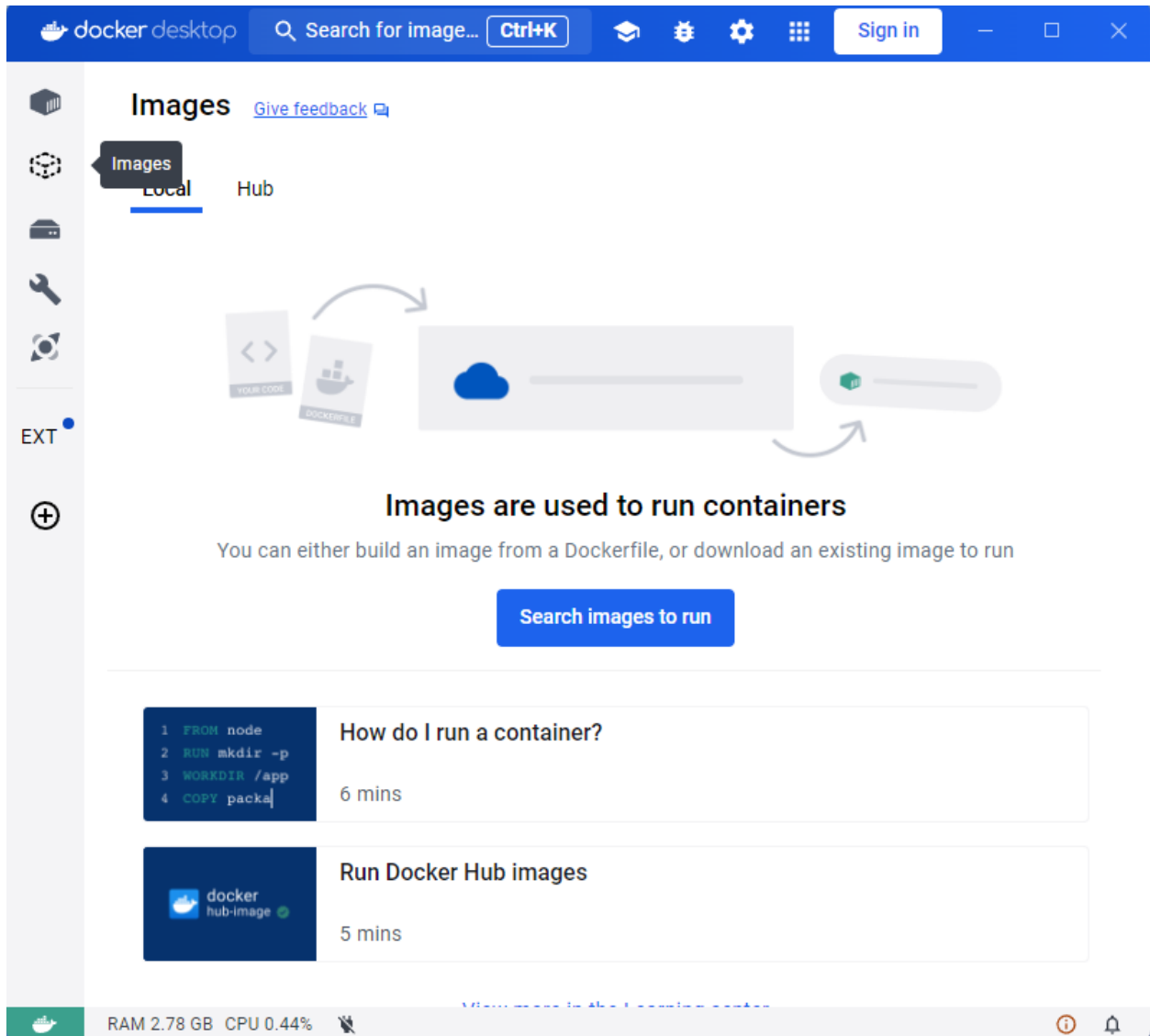


Fig. 20: Image local Repository

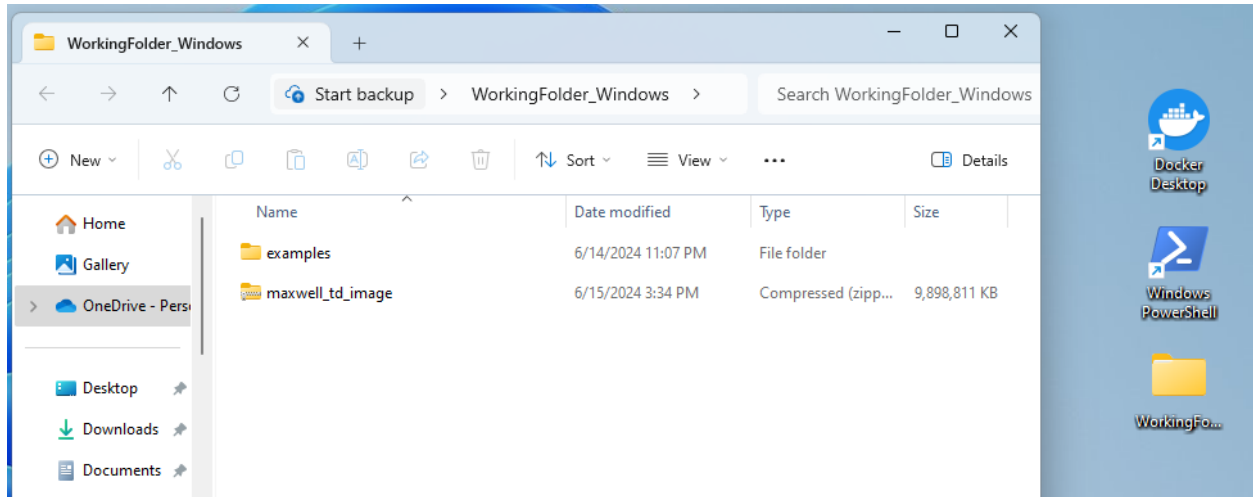


Fig. 21: Location of Maxwell-TD Docker Image (Zipped)

```

Administrator: Windows PowerShell

(base) PS C:\WINDOWS\system32> docker image ls
REPOSITORY TAG IMAGE ID CREATED SIZE
(base) PS C:\WINDOWS\system32> docker load --input C:\Users\Administrator\Desktop\WorkingFolder_Windows\maxwell_td_image.zip
5498e8c22f69: Loading layer [=====] 80.41MB/80.41MB
4cd407952594: Loading layer [=====] 11.07MB/11.07MB
022bf74291b2: Loading layer [=====] 156MB/156MB
eeb5315df33c: Loading layer [=====] 3.072kB/3.072kB
e942261d196e: Loading layer [=====] 18.94kB/18.94kB
421c5b38d6e0: Loading layer [=====] 2.031GB/2.031GB
520e0f301880: Loading layer [=====] 268.8kB/268.8kB
700fe921ad1f: Loading layer [=====] 10.75kB/10.75kB
b0dfaf1ca5c5: Loading layer [=====] 5.12kB/5.12kB
4e356886a648: Loading layer [=====] 4.948GB/4.948GB
bbdba4451161: Loading layer [=====] 395.8kB/395.8kB
f7c974b3317e: Loading layer [=====] 1.883GB/1.883GB
7cdc3703fc0: Loading layer [=====] 53.73MB/53.73MB
0dd976b078ca: Loading layer [=====] 15.96MB/15.96MB
d5814e9f0977: Loading layer [=====] 1.536kB/1.536kB
a73c20e8c67d: Loading layer [=====] 592.4MB/592.4MB
f17b654b2ba4: Loading layer [=====] 349.4MB/349.4MB
2689dc19db01: Loading layer [=====] 15.49MB/15.49MB
Loaded image: maxwell_td_image:latest
(base) PS C:\WINDOWS\system32> docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
maxwell_td_image latest df8498f00a2b 6 hours ago 10.1GB
(base) PS C:\WINDOWS\system32>

```

Fig. 22: Load and check Docker Image

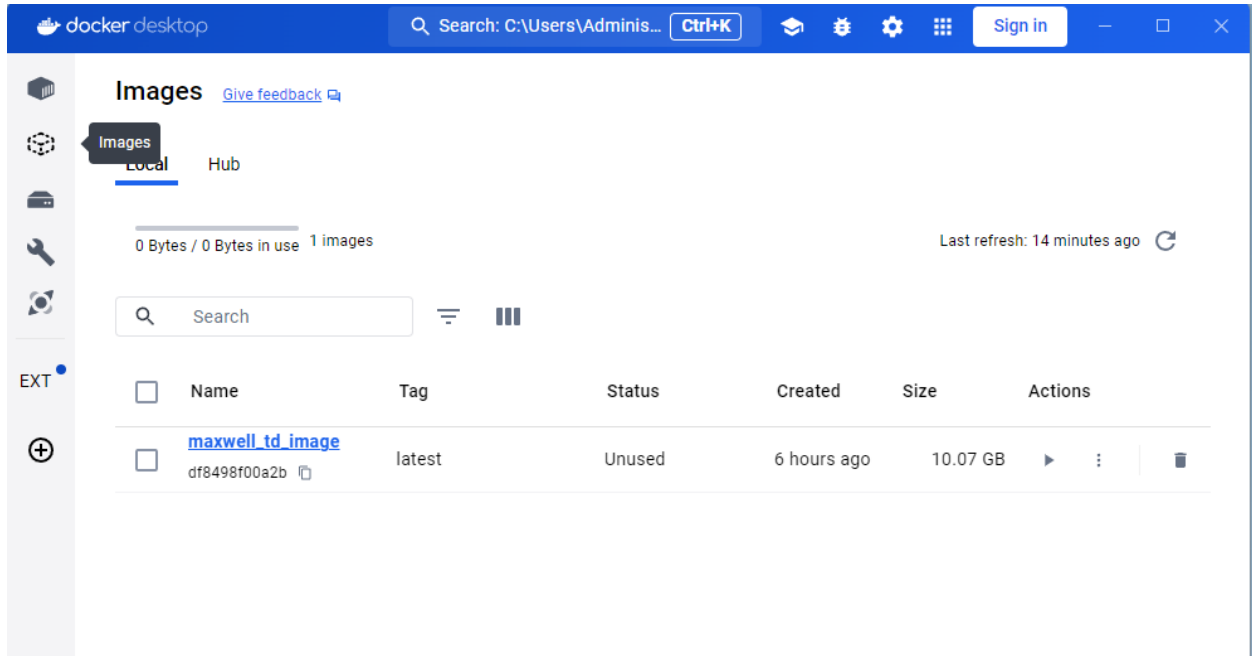


Fig. 23: Loaded Docker Image can be seen in Docker Desktop

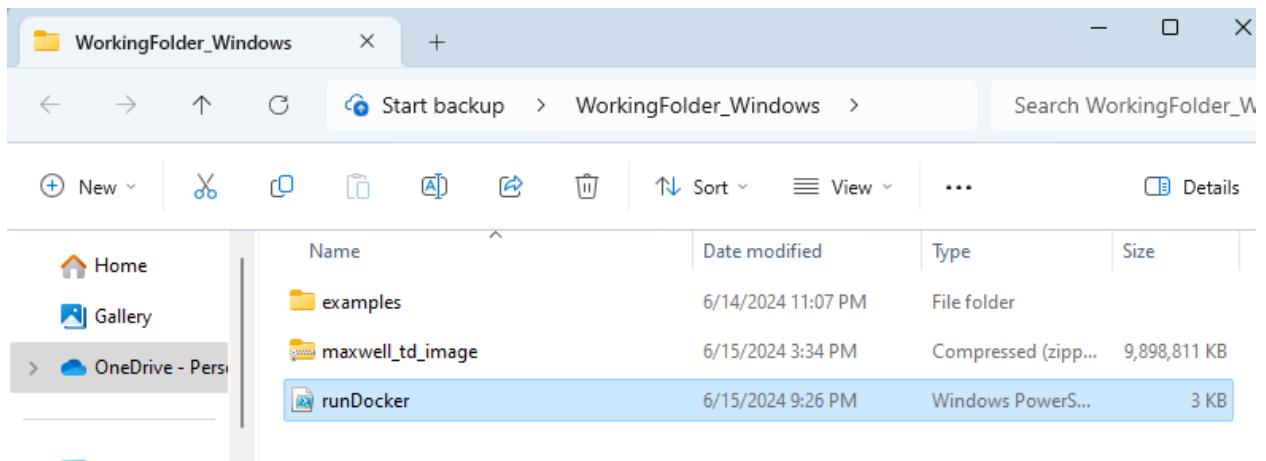


Fig. 24: Use PowerShell Script to run Docker Image

- **\$containerVolumePath:**

This variable defines the path within the Docker container (running on Linux environment) where the host volume (**\$hostVolumePath**) will be mounted. In this script, it is set to `/dgttd/model/`. It is recommended that users do not change this variable unless necessary, to ensure compatibility with the Docker image's expected mount point.

```

36 # Set variables
37 $imageName = "maxwell_td_image"
38 $containerName = "maxwell_td_container"
39 $hostVolumePath = "C:\Users\Administrator\Desktop\WorkingFolder_windows\examples"
40 $containerVolumePath = "/dgttd/model"
41 # $hostRunScriptPath = "C:\Users\Administrator\Desktop\Docker\runCUDA.sh"
42 # $containerRunScriptPath = "/dgttd/runCUDA.sh"

```

Fig. 25: User-defined parameters to change location of User data

Next, the script proceeds to execute several commands to display the paths and names. It also checks the CUDA version installed on the Windows device. Afterward, it runs the Docker command `docker run --rm --gpus all -it --name ${containerName} -v ${hostVolumePath}:${containerVolumePath} ${imageName}`.

Explanation of Docker Command Flags

- `docker run`: This command starts a new Docker container.
- `--rm`: Automatically removes the container when it exits. Useful for temporary containers.
- `--gpus all`: Enables access to all GPUs in the container.
- `-it`: Allocates a pseudo-TTY and keeps stdin open. Allows interactive terminal access.
- `--name ${containerName}`: Assigns a name to the container instance based on the value stored in the `$containerName` variable.
- `-v ${hostVolumePath}:${containerVolumePath}`: Mounts a volume from the host machine (Windows) into the Docker container. `${hostVolumePath}` is the path on the host system, and `${containerVolumePath}` is the path inside the container where the volume will be mounted.
- `${imageName}`: Specifies the Docker image to use for creating the container.

Additional Information

- The `--rm` flag ensures that the container is automatically removed after it stops running, helping to manage resources efficiently.
- `--gpus all` allows the Docker container to access all available GPUs on the host machine, which is crucial for applications that require GPU acceleration.
- `-it` enables interactive mode with a terminal session, which is useful for interacting directly with the container if needed.

This Docker command, when executed within the PowerShell script, sets up and runs the Docker container named `${containerName}` with GPU support, mounts necessary volumes, and utilizes the specified Docker image `${imageName}` for the Maxwell-TD application. Adjust `${containerName}`, `${hostVolumePath}`, `${containerVolumePath}`, and `${imageName}` as per your specific environment and requirements.

Note: When opening a Windows text file in a Unix environment, extra `\r` characters at the end of each line can cause errors such as `/bin/bash^M: bad interpreter: No such file or directory`. This is because Unix-based systems expect lines to end with `\n` (newline) instead of `\r\n` (carriage return followed by newline) used in Windows.

To fix this issue, you can convert the line endings from Windows format to Unix format using utilities like `dos2unix`, which removes the extra `\r` characters. Alternatively, you can use the included function `Remove-CarriageReturns`

```

46 # Display settings
47 Write-Host "Running Docker container with the following settings:"
48 Write-Host "Image Name: $imageName"
49 Write-Host "Container Name: $containerName"
50 Write-Host "Host Volume Path: $hostVolumePath"
51 Write-Host "Container Volume Path: $containerVolumePath"
52 Write-Host "Host Run Script Path: $hostRunScriptPath"
53 Write-Host "Container Run Script Path: $containerRunScriptPath"
54
55 # Check CUDA driver version on the host
56 Write-Host "Checking CUDA driver version on the host..."
57 $nvidiaSmiOutput = & nvidia-smi
58 $cudaVersion = $nvidiaSmiOutput | Select-String -Pattern "Driver Version" | ForEach-Object { $_ -replace ".*Driver Version:\s*", "" }
59 Write-Host "CUDA Driver Version: $cudaVersion"
60
61
62 # Construct the Docker run command
63 $dockerRunCommand = @"
64 docker run --rm --gpus all -it --name ${containerName} -v ${hostVolumePath}:${containerVolumePath} ${imageName}
65 "@
66
67 # Echo out the Docker run command for debugging
68 Write-Host "-----"
69 Write-Host "Docker run command:"
70 Write-Host $dockerRunCommand
71 Write-Host "-----"
72
73 # Run Docker container
74 Invoke-Expression nvidia-smi
75 Invoke-Expression $dockerRunCommand
76
77
78 # Check if the container started successfully
79 if ($LASTEXITCODE -ne 0) {
80 | Write-Host "Error: Failed to start Docker container" -ForegroundColor Red
81 } else {
82 | Write-Host "Docker container ended successfully" -ForegroundColor Green
83 }

```

Fig. 26: Commands to run Docker image

in `runDocker.ps1` to process the files. Once corrected, your script should execute correctly within Docker or any Unix-based environment without encountering interpreter errors.

To run the script, navigate to the directory where `runDocker.ps1` is located and execute `./runDocker.ps1`. The script will display the CUDA toolkit version and open a BASH shell in the Docker container with `maxwell_td_image`. This environment simulates a Linux environment.

How to use Maxwell-TD

Upon starting the Docker container, you will be in the `/dgttd/` directory, where the Discontinuous Galerkin Time-Domain (DGTD) code is located. The compiled code can be found in the `/dgttd/build/` directory, with the executables named `maxwelltd_CUDA_LTS_DOUBLEpre_FLOATpro` and `maxwelltd_CUDA_LTS_DOUBLEpre_FLOATpro-`. User data, mounted from the Windows filesystem, is available in the `/dgttd/model/` directory. Before running the simulation, the compiled executable must be copied to the simulation folder (`/dgttd/model/MaxwellTD_data/`). This can be done by executing the following command:

```
cp /dgttd/build/maxwelltd_CUDA_LTS_DOUBLEpre_FLOATpro* /dgttd/model/MaxwellTD_data/
```

Simulation files for the MaxwellTD model are located in the directory `/dgttd/model/MaxwellTD_data`. Additionally, we have included CST reference data for the same simulation model in the folder `CST_data`. A Python script named `compare_maxwelltd_CST.py` has been provided to facilitate extraction and comparison of data between the DGTD simulation and CST data.

To run the DGTD simulation, we can navigate to `/dgttd/model/MaxwellTD_data/` and run the shell script `CUDA_LTS_RUN.sh`.


```
1 # PowerShell script to run a Docker container with volume mounting
2
3 function Remove-CarriageReturns {
4     param (
5         [Parameter(Mandatory=$true)]
6         [string]$FilePath
7     )
8
9     # Check if the file exists
10    if (-Not (Test-Path -Path $FilePath)) {
11        Write-Error "The file '$FilePath' does not exist."
12        return
13    }
14
15    try {
16        # Read the content of the file
17        $fileContent = Get-Content -Raw -Path $FilePath
18
19        # Remove carriage return characters
20        $fileContent = $fileContent -replace "`r", ""
21
22        # Write the modified content back to the file
23        Set-Content -Path $FilePath -Value $fileContent
24
25        Write-Output "Carriage return characters removed from '$FilePath'."
26    } catch {
27        Write-Error "An error occurred: $_"
28    }
29 }
30
31 # Process Shell scripts if created in windows
32 # Remove-CarriageReturns -FilePath "./runCUDA.sh"
33 # Remove-CarriageReturns -FilePath "$hostVolumePath/CUDA_LTS_RUN.sh"
34
```

Fig. 27: Miscellaneous function to treat textfiles from Windows System (Optional)

```

root@6b377de16a24: /dgt
(base) PS C:\Users\Administrator\Desktop\WorkingFolder_Windows> .\runDocker.ps1
Running Docker container with the following settings:
Image Name: maxwell_td_image
Container Name: maxwell_td_container
Host Volume Path: C:\Users\Administrator\Desktop\WorkingFolder_Windows\examples
Container Volume Path: /dgt/model
Host Run Script Path:
Container Run Script Path:
Checking CUDA driver version on the host...
CUDA Driver Version: 551.78      CUDA Version: 12.4
-----
Docker run command:
docker run --rm --gpus all -it --name maxwell_td_container -v C:\Users\Administrator\Desktop\WorkingFolder_Windows\examples:/dgt/model maxwell_td_image
-----
Sat Jun 15 21:33:28 2024
-----
+-----+-----+-----+-----+-----+-----+
| NVIDIA-SMI 551.78      | Driver Version: 551.78      | CUDA Version: 12.4      |
+-----+-----+-----+-----+-----+-----+
| GPU   Name   Perf      | TCC/WDDM | Bus-Id   | Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf      | Pwr:Usage/Cap |          | Memory-Usage | GPU-Util  Compute M. |
|-----+-----+-----+-----+-----+-----+
| 0     NVIDIA GeForce RTX 3060 | WDDM     | 00000000:01:00:00  | On     | 2%      Default  N/A |
| 0%    39C   P8          | 16W / 170W  |          | 570MiB / 12288MiB |           |           |
+-----+-----+-----+-----+-----+-----+
-----
Processes:
-----
GPU   GI   CI   PID  Type  Process name
-----
0     N/A  N/A   1680  C+G  C:\Windows\System32\dw...
0     N/A  N/A   1812  C+G  ...nt.CBS_cw5n1h2txyewy\SearchHost.exe
0     N/A  N/A   3664  C+G  ...2txyewy\StartMenuExperienceHost.exe
0     N/A  N/A   8820  C+G  ...siveControlPanel\SystemSettings.exe
0     N/A  N/A   10600 C+G  C:\Windows\explorer.exe
0     N/A  N/A   12832 C+G  ...t\SelfServicePlugin\SelfService.exe
0     N/A  N/A   13052 C+G  ...t\Docker\Frontend\Docker Desktop.exe
0     N/A  N/A   13652 C+G  ...5n1h2txyewy\ShellExperienceHost.exe
0     N/A  N/A   14036 C+G  ...ekyb3d8bbwe\PhoneExperienceHost.exe
0     N/A  N/A   15128 C+G  ...on\126.0.2592.56\msedgebview2.exe
0     N/A  N/A   15992 C+G  ...les\Microsoft OneDrive\OneDrive.exe
0     N/A  N/A   16724 C+G  ...rosoft\Edge\Application\msedge.exe
0     N/A  N/A   18184 C+G  ...es (x86)\Dropbox\Client\Dropbox.exe
0     N/A  N/A   20096 C+G  ...on\126.0.2592.56\msedgebview2.exe
0     N/A  N/A   22688 C+G  ...Programs\Microsoft VS Code\Code.exe
-----
=====
== CUDA ==
=====
CUDA Version 12.4.0

Container image Copyright (c) 2016-2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.

This container image and its contents are governed by the NVIDIA Deep Learning Container License.
By pulling and using the container, you accept the terms and conditions of this license:
https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license

A copy of this license is made available in this container at /NGC-DL-CONTAINER-LICENSE for your convenience.

root@6b377de16a24: /dgt#
    
```

Fig. 28: Running PowerShell Script

```

root@6b377de16a24: /dgt/model
root@6b377de16a24: /dgt# ls
CMakeLists.txt Dockerfile GPU_N README.md build computes config dgt-d-performance.hpp dgt.cpp fem lib mod1 requirements.txt utils
root@6b377de16a24: /dgt# cd model
root@6b377de16a24: /dgt/model# ls
CMakeLists.txt mod1 compare_maxwelltd_CST.py
root@6b377de16a24: /dgt/model# cp /dgt/build/maxwelltd_CUDA_LTS_DOUBLEpre_FLOATpro* /dgt/model/MaxwellTD_data/
root@6b377de16a24: /dgt/model#
    
```

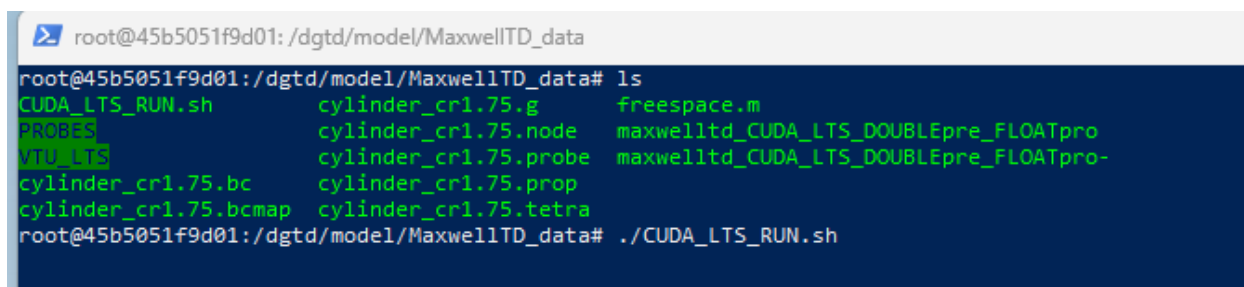
Fig. 29: Local Directory in Docker Image

```

root@6b377de16a24: /dgt/model/MaxwellTD_data
root@6b377de16a24: /dgt/model# ls
CST_data MaxwellTD_data compare_maxwelltd_CST.py
root@6b377de16a24: /dgt/model# cd MaxwellTD_data/
root@6b377de16a24: /dgt/model/MaxwellTD_data# ls
CUDA_LTS_RUN.sh cylinder_cr1.75.bc cylinder_cr1.75.node cylinder_cr1.75.tetra maxwelltd_CUDA_LTS_DOUBLEpre_FLOATpro-
PROBE1 cylinder_cr1.75.bcmap cylinder_cr1.75.probe freespace.m
PROBE2 cylinder_cr1.75.g cylinder_cr1.75.prop maxwelltd_CUDA_LTS_DOUBLEpre_FLOATpro
    
```

Fig. 30: Simulation folder

```
./CUDA_LTS_RUN.sh
```



```

root@45b5051f9d01: /dgttd/model/MaxwellTD_data
root@45b5051f9d01:/dgttd/model/MaxwellTD_data# ls
CUDA_LTS_RUN.sh      cylinder_cr1.75.g      freespace.m
PROBES               cylinder_cr1.75.node   maxwelltd_CUDA_LTS_DOUBLEpre_FLOATpro
VOLUME               cylinder_cr1.75.probe  maxwelltd_CUDA_LTS_DOUBLEpre_FLOATpro-
cylinder_cr1.75.bc   cylinder_cr1.75.prop
cylinder_cr1.75.bcm  cylinder_cr1.75.tetra
root@45b5051f9d01:/dgttd/model/MaxwellTD_data# ./CUDA_LTS_RUN.sh

```

Fig. 31: Running Simulation script

This will initiate the DGTD simulation.

Following the completion of the simulation, you can execute the Python script `compare_maxwelltd_CST.py` to compare the results with those obtained from the CST simulation.

```
python3 compare_maxwelltd_CST.py
```

Common issues

Incompatible GPU drivers/toolkit

It's essential to verify that your Windows device has the correct CUDA toolkit installed to run Maxwell-TD. You can check the CUDA toolkit version by using 'nvidia-smi' in the console terminal. The minimum required version to run Maxwell-TD is CUDA toolkit 12.4.

If you need to update your CUDA toolkit, you can visit the [NVIDIA website \(https://developer.nvidia.com/cuda-toolkit\)](https://developer.nvidia.com/cuda-toolkit). Click on 'NVIDIA CUDA Toolkit Download' to access the latest version. Alternatively, if you require an earlier version, you can navigate to 'Archive of Previous CUDA Releases' on the NVIDIA website to find the specific CUDA toolkit version you need.

Follow the download and installation instructions provided on the NVIDIA website to update or install the CUDA toolkit on your Windows device. This ensures that Maxwell-TD and other CUDA-dependent applications can function correctly.

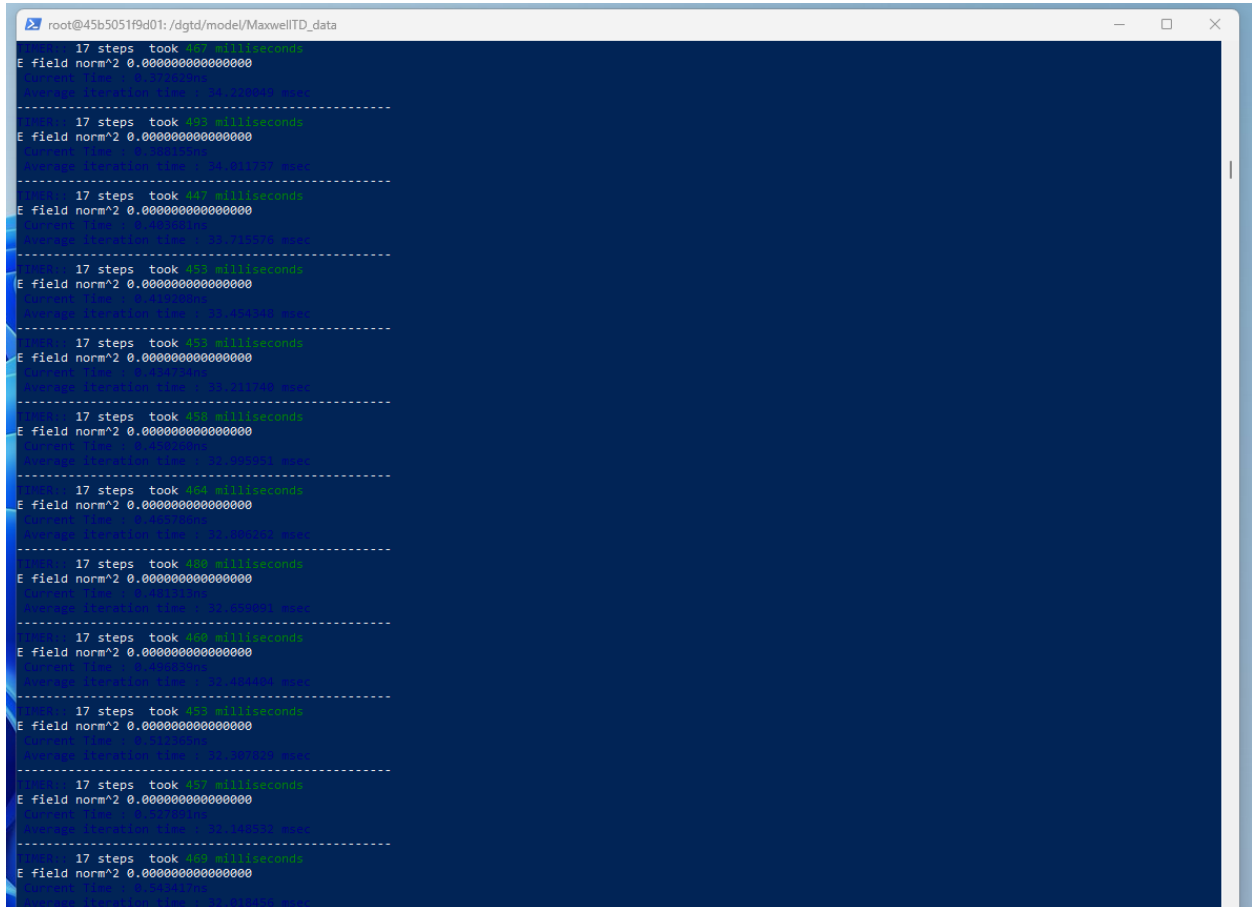
Windows Version

Make sure your Windows 10 or 11 system meets the following requirements:

- **WSL Version:** Ensure WSL version 1.1.3.0 or newer.
- **Windows 11:** 64-bit Home or Pro version 21H2 or later, or Enterprise or Education version 21H2 or later.
- **Windows 10:** 64-bit Home or Pro version 22H2 (build 19045) or later, or Enterprise or Education version 22H2 (build 19045) or later. The minimum supported version is Home or Pro 21H2 (build 19044) or later, or Enterprise or Education 21H2 (build 19044) or later.

Note:

- Docker Desktop on Windows is supported only on versions of Windows that are still actively serviced by Microsoft. It is not compatible with Windows Server versions like Windows Server 2019 or Windows Server 2022.



```
root@45b5051f9d01: /dgttd/model/MaxwellTD_data
17 steps took 447 milliseconds
E field norm^2 0.0000000000000000
-----
17 steps took 448 milliseconds
E field norm^2 0.0000000000000000
-----
17 steps took 447 milliseconds
E field norm^2 0.0000000000000000
-----
17 steps took 453 milliseconds
E field norm^2 0.0000000000000000
-----
17 steps took 450 milliseconds
E field norm^2 0.0000000000000000
-----
17 steps took 450 milliseconds
E field norm^2 0.0000000000000000
-----
17 steps took 448 milliseconds
E field norm^2 0.0000000000000000
-----
17 steps took 450 milliseconds
E field norm^2 0.0000000000000000
-----
17 steps took 453 milliseconds
E field norm^2 0.0000000000000000
-----
17 steps took 457 milliseconds
E field norm^2 0.0000000000000000
-----
17 steps took 450 milliseconds
E field norm^2 0.0000000000000000
```

Fig. 32: Simulating running

```

root@45b5051f9d01: /dgttd/model/MaxwellTD_data
Final Pade Point 27completed
  Process : took 1118 milliseconds
Final Pade Point 28completed
  Process : took 1223 milliseconds
Final Pade Point 29completed
  Process : took 1229 milliseconds
Final Pade Point 30completed
  Process : took 1203 milliseconds
Final Pade Point 31completed
  Process : took 1207 milliseconds
Final Pade Point 32completed
  Process : took 1215 milliseconds
Final Pade Point 33completed
  Process : took 1188 milliseconds
Final Pade Point 34completed
  Process : took 1209 milliseconds
Final Pade Point 35completed
  Process : took 1204 milliseconds
Final Pade Point 36completed
  Process : took 1195 milliseconds
Final Pade Point 37completed
  Process : took 1235 milliseconds
Final Pade Point 38completed
  Process : took 1197 milliseconds
Final Pade Point 39completed
  Process : took 1221 milliseconds
Final Pade Point 40completed
  Process : took 1213 milliseconds
Final Pade Point 41completed
  Process : took 1216 milliseconds
Final Pade Point 42completed
  Process : took 1223 milliseconds
Final Pade Point 43completed
  Process : took 1208 milliseconds
Final Pade Point 44completed
  Process : took 1205 milliseconds
Final Pade Point 45completed
  Process : took 1177 milliseconds
Final Pade Point 46completed
  Process : took 1179 milliseconds
Final Pade Point 47completed
  Process : took 1212 milliseconds
Final Pade Point 48completed
  Process : took 1187 milliseconds
Final Pade Point 49completed
  Process : took 1191 milliseconds
Final Pade Point 50completed
  Process : took 1183 milliseconds
Final Pade Point 51completed
  Process : took 1200 milliseconds
Final Pade Point 52completed
  Process : took 1209 milliseconds
Final Pade Point 53completed
  Process : took 1195 milliseconds
Final Pade Point 54completed
  Process : took 1188 milliseconds
Final Pade Point 55completed
  Process : took 1175 milliseconds
  Process : took 67268 milliseconds
root@45b5051f9d01: /dgttd/model/MaxwellTD_data#

```

Fig. 33: End of simulation

```

root@45b5051f9d01: /dgttd/model
root@45b5051f9d01: /dgttd/model/MaxwellTD_data# ls
CUDA_LTS_RUN.sh  cylinder_cr1.75.bc  cylinder_cr1.75.node  cylinder_cr1.75.tetra  maxwelltd_CUDA_LTS_DOUBLEpre_FLOATpro-
PROBES          cylinder_cr1.75.bcmap  cylinder_cr1.75.probe  freespace.m
CPU_LTS         cylinder_cr1.75.g  cylinder_cr1.75.prop  maxwelltd_CUDA_LTS_DOUBLEpre_FLOATpro
root@45b5051f9d01: /dgttd/model/MaxwellTD_data# cd ..
root@45b5051f9d01: /dgttd/model# ls
compare_maxwelltd_CST.py
root@45b5051f9d01: /dgttd/model# python3 compare_maxwelltd_CST.py

```

Fig. 34: Running post-processing PYTHON script

```

Administrator: Windows PowerShell
(base) PS C:\WINDOWS\system32> nvidia-smi
Sat Jun 15 21:29:31 2024
-----
NVIDIA-SMI 551.78              Driver Version: 551.78          CUDA Version: 12.4
-----
GPU  Name          TCC/WDDM   Bus-Id      Disp.A   Volatile Uncorr. ECC
  Fan  Temp   Perf          Pwr:Usage/Cap     Memory-Usage   GPU-Util  Compute M.
                                           Memory-Usage   GPU-Util  Compute M.
-----
  0   0   39C    P8          16W / 170W      649MiB / 12288MiB      2%          Default
                                           649MiB / 12288MiB      2%          Default
-----

Processes:
GPU  GI  CI  PID  Type  Process name                      GPU Memory
  ID  ID  ID                                Usage
-----
  0   N/A  N/A  1680  C+G  C:\Windows\System32\dwm.exe        N/A
  0   N/A  N/A  1812  C+G  ...nt.CBS_cw5n1h2txyewy\SearchHost.exe  N/A
  0   N/A  N/A  3664  C+G  ...2txyewy\StartMenuExperienceHost.exe  N/A
  0   N/A  N/A  8820  C+G  ...siveControlPanel\SystemSettings.exe  N/A
  0   N/A  N/A  10600  C+G  C:\Windows\explorer.exe           N/A
  0   N/A  N/A  12832  C+G  ...t\SelfServicePlugin\SelfService.exe  N/A
  0   N/A  N/A  13052  C+G  ...\Docker\Frontend\Docker Desktop.exe  N/A
  0   N/A  N/A  13652  C+G  ...5n1h2txyewy\ShellExperienceHost.exe  N/A
  0   N/A  N/A  14036  C+G  ...ekyb3d8bbwe\PhoneExperienceHost.exe  N/A
  0   N/A  N/A  15128  C+G  ...on\126.0.2592.56\msedgewebview2.exe  N/A
  0   N/A  N/A  15992  C+G  ...les\Microsoft OneDrive\OneDrive.exe  N/A
  0   N/A  N/A  16724  C+G  ...crosoft\Edge\Application\msedge.exe  N/A
  0   N/A  N/A  17060  C+G  ...oogle\Chrome\Application\chrome.exe  N/A
  0   N/A  N/A  18184  C+G  ...es (x86)\Dropbox\Client\Dropbox.exe  N/A
  0   N/A  N/A  20096  C+G  ...on\126.0.2592.56\msedgewebview2.exe  N/A
  0   N/A  N/A  22688  C+G  ...Programs\Microsoft VS Code\Code.exe  N/A
-----
(base) PS C:\WINDOWS\system32>

```

Fig. 35: Check CUDA toolkit version

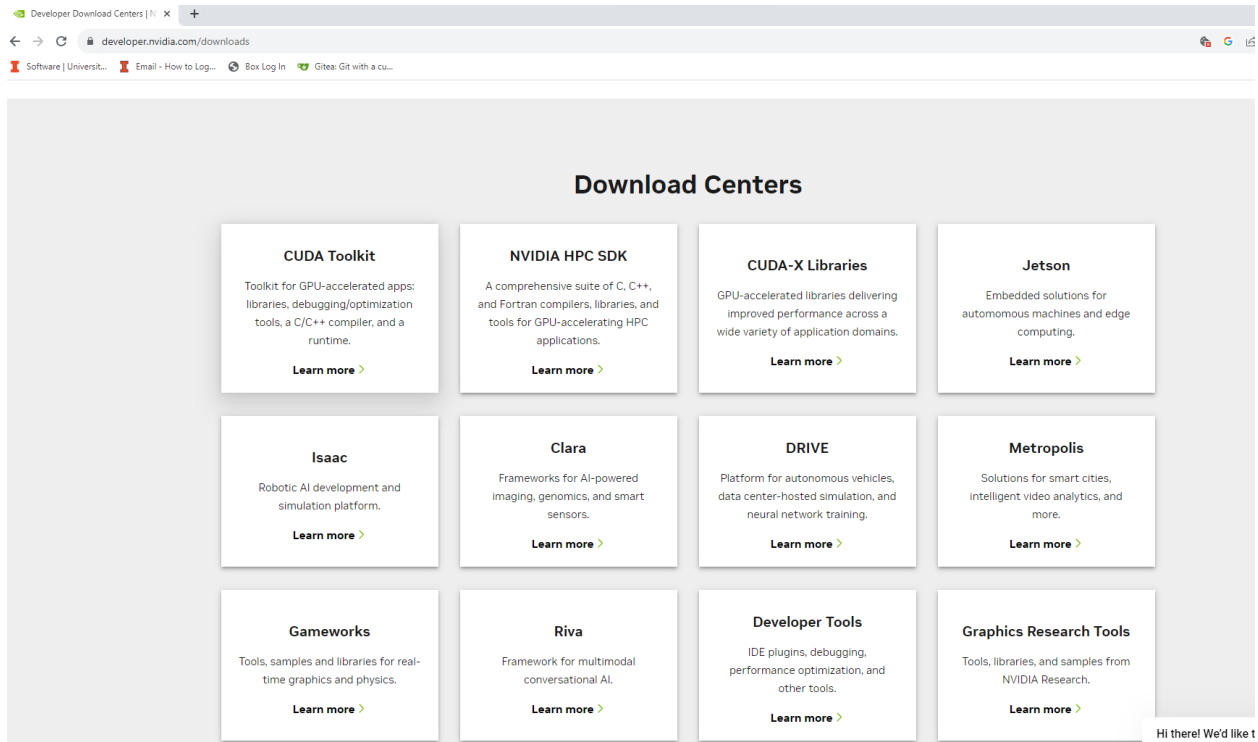


Fig. 36: NVIDIA website to download CUDA toolkit

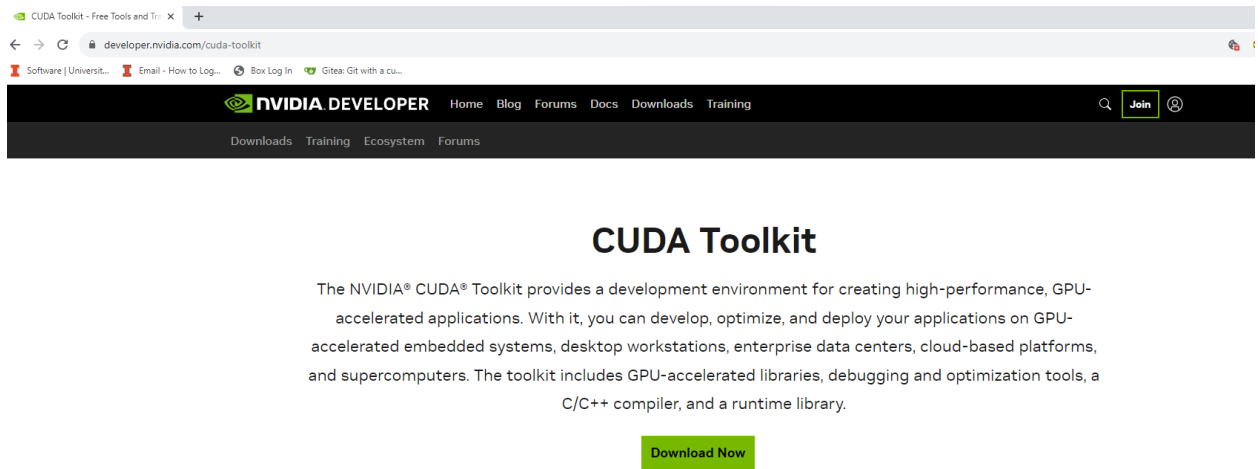


Fig. 37: NVIDIA CUDA toolkit download

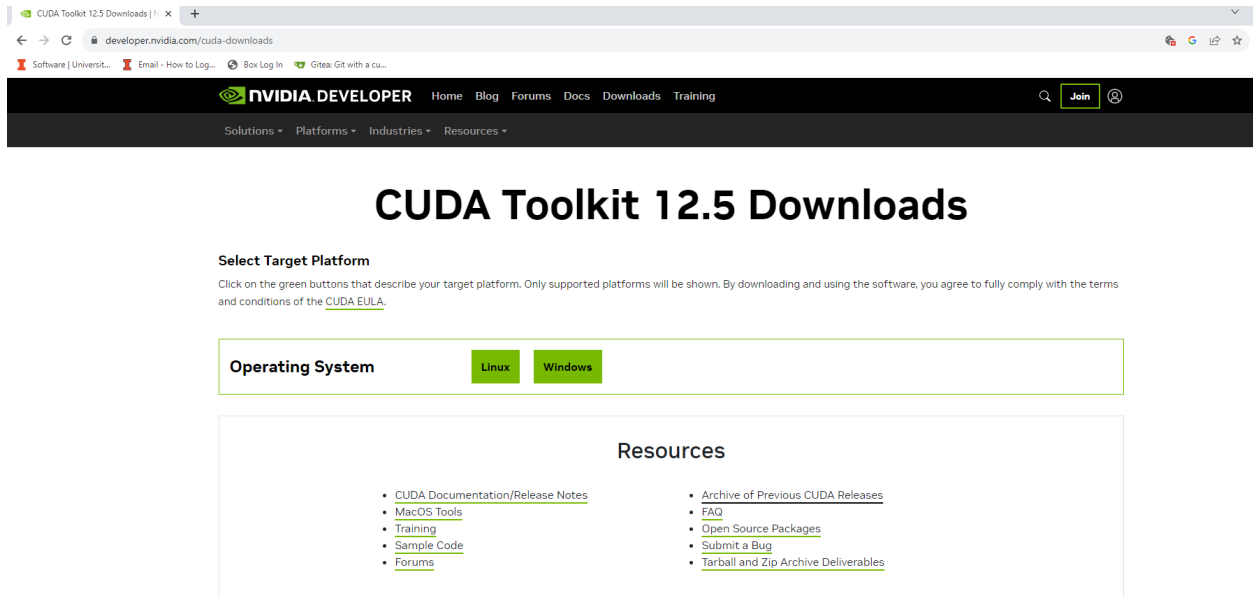


Fig. 38: NVIDIA CUDA toolkit (Latest version)

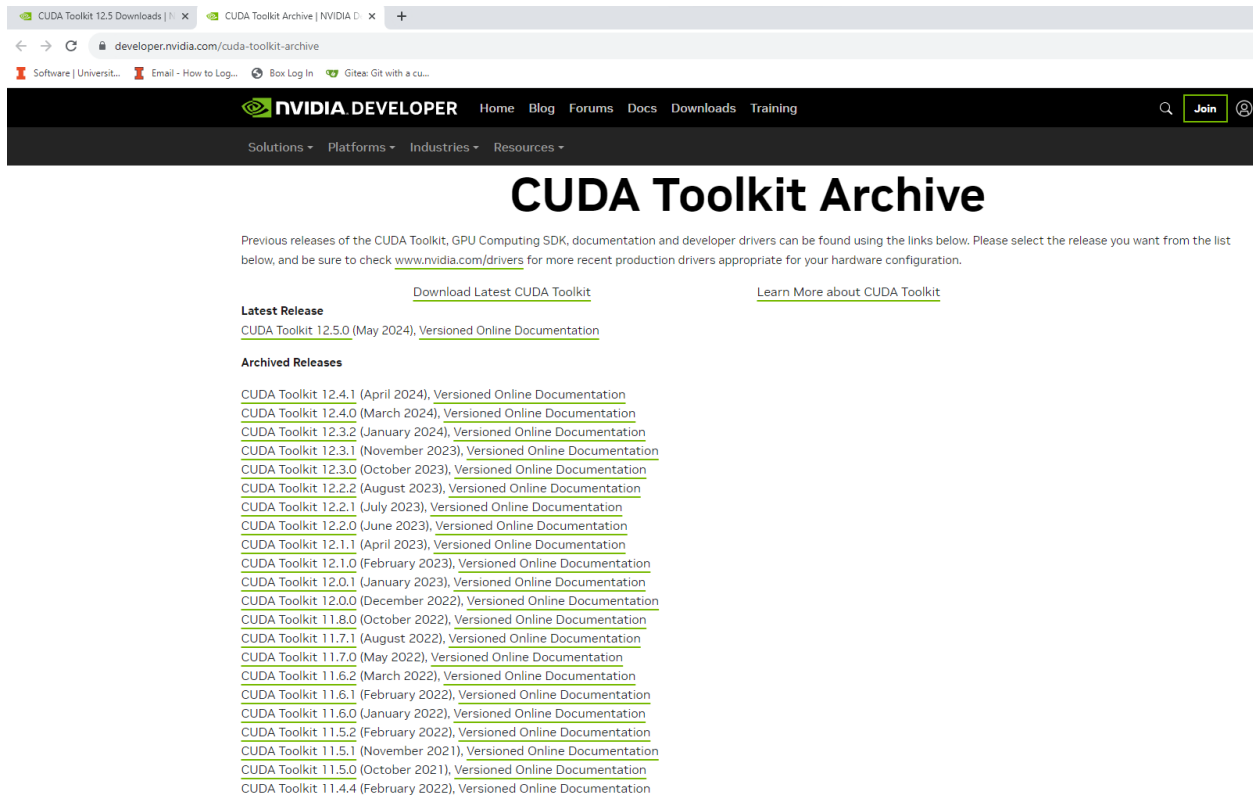


Fig. 39: NVIDIA CUDA toolkit (Earlier versions)

- Containers and images created with Docker Desktop are shared across all user accounts on the same machine, as they use a common VM for building and running containers. However, sharing containers and images between user accounts is not supported when using the Docker Desktop WSL 2 backend.

Sufficient Memory to upload image

Make sure your system has adequate storage space for the image. The current image requires 10GB for storage while zipped. Once extracted, it occupies an additional 10GB on your system, although it remains hidden from regular users. You can view these hidden images and containers using Docker Desktop.

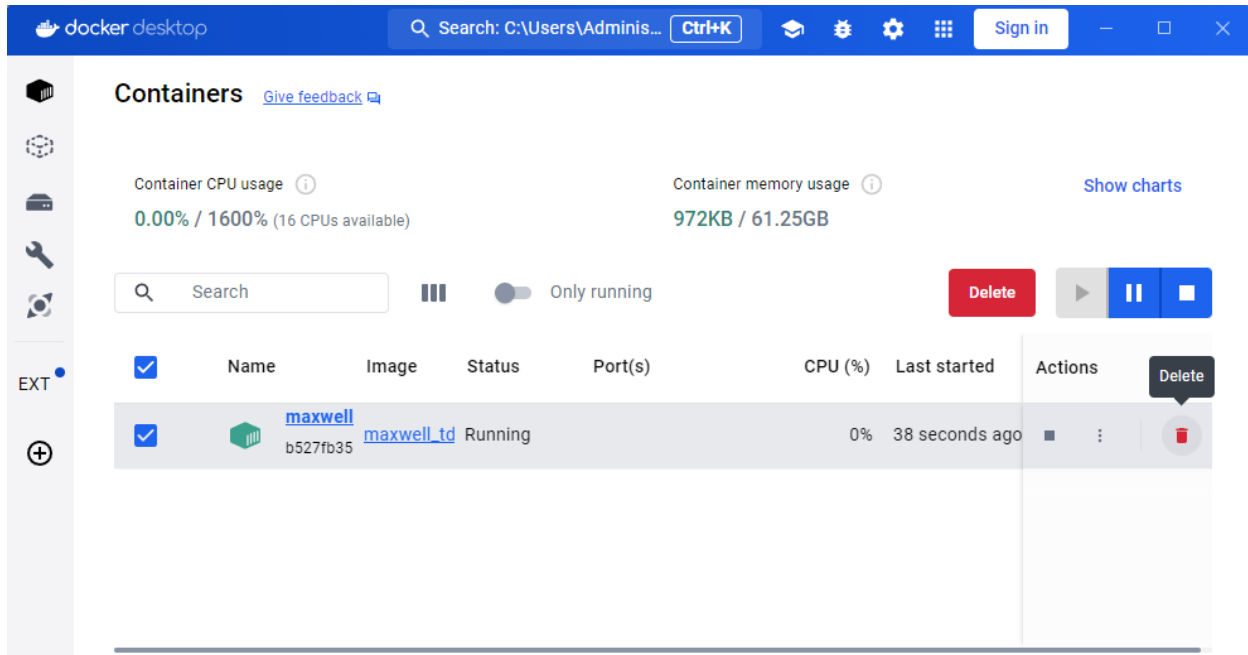


Fig. 40: View all containers or image on Docker Desktop

Users can remove any unwanted images or containers using the 'delete' button.

If necessary, resort to command-line operations to reclaim system resources. Execute these commands in a PowerShell console with administrative privileges.

```
# Purging All Unused or Dangling Resources
docker system prune

# Remove stopped containers and all unused images
docker system prune -a

# Removing Docker Images

## Remove Specific Images
# List all images (including intermediate layers)
docker images -a

# Remove specific image(s) using their ID or tag
```

(continues on next page)

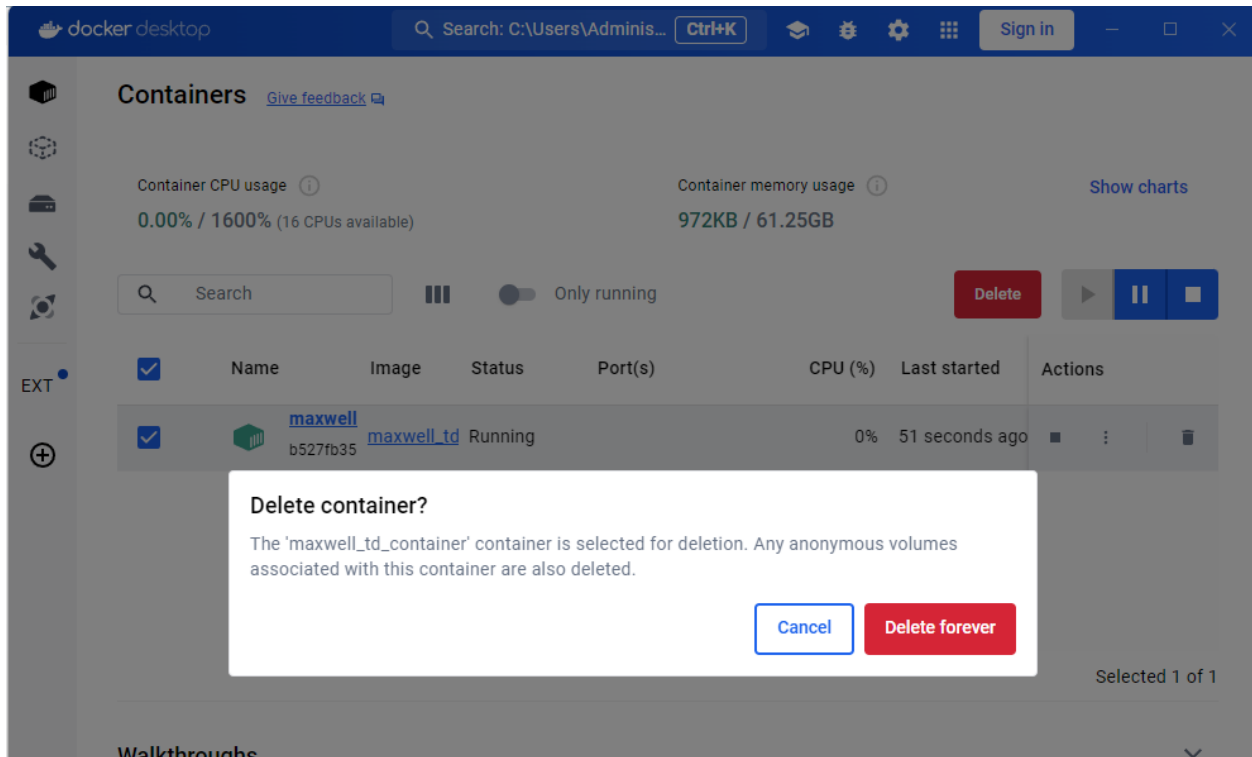


Fig. 41: Delete if not needed to free up space

(continued from previous page)

```

docker rmi <image_id_or_name>

## Remove Dangling Images
# List dangling images
docker images -f dangling=true

# Remove dangling images
docker image prune

## Remove Images Matching a Pattern
# List images matching a pattern (using grep)
docker images -a | grep "pattern"

# Remove images matching a pattern using awk and xargs
docker images -a | grep "pattern" | awk '{print $1":"$2}' | xargs docker rmi

## Remove All Images
# Remove all Docker images
docker rmi $(docker images -a -q)

# Removing Containers

## Remove Specific Containers
# List all containers (including stopped ones)
docker ps -a

```

(continues on next page)

(continued from previous page)

```

# Remove specific container(s)
docker rm <container_id_or_name>

## Remove All Exited Containers
# List all exited containers
docker ps -a -f status=exited

# Remove all exited containers
docker rm $(docker ps -a -f status=exited -q)

## Remove Containers Matching a Pattern
# List containers matching a pattern (using grep)
docker ps -a | grep "pattern"

# Remove containers matching a pattern using awk and xargs
docker ps -a | grep "pattern" | awk '{print $1}' | xargs docker rm

## Stop and Remove All Containers
# Stop all containers
docker stop $(docker ps -a -q)

# Remove all containers
docker rm $(docker ps -a -q)

```

BIOS settings

To enable Docker server virtualization on Windows, it's crucial to ensure that virtualization is enabled in your computer's BIOS settings. This setting can only be modified by restarting your machine and accessing the BIOS configuration. Virtualization support in BIOS allows Windows to utilize Docker server features effectively. It's a necessary step to ensure Docker containers run efficiently on your system.

Change Simulation Parameters

To modify simulation parameters, you can edit the `CUDA_LTS_RUN.sh` shell script. This script contains flags or variables that control various aspects of the DGTD simulation. The script includes comments or descriptions to explain the variables' or flags' purpose.

```

echo "======"
echo "EXECUTABLE"
EXE="./maxwelltd_CUDA_LTS_DOUBLEpre_FLOATpro"
echo $EXE
echo "FileName"
echo "======"

# 0. [EXE]
# 1. Filename: Simulation Filename
# 2. Freq: Central Modulation Frequency (MHz)
# 3. Planewave Waveform (Excitation Type): (0)Time Harmonic (1)Gauss (2)Neumann (3)
↪Modulated Gauss;

```

(continues on next page)

(continued from previous page)

```

# 4. Port Waveform (0) Time Harmonic Pulse (1) Gaussian Pulse
# 5. Tdelay: Delay of excitation pulse (sec)
# 6. Tau: Width of pulse (sec)
# 7. FinalTime: Termination Time (sec)
# 8. Gamma (Penalty) --> 1 upwind , 0 central
# 9. VTU --> 0 (YES), 1 (NO)
#10. Surface Field Output BC Surfaces --> (0) Off (1) PEC only (2) FieldPlane only (3)
↳FieldPlane and PEC
#11. Surface Output Types --> (0) Field (1) Current (2) Field and Current
#12. Poly order --> (1) First Order (2) Second Order
#13. Free Space Comparison(A analytical): (0) True and (1) False
#14. SAMPLINGRATE : parameter of Pade approximation
#15. PADE : (0) Pade mode is Off (2) Pade mode is on

FILENAME=cylinder_cr1.75
FREQ=2650
PLANEWAVEFLAG=1
PORTFLAG=1
TD=2.5e-9
TAU=2.5e-10
FINALTIME=25e-8
GAMMA=0.025
VTU=1
SURFFIELDOUTBCSURFS=0
SURFOUTTYPES=0
POLYORDER=2
ANALYTICAL=1
SAMPLINGRATE=12.5
PADE=2

rm *.log
rm *.vtu
rm AnalyticalIncidentField*
rm Probes_*
rm *.TD*
rm -r ./TimeDomainVoltages
$EXE $FILENAME $FREQ $PLANEWAVEFLAG $PORTFLAG $TD $TAU $FINALTIME $GAMMA $VTU
↳$SURFFIELDOUTBCSURFS $SURFOUTTYPES $POLYORDER $ANALYTICAL $SAMPLINGRATE $PADE | tee -a
↳compute.log
rm -r ./VTU_LTS
mkdir ./VTU_LTS
mv *.vtu ./VTU_LTS
rm -r ./PROBES
mkdir ./PROBES
mv *.csv ./PROBES
mv *.TD* ./PROBES
mv *.log ./PROBES

```