Standardizing DIRAC's Cloud Interfaces

Daniela Bauer[1,*] and *Simon* Fayer[1]

[1]Blackett Laboratory, Department of Physics, Imperial College London, London SW7 2AZ, United Kingdom

Abstract.
DIRAC is a widely used framework for distributed computing. It provides a layer between users and computing resources by offering a common interface to a number of heterogeneous resource providers. In these proceedings we describe a new implementation of the DIRAC to Cloud interface.

1 Introduction

DIRAC was originally developed as a workload management system for the LHCb experiment. In 2014 the DIRAC consortium was founded to oversee the release of DIRAC as an experiment agnostic tool. It has since been adopted by a number of communities, either as a single-VO instance (e.g. Belle II, CTA, ILC and others) or in a multi-VO configuration (e.g. GridPP and EGI). In the UK the GridPP DIRAC instance provides access to resources for around 10 communities, mostly in the neutrino, dark matter and astronomy disciplines. The IRIS project in the UK is a government initiative aimed at providing computing resources to a range of scientific projects outside of the traditional WLCG context. A large amount of these resources are cloud based, meaning reliable cloud support became a priority for the GridPP DIRAC instance.

2 The DIRAC Cloud Interface

DIRAC [1] originally provided support for dynamic workload management on a cloud via its VMDIRAC extension [2]. When the VMDIRAC extension was envisaged, it was common to use commercial clouds via reservations and the design of the project reflected this. The VMDIRAC code itself was developed as a self-contained plugin, duplicating a significant amount of code from the core DIRAC project which then needed maintaining separately.

With experience, the modern usage of clouds within the DIRAC communities is primarily targeted at research rather than commercial clouds; this allows us to approach the cloud resources in a similar manner to a grid resource.

3 Implementation

The aim of the new implementation was to minimize the cloud specific code within the DIRAC framework and to use standard cloud interface libraries (e.g. Apache libcloud [3]) to avoid the pitfalls of homebrew code.

*e-mail: lcg-site-admin@imperial.ac.uk

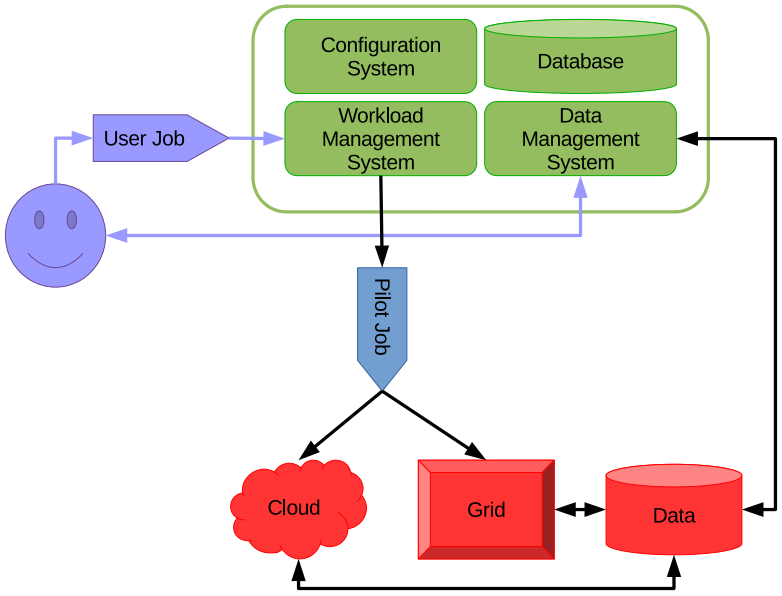


Figure 1: A schematic of the DIRAC framework. Not all possible extensions are shown.

For grid-like resources, the workload management component of DIRAC uses a pilot based submission system [4]. A schematic of the DIRAC framework is shown in Figure 1. A service determines how many pilots should be submitted and uses a resource specific plugin known as a ComputingElement (CE) to manage its pilots; these plugins implement either a local or remote interface depending on what is appropriate for their target resource. Remote implementations exist for HTCondorCE, ARC and SSH (primarily used for direct access to HPC resources). Local implementations run inside the pilot and perform tasks such as containerization (SingularityCE) and multi-core slot partitioning (PoolCE). In the new cloud interface design, a new cloud specific CE plugin can be created, significantly reducing the amount of code duplication. An overview of changes to the module structure is shown in Figure 2.

Apache libcloud is an open source collection of python based cloud interfaces, among them OpenStack and a number of commercial cloud solutions, maintained by the Apache foundation. The majority of our target cloud sites are OpenStack based. In general we prefer to use an application specific token credential for authenticating against the cloud services; in OpenStack these are known as application credentials. Apache libcloud initially had no support for application credentials but we were able to submit a patch to enable this functionality, which has been accepted into the upstream libcloud release. Adding this code to libcloud instead of DIRAC has the advantage of a much wider user base benefiting from it. In turn, DIRAC benefits from updates to libcloud without having to maintain their own cloud-specific modules.

Conceptually each virtual machine (VM) is treated as a single pilot job, which is processed by DIRAC in the same manner as pilot jobs sent to grid resources. This allows the creation of a CloudCE plugin which inherits from the ComputeElement like any other CE module in DIRAC. Instead of communication with a grid compute element, the code calls the respective libcloud interface with the correct parameters/credentials. It's only through this

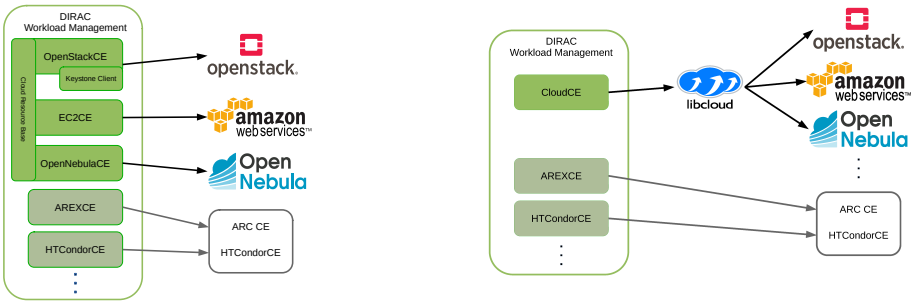
interface that DIRAC interacts with the clouds it submits to. The pilot payload script and data are added as instance metadata in cloud-init [5] format; this allows any image containing cloud-init to decode and start the DIRAC pilot bootstrap scripts. The previous VMDIRAC implementation used a custom wrapper to set-up the pilot. The new CloudCE is able to use the standard pilot wrapper directly, greatly reducing the maintenance required.

3.1 Other cloud considerations

Clouds present an additional challenge in that for grid-like jobs, DIRAC uses the proxy renewal mechanism provided by the CEs for the pilot proxies. In clouds the necessary inbound external connectivity cannot be guaranteed. Therefore DIRAC needs to create a sufficiently long-lived pilot proxy when starting the VM. User proxies are renewed by the DIRAC pilot agent as usual and do not need special consideration for a cloud service.

The lack of inbound external connectivity also has implication for the retrieval of pilot logs, which is sometimes necessary for debugging. This is being addressed by changing the pilot logging to a push model [7].

Initially we envisaged that clouds would use the private project networks, but as usage on our local OpenStack cloud ramped up, the NAT became overloaded and started dropping connections. This can be ameliorated by using the provider network directly, assigning a bridged public IP address to each instance directly. To minimize our IPv4 footprint we switched to using the PoolCE to allocate single core jobs within a larger multi-core flavor.



(a) Old VMDIRAC structure

(b) New CloudCE structure

Figure 2: Conceptual overview of the old and new DIRAC cloud interface implementation.

4 Status and Plans

The code has been available in DIRAC since July 2022 (release v7r3p21) and was deployed on the GridPP DIRAC at the same time. It has since been shown to work reliably for a large number of jobs (see Figure 3). We hope that through this standardization future work to include other cloud platforms and/or sites would only require at-most minor tweaks. However, once tokens become the default authentication method in our user communities, we anticipate shifting our focus to implementing token support.

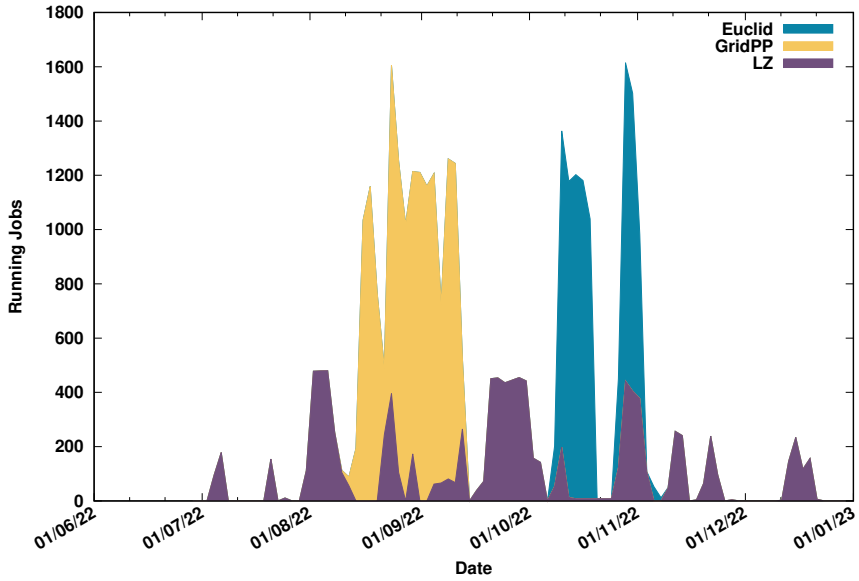


Figure 3: GridPP DIRAC Cloud usage since the deployment of the CloudCE interface.

References

[1] DIRAC consortium, *DIRAC* https://dirac.readthedocs.io/en/latest/, [accessed 2023-05-26]

[2] Víctor Fernandez Albor et al., *Cloud flexibility using DIRAC interware* (2014), J. Phys.: Conf. Ser. **513** 03203

[3] *Apache Libcloud* https://libcloud.apache.org/, [accessed 2023-05-26]

[4] F. Stagni et al., *DIRAC universal pilots* (2017), J. Phys.: Conf. Ser. **898** 092024

[5] *cloud-init - The standard for customising cloud instances* https://cloud-init.io/, [accessed 2023-11-27]

[6] *GridPP - Distributed Computing for Data-Intensive Research* https://www.gridpp.ac.uk/, [accessed 2023-05-26]

[7] J. Martyniak, S.Fayer, F. Stagni, https://indico.jlab.org/event/459/contributions/11519/, [accessed 2023-06-09]