**Instructions for Setting Up a Trusted Execution Environment with Intel SGX and Gramine Shielded Containers (GSC) on Ubuntu**

***Install Intel SGX SDK***

# Install the required tools to build Intel SGX SDK

sudo apt-get install build-essential ocaml ocamlbuild automake autoconf libtool wget python-is-python3 libssl-dev git cmake perl

# Install additional required tools and the latest Intel SGX SDK installer to build the Intel SGX

sudo apt-get install libssl-dev libcurl4-openssl-dev protobuf-compiler libprotobuf-dev debhelper cmake reprepro unzip pkgconf libboost-dev libboost-system-dev libboost-thread-dev lsb-release libsystemd0

(sudo apt-get install protobuf-c-compiler libprotobuf-c-dev lsb-release

# Note: The above three packages were installed with apt and seemed to have no effect on the ability to run.)

# Download the source code and prepare the submodules and prebuilt binaries

sudo git clone https://github.com/intel/linux-sgx.git

cd linux-sgx

sudo make preparation

# Navigate to the toolset directory specific to your Ubuntu version

cd external/toolset/ubuntu20.04

# Copy the mitigation tools corresponding to the current OS distribution from external/toolset/ubuntu20.04 to /usr/local/bin

# Ensure they have execute permission

pwd

sudo cp -r $(pwd)/linux-sgx/external/toolset/ubuntu20.04/\* /usr/local/bin

# Check the presence of build tools

which ar as ld objcopy objdump ranlib

• which ar as Id objcopy 
objdump ranlib 
/usr/local/bin/ar 
/usr/local/bin/as 
/usr/local/bin/ld 
/usr/local/bin/objcopy 
/usr/local/bin/objdump 
/usr/local/bin/ranlib 

cd

# Update package lists

sudo apt-get update

# Install make if not installed

sudo apt-get install make

# Navigate back to the linux-sgx directory

cd linux-sgx

# Build Intel SGX SDK with debug information

sudo make sdk DEBUG=1

# This step takes a longer time to complete

# Clean the files generated by previous builds

sudo make clean

# Build Intel SGX SDK installer with debug information kept in the tools and libraries

sudo make sdk\_install\_pkg DEBUG=1

# This step takes a longer time to complete

# After building, the output should be similar to the following:

# Generated SDK installer: ./linux/installer/bin/sgx\_linux\_x64\_sdk\_[version].bin

Generated sdk installer: ./1inux/insta11er/bin/sgx_1inux_x64 sdk 
2.21.1øø.1.bin 

# Install required tool to use Intel SGX SDK

sudo apt-get install build-essential python-is-python3

# Navigate to the installer directory

cd linux/installer/bin

# Run the SDK installer

sudo ./sgx\_linux\_x64\_sdk\_[version].bin

# Note: Replace [version] with the actual version number you have, for example, 2.21.100.1

# When prompted to install in the current directory, it is recommended to select "no" and then input /opt/intel/ to install the SGX SDK under the /opt/intel/ directory.

# Set up the needed environment variables as indicated in the guide

source /home/mtnh/ttnux-sgx/linux/instatter/btn/sgxsdk/environment 
source /home/minh/linux-sgx/ltnux/installer/b 
in/sgxsdk/environment 

# At this point, the SDK is installed

***Install Intel SGX PSW***

# To start installing the PSW, if you encounter errors with the following commands:

# sudo make deb\_psw\_pkg

# or

# sudo make deb\_psw\_pkg DEBUG=1

# Follow the installation guide from Zhihu for PSW installation

# Reference: [ubuntu20.4 intel-sgx环境配置 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/560110720)

dpkg-shlibdeps: error: cannot find library libsgx_urts.so.2 needed by debian/1ibsgx-qe3-10gic/usr/1ib/x86_64-1inux-gnu/1ibsg 
(ELF format: 'elf64-x86-64' abi: •euløe3eeøeøeøee•; RPATH: 
dpkg-shlibdeps: error: cannot continue due to the error above 
Note: libraries are not searched in other binary packages that do not have any shlibs or symbols file. 
To help dpkg-shlibdeps find private libraries, you might need to use -1. 
dh_shlibdeps: dpkg-shlibdeps -Tdebian/1ibsgx-qe3-10gic.substvars -1/usr/1ib64 --ignore-missing-info debian/1ibsgx-qe3 
-logic/usr/1ib/x86_64-1inux-gnu/1ibsgx_qe3_10gic.so returned exit code 2 
dh shlibdeps: error: Aborting due to earlier error 
make[3]: *** [debian/rules:ll: Error 2 
make[3]: Leaving directory '/home/azureuser/1inux-sgx/externa1/dcap_source/QuoteGeneration/insta11er/1inux/deb/1ibsgx-qe3-10 
gic/1ibsgx-qe3-10gic-1.18. Iøø.l' 
make[2]: *** [debian/ru1es:8: binary] Error 2 
make[2]: Leaving directory '/home/azureuser/1inux-sgx/externa1/dcap_source/QuoteGeneration/insta11er/1inux/deb/1ibsgx-qe3-10 
gic/1ibsgx-qe3-10gic-1.18. Iøø.l' 
dpkg-buildpackage: 
error: debian/rules binary subprocess returned exit status 2 
make[l]: *** [Makefile :142: deb_sgx_qe3_10gic_pkg] Error 2 
make[l]: Leaving directory '/home/azureuser/1inux-sgx/externa1/dcap_source/QuoteGeneration' 
make: *** (Makefile:166: deb libsgx qe3 logic I Error 2 

# In the linux-sgx directory, run the following commands:

echo 'deb [arch=amd64] https://download.01.org/intel-sgx/sgx\_repo/ubuntu focal main' | sudo tee /etc/apt/sources.list.d/intel-sgx.list

sudo wget https://download.01.org/intel-sgx/sgx\_repo/ubuntu/intel-sgx-deb.key

sudo apt-key add intel-sgx-deb.key

sudo apt-get update

# Now, the PSW should be installed

# Next, install the 3 services provided by SGX PSW: launch, EPID-based attestation, and algorithm agnostic attestation

sudo apt-get install libsgx-launch libsgx-urts

sudo apt-get install libsgx-epid libsgx-urts

sudo apt-get install libsgx-quote-ex libsgx-urts

sudo apt-get install libsgx-dcap-ql

# Testing the installation

# Navigate to the installation directory, then to the SampleCode/SampleEnclave directory

cd /opt/intel/sgxsdk/SampleCode/SampleEnclave

# Prepare the build environment

source /opt/intel/sgxsdk/environment

# Build the sample code

sudo make

# Run the sample application

./app

# The output should be similar to the following:

# Checksum(0x0x7ffda4d55720, 100) = 0xfffd4143

# Info: executing thread synchronization, please wait...

# Info: SampleEnclave successfully returned.

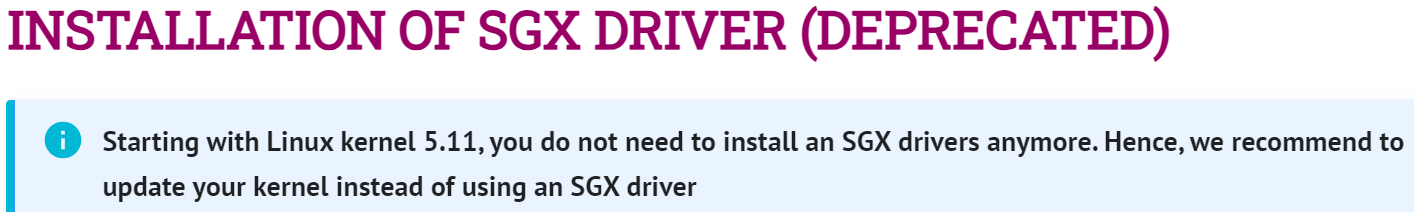
# Enter a character before exit ...

***Install Intel SGX Drivers***

Note: Manual installation of the Intel SGX Driver is not required for VMs created from the Azure Marketplace using Ubuntu & Windows images, as they come pre-installed with the necessary drivers.

Intel SGX drivers are included with both Ubuntu and Windows images available in the Azure Marketplace. This ensures that your VMs are ready for SGX-enabled workloads right after deployment.

To make sure that you have the latest version of the Intel SGX drivers, regularly check the [Intel SGX DCAP drivers list](<https://learn.microsoft.com/en-us/azure/confidential-computing/quick-create-marketplace>).



***Install GSC (Gramine Shielded Containers)***

## GSC - Gramine Shielded Containers Installation Guide

The `gsc` tool transforms a Docker image into a new image (called `gsc-<image-name>`) which includes the Gramine Library OS, manifest files, Intel SGX related information, and executes the application inside an Intel SGX enclave using the Gramine Library OS.

## Stages of Building Graminized SGX Docker Images

The build process from image `<image-name>` follows three main stages and produces an image named `gsc-<image-name>`:

- `gsc build-gramine` performs only the first stage,

- `gsc build` performs the first two stages, and finally

- `gsc sign-image` performs the last stage.

The transformation of a Docker image into a Gramine Shielded Container (GSC) image that can be executed inside an Intel SGX enclave involves three detailed stages:

1. Building Gramine.

The first stage builds Gramine from sources based on the provided configuration (see config.yaml) which includes the distribution (e.g., Ubuntu 20.04), Gramine repository, and the Intel SGX driver details. This stage can be skipped if gsc build uses a pre-built Gramine Docker image.

2. Graminizing the application image.

The second stage copies the important Gramine artifacts (e.g., the runtime and signer tool) from the first stage (or if the first stage was skipped, it pulls a prebuilt Docker image defined via the configuration file).

It then prepares image-specific variables such as the executable path and the library path, and scans the entire image to generate a list of trusted files. GSC excludes files and paths starting with /boot, /dev, .dockerenv, .dockerinit, /etc/mtab, /etc/rc, /proc, /sys, and /var, since checksums are required which either don’t exist or may vary across different deployment machines. GSC combines these variables and list of trusted files into a new manifest file.

In a last step the entrypoint is changed to launch the apploader.sh script which generates an Intel SGX token (only if needed, on non-FLC platforms) and starts the gramine-sgx loader.

Note that the generated image (gsc-<image-name>-unsigned) cannot successfully load an Intel SGX enclave, since essential files and the signature of the enclave are still missing (see next stage).

3. Signing the Intel SGX enclave.

The third stage uses Gramine’s signer tool to generate SIGSTRUCT files for SGX enclave initialization. This tool also generates an SGX-specific manifest file. The required signing key is provided by the user via the gsc sign-image command and copied into this Docker build stage. The generated image is called gsc-<image-name> and includes all necessary files to start an Intel SGX enclave.

## Run a GSC Example

# First, clone the GSC repository:

git clone https://github.com/gramineproject/gsc.git

cd gsc

# Install the necessary dependencies:

# Note: Replace 'docker-ce' with 'docker.io' if you encounter errors.

sudo apt-get install docker-ce python3 python3-pip

pip3 install docker jinja2 tomli tomli-w pyyaml

# Add your user to the Docker group:

sudo adduser $USER docker

# Create a configuration file:

cp config.yaml.template config.yaml

# Generate the signing key (if you don’t already have one):

openssl genrsa -3 -out enclave-key.pem 3072

# Pull the public Python image from Dockerhub (pin to the Debian Bullseye version):

sudo docker pull python:bullseye

# Install tomli and tomli\_w before next step:

sudo pip3 install tomli

sudo pip3 install tomli\_w

# Graminize the Python image using `gsc build`:

./gsc build --insecure-args python:bullseye test/generic.manifest

# Sign the graminized Docker image using `gsc sign-image`:

./gsc sign-image python:bullseye enclave-key.pem

# Retrieve SGX-related information from the graminized image using `gsc info-image`:

./gsc info-image gsc-python:bullseye

# Test the graminized Docker image:

# Note: Remove backslashes and adjust the device flag if needed. Change the `--device=/dev/sgx\_enclave` flag in the `docker run` command to match the device path of your installed version of the Intel SGX driver, if necessary.

# Run graminized Docker images:

# docker run [OPTIONS] gsc-<IMAGE-NAME> [<ARGUMENTS>]

sudo docker run --device=/dev/sgx\_enclave \

-v /var/run/aesmd/aesm.socket:/var/run/aesmd/aesm.socket \

gsc-python:bullseye -c 'print("HelloWorld!")'

# For debugging, start an interactive Bash session in the graminized Docker image:

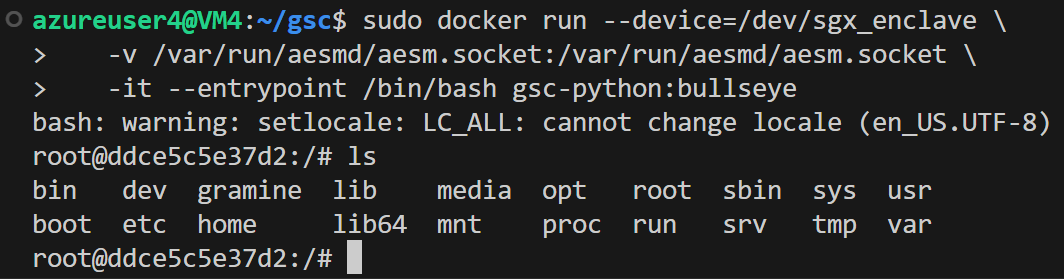
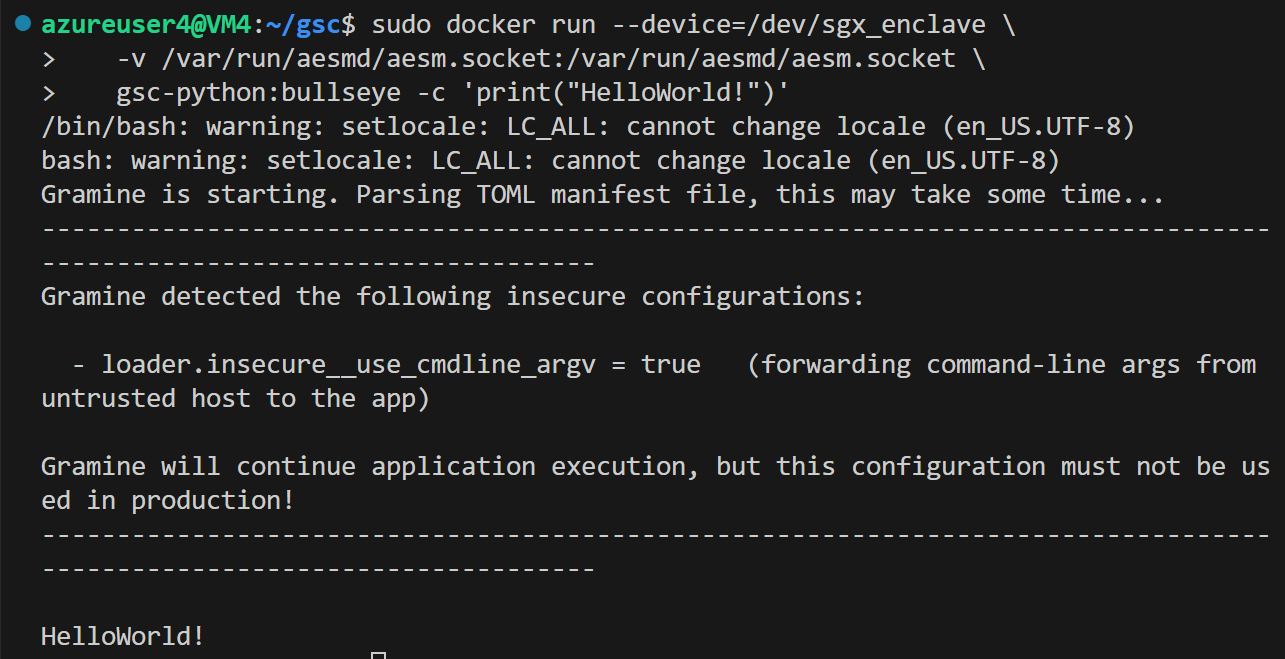
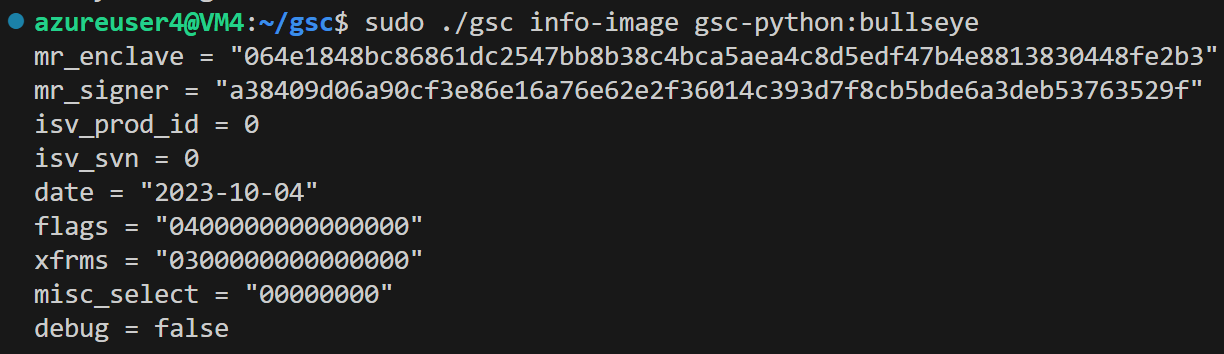
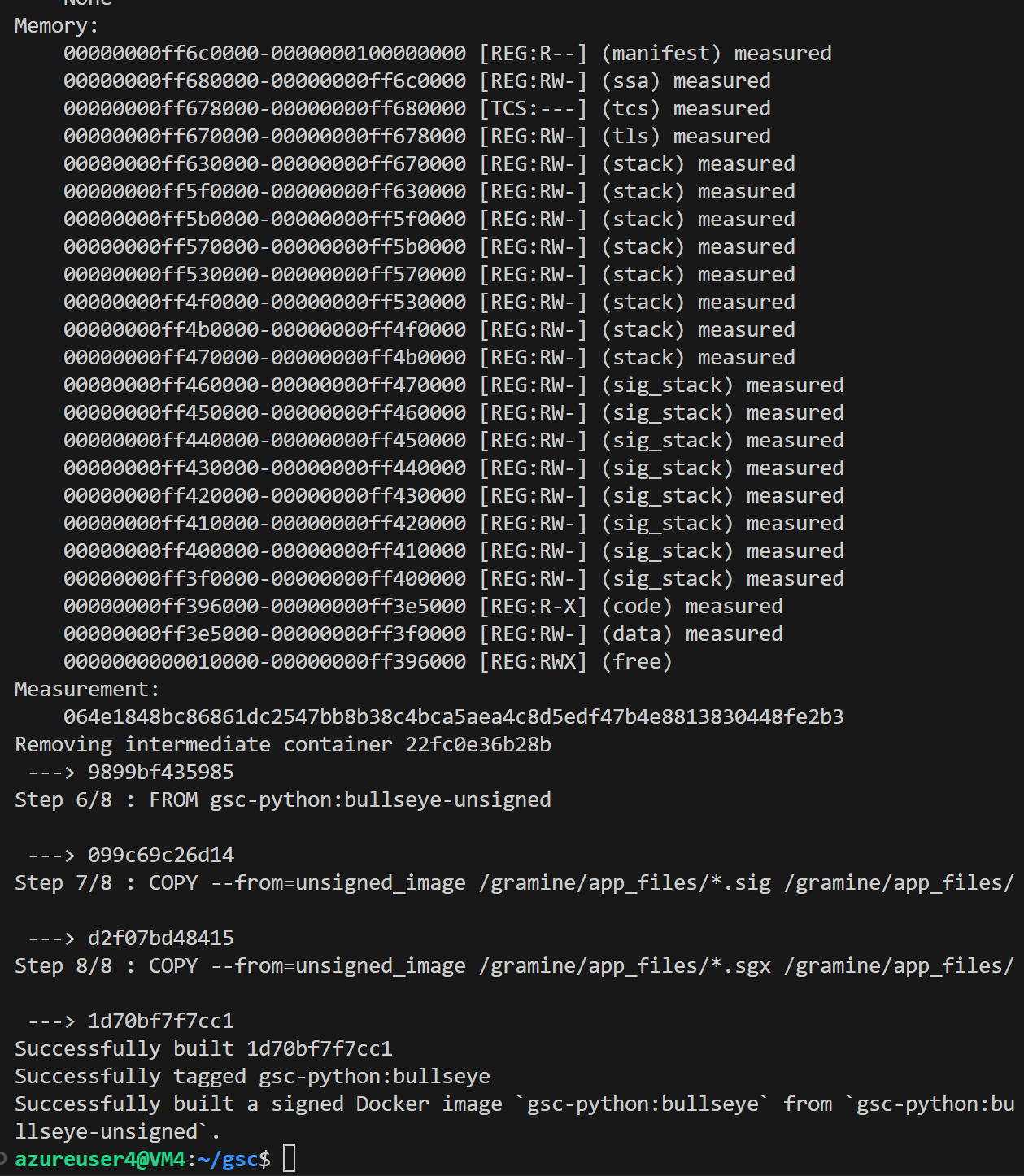
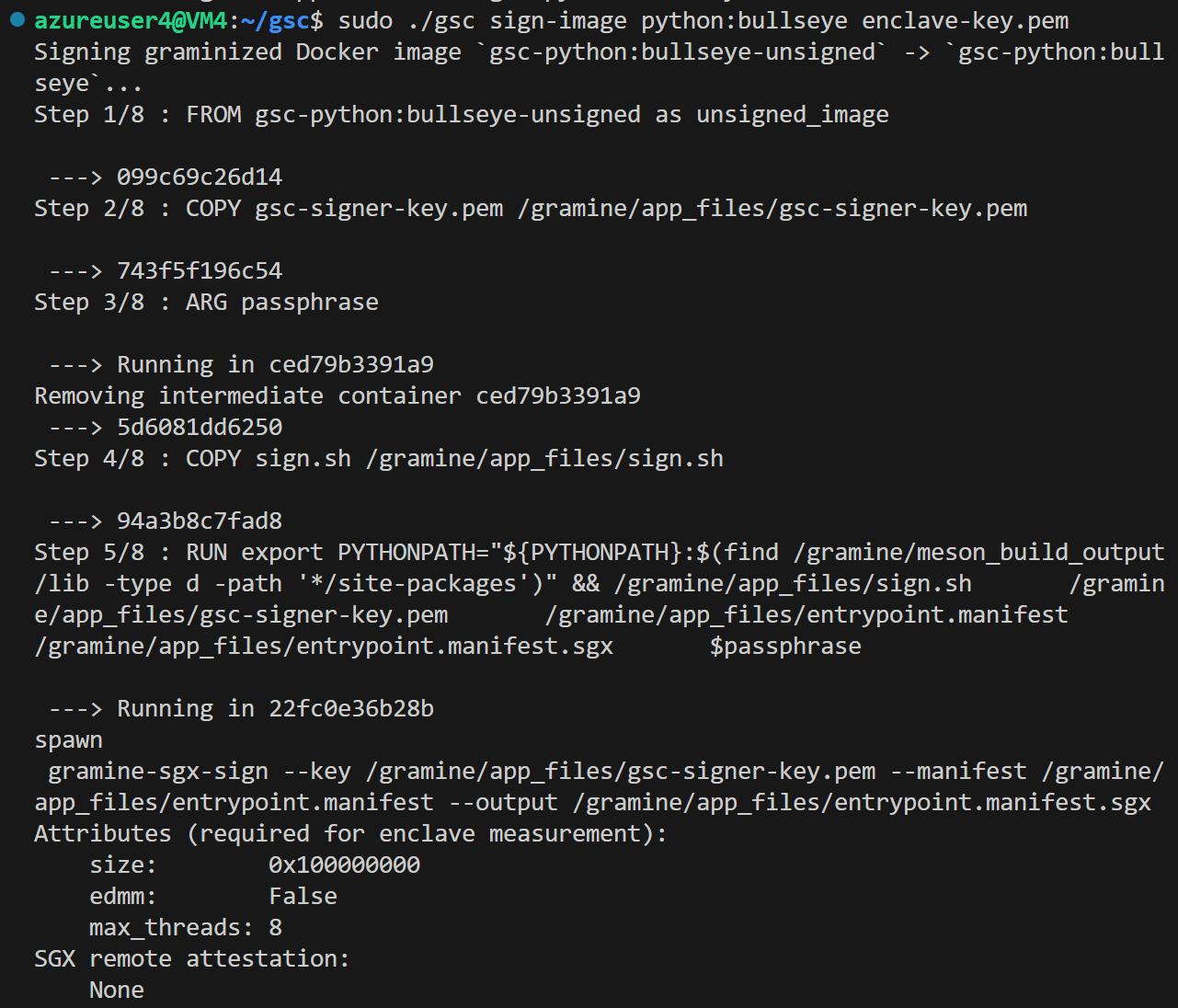
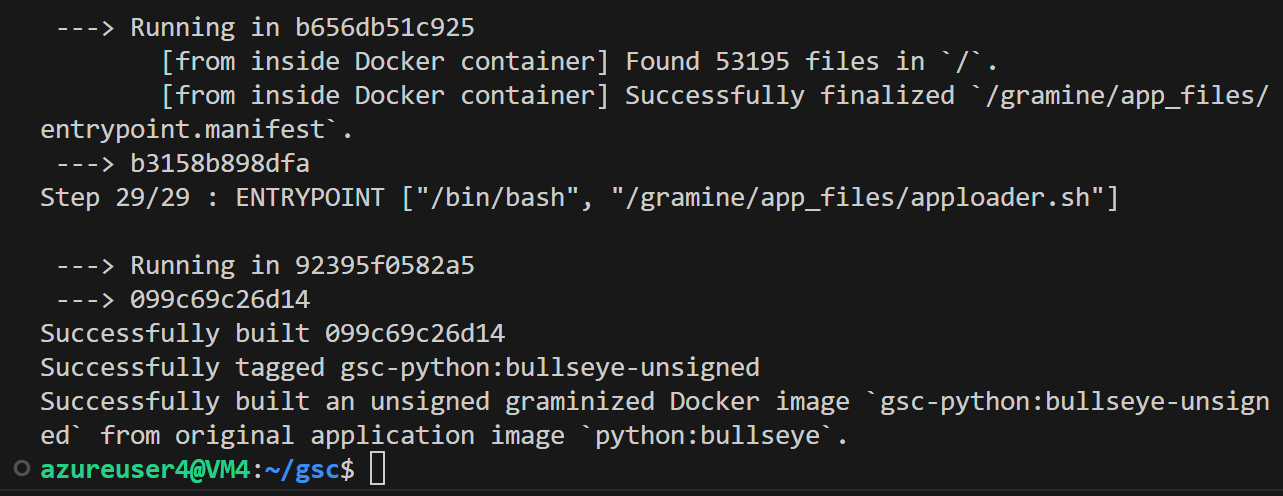
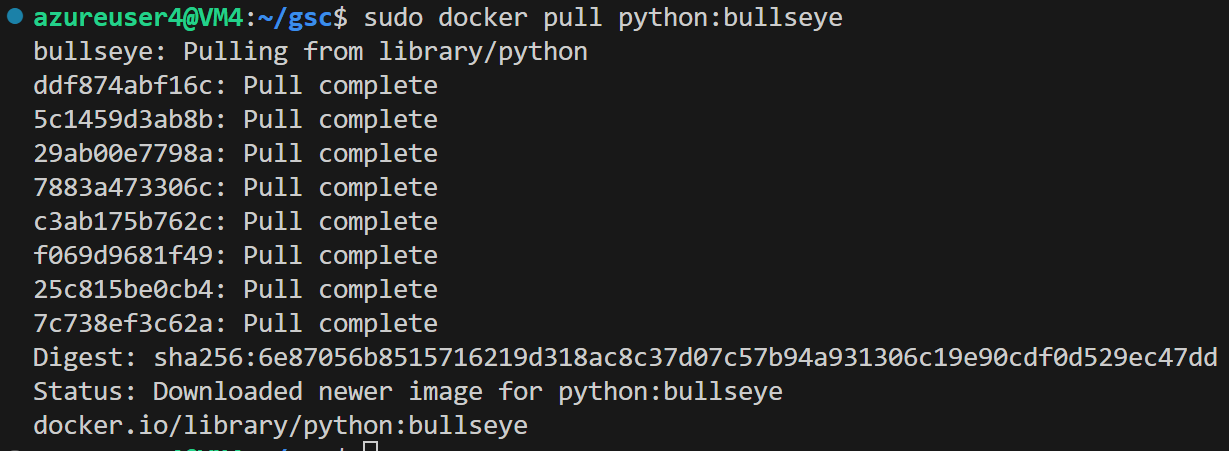
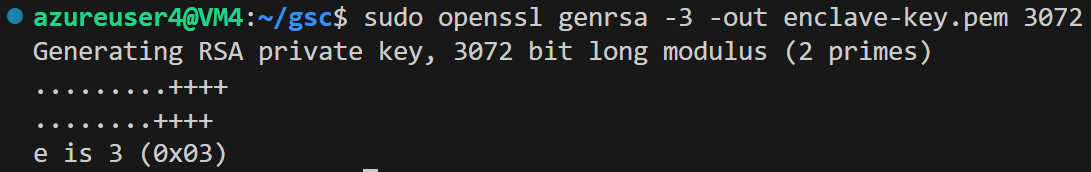
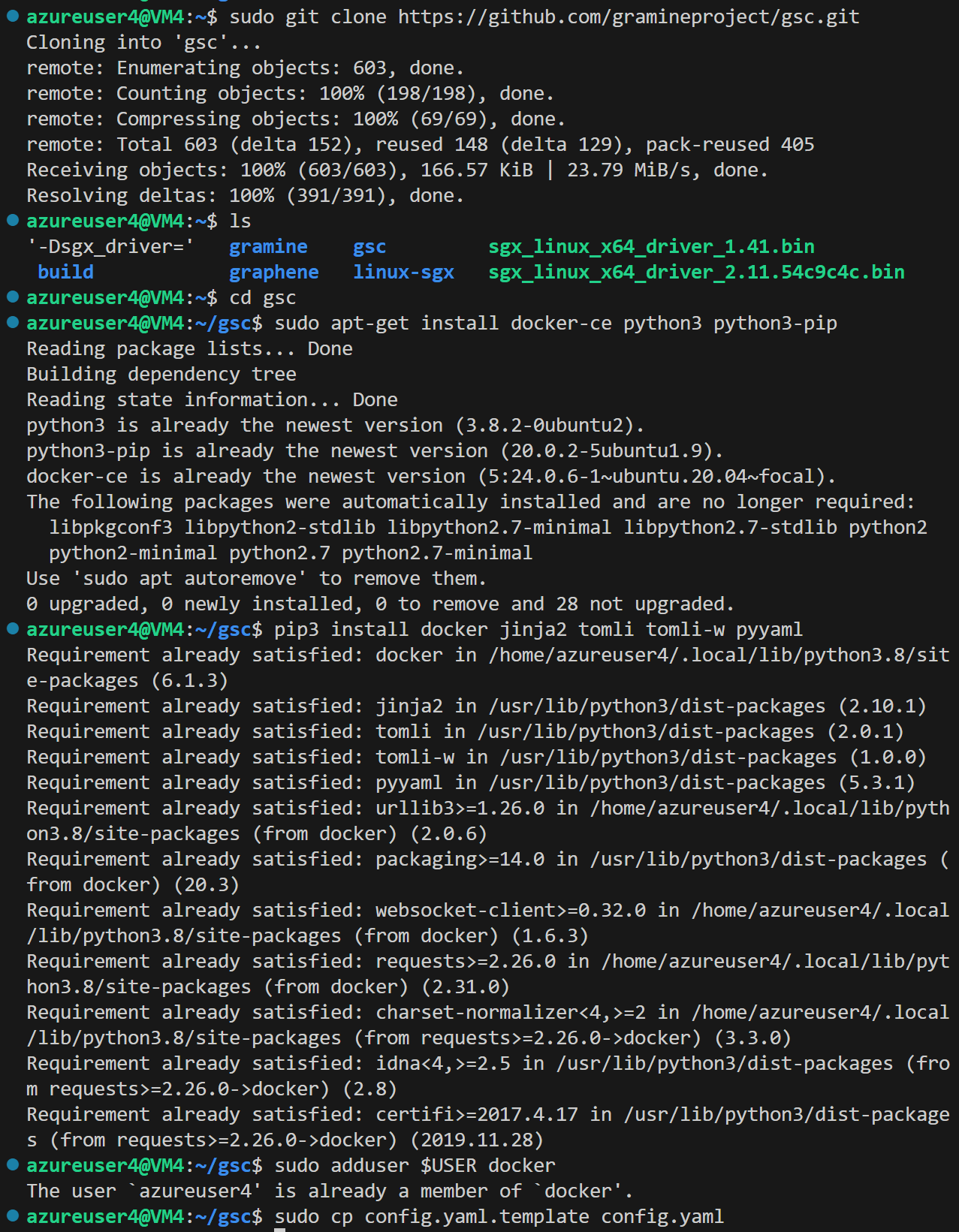
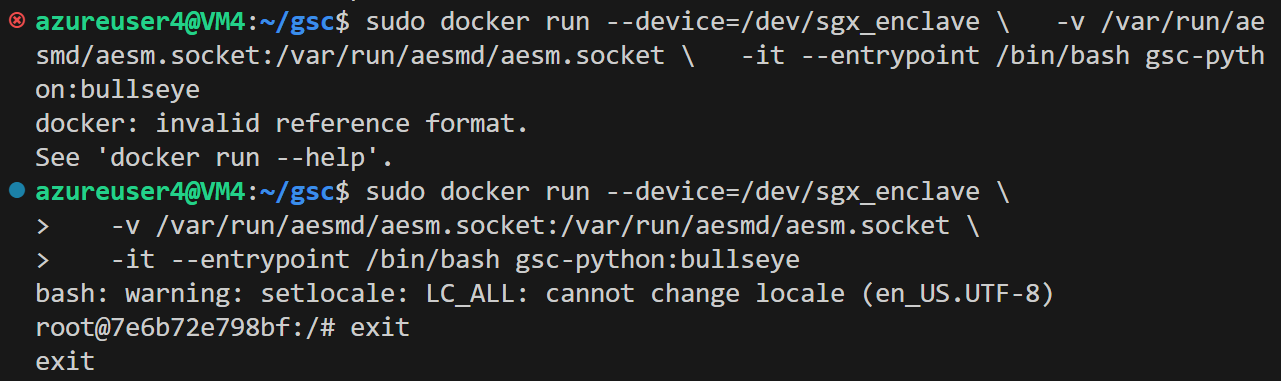
# Note: Remove backslashes and adjust the device flag if needed.

sudo docker run --device=/dev/sgx\_enclave \

-v /var/run/aesmd/aesm.socket:/var/run/aesmd/aesm.socket \

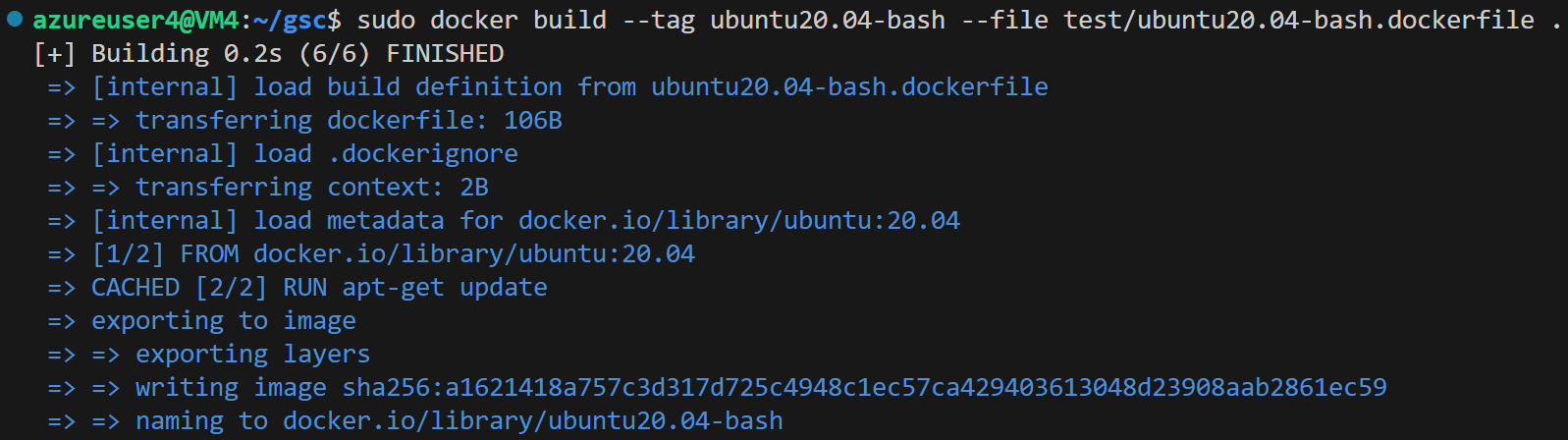
-it --entrypoint /bin/bash gsc-python:bullseye

If you encounter any errors during the last step, refer to the troubleshooting section for resolution.

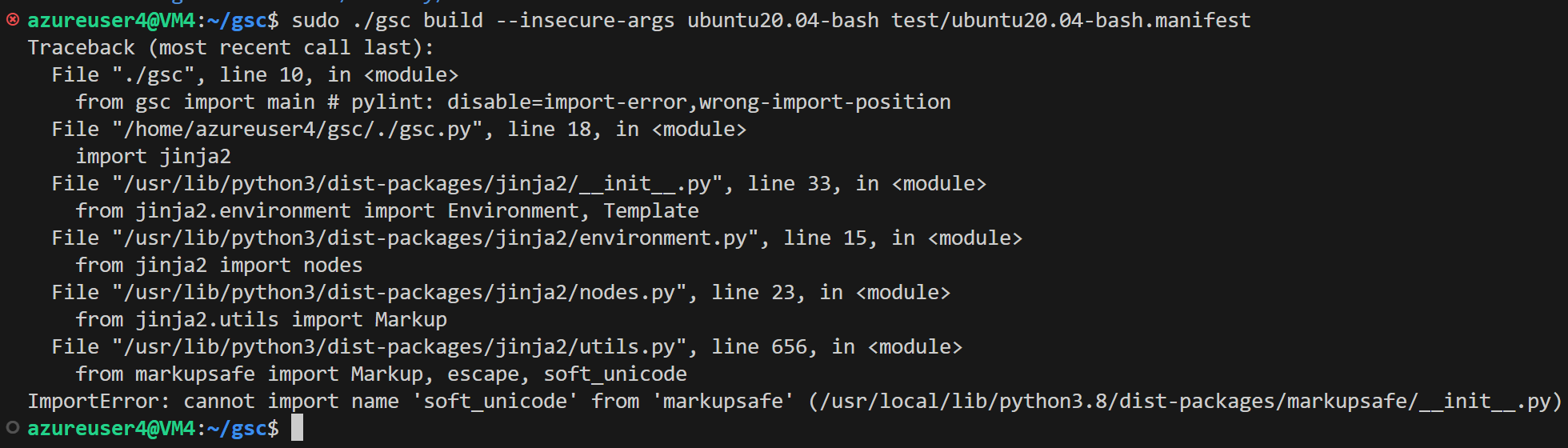


## Running the ubuntu20.04-bash.dockerfile

docker build --tag ubuntu20.04-bash --file test/ubuntu20.04-bash.dockerfile .



./gsc build --insecure-args ubuntu20.04-bash test/ubuntu20.04-bash.manifest



***Troubleshooting***

# If you encounter an error message similar to the one below, please refer to the provided solution.

\*\*Error message:\*\*

ImportError: cannot import name 'soft\_unicode' from 'markupsafe'

\*\*Solution:\*\*

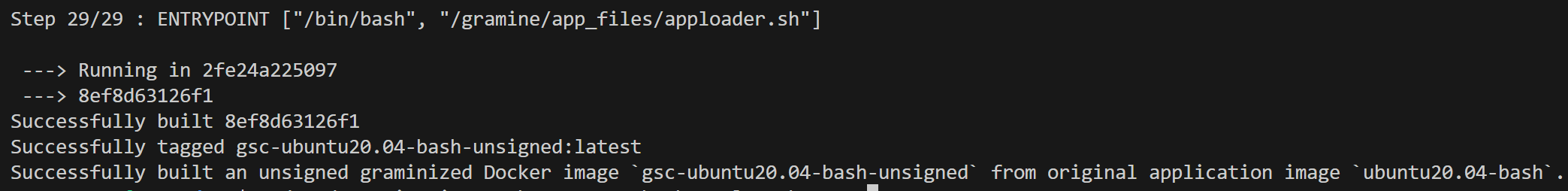
# Install a specific version of the `markupsafe` package that doesn't produce the deprecation warning about `soft\_unicode` being renamed to `soft\_str`. The warning indicates that the old name will be removed in MarkupSafe 2.1.

# Reference: [遇到ImportError: cannot import name ‘soft\_unicode’ from ‘markupsafe’问题\_cannot import name 'soft\_unicode' from 'markupsafe\_WMSmile的博客-CSDN博客](https://blog.csdn.net/wm9028/article/details/124455181)

pip install markupsafe==2.0.1

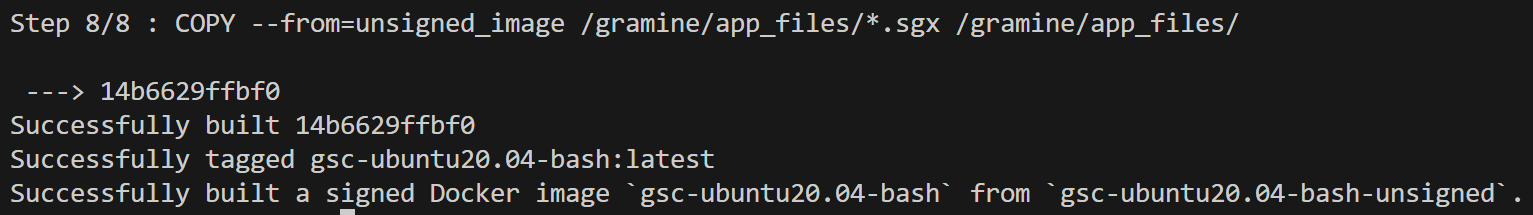
# After resolving the issue, try running the command again:

./gsc build --insecure-args ubuntu20.04-bash test/ubuntu20.04-bash.manifest



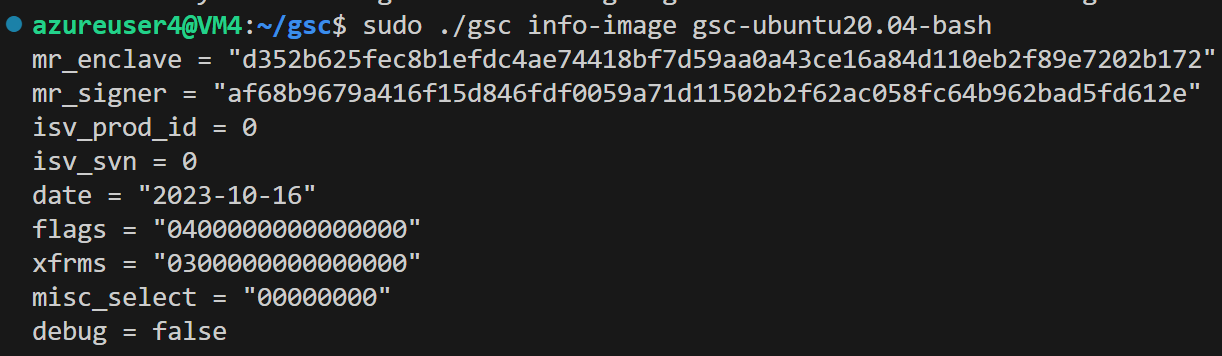
# Continue with the signing of the image:

./gsc sign-image ubuntu20.04-bash enclave-key.pem



# To get information about the signed image:

./gsc info-image gsc-ubuntu20.04-bash

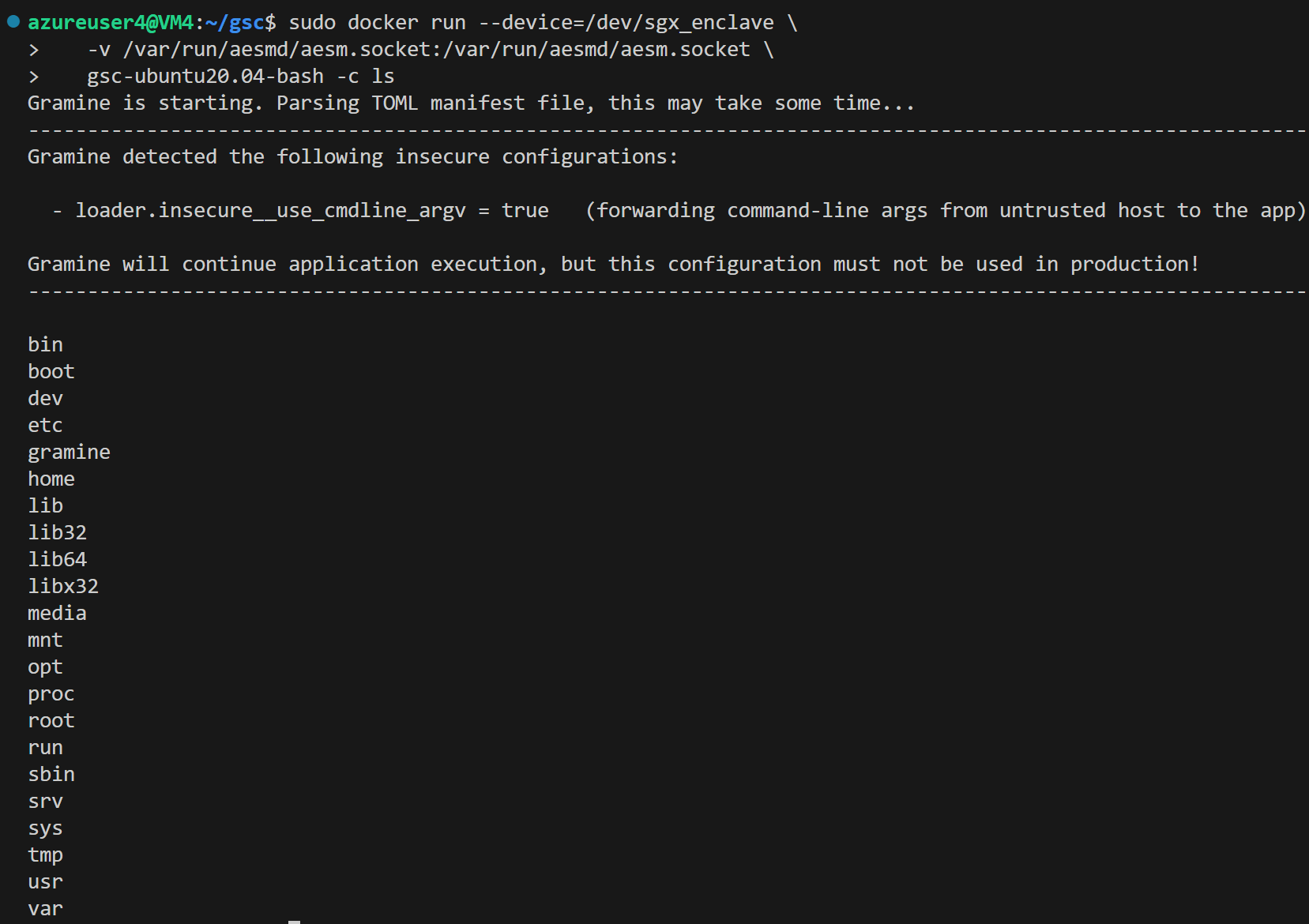


# Finally, to run the Docker container:

docker run --device=/dev/sgx\_enclave \

-v /var/run/aesmd/aesm.socket:/var/run/aesmd/aesm.socket \

gsc-ubuntu20.04-bash -c ls



If you encounter any issues with last step, please refer to the official GSC Contributions documentation for the correct command.

## Running the Ubuntu 20.04 Bash Container

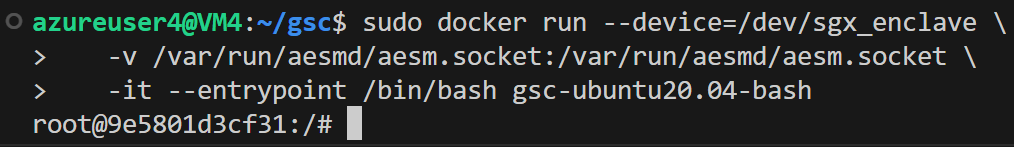
To run the Ubuntu 20.04 Bash container, use the following command:

docker run --device=/dev/sgx\_enclave \

-v /var/run/aesmd/aesm.socket:/var/run/aesmd/aesm.socket \

-it --entrypoint /bin/bash gsc-ubuntu20.04-bash

# Obtaining the container



## The same method to run the ubuntu20.04-hello-world.dockerfile

# Build the Ubuntu 20.04 Hello World dockerfile

docker build --tag ubuntu20.04-hello-world --file test/ubuntu20.04-hello-world.dockerfile .

# Build the GSC image

./gsc build --insecure-args ubuntu20.04-hello-world test/ubuntu20.04-hello-world.manifest

# Sign the GSC image

./gsc sign-image ubuntu20.04-hello-world enclave-key.pem

# Get information on the GSC image

./gsc info-image gsc-ubuntu20.04-hello-world

# Run the Ubuntu 20.04 Hello World container

docker run --device=/dev/sgx\_enclave \

-v /var/run/aesmd/aesm.socket:/var/run/aesmd/aesm.socket \

-it --entrypoint /bin/bash gsc-ubuntu20.04-hello-world