Ship Python/Abaqus applications to HPC

Simone Poncioni

29 May 2024

When building python code for research, reproducibility is paramount. The best way to ensure reproducibility and portability is to build a container with all the dependencies and the code. In this tutorial, I will show how to build a container with a Python application and ship it to Docker Hub, and subsequently build an Apptainer image that meets our needs at the University of Bern and its HPC cluster Ubelix.

1. Build a container with your Python application

Similarly to what done `Dockerfile.ubuntu24.04.ifort` (you can find it at the bottom of the page), we first need to create a Dockerfile. A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Using docker build users can create an automated build that executes several command-line instructions in succession. In this case:

BASE IMAGE SETTINGS

```
FROM ubuntu:24.04                          # select a compatible base image

ENV DEBIAN_FRONTEND noninteractive         # set environment variables (non-
   interactive means no prompts during installation)
ENV LC_ALL=en_US.UTF-8                     # ! set these locales to avoid issues
   with Ubelix/Rocky Linux 9
ENV LANG=en_US.UTF-8
```

IFORT COMPILER INSTALLATION

```
# install ifort compiler
RUN wget -O- https://apt.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-
   PRODUCTS.PUB | gpg --dearmor | sudo tee /usr/share/keyrings/oneapi-archive-
   keyring.gpg > /dev/null
RUN echo "deb [signed-by=/usr/share/keyrings/oneapi-archive-keyring.gpg] https
   ://apt.repos.intel.com/oneapi all main" | sudo tee /etc/apt/sources.list.d/
   oneAPI.list
RUN sudo apt update && sudo apt upgrade -y
RUN sudo apt-get install -y intel-basekit intel-hpckit
RUN /bin/bash -c "source /opt/intel/oneapi/setvars.sh"
RUN unset PYTHONPATH
```

And so on. Follow basic recommendations here and here to build an image suitable for your needs. Once the Dockerfile is ready, build the image with the following command:

```
cd <path/to/dockerfile>
docker build -f dockerfile.name -t tag/release-version .
```

You can also add build arguments to the Dockerfile. For example, the username and the token to a private repository on Docker Hub:

```
cd <path/to/dockerfile>
docker build -f --build-arg USERNAME=<username> -t tag/release-version .
```

You can create an account on Docker Hub and push the image to your repository. This way, you can share the image with collaborators or use it on different machines. To push the image to Docker Hub, follow these steps:

1. Log in to Docker Hub:

```
docker login
```

2. Tag the image with your Docker Hub username and the repository name:

```
docker tag tag/release-version <username>/<repository-name:tag/release-version>
```

3. Push the image to Docker Hub:

```
docker push <username>/<repository-name:tag/release-version>
```

Amazing! Now you have built your container with all the dependecies and the code. You can share it with collaborators or use it on different machines. In the next section, I will show how to build an Apptainer image with our needs at the University of Bern and its Ubelix HPC cluster.

2. Build an Apptainer image

Building the image is relatively straightforward once you have put in place an image on Docker Hub. Login to Ubelix and open a bash session on a node with:

```
<submit-node> srun --time=01:00:00 --mem-per-cpu=10G --pty bash
```

Then, follow these steps:

Interactive build

```
<node> mkdir -p <path/to/apptainer>
<node> cd <path/to/apptainer>
export APPTAINER_BINDPATH="$HOME/:$HOME/" # more details on bind directories
    below
<node> apptainer build --force singularity_image_name.sif docker://username/
    imagename:tag
```

Bind directories

Per default the started application runs withing the container. The container works like a separate machine with own operation system etc. Thus, per default you have no access to files and directories outside the container. This can be changed using binding paths.

If files are needed outside the container, e.g. in your HOME you can add the path to APPTAINER_BINDPATH="src1[:dest1],src2[:d
All subdirectories and files will be accessible. Thus you could bind your HOME directory as:

```
export APPTAINER_BINDPATH="$HOME/:$HOME/"
# or simply
export APPTAINER_BINDPATH="$HOME"
```

Automated build

Great! Tipically, one hour should be enough to build the image. If you don't want to build it interactively, you can submit a job with the following script:

```
# Job name
#SBATCH --job-name="build_container"

# Runtime and memory
#SBATCH --time=00:40:00
#SBATCH --cpus-per-task=10
```

```
#SBATCH --mem-per-cpu=4G
#SBATCH --tmp=64G

# Workdir
#SBATCH --chdir=/storage/workspaces/artorg_msb/hpc_abaqus/poncioni/apptainer
#SBATCH --out=%x.out
#SBATCH --err=%x.err


# Run command
# '--force' flag is used to overwrite the image if it already exists
srun apptainer build --force singularity_image_name.sif docker://username/
    imagename:tag
```

Further documentation can be found here and here.

3. Run the container

Our use case is to run a Python application within the container, but which can communicate with the host machine (e.g. to read and write files, or run Abaqus). To do so, remember to bind the directories as shown above. Following this step, the container can be run with the following command (or with a job script):

Run the container interactively

```
# Run command
srun apptainer exec /storage/workspaces/artorg_msb/hpc_abaqus/poncioni/
    apptainer/hfe_development_ifort.sif \
/bin/bash -c "source␣/opt/intel/oneapi/setvars.sh␣&&␣source␣/opt/miniconda/etc/
    profile.d/conda.sh␣&&␣conda␣activate␣hfe-essentials␣&&␣python␣abq_submit.py"
```

Where:

```
srun apptainer exec # runs the container

/storage/workspaces/artorg_msb/hpc_abaqus/poncioni/apptainer/
    hfe_development_ifort.sif # path to the container

/bin/bash -c # opens a bash session within the container
source /opt/intel/oneapi/setvars.sh # sources the ifort compiler settings
&& # ...within the same bash session...
source /opt/miniconda/etc/profile.d/conda.sh && conda activate hfe-essentials #
     sources the conda environment and activates it
&& # ...within the same bash session...
python abq_submit.py # runs the Python application
```

Run the container with a job script

Alternatively, you can run the container with a job script. This way, you can submit the job to the Ubelix HPC cluster and run the container in the background. Here is an example of a job script:

```
#!/bin/bash

# User info
#SBATCH --mail-user=user.name@unibe.ch
#SBATCH --mail-type=begin,end,fail

# Job name
#SBATCH --job-name="job_name"
```

```
# Runtime and memory
#SBATCH --time=00:30:00
#SBATCH --cpus-per-task=10
#SBATCH --mem-per-cpu=4G
#SBATCH --tmp=64G

# Set Workdir
#SBATCH --chdir=/storage/workspaces/artorg_msb/hpc_abaqus/path/to/your/
    application
#SBATCH --output=out/hfe_%A_%a.out
#SBATCH --error=out/hfe_%A_%a.err


#
    ################################################################################

### Load modules and Workspace

HPC_WORKSPACE=hpc_abaqus module load Workspace

unset SLURM_GTIDS

# Run command
srun apptainer exec /storage/workspaces/artorg_msb/hpc_abaqus/poncioni/
    apptainer/hfe_development_ifort.sif \
/bin/bash -c "source␣/opt/intel/oneapi/setvars.sh␣&&␣source␣/opt/miniconda/etc/
    profile.d/conda.sh␣&&␣conda␣activate␣hfe-essentials␣&&␣python␣abq_submit.py"
```

Good job! This way, you have built a container with all the dependencies and the code, and shipped it to Docker Hub. You have also built an Apptainer image with your needs at the University of Bern and its Ubelix HPC cluster. You can now run the container interactively or with a job script. This way, you can ensure reproducibility and portability of your code, no matter where you are, and which system is hosting your code.

Taking it one step further with `Github Actions`

Most of the times these steps are iterative and incremental. Who will know in two weeks what the dependencies will be? Thus, it is important to keep track of the changes and the decisions made. This can be done by using a version control system like Git. Github has handy workflows that can be set up for automatic building and deployment of the container. In this section, we will use Github to keep track of our code and its changes.

Imagine this scenario: You are building an application that uses a newly developed Python pipeline/package. You have built a container with all the dependencies and the code, and shipped it to Docker Hub. You have also built an Apptainer image with your needs at the University of Bern and its Ubelix HPC cluster. Two week later, your colleague asks you to add a new feature to the application. You have to add the new feature to the code, rebuild the container, and ship it to Docker Hub. Do you really want to repeat these steps another time manually? No way! Let's use Github to automate the process:

1. Access your Github repository on the web:
2. Click on the "Actions" tab:
3. Click on "New workflow":
4. 'Categories/Continuous integration' -> 'Docker image'

Modify the workflow according to your needs, starting from the following template and this documentation:

```
name: Build Docker Container

on:
```

```
  push:
    branches:
      - main
      - master

jobs:
  docker:
    runs-on: ubuntu-latest
    steps:
      -
        name: Set up QEMU
        uses: docker/setup-qemu-action@v2
      -
        name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2
      -
        name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}
      -
        name: Build and push
        uses: docker/build-push-action@v5
        with:
            context: "{{defaultContext}}:02_CODE"
            file: <Dockerfile.filename>
            push: true
            tags: ${{ secrets.DOCKERHUB_USERNAME }}/<imagename:tag>
```

Now, the next time that you will push your code, Github will make sure to build the container and push it to Docker Hub automatically. If you are working on Ubelix, remember to rebuild the Apptainer image with the latest update!

Example of a Dockerfile for HFE simulations on Ubelix

```
FROM ubuntu:24.04

# Set environment variables
ENV DEBIAN_FRONTEND noninteractive
ENV LC_ALL=en_US.UTF-8
ENV LANG=en_US.UTF-8

# Install dependencies
RUN apt-get update && apt-get install -y \
apt-utils cargo clang clang-tidy cmake \
fonts-cmu ftp gcc gcovr \
git g++ gfortran libboost-dev libcgns-dev \
libfltk1.3-dev libfreetype6-dev libgl1-mesa-dev libgl1-mesa-dri libhdf5-dev \
libocct-data-exchange-dev libocct-foundation-dev libocct-ocaf-dev libopenblas-
    dev libopenmpi-dev \
libpetsc-complex-dev libxfixes-dev libxcursor-dev libxft-dev libxi-dev \
libxinerama-dev libxmu-dev libslepc-complex3.19-dev mesa-common-dev python3-pip
     \
swig tcl-dev tk-dev valgrind vim wget \
sudo locales \
&& apt-get clean
```

```
# Set locales
RUN locale-gen en_US.UTF-8

# install ifort compiler
RUN wget -O- https://apt.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-
    PRODUCTS.PUB | gpg --dearmor | sudo tee /usr/share/keyrings/oneapi-archive-
    keyring.gpg > /dev/null
RUN echo "deb [signed-by=/usr/share/keyrings/oneapi-archive-keyring.gpg] https
    ://apt.repos.intel.com/oneapi all main" | sudo tee /etc/apt/sources.list.d/
    oneAPI.list
RUN sudo apt update && sudo apt upgrade -y
RUN sudo apt-get install -y intel-basekit intel-hpckit
RUN /bin/bash -c "source /opt/intel/oneapi/setvars.sh"
RUN unset PYTHONPATH

# Install miniconda
RUN wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
    -O ~/miniconda.sh && \
bash ~/miniconda.sh -b -p /opt/miniconda && rm ~/miniconda.sh
ENV PATH="/opt/miniconda/bin:$PATH"

# Create a conda environment
RUN conda create -n hfe-essentials python=3.12
SHELL ["conda", "run", "-n", "hfe-essentials", "/bin/bash", "-c"]
# Copy hfe requirements.txt and install requirements
COPY ./requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
RUN pip install -U scikit-image
RUN pip install imutils

# Set GitHub username and personal access token (I am cloning a private
    repository below)
ARG GITHUB_USERNAME
ARG GITHUB_TOKEN
ARG USER_UID=1000
ARG USER_GID=1000

# Install the meshing package
RUN git clone https://${GITHUB_USERNAME}:${GITHUB_TOKEN}@github.com/artorg-
    unibe-ch/spline_mesher.git ./pyhexspline/spline_mesher
WORKDIR ./pyhexspline/spline_mesher
RUN pip install -e .

# Change ownership of the conda environment to the hfe user
RUN useradd -ms /bin/bash hfe
RUN chown -R hfe:hfe /opt/miniconda/envs/hfe-essentials

USER hfe
WORKDIR /home/hfe
RUN mkdir -p ~/.ssh
RUN chmod 700 ~/.ssh
ENV PATH="/opt/cargo/bin:${PATH}"

# conda init, source .bashrc, conda activate
# https://pythonspeed.com/articles/activate-conda-dockerfile/
RUN conda init bash
```

```
RUN echo "conda activate hfe-essentials" >> ~/.bashrc
```