EOSC Technical Specification

Hub Federated Core Services

HTC/HPC Compute - Multitenant Containerised job submission

Introduction

Most scientific challenges require running computationally demanding tasks. Typically, these computing challenges can be tackled by gathering several computing resources that concurrently run the tasks. In some cases, the computational problem can be addressed by multiple and loosely coupled tasks that can run over different data blocks or different parameter sets, and in some cases, the problem requires gathering several computing elements together to solve every single task in a closely coupled parallelism. The former is addressed through High-Throughput Computing (HTC) models and the latter by the High-Performance Computing (HPC) models. In the HTC/HPC Compute TCOM we address services for running a large set of independent tasks and to jointly use several computing resources to run a parallel job.

In the last years, the use of containerised jobs has boosted due to the enormous convenience of containers for application delivery. Application dependencies are embedded into the containers reducing the effort and side-effects of the installation of software. However, popular container technologies such as Docker use daemon processes that run on privileged users, which is not acceptable by many datacentre policies. There are solutions for running jobs on containers that run on the userspace. This approach reduces the capabilities of a containerised job to those of the user running the job, which makes it suitable for HPC, HTC and Cloud Compute infrastructures.

This macro-feature is complementary to the Multitenant Job Submission[1] and it should be considered as an extension.

[1] https://docs.google.com/document/d/166AclBzyk5GrwKPPliKdWzzCu_LYjmS77R-zBuF0b4k/edit#

Adopted Standards

Standard	Short description	References
OpenStack (especially Nova API and Keystone API v3)	OpenStack is an Open Source cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed and provisioned through APIs with common authentication mechanisms.	OpenStack API
OpenID tokens & SAML	The service must leverage habitual authentication and authorization mechanisms used in Scientific Research Infrastructures to avoid users to handle multiple credentials. EGI Check-in is an example (https://wiki.egi.eu/wiki/AAI_guide_for_SPs)	https://openid.net/connect/, http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.pdf
Docker Engine API	HTTP API served by Docker Engine to manage the lifecycle of Docker containers.	https://docs.docker.com/engine/api/v1.40/
Open Container Initiative	OCI currently contains two specifications: the Runtime Specification (runtime-spec) and the Image Specification (image-spec).	https://www.opencontainers.org/
Singularity Container Interface	Python API for working with Singularity containers.	https://github.com/singularityhub/api

High-level Service Architecture

The high-level service architecture shown in this section deals with the additional part with respect to the Multitenant Job Submission. In this part, the HTC and Cloud Compute services should be unaware of the container management.

Container management may require automatic container building and container registries, although the user could provide their own containers. This way we address both final users that

have complex dependencies embedded on containers and site administrators who want to install the minimum software requirements on their platforms.

HPC platforms require running the jobs on containers that run on the userspace. Meanwhile, Cloud Compute platforms may not have such requirement, as they could safely provide privileged user access to the virtual resources without compromising the security of the system. However, using the same approach will largely reduce complexity.

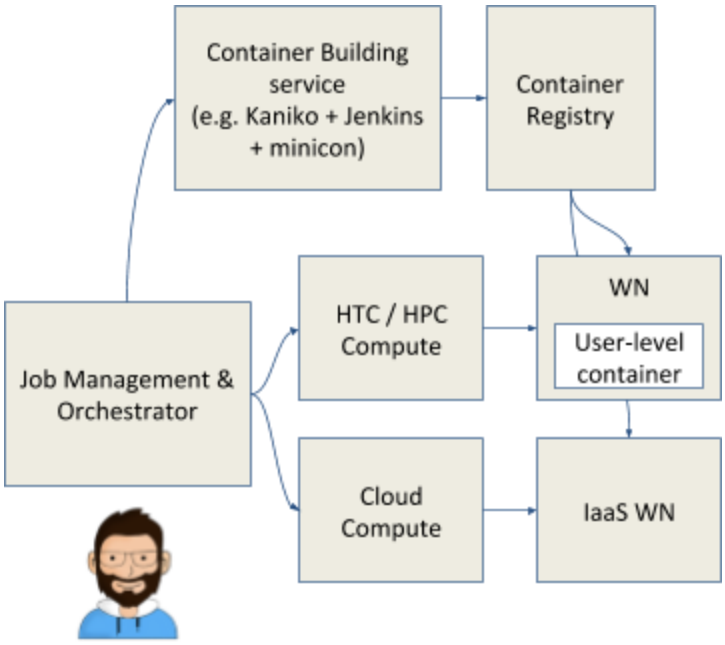

Figure 1: Potential high-level architecture for the containerised job submission Macro-feature.

The execution backends are described in the HPC/HTC Multitenant Job Submission building blocks specification[2] and the reader should refer to this document for more details. Basically, a WN is a Working Node which is part of a processing pool managed through the HTC or HPC compute services. An IaaS WN would be a VM instantiated for the processing of a containerised job.

It is important to differentiate between HTC/HPC compute resources and Cloud compute resources, as the former are typically restricted to unprivileged processes, which prevents the use of Docker containers as is. Alternatives such as uDocker (https://github.com/indigo-dc/udocker), Singularity (https://sylabs.io/docs/), CharlieCloud (https://github.com/hpc/charliecloud) can be used to overcome this limitation.

Interoperability guidelines

With respect to the user, the service should expose the following interface:

[2] https://docs.google.com/document/d/166AclBzyk5GrwKPPliKdWzzCu_LYjmS77R-zBuF0b4k/edit#

- Management of already containerised jobs. The user should be able to express along with the job description the reference of the container to be used, as well as other related information, such as forcing the container pull.
- Automatic containerisation of jobs. Site administrators may consider relevant to embed the job within a container. For this purpose, the service may create a new container instance, based on a general-purpose container instance selected by the user, with the additional data environment required for the execution.

This document extends the HPC/HTC Multitenant Job Submission building blocks specification[3] which describes the parts of the interface that are common to this use case.

We do not include here the interoperability of guidelines of the Multitenant job submission macro feature, which are already listed in the referred document. We expect that the additional services (registry and container building) will interact with the AAI interoperability.

Examples of solutions implementing this specification

This solution is partially addressed by the following technologies[4]

- User-level Containers
    - uDocker: Docker containers on the user space with compatibility to Singularity (https://github.com/indigo-dc/udocker).
    - Singularity. Singularity containers can be used to package entire scientific workflows, software and libraries, and even data without superuser privileges (https://singularity.lbl.gov/).
    - Charliecloud - user-defined software stacks for high-performance computing (HPC) centers (https://github.com/hpc/charliecloud).
- SCAR. This framework creates uDocker containers on the fly and AWS Lambda functions to run them. The creation of uDocker containers could be relevant to the service (https://github.com/grycap/scar).
- O-SCAR. This framework provides the capability of implementing Function as a Service serverless computing framework. The container management and even the FaaS programming model could be of great interest (https://github.com/grycap/oscar)

[3] https://docs.google.com/document/d/166AclBzyk5GrwKPPliKdWzzCu_LYjmS77R-zBuF0b4k/edit#
[4] See “Comparison of Container-based Virtualization Tools for HPC Platforms”, https://indico.lip.pt/event/575/contributions/1856/