Containers for Machine Learning in HEP

Matthew Feickert
matthew.feickert@cern.ch

3rd IML Machine Learning Workshop

April 16th, 2019
Collaborators

Lukas Heinrich
CERN

Alessandra Forti
U Manchester

Manuel Guth
Freiburg

Dan Guest
UC Irvine
Current HEP Computation

- Highly parallelizable
- Good for distributed, bulk computation
- Worldwide LHC Computing Grid (GRID) provides global infrastructure
  - Storage and compute
  - Code goes to where the data is
Problem: Software Distribution

- Need to get the **runtime environment** to data location

### Operating System
- libc
- openssl
- sed, curl...

### VO specific software
- Experiment Frameworks often with
  - custom compiler
  - custom language runtimes (python)
  - LCG

### Application specific s/w
- Application specific code

#### The basics: OS + utils
- Common Experiment
- Your Code

- Relies on three parties to work (sys admins, experiments, users)
- Great for distributing crucial software, not for experimenting with new tools
- User only controls their code, upstream changes break reproducibility
Machine Learning needs are a challenge for current systems. It moves fast, experiments with bleeding edge tools, and a non-traditional HEP stack developed by industry.

Idea: Distribute software stack with containers.
- Let the user assemble the full stack and optimize GRID for running containers.
- Host can be greatly simplified to core OS to run containers.

Containers

Docker Image (think: executable filesystem snapshot)
Containers for ML

- ML applications currently one area that can realistically and usefully utilize hardware accelerators (GPUs)
- New HPC machines coming online ability to support HEP workloads
- Use of GPU requires custom compiled software
  - Example: TensorFlow GPU with CUDA
- With containers this is plug and play compute!
Containers for HEPML

- With containers can run large-scale jobs on WLCG infrastructure
  - hyperparameter scans
  - flavour tagging retraining

Flavour tagging ROC for hyperparameter scans
[A. C. Forti, L. Heinrich, M. Guth, ACAT 2019]
Containers for HEPML

Flavour tagging hyperparameters scans on the GRID with containers

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Jobset</th>
<th>Type</th>
<th>Working Group</th>
<th>User</th>
<th>Destination</th>
<th>Task status</th>
<th>Nevents</th>
<th>Ninputfiles</th>
<th>HSO6*sec</th>
<th>Average maxpss</th>
<th>Created</th>
<th>Modified</th>
<th>Cores</th>
<th>Priority</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>18165474</td>
<td>3728</td>
<td>analy</td>
<td></td>
<td>Lukas Alexander Heinrich</td>
<td>done</td>
<td>0</td>
<td>0 (%)</td>
<td>None 14016 14016 0</td>
<td>1 1 (100%)</td>
<td>2018-11-23 18:41:49</td>
<td>2018-11-23 19:17:29</td>
<td>1</td>
<td>1000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Submitted to GPU site at Manchester
User Interface

Integrated into main command line interface for GRID submission (PanDA):

```
prun
    --containerImage docker://<image> \
    --exec "<shell script>" \
    --inDS <input dataset> \
    --outDS <output dataset> \
    --outputs <output files> \
    --site <site name> \
    --forceStaged
```

In development/testing: dedicated pcontainer CLI (drop all software related flags)
Machine Learning Base Images

- **ML base images**
  - OS flavors
    - Ubuntu 18.04
    - CentOS 7
  - Goal: To provide modern Pythonic machine learning stack, tools, environment

- **ATLAS ML base images**
  - Copying ATLAS Software Infrastructure Team to make stack of images
    - ATLAS OS (CentOS 7)
    - AnalysisBase (CentOS 7)
    - ATLAS ML base (CentOS 7)
  - Goal: Provide AnalysisBase and ATLAS tools along with ML base

- **Singularity compatible** mount points for GRID
- Common ML tools and starting image for personal ML workflows
Base Image Development

Dockerfiles versioned on GitLab...

...and deployed to Docker Hub

...built by CI...
Base Image Environment

Modern **Pythonic** ML environment with common (HEP)ML tools

```
$ python3 --version
Python 3.6.8 # From CPython source
$ pip3 --version
pip 19.0.3
$ cat requirements.txt
numpy==1.16
scikit-learn==0.20
tensorflow==1.12
keras==2.2
torch==1.0
uproot==3.4
matplotlib==3.0
pandas==0.24
jupyter==1.0
```

```
$ git --version
git version 2.17.1
$ h5ls --version
h5ls: Version 1.10.0-patch1
$ jq --version
jq-1.5
$ pip3 freeze | grep h5py
h5py==2.9.0
# and more
```
Supports Interactive Sessions

```
import numpy as np
import tensorflow as tf
from keras.layers import Input, Flatten, Dense, Dropout
from keras.models import Model

Using TensorFlow backend.

Load and prepare the MNIST dataset and convert the samples from integers to floating-point numbers

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
1149376/11490454 [==============================] - 3s 0us/step

Build the keras model using the functional API

```
inputs = Input(shape=x_train.shape[1:]):
x = Flatten()(inputs)
x = Dense(512, activation=tf.nn.relu)(x)
x = Dropout(0.2)(x)
predictions = Dense(10, activation=tf.nn.softmax)(x)

model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

Train and evaluate the model loss and accuracy

```
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)

Epoch 1/5
60000/60000 [==============================] - 12s 197us/sample - loss: 0.2197 - acc: 0.9350
Epoch 2/5
60000/60000 [==============================] - 12s 196us/sample - loss: 0.0973 - acc: 0.9702
Epoch 3/5
60000/60000 [==============================] - 12s 202us/sample - loss: 0.0868 - acc: 0.9789
Epoch 4/5
60000/60000 [==============================] - 12s 204us/sample - loss: 0.0542 - acc: 0.9823
Epoch 5/5
60000/60000 [==============================] - 12s 281us/sample - loss: 0.0428 - acc: 0.9859

Out[4]: [0.0639623103929932, 0.9883]
Upcoming: optimized images

**GPU**

- Images with CUDA/cuDNN support for GPU acceleration
- GPU support for TensorFlow and PyTorch

**Size**

- Slimmed and specialized base images for size optimization
  - Example: Don't distribute Jupyter if only want to train models on GRID
- In some cases can experiment with Alpine images to drop MBs of size
Summary

- HEPML needs non-traditional software stack. Served well by containers.
- Containers allow for local and production to be the same environment.
- GRID submission CLI of jobs with containers live now.
- GRID + containers allows for unique workflows at scale and efficient use of GPU site resources.
- ATLAS Machine Learning Forum provides ML base images openly distributed on Docker Hub.
  - Part of CVMFS sync, so on CVMFS now.
  - Contributions welcome on GitLab.
Backup
How many WLCG sites support GPUs for hardware acceleration?

- At the moment 2
  - Manchester
  - DESY
- Looking to get more online soon given the results from collaboration between Lukas Heinrich (CERN) and Alessandra Forti (University of Manchester)!
Do the images work on the WLCG?

- Yes! They can be used through Singularity as Lukas has shown for a few sites
- Checkout the `pcontainer` API

```bash
lheinic@lxplus084:~$ pcontainer --help
usage: pcontainer [options]
   HowTo is available at https://twiki.cern.ch/twiki/bin/view/PanDA/PandaContainer

optional arguments:
  -h, --help            show this help message and exit
  --version             Displays version
  --loadJson LOADJSON   Read command-line parameters from a json file which contains a dict of (parameter: value)
  --dumpJson DUMPJSON   Dump all command-line parameters and submission result such as returnCode, returnOut, jediTaskID, etc to a json file
  --containerImage CONTAINERIMAGE
                         Name of a container image
  --excludedSite EXCLUDEDSITE
                         List of sites which are not used for site section, e.g., ANALY_ABC, ANALY_XYZ
  --site SITE
                         Site name where jobs are sent. If omitted, jobs are automatically sent to sites where input is available. A comma-separated list of sites can be specified (e.g. siteA,siteB,siteC), so that best sites are chosen from the given site list
  --architecture ARCHITECTURE
                         Description string for HW requirements
  --outDS OUTDS
                         Base name of the output dataset container. Actual output dataset name is defined for each output file type
  --outputs OUTPUTS    Output file names. Comma separated. e.g., --outputs out1.dat,out2.txt. You can specify a suffix for each output container like <datasetNameSuffix>:<outputFileName>. e.g., --outputs AAA:out1.dat,BBB:out2.txt, so that output container names are outDS_AAA/ and outDS_BBB/ instead of outDS_out1.dat/ and outDS_out2.txt/
  --exec EXEC          execution string. e.g., --exec "./myscript arg1 arg2"
  -v, --verbose        Verbose
  --priority PRIORITY  Set priority of the task (1000 by default). The value must be between 900 and 1100. Note that priorities of tasks are relevant only in each user's share, i.e., your tasks cannot jump over other user's tasks even if you give higher priorities.
```
Does the WLCG support other container runtimes than Singularity?

- At time of this talk only Singularity is supported at most sites
- Active work to support other container runtimes (e.g., Docker (rootless), Podman)
Can I use these base images to make my own images?

- Yes! That's the idea. :)
- Here's a minimal working example:
  - **Write** Dockerfile
    ```
    FROM atlasml/ml-base:py-3.6.8
    USER root
    RUN pip3 install --no-cache-dir xgboost
    USER atlas
    ```
  - **Build and tag image**
    ```
    docker build -f Dockerfile -t xgboost-example --compress .
    ```
  - **Run container**
    ```
    docker run --rm xgboost-example \\
    python3 -c "import xgboost, uproot; print(xgboost, uproot)"
    ```
Do these base images work on LXPLUS?

- Yes! You can use Singularity to run the Docker images as Singularity containers
- See the documentation for examples: Running on LXPLUS with Singularity
References


The end.