EPJ Web of Conferences **245**, 07052 (2020) https://doi.org/10.1051/epjconf/202024507052
CHEP 2019

Running ALICE Grid Jobs in Containers

A new approach to job execution for the next generation ALICE Grid framework

Maxim Storetvedt[1*], *Latchezar* Betev[2], *Håvard* Helstrup[1], *Kristin* Fanebust Hetland[1], and *Bjarte* Kileng[1] for the ALICE Collaboration

[1]Faculty of Engineering and Science, Western Norway University of Applied Sciences, Bergen, Norway
[2]CERN, Geneva, Switzerland

Abstract. The new JAliEn (Java ALICE Environment) middleware is a Grid framework designed to satisfy the needs of the ALICE experiment for the LHC Run 3, such as providing a high-performance and high-scalability service to cope with the increased volumes of collected data. This new framework also introduces a split, two-layered job pilot, creating a new approach to how jobs are handled and executed within the Grid. Each layer runs on a separate JVM, with a separate authentication token, allowing for a finer control of permissions and improved isolation of the payload. Having these separate layers also allows for the execution of job payloads within containers. This allows for the further strengthening of isolation and creates a cohesive environment across computing sites, while avoiding the resource overhead associated with traditional virtualisation.

This contribution presents the architecture of the new split job pilot found in JAliEn, and the methods used to achieve the execution of Grid jobs while maintaining reliable communication between layers. Specifically, how this is achieved despite the possibility of a layer being run in a separate container, while retaining isolation and mitigating possible security risks. Furthermore, we discuss how the implementation remains agnostic to the choice of container platform, allowing it to run within popular platforms such as Singularity and Docker.

1 Introduction

The ALICE Collaboration [1] is moving towards a new Grid middleware framework in preparation for Run 3, aimed at better handling the expected increase in the volumes of collected data following the current detector upgrades. Named JAliEn (Java ALICE Environment) [2] for its use of Java as the language of choice, this new middleware is a complete rewrite of AliEn (ALICE Environment) [3], the current production Grid middleware. With a completely new codebase, JAliEn aims to avoid inheriting deprecated code and old "quick-fixes", while simultaneously provide a high-scalability and high-performance service for Run 3. The JAliEn middleware uses a versatile toolkit to provide its service – in addition to the new Java codebase, it also utilises more efficient database backends, comes with load balancing

*e-mail: msto@hvl.no

functionality and uses a hierarchical approach to configuration. It also comes with a new authentication scheme, based around the concept of "token certificates". The new token-based authentication scheme can be used to strengthen isolation by assigning fine-grained permissions to each specific task. To fully harness this mechanism, the job pilot has thus been split into two independent parts (i.e processes). These changes can also be used to wrap part of the job pilot in a container, so to further strengthen isolation, and introduce a cohesive execution environment across sites.

Having the job pilot operate as two separate components not only improves isolation, but also introduces changes to how jobs behave and execute on the Grid. This creates new challenges in how the components interact and communicate, challenges that are further exacerbated by potentially having a part of the pilot running in its own container. This contribution aims to address how these challenges were overcome in order to implement the new JAliEn job pilot, while highlighting key architectural aspects. Furthermore, it also addresses rising concerns in terms of vendor lock-in coming from the use of containers, and how steps were taken to overcome these concerns.

2 Authentication tokens in JAliEn

As opposed to the X.509 proxies found in AliEn, token certificates are used for authentication within JAliEn. These are full, time-limited, X.509 certificates, signed by an internal certification authority (CA). The identities of these tokens are divided into groups representing Grid roles using the distinguished name (DN) field of the certificate, with each role being given specific privileges and limitations [4].

Each task within JAliEn should only require a single token, corresponding to the appropriate Grid role. This can reduce the number of active permissions to a minimum, yet this practice is not fully upheld in all of JAliEn. The job pilot, which is modelled after the original AliEn, has two responsibilities (i.e Grid roles): job matching and job execution. For this to work within JAliEn, the pilot is required to hold a token corresponding to each role. In this case, a Job Agent Token (for job matching permissions), and a Job Token (for job execution permissions). This approach arguably defeats the purpose of having a fine-grained authentication scheme, as permissions are no longer limited to one specific task/role.

3 A two-layer job pilot

To fully harness the token-based authentication mechanism, the JAliEn job-pilot has been split into two separate components ("layers"): a "Job Agent" for job matching, and a "Job Wrapper" for job execution. Each layer represents a unique Grid role, and comes with a single token corresponding to that role – thus avoiding having a monolithic multi-role jobpilot. The two job-pilot components act as two completely independent processes, running each on its own Java Virtual Machine (JVM), with no shared resources.

3.1 Using the new job pilot on the Grid

Like the initial, monolithic, JAliEn job-pilot, the split two-layer version is an in-place replacement for the previous pilot found in AliEn. While equally started from an init script found in the batch queue, it differs by only starting the job-matching agent process using an embedded agent token within the same script (see Fig. 1). This token provides the agent with just enough permissions to match jobs.

In addition to a unique job ID and description, a matched job will return an authentication token specific to the job. Once received by the agent, this token can be used to start the

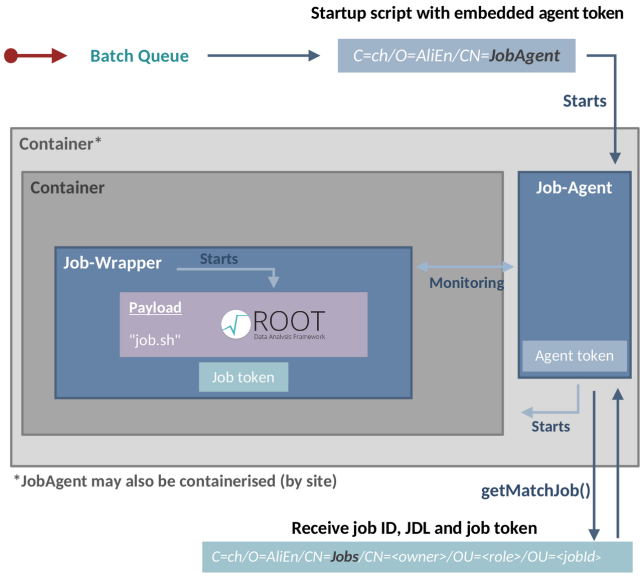

Figure 1. Overview of the split, two layer, job-pilot. A startup script is placed in the batch queue, containing an agent token. It is used to start the JobAgent, fetch an appropriate job, and a token specific to that job. The agent will thereafter start a wrapper process, provide it with necessary job data, and the received job token. The wrapper will use this token to run the required payload (job).

second pilot process, responsible for job execution. This process may be started in a separate container to strengthen isolation, provided this is supported by the host (see Sect. 3.2).

The second pilot process acts solely as a wrapper around the job payload, with the job token granting it just enough permissions to download all necessary files, and to start and run the required executable. A data pipe is maintained between the initial agent process and the wrapper during its lifecycle, providing monitoring and data transfer between the two processes. This allows the two processes to do communication and coordination, independent of being on different JVMs, or possibly separated by a container (Sect. 3.3).

3.2 Strengthening isolation through containers

While using the previous AliEn middleware, sites wanting to create more isolation between the job payload and the host node could apply virtual machines (VMs) to achieve this. More recently, this has also been achievable through the use of namespace containers. As both job matching and job execution would be handled by the job-pilot, the pilot in its entirety would in this case need to be isolated and placed in a VM/container by the site.

In JAliEn, the split job-pilot is composed of two completely independent processes, with one process functioning solely as a payload execution wrapper. As opposed to having to encapsulate the full pilot in a VM or container to separate the payload from the host, this allows for only having to isolate the process responsible for job execution. This enables the other, job-matching, component to remain outside of the encapsulation. Having a separate job-matching component outside of the encapsulation opens up a number of new possibilities in terms of how isolation can be handled and managed. Specifically, this responsibility can be delegated and integrated directly within the component – using it to start new execution

wrapper instances in a container, automatically, whenever possible[1]. Consequently, no action would longer be needed by sites wanting to provide isolation between job payloads and their executing hosts. Instead, sites may optionally still utilise a container/VM solely to also separate the pilot from the host, with payload isolation being fully self-managed by the pilot.

3.3 Communicating across layers

While a two layer job-pilot provides means to improve payload isolation within JAliEn, the very same isolation creates challenges in terms of communication between the layers. Data, such authentication tokens, must occasionally be transferred across the two independent processes. Each running on a separate JVM, and with one JVM possibly being in a container. Common solutions to the challenges above – such as transferring data through shared files or Unix sockets – require breaking down parts of the isolation between the two layers. These solutions also rely on the underlying filesystem, which in some Grid sites may be networked and in many occasions be slow and/or unreliable. Other approaches, such as using shared memory regions, risk significantly increasing the overall complexity. The chosen solution for communications within the split job-pilot, presented below, was considered the best compromise between isolation and code complexity.

In Unix-like systems, parent processes are given access to the file descriptors(FDs) of their children – visible in the filesystem under `/proc/<processID>/fd`. This includes the FDs for the processes' standard input (STDIN) and standard output (STDOUT). Within the split JAliEn job-pilot, the execution wrapper process is a child of the agent process, thus granting the agent the ability to write directly to its STDIN. Albeit this ability is commonly used to provide input strings, it can also be used to transfer serialised data. This enables the construction of a makeshift data-pipe: serialise and send data through the STDIN of the wrapper process, and conversely read from the STDOUT. By relying on STDIN/STDOUT, data transfer between the two pilot processes can be achieved in a straightforward manner, independent of them being on separate JVMs. This remains true even when the wrapper process is isolated in a container. FDs are indexed in an FD table, which are mapped to files in an open files table, and then onward to inodes stored in the inode table – containing pointers to the corresponding data block addresses. As these tables are all managed by the kernel – one of the few things that remain shared between container and host – access to the STDIN/STDOUT remains present, despite having the child (wrapper) process running in its own container. An overview of this process is depicted in Fig. 2.

4 Containers and vendor lock-in

Despite being available for a number of years, containers may arguably still be regarded as a relatively new concept. Frequent changes remain common, with new requirements and solutions being continuously introduced. By integrating containers within JAliEn, adapting to these frequent changes may become a challenge. Platform specific calls may be required in the code, and the middleware may/will be spread across numerous distributed sites in the Grid. This creates both version and vendor lock-in concerns – simply using any version of a container platform[2] in production introduces the risk of becoming bound to both that platform/vendor and its specific version.

In order to reduce the concerns coming from potentially being locked to a specific container solution, the split JAliEn job-pilot aims to be as platform agnostic as possible. In other

[1] While also technically achievable using VMs, only containers have so far been considered due to their minimal overhead and ease of system integration.

[2] Such as Docker [6] and Singularity [7].

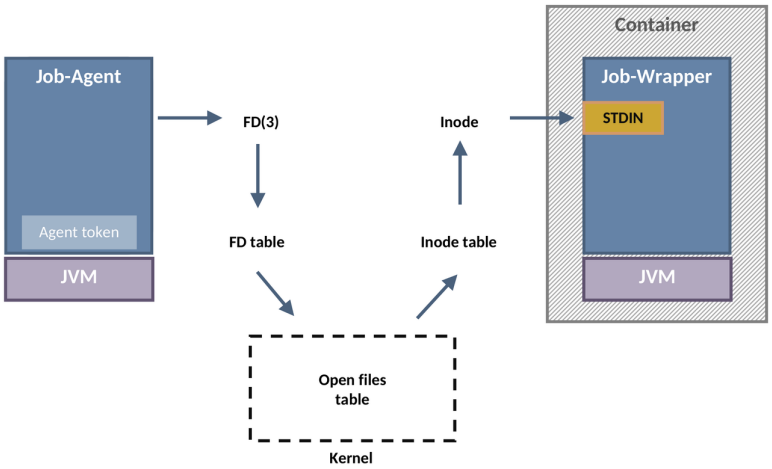

Figure 2. The agent process has access to the file descriptor (FD) for the wrapper STDIN (numbered '3', when viewed from the file-system). This can be used to serialise and send data to the wrapper process, despite it being in a container – this will lead to a lookup in the open files table, which is managed by the kernel and common for both processes.

words, to make it as straightforward as possible to swap whichever container platform is used by JAliEn, with another desired alternative.

4.1 Mitigating lock-in concerns

Keeping platform specific calls to a minimum has been the key to achieving a more platform agnostic containerised job-pilot within JAliEn. As opposed to relying on the platform to do actions such as initialising the container environment or transfer files, many of these actions can be replaced by having files and setup scripts within CVMFS – which is bind-mounted from the host within newly launched containers [5].

As an example, both Docker and Singularity – two of the more popular container platforms – have different ways of importing environment variables into their containers. Consequently, the code would need to be adjusted to account for this change when moving between the two platforms. By having these variables initialised through a common script in CVMFS, having to do such adjustments can easily be avoided. CVMFS is also used to store files, packages and container images, which are referenced through a generic symlink, set to always point towards the latest image. Some container platforms, such as Singularity, may also be run directly from CVMFS, without having to be installed on the Grid site first.

Job specific data is handled by the agent process, and piped directly to the wrapper without interacting with the underlying container platform. The remaining responsibilities may further be delegated to the wrapper, which once started, will be running inside the container

When combined, these steps allow JAliEn to have a single, generic, call to the underlying container platform, e.g "`docker run`" or "`singularity run`", which can be easily replaced if necessary.

5 Summary and outlook

To better handle the expected increase in the volumes of collected data in Run 3, as a result of undergoing detector upgrades, ALICE is preparing to use the new JAliEn Grid middleware.

To fully harness its new approach to authentication and improve isolation, a new two-layer job-pilot has been introduced – where one layer provides job matching, while the other acts as a payload execution wrapper. This allows for a more fine-grained control of permissions, in addition to having jobs automatically executed in their own separate containers. The latter is made possible by having the necessary data transferred through a makeshift pipe using the wrapper STDIN/STDOUT, which allows the two layers to communicate.

Deploying containers in production raises concerns regarding vendor lock-in. To mitigate these concerns, container specific calls have been kept as generic as possible, instead relying on CVMFS for scripts and dependencies, with the remaining responsibilities being handled by the wrapper process.

The new, containerised, split job-pilot was merged with the upstream JAliEn code in September 2019. Together with the rest of JAliEn, it is now undergoing testing in preparation for Run 3, and will gradually be deployed in production.

References

[1] The ALICE Collaboration, JINST **3**, S08002 (2008)
[2] A.G. Grigoras, C. Grigoras, M.M. Pedreira, P. Saiz, S. Schreiner, JPCS **523**, 012010 (2014)
[3] S. Bagnasco, L. Betev, P. Buncic, F. Carminati, C. Cirstoiu, C. Grigoras, A. Hayrapetyan, A. Harutyunyan, A.J. Peters, P. Saiz, JPCS **119**, 062012 (2008)
[4] M.M. Pedreira, C. Grigoras, V. Yurchenko, M. Storetvedt, EPJ-WoC **214**, 03042 (2019)
[5] J. Blomer, P. Buncic, R. Meusel, The CernVM File System, CERN Technical Report, 2013. URL: http://jblomer.web.cern.ch/jblomercvmfstech-2.1-0.pdf (Accessed 12.10.2019)
[6] Docker [Computer Software] (2013-). Retrievable at: https://docker.com
[7] Singularity [Computer Software] (2015-). Retrievable at: https://sylabs.io/