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

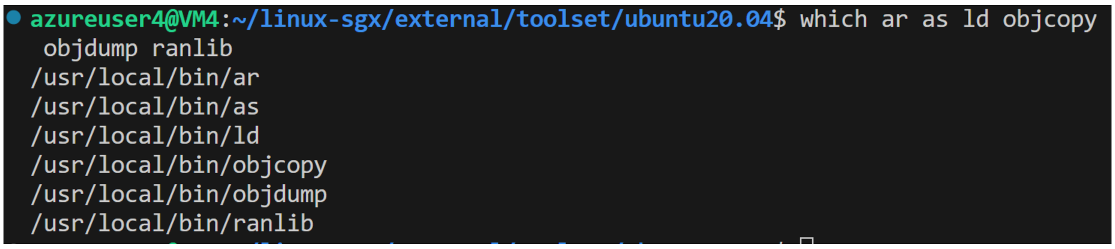
# Ensure they have execute permission

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



```
azureuser4@VM4:~/linux-sgx/external/toolset/ubuntu20.04$ which ar as ld 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

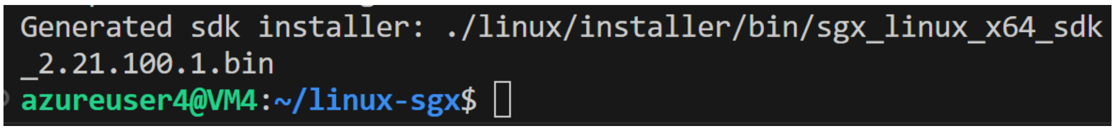
# Build Intel SGX SDK installer with debug information kept in the tools and libraries

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: ./linux/installer/bin/sgx_linux_x64_sdk
_2.21.100.1.bin
azureuser4@VM4:~/linux-sgx$ 
```

# 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

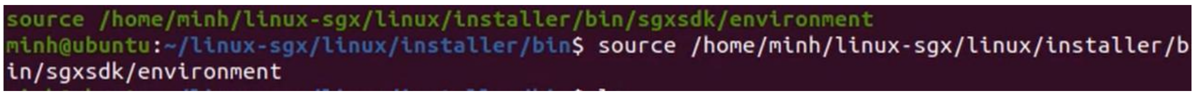
# Run the SDK installer

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/minh/linux-sgx/linux/installer/bin/sgxsdk/environment
minh@ubuntu:~/linux-sgx/linux/installer/bin$ source /home/minh/linux-sgx/linux/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:

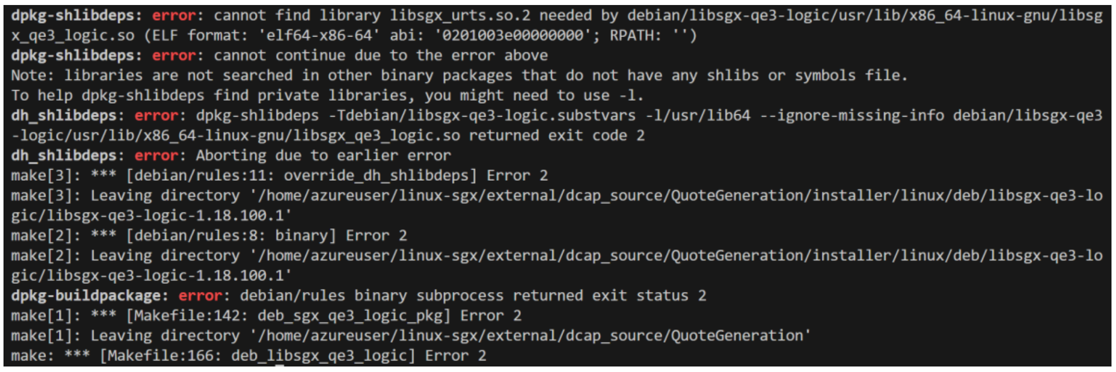
# sudo make deb_psw_pkg

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)



```
dpkg-shlibdeps: error: cannot find library libsgx_urts.so.2 needed by debian/libsgx-qe3-logic/usr/lib/x86_64-linux-gnu/libsg
x_qe3_logic.so (ELF format: 'elf64-x86-64' abi: '0201003e00000000'; 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 -l.
dh_shlibdeps: error: dpkg-shlibdeps -Tdebian/libsgx-qe3-logic.substvars -l/usr/lib64 --ignore-missing-info debian/libsgx-qe3
-logic/usr/lib/x86_64-linux-gnu/libsgx_qe3_logic.so returned exit code 2
dh_shlibdeps: error: Aborting due to earlier error
make[3]: *** [debian/rules:11: override_dh_shlibdeps] Error 2
make[3]: Leaving directory '/home/azureuser/linux-sgx/external/dcap_source/QuoteGeneration/installer/linux/deb/libsgx-qe3-lo
gic/libsgx-qe3-logic-1.18.100.1'
make[2]: *** [debian/rules:8: binary] Error 2
make[2]: Leaving directory '/home/azureuser/linux-sgx/external/dcap_source/QuoteGeneration/installer/linux/deb/libsgx-qe3-lo
gic/libsgx-qe3-logic-1.18.100.1'
dpkg-buildpackage: error: debian/rules binary subprocess returned exit status 2
make[1]: *** [Makefile:142: deb_sgx_qe3_logic_pkg] Error 2
make[1]: Leaving directory '/home/azureuser/linux-sgx/external/dcap_source/QuoteGeneration'
make: *** [Makefile:166: deb_libsgx_qe3_logic] 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

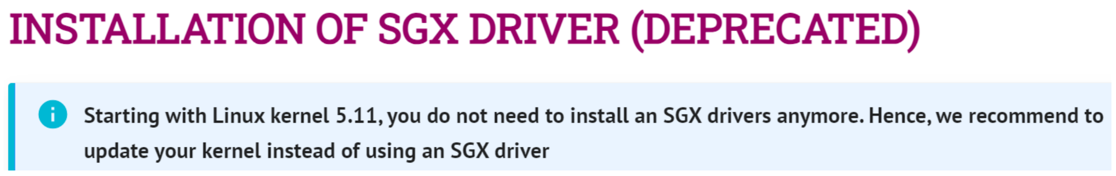
```
sudo apt-get install libsgx-dcap-ql

# 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).



INSTALLATION OF SGX DRIVER (DEPRECATED)

Starting with Linux kernel 5.11, you do not need to install an SGX drivers anymore. Hence, we recommend to update your kernel instead of using an SGX driver

<u>Install GSC (Gramine Shielded Containers)</u>

## 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.

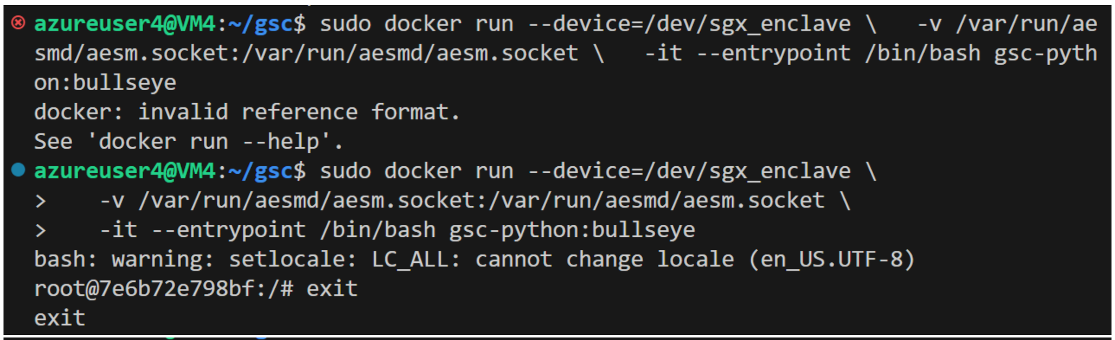
The third stage uses Gramine's signer tool to generate SIGSTRUCT files for SGX enclave initialization. This tool also generates an SGX-specific manifest file. The required signing key is provided by the user via the gsc sign-image command and copied into this Docker build stage. The generated image is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

## Run a GSC Example

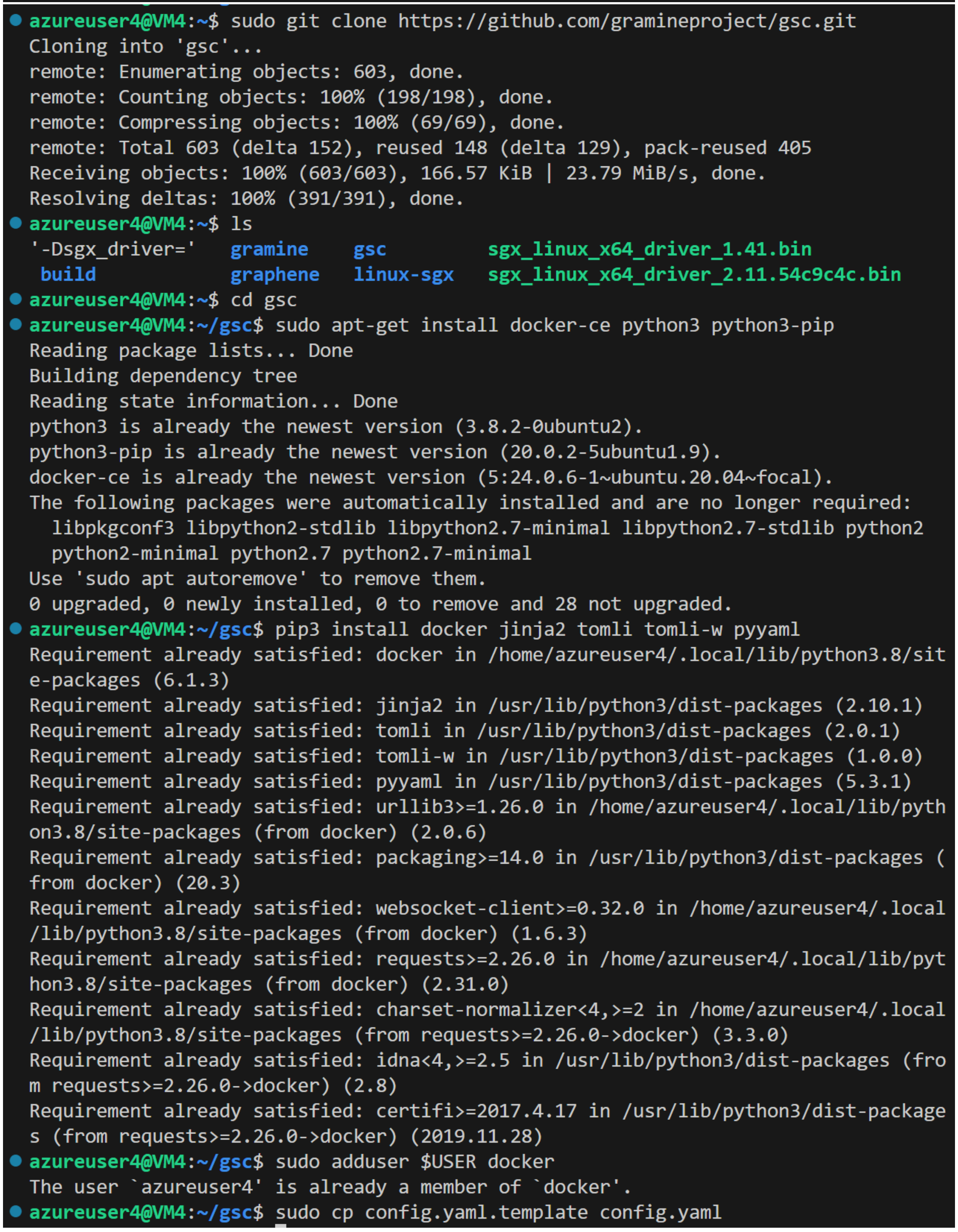
```
# First, clone the GSC repository:
git clone https://github.com/gramineproject/gsc.git
cd gsc

# 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
```

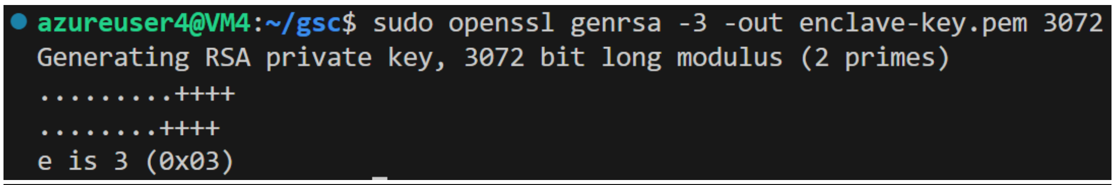
If you encounter any errors during the last step, refer to the troubleshooting section for resolution.



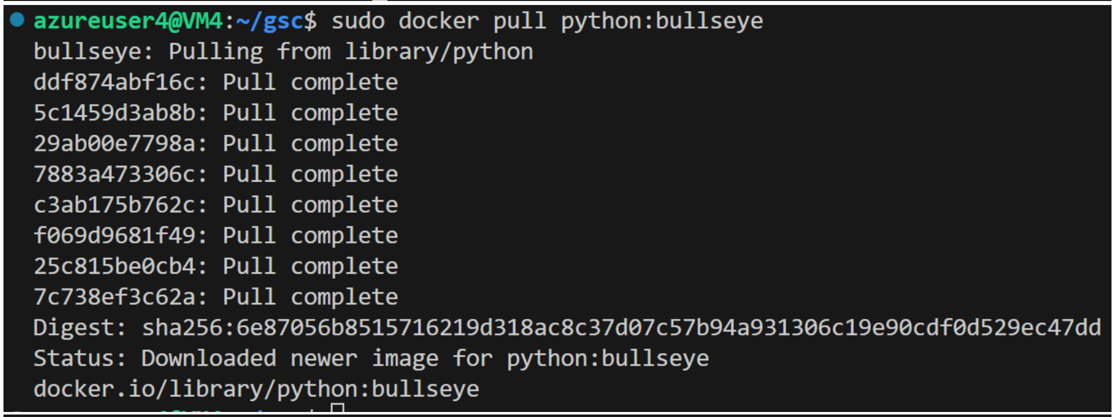
```
azureuser4@VM4:~/gsc$ sudo docker run --device=/dev/sgx_enclave \   -v /var/run/ae
smd/aesm.socket:/var/run/aesmd/aesm.socket \   -it --entrypoint /bin/bash gsc-pyth
on:bullseye
docker: invalid reference format.
See 'docker run --help'.
azureuser4@VM4:~/gsc$ sudo docker run --device=/dev/sgx_enclave \
>    -v /var/run/aesmd/aesm.socket:/var/run/aesmd/aesm.socket \
>    -it --entrypoint /bin/bash gsc-python:bullseye
bash: warning: setlocale: LC_ALL: cannot change locale (en_US.UTF-8)
root@7e6b72e798bf:/# exit
exit
```



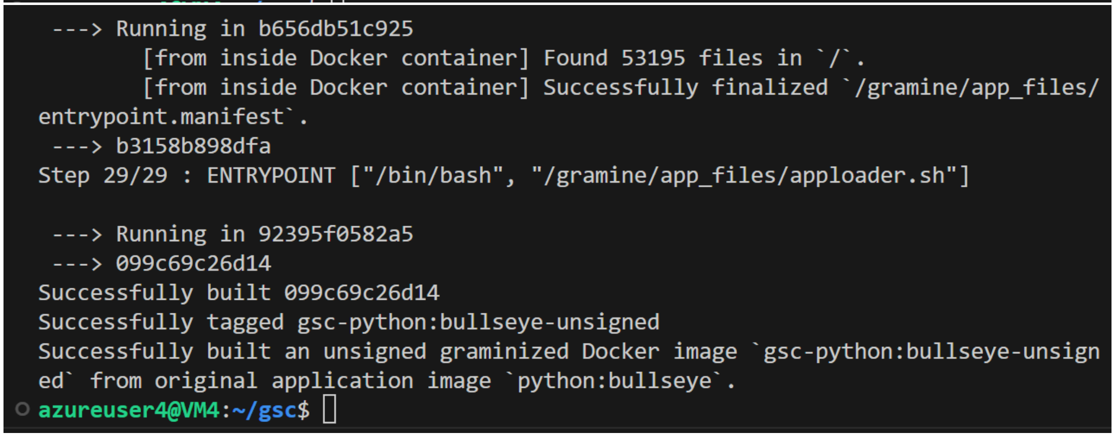
```
azureuser4@VM4:~$ sudo git clone https://github.com/gramineproject/gsc.git
Cloning into 'gsc'...
remote: Enumerating objects: 603, done.
remote: Counting objects: 100% (198/198), done.
remote: Compressing objects: 100% (69/69), done.
remote: Total 603 (delta 152), reused 148 (delta 129), pack-reused 405
Receiving objects: 100% (603/603), 166.57 KiB | 23.79 MiB/s, done.
Resolving deltas: 100% (391/391), done.
azureuser4@VM4:~$ ls
'-Dsgx_driver='   gramine    gsc         sgx_linux_x64_driver_1.41.bin
 build            graphene   linux-sgx   sgx_linux_x64_driver_2.11.54c9c4c.bin
azureuser4@VM4:~$ cd gsc
azureuser4@VM4:~/gsc$ sudo apt-get install docker-ce python3 python3-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3 is already the newest version (3.8.2-0ubuntu2).
python3-pip is already the newest version (20.0.2-5ubuntu1.9).
docker-ce is already the newest version (5:24.0.6-1~ubuntu.20.04~focal).
The following packages were automatically installed and are no longer required:
  libpkgconf3 libpython2-stdlib libpython2.7-minimal libpython2.7-stdlib python2
  python2-minimal python2.7 python2.7-minimal
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 28 not upgraded.
azureuser4@VM4:~/gsc$ pip3 install docker jinja2 tomli tomli-w pyyaml
Requirement already satisfied: docker in /home/azureuser4/.local/lib/python3.8/sit
e-packages (6.1.3)
Requirement already satisfied: jinja2 in /usr/lib/python3/dist-packages (2.10.1)
Requirement already satisfied: tomli in /usr/lib/python3/dist-packages (2.0.1)
Requirement already satisfied: tomli-w in /usr/lib/python3/dist-packages (1.0.0)
Requirement already satisfied: pyyaml in /usr/lib/python3/dist-packages (5.3.1)
Requirement already satisfied: urllib3>=1.26.0 in /home/azureuser4/.local/lib/pyth
on3.8/site-packages (from docker) (2.0.6)
Requirement already satisfied: packaging>=14.0 in /usr/lib/python3/dist-packages (
from docker) (20.3)
Requirement already satisfied: websocket-client>=0.32.0 in /home/azureuser4/.local
/lib/python3.8/site-packages (from docker) (1.6.3)
Requirement already satisfied: requests>=2.26.0 in /home/azureuser4/.local/lib/pyt
hon3.8/site-packages (from docker) (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /home/azureuser4/.local
/lib/python3.8/site-packages (from requests>=2.26.0->docker) (3.3.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/lib/python3/dist-packages (fro
m requests>=2.26.0->docker) (2.8)
Requirement already satisfied: certifi>=2017.4.17 in /usr/lib/python3/dist-package
s (from requests>=2.26.0->docker) (2019.11.28)
azureuser4@VM4:~/gsc$ sudo adduser $USER docker
The user `azureuser4' is already a member of `docker'.
azureuser4@VM4:~/gsc$ sudo cp config.yaml.template config.yaml
```



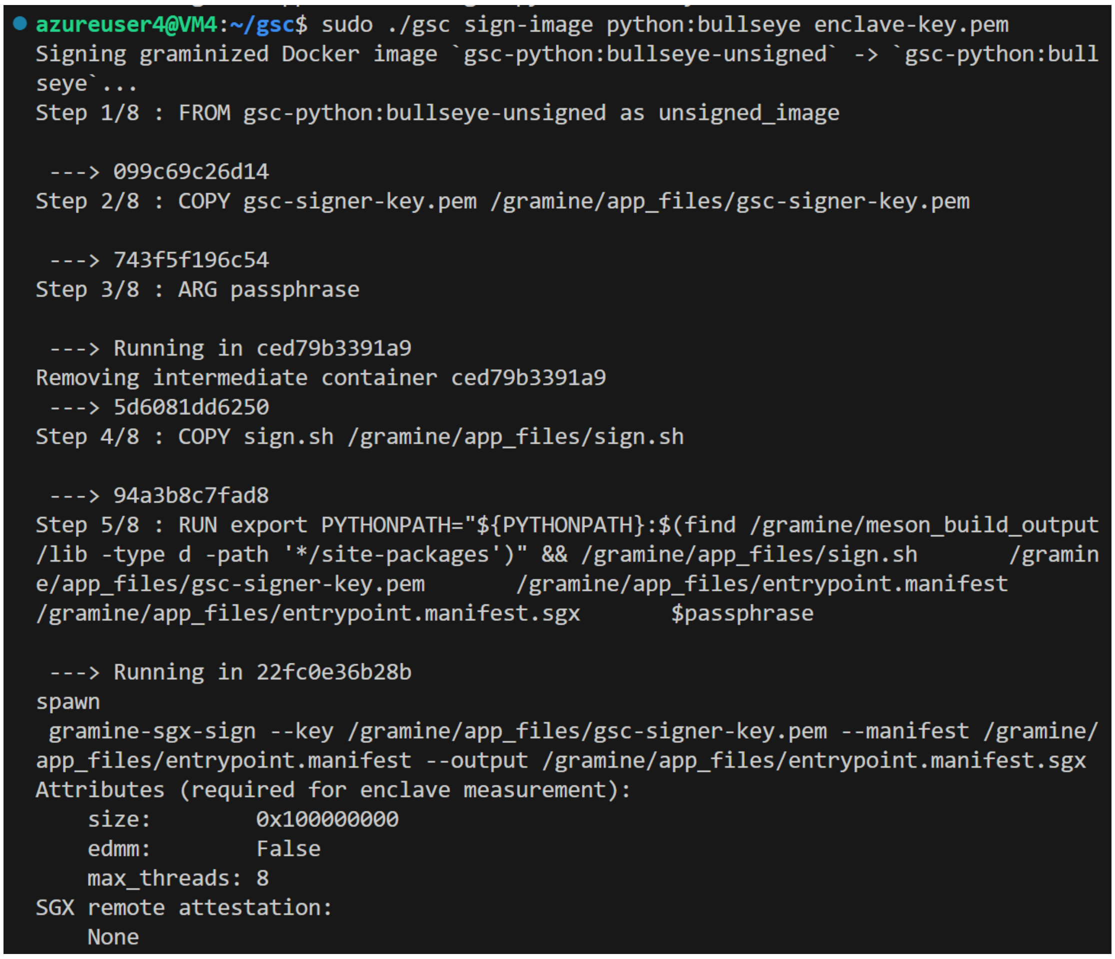
```
azureuser4@VM4:~/gsc$ sudo openssl genrsa -3 -out enclave-key.pem 3072
Generating RSA private key, 3072 bit long modulus (2 primes)
.........++++
........++++
e is 3 (0x03)
```



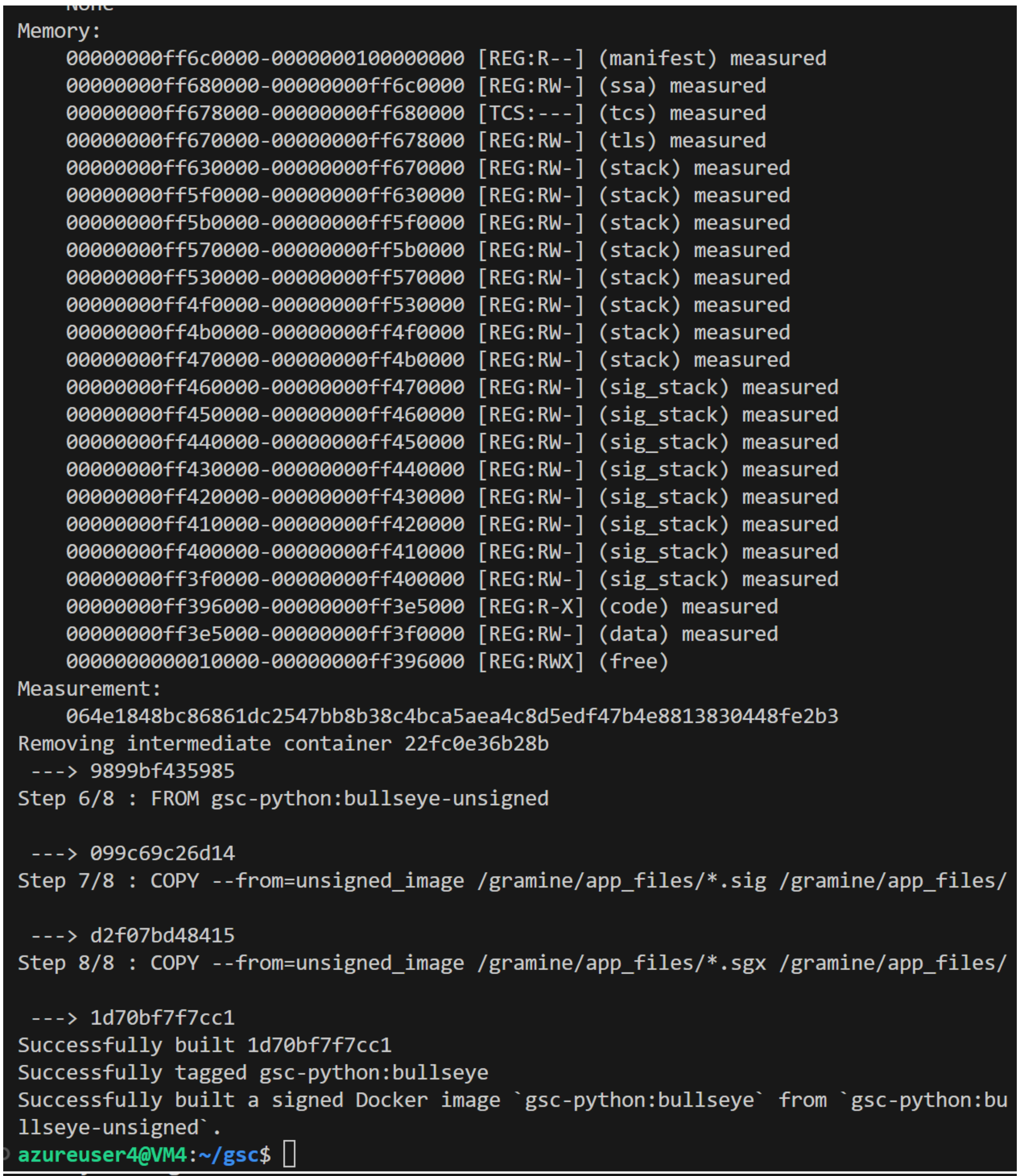
```
azureuser4@VM4:~/gsc$ sudo docker pull python:bullseye
bullseye: Pulling from library/python
ddf874abf16c: Pull complete
5c1459d3ab8b: Pull complete
29ab00e7798a: Pull complete
7883a473306c: Pull complete
c3ab175b762c: Pull complete
f069d9681f49: Pull complete
25c815be0cb4: Pull complete
7c738ef3c62a: Pull complete
Digest: sha256:6e87056b8515716219d318ac8c37d07c57b94a931306c19e90cdf0d529ec47dd
Status: Downloaded newer image for python:bullseye
docker.io/library/python:bullseye
```



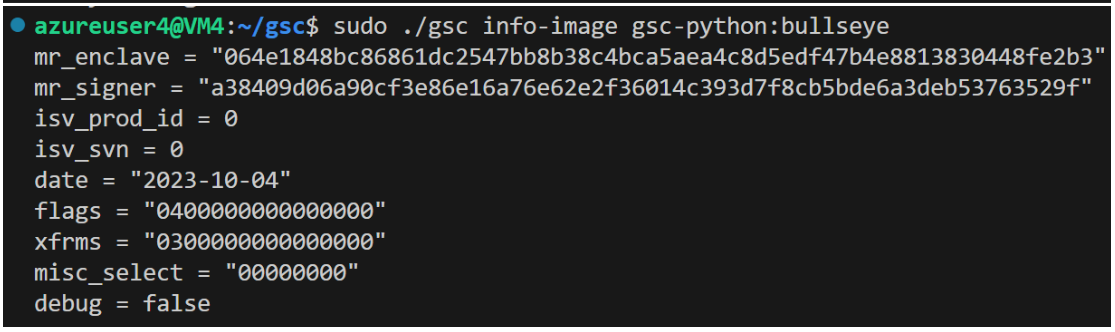
```
 ---> Running in b656db51c925
        [from inside Docker container] Found 53195 files in `/`.
        [from inside Docker container] Successfully finalized `/gramine/app_files/
entrypoint.manifest`.
 ---> b3158b898dfa
Step 29/29 : ENTRYPOINT ["/bin/bash", "/gramine/app_files/apploader.sh"]

 ---> Running in 92395f0582a5
 ---> 099c69c26d14
Successfully built 099c69c26d14
Successfully tagged gsc-python:bullseye-unsigned
Successfully built an unsigned graminized Docker image `gsc-python:bullseye-unsign
ed` from original application image `python:bullseye`.
azureuser4@VM4:~/gsc$ 
```



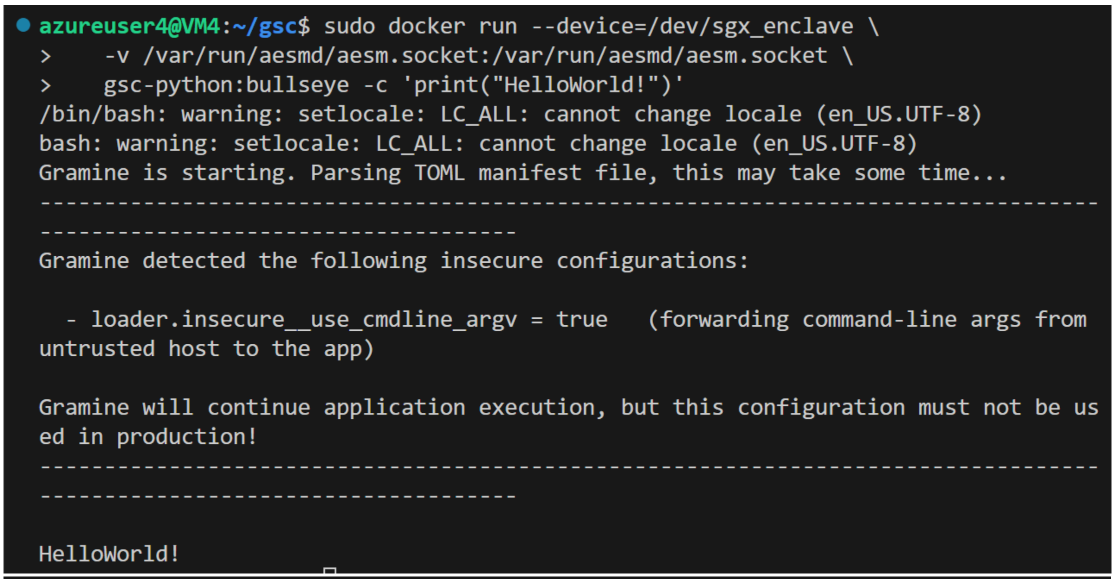
```
● azureuser4@VM4:~/gsc$ sudo ./gsc sign-image python:bullseye enclave-key.pem
  Signing graminized Docker image `gsc-python:bullseye-unsigned` -> `gsc-python:bull
  seye`...
  Step 1/8 : FROM gsc-python:bullseye-unsigned as unsigned_image

   ---> 099c69c26d14
  Step 2/8 : COPY gsc-signer-key.pem /gramine/app_files/gsc-signer-key.pem

   ---> 743f5f196c54
  Step 3/8 : ARG passphrase

   ---> Running in ced79b3391a9
  Removing intermediate container ced79b3391a9
   ---> 5d6081dd6250
  Step 4/8 : COPY sign.sh /gramine/app_files/sign.sh

   ---> 94a3b8c7fad8
  Step 5/8 : RUN export PYTHONPATH="${PYTHONPATH}:$(find /gramine/meson_build_output
  /lib -type d -path '*/site-packages')" && /gramine/app_files/sign.sh       /gramin
  e/app_files/gsc-signer-key.pem       /gramine/app_files/entrypoint.manifest
  /gramine/app_files/entrypoint.manifest.sgx       $passphrase

   ---> Running in 22fc0e36b28b
  spawn
   gramine-sgx-sign --key /gramine/app_files/gsc-signer-key.pem --manifest /gramine/
  app_files/entrypoint.manifest --output /gramine/app_files/entrypoint.manifest.sgx
  Attributes (required for enclave measurement):
      size:        0x100000000
      edmm:        False
      max_threads: 8
  SGX remote attestation:
      None
```



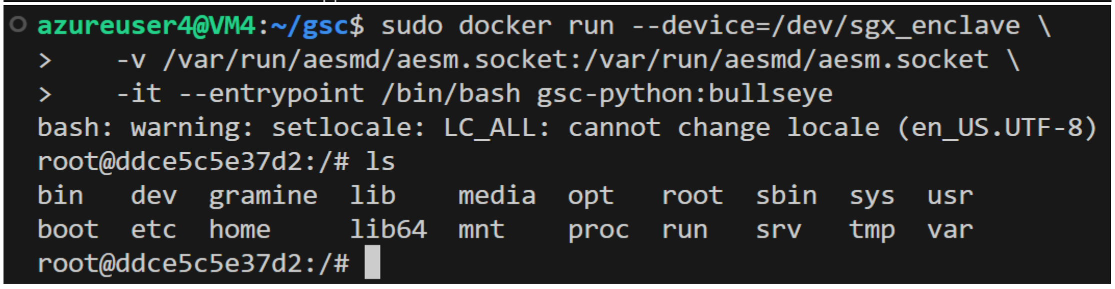
```
    None
Memory:
    00000000ff6c0000-0000000100000000 [REG:R--] (manifest) measured
    00000000ff680000-00000000ff6c0000 [REG:RW-] (ssa) measured
    00000000ff678000-00000000ff680000 [TCS:---] (tcs) measured
    00000000ff670000-00000000ff678000 [REG:RW-] (tls) measured
    00000000ff630000-00000000ff670000 [REG:RW-] (stack) measured
    00000000ff5f0000-00000000ff630000 [REG:RW-] (stack) measured
    00000000ff5b0000-00000000ff5f0000 [REG:RW-] (stack) measured
    00000000ff570000-00000000ff5b0000 [REG:RW-] (stack) measured
    00000000ff530000-00000000ff570000 [REG:RW-] (stack) measured
    00000000ff4f0000-00000000ff530000 [REG:RW-] (stack) measured
    00000000ff4b0000-00000000ff4f0000 [REG:RW-] (stack) measured
    00000000ff470000-00000000ff4b0000 [REG:RW-] (stack) measured
    00000000ff460000-00000000ff470000 [REG:RW-] (sig_stack) measured
    00000000ff450000-00000000ff460000 [REG:RW-] (sig_stack) measured
    00000000ff440000-00000000ff450000 [REG:RW-] (sig_stack) measured
    00000000ff430000-00000000ff440000 [REG:RW-] (sig_stack) measured
    00000000ff420000-00000000ff430000 [REG:RW-] (sig_stack) measured
    00000000ff410000-00000000ff420000 [REG:RW-] (sig_stack) measured
    00000000ff400000-00000000ff410000 [REG:RW-] (sig_stack) measured
    00000000ff3f0000-00000000ff400000 [REG:RW-] (sig_stack) measured
    00000000ff396000-00000000ff3e5000 [REG:R-X] (code) measured
    00000000ff3e5000-00000000ff3f0000 [REG:RW-] (data) measured
    0000000000010000-00000000ff396000 [REG:RWX] (free)
Measurement:
    064e1848bc86861dc2547bb8b38c4bca5aea4c8d5edf47b4e8813830448fe2b3
Removing intermediate container 22fc0e36b28b
 ---> 9899bf435985
Step 6/8 : FROM gsc-python:bullseye-unsigned

 ---> 099c69c26d14
Step 7/8 : COPY --from=unsigned_image /gramine/app_files/*.sig /gramine/app_files/

 ---> d2f07bd48415
Step 8/8 : COPY --from=unsigned_image /gramine/app_files/*.sgx /gramine/app_files/

 ---> 1d70bf7f7cc1
Successfully built 1d70bf7f7cc1
Successfully tagged gsc-python:bullseye
Successfully built a signed Docker image `gsc-python:bullseye` from `gsc-python:bu
llseye-unsigned`.
azureuser4@VM4:~/gsc$ 
```



```
azureuser4@VM4:~/gsc$ sudo ./gsc info-image gsc-python:bullseye
 mr_enclave = "064e1848bc86861dc2547bb8b38c4bca5aea4c8d5edf47b4e8813830448fe2b3"
 mr_signer = "a38409d06a90cf3e86e16a76e62e2f36014c393d7f8cb5bde6a3deb53763529f"
 isv_prod_id = 0
 isv_svn = 0
 date = "2023-10-04"
 flags = "0400000000000000"
 xfrms = "0300000000000000"
 misc_select = "00000000"
 debug = false
```



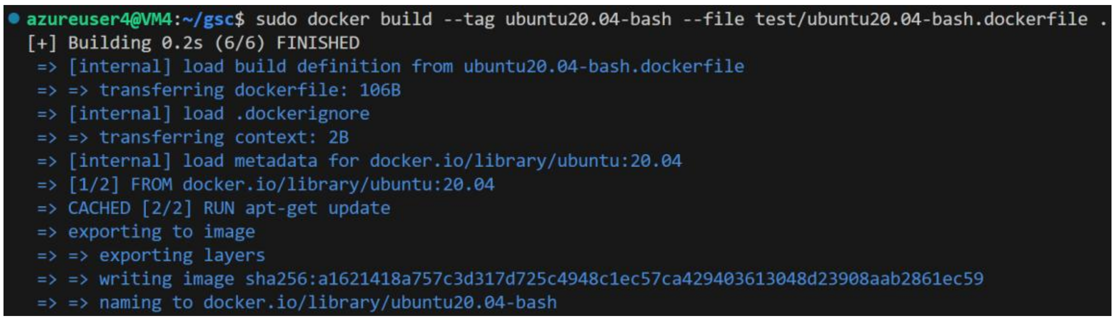
```
azureuser4@VM4:~/gsc$ sudo docker run --device=/dev/sgx_enclave \
>    -v /var/run/aesmd/aesm.socket:/var/run/aesmd/aesm.socket \
>    gsc-python:bullseye -c 'print("HelloWorld!")'
/bin/bash: warning: setlocale: LC_ALL: cannot change locale (en_US.UTF-8)
bash: warning: setlocale: LC_ALL: cannot change locale (en_US.UTF-8)
Gramine is starting. Parsing TOML manifest file, this may take some time...
--------------------------------------------------------------------------------
------------------------------------
Gramine detected the following insecure configurations:

  - loader.insecure__use_cmdline_argv = true    (forwarding command-line args from
untrusted host to the app)

Gramine will continue application execution, but this configuration must not be us
ed in production!
--------------------------------------------------------------------------------
------------------------------------

HelloWorld!
```



```
azureuser4@VM4:~/gsc$ 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
bash: warning: setlocale: LC_ALL: cannot change locale (en_US.UTF-8)
root@ddce5c5e37d2:/# ls
bin   dev  gramine  lib    media  opt   root  sbin  sys  usr
boot  etc  home     lib64  mnt    proc  run   srv   tmp  var
root@ddce5c5e37d2:/# 
```

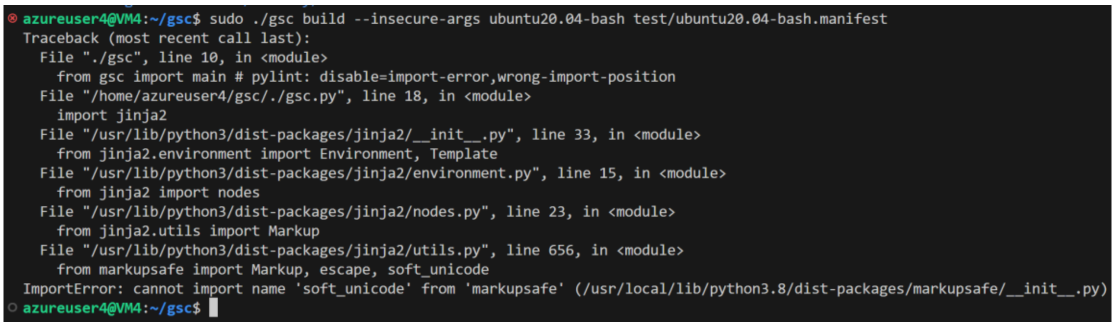
## Running the ubuntu20.04-bash.dockerfile

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



```
azureuser4@VM4:~/gsc$ sudo docker build --tag ubuntu20.04-bash --file test/ubuntu20.04-bash.dockerfile .
[+] Building 0.2s (6/6) FINISHED
 => [internal] load build definition from ubuntu20.04-bash.dockerfile
 => => transferring dockerfile: 106B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/ubuntu:20.04
 => [1/2] FROM docker.io/library/ubuntu:20.04
 => CACHED [2/2] RUN apt-get update
 => exporting to image
 => => exporting layers
 => => writing image sha256:a1621418a757c3d317d725c4948c1ec57ca429403613048d23908aab2861ec59
 => => naming to docker.io/library/ubuntu20.04-bash
```

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



```
azureuser4@VM4:~/gsc$ sudo ./gsc build --insecure-args ubuntu20.04-bash test/ubuntu20.04-bash.manifest
Traceback (most recent call last):
  File "./gsc", line 10, in <module>
    from gsc import main # pylint: disable=import-error,wrong-import-position
  File "/home/azureuser4/gsc/./gsc.py", line 18, in <module>
    import jinja2
  File "/usr/lib/python3/dist-packages/jinja2/__init__.py", line 33, in <module>
    from jinja2.environment import Environment, Template
  File "/usr/lib/python3/dist-packages/jinja2/environment.py", line 15, in <module>
    from jinja2 import nodes
  File "/usr/lib/python3/dist-packages/jinja2/nodes.py", line 23, in <module>
    from jinja2.utils import Markup
  File "/usr/lib/python3/dist-packages/jinja2/utils.py", line 656, in <module>
    from markupsafe import Markup, escape, soft_unicode
ImportError: cannot import name 'soft_unicode' from 'markupsafe' (/usr/local/lib/python3.8/dist-packages/markupsafe/__init__.py)
azureuser4@VM4:~/gsc$ 
```

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:

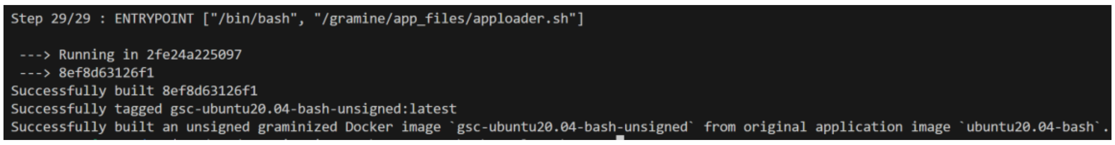
# Install a specific version of the `markupsafe` package that doesn't produce the deprecation warning about `soft_unicode` being renamed to `soft_str`. The warning indicates that the old name will be removed in MarkupSafe 2.1.

# 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

pip install markupsafe==2.0.1

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

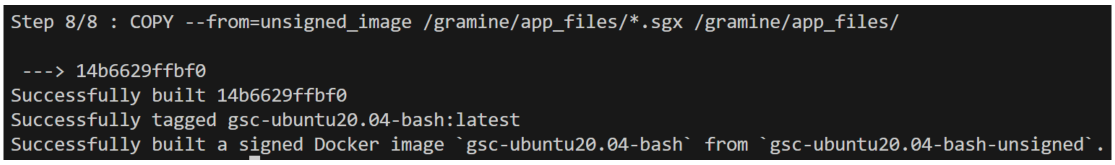
./gsc build --insecure-args ubuntu20.04-bash test/ubuntu20.04-bash.manifest



```
Step 29/29 : ENTRYPOINT ["/bin/bash", "/gramine/app_files/apploader.sh"]

 ---> Running in 2fe24a225097
 ---> 8ef8d63126f1
Successfully built 8ef8d63126f1
Successfully tagged gsc-ubuntu20.04-bash-unsigned:latest
Successfully built an unsigned graminized Docker image `gsc-ubuntu20.04-bash-unsigned` from original application image `ubuntu20.04-bash`.
```

# Continue with the signing of the image:

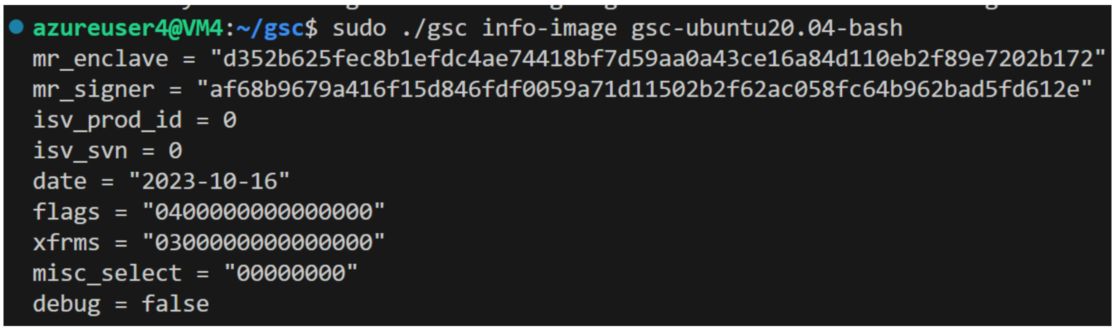
./gsc sign-image ubuntu20.04-bash enclave-key.pem



```
Step 8/8 : COPY --from=unsigned_image /gramine/app_files/*.sgx /gramine/app_files/

 ---> 14b6629ffbf0
Successfully built 14b6629ffbf0
Successfully tagged gsc-ubuntu20.04-bash:latest
Successfully built a signed Docker image `gsc-ubuntu20.04-bash` from `gsc-ubuntu20.04-bash-unsigned`.
```

# To get information about the signed image:

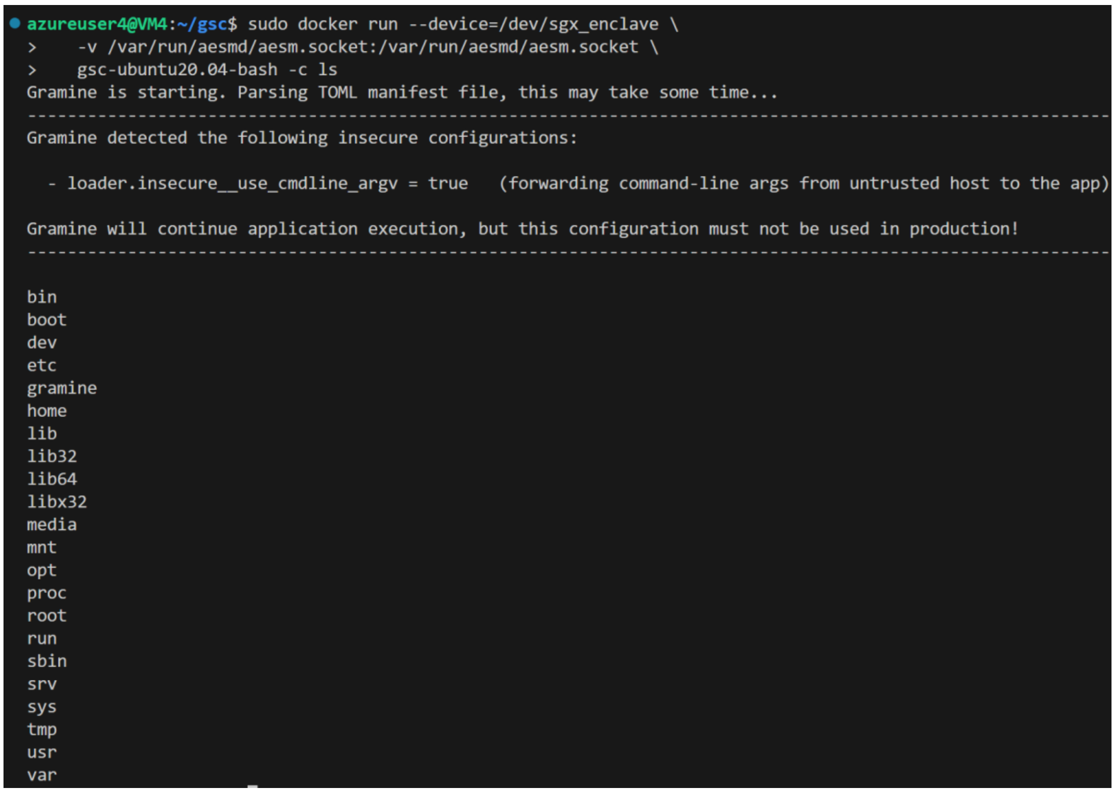
./gsc info-image gsc-ubuntu20.04-bash



```
azureuser4@VM4:~/gsc$ sudo ./gsc info-image gsc-ubuntu20.04-bash
  mr_enclave = "d352b625fec8b1efdc4ae74418bf7d59aa0a43ce16a84d110eb2f89e7202b172"
  mr_signer = "af68b9679a416f15d846fdf0059a71d11502b2f62ac058fc64b962bad5fd612e"
  isv_prod_id = 0
  isv_svn = 0
  date = "2023-10-16"
  flags = "0400000000000000"
  xfrms = "0300000000000000"
  misc_select = "00000000"
  debug = false
```

# 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
```



```
azureuser4@VM4:~/gsc$ sudo docker run --device=/dev/sgx_enclave \
>    -v /var/run/aesmd/aesm.socket:/var/run/aesmd/aesm.socket \
>    gsc-ubuntu20.04-bash -c ls
Gramine is starting. Parsing TOML manifest file, this may take some time...
-----------------------------------------------------------------------------------------------------------
Gramine detected the following insecure configurations:

  - loader.insecure__use_cmdline_argv = true   (forwarding command-line args from untrusted host to the app)

Gramine will continue application execution, but this configuration must not be used in production!
-----------------------------------------------------------------------------------------------------------

bin
boot
dev
etc
gramine
home
lib
lib32
lib64
libx32
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
```

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

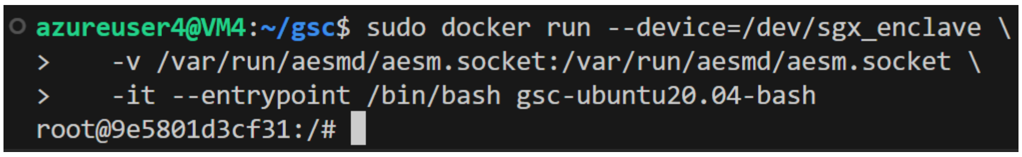
To run the Ubuntu 20.04 Bash container, use the following command:

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



```
azureuser4@VM4:~/gsc$ sudo 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
root@9e5801d3cf31:/#
```

## 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