2022 International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON)
Karnataka, India. Dec 23-25, 2022

2022 International Conference on Smart Generation Computing, Communication and Networking (SMART GENCON) | 978-1-6654-5499-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/SMARTGENCON56628.2022.10084107

Study of container-based JupyterLab and AI Framework on HPC with GPU usage

Mandeep Kumar
University Institute of Computing
Chandigarh University
Mohali, India
Locuz Enterprise Solutions Ltd
Delhi, India
https://orcid.org/0000-0002-4851-4719
themandeepkumar@gmail.com

Gagandeep Kaur
University Institute of Computing
Chandigarh University
Mohali, India
https://orcid.org/0000-0002-1513-8446
gagandeepkaurlogani@gmail.com

***Abstract*—Container technologies enable portability and reproducibility for applications. A containerized application or service can be quickly prototyped, built, and deployed. Artificial Intelligence (AI) workloads with large datasets are driven by High Performance Computing (HPC) with Graphics Processing Unit (GPU) usage to expedite solving more complex real-world problems. The most widely used container technology is Docker, but due to the root access needed to run Docker containers and related security issues, it cannot be easily adopted in secure HPC systems. Singularity meets the needs of the HPC environment and permits the use of identical Docker containers without alterations to address security issues. This work presents an approach for execution of container-based JupyterLab and AI framework with workload manager using user credentials on HPC with GPU usage. We provide comprehensive performance evaluations of JupyterLab and AI framework containers transformed from Docker to Singularity to solve more complex real-world problems with GPU in an HPC environment in the context of native.**

Keywords—Artificial Intelligence, Containerization, Docker, Graphics Processing Unit, High Performance Computing, JupyterLab, Performance, Portability, Singularity, SLURM, Workload Manager

I. Introduction

Nowadays, technology has reached the point where High Performance Computing (HPC) with Graphics Processing Unit (GPU) usage drives Artificial Intelligence (AI) workloads with large datasets to speed up the process of solving more complex real-world problems. The DevOps trend has aided the introduction of container-based deployment techniques into the software development and application deployment. Container enables portability and reproducibility for applications. A containerized application or service can be quickly prototyped, built, and deployed by a developer. The ability for users to create runtime environments specific to their workloads makes containers interesting to the scientific community because it enables data scientists to efficiently manage their AI workloads with portability and reproducibility.

The most widely used container technology is Docker, but due to the root access needed to run Docker containers and related security issues, as well as the lack of native support for HPC workload managers, it cannot be integrated in secure HPC systems [1]. Singularity meets the needs of the HPC environment and permits the use of identical Docker containers without alterations to address security issues [2].

JupyterLab is the most recent web-based interactive development environment for notebooks, code, and data [3]. A modular design encourages expanding and enhancing functionality, while providing a straightforward, efficient, document-centric experience. Artificial intelligence frameworks are usually run on workstations in interactive mode, and JupyterLab are widely used. Interactive execution mode, on the other hand, is not the typical way to run applications on HPC environments that use a workload manager to submit workloads. The main contributions of this work are,

1. We examine some of the latest containerization work for AI workloads and HPC.
2. We present the approach for execution of container-based JupyterLab and AI framework with workload manager using user credentials on HPC with GPU usage.
3. We provide comprehensive performance evaluations of JupyterLab and AI framework containers transformed from Docker to Singularity to solve more complex real-world problems with GPU in an HPC environment in the context of native.

II. Literature Review

In the literature, previous studies on this area can be found that discuss containerization techniques for HPC environments. Canon et al. [4] addressed several difficulties with using containers for HPC applications as well as the current methods employed by many HPC container runtimes. Newlin et al. [5] examined the use of advanced research computing containers using the MLPerf benchmark to characterize the performance overhead of Singularity containers.

Rudyy et al. [6] compared native performance with Docker, Shifter, and Singularity in an HPC environment, where Singularity outperformed. Beltre et al. [7] examined the potential of Kubernetes for HPC and compared its capabilities and effectiveness with Docker Swarm and bare metal HPC application execution. Brayford et al. [8] discussed the challenges of deploying AI frameworks in a secure HPC environment and used Charliecloud to deploy AI frameworks.

Höb et al. [9] proposed a framework to automatically deploy optimized container computations with minimal overhead, which minimizes the time a scientist spends configuring job submissions manually. Chakraborty et al. [10] introduced Popper, a container native workflow engine

that does not rely on a Kubernetes cluster or any other services besides a container engine like Docker or Podman. Misale et al. [11] examined the issue of effectively running HPC workloads on Kubernetes clusters and contrasted KubeFlux with the default scheduler of Kubernetes on a Red Hat OpenShift cluster on the IBM Cloud.

Martin et al. [12] proposed an open-source framework that allows the management of ML/AI pipelines via data streams, and its components managed entirely through containerization technologies. Kumar et al. [13] profiled container-based MPI applications on InfiniBand-based HPC and discussed the approaches for developing and running containerized applications, in addition to comparing performance with bare metal.

This work contributes in many aspects to other works of literature. We present the approach for execution of container-based JupyterLab and AI framework with workload manager using user credentials on HPC with GPU usage. We provide comprehensive performance evaluations of JupyterLab and AI framework containers transformed from Docker to Singularity to solve more complex real-world problems with GPU in an HPC environment in the context of native.

III. Container Execution

In this section, we describe an approach to running containers with workload manager using user credentials on an HPC with GPU usage.

Singularity makes it possible to continue using the same Docker container without making any changes or being concerned about security. It is simple to transform a Docker image into a Singularity image using the *singularity pull* or *build* command. Use the *singularity run* or *exec* command with the *--nv* option to set up the container's environment for NVIDIA GPU support in order to execute containerized applications transformed from Docker to Singularity on HPC with GPU usage. Frameworks for artificial intelligence are typically run in interactive mode on workstations. Interactive execution mode, on the other hand, is not the typical way to run applications on HPC environments that use a workload manager to submit workloads. Singularity can be used in the HPC workload manager script. We are using SLURM as workload manager to run Singularity on HPC with GPU usage [14]. The container conversion from Docker to Singularity on HPC is shown in List 1, and the SLURM script for Singularity container execution on HPC with GPU usage is shown in List 2. The log file contains the JupyterLab access details. To access the JupyterLab server, copy the URL from the log file and use the corresponding HPC IP address. The process of Singularity conversion and execution on HPC with GPU usage is depicted in Fig. 1.

List 1. Container conversion from Docker to Singularity on HPC.

```
module load singularity/3.10.0
singularity build tensorflow-22.10-tf2-py3.sif docker://nvcr.io/nvidia/tensorflow:22.10-tf2-py3
```

List 2. SLURM script for Singularity container execution on HPC with GPU usage.

```
#!/bin/bash
#SBATCH --job-name=jupyterlab
#SBATCH --ntasks=8
#SBATCH --gres=gpu:8
#SBATCH --partition=debug
module load singularity/3.10.0
singularity run --nv /opt/apps/sif/tensorflow-22.10-tf2-py3.sif jupyter lab --no-browser --ip=0.0.0.0 &> jupyterlab.log
```

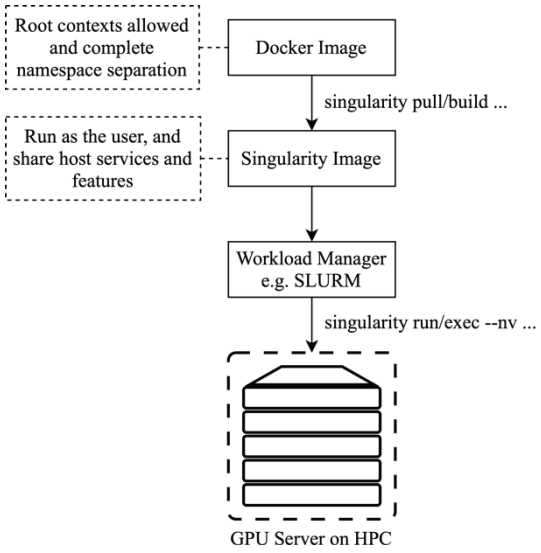

Fig. 1. Process of Singularity container execution on HPC with GPU usage.

IV. Performance Evaluation

In this section, we measure the performance of containers transformed from Docker to Singularity on HPC with GPU access in the context of native.

The HPL benchmark, a software package that enables the solution of (random) dense linear systems using double-precision (64 bit) arithmetic on distributed memory systems, is frequently used by the HPC community to assess system performance [15]. The NVIDIA HPC-Benchmarks docker container offers the widely known HPL benchmarks, which have been tuned for usage with NVIDIA accelerated HPC systems [16]. We created a Singularity image for the HPL benchmark optimized for performance on NVIDIA accelerated HPC systems and used it to run HPL benchmarks on GPU sever in HPC. For comprehensive performance evaluation of GPU Server in HPC, we performed HPL with a Singularity container. The HPL FP64 (non-tensor) performance is compared to the native and the theoretical peak FP64 (non-tensor) performance, which is depicted in Fig. 2.

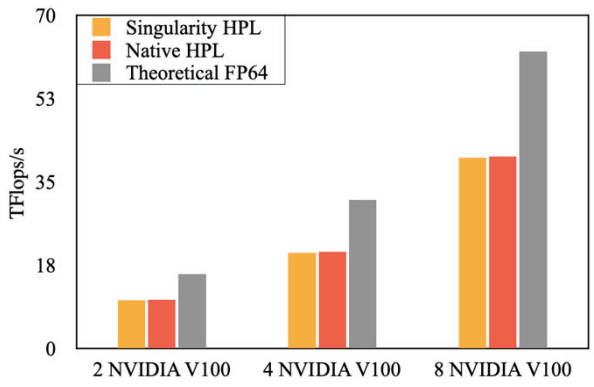

Fig. 2. Comparison of HPL FP64 (non-tensor) benchmark performance on HPC with GPU usage in terms of native and theoretical peak FP64 (non-tensor) performance with a container transformed from Docker to Singularity.

We created a Singularity image with the most recent JupyterLab and TensorFlow docker container, which is optimized for the NVIDIA V100 GPU, and used it to train an AI model on GPU server in HPC [17]. For a comprehensive performance comparison, we installed the NVIDIA V100 GPU optimized JupyterLab with TensorFlow natively on GPU server in HPC. TensorFlow CNN benchmark is being used [18]. We used SLURM workload manager to submit the container-based JupyterLab and AI framework to the GPU server in HPC and executed the JupyterLab singularity image on the browser with user credentials. JupyterLab notebook as well as terminal with Singularity on GPU server in HPC with GPU access and an auto-mounting user home with credentials are depicted in Fig. 3 and Fig. 4, respectively. AI model training using JupyterLab notebook in Singularity on GPU server in HPC is depicted in Fig. 5.

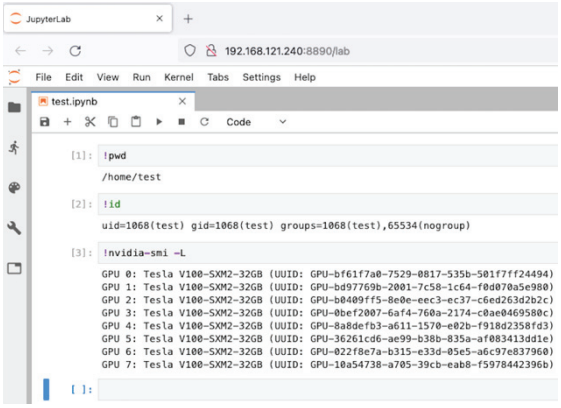

Fig. 3. JupyterLab Notebook in Singularity on HPC with GPU access and an auto-mounting user home with credentials.

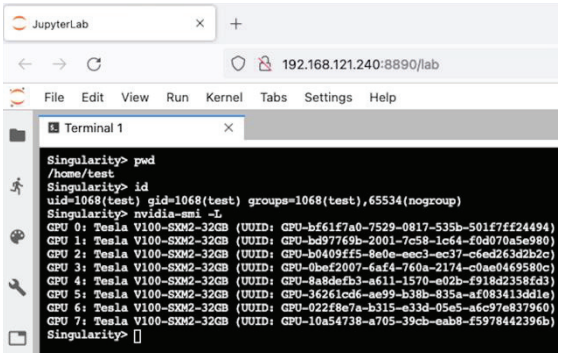

Fig. 4. JupyterLab Terminal in Singularity on HPC with GPU access and an auto-mounting user home with credentials.

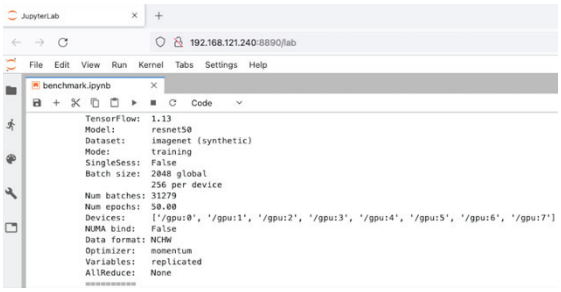

Fig. 5. AI model training using JupyterLab Notebook in Singularity on HPC with GPU usage.

The performance comparisons in terms of time and train images of ResNet50 with ImageNet (synthetic) on HPC with and without the JupyterLab Singularity container with GPU usage are depicted in Fig. 6 and Fig. 7, respectively. When using JupyterLab and TensorFlow in an isolated Singularity container environment to train AI models like ResNet50, we observed that the performance overhead is within the normal range of variation. We used user credentials to run JupyterLab in Singularity and found no need for root credentials to run containerized AI frameworks.

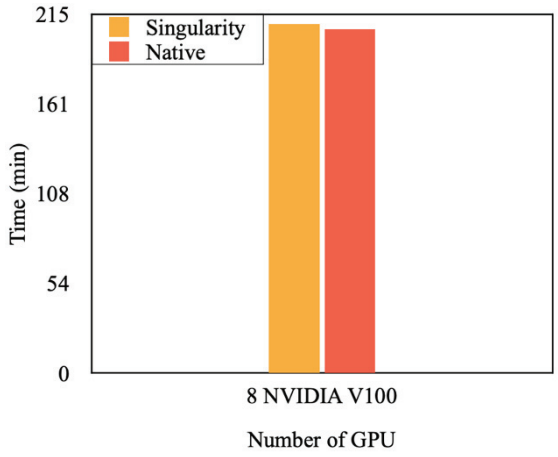

Fig. 6. Execution time of ResNet50 with ImageNet (synthetic) with and without the JupyterLab Singularity container on HPC with GPU usage.

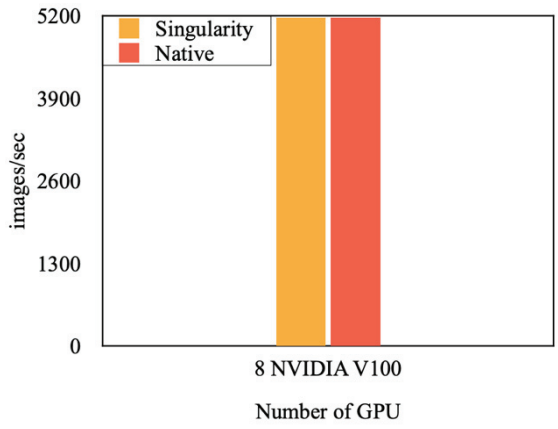

Fig. 7. Total train images of ResNet50 with ImageNet (synthetic) with and without the JupyterLab Singularity container on HPC with GPU usage.

V. Conclusion

In this work, we presented the approach for JupyterLab and AI framework running in an isolated container environment with a workload manager using user credentials on HPC with GPU usage. We provided comprehensive performance evaluations of JupyterLab and AI framework containers transformed from Docker to Singularity to solve more complex real-world problems with GPU in an HPC environment in the context of native. We found that using JupyterLab and the AI framework in an isolated Singularity container environment to train AI models like ResNet50 on HPC with GPU usage in the context of native, the performance overhead is within the normal range of variation. We used user credentials to run Singularity and found that there is no need for root credentials to run a containerized JupyterLab and AI Framework, so we can easily adopt it in a secure HPC environment. Singularity continues to utilize the same Docker containerized AI workloads as before, assuring security without the

performance overhead of AI workflow containerization to address more challenging real-world problems.

REFERENCES

[1] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, 2014(239), 2.

[2] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PLoS ONE* 12(5): e0177459, 2017.

[3] "JupyterLab: A Next-Generation Notebook Interface," [Online]. Available: https://jupyter.org

[4] R. S. Canon and A. Younge, "A Case for Portability and Reproducibility of HPC Containers," in *Proc. 2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, 2019, pp. 49-54.

[5] M. Newlin, K. Smathers, and M. E. DeYoung. "ARC Containers for AI Workloads: Singularity Performance Overhead," in *Proc. Practice and Experience in Advanced Research Computing on Rise of the Machines (learning) (PEARC '19).* Association for Computing Machinery, New York, NY, USA, Article 1, 1–8.

[6] O. Rudyy, M. Garcia-Gasulla, F. Mantovani, A. Santiago, R. Sirvent, and M. Vázquez, "Containers in HPC: A Scalability and Portability Study in Production Biological Simulations," in *Proc. IEEE International Parallel and Distributed Processing Symposium (IPDPS*), 2019, pp. 567-577.

[7] A. M. Beltre, P. Saha, M. Govindaraju, A. Younge and R. E. Grant, "Enabling HPC Workloads on Cloud Infrastructure Using Kubernetes Container Orchestration Mechanisms," in *Proc. 2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, 2019, pp. 11-20.

[8] D. Brayford, S. Vallecorsa, A. Atanasov, F. Baruffa, and W. Riviera, "Deploying AI Frameworks on Secure HPC Systems with Containers," in *Proc. IEEE High Performance Extreme Computing Conference (HPEC),* 2019, pp. 1-6.

[9] M. Höb, and K. Dieter, "Enabling EASEY Deployment of Containerized Applications for Future HPC Systems," *In: Krzhizhanovskaya V. et al. (eds) Computational Science – ICCS 2020. ICCS 2020. Lecture Notes in Computer Science,* vol 12137. Springer, Cham.

[10] J. Chakraborty, C. Maltzahn and I. Jimenez, "Enabling Seamless Execution of Computational and Data Science Workflows on HPC and Cloud with the Popper Container-native Automation Engine," in *Proc. 2020 2nd International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC*), 2020, pp. 8-18.

[11] C. Misale, M. Drocco, D. J. Milroy, C. E. A. Gutierrez, S. Herbein, D. H. Ahn and Y. Park, "It's a Scheduling Affair: GROMACS in the Cloud with the KubeFlux Scheduler," in *Proc. 2021 3rd International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)*, 2021, pp. 10-16.

[12] C. Martín, P. Langendoerfer, P. S. Zarrin, M. Díaz, and B. Rubio, "Kafka-ML: Connecting the data stream with ML/AI frameworks," *Future Generation Computer Systems,* vol. 126, 2022, pp. 15-33.

[13] M. Kumar and G. Kaur, "Containerized MPI Application on InfiniBand based HPC: An Empirical Study," in *Proc. 2022 3rd International Conference for Emerging Technology (INCET)*, 2022, pp. 1-6.

[14] M. A. Jette, A. B. Yoo and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Proc. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2003, pp. 44-60.

[15] A. Petitet, R. Whaley, J. Dongarra and A. Cleary, "HPL – a Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers," 2008, [Online]. Available: https://www.netlib.org/benchmark/hpl/

[16] "NVIDIA NGC Container HPC-Benchmarks," [Online]. Available: https://catalog.ngc.nvidia.com/orgs/nvidia/containers/hpc-benchmarks

[17] "NVIDIA NGC Container Tensorflow," [Online]. Available: https://catalog.ngc.nvidia.com/orgs/nvidia/containers/tensorflow

[18] "TensorFlow benchmarks," [Online]. Available: https://github.com/tensorflow/benchmarks