2020 2nd International Workshop on Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC)

Enabling Seamless Execution of Computational and Data Science Workflows on HPC and Cloud with the Popper Container-native Automation Engine

Jayjeet Chakraborty
UC Santa Cruz
orcid.org/0000-0003-1647-2494

Carlos Maltzahn
UC Santa Cruz
orcid.org/0000-0001-8305-0748

Ivo Jimenez
UC Santa Cruz
orcid.org/0000-0002-2222-1985

***Abstract*—The problem of reproducibility and replication in scientific research is quite prevalent to date. Researchers working in fields of computational science often find it difficult to reproduce experiments from artifacts like code, data, diagrams, and results which are left behind by the previous researchers. The code developed on one machine often fails to run on other machines due to differences in hardware architecture, OS, software dependencies, among others. This is accompanied by the difficulty in understanding how artifacts are organized, as well as in using them in the correct order. Software containers (also known as Linux containers) can be used to address some of these problems, and thus researchers and developers have built scientific workflow engines that execute the steps of a workflow in separate containers. Existing container-native workflow engines assume the availability of infrastructure deployed in the cloud or HPC centers. In this paper, we present Popper, a container-native workflow engine that does not assume the presence of a Kubernetes cluster or any service other than a container engine such as Docker or Podman. We introduce the design and architecture of Popper and describe how it abstracts away the complexity of multiple container engines and resource managers, enabling users to focus only on writing workflow logic. With Popper, researchers can build and validate workflows easily in almost any environment of their choice including local machines, Slurm based HPC clusters, CI services, or Kubernetes based cloud computing environments. To exemplify the suitability of this workflow engine, we present a case study where we take an example from machine learning and seamlessly execute it in multiple environments by implementing a Popper workflow for it.**

I. Introduction

Researchers working in various domains related to computational and data-intensive science upload experimental artifacts like code, figures, datasets, and configuration files, to open-access repositories like Zenodo [1], Figshare [2], or GitHub [3]. According to [4], approximately 1% of the artifacts available online are fully reproducible and 0.6% of them are partially reproducible. A 2016 study by Nature found that from a group of 1576 scientists, around 70% of them failed to reproduce each other's experiments [5]. This problem occurs mostly due to the lack of proper documentation, missing artifacts, or encountering broken software dependencies. This results in other researchers wasting time trying to figure out how to reproduce those experiments from the archived artifacts, ultimately making this process inefficient, cumbersome, and error-prone [6].

Numerous existing research has tried to address the problem of reproducibility [7] by different means; for example, logging and tracing system calls, using workflow engines, using correctly provisioned shared and public testbeds, by recording and replaying changes from a stable initial state, among many others [8]. These approaches have led to the development of various tools and frameworks to address these problems of reproducibility [9,10], with scientific workflow engines being a predominant one [11–13]. A workflow engine organizes the steps of a scientific experiment as the nodes of a directed acyclic graph (DAG) and executes them in the correct order [14]. Nextflow [15], Pegasus [16] and Taverna [17] are examples of widely used scientific workflow engines. But some phenomena like unavailability of third-party services, missing example input data, changes in the execution environment, insufficient documentation of workflows make it difficult for scientists to reuse workflows, resulting in *workflow decay* [18].

One of the main reasons behind *workflow decay* is the difficulty in reproducing the environment where a workflow is developed and originally executed [19]. Virtual machines (VM's) can be used to address this problem, as its isolation guarantees make it suitable for running steps or the entirety of a workflow inside a separate VM [20,21]. A VM is typically associated with large resource utilization (e.g. long start times and high memory usage), making OS-level virtualization technologies a better-suited tool for reproducing computational environments with fewer overheads [22,23]. Although software (Linux) containers are a relatively old technology [24], it was not until recently, with the rise of Docker, that they entered mainstream territory [25].

Docker has been a popular container runtime for a long time, with other container runtimes such as Singularity [26], Rkt [27], Charliecloud [28], and Podman [29] having emerged. With containers, the container-native software development paradigm emerged, which promotes the building, testing, and deployment of software in containers, simultaneously giving rise to the practice of running scientific experiments inside containers to make them platform independent and reproducible