J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr
Copyright ⓒ 2024 KSII

HPC 클러스터 기반 사용자 맞춤형 컨테이너 이미지 관리자 및 빌더☆

HPC Cluster-based Customized Container Image Manager and Builder

이 국 화[1*] 우 준[1] 홍 태 영[1]
Gukhua Lee Joon Woo Taeyoung Hong

요 약

본 논문은 고성능컴퓨팅 환경에서 사용자 맞춤형 컨테이너 이미지를 관리하고 구축하기 위한 새로운 접근 방식을 소개하며, 컴퓨팅 워크플로에서 유연성, 확장성 및 효율성에 대한 증가하는 요구사항 들을 해결하였다. 이 논문의 기여에는 컨테이너 기반 고성능컴퓨팅 인프라 내에서 사용자 맞춤형 컨테이너 이미지 관리자 및 빌더의 개발과 통합이 포함된다. 이 시스템을 통해 사용자는 맞춤화된 AI 서비스 플랫폼을 손쉽게 생성하고, 관리하고, 배포할 수 있어 필요한 패키지와 프레임워크를 구성하는 데 적은 시간과 노력을 들일 수 있다. 논문에서 개발한 이미지 관리자는 여러 사용자 요청을 동시에 처리하여 컴퓨팅 노드에서 작동하는 이미지 빌더에 작업을 효율적으로 분배할 수 있다. 이미지 빌더는 대기 중인 작업을 처리하고 실행 중인 인스턴스를 대상으로 사용자 맞춤형 컨테이너 이미지를 생성하고, 이러한 이미지를 프라이빗 컨테이너 레지스트리에 저장하도록 설계되어 원활한 액세스와 재사용성을 보장한다. 본 연구는 누리온 슈퍼컴퓨터와 뉴론 GPU 시스템을 포함한 고성능컴퓨팅 클러스터 기반 시스템에 구현하여 시스템의 기능 효과를 검증하여 실제 서비스 환경에서 확장성과 상호 운용성을 입증하였다. 또한 본 논문에서는 기존 컨테이너 기반 슈퍼컴퓨팅 프레임워크와의 원활한 통합을 보장하는 아키텍처와 메커니즘을 제안하여 리소스 활용을 최적화하고 AI 서비스 플랫폼의 배포 복잡성을 줄이는 역할을 하였다.

☞ 주제어 : 고성능컴퓨팅, 사용자 맞춤형 컨테이너 이미지, 이미지 관리자, 슈퍼컴퓨팅 환경

ABSTRACT

This paper introduces a novel approach for managing and building customized container images in high-performance computing (HPC) environments, addressing the growing need for flexibility, scalability, and efficiency in computational workflows. Our contributions include the development and integration of a custom container image manager and builder within a container-based HPC infrastructure. This system enables users to effortlessly create, manage, and deploy personalized AI service platforms, significantly enhancing the user experience by reducing the time and effort required to configure essential packages and frameworks. The image manager we developed is capable of processing multiple user requests concurrently, distributing tasks efficiently to image builders operating on compute nodes. Meanwhile, the image builder is designed to handle queued tasks, generate customized container images based on active instances, and store these images in a private container registry, ensuring seamless access and reusability. We validated our system's effectiveness by implementing it on HPC cluster-based systems, including the Nurion supercomputer and the Neuron GPU system, demonstrating its scalability and interoperability in real-world environments. Additionally, we established an architecture and mechanism that ensures seamless integration with existing container-based supercomputing frameworks, underscoring our system's capability to optimize resource utilization and streamline the deployment of AI service platforms.

☞ keyword : High-performance computing, Customized container image, Image manager, Supercomputing environment

1. Introduction

The increasing demand for HPC in various fields such as scientific research, data-intensive applications, and AI

[1] Korea Supercomputing Infrastructure Center, Korea Institute of Science and Technology Information, Daejeon, 34141, Korea.
* Corresponding author (ghlee@kisti.re.kr)
[Received 14 May 2024, Reviewed 18 May 2024(R2 13 August 2024), Accepted 26 September 2024]
☆ This research has been performed as a project of Project No. K24L2M1C1 (The national flagship supercomputer infrastructure implementation and service) supported by the Korea Institute of Science and Technology Information (KISTI).
☆ A preliminary version of this paper was presented at ICONI 2023.

development has driven the evolution of computational environments [1,2]. Container technologies have emerged as a pivotal innovation, offering flexibility, portability, and efficiency in managing complex applications across diverse computing platforms. These technologies have significantly impacted HPC environments by enabling the encapsulation of applications and their dependencies into lightweight, portable units that can be executed consistently across different systems.

Containers provide a robust solution to the challenges posed by traditional HPC infrastructures, such as dependency management, resource allocation, and scalability. By isolating applications within containers, users can achieve reproducibility of computational experiments, streamline deployment processes, and optimize the utilization of HPC resources. As a result, containerization has become a key enabler for advancing HPC capabilities, particularly in areas requiring rapid iteration and deployment, such as AI and machine learning.

Various organizations, including the Jülich Supercomputing Center (JSC), Swiss National Supercomputing Center (CSCS), and National Energy Research Scientific Computing Center (NERSC), provide Jupyter services on supercomputers, and Google offers the Colab service [3] to its users. The foundational infrastructure configurations for these services are container-based [4]. However, these configurations pose the inconvenience of requiring users to install and configure essential packages and frameworks each time they access and execute the service using an initialized container image [5].

This paper introduces a novel approach to managing and building customized container images tailored for HPC environments. Our solution leverages (1) containerization to enhance the flexibility and efficiency of HPC workflows, (2) providing users with the ability to create, manage, and deploy personalized computational environments with ease. By integrating a sophisticated (3) image manager and (4) builder within a container-based HPC infrastructure, we aim to address the specific needs of HPC users, offering a scalable and user-friendly platform that optimizes resource utilization and streamlines the deployment of AI service platforms such as Jupyter and RStudio.

In the following sections, we explore the existing research on containerization in HPC environments, comparing various approaches based on key aspects such as HPC integration, security, resource overhead, customization, ease of use, and user privileges in section 2. We then present the architecture of our proposed solution in section 3, detailing the components and mechanisms that enable the efficient management and building of customized container images in an HPC context in section 4. Finally, we discuss the implementation of our system, highlighting its effectiveness and scalability in real-world HPC environments (a web-based portal called MyKSC) in section 5, and conclude with potential future enhancements to further improve its capabilities in the last section. The whole approach has been applied to KISTI-5 supercomputer, known as the CPU-only system Nurion, and the GPU system Neuron.

2. Related Work

The advent of container technologies has profoundly transformed the HPC landscape by offering efficient, portable, and scalable solutions for managing applications and resources. This section reviews existing research in HPC containerization, with a particular focus on customized container image management and building. Various approaches are compared based on their key features and capabilities. Table 1 presents a comparative analysis of different studies on containerization in HPC environments. The comparison is based on several critical aspects: HPC integration, security, resource overhead, customization, and ease of use, rated as High (H), Moderate (M), or Low (L), based on subjective evaluation. Additionally, root privilege requirements are indicated as either Yes or No.

① HPC Integration: This refers to the degree to which a solution is architected to operate efficiently within HPC environments. Effective HPC integration involves the optimization of computational resources and infrastructure, ensuring that the solution leverages the full capabilities of the HPC system.

② Security:Security encompasses the strategies and protocols implemented to safeguard data and applications within a solution. In the context of HPC, this is

particularly crucial for maintaining secure and reliable operations in environments with multiple users, where data integrity and access control are paramount.

③ Resource Overhead:Resource overhead refers to the additional computational or memory resources that a solution consumes. This factor is critical as it affects the overall efficiency and performance of the HPC system. Lower resource overhead typically leads to better utilization of the system's capabilities.

④ Customization: Customization reflects the extent to which a solution can be configured or tailored to meet specific user requirements. High levels of customization allow users to adapt the solution to a wide range of application needs, enhancing its applicability and usefulness in various scenarios.

⑤ Ease of Use:Ease of use is a measure of how accessible and user-friendly a solution is. This includes considerations such as the simplicity of its interface, the quality and availability of documentation, and the overall learning curve required for users to effectively operate the solution. Solutions that are easy to use are more likely to be widely adopted and successfully implemented.

⑥ User Privileges: This aspect addresses the access rights necessary to utilize or manage a solution. Specifically, it examines whether root (administrator) access is required, which has significant implications for both the security and flexibility of the solution. Solutions that do not require elevated privileges are often more secure and easier to deploy in diverse environments.

(Table 1) Feature comparison of related work

	①	②	③	④	⑤	⑥
[1]	H	H	L	H	M	No
[2]	H	H	L	H	H	No
[3]	M	M	M	H	H	Yes
[4]	H	H	L	M	M	No
[5-10]	H	H	L	H	M	No
[11-14]	M	M	M	H	H	Yes
[15-18]	H	H	L	H	M	No
Ours	H	M	H	H	H	No

Ferreira et al. [1], **Suarez et al.** [2], and several other works focus on the high integration of HPC environments with container technologies, ensuring that these solutions are secure, do not require root privileges, and have low resource overhead. These studies emphasize the need for extensive customization and high performance, which are crucial for demanding HPC applications like fusion research and deep learning.

Younge et al. [4] and **Hursey** [5] provide insights into container deployment on supercomputers and clouds, highlighting the performance benefits and security considerations of containerized HPC applications. These works underline the importance of user privileges and resource efficiency. **Singh, Tiwari** and **Dhar** [6], **Brown** and **Johnson** [7], and [9] focus on enhancing the performance of machine learning workloads and deep learning applications on HPC systems. They emphasize the necessity for optimized container images and high-performance execution.

Ali et al. [3], **Zhang** and **Lomeo** [11], and **Pandey** and **Diwakar** [12] explore cloud-based solutions, which, while powerful, often require root privileges and may introduce moderate resource overhead. These solutions are highly customizable and user-friendly, making them suitable for rapid deployment and real-time processing tasks. **Ben-Nun** [10] and **Wu et al.**[15] discuss optimizing containerized workflows and managing massive datasets, respectively. These studies highlight the integration of container technologies in HPC for improved performance and resource management.

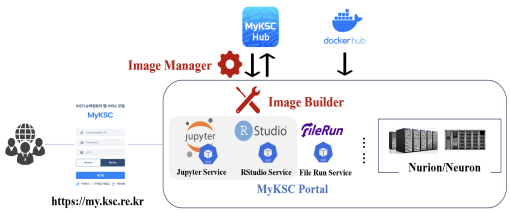
Diaz et al. [16] and **Takizawa et al.** [17] provide early and recent perspectives on cloud and HPC integration using containerization. They focus on the secure and scalable deployment of applications, ensuring high performance and ease of use. **Zhiravetska et al.** [18] illustrate the application of HPC containerization in educational contexts, showcasing how containerized environments can enhance learning and research.

The comparative analysis reveals that while several solutions and research works have made significant strides in integrating container technologies with HPC, they vary in their approach to security, customization, and ease of use. The proposed "HPC Cluster-based Customized Container Image Manager and Builder" aims to leverage the best practices identified in these studies to offer a comprehensive,

efficient, and secure solution tailored for HPC environments.

3. Architecture

The service architecture depicted in Figure 1 illustrates the framework for delivering tailored AI service platforms, namely Jupyter and RStudio, via MyKSC, leveraging both an image manager and image builder. Our objective is to establish a cohesive architecture and mechanism within a container-based HPC environment.



(Figure 1) The overall service architecture

Our architecture operates within a container-based HPC environment, ensuring flexibility, scalability, and reproducibility of computational tasks. By encapsulating each service within a container, we facilitate seamless deployment and management of AI platforms while maintaining consistent runtime environments.

The image manager serves as a crucial component, orchestrating communication between the image builder and the MyKSC Hub. It functions as a centralized control point, responsible for coordinating the creation, storage, and distribution of container images. Leveraging Docker Private Registry as the underlying infrastructure, the image manager streamlines the management of containerized services.

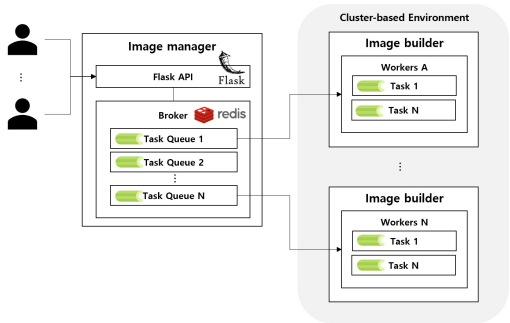
At the heart of our architecture lies the MyKSC Hub, a comprehensive platform designed to host and administer AI services. Built upon the foundation of Docker Private Registry, the MyKSC Hub serves as a secure and efficient repository for storing container images. It provides users with seamless access to Jupyter and RStudio environments, enabling them to leverage advanced AI capabilities for their research and development tasks.

Our architecture embodies an integrated mechanism that harmonizes the functionalities of various components to deliver a cohesive AI service platform. The orchestrated interplay between the image manager, image builder, and MyKSC Hub ensures smooth operation and optimal performance of the customized AI environments. Users can effortlessly deploy, manage, and interact with Jupyter and RStudio instances, thereby enhancing their productivity and computational workflows.

By embracing containerization within a container-based HPC environment, coupled with a sophisticated architecture comprising the image manager and MyKSC Hub, we establish a robust foundation for delivering customized AI service platforms. This integrated approach not only simplifies the deployment process but also enhances the overall user experience, empowering researchers and practitioners to harness the full potential of AI technologies for their diverse endeavors.

3.1 Image Manager

Figure 2 illustrates the pivotal role of the image manager in bolstering processing efficiency within the system. By integrating a task queue mechanism, the image manager optimizes the handling of image-saving requests, enabling concurrent processing from multiple users in a distributed environment across the HPC cluster-based system.



(Figure 2) Image manager and builder architecture

The task queue mechanism embedded within the image manager facilitates the efficient processing of image-saving requests. Through distributed and parallel processing techniques, tasks are systematically queued and executed, ensuring optimal resource utilization and minimizing latency.

This approach enhances system throughput and responsiveness, thereby accommodating the demands of concurrent users seamlessly.

At the core of the image manager lies a RESTful API, developed using Flask, a lightweight web framework for Python. This API serves as the interface for processing requests originating from the web portal, providing a standardized and accessible means for users to interact with the system. By adhering to REST principles, the API promotes modularity, scalability, and interoperability, facilitating seamless integration with other system components.

To delegate tasks to the image builder residing on compute nodes, the image manager leverages a Redis message broker. Redis, renowned for its high-performance and low-latency characteristics, serves as the intermediary for transmitting task-related messages between the image manager and compute nodes. This decoupled architecture enhances system reliability and scalability, enabling efficient task distribution and management across the distributed environment.

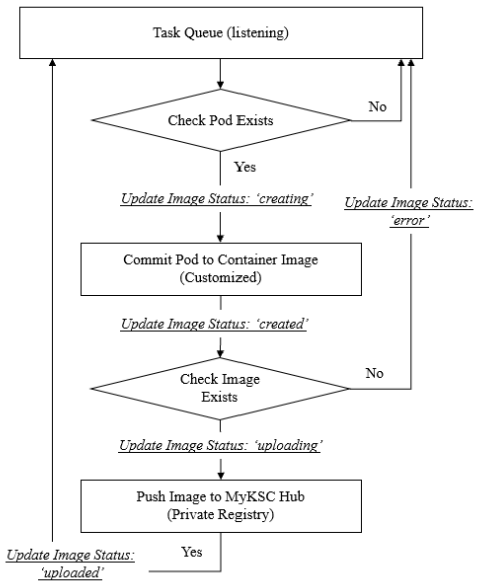
Furthermore, the image manager incorporates functionality for monitoring and recording the status of image processing tasks in a dedicated database. By maintaining a comprehensive record of task execution, including status updates and completion notifications, the system ensures transparency and accountability throughout the image-saving process. Users can track the progress of their requests in real-time, fostering trust and confidence in the system's performance and reliability.

Through the seamless integration of task queueing mechanisms, RESTful API services, Redis message brokering, and database-driven status tracking, the image manager serves as a cornerstone for enhancing processing efficiency and user experience within the system. By orchestrating distributed and parallel processing of image-saving requests, the image manager empowers the system to handle concurrent user demands effectively, thereby optimizing resource utilization and system responsiveness.

3.2 Image Builder

The image builder plays a pivotal role in the system's architecture, executing user requests received from the image manager with the aid of the Celery framework. This component is responsible for a series of intricate tasks aimed at transforming a running container into a reusable container image, facilitating seamless deployment and sharing across the system.

Utilizing the Celery distributed task queue framework, the image builder efficiently processes user requests in an asynchronous and distributed manner. Celery orchestrates task execution across compute nodes, enabling parallel processing and load balancing to maximize system throughput and responsiveness. This distributed architecture enhances scalability and fault tolerance, ensuring robust performance under varying workload conditions.



(Figure 3) Image builder flowchart

As shown in Figure 3, the primary task of the image builder revolves around the transformation of a running container into a container image. This process involves capturing the current state of the container, including its file system, configuration, and dependencies, and encapsulating it into a portable image format. Once the image creation process is complete, the image builder orchestrates the transfer of the newly created image to the designated

container registry, namely the MyKSC Hub. This private registry serves as a centralized repository for storing and managing container images, providing secure and efficient access to users across the system.

In addition to image creation and transfer, the image builder is tasked with maintaining comprehensive records of image processing tasks in the system's database. By logging essential metadata, such as task status, completion timestamps, and associated user information, the image builder ensures transparency and accountability throughout the image-building process. This record-keeping functionality enables users to track the progress of their requests and retrieve relevant information regarding completed tasks, fostering trust and confidence in the system's operations.

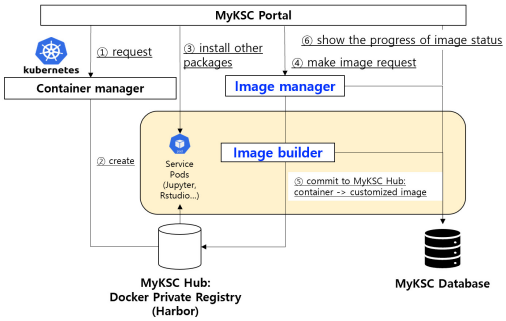
Installed on each compute node within the HPC cluster, the image builder is meticulously configured to handle multiple tasks concurrently, leveraging the available computational resources efficiently. Furthermore, the image builder's architecture is designed to adapt dynamically to fluctuations in resource utilization, allowing for seamless scalability and resource allocation based on workload demands. This dynamic expansion capability ensures optimal utilization of compute resources while maintaining responsiveness and performance across the system.

The image builder serves as a critical component in the system's architecture, orchestrating the transformation of running containers into reusable container images with precision and efficiency. Through its integration with the Celery framework, coupled with robust task execution capabilities, the image builder empowers the system to handle diverse user requests seamlessly. By facilitating image creation, transfer, and database record maintenance, the image builder enhances system reliability, scalability, and user experience, laying the foundation for efficient containerized workflows within the HPC environment.

4. Implementation

4.1 Integration Components

Within the AI services, every application runs as a Kubernetes-based service pod. We created a designed image manager and builder for RStudio and Jupyter pods. The six components of the workflow were used to construct the integration architecture using Kubernetes, as shown in Figure 4. The integration workflow outlined above illustrates the orchestrated interaction between users, Kubernetes container management, and specialized components for image management and customization. By leveraging Kubernetes' capabilities alongside custom-built image management components, users benefit from a seamless and flexible environment for developing and deploying AI applications within the Kubernetes cluster.



(Figure 4) Integration components workflow

① Users interact with the container manager, which in this case is Kubernetes, to submit requests for creating service applications, such as Jupyter and RStudio. Kubernetes is a container orchestration platform that automates the deployment, scaling, and management of containerized applications.

② The container manager utilizes pre-configured YAML scripts to deploy pods for the requested service applications. A pod is the smallest deployable unit in Kubernetes and comprises one or more containers that share resources and network space.

③ Once the pods for Jupyter or RStudio are deployed, users can access them and install the necessary AI model packages or libraries directly into the service pods (containers). This allows users to customize their development environment according to their specific requirements.

④ After configuring the development environment within the service pods, users can utilize the MyKSC portal to submit a request to the image manager to save the current

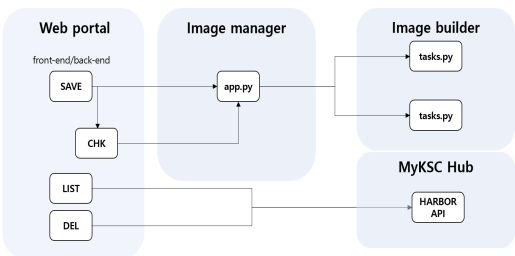
environment. This request is sent via the portal in multiple formats, including the JSON data format. The image manager processes these requests and sends them to the image builder component that runs on a specific node within the Kubernetes cluster.

⑤ Upon receiving commands from the image manager, the image builder uses the NERDCTL tool to commit the running container to a customized image. NERDCTL is a command-line interface that manages container images and containers. Once customization is complete, the customized image is uploaded to the MyKSC Hub, which serves as a repository for storing and managing container images within the Kubernetes cluster.

⑥ Users can then access and manage their customized images through the MyKSC portal. This portal provides a user-friendly interface for viewing and managing container images, allowing users to easily track and utilize their customized development environments as required.

4.2 Functional Components

The backend of the web portal was integrated with four functions (shown in Figure 5). The SAVE function makes it easier to send user data and the required parameter values in JSON data format to the image manager. The state of the image processing was continuously observed using the CHK function. Depending on the resource type, the images were divided into GPU and CPU categories. App.py contains the code for the SAVE and CHK functions, which use Flask, Celery, and Redis. Tasks.py defines the tasks for every builder in app.py. The Harbor API is used by the LIST and DEL functions to list user images and make it possible to delete each user image.



(Figure 5) Functional components workflow

The SAVE function simplifies the process of transmitting user data and required parameter values to the image manager by encapsulating them in JSON data format. This function enhances user experience by providing a streamlined interface for submitting requests to the image manager component. Users can effortlessly specify their customization preferences and submit them to initiate the image processing workflow.

The CHK function continuously monitors the state of image processing, providing real-time feedback to users regarding the progress and status of their requests. By actively observing the image processing workflow, users can stay informed about the completion status and any potential errors or issues encountered during the process. This proactive monitoring capability enhances transparency and user engagement, ensuring a seamless experience throughout the image customization process.

Depending on the resource requirements of users, the images generated by the system are categorized into GPU and CPU categories. This classification enables users to select and deploy customized images tailored to their specific computational needs. By offering distinct categories based on resource types, the system ensures optimal utilization of available computing resources and accommodates diverse user preferences and requirements

The backend functionalities, including the SAVE and CHK functions, are implemented within the app.py file using Flask, Celery, and Redis. Flask serves as the web framework for building the web portal's backend, providing tools and utilities for handling HTTP requests and responses. Celery, a distributed task queue framework, facilitates asynchronous task execution, enabling efficient processing of image customization requests. Redis, acting as the message broker, facilitates communication between Flask and Celery, ensuring reliable task dispatch and execution.

Additionally, the tasks.py file defines the tasks associated with each builder component, complementing the functionalities implemented in app.py. This modular approach enhances code organization and maintainability, facilitating scalability and extensibility of the backend system.

The LIST and DEL functions leverage the Harbor API to list user images and enable the deletion of individual user

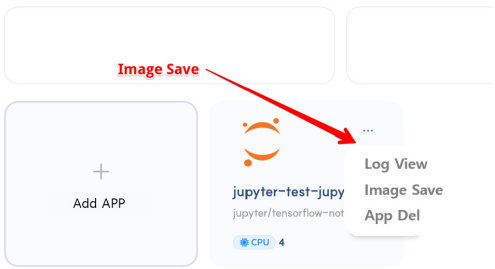
images, respectively. Harbor, a cloud-native registry for storing, signing, and scanning container images, provides a robust API for interacting with container images stored within the repository. By integrating with the Harbor API, the web portal enables users to manage their customized images efficiently, including listing available images and performing deletion operations as needed.

The integration of backend functions within the web portal enhances user interaction and facilitates efficient management of image customization tasks. By leveraging Flask, Celery, Redis, and the Harbor API, the backend system offers a seamless experience for submitting, monitoring, and managing image processing requests. This integrated approach ensures reliability, scalability, and flexibility, empowering users to customize and deploy AI environments tailored to their specific needs and preferences.

4.3 Experimental Result

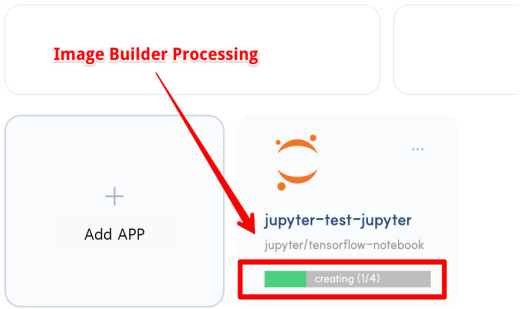
To enhance user convenience and streamline the image creation process, we have integrated several user-friendly features into our web portal:

One-Button Image Creation: We have introduced a one-button function on our web portal designed to simplify the process of creating customized images. This feature allows users to generate images of their current environment with just a single click. By minimizing the number of steps required, we aim to make image creation more accessible and efficient for all users. This functionality is illustrated in Figure 6.



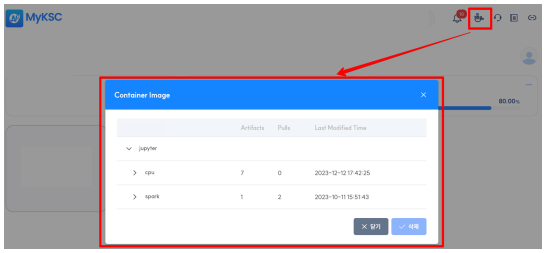
(Figure 6) User interface of image save button

Real-Time Progress Monitoring: As users initiate the image creation process, they can easily track its progress through a visual processing bar displayed on the web interface. This progress bar provides real-time updates, showing how far along the image creation process is and allowing users to stay informed about the status of their request. This feature ensures users are aware of the ongoing process and can anticipate when their image will be ready. The processing bar is depicted in Figure 7.



(Figure 7) User interface of image process

Image Management Interface: Once the image creation is complete, users can access a comprehensive management interface on the web portal to view and handle their created images. This interface presents a list of all images associated with the user account, allowing users to easily organize, review, and manage their images. The management functionality is designed to be straight forward and intuitive, providing users with control over their image assets. This aspect of the portal is shown in Figure 8.



(Figure 8) User interface of customized image list

5. Conclusion

In this paper, we introduced a novel approach for managing and creating customized container images within

HPC environments. Our solution integrates a sophisticated image manager and builder into a container-based HPC infrastructure, addressing key needs for flexibility, scalability, and efficiency in computational workflows. The implemented system facilitates the effortless creation, management, and deployment of personalized AI service platforms, optimizing the user experience by minimizing the manual configuration of essential packages and frameworks. Validation on real-world HPC systems, including the Nurion supercomputer and the Neuron GPU system, demonstrates the solution's effectiveness, scalability, and seamless integration with existing frameworks. Meanwhile, our experimental result has following limitations:

Scalability Constraints: Although our system performs well under current conditions, its scalability may be limited by the capacity of the underlying compute nodes and network infrastructure. As user demand grows, further enhancements may be required to maintain performance and responsiveness.

Resource Overhead: While the system optimizes resource utilization, the process of creating and managing container images still incurs some level of resource overhead. This overhead could impact performance in highly resource-constrained environments.

According to these limitations, future work will focus on enhancing the system's scalability by optimizing task distribution and resource allocation. This may involve incorporating advanced load-balancing techniques and exploring more scalable infrastructure solutions. Research into reducing the resource overhead associated with container image management is planned. This could include optimizing image creation processes and refining resource allocation strategies to improve overall system efficiency.

In summary, while our proposed system significantly improves the management and creation of customized container images in HPC environments, there are areas for further development. Addressing these limitations and pursuing the outlined expansions will enhance the system's robustness, efficiency, and user satisfaction, contributing to the advancement of HPC capabilities and applications.

Reference

[1] Diogo R. Ferreira, and JET Contributors, "Using HPC Infrastructures for Deep Learning Applications in Fusion Research," Plasma Physics and Controlled Fusion, Vol 63., No. 8, 2021. https://doi.org/10.1088/1361-6587/ac0a3b

[2] Estela Suarez, Norbert Eicker, Thomas Moschny, Simon Pickartz, Carsten Clauss, Valentin Plugaru, Andreas Herten, Kristel Michielsen, Thomas Lippert, "Modular Supercomputing Architecture," White Paper of a Success Story of European R&D, 2022. https://www.etp4hpc.eu/pujades/files/ETP4HPC_WP_MSA_20220519.pdf

[3] I. Ali, A. Khan and M. Waleed, "A Google Colab Based Online Platform for Rapid Estimation of Real Blur in Single-Image Blind Deblurring," 2020 12th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Bucharest, Romania, pp. 1-6, 2020. https://doi.org/10.1109/ECAI50035.2020.9223244.

[4] A. J. Younge, K. Pedretti, R. E. Grant and R. Brightwell, "A Tale of Two Systems: Using Containers to Deploy HPC Applications on Supercomputers and Clouds," 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Hong Kong, China, pp. 74-81, 2017. https://doi.org/10.1109/CloudCom.2017.40.

[5] J. Hursey, "Design Considerations for Building and Running Containerized MPI Applications," 2020 2nd International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC), Atlanta, GA, USA, pp. 35-44, 2020. https://doi.org/10.1109/CANOPIEHPC51917.2020.00010.

[6] S. T. Singh, M. Tiwari and A. S. Dhar, "Machine Learning based Workload Prediction for Auto-scaling Cloud Applications," 2022 OPJU International Technology Conference on Emerging Technologies for Sustainable Development (OTCON), Raigarh, Chhattisgarh, India, pp. 1-6, 2023.

https://doi.org/10.1109/OTCON56053.2023.10114033.

[7] S. Brown, O. Johnson and A. Tassi, “Reliability of Broadcast Communications Under Sparse Random Linear Network Coding,” in IEEE Transactions on Vehicular Technology, vol. 67, no. 5, pp. 4677-4682, May 2018, https://doi.org/10.1109/TVT.2018.2790436.

[8] M. Riedel et al., “Practice and Experience in using Parallel and Scalable Machine Learning with Heterogenous Modular Supercomputing Architectures,” 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Portland, OR, USA, pp. 76-85, 2021. https://doi.org/10.1109/IPDPSW52791.2021.00019.

[9] F. Torres-Cruz et al., “Comparative Analysis of High-Performance Computing Systems and Machine Learning in Enhancing Cyber Infrastructure: A Multiple Regression Analysis Approach,” 2022 2nd International Conference on Innovative Practices in Technology and Management (ICIPTM), Gautam Buddha Nagar, India, pp. 69-73, 2022. https://doi.org/10.1109/ICIPTM54933.2022.9753839.

[10] T. Ben-Nun, T. Gamblin, D. S. Hollman, H. Krishnan and C. J. Newburn, “Workflows are the New Applications: Challenges in Performance, Portability, and Productivity,” 2020 IEEE/ACM International Workshop on Performance, PortabiliMty and Productivity in HPC (P3HPC), GA, USA, pp. 57-69, 2020. https://doi.org/10.1109/P3HPC51967.2020.00011.

[11] S. Zhang, J. Lomeo, “Cloud-based Image Management Solutions for Digital Transformation of Drug Product Development,” Microscopy and Microanalysis, Vol. 27, No. S1, pp. 296-297, 2021. https://doi.org/10.1017/S143192762100163X

[12] N. K. Pandey and M. Diwakar, “A Review on Cloud based Image Processing Services,” 2020 7th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, pp. 108-112, 2020. https://doi.org/10.23919/INDIACom49435.2020.9083718.

[13] P. Kanjanamek, N. Chaiyabud, N. Kitikhungumjon and S. Fugkeaw, “An Adaptive Cloud-Based Image Steganography System with Fast Stego Retrieval,” 2024 16th International Conference on Knowledge and Smart Technology (KST), Krabi, Thailand, pp. 29-34, 2024. https://doi.org/10.1109/KST61284.2024.10499672.

[14] A. Abdelmageed et al., “Cloud-Based AI-Enhanced Dual-Mode System For Automatic Coronary Artery Calcification Detection and Quantification,” 2024 41st National Radio Science Conference (NRSC), New Damietta, Egypt, pp. 270-277, 2024. https://doi.org/10.1109/NRSC61581.2024.10510468.

[15] Z. Wu, P. Ma, X. Zhang and G. Ye, “Efficient Management and Processing of Massive InSAR Images Using an HPC-Based Cloud Platform,” in IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 17, pp. 2866-2876, 2024. https://doi.org/10.1109/JSTARS.2023.3349214.

[16] J. Diaz, G. von Laszewski, F. Wang and G. Fox, “Abstract Image Management and Universal Image Registration for Cloud and HPC Infrastructures,” 2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, pp. 463-470, 2012. https://doi.org/10.1109/CLOUD.2012.94.

[17] S. Takizawa, M. Shimizu, H. Nakada, H. Matsuba and R. Takano, “CloudQ: A Secure AI / HPC Cloud Bursting System,” 2022 IEEE/ACM International Workshop on HPC User Support Tools (HUST), Dallas, TX, USA, pp. 48-50, 2022. https://doi.org/10.1109/HUST56722.2022.00012

[18] A. Zhiravetska, J. Chaiko, N. Kunicina and J. Maksimkina, “Study Courses Digitalisation at RTU On the Basis of HPC Platform and Combined Learning Methodology,” 2023 IEEE 64th International Scientific Conference on Power and Electrical Engineering of Riga Technical University (RTUCON), Riga, Latvia, pp. 1-6, 2023. https://doi.org/10.1109/RTUCON60080.2023.10412966.

◐ 저 자 소 개 ◑

이 국 화 (Gukhua Lee)
2011년 연변과학기술대학교 컴퓨터공학과(공학사)
2013년 건국대학교 일반대학원 신기술융합학과(공학석사)
2018년 건국대학교 일반대학원 신기술융합학과(공학박사)
2018년~2021년 한국과학기술정보연구원 슈퍼컴퓨팅인프라센터 박사후 연구원
2022년~현재 한국과학기술정보연구원 슈퍼컴퓨팅인프라센터 선임기술원
관심분야 : 슈퍼컴퓨팅, 고성능컴퓨팅, 클라우드 컴퓨팅, 데이터 사이언스, 데이터 분석, etc.
E-mail : ghlee@kisti.re.kr

우 준 (Joon Woo)
1998년 한남대학교 컴퓨터공학과(공학사)
2000년 한남대학교 대학원 컴퓨터공학과(공학석사)
2018년 충남대학교 대학원 컴퓨터공학과(공학박사)
2000년~현재 한국과학기술정보연구원 슈퍼컴퓨팅인프라센터 책임연구원
관심분야 : 슈퍼컴퓨팅, 고성능컴퓨팅, 서버 가상화, etc.
E-mail : wjnadia@kisti.re.kr

홍 태 영 (Taeyoung Hong)
1999년 성균관대학교 물리학과(학사)
2002년 성균관대학교 일반대학원 물리학과(이학석사)
2003년~현재 한국과학기술정보연구원 슈퍼컴퓨팅인프라센터 책임연구원
2019년~현재 한국과학기술정보연구원 슈퍼컴퓨팅인프라센터 센터장
관심분야 : 슈퍼컴퓨팅, 고성능컴퓨팅, 컴퓨터 아키텍처, 차세대 고성능 컴퓨터, etc.
E-mail : tyhong@kisti.re.kr