ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA167

CLOUD COMPUTING PLATFORM FOR HIGH-LEVEL PHYSICS APPLICATIONS DEVELOPMENT *

T. Zhang† and D. Maxwell
Facility for Rare Isotope Beams, Michigan State University, East Lansing, USA

Abstract

To facilitate software development for the high-level applications on the particle accelerator, we proposed and prototyped a web-based computing platform, the so-called 'phyapps-cloud'. Based on the technology stack composed by Python, JavaScript, Docker, and Web service, such a system could greatly decouple the deployment and development, that is the users only need to focus on the feature development by working on the infrastructure that served by 'phyapps-cloud', while the service provider could focus on the development of the infrastructure. In this contribution, the development details will be addressed, as well as the demonstration of developing Python scripts for physics tuning algorithm on this platform.

INTRODUCTION

With the rapid evolution of information technology, the development of high-level controls software in the accelerator community has been dramatically changing. To apply the most state-of-the-art technology to the accelerator controls application development is like standing on the shoulders of giants, the modern application could always be expected.

High-level physics controls software is to solve the problems regarding how to control the machine with some complex physics tuning algorithms, could depend on complicated physics model, or some generic data crunching routine. The development of the high-level physics applications usually requires a backend service running which can be treated as the data source to the applications, including to accept the input from the applications and to respond to the applications with output, such service typically is a so-called virtual accelerator, which is driven by a specific physics model engine.

The data communication between the application and virtual accelerator should be well abstracted, such that the developed application can work with real accelerator rather than the virtual one. For example, the implemented EPICS-based [1] virtual accelerator is an IOC application, which also exists in the controls network, the high-level application should be able to work with either of them.

The conventional way to develop the high-level physics applications usually requires the developer to install all the required software in a Linux workstation or make use of the well-packed VirtualBox [2] appliance in the VirtualBox application in any host OS. At FRIB, all the requirements have already been packaged into Debian packages, the developer can easily install all of them by 'apt-get' commands on any computer running Debian 8 or 9 OS, and the development work could start. While there is still the maintenance overhead to keep the packages updated and troubleshoot the configuration issues. So, the web-based platform for physics applications named as 'phyapps-cloud' was designed and implemented [3].

Here is the significant advantage of 'phyapps-cloud'. On one hand, the users (the app developers and users) can do the development in the web browser from anywhere, there is no requirement for the development environment. On the other hand, the platform developer can keep 'phyapps-cloud' always updated, with little maintenance effort. By utilizing Docker [4] to containerize the physics applications development environment, the maintenance effort can be further reduced. In this paper, the design, implementation and use case of 'phyapps-cloud' is presented.

SYSTEM ARCHITECTURE

The designed architecture for 'phyapps-cloud' is sketched in Fig. 1. Below lists the main components to compose 'phyapps-cloud':

- `Gateway`: REST web service features the main entrance of 'phyapps-cloud', and the center for the users and services management;
- `Configurable Proxy`: Web service features configurable proxy, which provides a way to update and manage a proxy table by REST API;
- `Service-`*i*: Private service created by the users, currently, two different services are supported, both features with Jupyter-Notebook service and FRIB physics application development environment, but one also comes with the FRIB virtual accelerator service, while the other does not.

The `Configurable Proxy` web service is maintained by the opensource community [5], which is one part of the JupyterHub project [6], while the others are developed at FRIB as a byproduct of 'phantasy-project' [7], they are an alternative method of deployment for physics applications development based on web technology.

`Gateway` REST web app is developed with FLASK [8], which is a lightweight WSGI web application framework for Python [9]. It serves as the main portal where the user connects to 'phyapps-cloud', as Fig. 2 shows. All the new users can sign up for the first time to get onto the platform, then

* Work supported by the U.S. Department of Energy Office of Science under Cooperative Agreement DE-SC0000661, the State of Michigan and Michigan State University.
† zhangt@frib.msu.edu

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA167

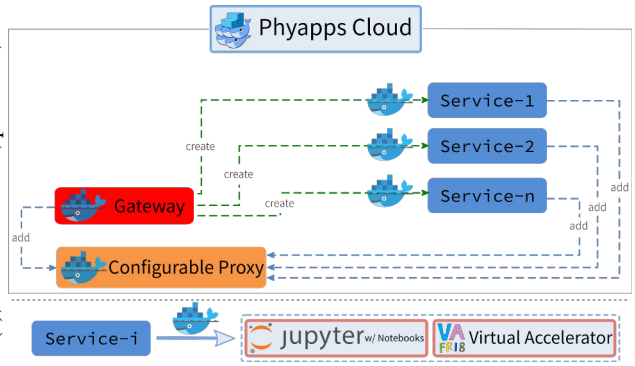

Figure 1: Overview of the architecture of 'phyapps-cloud' platform. All the apps are Docker containers, managed by Docker Swarm. The users can connect through `Configurable Proxy` to the `gateway`, where all the private computing services (`Service-i`) could be spawned.

each user could control their own computing service supported by 'phyapps-va' and 'phyapps-nb'. 'phyapps-va' is the Jupyter-notebook service with FRIB virtual accelerator, and 'phyapps-nb' is the one without the virtual accelerator. All the users should have the private space of Jupyter-notebook environment to work with, if the virtual accelerator is reachable, the user can test the EPICS controls physics applications. The separated private space for each user is implemented by containerizing the computing services by Docker [4]. In fact, all the web apps including `gateway` app and phyapps computing services, are packed with Docker containers.

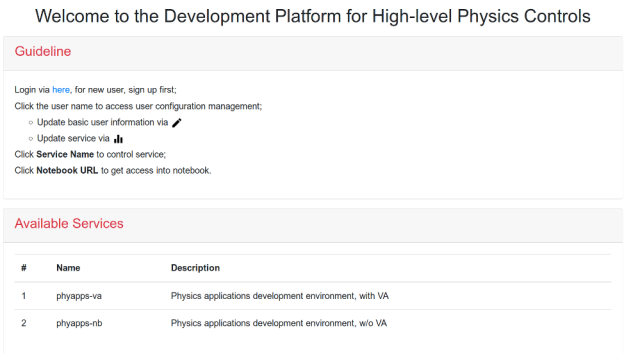

Figure 2: The index page of `gateway` web app, which shows the brief information of 'phyapps-cloud', hyperlinks for other operations, e.g. 'Activity' to show the current system usage information by the registered users, and user login/sign-up pages.

In the user workspace (see Fig. 3), the phyapps services are distributed with a list of default Jupyter notebooks for the convenience of the development, based on which, quick demonstration could be expected. Once the container of these phyapps service is updated, the user will be able to get by restarting the computing service. As the 'phyapps-cloud' platform maintainer, additional computing service with different preinstalled Python packages could be created and integrated, which indicates good extendibility. Currently, all the phyapps computing services are preinstalled with phantasy-project packages, one can develop applications for FRIB driver LINAC while testing against the embedded virtual accelerator.

If the target facility is not FRIB, a new computing service is required to be created, i.e. to build another Docker container image, within which, the new 'phantasy-machines' package for the target facility should be created, and since the current virtual accelerator built for FRIB is based on FLAME physics model [10], the target facility should be able to model with FLAME, if not, additional work is required [7].

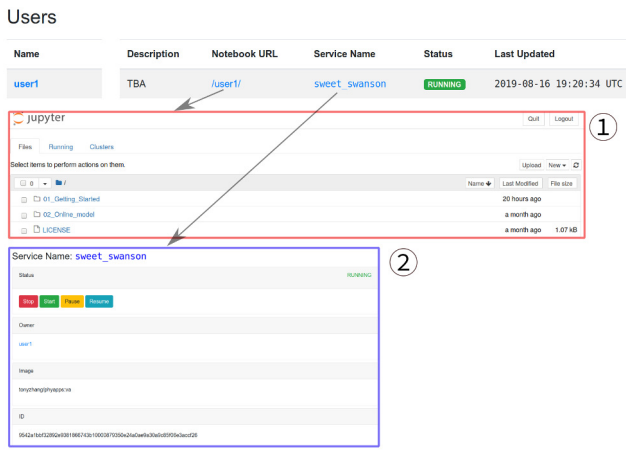

Figure 3: The activity page of `gateway` web app, which shows the status of all users, click the 'Notebook URL' will direct to the Jupyter-Notebook page for conducting the work (as shown in ①), while the phyapps service could be controlled by clicking 'Service Name' hyperlink.

Currently, each user can only have one computing service, if a new one is created, the old one will be overridden. The computing service supports the operations like 'Start', 'Stop', 'Pause' and 'Resume'. Also, the administrative account with highest privilege is initialized when the `gateway` app is started up for the first time. The administrator is able to manage all the user and computing service resources, the details can be found in the project site of 'phyapps-cloud' [3].

DEPLOYMENT

To make 'phyapps-cloud' easy to deploy and make the most of the clouding computing technology, Docker Swarm [11] deployment mode is applied.

Docker Swarm is a clustering and scheduling tool for Docker containers. Containers are isolated from one another and bundle their own software, libraries and configuration files, they can communicate with each other through well-defined channels. Docker is a set of platform-as-a-service (PaaS) [12] products that use OS-level virtualization to deliver software in containers.

As Fig. 1 shows all the components are Docker containers, the 'phyapps-cloud' itself is a Docker Swarm stack. Swarm

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA167

stack is managed by docker compose YAML configuration file [13]. Before the deployment of the service stack, the Swarm environment needs to be initialized to support other nodes added as a worker into this swarm, thus, to scale the entire stack [14]. After that, 'phyapps-cloud' platform could be created on any Linux workstation by the following minimal docker-compose YAML file (see Fig. 4), the extended version could be found from the repository of 'phyapps-cloud' [3].

```yaml
version: '3.7'
services:
  db:
    image: mysql:8.0.16
    command: --default-authentication-plugin=mysql_native_password
    volumes:
      - db-data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
      - MYSQL_DATABASE=${DATABASE_NAME}
      - MYSQL_USER=${DATABASE_USER}
      - MYSQL_PASSWORD=${DATABASE_PASS}
    ports:
      - target: 3306
        published: 3307
        protocol: tcp
        mode: host
    deploy:
      restart_policy:
        condition: on-failure
  chp:
    image: "jupyterhub/configurable-http-proxy"
    environment:
      - CONFIGPROXY_AUTH_TOKEN=${TOKEN}
    command: --default-target http://127.0.0.1:5050 --ip ${SRV_IP} --port 8000
    networks:
      hostnet: {}
    deploy:
      restart_policy:
        condition: on-failure
  gateway:
    image: "tonyzhang/phyapps-gateway:latest"
    environment:
      - DB_NAME=${DATABASE_NAME}
      - DB_USER=${DATABASE_USER}
      - DB_PASS=${DATABASE_PASS}
      - PROXY_TOKEN=${TOKEN}
      - PROXY_BASE=http://127.0.0.1:8001/api/routes
      - DPATH=${PWD}/data
    networks:
      hostnet: {}
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    deploy:
      restart_policy:
        condition: on-failure
volumes:
  db-data:
networks:
  hostnet:
    external: true
    name: host
```

Figure 4: Docker compose YAML file for 'phyapps-cloud' swarm configuration.

To make the Swarm deployment painless, another Makefile is created to handle all the macros show in the YAML file [15]. The final deploy command line is `SRV_IP=<IP> make deploy`, where `<IP>` is the IP address or domain of the host workstation, say `10.20.30.40`. Then the user can reach the platform by visiting `https://10.20.30.40:8000`. Please note the `gateway` is proxied by `Configurable Proxy` service.

The swarm service is easy to manage, by adding a few lines to the compose YAML file to make use the service named `portainer` [16], which supports the platform maintainer to manage all the Docker containers in the nice web UI. Any other containerized service could be added into the swarm stack, too.

CONTAINERIZATION

`gateway` web app The `gateway` web app is a FLASK app, with MySQL as the backend database to keep all the data, to make it deploy within the Docker Swarm framework, containerization is required.

The essential practice to make a Docker container image is to produce the final Docker image as small as possible, such that, the container startup time and network traffic requirements will be reduced. All the packed containers are pushed to Docker Cloud [17], where all published container images are kept from developers around the world.

Figure 5 shows the Dockerfile that is required by the building process, which will generate a Docker container image with only 60 MB [18].

```
Dockerfile                                                    buffers
 1 FROM python:3.6.9-alpine
 2 LABEL maintainer="Tong Zhang <zhangt@frib.msu.edu>"
 3
 4 WORKDIR "/phyapps"
 5
 6 ADD requirements-freezed.txt .
 7 RUN apk add --no-cache py3-cffi py3-openssl
 8
 9 RUN mkdir notebooks && \
10     pip install --no-cache-dir -r requirements-freezed.txt && \
11     rm requirements-freezed.txt
12
13 ADD phycloud.tar.bz2 .
14 ADD mgmt /usr/local/bin
15 ENV FLASK_APP "/phyapps/application.py"
16
17 EXPOSE 5050
18
19 CMD ["sh", "-c", "python application.py"]

NORMAL  Dockerfile                    doc…   5% ☰   1/19 ㏑ :  1
"Dockerfile" 19L, 439C
```

Figure 5: Dockerfile for building `gateway` Docker container image.

phyapps services The containerization for the phyapps computing service is a bit complicated. First the container so-called `phantasy-base` [19] is created based on the image `debian:stretch-slim` [20], and then installing all the physics applications software packages for Debian Stretch OS, which have already been packaged for the development at FRIB [7]. Then based on `phantasy-base` image, `phyapps:va` and `phyapps:nb` are created, with the additional of the Jupyter-Notebook and virtual accelerator service [21].

USE CASE

Once the Docker Swarm stack is running, users can connect from the web browser. For the new user, register with 'Sign Up' form, then in the user page, new phyapps computing service can be created. Figure 6 shows the user has all the control of the phyapps computing services.

Once created, the computing service can be started and a new private Jupyter-Notebook workspace is ready for work with, by simply clicking the 'Notebook URL' in the 'Activity' page. Each user can only successfully be redirected to their own service, since a secret TOKEN used as the Jupyter-Notebook service authentication is generated when the private container is created.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA167

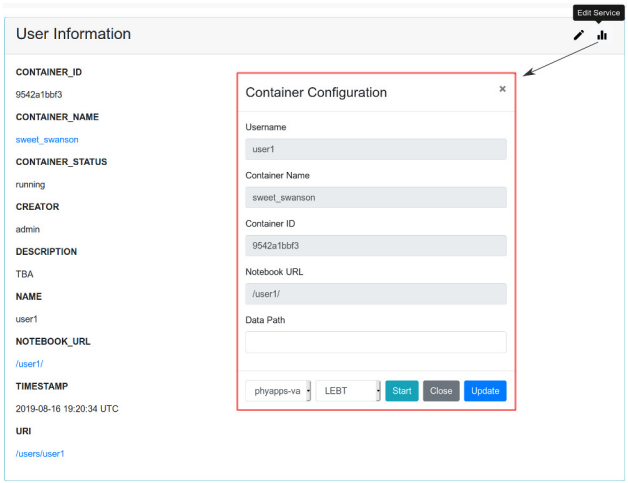

Figure 6: The User page of 'user1' in `gateway` app, where 'user1' can start a new computing service w/ or w/o FRIB virtual accelerator service at LEBT, MEBT or LINAC segment.

In the Jupyter-Notebook workspace, the FRIB LEBT segment could be modelled with the live settings from virtual accelerator. Firstly, the following scripts should output the computing service information:

```
>>> import os, platform
>>> uname = os.uname()
>>> print("Hostname: {},
>>>       System: {},
>>>       Dist: {}".format(
>>>       uname.nodename,
>>>       uname.sysname,
>>>       ' '.join(platform.dist())))
```

The output: `Hostname: 9542a1bbf328, System: Linux, Dist: debian 9.9`, indicate that the running container ID is `9542a1bbf328` (check by command `docker container ps`), OS is Debian 9.9 Linux.

Then all the script that developed for online modeling could be input for testing. For example, the following snippet shows how to use `phantasy` to instantiate the whole LEBT segment of FRIB virtual accelerator (machine name is `FRIB_VA`).

```
>>> from phantasy import MachinePortal
>>> mp = MachinePortal(machine="FRIB_VA",
>>>                    segment="LEBT")
>>> lat = mp.work_lattice_conf
```

And pull the live device settings of virtual accelerator to the model environment by `lat.sync_settings()`, then run the FLAME model by `path, fm = lat.run()`, which will output a tuple of generated FLAME lattice file (`path`) and the FLAME model object (`fm`) [7]. The FLAME model object can be used to do further studies, e.g. show the beam envelope by `plot_orbit(('pos', 'xrms'), ('pos', 'yrms'), flame_model=fm)`, Fig. 7 shows the envelope plot [22].

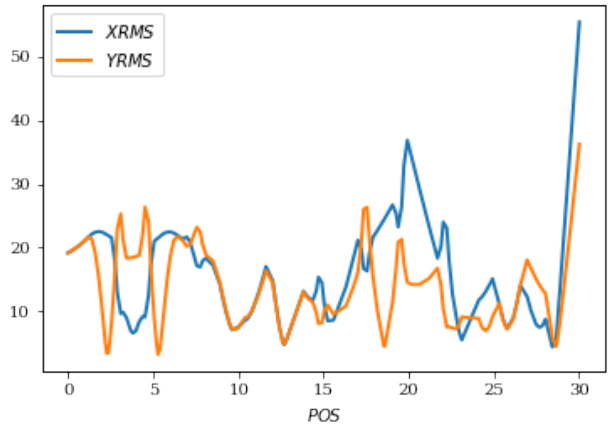

Figure 7: Online modeled envelope of LEBT segment of FRIB virtual accelerator with FLAME.

The user can close the browser and resume the work at any time later, all the work done in the Jupyter-Notebook stays until the computing service is killed by the system administrator. Multiple users can log onto 'phyapps-cloud' to do the work at the same time, to develop any machine tuning algorithms with less effort.

CONCLUSION

In this paper, we present the details of a web-based cloud computing platform for the physics applications development for FRIB driver LINAC. The well-designed `gateway` service is created to serve as the portal of 'phyapps-cloud', from where, multi-users can log on to the Python scripting environment provided by a private Jupyter-Notebook service, which includes the FRIB virtual accelerator and preinstalled `phantasy` framework. All the developed services are Docker containerized, composed Docker compose YAML configuration and Makefile files to make the Swarm stack deploy in just single line in any Linux workstation. All the network access is managed by the configurable http proxy service, to make the whole system secure.

It is also worth mentioning that 'phyapps-cloud' swarm could be deployed to the commercial cloud service providers, e.g. Amazon AWS [23], Microsoft Azure [24], Google Cloud Platform [25], etc. Since the UI of `gateway` web app is developed with Bootstrap V4 [26], which is mobile friendly, the users can still have a good user-experience in a mobile devices like smart phone, tablet, etc., such kind of machine tuning style would be truly the so-called accelerator-control-in-the-palm.

ACKNOWLEDGMENTS

The authors would like to thank D. Chabot and M. Konrad for useful discussions.

REFERENCES

[1] EPICS, `https://epics-controls.org`

[2] VirtualBox, `https://www.virtualbox.org`

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-MOPHA167

[3] GitHub repository of 'phyapps-cloud' project, https://github.com/archman/phyapps-cloud

[4] Docker, https://www.docker.com

[5] Configurable HTTP proxy, https://github.com/jupyterhub/configurable-http-proxy

[6] JupyterHub project, https://github.com/jupyterhub

[7] T. Zhang *et al.*, "High-level Physics Controls Applications Development for FRIB", presented at ICALEPCS '19, NY, USA, October 2019, paper TUCPR07, this conference.

[8] FLASK project, https://flask.palletsprojects.com

[9] Python, https://www.python.org

[10] Z. He *et al.*, "The fast linear accelerator modeling engine for FRIB online model service", *Computer Physics Communications* , vol. 234, pp.167 - 168, 2019, doi:10.1016/j.cpc.2018.07.013

[11] Docker Swarm, https://docs.docker.com/engine/swarm

[12] PAAS, https://en.wikipedia.org/wiki/Platform_as_a_service

[13] Docker compose YAML file, https://docs.docker.com/compose

[14] Swarm initialization, https://docs.docker.com/engine/reference/commandline/swarm_init

[15] Makefile for 'phyapps-cloud' deployment, https://github.com/archman/phyapps-cloud/blob/master/Makefile

[16] Portainer project, https://www.portainer.io

[17] Docker cloud, https://hub.docker.com

[18] Docker image for 'gateway', https://cloud.docker.com/repository/docker/tonyzhang/phyapps-gateway

[19] Docker image for 'phantasy-base', https://cloud.docker.com/repository/docker/tonyzhang/phantasy-base

[20] Official site for Debian docker images, https://hub.docker.com/_/debian

[21] Docker image for 'phyapps' computing service, https://cloud.docker.com/repository/registry-1.docker.io/tonyzhang/phyapps

[22] GitHub repository of 'flame-utils' project, https://github.com/frib-high-level-controls/flame-utils

[23] Amazon AWS, https://aws.amazon.com

[24] Microsoft Azure, https://azure.microsoft.com

[25] Google cloud platform, https://cloud.google.com

[26] Bootstrap 4.3, https://getbootstrap.com/docs/4.3