ISSN 2278-3091
Volume 9, No.4, July – August 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse166942020.pdf
https://doi.org/10.30534/ijatcse/2020/166942020

Containerization for shipping Scientific Workloads in Cloud

Manish Kumar Abhishek[1], D. Rajeswara Rao[2]
[1]Department of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, India,
manish.abhishek@gov.in
[2]Department of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, India, rajeshpitam@gmail.com

ABSTRACT

In Cloud computing environment, researchers are actively looking for opting containerization technology using various tools to achieve better performance in High Performance Computing (HPC) applications execution. In virtualization world, containers are getting more popular and found suitable in comparison of virtual machines as they are giving better performance. Along with agility, support micro services and integrated easily with management and monitoring tools. To have containerization in place, Docker is one of the most suitable open source platforms with Operating System (OS) level virtualization. Singularity is also one of the popular solutions to work with HPC applications. Before directly start using the technologies and tools, they must be analyzed, explored and have a proof of concept for performance first. This paper presents (1) Containers evaluation using open source platform Singularity and Docker (2) monitoring of containers using orchestration and monitoring system Kubernetes (3) feature analysis of scientific workloads using Containers in cloud. Feature analysis reports having performance result sets are primarily aimed for helping the DevOps on making the decisions for choosing the right technology and tools to run their parallel and high performance computing applications.

Key words: Cloud Computing, Docker, High Performance Computing, Kubernetes, Singularity, Virtualization.

1. INTRODUCTION

Containerization is totally based on operating system virtualization where all the required libraries and dependencies are going to be bundled in underlying layer. During the deployment of an application generally developer does not think from future perspective and over a period, a requirement getting changed and same application is going to be patch or grows with addition of feature implementation. After few years, application code become legacy code and new developers must dig their mind even to fix a small issue. To overcome this kind of monolith behavior, micro services came into picture that plays an important role in development and deployment of an application. To deploy micro services, Docker containers are going to be best choice which will smooth the life cycle of software deployment [1]. It will maintain not only upgrades, but also fix packs, patches that need to be delivered to customer seamlessly. It will provide a better testing environment via replacing the physical servers in form of containers having same dependencies, libraries bundled along with OS layer. There is no need to use hypervisors like KVM, Xen etc. Containers will provide the smooth environment for application maintenance, migration of micro services via tagging Docker images time to time and agility, continuous integration and delivery of software implemented using various languages and several platforms. During their starting phases, containers were very well adopted in micro service kind of architecture instead of parallel and high-performance computing applications like MPI etc. for research domain. They were not primarily target for handling scientific workloads. As micro service architecture has been adopted by IT industry very well, containers behavior of portability, reproducibility, elasticity and scalability pushed its usage to try and evaluate its performance in HPC community in research area. HPC was already popular with its experiments in Cloud computing area and here entrance of Docker gives it another positive insight to achieve better performance as both core components were same that is none other than Virtualization. Containers are based on the OS level kind of virtualization instead of having full server virtualization which makes it very light weight, scalable, agile in nature and easily adaptable for software development and deployment in emerging technologies.

Docker containers are meant for providing I/O, interconnect network capabilities and independent computational resources where Singularity containers are dedicated for HPC application execution. In HPC environment, scheduler is required for scheduling the jobs being queued and Slurm is one of the popular one. It has been identified that YARN [2] and Mesos [3] are also make their place as a suitable resource manager in HPC. Apart from these open source technologies, Kubernetes orchestrator also embarks their journey in HPC community. One of the major benefits from containers is fault tolerance. MPI applications can be easily run with Singularity usage where isolation of execution can be achieved via Docker. Singularity focus is to coarse grained the computing resources instead of doing fine grain rank basis allocation supported by Docker. During the starting phase of cloud computing, industry was worried about the security [4] but as it addresses all the challenges via opting root privileges to run application along with different aspects like load balancing

allows cloud computing to "scale up to increasing demand" [5], Docker containers make their place in this era for executing HPC applications. But on other hand orchestration, containers placement, performance, scheduling of jobs, allocation of computing resources was still a concern and need to be quantified.

This paper includes, 1) Containers evaluation using open source platform Singularity and Docker (2) monitoring of containers using orchestration and monitoring system Kubernetes (3) feature analysis of scientific workloads using Containers in cloud. Our experimental results are captured via maintaining private cloud infrastructure using OpenStack which offers the provisioning of Docker containers using Cent OS 7 version as OS. Within HPC cluster we are running MPI application. For doing evaluation, we have considered:

- Run the MPI application on Docker containers vs bare metal and computed the performance.
- Using Kubernetes, overlay network has been established. Each node within HPC cluster is going to have one container lying over the same overlaying network layer but with different IP addresses.
- One stack built with multiple containers on a single node connected with same network layer.
- For benchmarking, we have used different set of computing resources for MPI application in form of different classes. Testbeds are going to have result report with constant count of MPI rankings which clearly showing the variance during the placement of nodes or allocation of nodes changes within HPC cluster.

2. MATERIAL AND METHODS

We have explored and analysis the various Open source technologies in terms of virtualization, orchestration, monitoring, schedulers, open resource managers and picks Docker containers, Kubernetes, Singularity respectively for having all as a combination to quantifying the performance of HPC applications. Figure 1 shows the containerization approach for HPC.

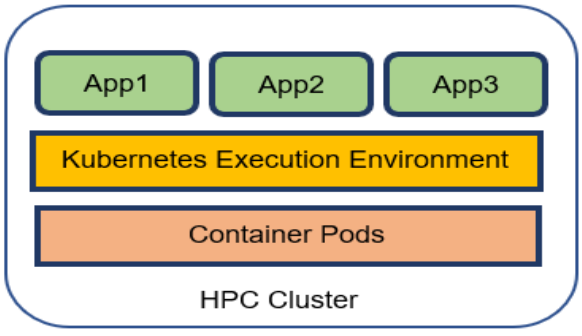

Figure 1: Containerization approach for HPC

A. Docker Containers

In Linux systems, to support isolation from host; Linux containers provider one layer of virtualization over the Operating system so that multiple processes can run on single host and this idea is leverage by the Docker to build, deploy and shipment of containers across nodes. It separates the execution environment in form of light weight virtual machine [6] along with fixed computational resources and network bandwidth. in short, it provides the encapsulation and abstraction of underlying guest OS instead of from the underlying layer of hardware. It provides virtual network to interconnect with other containers deployed on same node with the help of private network addresses. With the help of CLI commands containers can be easily provisioned and managed with the help of container images.

B. Singularity Containers

Singularity containers are targeted for coarse grained resources allocation rankings as well as its in-built support for MPI application integration makes its popular for scientific workloads in HPC community [7]. It is facilitating a mechanism for application packaging in addition to execution environment. Singularity containers will run like an individual normal process on computing node instance of HPC cluster that makes its integration quick with scheduler. Instead of keeping images in file format like Docker does, it will persist the whole image as a single file. It has its hub where images can be registered in repository having support of CRUD operation which can be public in nature.

C. Kubernetes

Kubernetes is an open source orchestration tool [9] used to manage, monitor, automate the deployed container pods for a better prediction of hosting the containers as well as to easily scale up or down the container pods within cluster. Kubernetes master node is going to manage all the deployed pods via command line interface kubectl. It is easily integrated with Docker or singularity containers along with computing resource specifications. It is very beneficial for handling the scientific workloads in form of running process as individual containers.

3. RESULTS AND DISCUSSION

3.1 Experimental Setup

Experimental testbeds have included four bare-metal compute instances using OpenStack cloud infrastructure. Table 1 is having details of our stacks.

Table 1: Test Bed Specifications

Resource	Details
Operating System	CentOS 7.0
Processors Count	12 X Intel Xeon X5650 @2.67GHz
Memory	226GB
Network	Emulated 1GigE
CPU	32 cores
RAM	100GB
OPEN MPI	3.0.0

Instances are configured with 32 cores of CPU and 226 GB, 100GB in terms of memory and RAM. To understand the features of MPI application, it is profiled for benchmarking. Later, we have move towards the setup of containers to derive the variance in performance factor. Computational work has been computed via HPCG and from memory perspective; we have used KMI Hash [10]. We have used iterative approach with a count of eight to find out the average of each experimental test.

3.2 Evaluation

MPI application is in form of Singularity image has been deployed in form of Docker containers bundled with needful MPI libraries running within HPC cluster on the top of OpenStack [11] Cloud and all container pods within HPC cluster has been managed via Kubernetes master node. Using ansible script, spawn the whole cluster via helm install and later monitored via Rancher that is an open source platform to address the security and management challenges across Kubernetes clusters for better handling of container workloads within HPC cluster. We have done settings for multiple containers in such a way that it will get configured per computing node and split the ranks of MPI among containers throughout nodes. In the starting phase one container is aimed for only one MPI rank and slowly grows the ranking count per container and keeping the total count a constant value. Pods were equally spread across computing nodes where every container is going to have similar count of MPI rankings.

A. HPCG

We have evaluated the performance factor via HPCG benchmarking with a variance of MPI ranks. In the first set, we have considered 64 ranks with bare metal physical server, then one Docker container with its own underlying overlay network and with singularity container. In second set of execution, we have considered 64 MPI ranks spread across the containers and in last set evaluated the 18 MPI ranks with variance in count of computing nodes within HPC cluster. Figure 2 (a), 2 (b) and 2 (c) are showing the difference of performance computation having all the considered use cases.

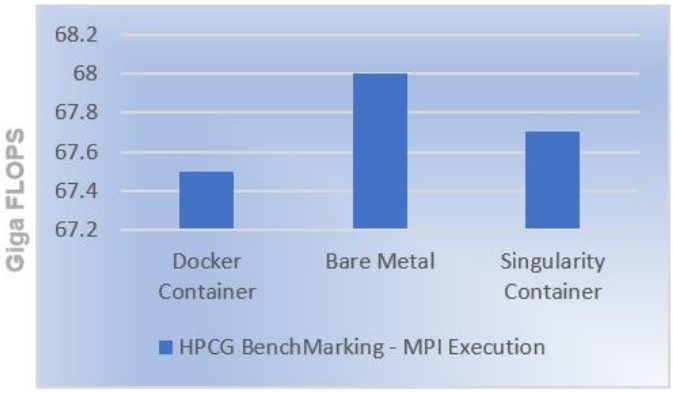

Figure 2(a): MPI execution strategies

We can clearly see that results are comparable where Docker container with overlay network is having higher performance downgrade rather than singularity container which slightly degrading the performance in comparison of bare metal.

Figure 2(b): MPI rankings per container

Evaluating MPI ranks per container, we found that when rank as one, performance was poorly downgraded in comparison to bare metal but once MPI ranking has been incremented by a factor of 2 or four, its performance become equivalent to bare metal.

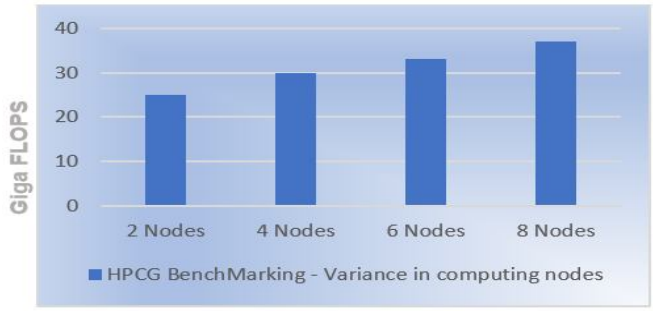

Figure 2(c): MPI rankings with variance of nodes

Incrementing the number of nodes in HPC results in higher performance. When within cluster, number of nodes in form of container pods has been increased; we gain a hit in performance around 30%. Incrementing from four to eight nodes, it was even better and having a gain of approx. 8%.

B. KMI Hash

From memory perspective we have done benchmarking using KMI hash which is data centric in nature and using mainly hashing technique. When we have experimented, the tests covering similar use cases that we have computed in case of HPCG benchmarking, found that Docker containers are providing similar throughput in comparison with bare metal. On another hand, Singularity containers provides within 0.5%.

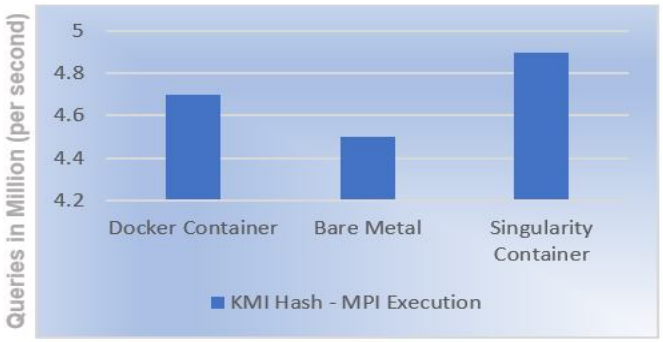

Figure 3(a): MPI execution strategies

Figure 3(b): MPI rankings per container

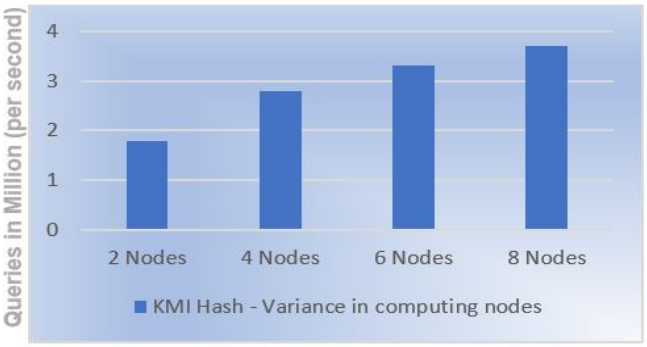

Figure 3(c): MPI rankings with variance of nodes

Figure 3(a), 3(b) and 3(c) shows clearly the comparison of performance factor in case of different strategies. Here, Docker container is performing well in comparison to bare metal throughput where Singularity containers are ahead from the Docker containers. Having one MPI rank does not have significant results but with increment in number of pods in terms of nodes has significantly improved the performance where total count of ranking was a constant value that is 28.

3.3 Related Work

In IT industry, Docker containers are now a buzz but due to its root privileges, it is not fully adapted in HPC environment. However, it has been overcome via developing a secure execution of containers in HPC environment using Slurm [12] scheduler but still containers throughput and network bandwidth is a concern. Singularity is now a days very popular for handling the HPC application. It is mainly for handling the scientific workloads only and its in-built integration support is one of the major reasons to use it for running HPC applications.

4. CONCLUSION

Containers can be easily used for achieving portability in HPC applications. They are very flexible, easy to use and managed easily in terms of computing nodes for HPC application. We have performed our experimental results with different benchmarks having mainly three use cases around MPI rankings and number of queries in terms of memory computation. Different containers running within HPC cluster has provided flexibility with a minimum overhead of performance. The results were mostly comparable irrespective of bare metal. Singularity containers performed very well to support MPI application but splitting of MPI rankings per container having restriction on allocation of resources is not supported but it can be achieved via Docker easily. On another hand in case of Singularity, there is no need of external installation of interconnected drivers to have fast communication across pods where Docker needs it. Performance using HPCG and KMI hash has been computed and shred the results where Docker containers are having its own requirement to handle the big data and its traits in public sector [13] and singularity is going to have its own. Mainly Singularity containers are aimed for handling scientific workloads but still there are open items in this area that can be addressed in future. The dynamic allocation of resources can be targeted as a future scope to avoid the overhead of appropriate resource utilization and how we can use the free computational resources for running non -HPC applications. A scalable framework can add more value to it.

ACKNOWLEDGEMENT

A special vote of thanks to the Koneru Lakshmaiah Education Foundation for supporting and facilitating me required infrastructure to carry out the analysis work and my guide as well as staff members who have helped me to analyze and complete this research work.

REFERENCES

1. Sarita and S. Sebastian, "**Transform Monolith into Microservices using Docker**," 2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA), Pune, 2017, pp. 1-5, doi: 10.1109/ICCUBEA.2017.8463820.
2. Vavilapalli, Vinod & Murthy, Arun & Douglas, Chris & Agarwal, Sharad & Konar, Mahadev & Evans, Robert & Graves, Thomas & Lowe, Jason & Shah, Hitesh & Seth, Siddharth & Saha, Bikas & Curino, Carlo & O'Malley, Owen & Radia, Sanjay & Reed, Benjamin & Baldeschwieler, Eric. (2013). **Apache Hadoop YARN: yet another resource negotiator. Proceedings of the 4th Annual Symposium on Cloud Computing**, SoCC 2013. 10.1145/2523616.2523633.

3. B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: **A platform for fine-grained resource sharing in the data center**," in Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 295–308: http://dl.acm.org/citation. cfm?id=1972457.1972488.
4. P. Patel, V. Tiwari and M. K. Abhishek, "**SDN and NFV integration in openstack cloud to improve network services and security**," 2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), Ramanathapuram, 2016, pp. 655-660, doi: 10.1109/ICACCCT.2016.7831721R.
5. Karimunnisa, Syed & Kompalli, Vijaya. (2019). "**Cloud Computing: Review on Recent Research Progress and Issues**". International Journal of Advanced Trends in Computer Science and Engineering. 8. 216-223. 10.30534/ijatcse/2019/18822019.
6. Dirk Merkel. 2014. **Docker: lightweight linux containers for consistent development and deployment**. Linux Journal 2014, 239 (2014), 2.
7. V.Sande Veiga et al.,"**Evaluation and Benchmarking of Singularity MPI containers on EU Research e-Infrastructure**," 2019 IEEE/ACM International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC), Denver, CO, USA, 2019, pp. 1-10, doi: 10.1109/CANOPIE-HPC49598.2019.00006.
8. Hideto Saito, Hui-Chuan Chloe Lee, and Ke-Jou Carol Hsu. 2016. **Kubernetes Cookbook**. Packt Publishing.
9. Andrew J. Younge, Kevin Pedretti, Ryan E. Grant, and Ron Brightwell. 2017. **A Tale of Two Systems: Using Containers to Deploy HPC Applications on Supercomputers and Clouds**. In 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, 74–81. https://doi.org/10.1109/CloudCom.2017.40.
10. "**KMI HASH Benchmark Summary**" https://asc.llnl.gov/CORAL-benchmarks/Summaries/KMI_Summary_v1.1.pdf.
11. Kumar, Rakesh & Gupta, Neha & Charu, Shilpi & Jain, Kanishk & Jangir, Sunil. (2014). **Open Source Solution for Cloud Computing Platform Using OpenStack**. 10.13140/2.1.1695.9043.
12. Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. **SLURM: Simple Linux Utility for Resource Management**. Springer, Berlin, Heidelberg, 44–60. https://doi.org/10.1007/10968987 {_}3
13. Wook, Muslihah. (2020). **Big Data Analytics Application Model Based on Data Quality Dimensions and Big Data Traits in Public Sector**. International Journal of Advanced Trends in Computer Science and Engineering. 9. 1247-1256 10.30534/ijatcse/2020/53922020.
14. Beltre, Angel & Saha, Pankaj & Govindaraju, Madhusudhan & Younge, Andrew & Grant, Ryan. (2019). **Enabling HPC Workloads on Cloud Infrastructure Using Kubernetes Container OrchestrationMechanisms**.10.1109/CANOPIE-HPC49598.2019.00007.
15. S. Sreepathi et al., "**Application Characterization Using Oxbow Toolkit and PADS Infrastructure," 2014 Hardware-Software Co-Design for High Performance Computing**, New Orleans, LA, 2014, pp. 55-63, doi: 10.1109/Co-HPC.2014.11.
16. Park, J., Smelyanskiy, M.: **Optimizing Gauss-Seidel Smoother in HPCG**. In: ASCR HPCG Workshop, Bethesda MD, 25 March 2014.
17. Phillips, E.H., Fatica, M.: **Implementing the Himeno benchmark with CUDA on GPU clusters**. In: IEEE International Symposium on Parallel & Distributed Processing IPDPS, pp. 1–10 (2010).
18. R. Escobar and R. V. Boppana, "**Performance Prediction of Parallel Applications Based on Small-Scale Executions**," 2016 IEEE 23rd International Conference on High Performance Computing (HiPC), Hyderabad, 2016, pp. 362-371 doi: 10.1109/HiPC.2016.049.
19. E. Vermij, L. Fiorin, C. Hagleitner and K. Bertels, "**Boosting the Efficiency of HPCG and Graph500 with Near-Data Processing**," 2017 46th International Conference on Parallel Processing (ICPP), Bristol, 2017, pp. 31-40, doi: 10.1109/ICPP.2017.12.
20. Y. Demiral, N. Çarkacı and U. Çekmez, "Bulut Üzerinde **DevOps Mimarisi DevOps Architecture in the Cloud**," 2019 27th Signal Processing and Communications Applications Conference (SIU), Sivas, Turkey, 2019, pp. 1-4, doi: 10.1109/SIU.2019.8806433.
21. McCalpin, J.D.: **Memory bandwidth and machine balance in current high-performance computers**. In: IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, December 1995.
22. Dongarra, J., Heroux, M.A.: **Toward a New Metric for Ranking High Performance Computing Systems**. Sandia report SAND2013-4744 (2013).