Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

DGX WORKFLOW UTILITIES

2023-03-14 Tue

by

Philipp Denzel

TABLE OF CONTENTS

DGX SERVERS

https://cai.cloudlab.zhaw.ch/index.html

- DGX.cloudlab.zhaw.ch
- DGX2.cloudlab.zhaw.ch
- DGX3.cloudlab.zhaw.ch
- DGX-A100.cloudlab.zhaw.ch

SSH SETUP FOR DGX SERVERS

```
> mkdir -p ~/.ssh && cd ~/.ssh && ssh-keygen -t ed25519 -C "phdenzel@gmail.com"

Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/phdenzel/.ssh/id_ed25519): dummy_dgx_id_ed25519
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in dummy_dgx_id_ed25519
Your public key has been saved in dummy_dgx_id_ed25519.pub
The key fingerprint is:
SHA256:43+n+1hhz/EIRhlDJKJAdl+wqRIpFD2Nq0BlCNJxCbA phdenzel@gmail.com
The keys randomart image is:
+--[ED25519 256]--+
|=oBO=+. o.oo=    |
|.=oo=+.o = . +   |
|E . oo. +   o    |
|.  ... .   .     |
| . .. . S   o o. |
|  .  . . . . o =o|
|        .     o +|
|         .  .o.  |
|          ..+=.  |
+----[SHA256]-----+
```

- Depending on your ssh client setup, I would still recommend setting a password and caching it with you ssh-agent, instead of generating a password-less key.
- On windows, I recommend WSL or the `putty` SSH-client.

Send the public key to DGX

```
> ssh-copy-id -i dummy_dgx_id_ed25519.pub denp@dgx.cloudlab.zhaw.ch
```

- You're then asked for your ZHAW password one last time.
- Alternatively, copy the public key over with scp and add its contents to `authorized_keys`.

SSH config (optional)

```
Host dgx
  Hostname DGX.cloudlab.zhaw.ch
  User denp
  Port 22
  IdentityFile /home/phdenzel/.ssh/dgx_id_ed25519

Host dgx2
  Hostname DGX2.cloudlab.zhaw.ch
  User denp
  Port 22
  IdentityFile /home/phdenzel/.ssh/dgx_id_ed25519

Host dgx3
  Hostname DGX3.cloudlab.zhaw.ch
  User denp
  Port 22
  IdentityFile /home/phdenzel/.ssh/dgx_id_ed25519

Host dgxa100
  Hostname DGX-A100.cloudlab.zhaw.ch
  User denp
  Port 22
  IdentityFile /home/phdenzel/.ssh/dgx_id_ed25519
```

SSH tunneling: binding ports

Create an Bind host port 8888 to dgx's port 53682

```
> ssh -L 8888:localhost:53682 dgx
```

or use dynamic port forwarding (SOCKS proxy on port 1080)

```
> ssh -D 1080 dgx
```

BASIC WORKFLOWS

1. SSH in to the DGX of your choice.
2. Build your docker image.
3. Spawn a (detachable) `screen` session.
4. Reserve resources via SLURM and start a shell
5. Launch a docker container
6. (Optionally) attach to the container interactively and run your job
7. (Detach,) detach, log out.
8. Log in (once the job is done) and clean any residuals (screen sockets, docker containers, etc.)

Apache Airflow workflow

1. Log in to the DGX of your choice.
2. Build your docker image.
3. Create/Amend an Airflow DAG (directed acyclic graph) script in the `dag/` directory
    - Airflow will schedule the tasks of the DAG
    - interface with SLURM
    - launch containers/tasks

DAGs

- define DAG execution scripts: `example_docker.py`

```
from __future__ import annotations

import os
from datetime import datetime

from airflow import models
from airflow.operators.bash import BashOperator
from airflow.providers.docker.operators.docker import DockerOperator

ENV_ID = os.environ.get("SYSTEM_TESTS_ENV_ID")
DAG_ID = "docker_test"

with models.DAG(
    DAG_ID,
    schedule="@once",
    start_date=datetime(2021, 1, 1),
    catchup=False,
    tags=["example", "docker"],
) as dag:
    t1 = BashOperator(task_id="print_date", bash_command="date", dag=dag)
    t2 = BashOperator(task_id="sleep", bash_command="sleep 5", retries=3, dag=dag)
    t3 = DockerOperator(docker_url="unix://var/run/docker.sock",  # Set your docker URL
                        command="/bin/sleep 30",
                        image="centos:latest",
                        network_mode="bridge",
                        task_id="docker_op_tester",
                        dag=dag)
    t4 = BashOperator(task_id="print_hello", bash_command='echo "hello world!!!"', dag=dag)

    (
        t1
        >> [t2, t3]
```

```
    >> t4
)
```

SCREEN

- `screen` allows to create session sockets on a (remote) host
- single SSH connection, multiple terminals

```
# Start a new named screen session:
    screen -S session_name
# Start a new daemon and log the output to `screenlog.x`:
    screen -dmLS session_name command
# Show open screen sessions (outside):
    screen -ls
# Reattach to an open screen:
    screen -r session_name
# Detach from inside a screen:
    Ctrl + A, D
# Show all shortcuts:
    Ctrl + A, ?
# Create a new window inside a session:
    Ctrl + A, c
```

```
# Hsplit/VSplit/unsplit a terminal into multiple frames:

    Ctrl + A, S/|/X

# Cycle window focus:

    Ctrl + A, Tab
```

SLURM

- SLURM is a resource and workload manager
- several program commands (typically prefixed with `s`)
  - `squeue`: inspect batch queue of running and planned jobs
  - `sbatch`: submit a batch script to the queue
  - `srun`: directly submit a resource allocation request and execute an application

I typically use

```
srun --job-name=denp-experiment --pty --ntasks=1 --cpus-per-task=4 --mem=32G --gres=gpu:1 bash
```

DOCKER 101

- What is Docker?
  - Platform for packaging, shipping, and running applications in docker *containers*.
- What is a container?
  - A container encompasses all dependencies and configurations needed to run an application.
    - portable, runnable on different platforms
    - applications are not installed on the OS, but in the (isolated) docker environment
    - made from docker *images*

DOCKER 101

- What are images?
  - Images contain the "instructions" aka *layers* how to create containers.
- Docker vs VMs?
  - OS have three layers (hardware, kernel, and applications), while VMs virtualize the kernel and applications layer (and sometimes even emulate hardware components), docker only abstracts the applications layer and runs on the hosts kernel.
    - docker has to be compatible with the host kernel (or provide further partial abstraction of the kernel)

Basic commands

```
# List all docker containers (running and stopped):
    docker ps --all
# Start a detached container from an image, with a custom name:
    docker run -d --name <container_name> image
# Start or stop an existing container:
    docker start|stop container_name
# Display the list of already downloaded images:
    docker images
# Open a shell inside a running container:
    docker exec -it container_name bash
# Remove a stopped container:
    docker rm container_name
# Remove a cached image:
    docker rmi image_id
# Fetch and stream the logs of a container:
    docker logs -f container_name
```

docker.service

Start docker service (on linux w/ systemd)

```
sudo systemctl start docker.service
```

or enable it on startup (on linux w/ systemd)

```
sudo systemctl enable --now docker.service
```

docker images

List images

```
> docker images

REPOSITORY   TAG                IMAGE ID       CREATED         SIZE
denp/base    latest             87b30d5cd1bc   25 hours ago    1.61GB
python       3.10.9-bullseye    eadf3ec97427   4 weeks ago     917MB
```

docker ps

List (all) containers

```
> docker ps -a

CONTAINER ID   IMAGE              COMMAND       CREATED          STATUS                      PORTS     NAMES
649de6effa65   denp/base:latest   "/bin/bash"   6 seconds ago    Exited (0) 6 seconds ago              xenodoc
79f278c80ae0   denp/base:latest   "/bin/bash"   8 seconds ago    Exited (0) 7 seconds ago              peacefu
4dc58367055c   denp/base:latest   "/bin/bash"   25 seconds ago   Exited (0) 24 seconds ago             quirky_
```

docker build

Build images (and tag them with `<short_name>/<descriptor>`)

```
> docker build -t denp/base base

Sending build context to Docker daemon   2.56kB
Step 1/5 : FROM python:3.10.9-bullseye AS basebuilder
 ---> eadf3ec97427
Step 2/5 : ENV DEBIAN_FRONTEND=noninteractive LANG=C.UTF-8 LC_ALL=C.UTF-8
 ---> Using cache
 ---> a9a75ecd566b
...
...
Removing intermediate container 5c62a569df52
 ---> ce9d93010e84
Successfully built ce9d93010e84
Successfully tagged denp/base:latest
```

docker run

Run a container interactively (and remove on exit)

```
> docker run --rm -it denp/base:latest
```

Run a container detached

```
> docker run -d denp/base:latest
```

Run a container with an attached docker volume (and)

```
> docker run -it -v [volume_name]:[path/in/container] denp/base bash;
```

Run a container with 32G size of tmpfs `/dev/shm`, i.e. RAM

```
> docker run -it --shm-size=32g denp/base bash;
```

Run a container and attach a docker volume

```
> docker run -it -p 8889:8888 denp/base bash;
```

docker volume

- It is possible to bind/mount local volumes to docker containers

List/remove volumes

```
> docker volume ls/rm [volume_name]
```

- To create a volume from an existing file/folder/dataset use

```
docker volume create -o o=bind,ro -o type=none -o device=/cluster/data/<name>/path/to/folder [volume_name]
```

docker network

- Docker uses sandboxed networks
  - on default each container's network is isolated
  - multiple containers can run/communicate in the same network if specified

List/create/remove networks

```
> docker network ls/create/rm [network_name]
```

BEST PRACTICES FOR WRITING DOCKERFILES

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

- Create Dockerimages for the tests
  - 1 container ⟶ 1 test
- We should stick to some conventions to minimize container sizes and overhead

Base images

- `nvidia/cuda:12.0.1-base-ubuntu22.04`
- if cuDNN is needed: `nvidia/cuda:12.0.1-cudnn8-runtime-ubuntu22.04`
- if CUDA compiler is needed: `nvidia/cuda:12.0.1-cudnn8-devel-ubuntu22.04`

Update repositories and clean up

```
RUN apt-get -y update && apt-get install -y \
    build-essential git git-lfs cmake gfortran \
    libboost-all-dev libboost-python-dev libblas-dev liblapacke-dev libfftw3-dev libgsl-dev \
    casacore-dev libcfitsio-dev wcslib-dev \
    && rm -rf /var/lib/apt/lists*/
```

Sort layers according to the change frequency

This will use more resources

```
FROM python:3.10.9-bullseye

WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
CMD ["python", "src/app.py"]
```

compare to

```
FROM python:3.10.9-bullseye

WORKDIR /app
COPY requirements.txt /app
RUN pip install -r requirements.txt
COPY . /app
CMD ["python", "src/app.py"]
```

Multi-stage

- if you need to install libraries just for building dependencies, use multi-stage builds

```
FROM alpine/git:2.36.3 as downloader

# Clone utility script
RUN <<EOF
cat <<'EOE' > /clone.sh
mkdir -p repositories/"$1" && cd repositories/"$1" && git init && git remote add origin "$2" && git fetch orig
EOE
EOF

RUN . /clone.sh ska https://github.com/phdenzel/ska.git 9a5a4411d8c77b4dc9916accce1f0501062d6375 \
    && rm -rf data doc scripts/*.py scripts/*.rs **/*.ipynb

# Build dependencies
FROM python:3.10.9-bullseye AS builder
ENV DEBIAN_FRONTEND=noninteractive \
    LANG=C.UTF-8 \
    LC_ALL=C.UTF-8 \
    PYTHONDONTWRITEBYTECODE=1 \
    PYTHONUNBUFFERED=1

# Install build dependencies
RUN apt-get update && apt-get install -y build-essential git git-lfs cmake gfortran libboost-all-dev libboost-
    && rm -rf /var/lib/apt/lists/*

ENV ROOT=/ska

COPY --from=downloader /git/repositories/ska ${ROOT}
RUN cd ${ROOT} && make && make install
RUN --mount=type=cache,target=/root/.cache/pip \
    pip install -r ${ROOT}/requirements.txt

FROM python:3.10.9-bullseye
```

```
COPY --from=builder ${ROOT}/repositories/ska /ska
COPY . /
ENTRYPOINT ["/ska/docker/entrypoint.sh"]
```