Containerization in Modern Scientific Applications

Nils Wentzell
Associate Data Scientist, CCQ

October 24th, 2018

Dylan Simon

FFTW

boost
C++ LIBRARIES

HDF

HDF5

Triqs

H5Py

NumPy

C++17

Python

SciPy

CMake

Mpi4Py

LLVM

Docker

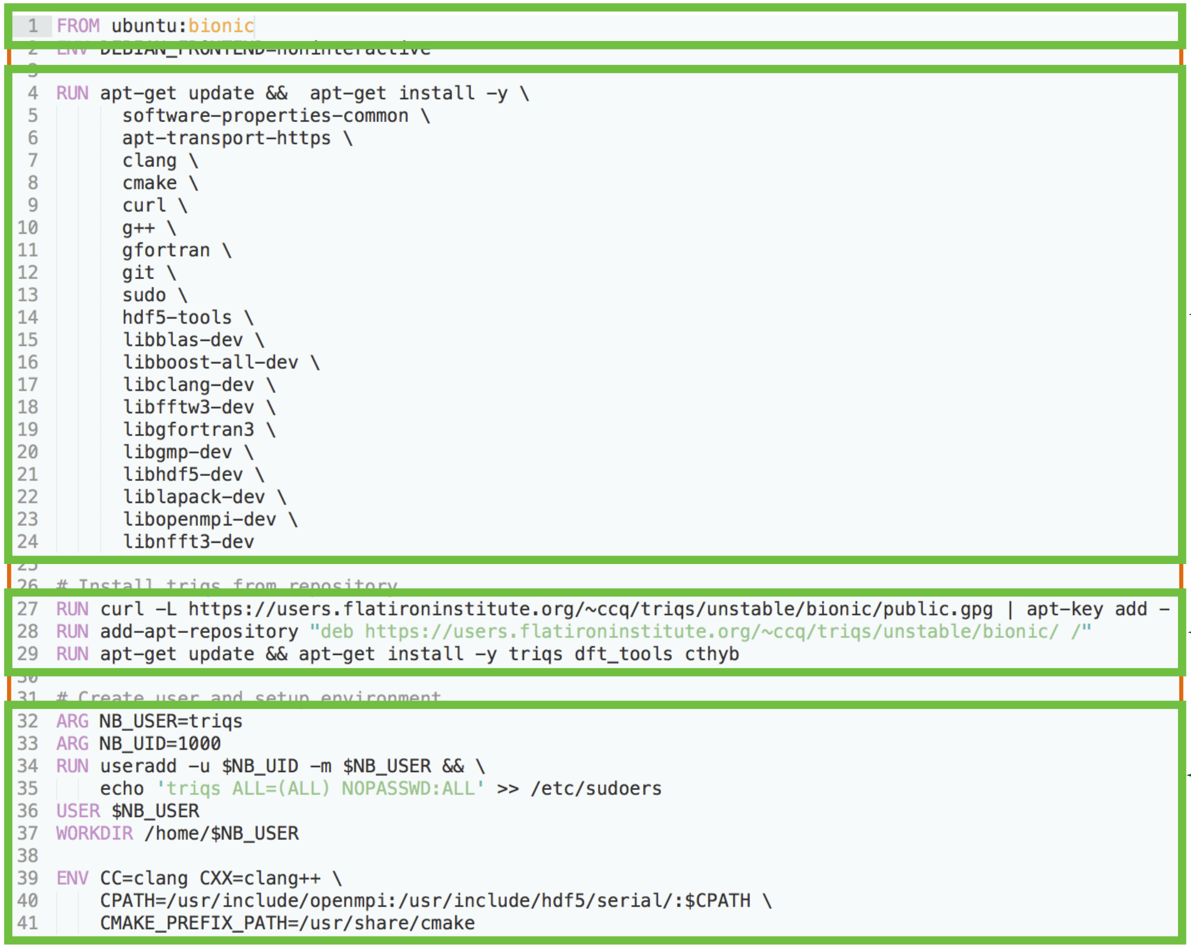
Singularity

- **Open-Source** Tools for Kernel-level **Containerization**
- **Native Performance** of Host System
- Embed Applications in a flexible **Linux** Environment
- **Package** and **Share** Applications easily as Images
- Public **Image Repositories**

Docker	Singularity
Commercial & Non-Profit Userbase	Targeted at **Scientific Computing**
Available for **Linux, Mac & Windows**	**Linux** Only
Docker-Daemon runs **as Root**	Singularity runs **as User**
Shared-Memory Parallelism	Native Support for **OpenMPI**
Mostly **Encapsulated** Environment	Seamless **Integration** with Host

Workflow

Image-Recipe



```
FROM ubuntu:bionic
ENV DEBIAN_FRONTEND=noninteractive

RUN apt-get update &&  apt-get install -y \
      software-properties-common \
      apt-transport-https \
      clang \
      cmake \
      curl \
      g++ \
      gfortran \
      git \
      sudo \
      hdf5-tools \
      libblas-dev \
      libboost-all-dev \
      libclang-dev \
      libfftw3-dev \
      libgfortran3 \
      libgmp-dev \
      libhdf5-dev \
      liblapack-dev \
      libopenmpi-dev \
      libnfft3-dev

# Install triqs from repository
RUN curl -L https://users.flatironinstitute.org/~ccq/triqs/unstable/bionic/public.gpg | apt-key add -
RUN add-apt-repository "deb https://users.flatironinstitute.org/~ccq/triqs/unstable/bionic/ /"
RUN apt-get update && apt-get install -y triqs dft_tools cthyb

# Create user and setup environment
ARG NB_USER=triqs
ARG NB_UID=1000
RUN useradd -u $NB_UID -m $NB_USER && \
    echo 'triqs ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers
USER $NB_USER
WORKDIR /home/$NB_USER

ENV CC=clang CXX=clang++ \
    CPATH=/usr/include/openmpi:/usr/include/hdf5/serial/:$CPATH \
    CMAKE_PREFIX_PATH=/usr/share/cmake
```

Pick a base Distribution

Install Dependencies

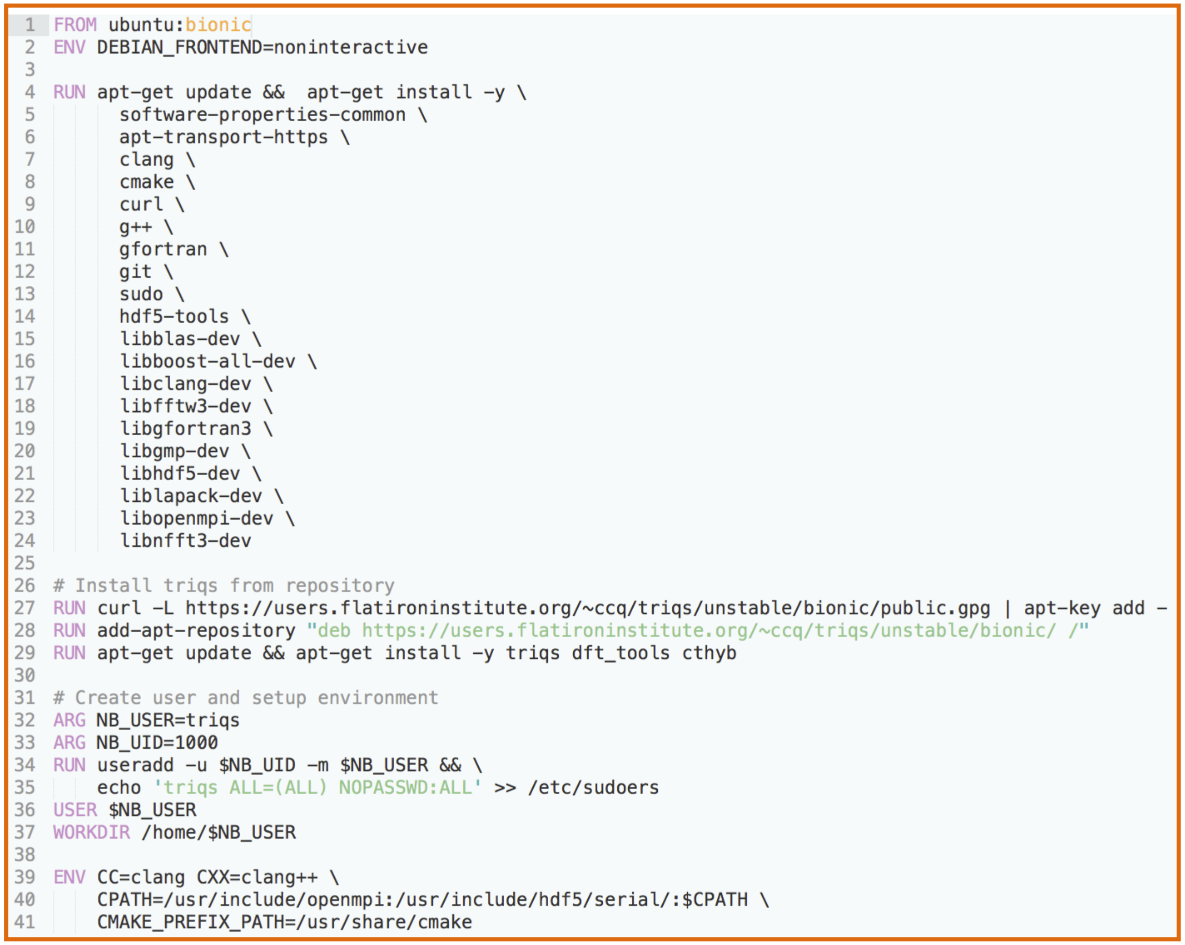
Setup your Application

Setup Environment

FLATIRON INSTITUTE

Workflow

Image-Recipe



```
FROM ubuntu:bionic
ENV DEBIAN_FRONTEND=noninteractive

RUN apt-get update &&  apt-get install -y \
      software-properties-common \
      apt-transport-https \
      clang \
      cmake \
      curl \
      g++ \
      gfortran \
      git \
      sudo \
      hdf5-tools \
      libblas-dev \
      libboost-all-dev \
      libclang-dev \
      libfftw3-dev \
      libgfortran3 \
      libgmp-dev \
      libhdf5-dev \
      liblapack-dev \
      libopenmpi-dev \
      libnfft3-dev

# Install triqs from repository
RUN curl -L https://users.flatironinstitute.org/~ccq/triqs/unstable/bionic/public.gpg | apt-key add -
RUN add-apt-repository "deb https://users.flatironinstitute.org/~ccq/triqs/unstable/bionic/ /"
RUN apt-get update && apt-get install -y triqs dft_tools cthyb

# Create user and setup environment
ARG NB_USER=triqs
ARG NB_UID=1000
RUN useradd -u $NB_UID -m $NB_USER && \
    echo 'triqs ALL=(ALL) NOPASSWD:ALL' >> /etc/sudoers
USER $NB_USER
WORKDIR /home/$NB_USER

ENV CC=clang CXX=clang++ \
    CPATH=/usr/include/openmpi:/usr/include/hdf5/serial/:$CPATH \
    CMAKE_PREFIX_PATH=/usr/share/cmake
```

Build

Binary Image

Share

```
docker build -t my_image -f my_recipe

singularity build my_image my_recipe
```

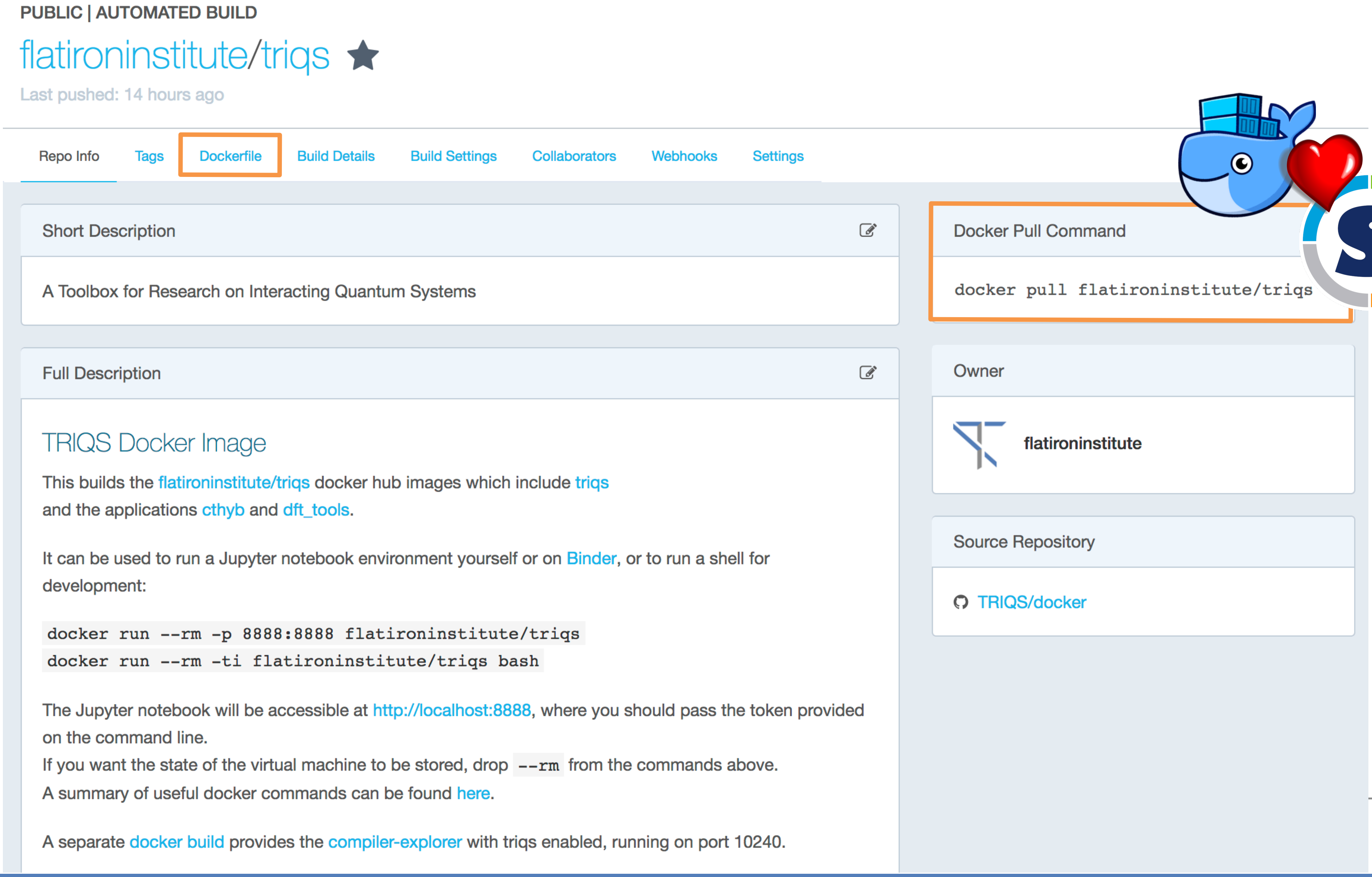
SINGULARITYHUB

Docker Hub

Container Library

FLATIRON INSTITUTE

hub.docker.com/r/flatironinstitute/triqs



PUBLIC | AUTOMATED BUILD

flatironinstitute/triqs ★

Last pushed: 14 hours ago

Short Description

A Toolbox for Research on Interacting Quantum Systems

Docker Pull Command

```
docker pull flatironinstitute/triqs
```

Owner

flatironinstitute

Source Repository

TRIQS/docker

Full Description

TRIQS Docker Image

This builds the flatironinstitute/triqs docker hub images which include triqs and the applications cthyb and dft_tools.

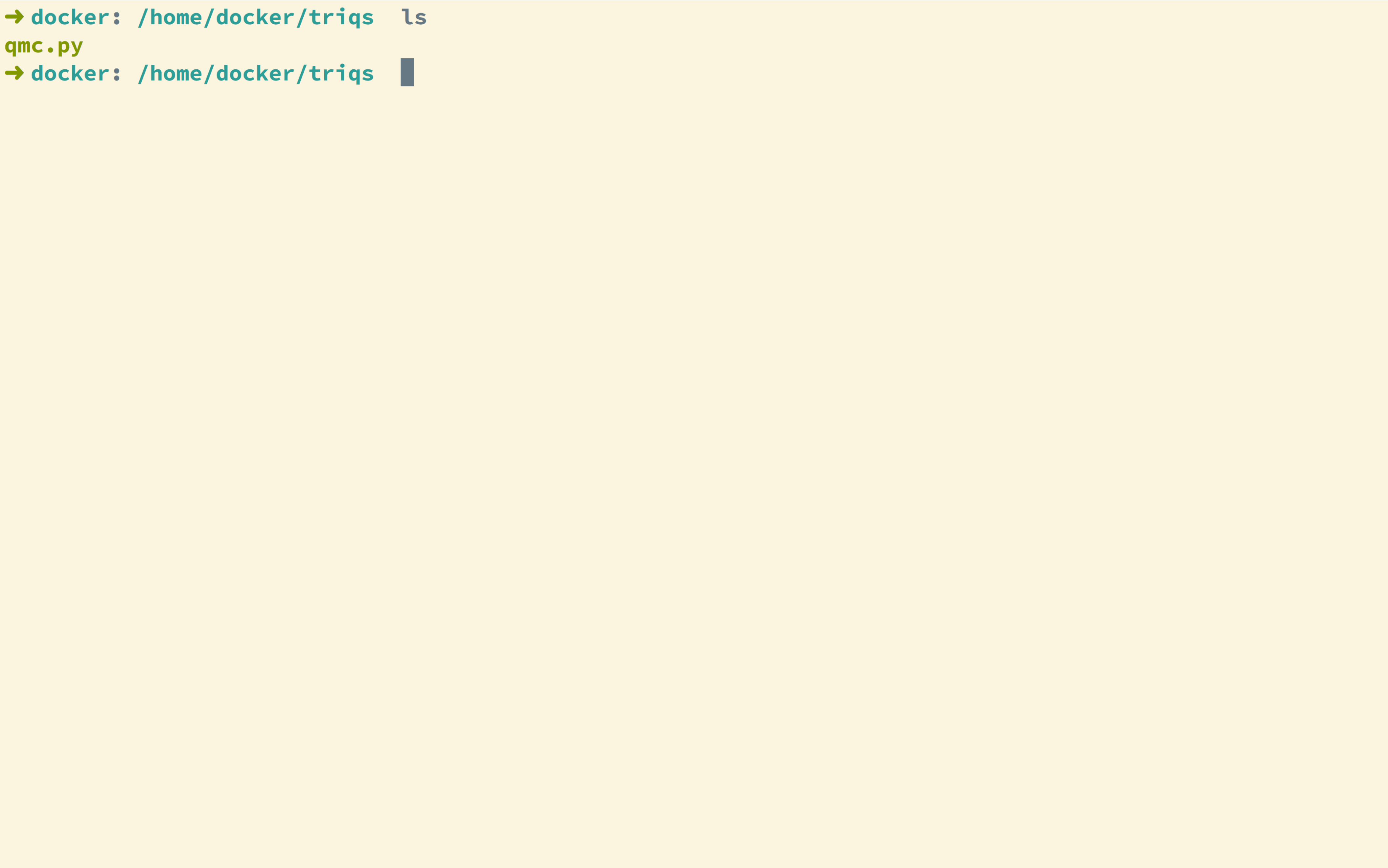
It can be used to run a Jupyter notebook environment yourself or on Binder, or to run a shell for development:

```
docker run --rm -p 8888:8888 flatironinstitute/triqs
docker run --rm -ti flatironinstitute/triqs bash
```

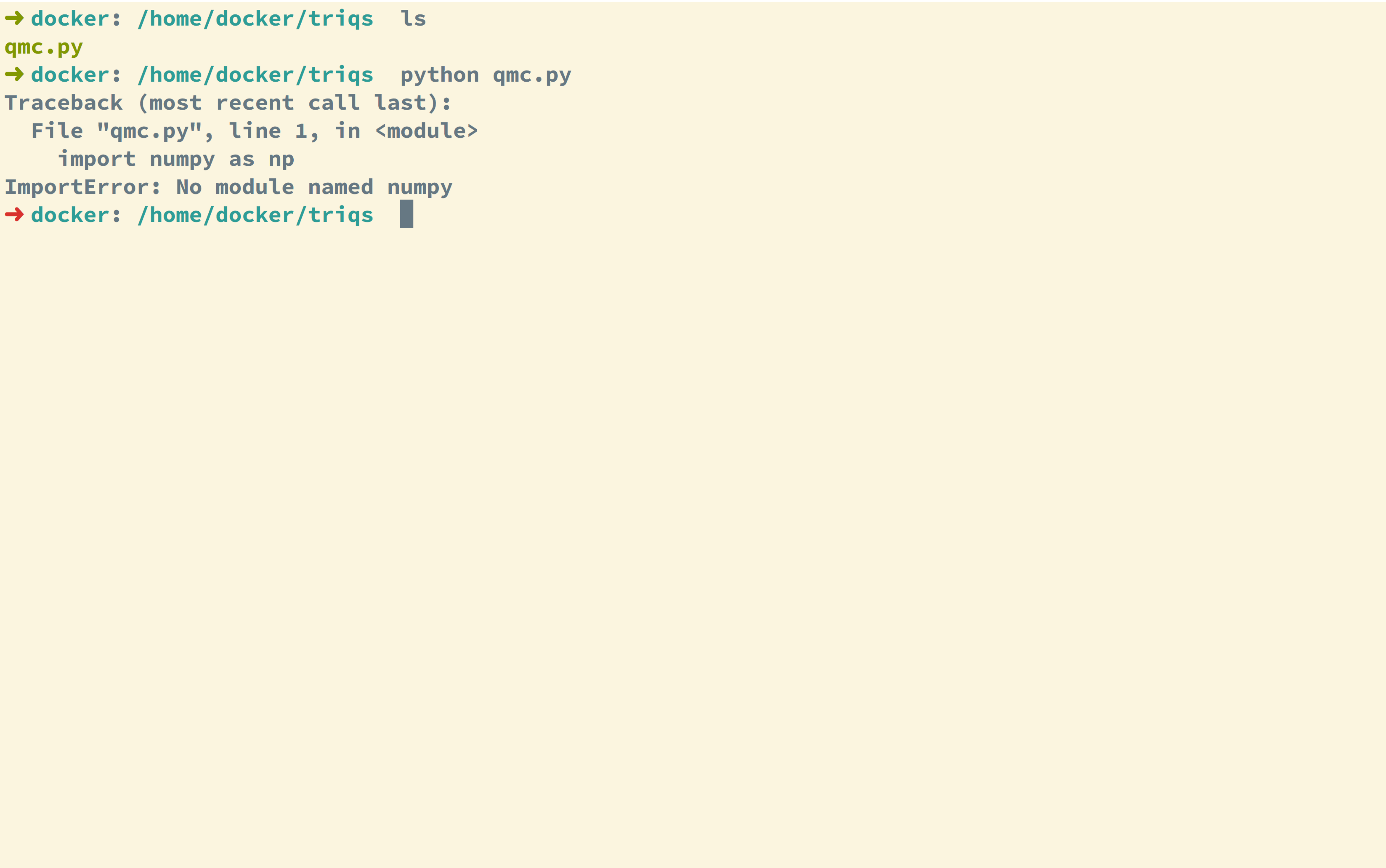
The Jupyter notebook will be accessible at http://localhost:8888, where you should pass the token provided on the command line.
If you want the state of the virtual machine to be stored, drop `--rm` from the commands above.
A summary of useful docker commands can be found here.

A separate docker build provides the compiler-explorer with triqs enabled, running on port 10240.

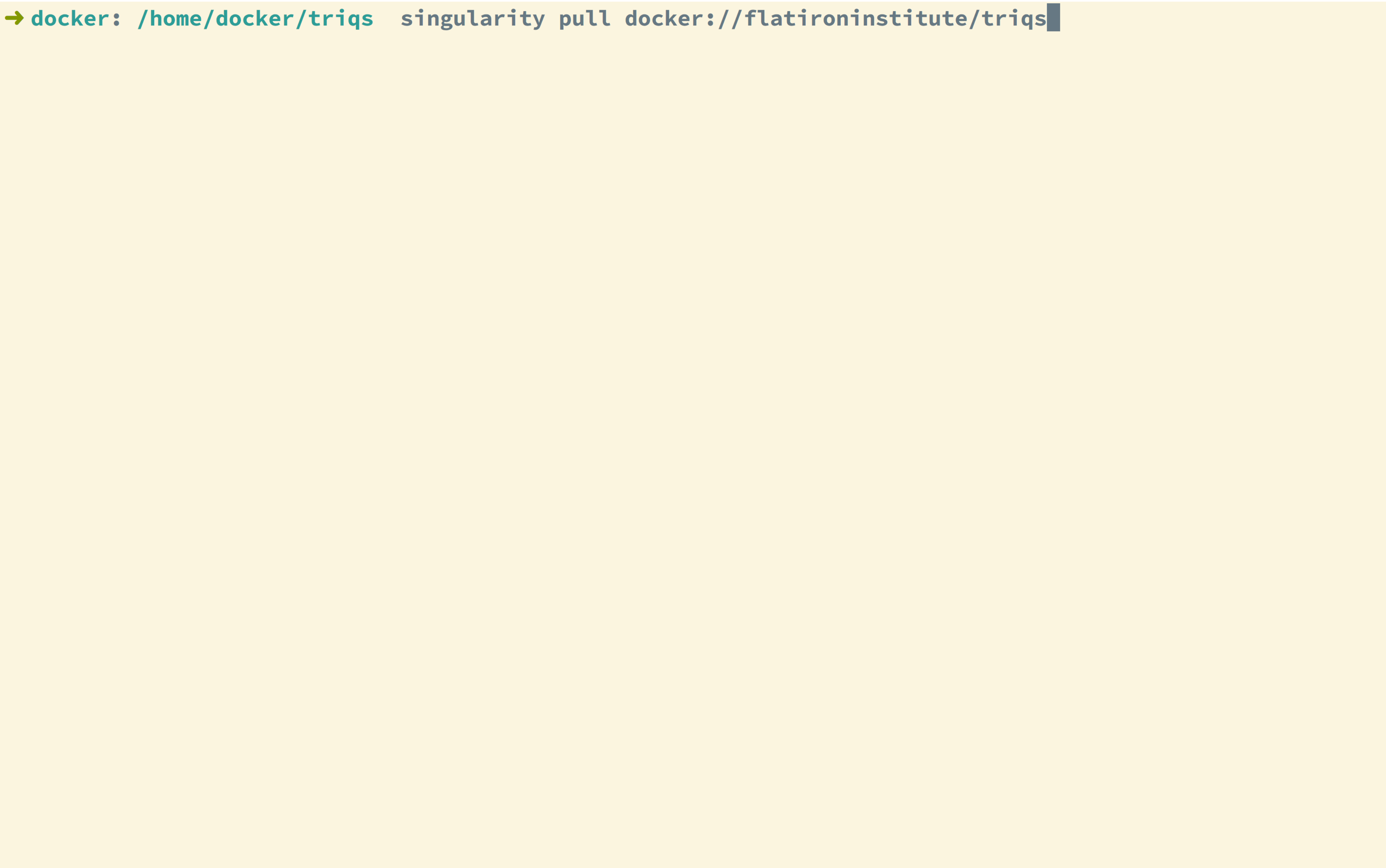
N
TE



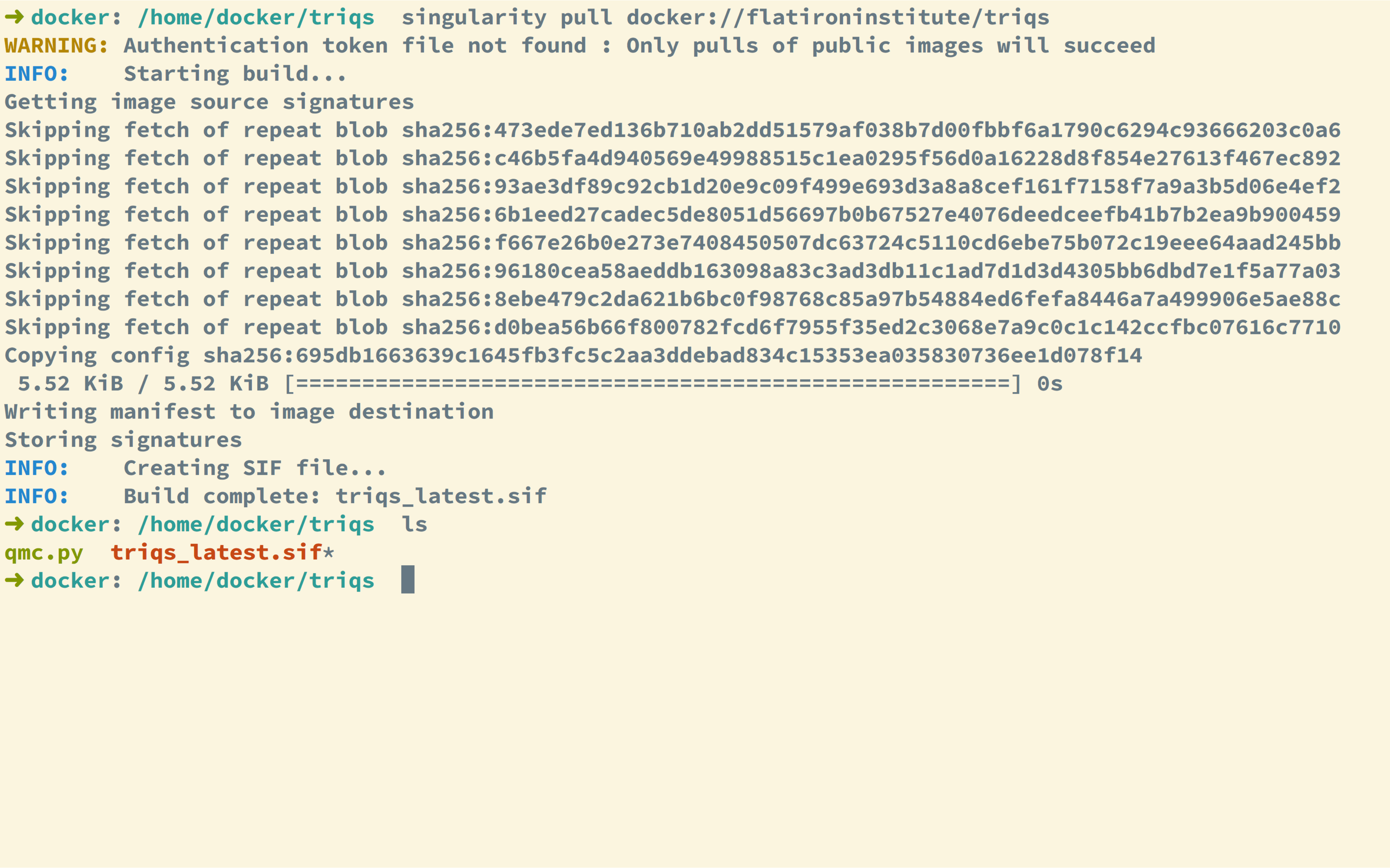
```
➜ docker: /home/docker/triqs  ls
qmc.py
➜ docker: /home/docker/triqs
```



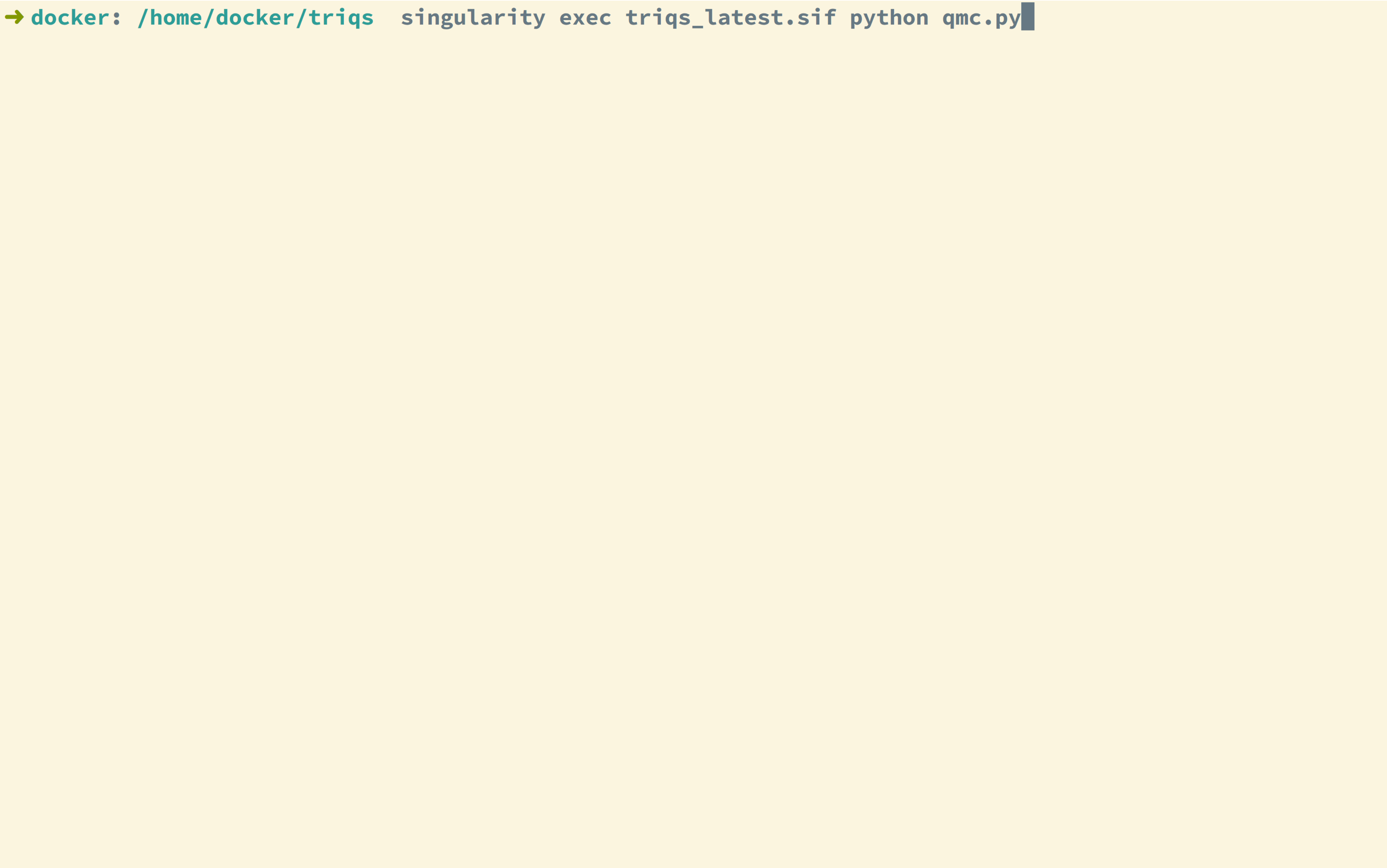
```
➜ docker: /home/docker/triqs  ls
qmc.py
➜ docker: /home/docker/triqs  python qmc.py
Traceback (most recent call last):
  File "qmc.py", line 1, in <module>
    import numpy as np
ImportError: No module named numpy
➜ docker: /home/docker/triqs  █
```



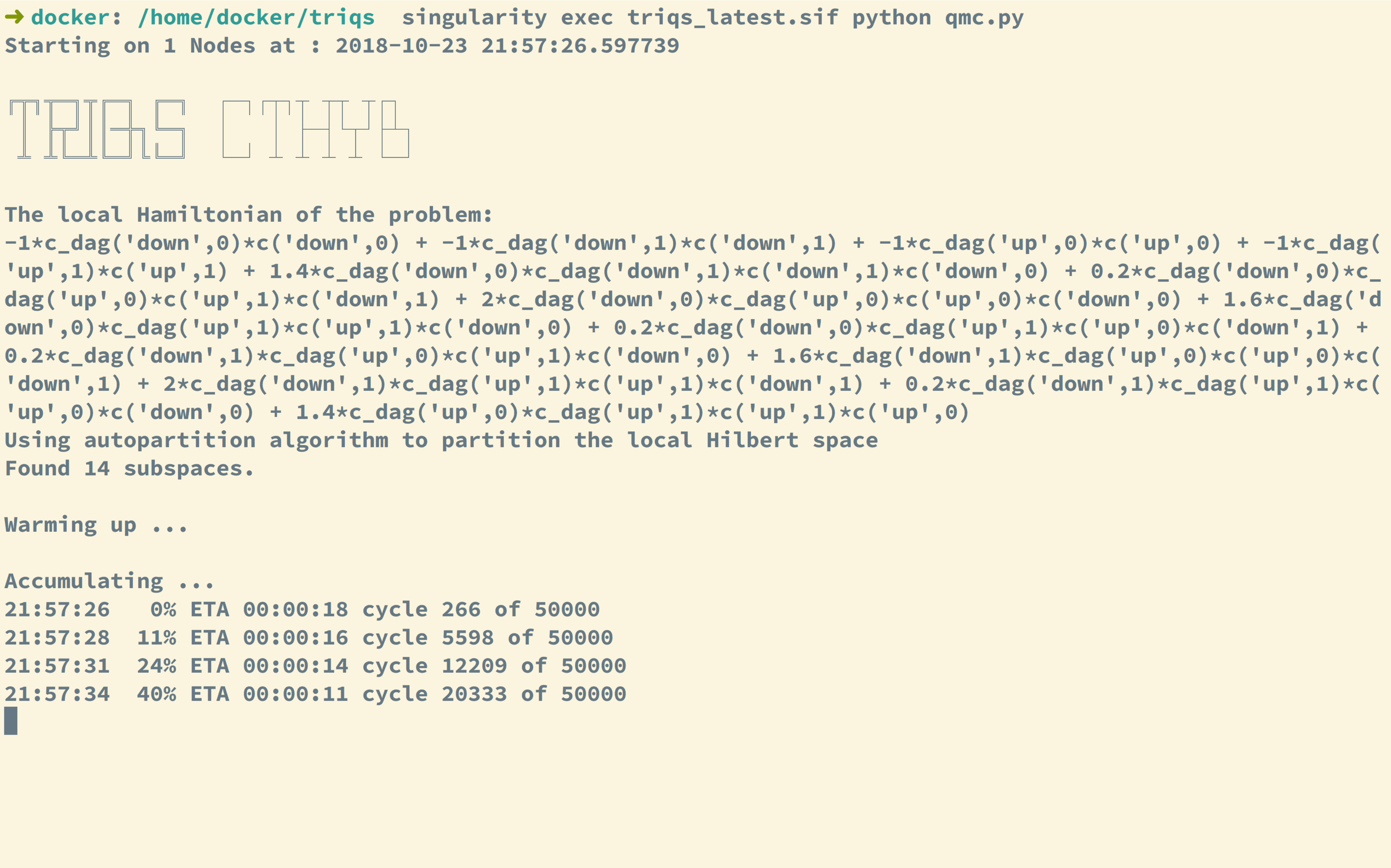
```
➜ docker: /home/docker/triqs  singularity pull docker://flatironinstitute/triqs
```



```
➜ docker: /home/docker/triqs  singularity pull docker://flatironinstitute/triqs
WARNING: Authentication token file not found : Only pulls of public images will succeed
INFO:    Starting build...
Getting image source signatures
Skipping fetch of repeat blob sha256:473ede7ed136b710ab2dd51579af038b7d00fbbf6a1790c6294c93666203c0a6
Skipping fetch of repeat blob sha256:c46b5fa4d940569e49988515c1ea0295f56d0a16228d8f854e27613f467ec892
Skipping fetch of repeat blob sha256:93ae3df89c92cb1d20e9c09f499e693d3a8a8cef161f7158f7a9a3b5d06e4ef2
Skipping fetch of repeat blob sha256:6b1eed27cadec5de8051d56697b0b67527e4076deedceefb41b7b2ea9b900459
Skipping fetch of repeat blob sha256:f667e26b0e273e7408450507dc63724c5110cd6ebe75b072c19eee64aad245bb
Skipping fetch of repeat blob sha256:96180cea58aeddb163098a83c3ad3db11c1ad7d1d3d4305bb6dbd7e1f5a77a03
Skipping fetch of repeat blob sha256:8ebe479c2da621b6bc0f98768c85a97b54884ed6fefa8446a7a499906e5ae88c
Skipping fetch of repeat blob sha256:d0bea56b66f800782fcd6f7955f35ed2c3068e7a9c0c1c142ccfbc07616c7710
Copying config sha256:695db1663639c1645fb3fc5c2aa3ddebad834c15353ea035830736ee1d078f14
 5.52 KiB / 5.52 KiB [======================================================] 0s
Writing manifest to image destination
Storing signatures
INFO:    Creating SIF file...
INFO:    Build complete: triqs_latest.sif
➜ docker: /home/docker/triqs  ls
qmc.py  triqs_latest.sif*
➜ docker: /home/docker/triqs  
```



```
➜ docker: /home/docker/triqs  singularity exec triqs_latest.sif python qmc.py
```

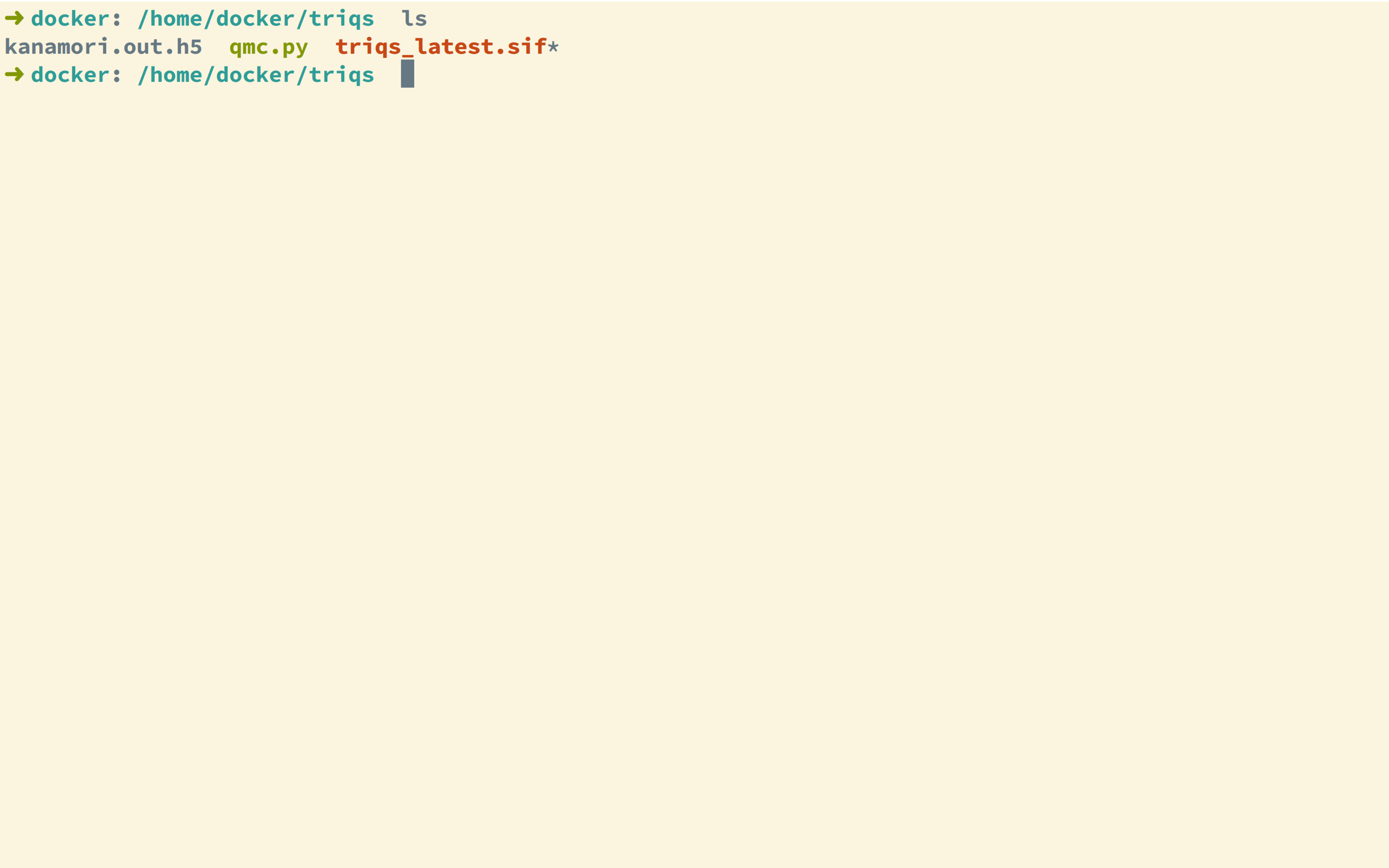


```
➜ docker: /home/docker/triqs  singularity exec triqs_latest.sif python qmc.py
Starting on 1 Nodes at : 2018-10-23 21:57:26.597739

The local Hamiltonian of the problem:
-1*c_dag('down',0)*c('down',0) + -1*c_dag('down',1)*c('down',1) + -1*c_dag('up',0)*c('up',0) + -1*c_dag(
'up',1)*c('up',1) + 1.4*c_dag('down',0)*c_dag('down',1)*c('down',1)*c('down',0) + 0.2*c_dag('down',0)*c_
dag('up',0)*c('up',1)*c('down',1) + 2*c_dag('down',0)*c_dag('up',0)*c('up',0)*c('down',0) + 1.6*c_dag('d
own',0)*c_dag('up',1)*c('up',1)*c('down',0) + 0.2*c_dag('down',0)*c_dag('up',1)*c('up',0)*c('down',1) +
0.2*c_dag('down',1)*c_dag('up',0)*c('up',1)*c('down',0) + 1.6*c_dag('down',1)*c_dag('up',0)*c('up',0)*c(
'down',1) + 2*c_dag('down',1)*c_dag('up',1)*c('up',1)*c('down',1) + 0.2*c_dag('down',1)*c_dag('up',1)*c(
'up',0)*c('down',0) + 1.4*c_dag('up',0)*c_dag('up',1)*c('up',1)*c('up',0)
Using autopartition algorithm to partition the local Hilbert space
Found 14 subspaces.

Warming up ...

Accumulating ...
21:57:26   0% ETA 00:00:18 cycle 266 of 50000
21:57:28  11% ETA 00:00:16 cycle 5598 of 50000
21:57:31  24% ETA 00:00:14 cycle 12209 of 50000
21:57:34  40% ETA 00:00:11 cycle 20333 of 50000
```

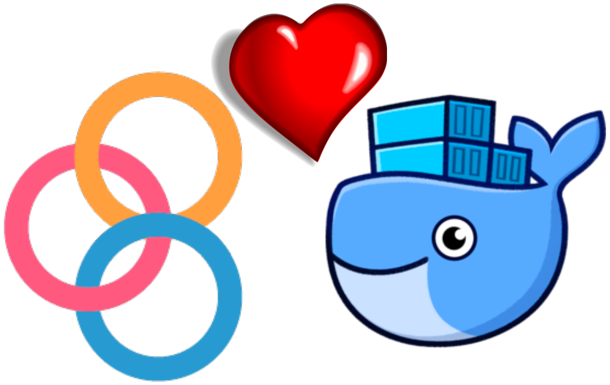


```
➜ docker: /home/docker/triqs  ls
kanamori.out.h5  qmc.py  triqs_latest.sif*
➜ docker: /home/docker/triqs
```

Binder

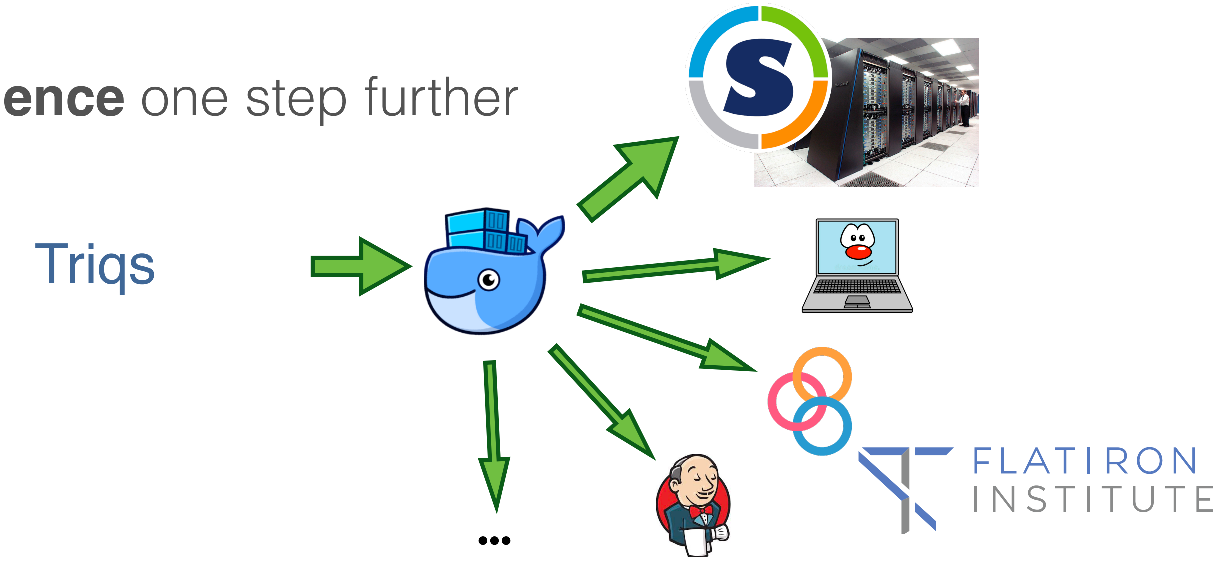
triqs.github.io/notebook

- Create and Host **Jupyter Notebook** Environments
- Great **Integration with Docker** Images
- Use the **TRIQS** Jupyter Notebook without installation!



Conclusions

- Powerful Tools for Kernel-level **Containerization**
- **Package & Share** your Application **with all Dependencies**
- Package **Recent Compilers** and use **Modern Language Features**
- Take **Reproducible Science** one step further



Triqs

TRIQS Example

```python
from pytriqs.gf import *
from pytriqs.archive import *
from pytriqs.plot.mpl_interface import *

beta = 1.0  # Inverse Temperature
niw  = 100  # Number of Matsubara Frequencies

# Initialize the Matsubara Green Function
iw_mesh = MeshImFreq(beta, 'Fermion', niw)
g = Gf(mesh=iw_mesh, target_shape=(1,1))
for iw in g.mesh:
    g[iw] = 1/(iw - 3)

# Store in HDF5 File
with HDFArchive('g.h5', 'w') as F:
    F['g'] = g

# Plot the Result
oplot(g, name='g')
plt.savefig('plot.png')
plt.show()
```

```
Usage:
  singularity [global options...]

Description:
  Singularity containers provide an application virtualization layer enabling
  mobility of compute via both application and environment portability. With
  Singularity one is capable of building a root file system that runs on any
  other Linux system where Singularity is installed.

Options:
  -d, --debug              print debugging information (highest verbosity)
  -h, --help               help for singularity
  -q, --quiet              suppress normal output
  -s, --silent             only print errors
  -t, --tokenfile string   path to the file holding your sylabs
                           authentication token (default
                           "/home/docker/.singularity/sylabs-token")
  -v, --verbose            print additional information
      --version            version for singularity

Available Commands:
  build       Build a new Singularity container
  capability  Manage Linux capabilities on containers
  exec        Execute a command within container
  help        Help about any command
  inspect     Display metadata for container if available
  instance    Manage containers running in the background
  keys        Manage OpenPGP key stores
  pull        Pull a container from a URI
  push        Push a container to a Library URI
  run         Launch a runscript within container
  run-help    Display help for container if available
  search      Search the library
  shell       Run a Bourne shell within container
  sign        Attach cryptographic signatures to container
  test        Run defined tests for this particular container
  verify      Verify cryptographic signatures on container
  version     Show application version
```