Data & Analysis Preservation: current work items

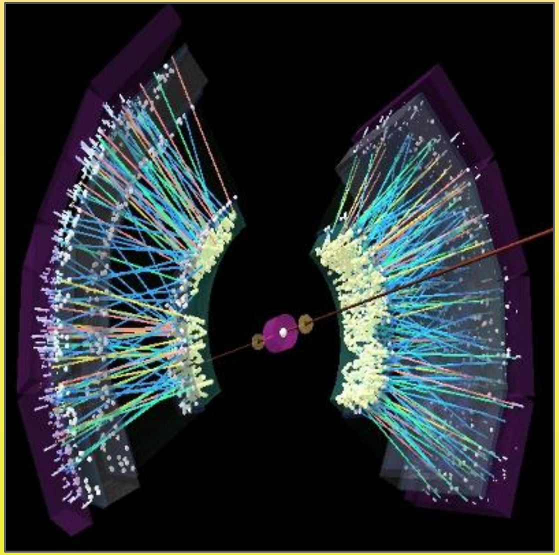
Maxim Potekhin

Nuclear and Particle Physics Software Group

BROOKHAVEN NATIONAL LABORATORY

PHENIX DAP Meeting

01/28/2021



PHENIX

This presentation

- HEPData
- Website updates
- OpenData
- Docker - a working example demonstrating an array of features
- REANA - a few comments; detailed examples deferred for later
- Analysis notes

HEPData

- Ongoing HEPData preparation and management
  - Please see the spreadsheet for items actively developed: https://docs.google.com/spreadsheets/d/1rABxzuM-h9Rukz08ut_m8xnMo0B_J1LKre8bM7B7264/edit?usp=sharing
- Multiple items in the pipeline
- Scientific notation for numbers, vs fixed point - tested, both work
- Need to consider a formal policy regarding precision for data points and errors
- In summary, a healthy level of activity

Website updates

- HEPData workflow diagram updated (i.e. PPG vs IRC)
- Section on HEPData numerical data issues added
- Link to the HEPData workflow spreadsheet added
- General cleanup

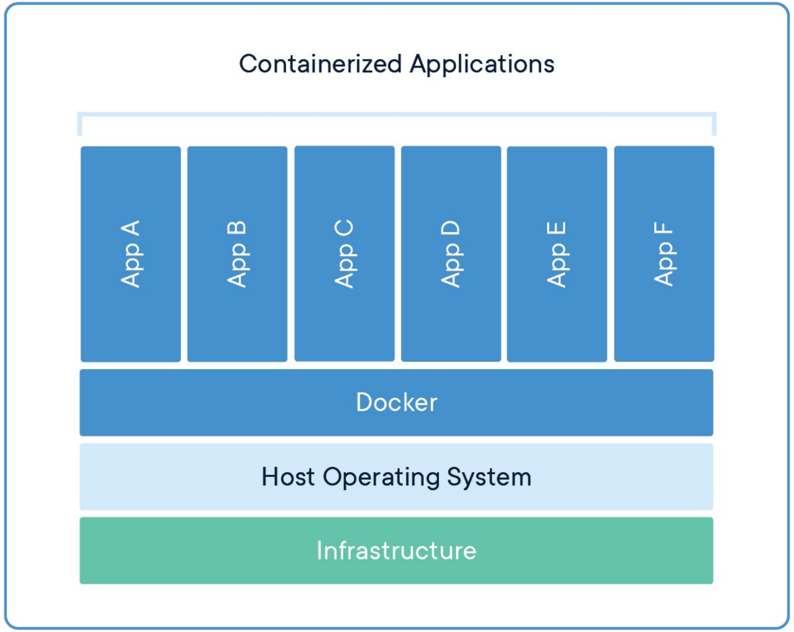
OpenData

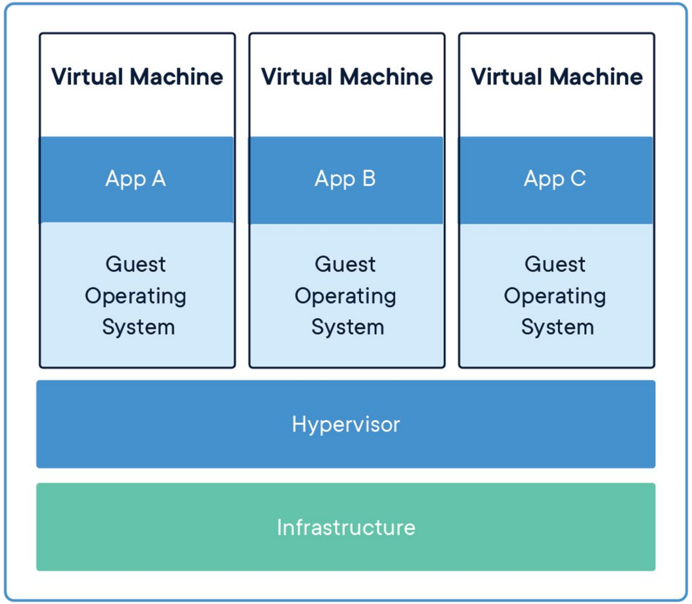
- Semantics of the website include built-in variable descriptions for a specific dataset
  - Implies a fixed “schema” for a single entry e.g. a Ntuple
- If heterogeneous data samples are included they will have to be documented in the textual description
  - Gabor’s entry contains two ntuples
- Still work in progress, TBD (discussing with Gabor)
- Useful learning curve
- Potential for hosting materials suitable for the PHENIX School

Docker/REANA

- Docker/Singularity containerization is already commonplace in EIC
- And now, initial adoption of REANA by the EIC community
    - cf. REANA instance created at U Manitoba
    - Used to run actual MC in a reproducible manner
- Progress with basic REANA testing at BNL
    - Differences in base OS images and implications for REANA under study
    - Modality of AFS usage understood
- In the following, we will review a simple Docker example to illustrate some mechanics
    - Currently there are no working PHENIX examples for Docker, this is a complex task which involves interaction with the BNL infrastructure - file system, databases etc
    - Work in progress

Containers vs VM





- Virtual machines require a “hypervisor” which is responsible for complete emulation of an OS
- However containers share the same OS kernel resulting in more economical storage and better performance
- Containers are made possible by the Linux resources isolation features (control groups and namespaces)

Tech note: namespaces

Docker uses a technology called `namespaces` to provide the isolated workspace called the *container*. When you run a container, Docker creates a set of *namespaces* for that container. These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

Docker Engine uses namespaces such as the following on Linux:

- **The `pid` namespace:** Process isolation (PID: Process ID).
- **The `net` namespace:** Managing network interfaces (NET: Networking).
- **The `ipc` namespace:** Managing access to IPC resources (IPC: InterProcess Communication).
- **The `mnt` namespace:** Managing filesystem mount points (MNT: Mount).
- **The `uts` namespace:** Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

Tech note: control groups

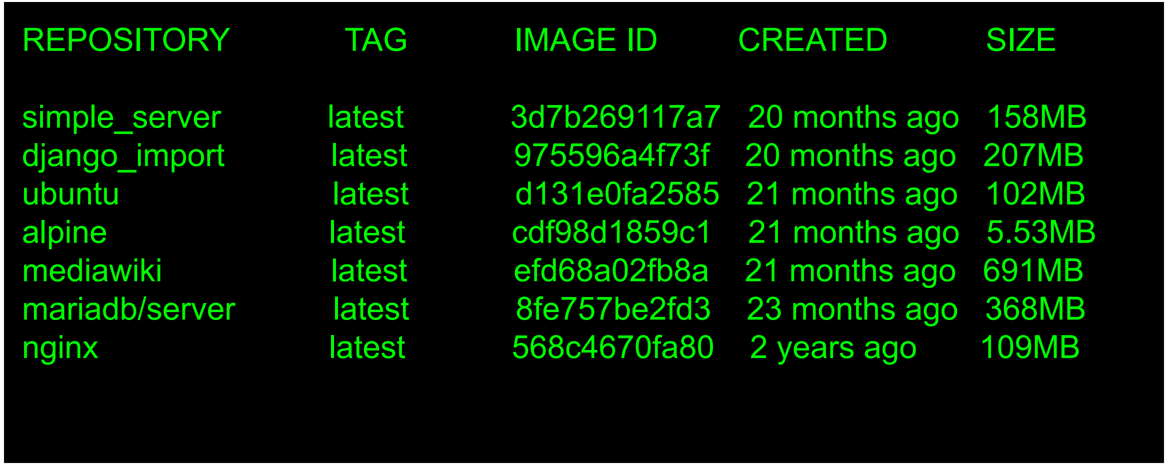
Docker Engine on Linux also relies on a technology called “control groups” (cgroups).

A cgroup limits an application to a specific set of resources. Control groups allow Docker Engine to share available hardware resources to containers and optionally enforce limits and constraints.

In the following, we’ll review a few slides derived from previous material and move on to a new simple but complete example.

Tech note: containers, images, repositories

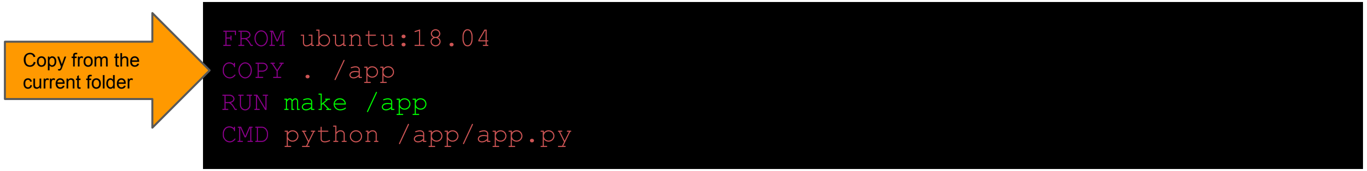
- “Image” is a read-only template residing in storage ⇒ used to create a running process - the “container”
- “Repository” is a storage and access system for images
- Examples:
  - Inspect images on a local machine: “docker image ls”
  - List running containers: “docker container ps”



```
REPOSITORY          TAG          IMAGE ID        CREATED           SIZE

simple_server      latest        3d7b269117a7   20 months ago    158MB
django_import      latest        975596a4f73f   20 months ago    207MB
ubuntu             latest        d131e0fa2585   21 months ago    102MB
alpine             latest        cdf98d1859c1   21 months ago    5.53MB
mediawiki          latest        efd68a02fb8a   21 months ago    691MB
mariadb/server     latest        8fe757be2fd3   23 months ago    368MB
nginx              latest        568c4670fa80   2 years ago      109MB
```

Tech note: building images

- Building images can be conceptualized as adding layers to an underlying image
- The Docker daemon orchestrates the build, the instructions are in the “Dockerfile”
- The build process can contain virtually anything - compiling libraries, installing Python packages, adding configuration and data files etc.
- Ultimately, the image contains all the artifacts created during the build
- “COPY” command works at build time and copies files or group of files to the internal file system of the container
- “RUN” defines an action to be performed when building an image



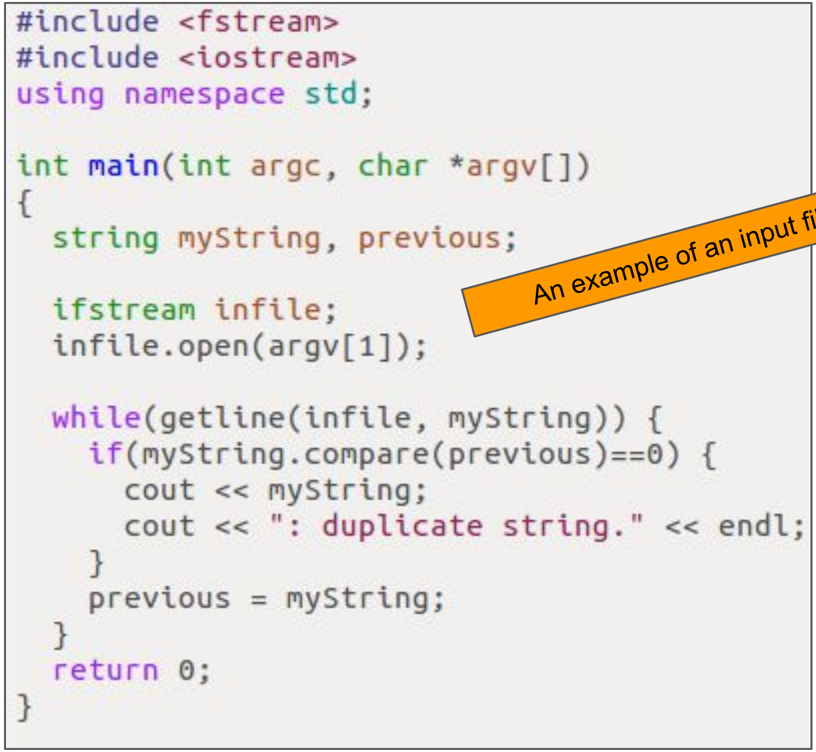
Copy from the current folder

```
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Docker build: a more complex example

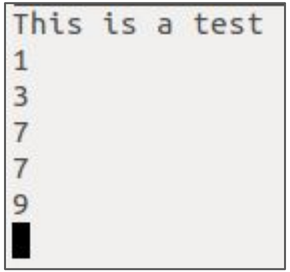
```
# Install newer version of CMake
RUN curl https://cmake.org/files/v3.14/cmake-3.14.6.tar.gz | tar -xz -C /tmp \
 && cd /tmp/cmake-3.14.6 && ./bootstrap && make -j $(nproc) && make install \
 && rm -fr /tmp/*

# Install ROOT5
RUN curl https://root.cern.ch/download/root_v5.34.38.source.tar.gz | tar -xz -C /tmp
\
 && mv /tmp/root /tmp/root-5-34-38 \
 && mkdir /tmp/root-build && cd /tmp/root-build \
 && cmake /tmp/root-5-34-38 \
    -DCMAKE_INSTALL_PREFIX=/usr/local \
    -Drpath=ON \
    -Dtable=ON \
    -Dpythia6=ON \
    -Dpythia6_nolink=ON \
    -Dvc=ON \
    -Dkrb5=OFF \
 && make -j $(nproc) \
 && make install \
 && rm -fr /tmp/*
```

Simple user code in C++... find duplicate lines



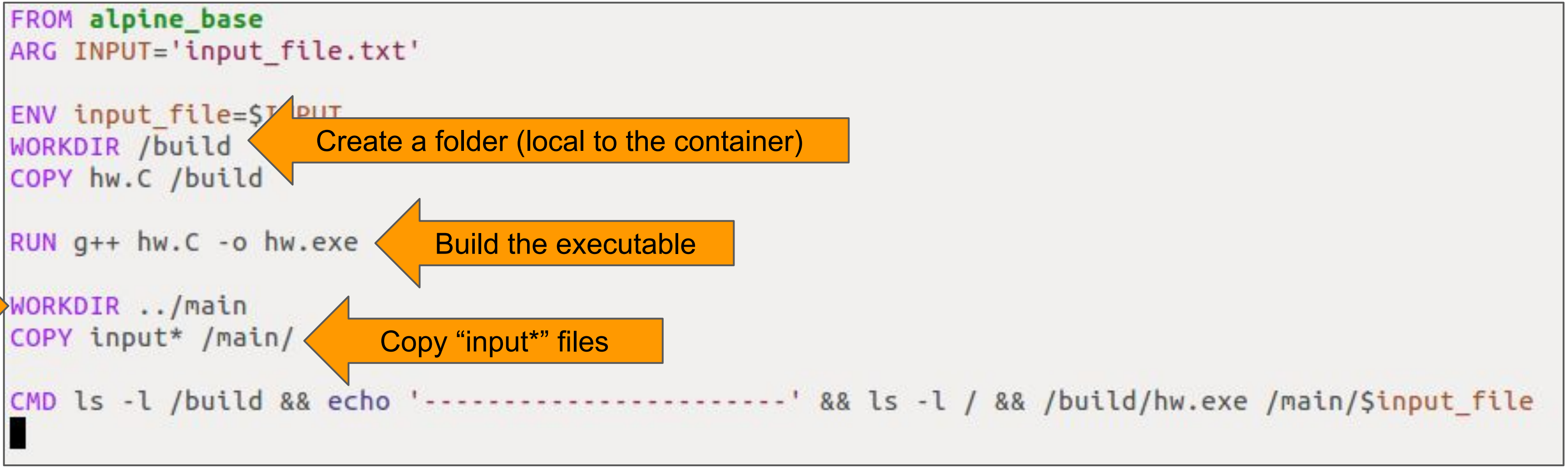
```cpp
#include <fstream>
#include <iostream>
using namespace std;

int main(int argc, char *argv[])
{
  string myString, previous;

  ifstream infile;
  infile.open(argv[1]);

  while(getline(infile, myString)) {
    if(myString.compare(previous)==0) {
      cout << myString;
      cout << ": duplicate string." << endl;
    }
    previous = myString;
  }
  return 0;
}
```

An example of an input file



```
This is a test
1
3
7
7
9
```

Dockerfile



```
FROM alpine_base
ARG INPUT='input_file.txt'

ENV input_file=$INPUT
WORKDIR /build
COPY hw.C /build

RUN g++ hw.C -o hw.exe

WORKDIR ../main
COPY input* /main/

CMD ls -l /build && echo '----------------------' && ls -l / && /build/hw.exe /main/$input_file
```

Create a folder (local to the container)

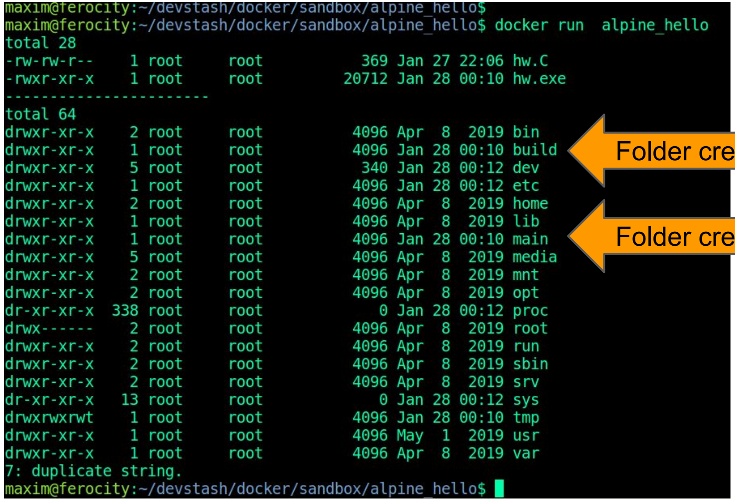
Build the executable

Copy “input*” files

In a nutshell - copy content from your local filesystem, manipulate it for placement in the image (which can include downloads, builds or any other operation). Folders internal to the container can be created as needed. There are environment variables internal to the Dockerfile (ARG), that can be changed at build time, and also the ones that can be defined at runtime (ENV). Build this image:

```
docker build -t alpine_hello .
```

Run the container



```
maxim@ferocity:~/devstash/docker/sandbox/alpine_hello$
maxim@ferocity:~/devstash/docker/sandbox/alpine_hello$ docker run  alpine_hello
total 28
-rw-rw-r--    1 root     root           369 Jan 27 22:06 hw.C
-rwxr-xr-x    1 root     root         20712 Jan 28 00:10 hw.exe
-----------------------
total 64
drwxr-xr-x    2 root     root          4096 Apr  8  2019 bin
drwxr-xr-x    1 root     root          4096 Jan 28 00:10 build
drwxr-xr-x    5 root     root           340 Jan 28 00:12 dev
drwxr-xr-x    1 root     root          4096 Jan 28 00:12 etc
drwxr-xr-x    2 root     root          4096 Apr  8  2019 home
drwxr-xr-x    1 root     root          4096 Apr  8  2019 lib
drwxr-xr-x    1 root     root          4096 Jan 28 00:10 main
drwxr-xr-x    5 root     root          4096 Apr  8  2019 media
drwxr-xr-x    2 root     root          4096 Apr  8  2019 mnt
drwxr-xr-x    2 root     root          4096 Apr  8  2019 opt
dr-xr-xr-x  338 root     root             0 Jan 28 00:12 proc
drwx------    2 root     root          4096 Apr  8  2019 root
drwxr-xr-x    2 root     root          4096 Apr  8  2019 run
drwxr-xr-x    2 root     root          4096 Apr  8  2019 sbin
drwxr-xr-x    2 root     root          4096 Apr  8  2019 srv
dr-xr-xr-x   13 root     root             0 Jan 28 00:12 sys
drwxrwxrwt    1 root     root          4096 Jan 28 00:10 tmp
drwxr-xr-x    1 root     root          4096 May  1  2019 usr
drwxr-xr-x    1 root     root          4096 Apr  8  2019 var
7: duplicate string.
maxim@ferocity:~/devstash/docker/sandbox/alpine_hello$
```

Folder created by the user

Folder created by the user

output

For demonstration purposes, we list the content of the folders internal to the container and then run the executable we created when building the image. This filesystem is internal to the container and not visible on your machine by default. However, it is possible to mount it if the data needs to be persistent. Hence, the concept of Docker volumes.

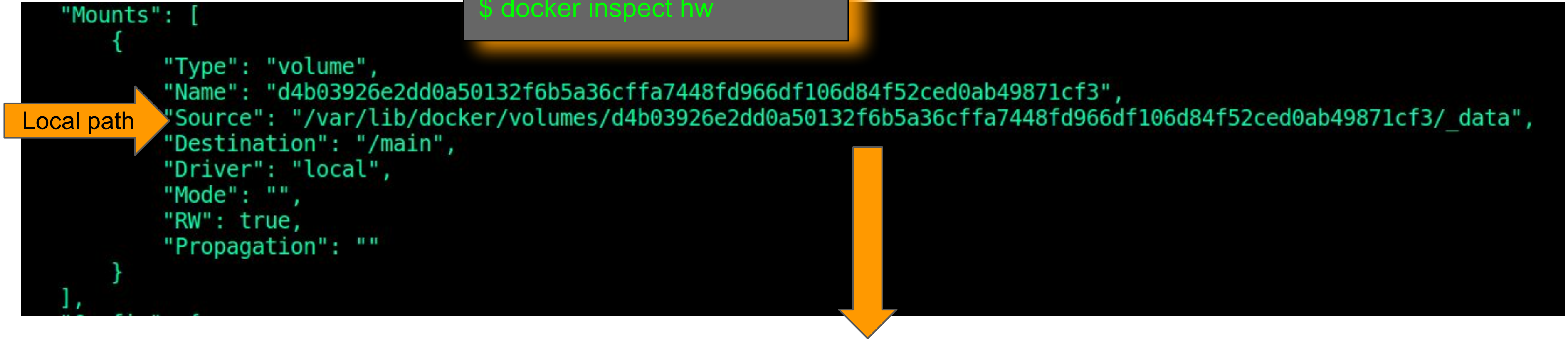
The “volume” option

```
$ docker run -v hw:/main  alpine_hello
```

Volume name

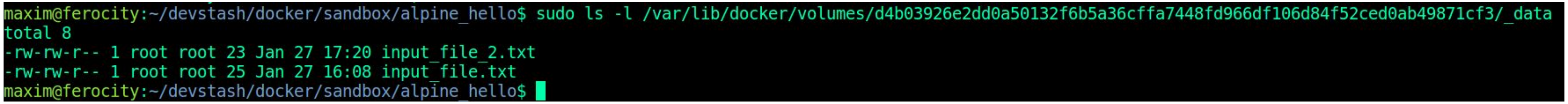
Internal folder

```
# Run the command, then
# check the volume metadata
$ docker inspect hw
```



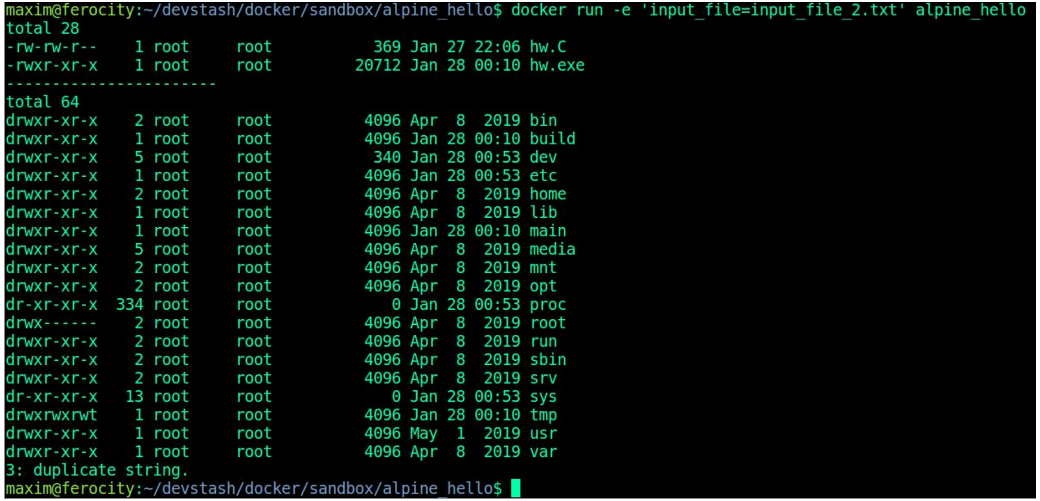
Local path

```
    "Mounts": [
        {
            "Type": "volume",
            "Name": "d4b03926e2dd0a50132f6b5a36cffa7448fd966df106d84f52ced0ab49871cf3",
            "Source": "/var/lib/docker/volumes/d4b03926e2dd0a50132f6b5a36cffa7448fd966df106d84f52ced0ab49871cf3/_data",
            "Destination": "/main",
            "Driver": "local",
            "Mode": "",
            "RW": true,
            "Propagation": ""
        }
    ],
```



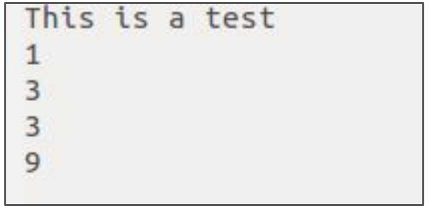
```
maxim@ferocity:~/devstash/docker/sandbox/alpine_hello$ sudo ls -l /var/lib/docker/volumes/d4b03926e2dd0a50132f6b5a36cffa7448fd966df106d84f52ced0ab49871cf3/_data
total 8
-rw-rw-r-- 1 root root 23 Jan 27 17:20 input_file_2.txt
-rw-rw-r-- 1 root root 25 Jan 27 16:08 input_file.txt
maxim@ferocity:~/devstash/docker/sandbox/alpine_hello$
```

Runtime environment



```
maxim@ferocity:~/devstash/docker/sandbox/alpine_hello$ docker run -e 'input_file=input_file_2.txt' alpine_hello
total 28
-rw-rw-r--    1 root     root           369 Jan 27 22:06 hw.C
-rwxr-xr-x    1 root     root         20712 Jan 28 00:10 hw.exe
-----------------------
total 64
drwxr-xr-x    2 root     root          4096 Apr  8  2019 bin
drwxr-xr-x    1 root     root          4096 Jan 28 00:10 build
drwxr-xr-x    5 root     root           340 Jan 28 00:53 dev
drwxr-xr-x    1 root     root          4096 Jan 28 00:53 etc
drwxr-xr-x    2 root     root          4096 Apr  8  2019 home
drwxr-xr-x    1 root     root          4096 Apr  8  2019 lib
drwxr-xr-x    1 root     root          4096 Jan 28 00:10 main
drwxr-xr-x    5 root     root          4096 Apr  8  2019 media
drwxr-xr-x    2 root     root          4096 Apr  8  2019 mnt
drwxr-xr-x    2 root     root          4096 Apr  8  2019 opt
dr-xr-xr-x  334 root     root             0 Jan 28 00:53 proc
drwx------    2 root     root          4096 Apr  8  2019 root
drwxr-xr-x    2 root     root          4096 Apr  8  2019 run
drwxr-xr-x    2 root     root          4096 Apr  8  2019 sbin
drwxr-xr-x    2 root     root          4096 Apr  8  2019 srv
dr-xr-xr-x   13 root     root             0 Jan 28 00:53 sys
drwxrwxrwt    1 root     root          4096 Jan 28 00:10 tmp
drwxr-xr-x    1 root     root          4096 May  1  2019 usr
drwxr-xr-x    1 root     root          4096 Apr  8  2019 var
3: duplicate string.
maxim@ferocity:~/devstash/docker/sandbox/alpine_hello$
```

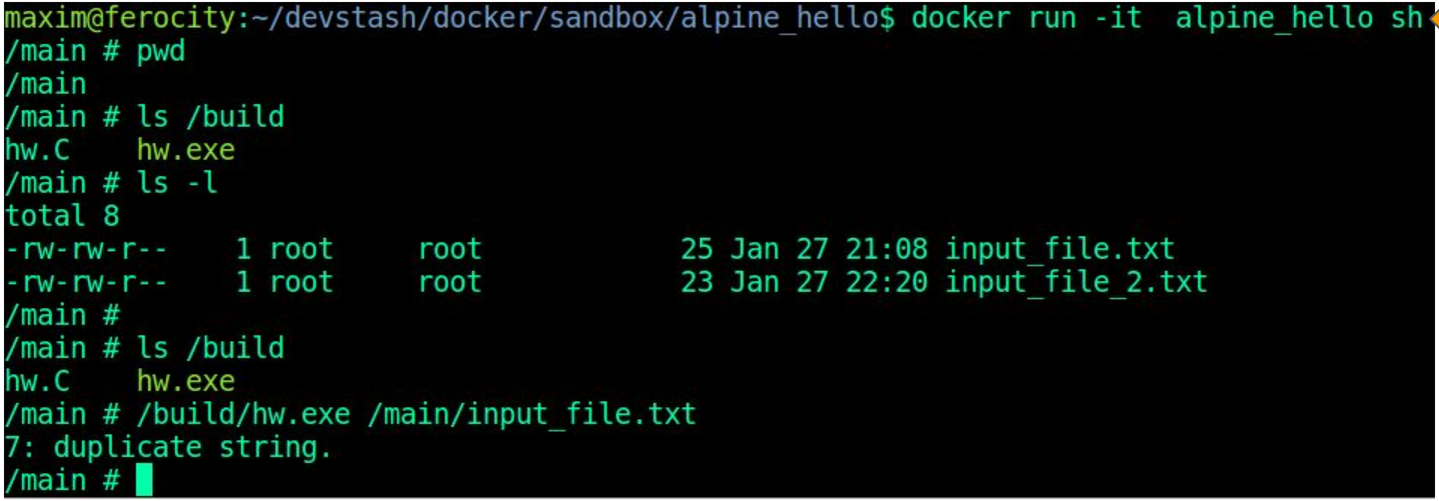
output



```
This is a test
1
3
3
9
```

Containers can be steered using environment variables set at runtime, potentially overriding default values that may have been set at build time.

Running container interactively (using shell)



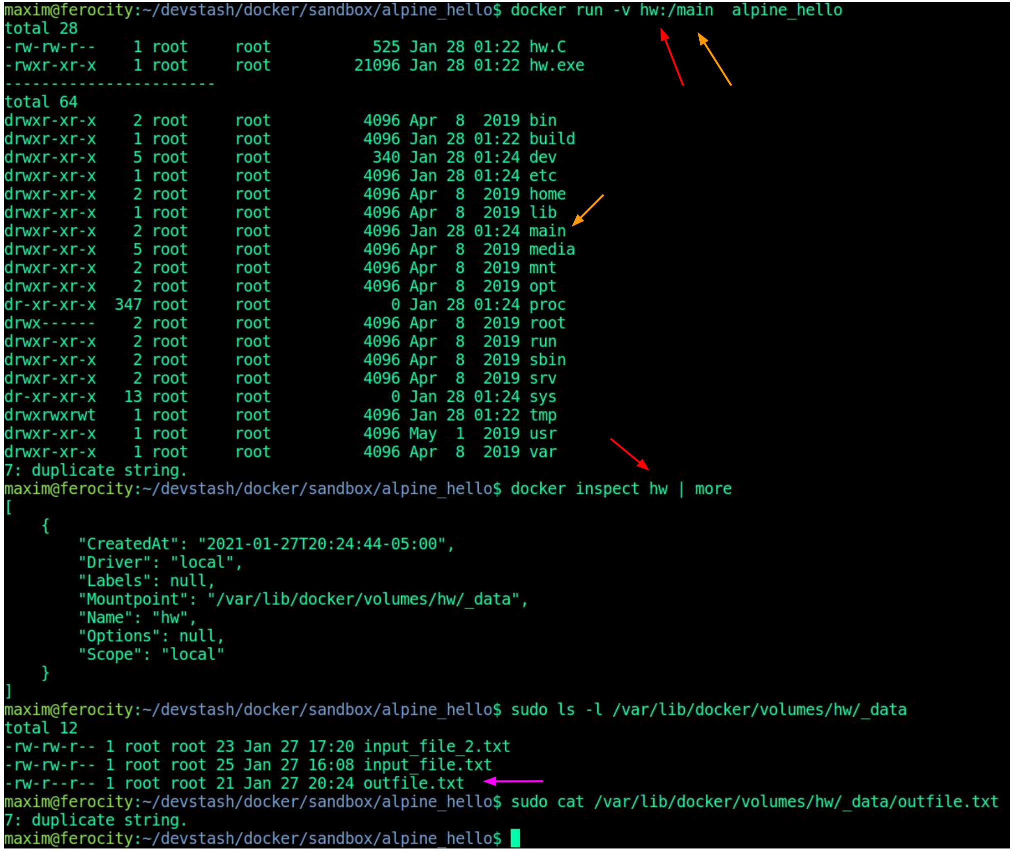
```
maxim@ferocity:~/devstash/docker/sandbox/alpine_hello$ docker run -it  alpine_hello sh
/main # pwd
/main
/main # ls /build
hw.C    hw.exe
/main # ls -l
total 8
-rw-rw-r--    1 root     root            25 Jan 27 21:08 input_file.txt
-rw-rw-r--    1 root     root            23 Jan 27 22:20 input_file_2.txt
/main #
/main # ls /build
hw.C    hw.exe
/main # /build/hw.exe /main/input_file.txt
7: duplicate string.
/main # 
```

Interactive mode

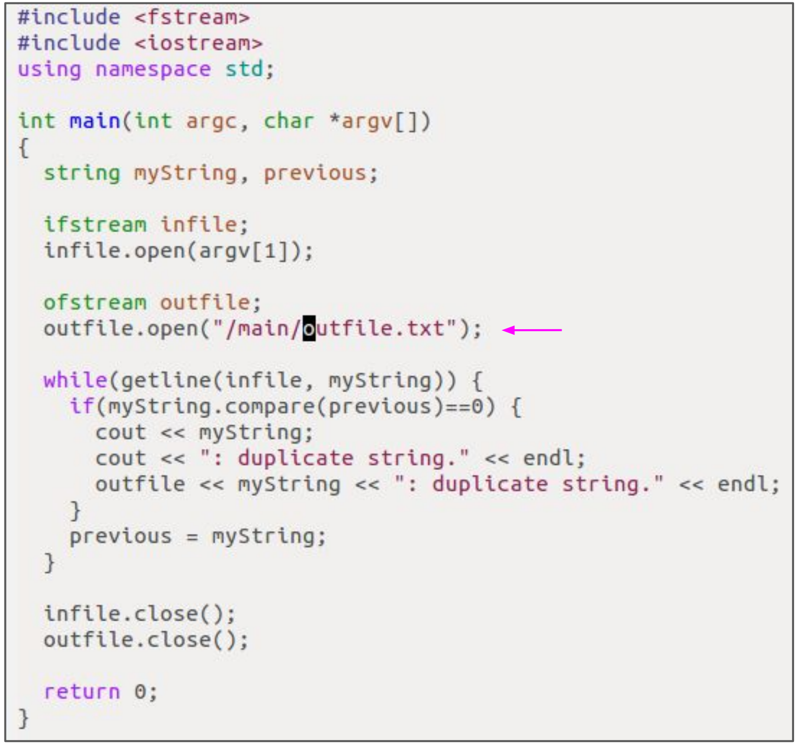
output

Any command available to the container can be specified at the runtime, including “sh”.
Using the “-it” option an interactive terminal can be connected to the “sh” process running in the container. Then, any operation can be performed in the usual Linux environment.

Volume: capture the output



```
maxim@ferocity:~/devstash/docker/sandbox/alpine_hello$ docker run -v hw:/main  alpine_hello
total 28
-rw-rw-r--    1 root     root           525 Jan 28 01:22 hw.C
-rwxr-xr-x    1 root     root         21096 Jan 28 01:22 hw.exe
-----------------------
total 64
drwxr-xr-x    2 root     root          4096 Apr  8  2019 bin
drwxr-xr-x    1 root     root          4096 Jan 28 01:22 build
drwxr-xr-x    5 root     root           340 Jan 28 01:24 dev
drwxr-xr-x    1 root     root          4096 Jan 28 01:24 etc
drwxr-xr-x    2 root     root          4096 Apr  8  2019 home
drwxr-xr-x    1 root     root          4096 Apr  8  2019 lib
drwxr-xr-x    2 root     root          4096 Jan 28 01:24 main
drwxr-xr-x    5 root     root          4096 Apr  8  2019 media
drwxr-xr-x    2 root     root          4096 Apr  8  2019 mnt
drwxr-xr-x    2 root     root          4096 Apr  8  2019 opt
dr-xr-xr-x  347 root     root             0 Jan 28 01:24 proc
drwx------    2 root     root          4096 Apr  8  2019 root
drwxr-xr-x    2 root     root          4096 Apr  8  2019 run
drwxr-xr-x    2 root     root          4096 Apr  8  2019 sbin
drwxr-xr-x    2 root     root          4096 Apr  8  2019 srv
dr-xr-xr-x   13 root     root             0 Jan 28 01:24 sys
drwxrwxrwt    1 root     root          4096 Jan 28 01:22 tmp
drwxr-xr-x    1 root     root          4096 May  1  2019 usr
drwxr-xr-x    1 root     root          4096 Apr  8  2019 var
7: duplicate string.
maxim@ferocity:~/devstash/docker/sandbox/alpine_hello$ docker inspect hw | more
[
    {
        "CreatedAt": "2021-01-27T20:24:44-05:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint": "/var/lib/docker/volumes/hw/_data",
        "Name": "hw",
        "Options": null,
        "Scope": "local"
    }
]
maxim@ferocity:~/devstash/docker/sandbox/alpine_hello$ sudo ls -l /var/lib/docker/volumes/hw/_data
total 12
-rw-rw-r-- 1 root root 23 Jan 27 17:20 input_file_2.txt
-rw-rw-r-- 1 root root 25 Jan 27 16:08 input_file.txt
-rw-r--r-- 1 root root 21 Jan 27 20:24 outfile.txt
maxim@ferocity:~/devstash/docker/sandbox/alpine_hello$ sudo cat /var/lib/docker/volumes/hw/_data/outfile.txt
7: duplicate string.
maxim@ferocity:~/devstash/docker/sandbox/alpine_hello$
```



```cpp
#include <fstream>
#include <iostream>
using namespace std;

int main(int argc, char *argv[])
{
  string myString, previous;

  ifstream infile;
  infile.open(argv[1]);

  ofstream outfile;
  outfile.open("/main/outfile.txt");

  while(getline(infile, myString)) {
    if(myString.compare(previous)==0) {
      cout << myString;
      cout << ": duplicate string." << endl;
      outfile << myString << ": duplicate string." << endl;
    }
    previous = myString;
  }

  infile.close();
  outfile.close();

  return 0;
}
```

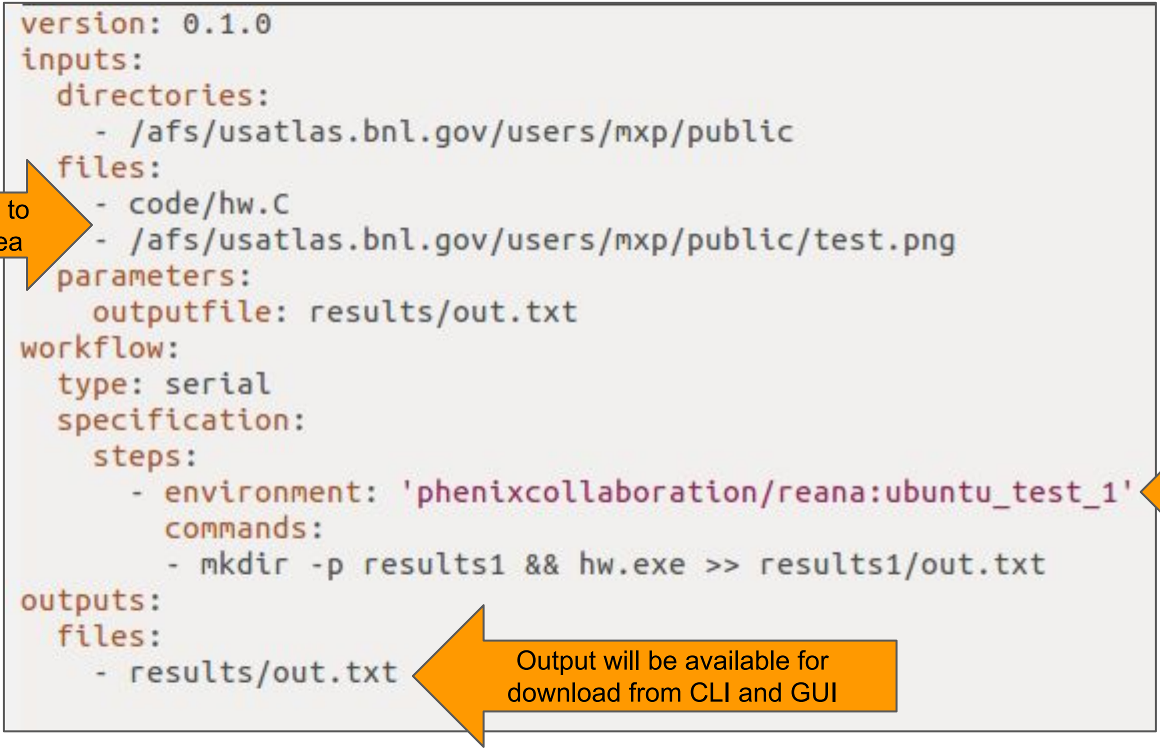
Docker in PHENIX: next steps

- ROOT macros can be run trivially in containers - could be a useful starting point to capture some parts of analyses (advantage being that an image would contain ALL necessary components) - an intelligent replacement for a tarball
- Challenge - many moving parts in complete analysis, studying use cases
  - Cf. a recent analysis note by Ron and Jamie - well done but complex
- When building images, need to keep REANA implications in mind
- As usual, a volunteer attempting to dockerize their analysis would be of much help
- Will use GitHub to host the code for working examples

REANA: a flashcard

- Combines:
    - Docker images
    - Input data
    - Workflow logic
    - ...all described in a file according to a specific format
- Supports:
    - Workflows - via the “common workflow language” interface (YAML)
- Runs:
    - Non-interactively, on a dedicated cluster
    - CLI utilities to monitor and manage the process
- Increased acceptance in the community as a tool to have a 100% reproducible computation process

REANA: previous Docker example



```
version: 0.1.0
inputs:
  directories:
    - /afs/usatlas.bnl.gov/users/mxp/public
  files:
    - code/hw.C
    - /afs/usatlas.bnl.gov/users/mxp/public/test.png
  parameters:
    outputfile: results/out.txt
workflow:
  type: serial
  specification:
    steps:
      - environment: 'phenixcollaboration/reana:ubuntu_test_1'
        commands:
        - mkdir -p results1 && hw.exe >> results1/out.txt
outputs:
  files:
    - results/out.txt
```

Staging files to the work area

Pulling image from the PHENIX repository

Output will be available for download from CLI and GUI

Some daylight between Docker and REANA

- REANA runs Docker containers. But what about the AFS dependencies?
- AFS volumes can be mounted on the Docker container in a way similar to shown above (tested) - this works because *AFS is installed on the host machine* where the container is running.
- Worker nodes in a REANA cluster may be (and are) a different story - the preference of the SDCC admins is to not mount AFS for practical and philosophical reasons. This can be negotiated if becomes a show stopper.
- Potential solution - generate REANA steering files which copy requisite content from AFS to the workflow. This will mean that many analysis scripts may need to be redesigned.
- Same applies to software provisioning from CVMFS.
- Discussions are underway and we will get more understanding soon.
- REANA can be picky about certain base images.

Analysis notes and related privacy issues and options

- Any type of file sharing option - with encryption
  - Passwords can be circulated to select participants only
  - Finding a truly portable solution may be a bit of a challenge, openssl is a strong contender (all platforms)
- Zenodo - private access option
  - Access on demand, decided by the PHENIX Zenodo curators
  - The only solution offering proper built-in indexing and search capabilities
- GitHub - a private repository
  - Accessible to users on a managed list
  - GitHub tags can be used for indexing (like keywords)
- BNLbox
  - Broadly speaking, an equivalent of Dropbox with vastly larger storage available
  - Web UI
  - File upload and download using a CLI script is possible
  - A fairly capable access control system

Status and Plans

- Setting up Docker and REANA environments
  - Labor-intensive, not expecting quick results
- Finalizing the OpenData entry, hopefully extending to more items
- Ongoing HEPData work - steady state by now
- Should we skip the next meeting to allow for more technical details to be developed?