A NEW GRID WORKFLOW FOR DATA ANALYSIS WITHIN THE ALICE PROJECT USING CONTAINERS AND MODERN CLOUD TECHNOLOGIES

Doctoral Dissertation by
Maxim Storetvedt

Thesis submitted for
the degree of Philosophiae Doctor (PhD)
in
Computer Science:
Software Engineering, Sensor Networks and Engineering Computing

Department of Computer Science,
Electrical Engineering and Mathematical Sciences

Faculty of Engineering and Science

Western Norway University of Applied Sciences

November 30, 2022

Series of dissertation submitted to
the Faculty of Engineering and Science,
Western Norway University of Applied Sciences.

ISBN: 978-82-8461-042-9

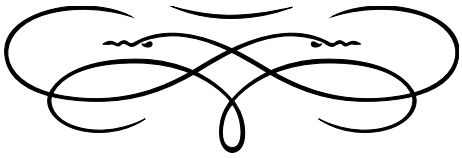
Author: Maxim Storetvedt
Title: A new Grid workflow for data analysis within the ALICE project
using containers and modern Cloud technologies

Printed production: Molvik Grafisk /
Western Norway University of Applied Sciences

Bergen, Norway, 2022

TO MY PARENTS,

for their adamant support and encouragement



PREFACE

The author of this thesis has been employed as a Ph.D. research fellow in the software engineering research group at the Department of Computer Science, Electrical Engineering and Mathematical Science at Western Norway University of Applied Sciences. The author has been enrolled into the PhD programme in Computer Science: Software Engineering, Sensor Networks and Engineering Computing, with a specialisation on software engineering.

The research presented in this thesis has been accomplished in cooperation with the ALICE Collaboration at CERN, Meyrin, Switzerland, through a Project Associate position.

This thesis is a monograph, organised in three parts. Part I is an overview of the research field, related technologies and current state of the art. Part II describes the contributions provided as part of the research work. Part III examines the use of the contributions, in turn providing a discussion in light of the underlying Research Question and broader scope.

ACKNOWLEDGMENTS

It must be recognised that this doctoral work could not have happened without the guidance, aid and influences of many people – and I would like to set aside a few words to express my ardent gratitude. First, to my supervisors at the Western Norway University of Applied Sciences, Bjarte Kileng, Håvard Helstup and Kristin Fanebust Hetland, who not only introduced me to the field many years ago, but also kindly and graciously provided me with countless help and guidance throughout them.

I would first like to thank Bjarte for taking me in as a Master's student, showing me the ropes and teaching me about the world of distributed computing, including that of the Grid. Always in good spirit, it was always straightforward to stop by his office for quick questions, discussions or technical inquiries. I also greatly appreciate his adamant approach to solving any technical issue that should arise, taking the time to ensure all required back-ends being up and running at all times.

I want to thank Håvard for broadening my research scope, introducing me to the ALICE Collaboration and CERN, and in turn allowing me to cooperate with other researchers on greater problems. Both before and throughout my dissertation, Håvard has also graciously and patiently reviewed my writings, and provided essential feedback for not only the contents, but also for details large and small regarding proper layout and formatting. I am also grateful for his effort in providing directions and pointers for navigating through all formalities, allowing me not only to formalise everything as needed for the position at CERN, but also to ensure all additional requirements were properly fulfilled.

I also want to thank Kristin for her aid in all things administrative, contributing to the time as a research fellow being problem free throughout – and likewise for all guidance and feedback. This includes taking the time to organise events for all PhD fellows, enabling us to meet, learn and exchange ideas. Additionally, I would also like to extend my sincerest and best wishes to her family.

In addition to my supervisors at the Western Norway University of Applied Sciences, I would also like to extend my gratitude to my supervisor at ALICE, Latchezar Betev, who not only provided me with an interesting problem to investigate for my thesis, but also gave me a chance to do so as a Project Associate. I am deeply grateful for not only the opportunity, but also for all the guidance and feedback I received – providing me with all the tools needed to embark on this project, as well as allowing me time to complete my duty work at the Western Norway University. This thesis would not have been possible without his help.

I also want to express my gratitude to the people at CERN and the ALICE Collaboration, who both helped and supported me during my research. My entry to the project domain was guided by Costin Grigoraş and Miguel Martinez Pedreira, and I want to express my gratitude to them for their valuable support. To Costin, for his guru-like Java insight – always having excellent ideas and suggestions –, and to Miguel, for aiding me in the initial stages of the project – helping me familiarise myself with the code. I would also like to thank my colleagues Nikola Hardi and Volodymyr Yurchenko, who have not only been my office neighbours, but with whom I have also had the great

pleasure of sharing a publication. Likewise, I would also like to thank all my colleagues at CERN and ALICE throughout the project – for all your interesting discussions, aid and suggestions.

I would also like to give a very special thanks to Maarten Litmaath at CERN, who has provided invaluable help and correspondence throughout the project. Not only have his VO-Box and Unix expertise been essential, but Maarten also graciously offered to review my writings, providing countless advice, feedback and suggestions – for which I am truly grateful.

I also want to take a moment to thank the Western Norway University of Applied Sciences. For funding my position as a Ph.D. research fellow, but also for allowing me the time to take on work as a Project Associate at CERN. I would also like to extend my thanks to the Department of Computer science, Electrical engineering and Mathematical sciences – and its superb staff. I would like to thank Adrian Rutle, Lars Michael Kristensen and Volker Stolz for always being friendly, and having time to answer any of my inquiries. I also want to thank Lars for taking the time to review and comment on my Petri Nets paper – and for the initial suggestion. Additionally, I want to also give thanks to Volker, and to Josva Kleist of Aalborg University, for being examiners at my midterm evaluation. While I was fortunate of having supervisors well versed in all things formal, I also want to thank Pål Ellingsen, who always had time to help in this regard.

I would like to also express my appreciation and thanks to my colleagues at the Western Norway University. To Faustin Ahishakiye and Violet Ka I Pun, it was great sharing teaching duties with you. To the first of my E506 desk-neighbours, Rui Wang, thank you for the fascinating discussions about language, culture, China – and everything in-between. To my second neighbour, Aravinthan Yogarajah, there was always an interesting chat to be had, be it even on politics or society. Also, to my fellow CERN associate, Simon Voigt Nesbø, it was great to explore Adelaide with you. Additionally, I want to direct my thanks to all my fellow Ph.D. candidates, that I had the pleasure of spending time with in the office, during lunches, seminars or project work – thank you for the great social environment.

Finally, I want to direct my heartfelt thanks to my family, and to Karsten Storetvedt, for convincing me that pursuing a Ph.D. would be a valuable experience, and to my parents – who supported and encouraged me not only throughout this project, but also through these unprecedented times.

ABSTRACT

Grid computing is a method for automatically distributing batch-style computing tasks on a global network of heterogeneous computing centres. ALICE (A Large Ion Collider Experiment) – one of the four large experiments at the CERN LHC – uses Grid resources to process large quantities of its collected data. While often compared to the more centralised and often commercial Cloud Computing paradigm, Grid resources tend to be geographically spread across multiple sites, containing computer clusters of different characteristics. Loosely coupled and generally heterogeneous, both in terms of hardware and software, these clusters come together to form a distributed system across multiple administrative domains.

Heterogeneous clusters as found in Grid computing may create challenges from having to cater to multiple system requirements, configurations and deployment practices. To alleviate some of the challenges that may arise, concepts from Cloud Computing have in recent times been applied to "cloudify" the Grid infrastructure. Through the use of technologies such as virtualisation, a desired hardware and software environment can be simulated on a range of underlying configurations. This allows for the creation of homogeneous environments within an otherwise heterogeneous Grid – an approach that is today utilised on numerous Grid sites. However, while virtualisation has gained ground within the Grid, new practices and technologies have since emerged. Specifically, containers and elasticity have rapidly risen in both adoption and popularity, and today are often found used within the Cloud. While often compared to traditional virtualisation, containers create an isolated environment on the same operating system kernel, thus avoiding virtualisation overhead.

The ALICE experiment is currently exploring the use of newer Cloud concepts within its Grid infrastructure. This comes as part of the development of a new Grid middleware framework (JAliEn – the Java ALICE Environment), which presents an opportunity to integrate these concepts directly within the core at a middleware level. This potential integration forms the base of the underlying research question and consequent contribution of the current thesis – to investigate ways to make ALICE offline computing more flexible and easier to administer, through the use of Cloud concepts and technologies. In turn constructing a new and optimised Grid workflow that not only alleviates present challenges, but may also help satisfy the requirements of the ALICE collaboration in the upcoming LHC Run 3.

SAMMENDRAG

Grid Computing er en teknikk for automatisk kjøring av større partier med distribuerte maskinoppgaver på et global nettverk bestående av heterogene datasentre. ALICE (A Large Ion Collider Experiment) – et av de fire store eksperimentene ved LHC på CERN – bruker Grid ressurser til å behandle store mengder av sine samlede data. Til tross for ofte å bli sammenlignet med det mer sentraliserte, og ofte kommersielle, Cloud Computing ("nettsky") paradigmet, pleier Grid ressurser å være geografisk spredt på flere datasentre bestående av klustere av forskjellige karakteristikker. Til tross for å være løst koblet, og med få avhengigheter, går disse klusterne sammen for å danne et distribuert system fordelt over flere administrative domener.

Heterogene klustere, slik som de finnes innad i Grid Computing, kan skape utfordringer ved å måtte tilrettelegge seg for flere forskjellige system-, konfigurasjons- og distribusjonskrav. For å redusere utfordringene som kan oppstå, har konsept og teknologier fra Cloud Computing i nyere tid blitt tatt i bruk for å gjøre Grid databehandling mer lik skytjenester. Ved å bruke teknologier slik som virtualisering, kan ønskede maskin- og programvaremiljøer bli simulert på et bredt utvalg av oppsett og konfigurasjoner. Dette åpner opp for å kunne skape homogene miljøer av oppsett innad i et ellers heterogent Grid miljø – en tilnærming som i dag brukes av en rekke Grid datasentre. Mens virtualisering har økt i bruk og popularitet innad i Grid miljøer, har imidlertid nye metoder, praksiser og teknologier begynt å dukke opp. Nærmere bestemt har både konteinere og elastisk lastbalansering økt raskt i både popularitet og bruk, og er i dag vanlig i sammenheng med skytjenester. Til tross for å ofte bli sammenlignet med vanlig virtualisering, gir konteinere muligheten til å opprette flere isolerte miljøer på toppen av samme systemkjerne, og unngår slik å påvirke ytelsen i særlig grad.

ALICE eksperimentet utforsker nå bruken av nyere konsept og teknologier fra Cloud Computing innad i sin Grid infrastruktur. Dette skjer som et ledd i utviklingen av et nytt Grid rammeverk (JAliEn – Java ALICE Environment), noe som skaper en mulighet for å kunne integrere disse konseptene og teknologiene direkte i selve kjernen av rammeverket. Denne muligheten danner grunnlaget for det underliggende forskningsspørsmålet og påfølgende bidraget i denne avhandlingen – å undersøke måter for hvordan databehandling innad i ALICE kan gjøres mer fleksibelt samt enklere å administrere, gjennom bruken av konsepter og teknologier fra Cloud Computing. Som resultat dannes det en ny og optimert arbeidsflyt for Grid databehandling som ikke kun bedrer på nåværende utfordringer, men som muligens også kan brukes til å tilfredsstille behovene til ALICE kollaborasjonen for kommende Run 3 av LHC.

CONTRIBUTIONS

This dissertation describes a system past its initial stages of development (JAliEn[1]), and contains contributions used for its setup and deployment in a production environment, through the use of technologies such as containers. The work outlined in this thesis was mainly carried out by the author, as part of a larger collaboration.

The original idea of a split JobAgent was conceived by Costin Grigoraş and Miguel Martinez Pedreira. Guidance and setup for the VO-Boxes was provided by Maarten Litmaath, who also carried out wider deployment beyond the initial test sites. Site Sonar was created by Kalana Wijethunga, and has incorporated code from the earlier Grid Sonar project by Nikola Hardi. The work involving the creation and development of the split agent, containerised VO-Boxes, as well as the further changes to JAliEn in light of the Research Question was done by the author – with the details of this work provided in Part II of this dissertation, as well as the subsequent Chapter 10. This includes the initial setup and deployment process, as well as the configuration and debugging necessary in order to achieve a functioning solution. The end result, as described in Chapter 12, is in turn set to be further extended by other collaboration members in correlation with the needs of the ALICE experiment.

[1] Final licence pending, and to be determined by ALICE in the future. Binaries are currently distributed under GPLv3 (https://www.gnu.org/licenses/gpl-3.0), due to included libraries.

LIST OF ABBREVIATIONS

ALICE A Large Ion Collider Experiment

AliEn ALICE Environment

API Application Programming Interface

ARC Advanced Resource Connector

ATLAS A Toroidal LHC Apparatus

BQ Batch Queue

CA Certificate Authority

CE Computing Element

CERN European Organization for Nuclear Research

CMS Compact Muon Solenoid

CS Central Services

CVMFS The CernVM Filesystem

DB Database

DHCP Dynamic Host Configuration Protocol

DIRAC Distributed Infrastructure with Remote Agent Control

DN Distinguished Name

DNS Domain Name System

EGI European Grid Infrastructure

EL Enterprise Linux

EOL End of Life

GPU Graphics Processing Unit

GSI Grid Security Infrastructure

GUID Global Unique Identifier

HEP High Energy Physics

HPC High-Performance Computing

HTTPS Hypertext Transfer Protocol Secure

IaaS Infrastructure as a Service

IP Internet Protocol

JA JobAgent

JDK Java Development Kit

JDL Job Description Language

JVM Java Virtual Machine

JW JobWrapper

LDAP Lightweight Directory Access Protocol

LFN Logical File Name

LHC The Large Hadron Collider

LHCb The Large Hadron Collider beauty

LRMS Local Resource Management System

LXC Linux Containers

MAC Media Access Control

MRE Managed Runtime Environment

OOD Object-Oriented Design

OS Operating System

OSG Open Science Grid

PaaS Platform as a Service

PanDA Production And Distributed Analysis

PFN Physical File Name

PID Process Identifier

PKI Public Key Infrastructure

POSIX The Portable Operating System Interface

RQ Research Question

SaaS Software as a Service

SE Storage Element

SL Scientific Linux

Slurm Simple Linux Utility for Resource Management

SOAP Simple Object Access Protocol

STDERR Standard Error

STDIN Standard Input

STDOUT Standard Output

TCP Transmission Control Protocol

TLS Transport Layer Security

TTL Time To Live

UID User Identifier

UMD Unified Middleware Distribution

URL Uniform Resource Locator

VM Virtual Machine

VO Virtual Organisation

WLCG Worldwide LHC Computing Grid

WN Worker Node

Contents

Part I

OVERVIEW

The utility model of computing - computing resources delivered over the network in much the same way that electricity or telephone service reaches our homes and offices today - makes more sense than ever.

—Scott McNealy [1]

CHAPTER 1

INTRODUCTION

Grid Computing is a distributed computing paradigm consisting of computing centres spread across the globe [2]. Being decentralised, loosely coupled and crossing multiple administrative domains, these computing centres come together as one unified large-scale system for executing batch-style computing jobs. The WLCG (Worldwide LHC Computing Grid) is one such system, designed by CERN to aid in processing data generated by the LHC (Large Hadron Collider) and its experiments [3][4][5].

ALICE (A Large Ion Collider Experiment) is one of four large experiments situated at the CERN LHC ring, and is dedicated to the study of heavy-ion collisions [6]. During operation, the detector generates data at rates of the order of 100GB/s after compression. Grid Computing is used extensively as one approach for accommodating and processing this data. Through a dedicated software, known as a middleware, computing tasks ("jobs") from ALICE members are accepted, managed and executed by distributing them among the individual computing centres in the Grid [7]. This includes assigning a received job to a computer that satisfies the given requirements of the job. However, unlike a centralised supercomputer, the hardware and software available on the Grid may differ, and the availability may be shifting.

Certain Grid jobs have specific resource requirements. Consequently, finding a compatible cluster, within the lowest possible network distance, is a priority. An argument could be made to utilise a more centralised approach, such as Cloud Computing, as opposed to a distributed system like a Grid – allowing access to a largely uniform set of computing resources, and avoiding moving large quantities of data [8]. However, the cost of using commercial services, such as Cloud Computing, for LHC computing is generally prohibitive, with funding countries generally preferring to provide computing resources from national infrastructure providers. An increasing number of Grid sites have thus instead opted for adopting concepts from Cloud Computing, such as virtualisation and elasticity, in their setup to bring over a subset of the benefits – contributing to a more "cloudified" Grid [9][10].

While a number of established Cloud practices and related technologies have started to appear on the Grid, these are largely implemented on top of existing platforms and solutions. Furthermore, new practices and concepts have since emerged in the Cloud, such as containerisation [11]. This thesis aims thus to explore means to benefit from newer Cloud technologies within a Grid context, while also finding ways to incorporate any potential benefits within the core of the Grid infrastructure used by ALICE.

This chapter will first provide a brief introduction to the ALICE experiment, Grid

Computing and the underlying technologies that will be the focus of the thesis. Then, present constraints and challenges within Grid computing will be discussed, followed by paradigms and technologies of interest that may potentially ameliorate these challenges. Relevant tools that have been used throughout the thesis will also be briefly introduced. The research question underlying the thesis will be discussed at the end of the chapter, followed by the approach and methods of research used to address it.

1.1 The ALICE Experiment

ALICE is one of in total eight – and one of four major – detector experiments situated at the CERN LHC ring [6]. Optimised for studying of heavy ion collisions, where the temperature and energy densities obtained in the collisions allow for the exploration of quark-gluon plasma: a state of matter where quarks and gluons are de-confined. This allows for the study of primordial conditions similar to those in the immediate moments after the Big Bang.

While the main focus of the detector is towards heavier Pb-Pb collisions, it is also used for the study of proton-proton and proton-nucleus collisions. This is done partially as a normalisation for comparing with the heavy-ion collision data, but also to study new aspects of physics at energies attainable by the LHC.

1.2 Computing in ALICE

The collisions within the ALICE detector have been recorded as data at a rate of 4GB/s after compression [6]. In 2020/2021, the detector was upgraded in preparation for the next run of the LHC, increasing the data rate to 3TB/s raw, and 100GB/s compressed [12][13]. Consequently, the bulk of generated data far exceeds the capacity of any single computing centre available to ALICE.

1.2.1 Grid Computing

An essential aid for handling and processing the masses of data generated by the ALICE detector is through the WLCG (Worldwide LHC Computing Grid) [4]. A key differentiator of Grid Computing from other remote computing paradigms, such as Cloud Computing, is its many heterogeneous computing centres across the world. These are generally smaller in size, where both hardware and software may differ not only between centres, but also between individual nodes. The connection and transfer rate between them may likewise differ. Nevertheless - together, these smaller centres form a powerful, best-effort, system intended for batch-style execution of computing jobs. This system provides a computing prowess equivalent of >1M modern CPU cores, and is used to store more than 500PB of data [14][15].

Grid middlewares

Both Grid computing sites and their users interact with the WLCG through a *Grid Middleware*: a software stack that is used to interface with participating resources in the WLCG and handle assigned batch-type computing tasks (known as "jobs") [16].

For ALICE, the Grid middleware used for the past 10+ years has been known as AliEn (The ALICE Environment) [7]. AliEn gives users a Unix-like directory tree and shell, allowing users to browse files on the Grid (through an abstracted filesystem) and perform simple commands (such as submitting Grid jobs).

1.2.2 *Resource matching*

Having largely heterogeneous computing centres, geographically spread in their relative position to recorded data, creates a number of challenges in regards to offline[1] data processing [17]. This is further exacerbated by users defining additional constraints for their jobs, such as having specific hardware or a preferred software package. As it is impractical to transfer large quantities of analysis data, it becomes a priority to find a matching, compatible, computing node in geographical vicinity - or at a computing centre with a connection that may compensate for the geographical distance.

1.3 Grid Computing: State of the art

Grid Computing has remained relevant despite the emergence of newer and more commercially popular services, continuing to be the main platform for all computational needs of the HEP (High Energy Physics) experiments at the LHC[2]. Available funding being a key constraint, with CERN member states prioritising national computing centres – and commercial services not being cost efficient for the intended large-scale use. Nevertheless, services such as Cloud Computing have popularised virtualisation and elasticity – technologies that may provide benefits within a Grid Computing context.

1.3.1 *Virtualisation*

Virtualisation provides an abstraction layer over computing resources, allowing the simulation of physical resources – which may in turn be used to partition hardware into smaller, isolated, resource subsets for applications to run in [18]. These resource subsets may be templated: allowing for quick deployment of certain hardware and software combinations.

The use of virtualisation may be used to greatly simplify the configuration of clusters. Furthermore, with one-click deployment of ad-hoc virtual machines, portability of each site is significantly improved. Another important gain of virtualisation is isolation. A computing job may be restricted to a virtual machine, with possible faults and crashes being contained within it. The virtual machine thus limits the impact on the underlying host, and helps improve security. Furthermore, virtualisation/isolation also provides means for performing resource management. By allocating a set number of resources to each virtual machine, a single job in one of these machines will be unable to consume all resources on the host. The number of virtual machines can also

[1] As opposed to *online* data systems that are tied to the run of a detector, *offline* systems process data that has already been stored in a computer system.

[2] http://alimonitor.cern.ch/display?page=jobStatusSites_RUNNING

be automatically scaled to match present demand, based on the resources available on the host (commonly known as "elasticity").

1.3.2 Grid Cloudification

Given the benefits described in the previous subsection, an increasing number of Grid sites have started to adopt the concepts of virtualisation and elasticity, contributing to a more "cloudified" Grid [9]. Deploying these concepts on existing Grid infrastructure has however been a gradual process, with many sites only adopting these practices in recent times. Consequently, while these concepts have risen in popularity within the Grid, other technologies and concepts have since emerged in the Cloud.

Containers: Towards more portable environments

One technology that has gained significant traction within Cloud Computing is containerisation [11]. Unlike virtual machines, containers do not require simulating a separate hardware stack. Instead, native code is run in a "user space" encapsulation, an approach often known as *operating system virtualisation* (i.e. creating a new environment on top of the native kernel). With no virtualised hardware layers and no virtual machine bootup, the theoretical impact on performance becomes negligible. Moreover, the reduced complexity contributes to container launch being near instantaneous, combined with a reduction in I/O delay (as data is not required to navigate through multiple virtualised hardware layers). The latter is of particular interest, as delayed I/O operations are a known bottleneck within virtualised systems – especially in a data-centric environments such as the Grid.

1.3.3 CVMFS: The CERN VM Filesystem

In addition to technologies and practices from the Cloud, the Grid has also seen the emergence of novel technologies, such as the CERN VM Filesystem (CVMFS) [19]. Originally intended as a way to bootstrap lightweight virtual machine instances, CVMFS has since become the default method for software distribution within the WLCG and participating collaborations. It is implemented as a read only POSIX (Portable Operating System Interface) filesystem, that may be mounted on top of an existing filesystem tree – generally under */cvmfs* [20]. Each entry in this filesystem is cached on-demand, meaning that data is downloaded as it is read (from a central repository), with no need to download every entry in the filesystem.

1.3.4 New technologies, old frameworks

Despite trends and technologies from Cloud Computing being deployed within the Grid, limitations exist within these deployments. Specifically, such deployments must currently be placed on top of the preexisting Grid implementations, and are in turn restricted by the capabilities of the underlying software and hardware. While the hardware is generally updated regularly, the software frameworks within large parts of the Grid have remained the same for over a decade (Chapter 4). Consequently, many

Cloud practices deployed within the Grid are not used to their full extent. To rectify this, a full rewrite of existing Grid solutions is needed.

The ALICE collaboration is in the process of developing a new Grid software stack intended to replace the aging AliEn middleware, in preparation for Run 3 of the LHC [21]. With the data rate of the ALICE detector set to increase, concerns were raised if AliEn would be able to scale to the needs of the collaboration. Specifically, concerns have been raised regarding its ability to scale in terms of both deployment, as well the extensibility of new features - especially given existing limitations of the maintainability of the code (Section 2.6.2). Dubbed JAliEn, the new middleware is a complete a rewrite of the original AliEn. Intended as a drop-in replacement, many of the original approaches and concepts have consequently been brought over. As JAliEn remains in development, there are still possibilities to take advantage of new trends and technologies.

A way forward?

The current state of the art of Grid Computing presents an interesting intersection of emerging trends and technologies – combined with the needs of the ALICE collaboration for LHC Run 3. Containers are particularly an interesting technology, having no virtualised hardware layers and no virtual machine bootup. The reduced complexity contributes to container launch being near instantaneous, combined with a reduction in I/O delay – a known bottleneck within virtualised systems [22][23]. With a significant proportion of the ALICE Grid infrastructure running on virtualised hardware, there is a significant potential for improvements by switching to container technologies. This can be done on multiple levels within the Grid infrastructure:

- Containers could potentially be used to run the services required by the ALICE collaboration on Grid sites (on "VO-Boxes").

- They may also be used to create lightweight worker nodes – allowing jobs to be run in their own homogenised execution environments in an otherwise heterogeneous Grid. As an added benefit, these would also be simpler to deploy and configure, as opposed to "bare-metal" workers.

- Additionally, given their low overhead, containers may potentially be integrated directly within Grid middleware, and in that regard be automatically launched when needed to strengthen isolation or provide a specific execution environment.

With the JAliEn middleware now in development, there is a potential path for realising all of the above points, including the latter. Furthermore, with the emergence of CVMFS, deploying these changes concurrently through multiple Grid sites becomes feasible. As opposed to utilising Cloud technologies on top of preexisting Grid solutions, an attempt could thus be made for combining several of the new trends and technologies in a from-scratch approach, maximising the possible benefits.

1.4 Research question

This thesis aims to focus on applying the emerging trends and technologies found within Cloud Computing – with a particular emphasis on containers, being a possible future replacement for large parts of the virtualised Grid. With this in mind, the following research question is initially produced:

How can one make ALICE offline computing more flexible and easier to administer by using container technologies?

However, CVMFS has changed how software and configurations are distributed across the Grid. This includes the way containers, their images and related software are accessed. With both containers and CVMFS being adopted across services and frameworks, this allows for a tighter integration between services. Specifically, it allows for the exploration and creation of new computing models or *workflows*, for use within ALICE. With this in mind, it becomes evident that the initial research question becomes a limitation, and should be adjusted to expand the research focus:

Research Question (RQ):
How can a new Grid workflow be constructed, using new technologies, within the ALICE Grid infrastructure to make ALICE offline computing more flexible and easier to administer?

1.4.1 Notes on RQ

To avoid ambiguity, the terms "*flexible*" and "*easier to administer*" are defined as being able to scale freely in terms of use, while at the same reducing the number of steps needed for deployment and utilisation. Specifically, taking inspiration from existing limitations, any new solution should not be constrained in its ability to scale across sites, as well as in its ability to scale in terms of new features and functionality – lest it is to end up in the same situation as AliEn, where development has halted. This provides for several quantitative properties to examine, such as:

- Active sites
- Active jobs
- Performance/stability
- Extensions

With the correlation between them being of particular interest. Furthermore, examining the steps required for deployment provides an entrypoint for a quantitative comparison between old and new solutions, in turn aiding in determining if something is *easier to administer*.

1.4.2 Research method

Given the scope of the RQ, the research work is consequently divided into three parts, each corresponding to a set of tasks undertaken in order to answer the respective RQ:

- The first part investigates the introduction of new Cloud-related technologies within CERN computing centres for backend services, as means for providing a reliable infrastructure for VO-Boxes.
- The second part focuses on integration within JAliEn, constructing the foundation for a new workflow where the middleware is capable of utilising new technologies as needed to optimise Grid job execution.
- The third part explores means of deployment, combining the prior two parts through a common distribution mechanism (e.g. CVMFS).

It must be noted that the above points will also need to take into account related dependencies in setup and infrastructure, such the execution environment(s) and data access. Furthermore, any new solution must also be compared to existing de-facto standards in order to determine if RQ has been satisfied, in particular in regards to the points in 1.4.1.

1.5 Outline

The thesis is composed of three parts. Part I provides an introduction and overview of the research field, related technologies and current state of the art. Part II describes the contributions provided as part of the research work. Lastly, Part III provides a discussion on the contributions in light of the underlying RQ and broader scope.

CHAPTER 2

GRID COMPUTING

This chapter provides a detailed overview of Grid Computing, along with a number of its areas of application. Central technologies, protocols and configuration approaches will be introduced, which will be a topic for discussion in forthcoming chapters. The chapter will also introduce a number of common Grid deployments, including more modern, virtualised, setups inspired by Cloud Computing.

2.1 Background

Grid Computing was first realised in the late 1990s, with the advance of the Internet and high-speed networks [24]. Its inception was the general idea of using networks as means for delivering computing resources from a remote source, much like a general utility service - i.e. electricity or water. The result is a form of non-interactive distributed computing, composed of many loosely coupled computers acting together in order to solve large computational problems.

The computers available in a Grid are generally a "best-effort" offering, provided by individuals and organisations across the world – often for the purpose of a common goal. Consequently, Grids often form a unified, large-scale, system that not only crosses multiple administrative domains, but is also generally composed of heterogeneous resources.

2.2 WLCG

With experiments at the LHC producing between 50 - 100 PB of data annually and preparing for yet higher numbers, there is a great need for computing resources [4][1]. These experiments were consequently also the driving forces in establishing a global Grid Computing network, organising it as the WLCG (Worldwide LHC Computing Grid) project. Linking up national and international Grid infrastructures, it consists of more than 170 computing centres in more than 40 countries, providing >1M computer cores for processing data. The WLCG exists alongside global and regional Grid infrastructures, such as the Open Science Grid (OSG)[2], European Grid

[1] The data produced annually has since increased.
[2] https://osg-htc.org/

Infrastructure (EGI) [3] and NordguGrid[4] – all of which are peered together, allowing them to interoperate and share workloads.

2.2.1 *Tiers*

The WLCG is organised into four levels ("tiers") of computing centres, based on the services provided by them (e.g. storage and/or data processing), labelled Tier 0 - Tier 3 [25]:

- **Tier 0**: Mainly composed of the CERN Data Centre, which is first and foremost used for custodial storage of raw data generated by the detectors at the LHC. It also performs a first pass processing on this data.
- **Tier 1**: Sites within this tier are considered to provide first-class support for the Grid, viz. custodial storage for large portions of LHC data, as well as highly contributing to its processing. The sites are maintained by round-the-clock support, and are tightly connected together through a dedicated high-bandwidth network.
- **Tier 2**: Tier 2 is mainly composed of universities or other scientific institutes, providing resources capable of adequate computing power for specific analysis, as well as a corresponding amount of non-custodial storage. The vast majority of sites within the WLCG are Tier 2.
- **Tier 3**: There is no formal engagement between the WLCG and Tier 3 sites – these may consist of local clusters within a university, up to large sites. Institutes in Tier 3 make arrangements only with the LHC experiments they support and not with WLCG.

2.2.2 *Virtual Organisations (VOs)*

Computing resources within the WLCG are shared between several collaborations and users. Virtual Organisations (VOs) aim to facilitate the sharing of resources by grouping together access rules and conditions into a VO, where its members share a form of commonality between them – such as a common goal or requirements [26]. While a VO generally consists of a network of smaller groups and organisations, it thus displays many of the characteristics of a formal organisation.

VO-Boxes

Computing clusters participating in the WLCG may provide VO-specific edge services in the form of a VO-Box: a dedicated machine serving as the cluster entrypoint for computing activities at a given site. It allows jobs to be submitted and handled, and enables the storage and network connectivity of a site to be monitored. This in turn to inform decisions on where to let jobs read their input data from and where to let them store their output results. While originally universal across deployments and VOs,

[3] https://www.egi.eu/
[4] http://www.nordugrid.org/

most Grid collaborations have since moved away from their use, incorporating the needed functionality directly into the central services of their respective middleware stack. VOboxes nevertheless remain vital in the ALICE Grid computing model, where they serve as the host for a key AliEn service known as the *Computing Element* (Section 2.6.1).

2.3 Jobs

A *job* is the unit of execution within Grid Computing: it describes something to be executed, at some point, within the Grid[5]. This is commonly a shell script used to bootstrap and start a larger application, requested by a user. It could however be anything compatible with the operating system (OS) / execution environment of the Grid node assigned to the job, be it a shell script, Python script or precompiled package. The one commonality shared between jobs is their batch-style, non-interactive, approach, where all inputs are prepared in advance, and outputs are written to files. Within the WLCG, and in the context of ALICE, jobs are used for providing offline processing of experiment data, through running the *ROOT framework* (Section 2.4).

Users may specify the requirements for a job and its execution, as well as other properties such as the location and output files, through a job specification file. The format of this file differs depending on the Grid middleware used. For ALICE and its middleware, AliEn, the Job Description Language (JDL) is used, where each file consists of lines in an **attribute = "expression";** format.

2.4 ROOT

ROOT is a framework used for data handling and processing, aimed at solving analysis challenges within high-energy physics [27]. It is today used across high-energy physics experiments, other scientific fields – and also in the commercial world. It is object-oriented, consisting of numerous classes that are grouped by functionality into shared libraries. These libraries are largely self-contained and can be loaded by adding include statements as required for specific tasks. ROOT provides packages and libraries for a range of such tasks, e.g.:

- Histogramming and graphing
- 2D/3D data visualisation
- Curve fitting
- Statistics
- Matrix algebra

An essential feature for the above is data, which is stored in ROOT files: a compressed, self-descriptive, binary format. Here, data may be contained in a structure known as a

[5]The term *job* may in some cases also denote the scheduled job/task of a local batch queue system at a site. To avoid ambiguity, this thesis will instead use the term *task* in these cases.

tree, which in turn consists of *branches* and *leaves*. Entries can be retrieved by advancing the tree index, providing fast access to data by avoiding memory allocation problems.

ROOT has the ability to interface with Grid middlewares and related tools, including the file catalogue (Section 2.6.5) for the fetching of additional data files (if not already provided in the JDL). This is enabled through an abstract interface known as *TGrid*, with *TAliEn* being the implementation of it used by ALICE and the AliEn middleware [28].

2.5 Software Distribution

CVMFS (The CERN VM File-system) is the current default for software distribution within the WLCG: a read only, networked, file-system that may be mounted onto an existing file tree [19]. Software can run directly from CVMFS, without needing explicit installations. Instead, each file is downloaded as it is read. If a software is properly unpacked, avoiding large singular files, launch can be near-instantaneous, as files are fetched and cached locally on demand.

CVMFS was originally intended as a way to bootstrap lightweight virtual machines (VMs) (hence the 'VM' in the name) for job execution, where all OS specifics/configurations/packages would be fetched only as required after loading the kernel, ensuring compatibility and avoiding overhead [29].

Software distribution within ALICE has traditionally relied on a service known as *PackMan*[6] [30]. As the name implies, it acts as a common package manager for Grid-related packages, though without the need for administrator privileges, as each package can be deployed in a cluster file system area dedicated to ALICE. Due to scalability issues, it was later intended to be replaced with a torrent-based approach, before the advance of CVMFS [31].

2.6 Grid middleware

A Grid middleware is the software stack used to interface with participating resources/clusters within the Grid, and may be composed of multiple components, services and protocols [2][16]. Together, these form the very core of the Grid and provide a shell for users, handle job distribution among available computing nodes, besides providing the tools necessary for each component (e.g. servers, storage, network) to participate in the Grid. These include data management tools, secure envelopes for transfers and relevant Application Programming Interfaces (APIs).

2.6.1 *AliEn - The ALICE Environment*

As briefly mentioned in Section 1.2.1, the middleware used[7] by the ALICE collaboration is AliEn (The ALICE Environment) [7]. Written mainly in Perl, with additions of shell/C++, and composed of multiple interdependent components, it extends to all parts of the ALICE Grid, be it the user facing shell, the central services or the computing

[6] Initially spelled "PacMan". Not to be confused with the PacMan package manager used within Arch Linux based distributions (`https://archlinux.org/pacman/`).

[7] At the time of project start.

nodes. This section aims to provide an overview of how these interdependent components come together to form one distributed Grid system. Specifically, the execution of a Grid job within AliEn can be subdivided into five unique actors: the user, central services, computing element, batch queue and compute node/JobAgent.

User

User-facing interactions within the AliEn Grid are done through a client shell. It provides POSIX-like calls and a filesystem abstraction over the distributed files stored across the Grid. It also allows users to specify/create JDL files, and submit these into the Grid for execution.

Central Services

The central services consist of the components required for database connections, configuration (e.g. LDAP (Lightweight Directory Access Protocol)) and file lookup, as well as authentication handling. They also host the Job Broker, which examines the JDLs submitted by users to attempt to find Grid sites that match the requirements.

Computing Element

The Computing Element (CE) is the service responsible for generating work across the worker nodes in a cluster. It is located within the VO-Box for the cluster, and queries the Job Broker within the Central Services for matching jobs. Should there be any compatible jobs available, the CE will then generate a startup script for a *JobAgent*, and have it inserted into the batch queue of the local cluster.

Batch queue

Sites maintain a back-end queue of scheduled tasks through a resource manager, and distribute these among the connected computing nodes in the site. In the context of AliEn, these tasks are scripts used to start the JobAgent process on each node. The specific batch queue implementation used may differ between Grid sites, and may be behind a gatekeeper service that in turn interfaces with the queue. The term Computing Element (CE) is also used for such a gatekeeper service (as for the AliEn middleware component in the above subsection). To avoid ambiguity, the term CE is used throughout this thesis solely to refer to the AliEn middleware component, while the term Local Resource Management System (LRMS) will be used for the aforementioned gatekeeper service. Common options for such systems, translating Grid jobs into local batch system tasks, include HTCondor, Slurm[8] and TORQUE [32][33][34][35][36]. AliEn may also interface with other middlewares, such as ARC (Advanced Resource Connector) from NordguGrid, in a similar manner (Section 2.6.4) [37].

[8]Formerly SLURM (Simple Linux Utility for Resource Management).

Computing node

The computing node (or "worker node" (WN)) is a machine running jobs for the resource management system at a Grid site. It contains a process that pulls scheduled tasks from the local work queue. AliEn tasks are all used to start JobAgents – processes used to again pull and then execute Grid jobs from the central queue, matching the configuration of the node. This approach is known as *late-binding*, where a dedicated pilot process, a *pilot job*, provides the final job matching. As opposed to early-binding, where a job is already specified in advance, having it provided at runtime allows for coordination between the resources available at the worker node, and the requirements of the job. This approach has in many ways proved to be successful, improving reliability and efficiency of job execution, and is today a de-facto standard across the WLCG [38].

2.6.2 *Limitations of AliEn*

The AliEn middleware has been used reliably in production for over a decade within the ALICE collaboration. However, deprecated code has accumulated over this timespan, resulting in a large trailing dependency backlog, where over 250 dependencies have to be bundled at all times. Furthermore having a majority of the code written in Perl also comes with another consequence: as other languages have succeeded Perl in terms of adoption and popularity, such as Python, dedicated interfaces need to be used whenever utilising code from other projects. In turn, the maintainability of AliEn has gradually decreased with time, exacerbating the implementation of new features and improvements.

There is also a core limitation present within AliEn, specifically, the lack of a common API. Instead, AliEn relies on SOAP (Simple Object Access Protocol) calls, commonly used to forward direct database calls to the central services. This approach comes with two issues attached:

- As these SOAP calls are carried on top of HTTPS (Hypertext Transfer Protocol Secure), states are not saved. This results in often having to re-authenticate incoming calls, and given the quantities of calls received by the central services, significant computing resources become needed to handle authentication. Consequently, a dedicated accelerator had to be installed at the central services in an attempt to alleviate the pressure of incoming requests.

- The central services consist of multiple components, each listening to their own ports, that in turn accept and ultimately forward calls to the backend database. This consequently leads to multiple components having to fight for access and write rights to the same database, in turn creating potential contention and lock-up issues.

The above points have contributed to making the scaling of AliEn unpredictable: with part of the back-end and central machinery already having scaling difficulties, adding more potential load from sites, resources, JobAgents and possibly multicore jobs, only adds to this uncertainty - in addition to the previously mentioned difficulties of maintainability. With the ALICE detector set to multiply its data rate for Run 3,

concerns were thus raised if AliEn would be able to scale to the needs of the ALICE collaboration [39].

2.6.3 JAliEn – The Java ALICE Environment

Given the above concerns, a decision was consequently made to create a new middleware for the ALICE collaboration – rewritten from scratch, and in a more maintainable language. This decision led to the creation of JAliEn: The Java ALICE Environment [21].

JAliEn is a drop-in replacement for the AliEn middleware written in Java, replacing the aging Perl modules of AliEn with a more maintainable OOD (Object-Oriented Design) approach. Being a drop-in replacement, it contains components/actors equivalent to those described in Section 2.6.1: a JCentral component to provide the central services, a new JobAgent, a new CE and the JShell for user interactions. There are also a number of key differences:

- There is now a central API, based on serialisable Java objects, with direct JCentral connections possible from any component.
- A new authentication mechanism based around *authentication tokens*. These are full X.509 certificates, where the Distinguished Name (DN) field is used to divide the tokens into groups corresponding to Grid roles, with each role given specific privileges and limitations.
- Introduction of a *JBox* component running alongside all JAliEn instances, handling upstream authentication based on a full user certificate, as well as a translation between Java serialised objects.
- Additional database (DB) backends, based around Cassandra/Scylla [40][41].
- Support for load-balancing, where multiple JCentral instances may run in unison to improve scalability.
- *TJAliEn*, a new implementation of TGrid for JAliEn.

As the middleware is still in active development, a number of features are still being modified, and their availability during production rollout may change.

2.6.4 Other middleware solutions

CERN Collaborations

The use of the AliEn middleware, and upcoming JAliEn, is tailored to ALICE and its needs. Alternative frameworks from other experiments can be found utilised across the WLCG, such as *DIRAC* (in Large Hadron Collider beauty (LHCb)), *PanDA* (in A Toroidal LHC Apparatus (ATLAS)) and the *CMS workflow management system / glideinWMS* (in Compact Muon Solenoid (CMS)) [42][43][44]. Their underlying approaches and system architectures are found to share many similarities: late-binding, pilot-based job execution, a central task queue and running the vast majority of jobs as close to

their input data as feasible. The crucial difference between (J)AliEn and the other frameworks is in the dependence on VO-Boxes at sites, which are essential elements in the former, while absent in the latter.

A further difference lies in the handling of proxy certificates (Section 2.6.6). In AliEn, all jobs at a given site use the proxy certificate of the VO-Box operator at that site, with the file catalogue checking and recording on the behalf of which user job output files are to be stored. In JAliEn, at the time of writing, jobs are still submitted with proxy certificates at the vast majority of sites, but such certificates are no longer used at sites where the VO-Box submits its jobs directly to the batch system. Each job script submitted by a VO-Box comes equipped with a JAliEn JobAgent token to be used for fetching a task from the JAliEn central task queue (Section 7.2). In the other three frameworks, all pilot jobs are submitted by the central services, typically located at CERN, using a small number of robot identities. When a pilot is assigned a given task from the central queue of the experiment, it may also receive a proxy certificate of the user who submitted that task and will then run the task with that credential instead of its own. Furthermore, PanDA and GlideinWMS have already switched to the use of WLCG tokens to submit their jobs (where applicable), as the WLCG is set to shift away from the use of X.509 proxy certificates in the future [45]. However, all jobs remain equipped with proxy certificates for the time being, in particular because there are no production Storage Elements yet that support WLCG tokens for data management operations. At the time of writing, token support for job submission has been implemented and tested in both JAliEn and DIRAC – but not put into production yet.

As will be seen in Section 12.2, the approaches taken by the various frameworks to adapt to a changing computing landscape have significant differences. These large-scale, distributed systems tend to be very domain-specific, having to work with higher-level frameworks and services that would have been hard to generalize across multiple experiments. That said, PanDA, GlideinWMS and DIRAC have all found uses outside of CERN [46][47][48].

Grid infrastructures

While CERN collaborations, such as ALICE, have their own middleware, they will need to interact with global and regional Grid infrastructures that may likewise provide middleware stacks in some form. The extent to which this is done varies, where some provide common toolkits used to build a functioning Grid, while others provide a full Grid middleware suite, reminiscent of that found within LHC experiments.

As an example, EGI performs software verification on a set of products and solutions from third-party providers, where provisioned software becomes integrated in what is known as the Unified Middleware Distribution (UMD) stack - tested and verified software that can be used for Grid purposes within the EGI [49]. EGI also runs a catch-all DIRAC workload management service for VOs that cannot afford to run their own instances.

OSG, on the other hand, provides a CE/LRMS implementation based around HTCondor, which provides access to the resources within its infrastructure [50]. NorduGrid takes this approach a step further through the use of ARC, a purpose-built

dedicated Grid-middleware suite, used to interface with all its computing resources. The ARC Computing Element has been designed to be able to function as an edge service for HPC (High-Performance Computing) clusters whose computing nodes do not have access to the internet. The ARC CE can download job input data onto a shared file system and upload the job output data from that same shared file system. For pilot-based workloads, whose input and output data are not known in advance, a translation service is available to convert such workloads into fully-defined jobs as expected by the participating clusters [51]. In order to utilise the resources provided by ARC clusters, (J)AliEn uses a dedicated ARC interface to submit jobs, similar to that of an LRMS system.

2.6.5 *Storage*

Section 2.6.1 mentions that users are presented with a filesystem abstraction when viewing files on the Grid, appearing as a common Unix file tree. Within ALICE, this is handled by AliEn (and its successor *JAliEn*), which provides a file abstraction by utilising *pointers*. The user-facing file catalogue is composed of a database of Logical File Names (LFNs), providing the Unix-like directory tree. Each LFN points to a GUID (the Global Unique Identifier), which provides uniqueness[9] across the file abstractions in the catalogue and their specific versions, and points to a set of Physical File Names (PFNs), each of which holds a copy of the logical file in its respective Storage Element (SE) in the Grid. This is illustrated in Figure 2.1.

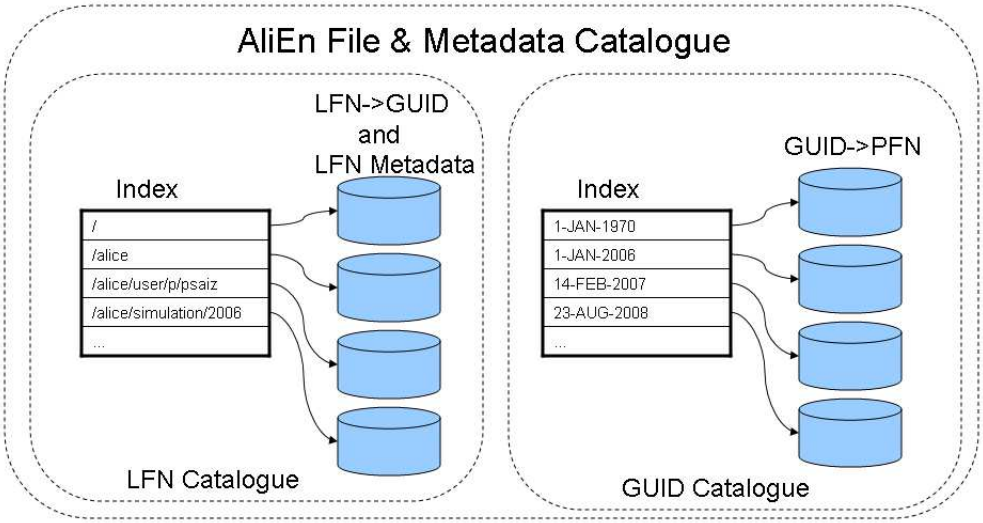


Fig. 2.1: Catalogue mapping within AliEn, where the user-facing LFN is mapped to a GUID, which in turn is used to map to the PFNs of a file [7].

[9]Generally through a dedicated hash algorithm, to prevent re-using values.

SEs are commonly powered by *XRootD*, a highly scalable file server which is a de-facto standard across the WLCG [52]. It runs on the Storage Elements positioned across Grid sites, allowing them to provide data storage in addition job execution[10]. Alternatives to XRootD, such as *dCache*, are also supported by AliEn [53].

2.6.6 Security

Security within the WLCG is modelled after the Grid Security Infrastructure (GSI), initially introduced with the now defunct Globus Toolkit middleware [54][55]. It introduces a set of tools, libraries and protocols for providing the following foundations for a secure Grid:

- *Authentication*: Validating the authenticity of an entity.
- *Authorisation*: Providing access specific to that entity.
- *Data integrity*: Ensure data is not altered during transmission.
- *Data confidentiality*: Ensure data is only read by intended entities.

The above terms are mainly satisfied through the use of PKI (Public Key Infrastructure), an asymmetric key cryptography that gives each user/entity a private and a public key – where each key can be used to decrypt the contents encrypted by the other, and vice-versa [56].

Each entity is given a certificate mapping them to a user identity, signed by the private key of a Certificate Authority (CA), a participant of the PKI considered trusted, and which may be verified through a corresponding public key, providing *authentication*. The user identity may be mapped to specific permissions as needed, which in turn provide means for enforcing *authorisation*. Encryption using public/private keys is likewise also used to provide *data confidentiality*, where data is encrypted using the public key of the receiver, and decrypted using their private key. The data transferred using this approach is generally a unique key, exchanged for the purpose of establishing a shared secret, in turn allowing the use of simpler/more performant symmetric encryption algorithms.

Lastly, *data integrity* is achieved by generating and comparing hashes of the encrypted data messages, before sending and after receiving. Should the hashes differ, it would indicate that the contents have been altered.

X.509

At the heart of GSI is *X.509*: a defined standard for the format of PKI certificates. It outlines fields for *subject*, *issuing CA*, *validity* and other information considered relevant. The standard also outlines the use of a certificate revocation list, used to identify certificates that have been revoked by the issuing CA before its expiration date.

[10] It must however be noted that not all Grid sites have their own SE.

Proxy certificates

Another central aspect of security within the WLCG is the use of *proxy certificates*. These are short lived certificates that are issued on behalf of a user, valid between a few hours and up to a few days. Their purpose is to allow services to act on behalf of the user, such as a job on a remote site needing access to the files of said user. It would thus need to take on the identity of that user for a short duration – something which could technically be achieved by transferring the private key, albeit with a high risk of compromising security. To reduce this risk, a separate certificate is instead generated, commonly known as a *proxy*. It comes with its own private/public key pair, while being signed with the private key of the generating user. This allows it to have equivalent privileges, albeit for a significantly shorter timespan.

2.7 Limitations of Grid Computing

While Grid computing is an essential tool for handling large quantities of data, such as those generated by ALICE, it is not without limitations. Being centred around batch-style execution of computing jobs, makes it not only unsuitable for interactive tasks, but also for time sensitive workloads, as there is no guarantee that a specific job will be executed within a certain timeframe. Furthermore, there is a lack of flexibility in terms of dynamic scaling, as computing resources are unable to instantly scale up, or down, based on demand. While this can be ameliorated to an extent based on the deployment type (Section 2.8), hardware must still be physically installed and thereafter configured before it can be put to use. Lastly, the Grid is largely *heterogeneous*, where the available computing resources may differ in several regards, such as in their hardware architecture, OS, installed packages, versions, and individual configurations, and likewise in terms of their availability and data proximity – requiring extra care to be taken in regards to job execution.

2.8 Deployments

This section examines possible site deployments of AliEn/JAliEn, highlighting current trends in contrast to more "traditional" approaches.

2.8.1 Traditional

The initial means of deploying AliEn at a computing site have required dedicated machines for each purpose, with their software stack manually installed and maintained by a site admin. This generally includes a dedicated front end machine serving as a VO-Box, with a local installation of AliEn for running the CE service, as well as local installations of any other required services, and their dependencies. The computing nodes use the shared VO-Box installation of AliEn, which is called by the agent script when an ALICE job is started by the batch queue. Site-specific configuration for AliEn is pulled centrally from LDAP, which can be overridden if necessary in the `~/.alien` directory. The deployment is illustrated in Figure 2.2.

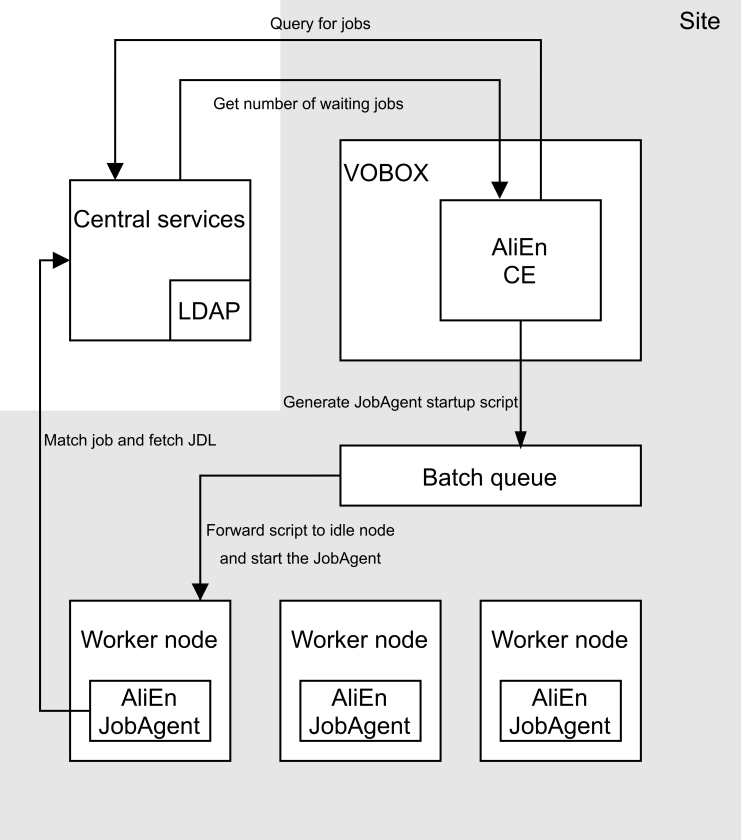


Fig. 2.2: A traditional AliEn site, with persistent worker nodes.

Traditional with CVMFS

With the emergence of CVMFS, it is now standard to run AliEn, and related packages, directly from it – substituting any local versions in the PATH with the CVMFS binary. This provides not only a simple way to set up a working environment, but also facilitates maintenance, as new versions are periodically pushed to CVMFS.

2.8.2 *Virtualised*

As opposed to manually configuring each machine in a site cluster, configuration can be made simpler through the use of *virtualisation*[11]. The computing resources at a Grid site may in this case be split up into multiple virtual machines, e.g. one per available CPU. Each of these run on the same preconfigured image, providing the exact same OS and environment as preferred for job execution, further simplifying configuration without having to resort to configuration management tools.

Elastic pool of resources

The use of virtualisation within a site can be taken one step further, by *pooling* the computing resources – i.e. combining all resources across site nodes through a common resource manager. This allows a Grid computing site to act more akin to Cloud Computing, where virtual machines may be launched on demand as jobs are

[11] More on this in Section 3.2.

assigned to the site, allowing unused resources to be freed and repurposed. Solutions, such as VMBatch, allow this functionality to be integrated directly within the local batch queue [57]. Figure 2.3 illustrates such a deployment.

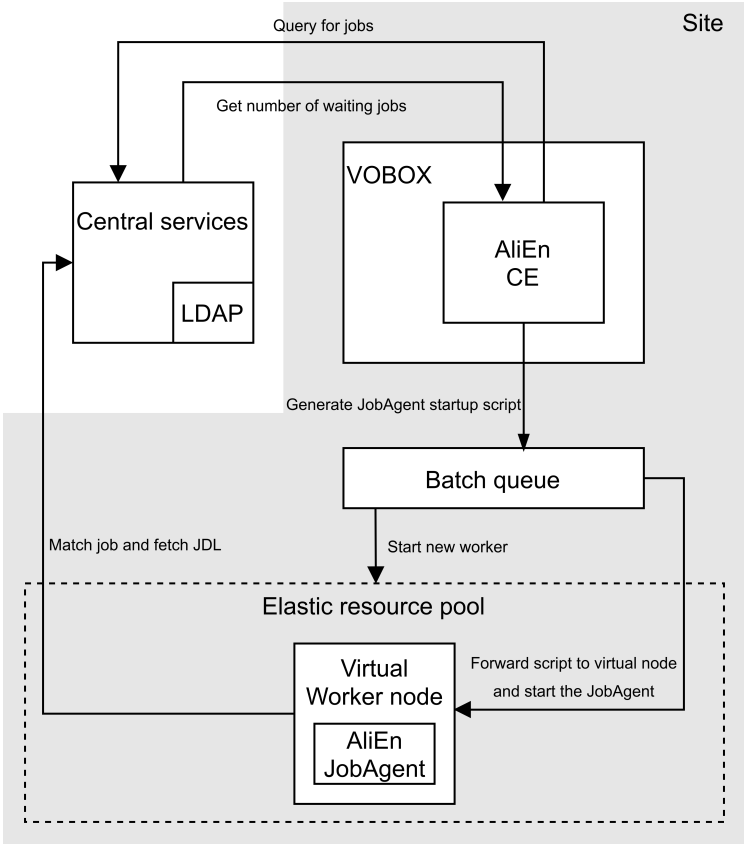


Fig. 2.3: A virtualised AliEn site, with a persistent resource pool that creates virtual worker nodes when needed.

CHAPTER 3

CLOUD COMPUTING

Cloud Computing provides an alternative approach to networked computing resources. With a large user base and commercial backing, it has popularised several novel technologies, such as virtualisation, containers and elasticity. This chapter provides an overview of Cloud Computing, introducing key concepts, technologies, as well as highlighting differences to Grid Computing.

3.1 Background

Cloud Computing shares many characteristics with Grid Computing, and is often labelled as its successor given its commercial success – despite originating orthogonally to Grids [58]. The first mention of it can be traced back to the mid-90s, roughly the same time as when Grid Computing was realised, though its origin is more closely related to that of *virtualisation*: the ability to split computing resources into multiple instances, that in turn may be configured with their own unique environments [18]. The advances in networking, and common access to the Internet, that emerged in the 1990s provided the possibility to offer these virtualised instances commercially over a network. As the sizes of data centres have since vastly grown, so to have the capabilities of Cloud Computing.

3.2 Virtualisation

As briefly mentioned in the above section, virtualisation provides means to split physical hardware into smaller, virtualised hardware stacks, that may in turn be customised and configured with their own runtime environments. In other words, it allows for the software implementation of a computer, a *virtual machine* (VM), that may be used to execute software like a traditional, physical machine. The concept emerged in the early days of computing at IBM, when the hardware costs were high, and the support for multiple concurrent users was limited [59].

Virtualisation works through the use of a *hypervisor*, a service that creates virtual hardware abstractions on top of the physical hardware. Each of these virtual abstractions may in turn run their own complete OS and software stacks, and appear as normal machines to end users. Hypervisors may run directly on top of the physical hardware (known as "bare metal" or Type 1 hypervisors), or on top of a preexisting OS (known as a Type 2 hypervisor). This is illustrated in Figure 3.1.

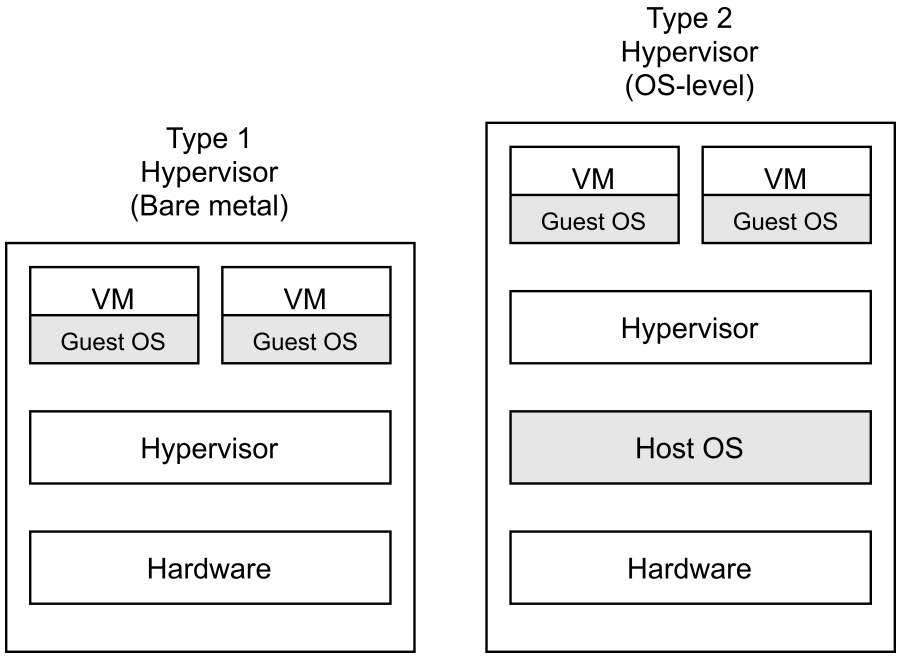


Fig. 3.1: Type 1 and Type 2 hypervisor stacks side-by-side. The latter running the VM instances on top of a preexisting host OS.

The term "virtual machine" is in this thesis used interchangeably with what is also known as a "system virtual machine": a full hardware abstraction layer that may be used to run several, full, OS stacks on the same physical hardware. The term VM may, however, in some specified occasions also be used for a "process virtual machine". Also known as Managed Runtime Environments (MREs), this is generally an OS layer abstraction, used by programming languages to ensure portability across OS platforms. A common example is the JVM: The Java Virtual Machine, which allows cross platform compatibility for Java applications, granted the target platform is capable of running the JVM [60].

3.3 Service Models

The types of services Cloud Computing providers may offer differ depending on the intended goal / requirements, and may ultimately be categorised as three distinct models based on their level of abstraction [8]. These are commonly known as:

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

3.3.1 Software as a Service (SaaS)

SaaS enables users to access and utilise applications running within a Cloud Infrastructure, generally through a web-browser or a thin client. All underlying infrastructure, such as network, servers and storage, are handled by the Cloud provider, allowing users a straightforward way to access a preconfigured application over a network.

3.3.2 Platform as a Service (PaaS)

PaaS lowers the abstraction one level, compared to SaaS, and enables users to deploy their own applications onto a preexisting infrastructure offered and maintained by a Cloud provider. The applications may in turn be created using libraries, languages and tools supported by the provider.

3.3.3 Infrastructure as a Service (IaaS)

IaaS descends one level below PaaS, allowing users to not only deploy their own applications, but also decide on the underlying OS and its configurations. In other words, IaaS gives users full access to their own virtual machine, which may be set up and configured in any way needed by the user, be it as an application, server or database. The underlying hardware and network connectivity are maintained by the Cloud provider.

3.4 Containers

While virtualisation and virtual machines have been considered to be at the very core of Cloud Computing, the rise of *containers* has contributed to a shift in this perception – a technology that provides many of the benefits of virtualisation, without the overhead of having virtualised hardware, and which is now a staple within Cloud Computing [11] [61].

Containers provide distinct runtime environments for applications, in many ways akin to that of VM, but achieves this through the use of features available in the Linux kernel. Through the use of technologies such as `cgroups`, `chroot`, `seccomp` and `namespaces`, unique and isolated environments may be created, each with their own filesystem, root account and packages. This can be used to not only create self-contained application/service wrappers, but also to run multiple Linux distributions side-by-side on top of the same Linux kernel – creating an end-result similar to a virtual machine. A comparison can be seen in Figure 3.2.

3.4.1 Container types

The above section mentions two possible uses of containers: to create a self-contained runtime for applications, and to run multiple services reminiscent to that of a VM. These use-cases utilise two distinct types of containers: (operating) system containers, and application containers [62].

System containers provide a user-space isolation that allows for the installation of different packages, libraries and file hierarchies. The end result may be a complete Linux distribution, providing a similar end-result to a VM, sans the use of a hypervisor. This results in the ability to run multiple processes in a portable environment, while avoiding any performance impact coming from the use of hardware virtualisation, in addition to having rapid startup times.

Application containers are the most popular/common form of containerisation. Intended for wrapping single applications/processes as independent packages, they allow for greater portability, as well as more control over security, resource restrictions

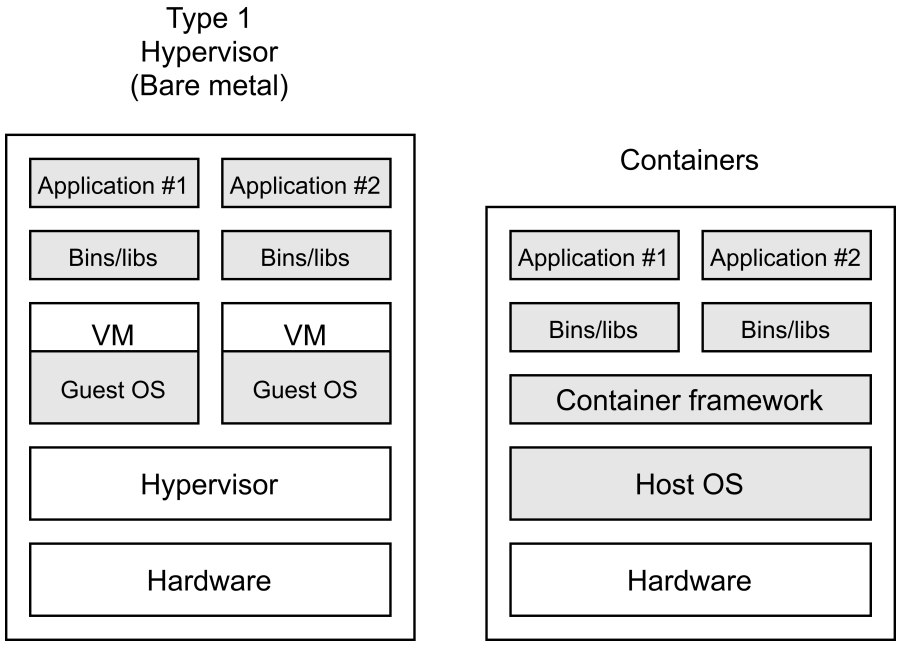


Fig. 3.2: Containers compared to a Type 1 hypervisor. The former do not need a hypervisor or guest OS, instead using a container framework to wrap applications and binaries in their own isolated environments.

and use of dependencies. By placing and shipping an application within its own container, the risks of inconsistencies, unreliability and compatibility issues may be greatly reduced.

3.4.2 *Container frameworks*

While containers mainly utilise existing capabilities of the Linux kernel[1], their rapid ascent to popularity can be attributed to the release and availability of tools such as *Docker* – tools that use the available kernel features to create a framework and format for running, deploying and sharing containers easily [65]. Containers may here be defined as ready-to-go images, stored in a central repository for sharing, and largely deployed automatically. Other common container frameworks are Podman, Singularity and LXC (Linux Containers), with the first two frameworks, as well as Docker, being intended for application containers, and the third for system containers [66][67][68].

3.5 Cloud deployment types

Cloud Computing may be deployed through multiple means, where the ownership, location and size of the deployment may differ [8]. These means of deployment may be summarised into four distinct deployment types, where each has its own intended uses and benefits:

[1]Containers are also natively available on FreeBSD and Windows, but with some limitations [63][64].

3.5.1 Public Cloud

As the name implies, public clouds are accessible to the general public over the internet, be it as a paid subscription or free of charge. It is the most common deployment type, and characterised by having all essential infrastructure managed by the service providers. This has the benefit of enabling lower costs, higher scalability and reduced management at the cost of privacy/security, as all data will be located off-premises.

3.5.2 Private Cloud

From a technical perspective, a private Cloud has few differences from a public Cloud – with the exception of it being managed and/or owned by a private entity, that in turn restricts usage to only those allowed access by said entity. This is generally a specific company, which hosts physical servers on-premise, and limits access to its own users. While this approach provides more privacy compared to a public Cloud, it requires having to maintain local servers on-site. This reduces the scaling flexibility found in publicly managed Clouds, as upgrading capacity will require investing in physical hardware.

3.5.3 Community Cloud

A community Cloud can in many ways be seen as an extension to a private Cloud, where the infrastructure may be managed and shared by multiple private entities and their users. This may aid in cooperation between the participating entities, as well as help offload costs.

3.5.4 Hybrid Cloud

A hybrid Cloud combines the approaches of both public- and private Clouds, in attempt to bring together the benefits of both. This results in a mix of on-premise, privately managed, resources, and remote resources managed by a Cloud service provider, which are bound together through a form of communication/data exchange. The end result may, as an example, allow a company to run their applications on a remote Cloud, where the capacity may scale depending on demand, while the authentication handlers of said applications may be hosted privately, to allow for more secure data handling.

3.6 Characteristics

Given the properties of Cloud Computing described throughout this chapter, the key characteristics may be reduced to the following key points:

- *Remote access to interactive resources* – Users may access remote resources over a network to perform tasks, including those considered mainly interactive.
- *On-demand resources* – Resources may quickly be scaled up or down through the Cloud provider.

- *On-demand self service* – Users may provision additional computing resources as needed through an online portal/API, without requiring human interaction.
- *Measured service* – Resources may be monitored, combined with a metering capability, to provide detailed usage statistics, which in turn may be used to optimise the service.
- *Elasticity* – Resources may be scaled up or down, automatically, depending on load/demand.
- *Multitenancy* – Resources are shared across a large pool of users, lowering costs, and increasing flexibility.
- *Maintenance and configuration* – Maintenance is primarily handled by the Cloud provider, which also offers preconfigured deployments.

3.7 Grid benefits

Within a Grid Computing context, the characteristics of Cloud Computing present interesting possibilities that in turn may be used to overcome shortcomings found in Grid Computing. In particular, Cloud Computing presents three key areas that may be beneficial if applied within the Grid:

- Virtualisation – Grid Computing consists of largely heterogeneous resources. Through virtualisation, preconfigured environments, tailored for specific jobs, may be created. This in turn may not only simplify configuration, but also be used towards creating a more homogeneous Grid – having a uniform environment across sites for jobs.
- Elasticity – Virtualisation within the Grid may be taken one step further, by launching a tailored VM instance to execute jobs as they come. The whole Grid site may in that case be set up as a shared resource pool (as mentioned in Section 2.8.2), where unused resources may be repurposed.
- Containerisation – Containers allow us to take the above suggestion even further: not only can we launch tailored virtualised instances when new jobs are detected, but we can incorporate similar features directly in the Grid middleware, thanks to the small overhead of containers.

CHAPTER 4

STATE OF THE ART

This chapter provides an overview of the state of the ALICE Grid and infrastructure, as well as related site deployments at CERN, at the time of project commencement. The aim being to highlight how relevant technologies are being utilised, as well as identifying their potential issues and bottlenecks.

4.1 Central infrastructure

The central infrastructure is in this chapter used to refer to the backend servers providing the main upstream API – mainly serving the middleware central services – as well as credential databases and storage, physically located at CERN. In addition, there are also several VO-Boxes present, providing entry points for several ALICE Grid clusters at CERN.

4.1.1 Central Services

At the time of project commencement, the Grid central services are dedicated to serving the AliEn middleware. There is a smaller number of JCentral instances coexisting on the same machines intended for JAliEn, solely intended for development and testing, and otherwise not used in production. Storage, authentication and LDAP are shared between the AliEn/JAliEn instances.

4.1.2 VO-Boxes

In addition to the central services, a number of ALICE Grid sites can also be found at CERN, each handling a fraction of the jobs. These sites are among the largest, and together account for roughly half of all ALICE jobs[1]. Their VO-Boxes are all virtualised, existing only as VMs spread across a number of dedicated "alienvm" machines. There are in total five alienvm machines, all of which utilise Ubuntu[2] as the base OS, with VMWare as their hypervisor.

[1] As reported on Alimonitor http://alimonitor.cern.ch/ [69]
[2] Spread across multiple Ubuntu LTS versions (14.04 - 18.04).

Package Management

Across all CERN VO-Boxes, both packages and installation are handled through CVMFS, running all services directly from the shared filesystem. Packman has been completely phased out.

Worker Nodes

Worker nodes for the CERN sites are taken from a pool of dedicated machines, shared across all LHC experiments, and maintained by CERN IT – this in contrast to the VO-Boxes, which are handled directly by the ALICE Offline team. Nodes are up-to-date, and in this case come with CentOS 7, with required packages for most jobs as well as CVMFS access.

4.2 Sites

A key attribute of the Grid is its heterogeneity – being composed of many distributed computing sites with different properties. Consequently, sites differ not only in regards to their batch systems or nodes, but also in their use of new and emerging technologies/practices. Key traits of the Grid at the time of project commencement are summarised here[3]:

Distributions

CERN has earlier collaborated with Fermilab to create the Scientific Linux (SL) GNU/Linux distribution, based off the commercial RedHat distribution [70]. SL6 has been the default recommendation for use within the Grid, be it on the computing nodes or VO-Boxes. However, CERN then discontinued SL in favour of CentOS [71], starting with CentOS 7 as the successor [72]. The WLCG is thus largely composed of CentOS 7 nodes, though with a significant portion still on SL6, waiting to be updated. However, not all Grid sites follow the recommendations, and a small number of sites can be found to use a version of Fedora[4] (largely compatible with CentOS) or Ubuntu[5].

Site Services

Site services refer to the VO-Boxes and their associated services, including the CE and batch queue system. As implied in the above section, most site VO-Boxes run on either CentOS 7 or SL6. CentOS 7 sites are more likely to run the AliEn CE directly from CVMFS, while some SL6 sites also have local installations. A few sites, such as GSI[6] Darmstadt, also run their VO-Boxes in a VM. The most common system for resource management is HTCondor, with Slurm and ARC also being popular options. Other systems, such as TORQUE, are also used.

[3] Data gathered using probing scripts and LDAP records (Section 10.1.3).
[4] https://getfedora.org/
[5] https://ubuntu.com/
[6] Gesellschaft für Schwerionenforschung.

Use of Cloud-related technologies

As mentioned in Section 3.7, the use of Cloud-related technologies within a Grid context can be advantageous. Consequently, a number of sites have started to incorporate these within their local job execution workflows – this generally through the use of virtualisation. Some sites only utilise virtualisation here as means to simplify setup and configuration, with a static number of VMs acting as replacement for physical hosts, though a growing number have their setups configured as a collection of pooled resources (Section 2.8.2) – an approach that has been extensively documented and tested [10]. Some LHC experiments already have job workflows using containers, as opposed to using VMs [73][74].

The consequences of the above Grid properties and site choices are further discussed in Chapter 10.

4.3 Middleware

4.3.1 *AliEn*

The AliEn middleware has driven the ALICE Grid for over a decade, and remains the only production middleware at project start within ALICE, for both the central services and site nodes. Development efforts have however shifted towards JAliEn, with AliEn now only receiving small maintenance patches. Furthermore, there is no longer any central development repository for AliEn, with all patches being applied directly to the AliEn CVMFS repository instead.

4.3.2 *JAliEn*

While JAliEn is positioned as the successor to AliEn, it remains under heavy development at project commencement [21]. It is consequently not used at any site, though as mentioned in Section 4.1.1, JAliEn central services are present and available, providing the needed APIs for testing.

In its initial state, JAliEn implements a large subset of the core AliEn functionality: it provides the minimum necessary API endpoints, and means to start and run jobs – albeit through a barebones approach, lacking a number of essential steps related to authentication, reporting and file handling. Furthermore, as the main priority is to first cover the functionality of AliEn, new features of JAliEn remain unused or unimplemented. This is a central point of Chapter 7.

4.4 Potential issues and bottlenecks

The state of the art described in this chapter presents an intersection of new and established practices, with trends from the Cloud making their way into Grid sites and services. While these trends, together with the new middleware developments of JAliEn, may be beneficial, bottlenecks may be present in their current use/deployment. Specifically, a number of concerns may be raised based on the descriptions presented earlier in the chapter, listed below.

4.4.1 Subset of features

Unlike a true Cloud with dedicated resources, a majority of Grid sites utilising Cloud resources deploy these on top of existing configurations, in turn preventing additional functionality: e.g. as opposed to creating a resource pool, VMs are deployed as replacements for physical hosts. While this brings the benefit of a simplified configuration, other features, such as elasticity, are absent. Furthermore, from a middleware perspective (both AliEn and JAliEn), jobs and their pilot processes are unaware of Cloud features/functionality, or of being executed in a virtualised environment. Having this ability could allow the middleware to not only better handle jobs (e.g. adding safeguards against VM issues), but also even provide means for the middleware to handle elasticity directly, by automatically launching VMs/containers for incoming jobs.

4.4.2 Increased complexity

Deploying a Cloud-inspired setup on top of existing configurations may initially be convenient for the sake of easily gaining access to new features. This may however come at a cost: both the old and new configurations will need to be maintained, as well as any intermediary wrapper used to combine them. Not only will this reduce benefits provided by using Cloud related technologies (simpler setup and configuration), but likewise also affect reliability and potentially increase fragmentation (see subsection below). One of the reasons for AliEn falling behind in development may be attributed to an accumulation of new modules being added on top over time.

4.4.3 Reliability

Installing new features and services on top of existing solutions increases the deployment stack. This opens up more potential sources of error, that in turn may cause the full stack of services to fail. There is also an increased risk of encountering performance issues, as data and settings may need to propagate through several software layers. A custom solution/wrapper may also be needed to make these services properly work together – as ad-hoc solutions being used by sites in order to adapt to VMs, containers and elasticity, may encounter issues in the long term.

4.4.4 Fragmentation

While the Grid already exists as a heterogeneous and fragmented system in terms of configuration, the combination of multiple ad-hoc approaches for integrating Cloud-related technologies, as well as the adoption of multiple new technologies from multiple vendors, makes the Grid even more fragmented than before. Where both sysadmins and software previously knew, to an extent, what to expect from execution on the Grid, new knowledge will now be needed in order to adapt to the changes.

Part II

CONTRIBUTION

Adapting old programs to fit new machines usually means adapting new machines to behave like old ones.

—Alan Perlis [75]

CHAPTER 5

FOUNDATIONS FOR A NEW GRID WORKFLOW

Having introduced core concepts and technologies in Chapter 2 and Chapter 3, combined with having examined possible concerns and bottlenecks in Chapter 4, this chapter returns attention to the thesis RQ. Consequences of the current state of the art will be discussed, followed by relevant approaches for potentially improving the situation. Afterwards, related work will be discussed, before drafting a foundation for a possible new workflow for the ALICE Grid.

5.1 Adapting to a changing landscape

Chapter 4 describes the current state of the ALICE grid as an intersection of new and established practices. A recurring theme in this description is the deployment of Cloud-related technologies, such as virtualisation, on top of existing configurations. While it provides means to quickly gain access to new features without having to drastically change infrastructure, there are also a number of caveats to this approach (as discussed in Section 4.4). The negative consequences of one of these caveats, the increased stack from adding VMs within existing clusters, have already been a source of issues for the central service hosts within CERN: networking may occasionally shut down, without warning or log entries, resulting in service hosts having to be constantly monitored, with on-call personnel having to reboot and relaunch services. This is but one of many sites within the Grid, and with Run 3 of the LHC looming – with its increased processing and storage needs – having any reliability or performance concerns is unsettling.

As opposed to shoehorning new computing models and technologies on top of existing solutions, a better approach may be to create a new solution, fully embracing and integrating new computing paradigms wherever these may be the most beneficial. This is of particular interest due to three current developments at project start:

- The emergence of containers, increasingly being embraced within the Grid
  - Potential for both coexisting with, and replacing, VMs in service hosting, execution and infrastructure.
- The now rapid development of JAliEn
  - Can in many ways be seen as a "blank canvas" with core features implemented, ready for extensions.

- Full adoption of CVMFS
  - A quick, centralised, way of rolling out software across sites.

It must be noted that a completely new solution, which also fully embraces new paradigms and computing models, cannot simply be a deployment or component. Large adjustments to the current Grid stack would be needed – in turn requiring a reimagined Grid workflow.

5.1.1 *Defining a Grid workflow*

The term *Grid workflow* has been used recurringly when discussing the possibilities of adopting new technologies within the Grid. Whereas a workflow would normally indicate a set of sequences of tasks required for completing a process, a *Grid workflow* would in this case be the steps and sequences needed for achieving job execution. As the Grid forms a distributed system, the resulting workflow encompasses not only the sequences directly responsible for launching the job, but also other central processes, such as data transfers, storage and package distribution.

5.1.2 *Rationale for a new solution*

The creation of a new solution must be seen in light of available technologies and options, and likewise be put up against the limitations and drawbacks found within current Grid deployments. Specifically, if there is great incentive to pursue the development of something new, as opposed to continuing with the Grid as it is. However, given the issues discussed in Section 4.4, as well as the beginning of Section 5.1, combined with the increased needs for reliable and performant computing resources in preparation for Run 3 of the LHC, the foreseen improvements are deemed crucial for future success. Furthermore, the new JAliEn middleware is also set to be put in production before Run 3: consequently, one way or another, the ALICE Grid is set to change in the near future – making this an optimal time for embracing a new, modernised, Grid workflow.

5.2 Building a new solution

A new solution should be able to replace the need for ad-hoc deployments of Cloud-related technologies on Grid sites. It should also be able to do this in a manner reducing issues, drawbacks and limitations. This section will investigate potential approaches for how this can be achieved.

5.2.1 *Containers*

Given large portions of the Grid stack are virtualised, containers are of particular interest for any new solution: having less complexity, their low overhead bodes well for both performance and reliability. Furthermore, their layered approach to storage reduces disk space requirements, in turn making them more portable and easier to scale. At the same time, their feature set largely overlaps with that of VMs. As mentioned in Section 1.3.4, containers may therefore possibly be integrated on all "levels" of a Grid framework:

- On worker nodes – providing Grid jobs with a tailored execution environment for their purpose.
- On central services and VO-Boxes – providing ready-to-go host configurations.
- In the middleware – automating the use and deployment of containers within the Grid.

While there is currently no native use of virtualisation within the AliEn middleware, the two former points would in this case be replacing the fragmented ad-hoc virtualisation approaches currently used by ALICE Grid sites.

It must be noted that natively incorporating containers within a Grid workflow raises questions on how to best approach this process. First and foremost, a decision is needed on what container *platform(s)* (toolkits) to support and use. At the time of project commencement, Docker is the de-facto container standard in terms of both deployments and container format, but is challenged within HPC, and increasingly in the Grid, by newcomer Singularity [65][67].

Docker

Docker is one of the more popular container platforms, having a rich API and a mature codebase. It is highly standardised and documented, with a centralised online hub for container images. Though initially designed to run enterprise microservices, it has since been adopted to serve a number of use-cases, including HPC and sites in the WLCG. A Docker instance will start with elevated privileges, and requires the presence of a background service running as root to function.

Singularity

At a glance, Singularity can be seen as a light-weight alternative to Docker, also providing a container framework, though with less overhead. There is likewise support for both the Docker image format and the Docker hub. However, Singularity was mainly designed for an HPC use-case scenario, and a number of tailored distinctions are present for that reason. Given its HPC background, it comes bundled with tighter integration to tools and batch systems such as Slurm . Furthermore, it is designed to not require any system, architectural or workflow changes to improve the integration with HPC: it supports numerous resource managers and multiple system architectures. There is also native support for access to GPU (Graphics Processing Unit) resources. To facilitate image distribution, single file container images are utilised. A dedicated online repository for Singularity images is also available.

Other platforms

Other container frameworks, including CoreOS Rkt[1], Podman[2] and Kata[3], were also considered. However, the two former remain in early stages of development, while the

[1] https://github.com/rkt/rkt
[2] https://podman.io
[3] https://katacontainers.io/

latter – Kata – base their containers on small virtual machines, a novel approach which still requires having an (albeit small) virtualised stack. Granted, Singularity is likewise a new framework, but its emphasis on HPC, which has a significant requirements overlap with Grid computing (see discussion below), makes it a compelling alternative nevertheless.

Comparison

The core distinctions between Singularity and Docker can be observed in how container isolation and user privileges are handled. While a Docker container starts with escalated privileges, a Singularity container will maintain the same user context as used outside the container. To gain root access, a container must be launched as root. This improves container isolation, which in turn may contribute to improving security (e.g. making breaking out of the container harder). Furthermore, file access is determined by a dedicated launch wrapper code (the file system is isolated, with the writeable areas being whitelisted), and user code can only interact with its own processes. Unlike when using tools such as *glexec*[4], there is no UID (User Identifier) switching required. Resource management is left to the batch-system.

Unlike more common enterprise-oriented tasks, which is the target of Docker, HPC tasks tend to use fewer, larger, containers carrying a given environment and all its required dependencies. As a result, Singularity is optimised for images that target a single application (often with complicated stacks or difficult dependencies), using the smallest number of namespaces possible. In the context of HPC, this gives it a lighter footprint and, potentially, better performance. Unlike Docker, it is also a single executable.

Concerns and decisions

The general goal for a containerised worker node within the WLCG would be to deploy a preconfigured environment, with its dependencies, for the execution of a job. In this aspect, the use-case closely resembles that found within HPC - one container, one environment, one application. Consequently, this use-case therefore also corresponds well with Singularity. In addition to a theoretical gain in performance over enterprise-oriented container platforms (such as Docker) in these scenarios, Singularity can also provide a number of equivalent features without requiring elevated privileges. There is no need for a background service operating as root, and commands such as create, import, export and mount are available to all users.

Given the above, it would be natural to focus on the integration and use of Singularity within the Grid. However, Singularity is a relatively new platform, and has not been tested in production as rigorously as Docker. Furthermore, being lightweight, Singularity has fewer features than Docker – especially in terms of networking and fine grained container isolation. That said, given Singularity provides a large subset of the functionality found within Docker, without requiring elevated privileges, the case for Singularity is compelling nonetheless: combined with its ability to run standalone,

[4]Used to associate certificate credentials to local Unix accounts [76].

this would greatly simplify the process of having it deployed on sites, as it would not require any additional actions on behalf of site admins.

Should Singularity nevertheless end up being too limited in its feature set, a possible solution could be to fork it – creating a new container platform specifically tailored to the WLCG. This platform could then in turn be used as a base for a new Grid framework.

5.2.2 Grid middleware

Containers alone do not make a new Grid framework: while promising in many aspects, containers will need to somehow be natively integrated in the distributed process of job execution. Otherwise, it would be no different from the current ad-hoc solutions. As briefly suggested in the previous section, one approach for achieving this would be to build a new solution from the base of Singularity, integrating all necessary tools/features within it, and deploying it at Grid sites.

A different approach to the above could be to harness the Grid middleware itself: containers being both lightweight and portable presents an opportunity to have these built-in as a feature. This approach is particularly interesting when considering a new middleware framework is in development: JAliEn. Being built from the ground up as a complete rewrite, yet with the core necessities for job execution already implemented, it presents a blank canvas for new features and enhancements. This would not only allow for the integration of containers in the workflow, but also open up for more core workflow changes from within the Grid framework itself, creating an opportune entrypoint for embracing and integrating new technologies and features. Specifically, the JAliEn middleware could be developed further to allow for:

- Isolating the executing job payload from the underlying host
  - Prevent potentially malicious executables from breaking/breaching the worker node. Better control the use of resources.
- Better integration of its own features
  - JAliEn comes with a new authentication scheme based around token-passing, which is not fully utilised.
- A streamlined, universal, deployment method
  - A one-click solution to allow for the adoption of the new middleware, independent of site.

– Where a key component in enabling the above features would be the integration of Cloud-related technologies directly within the middleware.

5.2.3 Handling containers

Using the Grid middleware to handle container deployment provides a way for naturally integrating containers within the execution process, while at the same time

removing the need for many of the current ad-hoc solutions. It would also allow both the middleware and the payload to be more aware of jobs running in their own isolated environments, and thus be able to act on this when needed (e.g. in case of a failure). However, the Grid is composed of many distributed components, and not everything is handled by the middleware.

VO-Boxes are an integral part of the ALICE Grid: providing an entrypoint for individual clusters. Being both complex and time-consuming to set up, having a VM template may assist in both configuring and deploying these. Furthermore, for AliEn, worker nodes are required to occasionally connect to the VO-Boxes, making it essential to keep them alive. Cloud solutions, such as VMs, may simplify the process of monitoring and restarting.

Within JAliEn, containers may be introduced as another step in the job execution process (as discussed in the previous subsection). However, this process would not include VO-Boxes, as these remain outside of the main execution chain. Unless the process of containerisation is extended beyond just job execution, VO-Boxes would thus be left as-is, spread across VMs and physical hosts depending on the site configuration. As mentioned earlier in Section 5.1, this is concerning for multiple reasons.

Having VO-Boxes containerised would provide many of the benefits of using VMs, in a smaller software stack, potentially avoiding issues such as those mentioned in 5.1. The containerisation would however have to come from outside JAliEn, as the VO-Box is managed by sysadmins – meaning that a new approach, outside the middleware itself, is needed for this purpose.

Section 5.2.2 suggests a one-click, universal, approach for deploying the new Grid middleware. While this would initially suggest that worker nodes may simply spin up a container for jobs, as opposed to being preconfigured in advance, this could also apply to the VO-Boxes: a one click, universal, container for quickly having all necessary site services up and running. The initial setup will need to be prepared by a site admin, but as opposed to setting up multiple services and configurations, it should only require one step. As VO-Boxes also include the Grid middleware CE service, this approach may also serve as a quick way for migrating to JAliEn – as a JAliEn CE will submit startup scripts for its own new JobAgents to the local batch queue.

5.2.4 *Distribution*

One question raised for the above approach, and the integration of containers within the Grid middleware in general, is how to provide the containerised environment: i.e. both the container runtime and any compatible image(s) would need to be made available to both the JAliEn middleware and site admins. While the container runtime, such as Singularity, could be preinstalled, the containers themselves would need to be updated occasionally. For job execution, the image could be pulled/propagated alongside jobs, while this would not apply to other container use-cases (such as VO-Boxes).

As mentioned in Section 4.1.2, the default mechanism for package management at project commencement is through CVMFS. However, being a distributed filesystem, it provides benefits beyond just package management. Binaries and files are read on-demand, and these may not necessarily be bound to a specific build or package. If a container image is converted to a directory, and spread onto CVMFS, a container would

be able to launch instantaneously, as individual files would be downloaded on-the-go as needed[5]. Images as a directory are natively supported by Singularity, and with all ALICE Grid sites having access to CVMFS, deploying an image here would make it easily available for all. Furthermore, the Singularity binary is capable of running in a standalone mode, meaning the runtime itself could be placed in CVMFS, and thereafter called by JAliEn, worker nodes or (admin) users as needed. In other words, CVMFS may provide simple distribution of both runtime and environments throughout the ALICE Grid, if used accordingly[6].

5.3 Related work

The process of establishing a new Grid middleware framework solution can be seen in light of the work described in Ref. [10], which covers many aspects of incorporating Cloud technologies within an ALICE Grid context – applying concepts such as virtualisation, multitenancy and elasticity, and technologies such as CernVM [10]. Much of this work was however confined to a specific T2 site. By building a new Grid workflow from the middleware as a base, this could potentially be ameliorated – with no need for a dedicated setup process, and quick adoption across all ALICE Grid sites.

Outside of ALICE

Other LHC collaborations are likewise investigating a tighter integration of Cloud-related technologies, such as containers, within their Grid workflows. From before, the ATLAS, CMS and LHCb collaborations already utilise VMs and Cloud resources either indirectly through local site configurations as ALICE, or directly through middleware interfaces (e.g. through *Harvester* on ATLAS PanDA (Section 12.2), and *vmdirac* on LHCb DIRAC [77]). More recently, there are also efforts to bring containers directly into the job execution process, across all of the aforementioned collaborations.

5.4 Drafting a new Grid workflow

Following the discussions on how to approach the building of a new Grid workflow in the previous Section 5.2, this section will proceed with creating a draft for a complete solution based on the insights gained.

A containerised JAliEn

A draft for a complete solution is shown in Figure 5.1. As can be seen in this figure, the JAliEn middleware is to be the very core of the new solution, with the workflow incorporated. However, core steps that may previously have gained benefit from VMs are now containerised, including the JobAgent and payload execution. This can be made possible through the new security model found within JAliEn, detailed further in Chapter 7.

[5] This idea is at the very core of the original CernVM project.

[6] As opposed to uploading everything contained within large image files, which would not benefit from the on-demand read found within CVMFS.

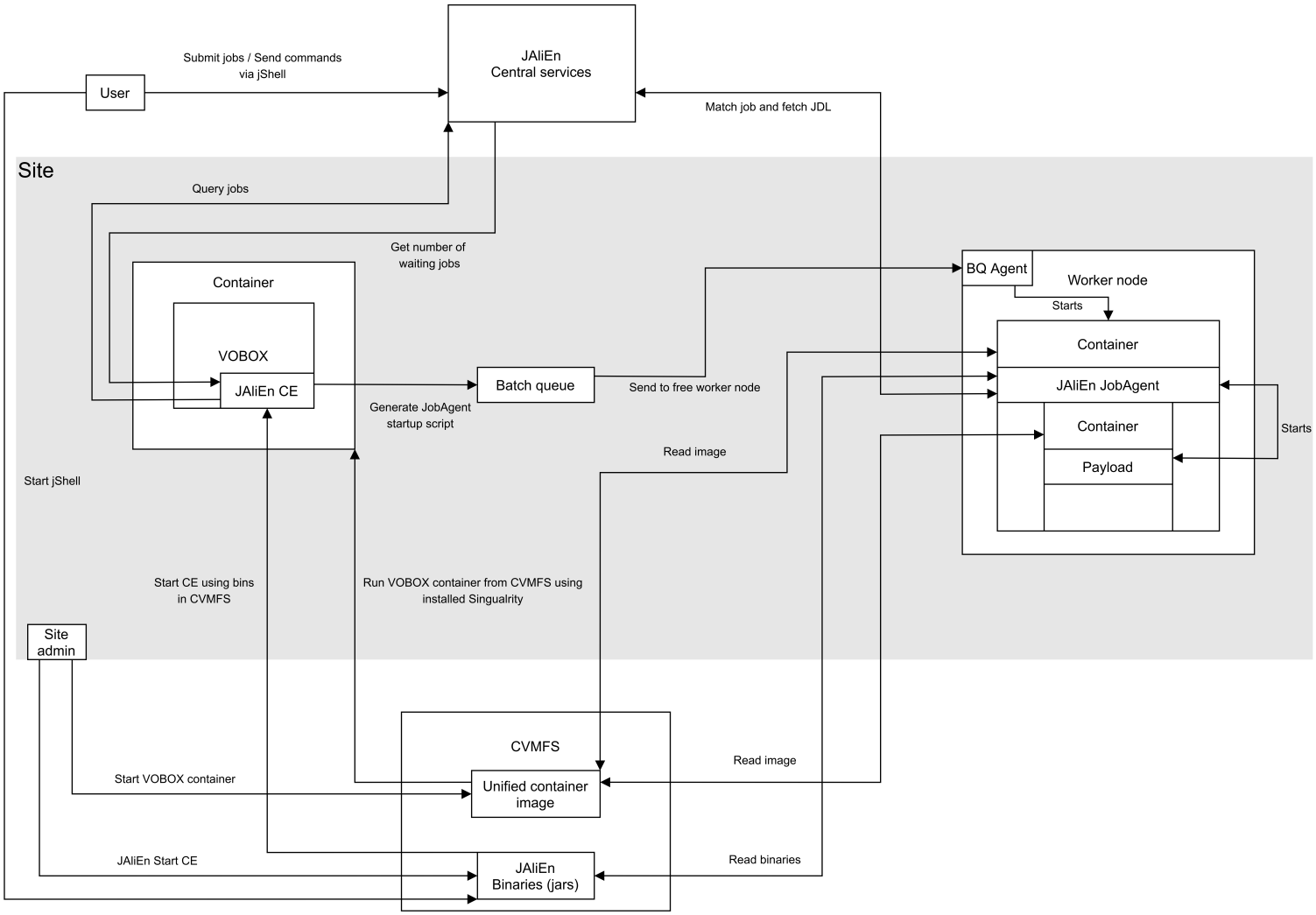


Fig. 5.1: A drafted new Grid workflow, using JAliEn and containers. Both JAliEn binaries and the container image are all taken from CVMFS. Some dependencies (such as Java) are provided within the container image, while Singularity is expected to be preinstalled on the hosts. This draft is in part inspired by the presentation on possible containerisation within JAliEn that is available from Ref. [78].

Furthermore, the use of JAliEn should likewise aid in term of scaling performance. As mentioned Section 2.6.3, the middleware contains several changes aimed at reducing previous key bottlenecks within AliEn, such as a new authentication scheme, a proper API and load-balancing features. As deployment and rollout takes places, the central services should consequently be able to handle the number of active jobs across sites, without unmanageable system load on the central machines – and more importantly, without impacting their ability to provide the necessary services. This will however also depend on the final implementation of the job pilot component, as excessive requests and calls may overturn the new scaling capabilities. The end result post-deployment will be further examined in Chapter 11.

One-click VO-Boxes

VO-Boxes need to be reliable. By having these migrated onto containers, the lack of a hypervisor should provide for more direct access to resources, in turn possibly reducing I/O delay and minimising system load, while at the same time making them both portable and easy to deploy. Furthermore, having a smaller deployment stack should also provide for less potential sources of error.

Deployment can further be simplified through container management tools. As opposed to having to meticulously configure each VO-Box, simply pulling and running a stored image should suffice. This could also speed up JAliEn adoption, as the new CE may already be included in the image.

Singularity

At the core of the push for containers is Singularity (known as Apptainer as of November 30, 2021[7]), which with its extra low overhead for HPC-like tasks and unprivileged (non-root) mode, makes it the best suitable candidate for use within the Grid. Furthermore, the developers are actively listening for feedback, and have been willing to include additional features considered useful for the Grid (See Section 10.1.2).

More reliance on CVMFS

CVMFS can be used as a single source for both binaries and images, and be tied together with an existing build system to automatically deploy new changes to runtime and containers across all sites. Furthermore, core system components may also be placed in CVMFS, just enough for JAliEn to start, which will thereafter set up an environment as preferred for jobs. This would make both the OS distribution and configuration of worker nodes moot, presuming a preconfigured container may be automatically launched by JAliEn with the necessary environment.

[7]On November 30, 2021, Singularity became Apptainer, with the first stable release made available at very end of this project. Consequently, development was mainly done using Singularity builds, with Apptainer only being deployed post initial rollout.

5.5 Remainder of Part II

The rest of Part II is organised as follows: Chapter 6 will investigate approaches for creating a new, containerised, VO-Box, akin to what has been described in this chapter. Chapter 7 examines the JAliEn middleware, its current state and needed improvements for a new Grid workflow. Based on the findings here, Chapter 8 will outline the development and changes involved, resulting in a new, containerised, JobAgent. Moving on to other core services, Chapter 9 will examine further changes, and needed steps towards deploying a new solution into production.

CHAPTER 6

CONTAINERS FOR VO-BOXES

VO-Boxes host essential site services, and should consequently be reliable, straightforward to set up and simple to maintain. VMs have contributed to achieving this, but not without caveats (Section 5.1). This chapter will examine how VO-Boxes may instead by placed in containers, consequently reducing the software stack and possibly reducing some of the current VM drawbacks and limitations. Afterwards, the end result will be put in a broader scope, examining how it may be further integrated with a possible new Grid workflow.

6.1 Anatomy of a VO-Box

The primary function of a VO-Box is to serve as an entrypoint for the ALICE VO at a site, allowing incoming jobs to be scheduled on the site worker nodes (as discussed in Section 2.2.2). The main service responsible for this is the batch system, while several ALICE job management processes are hosted on the VO-Box. The batch system chosen by the site generally provides a *submit* command, allowing a command or script to be submitted into the local batch queue, and in turn eventually be executed on an available worker. Scripts, containing the work to be executed on workers, are provided to VO-Boxes by the AliEn CE service, which is also hosted here – matching suitable jobs with the site, and in turn generating the script used to start new job agents. Thereafter calling the *submit* command provided by the batch system or by a gatekeeper service, translating Grid jobs into local batch system tasks.

In addition to the batch system and AliEn CE machinery, more components are required for a VO-Box. Authentication, which is handled through X.509 certificate proxies, must be provided (i.e. means to verify and renew), as well as means for privileged VO members to access the VO-Box itself, e.g. for maintenance or configuration changes. The latter is provided by GSISSH, a modified version of OpenSSH with support for credential forwarding [79]. Furthermore, a number of monitoring processes are also hosted on the VO-Box, some as part of MonALISA. There are also other, more common, tools running on VO-Boxes, with specific configurations, such as rsyslog and iptables.

As a whole, the following services are required for a VO-Box [80]:

- Batch system client (e.g. HTCondor, Slurm, TORQUE)
- AliEn Computing Element (CE) and related services, MonALISA

- GSISSH - for login access, if desired
- Proxy renewal service
- Cron (for periodically running tasks)
- Rsyslog

Or in other words, a VO-Box is not a single service, despite often being considered as one unit.

Within larger computing clusters, it is common to partition the available worker nodes into several smaller sites – allowing for more control on where to send jobs, and for which nodes. Consequently, this requires the use of multiple VO-Boxes. As opposed to having multiple physical hosts for each VO-Box, a sole host may also achieve this by having each VO-Box running as a VM. Within such a configuration, each VO-Box will have its own IP (Internet Protocol) and MAC (Media Access Control) address, and all routing specifics are handled by the hypervisor. This may however create problems if one wants to achieve the same through the use of containers.

Most common container frameworks, such as Singularity, are *application containers* (Section 3.4.1). As opposed to system containers, these are intended to run a single application/service, and thereafter terminate. Furthermore, networking is generally handled by bridging the connections as needed per application. This is close to the exact opposite to the needs of a VO-Box, which is composed of multiple services, and where several VO-Boxes may be running on the same host – meaning each service for each VO-Box must also be bridged without overlap.

6.2 Container challenges

Application containers are mainly intended as an execution wrapper for a single process, providing a suitable environment with all required dependencies satisfied. As seen in the previous section, this in turn creates two challenges for using such containers to run VO-Boxes:

- Process handling
  - Application containers are tied to the execution of a specific process. Once that process finishes, the container will terminate alongside it. This makes service management within the container limited, while also making the container prone to exiting if the "main" process should crash, in turn also terminating all other processes.
- Networking
  - Relying on the default bridging approach used by application containers requires having connections forwarded between the host and the container for each process. Not only would setting this up require more steps compared to a VM (and thereby going against the goal of the RQ), but also create challenges in setting up multiple VO-Box containers on the same host.

Furthermore, as will be seen in Section 6.5, other challenges are also present, which are not immediately apparent.

Granted, the challenges above may all be attributed to the use of application containers, whereas system containers would behave more like VMs, as is already common in the Grid. Targeting a framework intended for system containers, such as LXC, would potentially avoid these. However, many of the benefits associated with containers would also be lost – and in particular the lightweight portability.

The forthcoming sections will attempt to resolve the challenges mentioned above, in turn enabling the use of application containers for VO-Box use.

6.3 Handling networking

Before attempting to run VO-Box services in a container, a functioning approach to networking is required. By default, this is handled through bridging in application containers such as Singularity and Docker. Here, containers communicate to the outside world through an intermediary bridge interface provided by the container framework, which forwards packets from and to the host network interface and between containers. It also maps internal ports of service instances running inside the containers onto network facing ports on the host. While this is a largely straightforward approach when having single process containers, and when each container is responsible for a different service, it becomes less transparent when applied to VO-Boxes. With several running processes in each container, and the processes being identical between them, ports need to be remapped to avoid overlap on the host – and with multiple containers all with multiple identical services, this will need to be done for a large set of processes, which in turn can be hard to track. Furthermore, site configurations for each VO-Box are handled through a central LDAP service, which identifies individual VO-Boxes through their IP addresses and hostnames. Using the default bridging approach, each VO-Box container would have the same IP/hostname combination as for the host, making them indistinguishable from each other.

6.3.1 *MACVLAN*

A more suitable approach for networking VO-Box containers could come from *MACVLAN*: a feature that allows for the creation of several virtual network interfaces, on top of an existing physical interface [81]. Each of the virtual interfaces may have a unique MAC address, and its own IP address. This is illustrated in Figure 6.1.

When applied to containers, MACVLAN provides for convenient networking – allowing each container a unique MAC address and IP, and in turn allowing it to appear as a conventional host/VM on the network. This would allow VO-Box containers to act as drop-in replacements for existing solutions, without any changes to existing network configurations at sites. As of Linux 3.9, the feature is available as a production (non-experimental) feature directly in the kernel. However, this is not a guarantee that support is available within all container frameworks.

Section 5.2.1 raises concerns that despite the advantages of Singularity, it remains limited in features, especially in regards to networking. This becomes particularly evident when hoping to utilise connectivity beyond what is provided through the

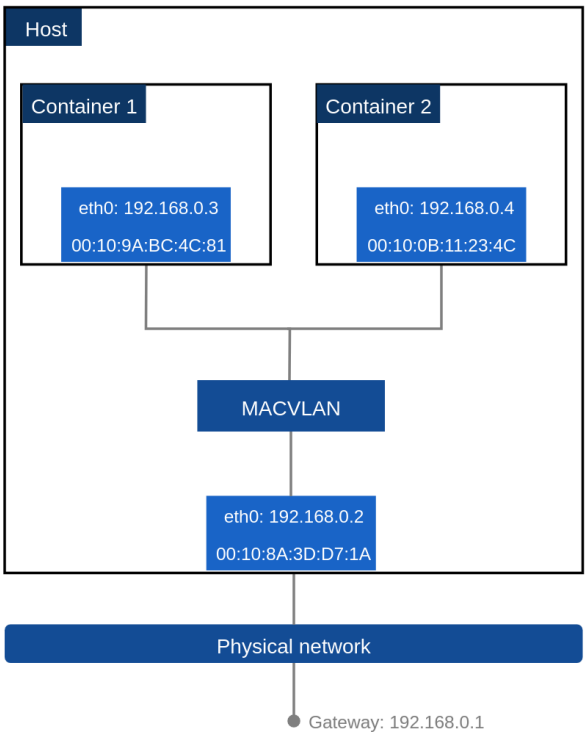


Fig. 6.1: Overview of the MACVLAN architecture for a network of two containers. Each container is assigned a unique virtual interface, on top of the physical host interface – each with its own MAC and IP address [82].

default bridging. In other words, there is no direct support for MACVLAN/IPVLAN[1]. However, the lack of direct support is not a showstopper by itself – the feature is natively available in the kernel, and workarounds exist. By providing Singularity with the `-net` flag, a new container will in that case be launched in a new network namespace. Once running, it is possible to manually create a new MACVLAN interface, set up routing, and attach the newly launched container to it. Unfortunately, this approach comes with major drawbacks: it requires identifying the namespace ID of the container, which changes when relaunched, and manually processing a number of commands requiring root privileges. This procedure will also need to be repeated every time the container is restarted. Granted, this could be automated through scripting, but arguments may be made if this would be a worthwhile replacement for the current VMs and ad-hoc scripts present on ALICE Grid sites.

6.3.2 *From Singularity to Docker for VO-Boxes*

When re-examining the benefits of Singularity, it becomes evident that many of its selling points are mainly relevant for *job execution*, and not necessarily for the hosting of services as within VO-Boxes. Consequently, it makes sense to turn instead to a container framework more suited for this purpose. While several alternatives exist (Section 5.2.1), Docker remains the more mature option for its developed feature-set. Specifically, it has a comprehensive networking stack, with support for a number of networking

[1]Similar to MACVLAN, albeit one layer up, IPVLAN allows assigning multiple IPs to a single interface. The MAC address, however, remains the same [83].

configurations, including built-in support for both MACVLAN and IPVLAN. While it is not as portable as Singularity, still requiring a background service to function, as well as a proper admin-approved installation on the host, these drawbacks can be considered acceptable for the use-case: a VO-Box is expected to continuously operate for months without restart, and is generally managed by a site administrator. With this in mind, the target container framework is thus shifted from Singularity over to Docker in the context of VO-Boxes – but remains the same for job execution and JAliEn (Chapter 8).

6.3.3 *MACVLAN on Docker*

In contrast to the limited networking options of Singularity, Docker allows for defining alternative networking configurations to the default bridge – including MACVLAN. Once a MACVLAN bridge interface has been created (by providing Docker with the host network subnet and gateway), new containers may be attached to it through the `-net` flag at launch. IPs and MAC addresses can be automatically assigned from a predefined subnet pool, or manually given to the container. In the latter case, this allows a container to be a direct drop-in replacement for a VM VO-Box, assuming its previous IP and MAC addresses, without needing any changes to the back-end network.

Containers are not completely identical to VMs in a network

A container connected to the local network through a Docker MACVLAN will largely appear as a conventional VM/host, having its own IP and MAC, but not completely. There is no support for DHCP (Dynamic Host Configuration Protocol), with IPs instead being assigned either by Docker or by hand. Furthermore, certain networking configurations have to be handled from within the configuration of Docker, even if this was earlier possible from within a VM or a physical host – including the DNS (Domain Name System) configuration. The latter is a potential pitfall, as settings from earlier VMs reapplied in a container will simply not persist, in turn causing issues in name resolution.

6.4 Handling startup and service management

Another key difference between VMs and application containers, such as Singularity and Docker, is the lack of an *init system* – the core component responsible for launching required processes at system boot, and managing their lifecycle[2]. As application containers are intended for running a single process, mainly providing an environment wrapper, the container is generally disposed of after its main process terminates. Indeed, an argument can be made that if a user ever has the need for more than once process in each container, then that user has misunderstood the core concept of containerisation.

[2]External tools, such as Docker-compose (`https://docs.docker.com/compose/`) and Kubernetes (`https://kubernetes.io/`), exist – though these are aimed at managing the lifecycle of multiple containers, and not the services within each.

Despite the argument above, use-cases for containers running multiple services exist, with Docker even acknowledging the existence of tools that may be used for this purpose, such as SupervisorD [84][85]. In the context of VO-Boxes, it must be emphasised that in a way, the container acts as one process, despite being composed of multiple processes. Many of these are tightly integrated, and some are core system components, which makes spreading them onto multiple containers convoluted. In other words, the VO-Box represents a use-case where it makes sense to have one container for multiple processes.

6.4.1 *Making an init system*

Docker containers may be preconfigured through a *Dockerfile*, a recipe containing all files to be copied, configurations to be set, and commands to be run after downloading a specified base container image. This will in turn create a new, ready to be deployed, container image. Dockerfiles also allow for specifying an *entrypoint*, which is the initial process to be run when the container is launched. This may be any executable, including a shell script, through which additional processes may again be launched.

The above approach presents a method for having multiple running processes when a container starts – or in other words, a very simple custom init system. However, it still lacks features pertaining to lifecycle management, such as automatic restarts and log handling.

6.4.2 *A piece of SysVinit and SystemD for containers*

An observation can be made that SysVinit, the init system within CentOS 6[3], has many similarities to the simple init system described in the previous subsection: it simply starts a selection of management scripts at launch. However, each of these management scripts is tailor-made to handle the lifecycle of a specific application/service. One approach for achieving the same result in containers can thus be to copy over these management scripts through a Dockerfile, and have a simple init script start them at launch. Furthermore, a function can be made to make a similar syntax available within the containers, e.g. enabling the use of "`service start|stop`", as expected from SysVinit. An example of such a function can be found in the top half of Figure 6.2. In essence, this creates a simple SysVinit clone for containers, which will be indistinguishable from a VM/physical machine by users.

While a successful deployment was made of such a SysVinit clone [82], it was subsequently dropped because of system changes: CentOS 7, as well as a majority of Linux distributions, have now abandoned Upstart/SysVinit in favour of SystemD [87]. This init system is more tightly integrated with both the system and the kernel, and does not rely on service scripts for management. This would not only mean that all service scripts used within the SysVinit clone would need to be manually maintained, but also that scripts and services relying on SystemD calls, if used within VM/physical VO-Boxes, would not be compatible with containerised VO-Boxes. A different approach is needed.

[3]It must be noted that unlike prior CentOS versions, CentOS 6 uses Upstart in reality, but with SysVinit-like scripts [86].

```
#Function to give users sysvinit-like call for service scripts
RUN echo 'service () { /services/"$@"; }' >> /etc/bashrc

----------------------------------------------------------------

#Function to give users/scripts systemd-like call for supervisor
RUN echo 'systemctl () { supervisorctl "$@"; }' >> /etc/bashrc
```

Fig. 6.2: By wrapping service scripts (upper half) and SupervisorD (lower half) in a function that mimics the syntax of proper init systems (such as SysVinit and SystemD), existing scripts and applications can be reused within the containers without changes.

Section 6.4 mentions SupervisorD as a tool for managing multiple processes within containers. While originally not intended to be an init replacement, it provides many of the desired features, such as automatic startup, lifecycle management and logging options. As opposed to re-using existing service scripts and configurations, as was the case for the SysVinit clone, each process to be managed by it requires a separate service entry in its configuration – one of the reasons why it was initially not considered. However, once set up, managing startup entries in a single configuration file should be far less prone to requiring changes/maintenance, compared to having to maintain multiple startup scripts as with the SysVinit approach.

Given the above, SupervisorD has thus been chosen as the method for managing services within VO-Box containers[4]. Granted, as most CentOS 7 applications now expect the presence of SystemD for both management and log offloading, all can be configured to run in a standalone mode, and with logs again written to file, allowing for SupervisorD to take on the management role. Furthermore, as with the SysVinit clone, a function can be used (Figure 6.2 (bottom)) to provide a SystemD like syntax (`systemctl start|stop`), to allow cross compatibility for scripts expecting its presence on the system. This should in turn make containerised VO-Boxes indistinguishable from their VM/physical counterparts from the perspective of a user.

6.5 Other challenges

With the transition from VMs to containers, challenges related to both process handling and networking were expected – the two areas that largely differ within containers (Section 6.2). However, other challenges were also encountered in the process of containerising VO-Boxes, that in turn need special configurations to overcome.

6.5.1 CVMFS

Modern VO-Boxes need CVMFS to function. This is a result of it not only being the default method for fetching required software and packages, but also because it hosts

[4]Dockerfile: https://gitlab.cern.ch/jalien/dockervobox/-/blob/master/Dockerfile

configuration files and essential tools (scripts) used by other applications. Furthermore, newer software, such as JAliEn, requires the presence of CVMFS in order to be able to function at all, independently of how it is installed (as will be discussed in Section 9.1.1). Consequently, all VO-Boxes must have a working CVMFS installation, providing the `/cvmfs` directory.

VMs provide CVMFS in the same manner as physical hosts: by installing the CVMFS package, and maintaining a separate CVMFS cache for each VM. This approach would however not be optimal for containers, as giving each container its own cache would make them far less portable, and possibly duplicate large amounts of data. As containers allow for the bind-mounting of directories, a better approach can thus be to solely install CVMFS on the host, and thereafter bind-mount `/cvmfs` within each container. This would remove the need for installing CVMFS in each individual container, as well as avoid duplicate CVMFS caches on the same physical host.

Bind-mounting is a convenient approach for gaining access to CVMFS across containers, but it is not perfect. Due to a bug in *autofs* – the service responsible for automatically mounting `/cvmfs` repositories on-demand – containers may at times be unable to access the CVMFS instance on the host[5]. Instead, the error message "Too many symbolic links" is shown. The bug was traced to being triggered only when containers were launched before the needed CVMFS repositories were mounted on the host. In other words, simply accessing the ALICE CVMFS repository with a `cd` before launching the containers would avoid this issue – or disabling autofs alltogether, instead using a script or system service (e.g. SystemD), to automatically mount the needed repositories at boot time.

6.5.2 *File descriptors*

File descriptors are a construct used within Unix-like systems to identify file and socket resources, used in the interface between user and kernel space. In other words, all open files are referred to by the kernel through file descriptors. Within VMs, each machine is given its own kernel on top of the virtual hardware stack. This is however not the case with containerisation, as the kernel is shared across all containers running on the same physical host. Consequently, this may lead to behaviour not seen in VMs for VO-Boxes.

A VO-Box has several services and responsibilities, and in the case of AliEn, also monitors the executing JobAgents across the site worker nodes (WNs). As a result, each VO-Box instance needs to keep track of a large number of files. For a conventional VM or physical VO-Box, this is not an issue. For containers, however, the shared kernel between the VO-Box instances will quickly cause the open file descriptor limit to be reached, causing each container to silently fail. When having more than one containerised VO-Box instance per kernel, the default limit for the maximum open file descriptors must thus be raised on the host using `ulimit` (for the soft limit), or by editing `/etc/security/limits.conf` (soft and hard limits).

[5] https://github.com/apptainer/singularity/issues/347

6.6 Adapting to Docker

In section 6.3.2, Singularity was dropped in favour of Docker for VO-Boxes. While the more comprehensive feature-set and network stack make it more suitable for running VO-Boxes, it also adds more complexity. Where Singularity can be seen as a simple namespace wrapper, Docker also comes with more mechanisms for container isolation, runs through a persistent background process, and by default uses a copy-on-write layered filesystem. In order to best utilise this container framework, several considerations and adjustments were required.

6.6.1 *Kernel access*

By default, Docker limits the access privileges containerised processes have to the kernel. This is a safeguard that is put in place in order to prevent rogue processes/users from potentially breaking out of the container, and gaining access to the underlying host – possibly as root, as this is a requirement for the Docker runtime. As a preventive measure, all access to kernel capabilities[6] is therefore disabled, unless explicitly permitted by the user on container startup. Unfortunately, this creates problems for multiple networking tools commonly used within VO-Boxes, such as `netstat` and `dnsmasq`, which simply refuse to function, even when executed as root. In an attempt to work around this issue, individual capability units were given to new containers, such as `CAP_NET_ADMIN`, but as more tools were used within the containerised VO-Boxes, more such capabilities had to be given. In the end, a decision was made to run all VO-Box containers as "privileged", without any kernel isolation. While this may present means for possibly breaking container isolation, the involved risk was considered acceptable given the benefits – as these containerised VO-Boxes would only be handled by site administrators. Furthermore, Singularity, which was the tool originally suggested for the containerised VO-Boxes, does not provide any kernel isolation at all.

6.6.2 *Keeping containers alive*

One of the main benefits of Singularity over Docker is its lack of background services and other processes, simply existing as a namespace wrapper for a given process. This is in stark contrast to Docker, where each container depends on being able to communicate with the background `dockerd`. Should this process terminate, all executing containers will be forced to terminate alongside it. Not only presents this a single point of failure for all containers on a host, but also challenges for reloading configurations and applying updates.

Starting with Docker 1.12, a feature known as "live restore" is available, allowing containers to avoid being killed if `dockerd` should terminate [88]. While not perfect, it allows containers to keep running independently for short durations, giving any terminated background service time to recover, or sysadmins time to update the system. It will however not work for larger Docker updates, as the containers will be unable to reconnect to the service due to the larger changes. Furthermore, containers can

[6] Linux divides the kernel access associated with superuser actions into distinct units, known as *capabilities*.

only stay alive independently for a few days, as they depend on the `dockerd` service for, among other things, offloading their logs. If absent for too long, the container log-buffers will eventually overflow and crash the containers.

The live restore feature remains far from the serviceless flexibility of Singularity, but it nevertheless aids in adding more redundancy to the system. While it remains an optional feature within Docker, it has thus been enabled for the containerised VO-Box hosts.

6.6.3 *Flattening images*

Docker uses a layered copy-on-write filesystem for its images, where all changes committed to a container image are written on a separate layer, without changing the base image itself. This allows multiple containers to reuse the same images, storing only their own changes in a separate layer, and consequently saving storage by avoiding unnecessary duplication of data – one of the very reasons why containers tend to have a lower storage footprint, and in turn increasing their portability. In other words, whenever additional changes are required for a container, these changes are stored as a separate layer on top of the existing container image.

Consecutive changes are also each stored in their own layer, stacked on top of any prior adjustments applied to a container. However, every additional layer applied on top of a container image introduces a slight I/O delay. While the delay caused by a single layer is largely negligible, as more layers are stacked on top of each other, it will gradually start to visibly impact performance (more on this in Section 6.7). As the VO-Box containers use images where changes remain frequent due to testing and development, this becomes an issue.

To avoid having to navigate several layers, in turn causing I/O delays, a solution is to *flatten* the container images once changes have stabilised. This is a feature available in Docker, where all layers are collapsed into a single-layer base image. It is a manual process, triggered when exporting images onto archives for sharing. Consequently, to avoid any possible overhead and performance penalties, all VO-Box images have been flattened by exporting and re-importing their images before deployment.

6.7 Viability of containerised VO-Boxes

To test the viability of having containerised VO-Boxes, the findings described within this chapter were used to deploy multiple instances in production for ALICE in July 2017[82]. These were monitored for prolonged durations, and with different configurations – testing different approaches to network, init and storage. The overall experience from these tests can be described as positive, with only minor caveats.

Figure 6.3 and Figure 6.4 depict two instances of containerised VO-Boxes being used in production, in comparison to a VM counterpart for the same time interval, for both the number of concurrent jobs[7] and the underlying load on the system[8]. Despite the container VO-Box being configured to manage more jobs (Figure 6.3),

[7] As AliEn JobAgents need constant communication with the VO-box, issues encountered by it can be reflected in the job throughput.

[8] The Unix load average can give an indication of overhead and current I/O load.

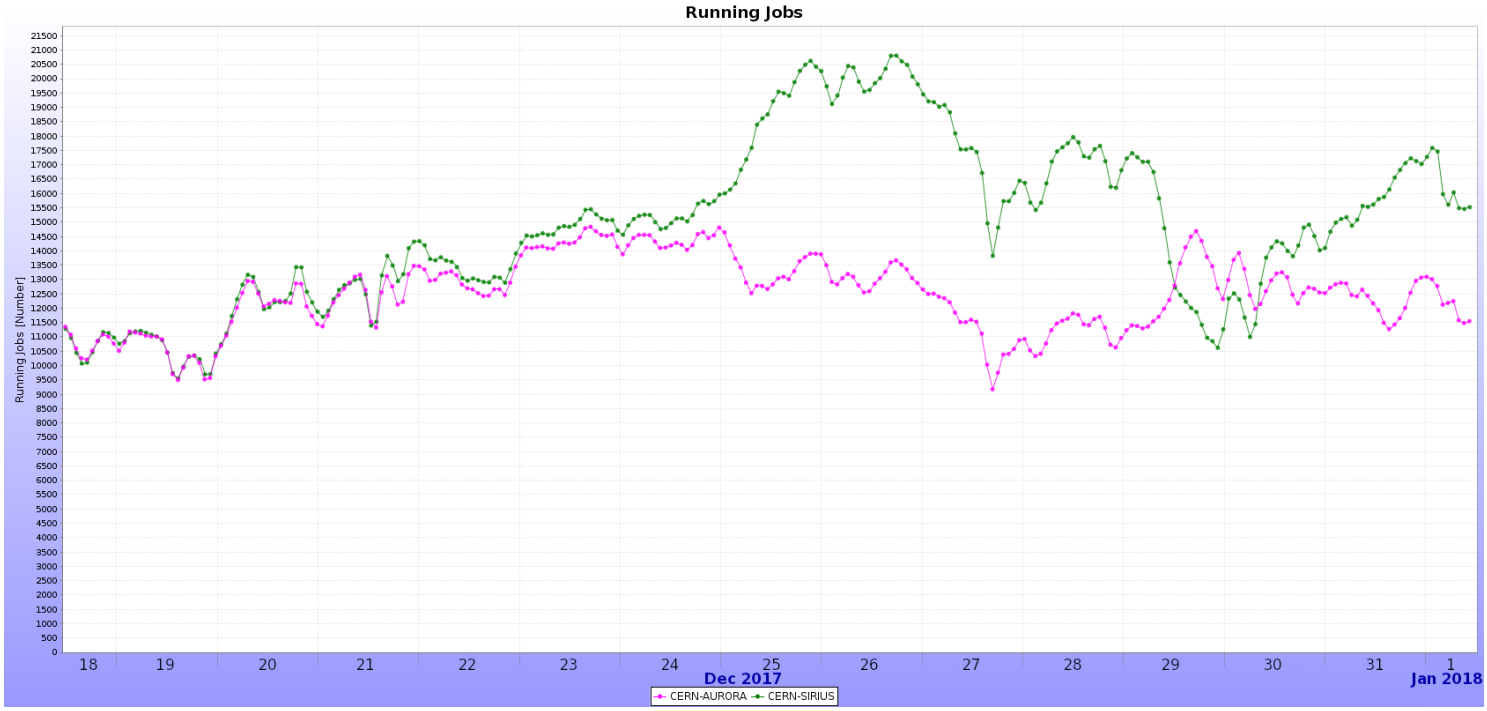


Fig. 6.3: Number of running production jobs for a containerised VO-Box (green), and a traditional VM VO-Box (magenta) over a two week period[82].

Fig. 6.4: Unix load average on the host for the containerised VO-Box (green), and traditional VM VO-Box (magenta), from Figure 6.3 [82].

without any apparent issues, its host can also be seen to have less system load than that of the VM (6.4) – despite running on older hardware (one Intel generation behind). Furthermore, initial experiences from using the container VO-Boxes also suggest them being less prone to faults and connectivity issues – something which has plagued their VM counterparts. To further confirm this observation, a VM VO-Box known for underperforming and having occasional network freezes ("CERN-CORONA") was converted and redeployed as a container in November of the same year[9]. With the reduced deployment stack provided by containers, it was theorised these issues could be avoided. Post deployment, no further network freezes were observed, and an immediate gain in the number of concurrent jobs was seen (Figure 6.5).

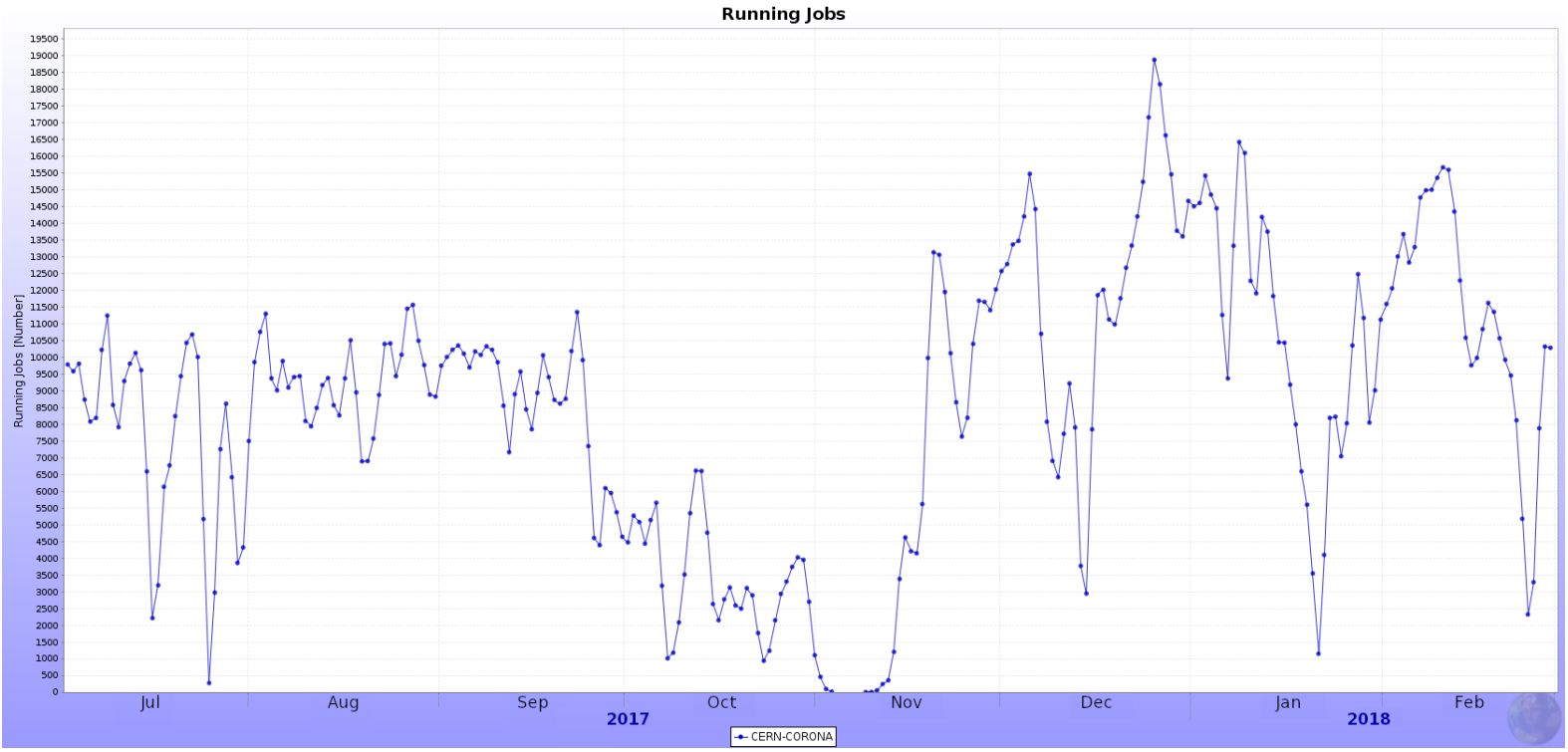


Fig. 6.5: Running production jobs for the VO-Box "CERN-CORONA", before and after its conversion from VM to container (early November 2017) [82].

The deployment of containerised VO-Boxes in production has, however, not been a flawless process. Many of the key points presented in Section 6.5 and Section 6.6 were only discovered after encountering them in production, such as the increased I/O latency of layered images[10]. Nevertheless, once properly set up and configured, containerised VO-Boxes have achieved good results in terms of both performance and stability throughout an initial one-year testing period, and the remaining VO-Box VMs at CERN have all been replaced with containers since. With these positive results in mind, the next step is to move beyond the VO-Boxes, towards modernising the job execution flow.

[9]Despite debugging attempts, the exact cause for these original issues was never found.
[10]Flattened images were already used for the above figures.

CHAPTER 7

THE STATE OF JALIEN

The JAliEn middleware is in an interesting position, with core functionality largely implemented, while at the same time being open to better utilising new features and technologies. In many ways, it can be considered a "blank canvas" for the integration of new functionality. This creates an opportunity not only for better utilisation of new features within JAliEn, but also for integrating it with new technologies – making it a core component for how a new Grid workflow may be organised (as in Section 5.4). This chapter will thus examine the JAliEn middleware in more detail, outlining its current status, and steps needed and taken in order to bring it in line with the discussion in Chapter 5.

7.1 A closer look at the JAliEn architecture

The layout of the JAliEn service model in many ways resembles that of AliEn (discussed in Section 2.6.1) with the same core actors:

- User
- Central services (CS)
- Computing Element (CE)
- Batch queue (BQ)
- Computing node (job agent (JA))

Out of these five actors, three are directly a part (component) of the JAliEn middleware (CS, CE, JA), while interfaces are provided for interacting with the other two (User, BQ) – again, in many ways similar to AliEn. The general idea for this approach has been to allow JAliEn to be a drop-in replacement for AliEn, while also allowing JAliEn to function in tandem with the legacy solution, when the latter is being phased out.

The right-side box of Figure 7.1 ("Future interaction model") depicts the initial plan for JAliEn and its interaction model between actors, with a comparison to the original AliEn to the left ("Current interaction model"). While largely similar at a glance, it can also be seen to introduce a set of new components as mentioned in 2.6.3, such as the JCentral and JBox.

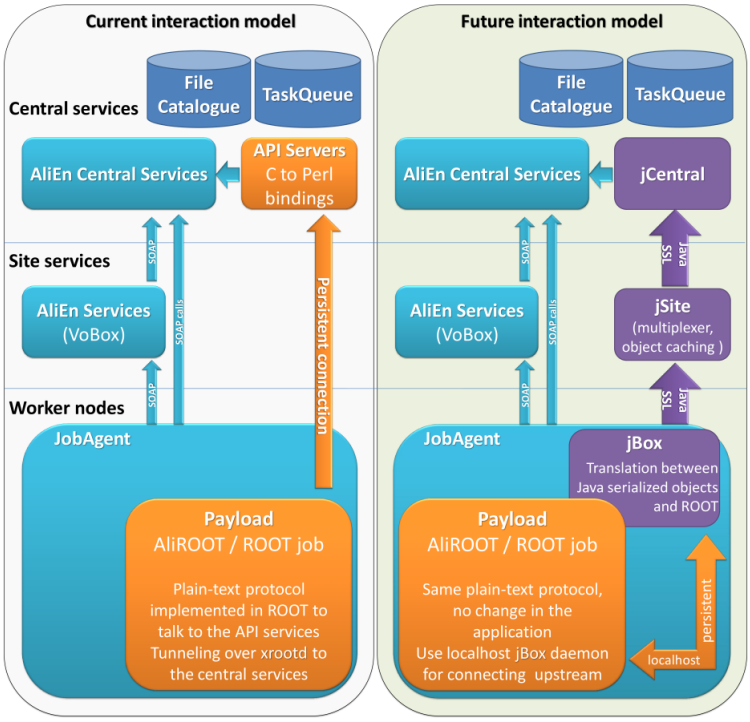


Fig. 7.1: Initial JAliEn architecture (right), in comparison with the existing AliEn (left) [21].

The JCentral is the authoritative source for all upstream API calls, and correspondingly handles any and all incoming requests: doing lookups in the file catalogue, database and/or taskqueue as needed. This differs from AliEn, which is split into different services for each purpose. AliEn services have each their own individual TCP (Transmission Control Protocol) port, with their own security constraints and back-end technologies, whereas the JCentral listens to a single TCP port, together with TLS (Transport Layer Security) for authentication.

The JBox is the main authentication handler within JAliEn, with a running instance present all around where upstream/downstream communication is needed: be it the central services, a VO-Box CE, computing node JobAgent or any connecting user interfacing with the shell. The authentication itself is very different from JAliEn – as opposed to the use of proxy certificates, full X.509 certificates are used instead (further discussed in Section 7.2).

A closer look at Figure 7.1 also reveals another component, which has not previously been discussed: the JSite. This was intended to be a connection multiplexer, in order to avoid having too many concurrent connections coming from site nodes. The nodes would instead connect to the JSite, which would thereafter forward the requests upstream through a single connection. However, better load balancing is another new feature within JAliEn (Section 2.6.3), and the number of JCentral instances may be scaled up or down depending on load – which efficiently handled the load during early testing. Consequently, the JSite has since been dropped from JAliEn (after the initial creation of the earlier figure), as the JCentral should be sufficiently capable of enduring high loads even without it.

7.2 Security within JAliEn

As mentioned in the section above, JAliEn forgoes using proxy certificates for common X.509 certificates. This is done through a novel approach: these certificates have been implemented as *token certificates* [89]. Each token certificate is a time-limited X.509 certificate used to represent different roles within the Grid, such as the user, a job or JobAgent. The roles are provided within the DN field of each token, signed by an internal certificate authority (CA).

Each Grid role comes with a different set of privileges and limitations – just enough for what is required by each role. Specifically, there are three roles (a.k.a. *identities*):

- User: Represents a Grid user, and comes with the same rights as defined by their Grid identity certificate – but only valid for a much shorter time, e.g. 48 hours.
- Job: Represents a Grid job, with privileges as needed for job execution, such as being able to fetch/upload files as needed by the executable.
- JobAgent: Represents the JobAgent process that manages Grid job execution on computing nodes, with permissions for fetching new jobs, updating traces and monitoring.

Additionally, restrictions are placed on which identities may request what tokens. Specifically, a job token may only be requested by a JobAgent, while a JobAgent token may only be requested by a CE – as the CE is responsible for generating the JobAgent startup scripts, as mentioned in Section 2.6.1. The CE, in turn, uses a host token, which is largely similar to a user token, albeit with a default validity of 12 months. Users may obtain a corresponding user token by presenting JAliEn with a valid Grid certificate.

7.3 Initial state of JAliEn

At the time of project commencement, JAliEn was not being actively used within ALICE. Its development was nevertheless advanced, and picking up pace with the approach of Run 3 of the LHC. In practice, it contained roughly the functionality needed to execute Grid jobs, albeit with a number of caveats.

First, the solution as a whole was largely untested, with new components being developed at different stages, and not necessarily fully intercompatible. Also, the provided functionality only allowed for the minimum required to run an executable, with limited file handling and upload support, and with some conditions being unhandled (more on this in Section 8.4). Furthermore, it also relied on the old AliEn central services to handle specific calls, such as unimplemented calls in the JAliEn user shell (JShell), and also for the *job optimiser* – a component used to transition jobs inserted into the central queue to a state where they can be picked up and executed by JobAgents at site computing nodes.

7.3.1 *Confirming state*

As mentioned above, JAliEn should be in a state where it is capable of running a payload executable. Specifically, it should be possible to pull a job from the central task

queue (which is implemented using a database, shared between AliEn and JAliEn), and have that executable run and in some form upload the results. This was ascertained by the original JAliEn developers, as well as through inspection of the code[1]. There were however several caveats to this functionality.

Some services can be shared between JAliEn and AliEn, such as the optimiser, database backends and shell (jobs submitted through either the AliEn shell and JShell are indistinguishable). This sharing of services was a requirement for the above JAliEn functionality to work, as there was no JAliEn CE running on any site (at the time of project commencement), no optimiser to transition inserted jobs in the taskqueue and there were limits to the features provided in the JShell.

Nevertheless, to confirm that the already implemented code functions as intended, a test was made to verify the functionality on the Grid. As there were no JAliEn sites, and consequently no CEs to start JAliEn JobAgents on worker nodes, this had to be done through a special approach: by *piggybacking* on top of AliEn jobs, using the existing infrastructure. As opposed to a production payload, a JDL would be created pointing to a startup script for a JAliEn JobAgent – similar to the startup scripts generated by CEs, which are afterwards inserted into the local batch queue. This was possible from the fact that JAliEn JobAgents communicate directly with JCentral (for which test instances were already available, as mentioned earlier in Section 4.1.1), and do not need to ping a local site CE. Instead, JobAgents may assume the identity of any site, if so instructed in the startup script. The only requirement is a valid JobAgent token, which was generated in advance for this purpose.

Figure 7.2 depicts the testing approach described above, where an AliEn JobAgent will consequently launch a JAliEn JobAgent, which in turns takes on the identity of a separate JAliEn site, and pulls a queued job for that site from JCentral. For this test, the corresponding results were a success, as jobs were able to be both executed and successfully saved. However, while this may confirm that a subset of features expected within JAliEn function as intended, the solution remains far from complete – as will be the topic of the following section.

7.3.2 *Initial limitations*

While Section 7.3 confirms the ability of JAliEn to execute Grid jobs, several limitations may also be drawn from it. Specifically, the following three limitations were seen during testing:

- *Computing Elements* – As development of JAliEn has progressed, the CE code needs to be updated to reflect the changes, with consequently no CE instances or sites being available for testing, as discussed in the previous section.
- *Shell features* – The JAliEn shell is not fully featured, with file creation having to be done from within the legacy AliEn.
- *File upload/download* – File handling is limited, with not all options being recognised for the download of input files, and conversely for the upload of output files.

[1] https://gitlab.cern.ch/jalien/jalien

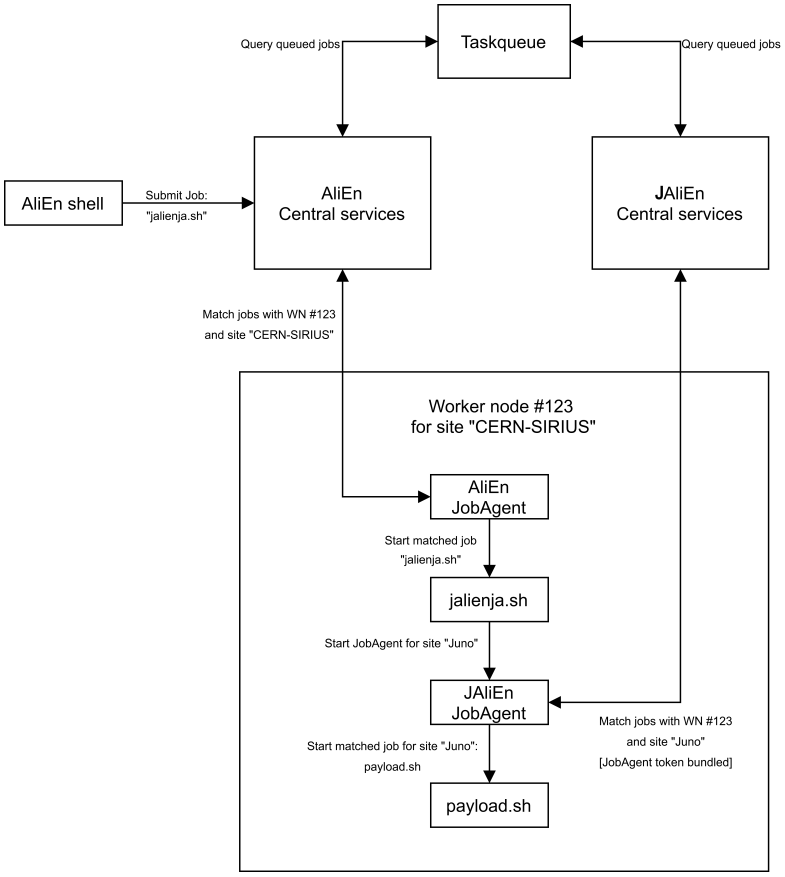


Fig. 7.2: Using an AliEn JobAgent to launch a JAliEn JobAgent as payload.

Furthermore, compared to the original goals/features of JAliEn (from Section 2.6.3), a number of additional observations may also be made:

- *Database* – The improved JAliEn database backends were intended to run on Scylla [41] – an alternative implementation of Apache Cassandra [40] aimed at low-latency and high throughput, in turn set to provide sizeable performance benefits. However, this transition has yet to be made.
- *Distribution* – For the test in the previous section, JAliEn (its JobAgent) had to be transferred as an inputfile, as there is no fully automated way of distributing JAliEn components.
- *Authentication* – Authentication handling is implemented, but remains liberal for the permissions of each role. Furthermore, the new security model based around tokens is partially used to *emulate* the legacy implementation within AliEn.

While not all of the above points are critical, some will nevertheless become central when folding JAliEn into a new Grid workflow, as will be seen later.

7.4 JAliEn for a new Grid workflow

Section 5.4 envisions JAliEn at the core of how a new, containerised, Grid workflow may function. However, given the limitations in the above sections, adopting the existing

JAliEn into this vision may have more benefits. A number of the issues mentioned may be directly derived from the current state of the JAliEn JobAgent, the CE (which is closely tied to the JA), and the distribution mechanisms – all of which are set to change for the draft in Section 5.4. This provides an excellent entrypoint for any changes, as the creation of a new Grid workflow may not only have its own merits, but also possibly help improve the JAliEn Grid middleware itself, readying it for the start of Run 3.

A key change proposed in Section 5.4 is the introduction of a containerised JobAgent, allowing jobs to automatically run in a suitable preconfigured environment, without the need for manual setup. However, containerisation is also a powerful mechanism for *isolation*, preventing malicious code inside the containerised wrapper from potentially harming the host system or other users on it. This may in turn provide an ample starting point for integrating JAliEn into a new workflow.

Container isolation may be used to overcome existing limitations within the JAliEn security, while at the same time natively integrating containers within the middleware. This would involve modifying/creating a new JobAgent, suitable for this purpose. Furthermore, changes from here may be further propagated to the CE, being closely tied to how JobAgents are executed, and in turn onwards to the distribution model from there. In other words, the JobAgent provides a convenient approach to start integrating the drafted changes of Section 5.4, gradually moving up the execution stack and hierarchy of JAliEn.

The next chapter will detail the necessary changes to the JobAgent to achieve this, with the CE to follow after that.

CHAPTER 8

ISOLATING GRID JOBS

Grid jobs are executed through a dedicated job pilot process, known as the JobAgent within AliEn. This remains the same within JAliEn, which implements its own version of it. The initial version of the JobAgent within JAliEn has been mainly aimed at replicating the feature set of the original, in turn forgoing new development and intended features. This chapter aims to rectify this through the introduction of a new, "two layered" JobAgent, which utilises containers to improve on both security and compatibility.

8.1 The purpose of a JobAgent

As introduced in Chapter 2, the JobAgent is the middleware component responsible for executing Grid jobs. It is started by a site batch queue (BQ) system, on worker nodes associated with that site, having been submitted by the Computing Element (CE) when it detected waiting Grid jobs compatible with the given site. Specifically, it is the process that, in the end, will start the executable specified in the JDL of a job.

The JobAgent works by pulling jobs from the central queue. After a site CE has identified a potential job match, it will submit a shell script to the BQ of the site, that in turn launches a JobAgent when executed on a worker node. Once started, the JobAgent will contact the central queue[1], and pulls the first job that matches with the local conditions and the configuration of the JobAgent. By examining the JDL, it will fetch any necessary input files, packages, as well as the main executable. The latter will afterwards be started as a child process once all preparations are done, where the JobAgent will also provide monitoring while the execution is ongoing. When a job completes, the JobAgent is additionally responsible for uploading any output files, as requested by the JDL, and cleanup of leftover files and processes.

To summarise, the responsibilities of the JobAgent are as follows:

- *Job Matching* – Find a job in the central queue compatible with the current computing node / site
- *Environment* preparation – Examine the job JDL and install any required packages
- *File download* – Fetch input files specified in the JDL

[1]Note that the main matching logic is handled by the JobBroker, a central service, once it receives worker node information from a JA.

- *Job execution* – Launch the payload executable
- *Monitoring* – Monitor the payload execution, providing traces and status updates
- *File upload* – Upon completion, upload the specified output files
- *Cleanup* – Remove leftover files and processes, and prepare for matching a new job

As mentioned in Section 7.2, JAliEn uses a new token-based form for authentication, where each Grid role has a token with permissions just enough to fulfill that specific role, be it a *user*, *job* or *JobAgent*. However, as can be seen in the section above, the responsibilities of a JobAgent are not solely confined to those of the JobAgent role, but also include tasks related to job execution. Consequently, in order to satisfy all of the original responsibilities of the JobAgent, its initial JAliEn implementation requires both a job- and JobAgent token in order to function. This arguably defeats the purpose of the new, token-based, authentication model, as permissions are no longer limited by each role.

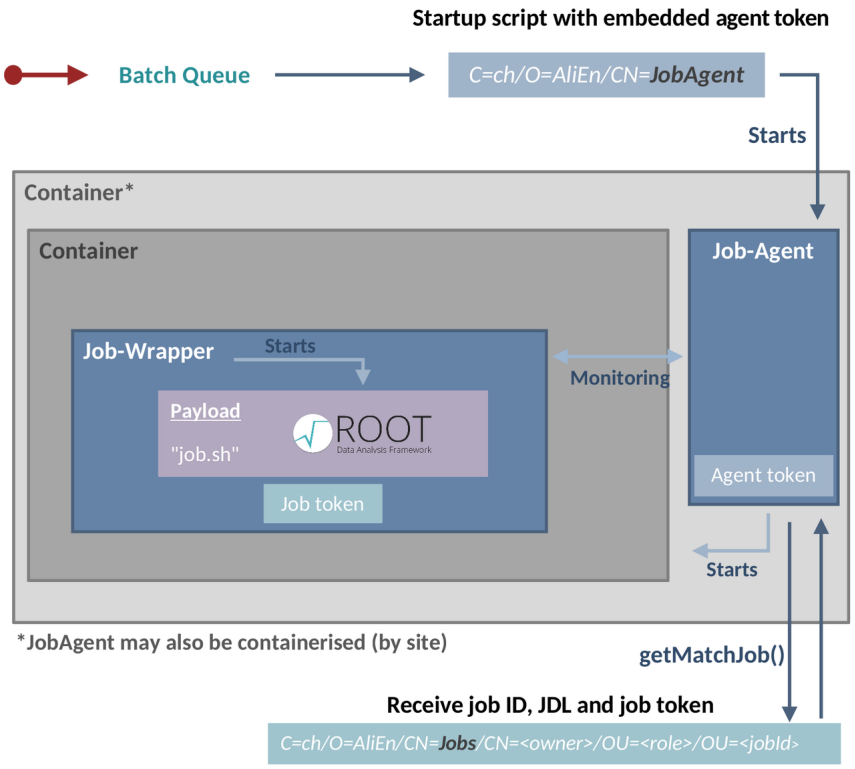


Fig. 8.1: A split, two layered, JobAgent. As opposed to directly starting the payload in a container, as in Figure 5.1, the JobAgent is instead split into two separate components, where the JobWrapper takes care of job execution within the container. This allows the JobAgent to have a separate token, and more control over the containerisation – now spanning across the isolation through two instances [90].

8.2 A new JAliEn JobAgent

To rectify the limitations of the JAliEn JobAgent, and following discussions with the teams concerned, a proposal was made for the creation of a new split ("two layered") JobAgent: having one component dedicated for job matching and monitoring (JobAgent role), and a separate for job execution (job role). This would allow for fully harnessing the presence of the new fine-grained token certificates, so to introduce more isolation within Grid job execution. Furthermore, this isolation enables the adoption of containerisation, which in turn may also be used to further strengthen it – and to integrate JAliEn with the vision of Section 5.4.

A proposal for how the split JobAgent may be realised is drafted in Figure 8.1. As illustrated in the figure, the original JobAgent has here been split into two separate, independent[2], processes: A JobAgent and a *JobWrapper* – each with its own JVM, and no shared resources. The JobAgent maintains all tasks related to matching/management and monitoring, while the JobWrapper only handles job execution (existing only as a *wrapper* for the payload). This also includes the fetching/uploading of files, as is also permitted by the job role/token.

Having the JobAgent split into two independent components simplifies the process of containerisation, as the task of initialising and managing the container may be folded into the existing monitoring tasks performed by the JobAgent – making it possible to build support for containerisation within the middleware itself, without having to rely on external scripts. This includes both launching the container, with a correct environment – as well as checking for underlying support for containers in the first place. However, as will be seen in the subsequent sections, this also introduces additional challenges.

8.2.1 *Requirements for two layers*

Figure 8.1 depicts the JobAgent starting the JobWrapper as an independent subprocess. In order to make it fully independent, it would need to be launched by a shell call, similar to that of the JobAgent – not using native Java APIs. However, not using native Java APIs comes with the caveat that the JobAgent (JA) process will have less information, and also, less means for managing and monitoring the JobWrapper (JW) process. This creates novel challenges, generally uncommon within Java:

- *Data transfer* – Job matching is performed by the JobAgent. This means all necessary files required for job execution will thus be fetched by the JA, while being needed by the JobWrapper for execution. Furthermore, the JW also requires a job token for its permissions, which may be requested by the JA. In other words, there is need for data exchange between the JA and JW, in order to transfer necessities such as
  - Job JDLs
  - Site info
  - Configuration options

[2]Separate/independent processes are necessary to isolate between roles and their active tokens.

  - Environment variables
  - Tokens

- *Monitoring* – One of the JA responsibilities is to update the central traces with information regarding the status and progress of job execution. With the JW being an independent process, this information is not available to the JA by default. In other words, there is need for a two-way passing of information between the disconnected JA and JW.

- *Process management* – The JA must monitor the JW and payload, however, it is also required to act on it: killing processes that run past their TTL (Time to Live), or overuse their allocated resources. With the JW being independent, the JA has no trivial method of identifying individual processes (i.e. which one is the JW, which one is the payload/children). Even if the payload has exceeded limits and must be terminated, the JW may still have pending operations, such as uploading error logs, and should thus avoid being terminated prematurely.

And importantly, the above points must be achieved in a manner that may also work while being *containerised* – which by design is intended to prevent trivial communication between the container/host.

8.2.2 *Achieving two layers*

The challenges outlined in the previous subsection may largely be attributed to the lack of communication between the two now independent components. Consequently, means are needed for allowing communication between the two, in a manner also compatible with the use of containers. By itself, this is not a unique challenge, as other use-cases that require communication between processes inside/outside the encapsulation of a container are common, here often using bind-mounts and shared files, as well as Unix sockets to exchange data. However, these approaches largely work through breaking down parts of the isolation between container and host, allowing for more access. More novel means of communication and resource sharing, such as through the use of a shared memory region, could be possible. This however comes at the cost of further increasing complexity, which will exacerbate long-term maintainability.

A makeshift data pipe for communications

For the implementation, a different approach has been chosen altogether[3]. It must be noted that the JW is nevertheless a subprocess of the JA, and while direct Java APIs for management are unavailable, OS specifics – i.e. that of the targeted Linux/Unix platform – still apply. This creates an opportunity for data exchange between the processes from within Java, without having to go through other means. Specifically, as the parent process, the JA can spawn the JW in a way that allows the JA to send messages to the JW standard input (STDIN) and read messages from the JW standard output (STDOUT). The communication goes through a pair of pipes or a socket pair,

[3] https://gitlab.cern.ch/jalien/jalien/-/merge_requests/21

created in advance and subsequently tied to STDIN and/or STDOUT as desired. This is a common way for propagating input messages to subprocesses, and is available as an `OutputStream` from the process once started. However, this `OutputStream` may also be used to transfer serialised data, such as JDLs, tokens and other variables. The only requirement is that the data is read in the same order as sent, within the receiving process (JW). This enables the creation of a makeshift data-pipe: serialise and send data from the JA by writing to the `STDIN` of the JW, and receive data by reading from its `STDOUT`.

The benefit of the above approach is not only in its simplicity, but also in it functioning when the JW is being executed within a container. Singularity will conveniently create container processes that start with STDIN, STDOUT and STDERR (Standard Error) inherited from itself, as if those processes had been created outside of the container. Also, as the "/proc" virtual file system is shared between host and container, the JA could even make use of that to connect to the JW, for example if the latter's STDIN and STDOUT were set up in a way to allow such a rendezvous. This approach turned out to have some caveats, however, as STDIN/STDOUT were observed to become polluted with occasional strings and characters as more child processes are spawned, making it impossible to transfer additional serialised data once the JW has fully started. While that problem remains to be debugged further, basic communication between the JA and JW may afterwards be handled through Unix signals instead, as will be discussed later below.

Managing the JW and its processes

An interesting problem that arises from the division of the JobAgent, is where the exact cutoff line should be between the two processes. This is straightforward for responsibilities which are determined by the permissions granted by the tokens, but for other tasks – such as tracking the job status and updating the traces, this is not as trivial – as it is permitted independently of the JobAgent and Job roles, and caveats are associated with giving it to either:

- Monitoring is generally associated with the JobAgent role. However, in the two-layered approach, the JA does not have any information regarding the progress of a job, as this is handled by the JW.
- The JW has a more fine-grained overview of both the status of the payload, and the overall progress of the job. Should the JW however fail, the JA – which is responsible for matching new jobs – would have no indication on how to proceed, not knowing if a job had failed, completed or how to otherwise act on it.

As a compromise, a consensus for a middle ground was reached: all general monitoring, such as tracking the TTL and resource use, would continue to be tracked by the JA. Job specifics, like tracking the Job progress, would be handled by the JW. However, the JW is to occasionally write the job status to a file – allowing the JA to know the state of a job in case the JW should encounter any issues, and to an extent, continue in its stead. This should allow for the best of both worlds, and avoid increasing the responsibilities of JW: instead, allowing the JW to exist as mainly an execution wrapper, with the JA

being able to monitor and (partially) recover from failures if the JW should crash or be terminated – updating the status and starting a new JW instance.

The JobAgent remains responsible for the majority of monitoring tasks, and this includes keeping payloads within their given resource and TTL limits. Should a process exceed any of these, it is the responsibility of the JA to terminate it. This is trivial within the original JAliEn JA, as the payload is a direct child. For the two-layer JA, it is instead handled by the JW, which introduces additional complexity – and even more when running it containerised.

The JA will occasionally need to terminate a payload. However, this must be done in a manner without disturbing the JW – which will update job states and upload logs. In other words, the JA must somehow identify which of its children is the executing payload, and only kill this specific process. One way is through counting, assuming that each subprocess is spawned in order, and grabbing the PID (Process Identifier) of the first positioned process relative to the JW (provided as a table). Unfortunately, this will not work when introducing containers, as frameworks such as Singularity launch multiple springboard processes in-between, shifting the positions/PIDs.

When the JW is launched, the command string used for launching the JA is repeated, but with the classname changed to that of the JW. However, this string may be further modified before execution – allowing for the addition of a unique identifier, that in turn may make it possible to easily find it back in a list of processes. Consequently, the unique `JobID` for the job to be executed is added as a JVM ID, enabling the JA to identify and fetch the PID of the JW process among all executing processes. Granted, while this will allow the JA to identify the *JW* and not the *payload* (job), it provides the JA with means to *signal* the JW with an intent – by sending a SIGTERM, which may be caught[4] by the JW, and in turn be set to terminate the payload, and proceed with uploading files and logs.

8.2.3 Enabling containers

Splitting the JobAgent into two separate layers as described in Section 8.2.2 is essential for providing an entrypoint for integrating containers with JAliEn. However, upon achieving a two-layered JA, what remains is to encapsulate the now independent JW process.

The end of Section 8.2.2 describes a way for identifying the JW process, by modifying the original launch command string intended to start it. Likewise, this string may also be further modified to add the appropriate calls for launching the command inside a container. There are however two questions that must be answered in order to realise this:

- The exact *appropriate calls*, i.e. determining if a job should run containerised, and how it should be launched.

- How to ensure the container has an appropriate environment, which is capable of executing the job – and initially starting the JW.

[4] Built-in feature of Java.

In order to supply the JobAgent with answers to the above questions, a separate "*containerizer*"[5] module has been created, aimed at simplifying the process of containerisation[6][7].

The containerizer lives in its own separate package, consisting of multiple Java classes implementing a factory-like pattern, intended to provide the correct containerisation when called by the JA. Specifically, it takes the launch string, intended to start the JobWrapper, and wraps it in the necessary commands needed for having the process start in a container. This is done by probing the host for the presence of a container framework (such as Singularity), and attempting to first launch a test inside it. Should this test succeed, the containerizer will in that case proceed with wrapping the JW launch string in a similar command as for the successful test – which the JA will then launch as usual, unaware of it now being set to run inside a container or not. Given that a factory pattern is used, there is also support for adding additional containerizers: while Singularity is the initial target, the technology itself remains rapidly developing, with new frameworks emerging, such as Podman. As JAliEn is expected to be used in production for a decade or longer, being locked to a single container framework as new solutions and improvements arrive is a cause for concern. Therefore, to prevent vendor lock-in, additional *containerizers* may be added – small Java classes that provide the necessary commands needed to use a specific container framework. When called, the factory will then probe each of the containerizers until a compatible one is found for the host, and proceed with using it for wrapping and returning the JW launch string back to the JA. An overview of this implementation outlined in Figure 8.2.

Having a dedicated containerizer also aids in simplifying that the correct environment is initialised within each container. In order to successfully start the JW process here, a requirement is that all the necessary packages, including Java, are already present in the container. A possibility is to manually update the container image (discussed in Section 8.3) to have it reflect the requirements and current packages within CVMFS. This would however mean a duplication of work, as each change intended for CVMFS would also need to be reflected within the image. Therefore, as the containerizer can adjust the startup call used for the JW, it may instead ensure that CVMFS is mounted within the container, and add an additional call to *alienv* – a tool used to automatically change the system environment variables to point to the packages and tools provided in CVMFS, as opposed to using the system binaries (discussed in detail in Section 9.1.1). This ensures that the JW, and consequently the payload, always start with a known environment as made available in CVMFS.

8.2.4 Enabling GPU support

The use of GPUs as accelerators to speed up job execution within the Grid has become more prevalent in recent years, and has become a requested feature for JAliEn [91]. However, the use of this feature within JAliEn must consider the use of containers, which by default will isolate the payload from additional hardware resources, such as

[5] Note the use of US spelling here, for bringing it in line with the rest of the naming scheme within JAliEn.

[6] https://gitlab.cern.ch/jalien/jalien/-/merge_requests/21

[7] https://gitlab.cern.ch/jalien/jalien/-/commit/ebf39429fc589d7c39d83884d985c4c185cfcc46

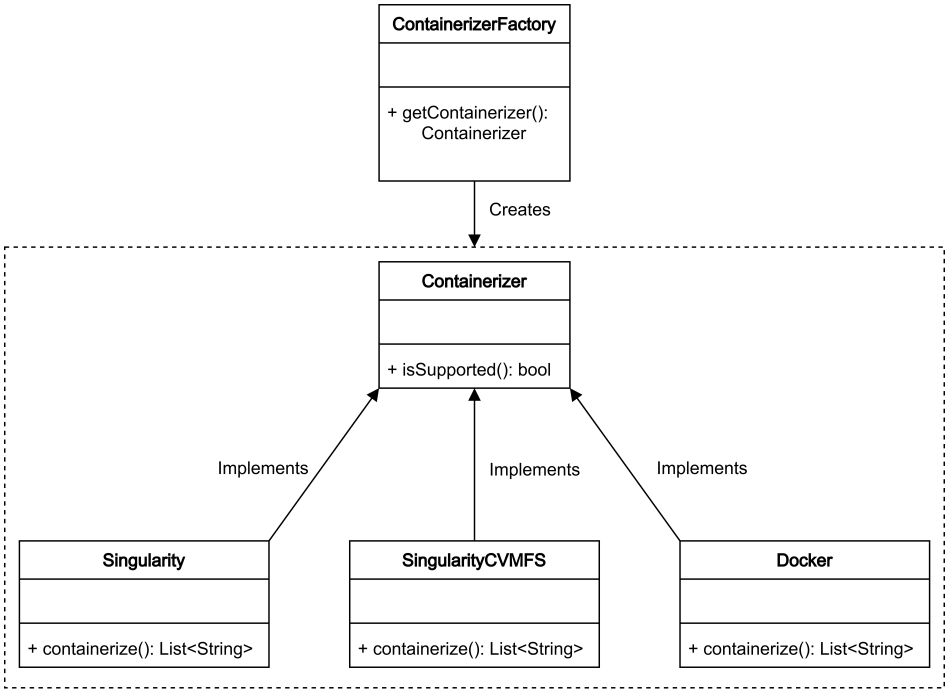


Fig. 8.2: Overview of the JAliEn Containerizer. The factory class calls on the Containerizer, which in turn will probe each of the Containerizers for compatibility, and return whichever is compatible, based on a predetermined preferred order. "SingularityCVMFS" is a custom CVMFS-based Singularity runtime, discussed in Section 10.1.4.

GPUs. Singularity requires certain flags to be provided in order to allow the container access to GPU resources – with an individual flag for AMD and Nvidia cards. These flags may however not be trivially passed to Singularity for all instances, as the absence of either AMD or Nvidia hardware when a flag has been passed earlier will prevent the container from executing – in turn failing the job.

In order to pass Singularity the correct flags at the correct time, the containerizer from the previous subsection has consequently been extended to also probe for compatible hardware, before providing the JobWrapper launch string with the appropriate containerisation commands[8]. This is done in a manner similar to how Singularity detects the presence of GPUs: for Nvidia, the presence of `/dev/nvidia[0-9]` files will be checked, as each dedicated system GPU will have an `nvidia` entry added, numerated in order. Conversely, and in a similar manner, the presence of the `/dev/kfd` file will be checked for AMD. Once these files are detected, the appropriate flags are added to Singularity, allowing jobs to use the GPUs from inside their container.

One thing to note is that some sites restrict the GPUs made available to jobs, using specific environment variables to only expose some of their resources. Consequently, these variables (`CUDA_VISIBLE_DEVICES` (Nvidia) and `ROCR_VISIBLE_DEVICES` (AMD)) need to be made available within each container. While this could be inserted by the JobAgent, the containerizer has instead been extended to provide this functionality – as this is the location of both the main containerisation code, and the initial check for GPU support.

[8] https://gitlab.cern.ch/jalien/jalien/-/merge_requests/108

8.2.5 Meta variables

One of the benefits of containers is their ability to provide a clean, uniform, environment across sites for job execution. However, occasionally site- or network specifics require the presence of additional environment variables being made available for the payload. In such situations, these variables and their values must somehow be propagated into the container environment.

In order to provide sites and their admins means to propagate their custom environment variables to the payload containers, the concept of *meta variables* has been introduced[9]. This allows any environment variables in need of being propagated to the payload to be defined in a separate variable known as `META_VARIABLES`, which will be picked up and forwarded by the CE, and afterwards examined by the JA once started on a worker node. The specified list of variables and their values are sent to the JW through the makeshift pipe from Section 8.2.2 – in turn making them available for the payload.

8.3 A container for the JAliEn JobAgent

Chapter 6 details the creation of a container for VO-Box use. While the initial proposal for a new Grid workflow suggested the use of a common container, based on Singularity, for both the VO-Box and JobAgent, the shift to Docker for the VO-Box containers has changed this outlook. Consequently, a dedicated container image is required for the JA – specifically, for the JW – that is capable of executing payloads. This comes with a number of requirements that must be met, in terms of both configurations as well as available packages. As mentioned in Section 8.2.3, *alienv* ensures that a number of packages from CVMFS are automatically loaded into the environment. Furthermore, the JW will also load additional packages into the environment as specified in the JDL of the payload. For these cases, it is sufficient to ensure that the subdirectory `/cvmfs` is available in the image, in order to allow bind-mounts for all versions of Singularity (more on this in Section 10.1.4)[10]. However, payloads may also expect the presence of other packages that are largely taken for granted within Grid worker nodes, which the absence of would cause jobs to fail.

A major question is which are the exact packages needed on a worker node in order for jobs to execute successfully. While there is a meta-package available, known as `HEP_OSlibs`[11], complete with a large collection of every package and library deemed to be possibly needed, its large size and overlap with packages in CVMFS make it best avoided if possible. In other words, the only means through which to know the exact packages needed would be by trial-and-error, attempting to run jobs, and adding additional packages the absence of which caused jobs to fail. This exact approach was taken by the Grid team at GSI[12] Darmstadt for their own containerised Grid implementation, who graciously shared the list of packages successfully used by their

[9] https://gitlab.cern.ch/jalien/jalien/-/commit/21141af89aeb08452a0fe63306cdc831b7a99d49

[10] And as mentioned in Section 6.5.1, to avoid duplicate CVMFS caches, it is more beneficial to have CVMFS installed on the host, and bind-mounted into each container.

[11] https://gitlab.cern.ch/linuxsupport/rpms/HEP_OSlibs

[12] Gesellschaft für Schwerionenforschung.

site in production as a Singularity definition file[13].

Using the obtained package list, a new Singularity image was created. Based on CentOS 7, the CERN recommended distribution at the time, it contains the packages provided above, as well as a number of additional debug packages, such as `strace`, to simplify development. Furthermore, it also contains pre-created `/cvmfs` and `/workdir` directories, used for bind-mounting, and will be the topic of Section 10.1.4.

A path to faster OS updates

By harnessing the fact that every job is to run containerised, quick OS updates become a possibility, in turn allowing production jobs to target later OS versions earlier. This is generally a slow and tedious process, as site admins would need to update all their worker nodes – and this has to be done across all sites. Through a distributed container image, that may be shared through CVMFS, it should suffice to simply share the image, and have it automatically pulled by the newly launched JobAgents. This could be used to quickly migrate all ALICE sites e.g. to CentOS 8, with minimal intervention needed.

8.4 Requirements for production

Before the now split and containerised JobAgent may be tested on a site or put in production, the remaining gaps in its functionality must first be filled. Specifically, Section 7.3 and Section 7.3.2 identify the following missing functionality:

- File handling is limited, with incomplete parsing of JDL files and reduced upload options, as well as registration of uploaded files in the central catalogue not being fully implemented for all conditions.
- Job states are simplified compared to AliEn, diverging in logic and making job progress harder to interpret.
- Cleanup is limited, with files and processes being left as-is.

To rectify the above limitations, the following changes have been proposed and implemented within the new JAliEn JobAgent:

- *A new file parsing scheme* – The list of files to upload will be taken across all outputs provided in the JDL (i.e. both for legacy (AliEn) and current (JAliEn) syntaxes, as well as combined). Duplicates are removed if found. Furthermore, detection of files to be uploaded is done recursively, allowing files to be collected across subdirectories, as supported by AliEn[14][15].
- *Archive registration* – AliEn allows for the upload of files as a single archive, having its contents displayed as individual files in the catalogue – in turn allowing for browsing and downloading them individually. The same functionality has been implemented within JAliEn through a new API call `bookArchiveContents`,

[13] The Singularity equivalent of a *Dockerfile* (Section 6.4.1).
[14] https://gitlab.cern.ch/jalien/jalien/-/merge_requests/57
[15] https://gitlab.cern.ch/jalien/jalien/-/merge_requests/89

which generates a new GUID and PFN, where the latter is composed of the PFN of the archive, combined with the filename of the archive entry[16].

- *Upload of error files* – If a job fails, the JA will no longer simply terminate with an error, but proceed with uploading error logs as provided in the JDL. If this entry is missing in the JDL, a default set of logs will be uploaded[17][18].
- *Adding missing job states* – Job states and their transitions are now similar to those of AliEn, with the exception of `SAVED`, previously an intermediary state between the completion of file upload and the files having been registered in the catalogue by an AliEn central service. In JAliEn, the registration of uploaded files is instead handled by the JA and the intermediary state is thus no longer needed. Additionally a new job state, `ERROR_S` has been introduced, for situations where a specified outputfile in the JDL was not found. This condition will be ignored if the job ends in an `ERROR` state[19].

8.5 Moving outside the JobAgent

The creation of a split, two-layered, JobAgent as described in this chapter, has allowed for a built-in containerisation of the payload, directly within JAliEn – in turn ensuring a uniform execution environment across all sites[20], as visualised in Section 5.4. However, the JobAgent is but one component of the overall job execution workflow, and changes made here will also need to be reflected throughout the rest of the Grid middleware – as will be described in the next chapter.

[16] https://gitlab.cern.ch/jalien/jalien/-/merge_requests/101
[17] https://gitlab.cern.ch/jalien/jalien/-/merge_requests/55
[18] https://gitlab.cern.ch/jalien/jalien/-/commit/800ffcc91456d64f13640ffed681e710c30cdcc7
[19] https://gitlab.cern.ch/jalien/jalien/-/commit/c8d964d1f98a520cd3ad29438b81f3fafb69e98c
[20] Provided deployment is a success, as will be discussed in Chapter 10.

CHAPTER 9

CONSTRUCTING A GRID WORKFLOW

Chapters 6 and 8 saw the creation of containerised versions of both the VO-Box and the JobAgent. While essential for realising the draft provided in Section 5.4, they remain as separate containerised units, with little consideration given to the current state of other components. Consequently, questions remain for terms of distribution, build and integration with other services. As will be seen, components, such as the CE, will need updating in order to function with the larger changes of the new JobAgent. This chapter will move beyond the core containerised services of both the VO-Box and JobAgent, and describe the changes needed to adapt and use these services towards the construction of a new Grid workflow.

9.1 The JAliEn Computing Element

As mentioned in Section 2.6.1, the Computing Element (CE) is responsible for distributing jobs on worker nodes on Grid sites. This is done by querying the central services for pending jobs, and generating JobAgent startup scripts – which in turn are placed in the local batch queue of a Grid site, and afterwards executed on the individual worker nodes. Within JAliEn, the CE is also responsible for requesting JobAgent tokens, which are fetched from the central services and inserted into the JobAgent startup script.

In Chapter 8, the JobAgent component was split in two, resulting in a separate JobWrapper component being responsible for the payload execution. The JW may in turn be containerised by the JA – which provides it with everything needed for executing both it and the payload. However, it remains the responsibility of the CE to provide the JA with a sufficient environment for starting it.

9.1.1 *Preparing the JobAgent script*

There are a number of necessities that must be provided in the JA startup script, in order for it to execute successfully. Specifically, the following must be present:

- VO-Box hostname
- The associated site name
- Users allowed to submit to the site and/or users blocked from using it
- JobAgent token

- JobAgent startup call

For the latter, this involves a call to *java*, providing it with the path of the JAliEn jar, and the JobAgent class. This is the same command that will again be used when starting the JW (Section 8.2.2), albeit with the initial class replaced.

Section 7.3 mentions that at project start, no JAliEn CEs were running, or available for testing. While fully capable of generating JA startup scripts, the CE had also not been updated to reflect changes in directories and file naming as development had progressed. Furthermore, means were needed for ensuring that the JAliEn dependencies, as well as JAliEn itself, were available on the nodes when the startup script was set to be executed.

To ensure the availability of the necessary files at startup, the JAliEn CE contained a hardcoded URL (Uniform Resource Locator) that was used to fetch a precompiled jar – as well as a hardcoded CVMFS reference for Java. This was however problematic, as both hardcoded paths pointed to old binaries, no longer compatible with more recent changes. Consequently, when testing JAliEn in Section 7.3, the CE could not be used, and all necessary binaries had to be provided as input files, and executed on top of an AliEn JobAgent. A better approach is needed.

Alienv and CVMFS

As opposed to using hardcoded paths and URLs, more automated means are needed for ensuring that correct and up-to-date binaries are made available to the startup scripts. This is a convenient use-case for *alienv*.

Alienv is an auxiliary product providing a shell script *alienv* for changing the system environment, based on Environment Modules [92]. It updates the existing environment, mainly by prefixing the various "PATH" variables with directories in CVMFS that correspond to the requested version of a supported product. In other words, if a system without Java needs it to execute a binary, a call like "`alienv enter Java`" may be made. This will update the environment with the Java binaries in CVMFS, allowing a user to simply call `java`, as if it was natively installed on the host system.

Alienv is tightly interconnected with *Alibuild*, an automated build system that tracks dependencies across Alienv environments [93][94], and allows for automatically pushing completed package builds to CVMFS – including their own Alienv environment. Releases are organised in *tags*, with a dependency tree to support interdependent tags and environments – with the specific tags and build recipes as used by ALICE stored in the public *alidist*[1] repository.

An Alienv tag is generally used to represent the corresponding Git tag of a package, allowing new versions to easily be made available in CVMFS, complete with an environment providing all dependencies as needed[2]. Consequently, Alienv provides a convenient mechanism for giving the JAliEn JobAgent startup script everything it needs to start and run the JobAgent.

[1] https://github.com/alisw/alidist

[2] E.g. an Alienv tag labelled JAliEn 1.3.0 corresponds to tag 1.3.0 in the JAliEn respository.

Harnessing alienv

Prior to any changes, a preexisting Alienv entry is already available for JAliEn - allowing the use of "`alienv enter JAliEn`" to enter an environment with everything needed to start a JAliEn instance – be it the ComputingElement (with some provisions), JShell or a JobAgent. To achieve this, the environment contains paths to CVMFS for binaries and other files such as:

- Java
- JAliEn jars
- JAliEn starter (jar wrapper)
- OpenSSL
- XRootD
- SQLite
- Certificate Authorities

By default, the Alienv call will set up the latest tag that has been made available through Alidist, while a specific version may also be provided, e.g.: `alienv enter JAliEn/1.3.0-1`. While the latest JAliEn tag provided within Alienv for JAliEn was initially outdated, having it updated to point to the very latest JAliEn tag in Git is a straightforward process – simply updating the build recipe to point to the repository of the updated tag. What is arguably more interesting, is how to best integrate the use of Alienv within the CE, in order to always provide the correct files and environment for JobAgent startup.

Alienv is but one of many executables that will occasionally need to be called from CVMFS by JAliEn[3]. To avoid hardcoding their directories throughout JAliEn, all CVMFS calls have consequently been moved to a separate CVMFS class – in turn separating the base CVMFS call from the desired executable. In other words, this class handles the specifics of each executable call from CVMFS, such as Alienv, with the base path to CVMFS easily changed. This allows the CE to request a specific environment for the JobAgent startup, where the CVMFS class will automatically provide it with the correct call and version.

The handling of the correct version is done through three separate means:

- *Environment* – If JAliEn is already running in an environment provided by Alienv, the same tag as used for the environment will be returned to the CE when asked.
- *Binary* – If there is no indication of Alienv being used, the same tag as used for the binary will be returned. This is propagated to JAliEn from Git upon compilation.
- *Manual* – In order to easily switch between versions, an option has also been provided to manually specify the desired tag. This can be done in a separate Java properties file named `version.properties`.

[3]Other examples include cleanup scripts and SiteSonar tests, which will be discussed later.

Through these means, together with Alibuild/Alidist, the JobAgent may be started solely through the use of its startup script. As a consequence, there is no need to preinstall JAliEn binaries on the worker nodes – everything needed to start the JobAgent is initialised through the startup script and Alienv.

9.1.2 Managing CEs

The JAliEn CE may be run by manually compiling the source code, and thereafter executing the corresponding jar file. With the changes for ensuring the availability of the necessary binaries and other files in the previous subsection, the JAliEn CE has everything it needs to be able to run. However, having to manually pull and compile the code for every release is a tedious process. Combined with occasional JVM incompatibilities, which may prevent successful compilations, it would encourage leaving older versions of the JAliEn CE running for longer. In contrast to JAliEn, the legacy AliEn CE may be fully run from CVMFS, without requiring prior installation – ensuring availability of the latest version. It would be optimal to allow this for the JAliEn CE as well.

As seen in Section 9.1.1, an Alienv entry is already provided for JAliEn, which has also been updated to reflect the latest corresponding Git tag. Like for the JobAgent, this may be used to provide the latest version of the CE to VO-Boxes, directly from CVMFS – both being part of the same JAliEn jar. A few changes were however needed for this to be fully transparent:

- Albeit a single jar binary is used for JAliEn, that binary is never called directly, but instead called by a separate `jalien` script provided by Alienv. This script needed to be adjusted in order for the CE to be a valid choice. Furthermore, the default memory options for the JVM provided also needed to be increased, as the CE is a larger component that is set to be executing for months at a time.
- While proper monitoring of the JAliEn CE remains to be provided, a separate PID-file is now generated at start time, allowing the JAliEn CE to be picked up as a valid AliEn service by existing monitoring services.
- The same directory structures were adapted within JAliEn as for AliEn, allowing logs and temporary files to be placed as expected by users – and likewise adjustable in LDAP the same way as for AliEn.
- Most of the configurations read from LDAP will now be updated automatically when changed, which avoids having to restart the CE manually in order to have them applied.
- In case of connection errors, the JAliEn CE would default to throwing a Java exception, and terminate. Instead, the connection will now be retried, allowing the CE to run uninterrupted in case of network fluctuations, or high load.

Furthermore, the JAliEn CE also featured a JBox, used for maintaining connections. The intention behind this addition was initially to allow the VO-Box to act as a multiplexer for connections in earlier designs, but it is now left unused once the CE has initialised itself - and was thus removed.

Management Scripts

The `jalien` script provided by Alienv may be further wrapped in a separate management script, which in turn automatically initialises Alienv, sets the correct environment for certificate paths, provides monitoring and otherwise manages the lifecycle of the CE – allowing for automatic restarts if the service should terminate unexpectedly. A bare-bones version of such a script was created while using the JAliEn CE for development and early deployments[4], with a separate project by colleagues in the ALICE Grid team providing a production-ready version of such scripts towards the end of the project[5].

9.2 Putting pieces together

Chapter 6 introduced a containerised VO-Box, for use by sites to host their services, such as the CE, while Chapter 8 presented a split, two-layered, JobAgent, with a containerised job execution wrapper. Within the context of the draft in Section 5.4, these two components may each be used to provide new workflows within their own corner of the Grid, though in a limited scope when compared to the Grid as a whole. However, as each of these components come into deployment, they may further be used together as building blocks for creating a cohesive, new, Grid workflow.

9.2.1 *A shift in responsibilities through JAliEn*

In addition to its initial focus on the CE, Section 9.1 also introduces *Alienv* and *Alibuild/Alidist* – which combined with CVMFS provide means for automatic versioning, builds and distribution. Specifically, Alienv provides modern means to deploy new versions of JAliEn across the Grid: once a new version is tagged in the Git repository, the updated tag may afterwards be propagated to Alidist, where Alibuild will pull the latest changes and start the build process – before pushing the latest tag to the Grid, with the new tag correspondingly being made available within Alienv. This is the mechanism used by the CE in 9.1 to make sure each JobAgent binary is made available to the startup scripts on individual worker nodes. Here, it must be noted that the latest tag automatically becomes the new default. This means that whenever there is an update to JAliEn, each subsequently launched JobAgent may automatically use the very latest version of it – shifting the responsibility for versioning from the site admins, to the ALICE Grid team.

9.2.2 *A common execution environment*

Section 8.2 introduced a new, split, JobAgent, which in turn could start and execute its second layer within a container. A dedicated container image was likewise made for this purpose (Section 8.3): based on CentOS 7, it contains everything needed to run a job. This image will likewise be distributed through CVMFS, and provides a uniform execution environment for jobs, independent of site or worker node. If Singularity (or

[4] https://jalien.docs.cern.ch/site/ce_guide/
[5] https://gitlab.cern.ch/jalien/vobox-scripts

any other container framework) could be made available across all sites and worker nodes, that would shift the responsibility for managing the execution environment also away from site admins, and in a more centralised direction – as in the previous subsection. This will be discussed in detail in Section 10.1.

9.2.3 *A common VO-Box configuration*

Chapter 6 introduced a containerised VO-Box, that could be used to launch a number of required VO-Box services automatically. Combined with the management scripts in Section 9.1.2, it can also be used to quickly launch a JAliEn CE, directly from CVMFS. Should the containerised VO-Box be adopted across Grid sites, it would be yet another step towards a more centralised and homogeneous set of configurations being used within the Grid, as with the above subsections.

As can be seen in the above sections, there is a shift towards a more homogeneous workflow – with more automated steps for deployment centred around CVMFS.

9.3 From draft to reality

Following the discussion of Section 9.2, there are now ample means available for constructing a new Grid workflow. However, it remains to be fully deployed – not only testing one component at a time, but their interactions as a whole. Furthermore, it would need to be scaled, adjusted for possible differences and incompatibilities between sites, and put in production across the Grid – if deemed suitable. This will be the focus of Part III, followed by an evaluation of the end results in light of the initial RQ, and the further impact of the work within Grid and distributed computing.

Part III

DEPLOYMENT AND EVALUATION

If deploying software is hard, time-consuming, and requires resources from another team, then developers will often build everything into the existing application in order to avoid suffering the new deployment penalty.

—Karl Matthias [95]

CHAPTER 10

DEPLOYMENT

Part II presents a foundation for a new workflow: a one-click containerised VO-Box, a two-layered JobAgent with automatic containerisation, a standardised container image, powered by the new JAliEn middleware – and all bound together through a common distribution mechanism around CVMFS. When combined, it allows Grid jobs to not only run isolated from the underlying worker nodes, but also in a standardised environment across sites. However, questions remain in terms of wider deployment, adoption and compatibility for such a workflow across the Grid.

10.1 Probing for compatibility

At the very core, two conditions must be satisfied in order to fully utilise the developments outlined in Part II: the availability of CVMFS (for software distribution) and Singularity (for containerisation)[1]. For the former, this is an existing requirement across all Grid sites, now being the default for software distribution (Section 4.1.2). However, for the latter, the availability of Singularity is not as guaranteed. Sites have been informed of the desire to have Singularity installed on worker nodes, but it has remained up to each site to go through with this.

10.1.1 *Not every installation of Singularity is equal*

To further complicate the availability of Singularity, it must also be mentioned that Singularity may be installed in two different manners – with each having a different subset of available functionality:

- *Privileged* – This is the default configuration, provided when installing Singularity through a package manager. It needs to be *setuid-root*, as it enables features which require elevated permissions, such as mounting single-file images and the ability to freely bind-mount directories within containers.
- *Unprivileged* – In this mode, Singularity runs without setuid-root. This makes it more secure, as no additional privileges are given, executing as the same user both inside and outside the container. This also allows it to run from an installation outside of the root file system. However, at the time of project commencement, it

[1] The exception being the VO-Boxes, which, if containerised, need to run on Docker.

is unable to bind mount directories from outside the container, and may only run extracted images – appearing as directories in the filesystem.

The container image described in Section 8.3 has been created in the format of a directory tree – this allows parts to be loaded on demand, without having to fetch a single large file before execution. It may theoretically also be run by unprivileged Singularity, as no elevation of privileges is required for mounting it on the filesystem. However, it does require the availability of bind-mounts in order to be used in production – i.e. being able to mount directories from the host inside the container – as the job working directory, where files are fetched and the payload executed, must be made available. In other words, there is a need for not only having Singularity available on worker nodes, but even as a *privileged* instance.

10.1.2 *From overlay to underlay*

Privileged Singularity contains the functionality needed to bind-mount directories on the host within containers. However, a privileged installation alone is not a guarantee that this functionality is made available for the JobAgent. Specifically, this requires the *overlay* option to be enabled in its configuration file – which is disabled by default. In other words, it does not suffice to have sites install Singularity: they will also need to enable the corresponding option.

Expecting every Grid site to provide a privileged installation of Singularity is not realistic. Historically, Grid sites tend to be conservative in the permissions they provide jobs with, aiming to minimise the damage caused by misbehaving or even malicious Grid jobs. Assuming all will provide Singularity with elevated permissions, let alone with overlay enabled, is thus not an option.

ALICE is not the only experiment at CERN drafting the use of containers within its Grid execution workflow. Consequently, a dedicated WLCG Containers Working Group has also been formed[2], allowing WLCG stakeholders to discuss container specific issues and approaches across experiments. Here, the issue of needing a privileged/configured Singularity for bind-mounting is likewise a relevant concern across the members of the working group. As a result, a possible solution was pitched in that forum: as opposed to bind-mounting a local folder into a read-only image, instead start from a local directory with read-write permissions and bind-mount the image subdirectories on top of it – which should be possible even without elevated privileges. This suggestion was later pitched to the developers of Singularity, and finally included as the *underlay* option[3].

Underlay enables bind-mounts, even on unprivileged installations of Singularity. However, it too has to be enabled in the Singularity configuration and requires a recent version of Singularity.

10.1.3 *Configurations across the Grid*

In order to better understand the software and configuration landscapes across the Grid, and consequently gain insight into how many sites would benefit to which extents

[2] https://twiki.cern.ch/twiki/bin/view/LCG/WLCGContainers
[3] https://github.com/apptainer/singularity/issues/1207

from the new JobAgent, a test was constructed to probe the configuration of the worker nodes within the Grid. Submitted as a common Grid job, the test consists of a script that scrapes the host and Singularity configurations of the worker node it lands on, and saving the results for inspection. A large number of instances of the test job were submitted, in order to cover as large a proportion of sites/nodes as possible, with the results being sanitised to avoid duplicates.

The results gained from the test jobs can be seen in Figure 10.1. To summarise, the following can be extracted from the figure:

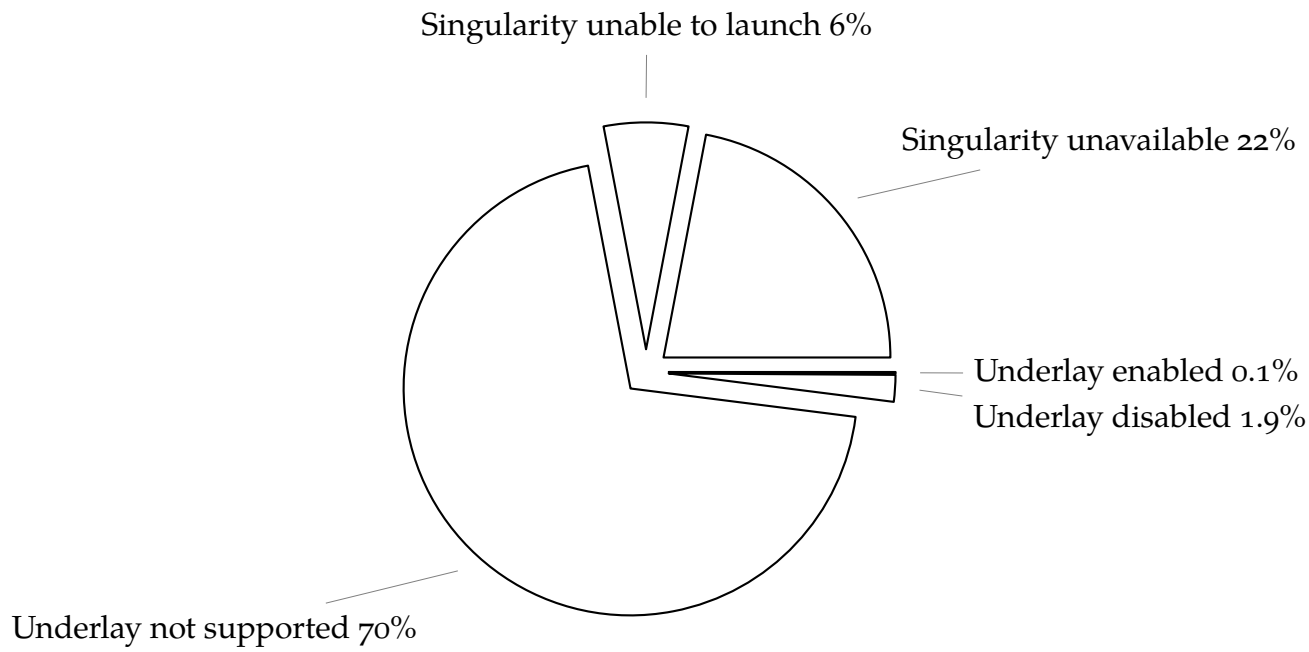


Fig. 10.1: Overview of Singularity compatibility across probed sites, with only 0.1% of worker nodes having a working installation of Singularity with underlay enabled.

- A majority of Grid worker nodes have Singularity installed on the system (78%)
- A significant subset of worker nodes have a broken installation (6%)
- Only a handful of worker nodes have a version supporting underlay (2%)– with only 0.1% having it enabled.
- As an interesting side-note, a sizeable number of sites that do not support Singularity are capable of running it directly from CVMFS standalone thanks to their recent kernel (13%). This will be a central point for distributing Singularity in Section 10.1.4.

The above results are discouraging, as this would imply that the vast majority of sites and worker nodes are unable to support the new containerised JAliEn JobAgent implementation. A solution or workaround is needed.

10.1.4 Working around site incompatibilities

With only a handful of sites found to have a suitable configuration in the previous subsection, a different approach is needed for deployment. Specifically, underlay (or overlay) is required for bind-mounting writable directories within container images that are read-only, which will always be the case when having the container of Section 8.3 in CVMFS. This is a requirement not only for accessing the working directory, prepared in advance by the JobAgent, inside the container, but also for mounting CVMFS inside it – to be used by jobs.

Another unprivileged way of bind-mounting

It must be noted that an exception exists for the requirement of elevated privileges (or underlay) in the context of bind-mounting: it is only required if the directory does not already exist within the container image. In other words, bind mounting `/tmp` from the host onto `/tmp` in a CentOS 7 container will always work, as it exists within the container image. This provides an opportunity for creating a workaround for the lack of support for overlay/underlay on Grid sites and their worker nodes.

While the initial testing of the split, containerised, JAliEn JobAgent was done using a generic CentOS 7 image, the creation of a custom image in 8.3 opens up possibilities for further customisation. Specifically, dedicated directories for both CVMFS and the job may be created in advance within the image, allowing for unprivileged bind-mounts on all Singularity enabled sites, without the need for newer versions of Singularity of special configurations. This is not a completely transparent process from the perspective of the JobAgent, as job working directories are assigned unique identifiers based on the job ID. Furthermore, the new container must be made available on worker nodes across all Grid sites.

To work around the above issues, the new JAliEn container of Section 8.3 has been published within CVMFS, under a new directory dedicated to container releases for ALICE[4]. This container has also been equipped with two new directories: `/cvmfs`, for CVMFS access, and also a `/workdir` for the job working directory. Correspondingly, this path has been hardcoded within the JAliEn JobAgent, where the unique directory for each particular job is now mounted here when containers are enabled: e.g. `/scratch/ALICE/jobs/alien-job-1234` will become `/workdir` when executed in the container. The JobAgent will handle the necessary preparations, making the process transparent for jobs. This allows the new JAliEn JobAgent to be containerised on all sites supporting Singularity in Section 10.1.3.

As an added benefit, the hardcoded handling of directories within the JobAgent also allows for having the standard temporary directory (`/tmp`) inside the job working directory (`/workdir/tmp`). This prevents jobs from filling up the temporary directories of Grid nodes, as everything in `/workdir` is cleaned up once execution completes.

A custom Singularity through CVMFS

The original approach to Singularity relies on sites both installing and maintaining it. As evident in Section 10.1.3, this approach has its caveats: versions may be

[4] `/cvmfs/alice.cern.ch/containers/fs/singularity/centos7`.

outdated, misconfigured or simply broken. This aligns well with the experiences of other members of the WLCG Containers Working Group – where it was unanimously decided to encourage the use of standalone, unprivileged, instances of Singularity from CVMFS instead.

As opposed to system installations of Singularity, and due to their unprivileged mode of operations, standalone installations of Singularity consequently rely on the availability of *user namespaces* for containerisation. By default, this feature was disabled[5] in CentOS. To avoid on relying on site admins and local Singularity configurations, one final configuration change is needed.

It must be mentioned that the default setting of having user namespaces disabled within an enterprise-focused distribution, such as CentOS, is done for a reason: enabling user namespaces comes with an additional risk for exploits [96]. That said, the Open Science Grid project has deemed this as an acceptable risk, as the Linux kernel is far more reviewed than Singularity [97]. Furthermore, the additional privileges obtained through user namespaces are limited.

Starting with CentOS 7.6 [98], user namespaces are enabled by default. In response to the emergence of sites supporting user namespaces, either manually enabled by site admins or as sites gradually upgrade to newer OS releases, a custom Singularity build was created for use with JAliEn, and consequently deployed to CVMFS[6]. Through a custom *containerizer* (Section 8.2.3), it allows sites to run jobs containerised using a fully configured, standalone, Singularity build – without any additional steps required by site admins, provided a recent version of CentOS is installed.

10.1.5 Tracking sites and their worker node configurations

Having both a workaround for achieving unprivileged bind mounts, as well as a custom Singularity build in CVMFS, again raises the question: how many sites are compatible with running jobs containerised, using the new JobAgent? This is but one of several questions which have emerged during development – what is the default JDK (Java Development Kit) on most worker nodes? Is there support for underlay? Is a specific package available? Or as in Section 10.1.3, is Singularity available across the Grid? – In other words, a need exists to have an idea of the configurations of worker nodes at sites within the Grid.

In preparation for a potential update to CentOS 8, possibly affecting the use of the containerised JAliEn JobAgent (as suggested in Section 8.3)[7], the need for having more insight into the heterogeneous configuration landscape within the Grid was made even more apparent: is it realistic to target CentOS 8 for Grid jobs? Which would imply the usability of Singularity on all Grid sites.

The probing script used in Section 10.1.3 can be used for gaining the above information, but it has its limits, as data must be extracted and processed manually. A project dedicated to reworking such scripts, in turn creating a separate tool for analysing the status of Grid worker nodes, was pitched to, and later accepted by, Google

[5] The number of user namespaces set to 0.

[6] `/cvmfs/alice.cern.ch/containers/bin/singularity/current`

[7] These plans were put on hold, as CentOS 8 was discontinued by RedHat [99].

Summer of Code[8]. This project led to the creation of *Site Sonar*: a dedicated tool for discovering the configuration of Grid worker nodes across the Grid. Various properties to check or probe for may be queried through its interface, with results shown per site. While initially based on a similar probing test job as in Section 10.1.3, the logic was later reworked to have it composed of several testing modules in CVMFS – with the JobAgent modified to call these upon startup. This gives full coverage over all Grid nodes in operation, as SiteSonar tests will be run each time a JobAgent is started.

SiteSonar has proved to be an essential tool throughout this project, allowing for gaining an overview of the Grid as a whole (as in Section 4.2), investigating container (Singularity) support, and asking sites to adjust their configurations accordingly, if possible[9]. Through the use of SiteSonar, it has also been verified that with the combined use of the unprivileged bind-mount workaround, and custom build of Singularity in CVMFS, the vast majority of Grid worker nodes are now capable of running the new, containerised, JAliEn JobAgent.

10.2 Initial deployment

Having confirmed compatibility across Grid worker nodes in the previous section, it remains for all components to be fully deployed, as discussed in Section 9.2 and Section 9.3. In this regard, this section outlines the additional steps required for achieving a fully working deployment.

10.2.1 Running the new JAliEn JobAgent

While the new JAliEn JobAgent was able to run by piggybacking on the legacy AliEn in Section 7.3, this was far from being a flawless process. Specifically, ALICE Grid jobs come equipped with a hard memory limit for execution across all sites, which is set to 8 GB. When that happens to be exceeded, the guilty process will be forcefully terminated. However, Java has an aggressive behaviour towards memory, which cannot be fully overridden – which in turn creates problems for deployment.

Physical memory to be allocated by Java may be set through the dedicated `-Xms` (minimum) and `-Xmx` (maximum) flags. These flags do not apply to the amount of *virtual* memory allocated by Java, however, where it will always allocate a specific amount relative to the total amount of physical memory on the host. The implications of this are that if a JobAgent is to start on a worker node with a large amount of available physical memory (>200 GB), the Java instance responsible for executing the JobAgent will quickly surpass 15 GB of allocated virtual memory. While this is not something that would normally be considered an issue – as this memory is not truly used, but only allocated "just in case" – resource monitoring within the Grid often does not differentiate between physical and virtual memory, resulting in the JobAgent being terminated shortly after startup.

The above behaviour remains the same irrespective of the JAliEn JobAgent running on top of the legacy AliEn (as in 7.3), or natively after updating the JAliEn CE (Section

[8] https://hepsoftwarefoundation.org/gsoc/2020/proposal_ALICEgridcontainers.html

[9] Collected data may currently only be viewed internally at CERN, while the individual tests are located in CVMFS (/cvmfs/alice.cern.ch/sitesonar/sitesonar.d).

9.1). From a Java perspective, this allocation of virtual memory is immutable, and may not be tweaked or adjusted as for physical memory. This means that for a large number of sites, the JAliEn JobAgent will simply be unable to run – irrespective of it using containers or not.

To prevent Java from allocating virtual memory above the 8 GB threshold (and to leave as much memory available as possible for the job payload), a workaround using 32-bit JDK binaries has been applied to the JobAgent. This makes Java unable to address and detect more than 4 GB of physical memory, which in turn greatly limits the allocation of virtual memory (<120 MB). To avoid impacting other tools relying on Java, this change has not been propagated to Alienv, but only applied to the JobAgent, using a separate JDK in a dedicated 32-bit directory in CVMFS (but likewise maintained by the CVMFS class from 9.1.1, to avoid fragmentation), with the Java path correspondingly inserted by the CE in the JobAgent startup script.

Support for larger memory thresholds was later implemented. However, despite this change, the above workaround has remained in place. Using 32-bit binaries greatly reduces the memory overhead observed by hosts, leaving more for the job payload.

10.2.2 Bundling system libraries

The use of 32-bit binaries for running the JAliEn JobAgent, as described in the previous subsection, has its benefits. However, starting with CentOS 7, 32-bit libraries, which are required for running 32-bit Java, are no longer preinstalled by default. While this could normally be solved through containerisation, the initial call to Java happens far too early in the chain: before the JobAgent has had a chance to start, and before there is a containerised JobWrapper.

Upon switching to 32-bit Java, nearly all CentOS 7 sites were unable to execute Grid jobs when tested. As all remaining CentOS 6 sites were also going to switch to CentOS 7, this would mean a complete halt in job execution[10]. Furthermore, having all sites install 32-bit libraries is not a feasible option either, as this is a slow and tedious process, which may not succeed: it comes down to the possibilities at individual sites and their responsiveness. It was for this very reason decided to attempt a switch to CentOS 8 solely through containers in Section 10.1.5. As the HEP_OSlibs package (which is installed across worker nodes) only supports 64-bit dependencies, a different approach has thus been taken for executing the JAliEn JobAgent.

The specific libraries required by the 32-bit JDK are mainly related to Glibc – an implementation of the C-standard library provided by the GNU Project, which is the system default on CentOS 7. This brings forward a potential workaround: as Java, packages and containers may all be distributed through CVMFS, staying independent of the environment provided on the host system, a possibility could be to include the required 32-bit libraries through CVMFS as well.

Bundling system libraries in CVMFS has proved to be possible, although being far more convoluted compared to the above examples, as it requires patching both JDK binaries and the corresponding libraries. Specifically, the following are needed:

- Libraries required to be placed in CVMFS

[10]The support for larger memory thresholds, as described near the end of Section 10.2.1 was only made available towards the end of the project.

  - `libc`
  - `ld-linux`
  - `libdl`
  - `libnsl`
  - `libnss`
  - `libpthread`
  - `libresolv`
  - `librt`
  - `libz`
  - `libm`

- JDK patches
  - Change interpreter for the `java` binary to CVMFS `ld-linux`
  - Set *rpath* to CVMFS library directory for `java` binary, server `libjvm` and client `libjvm`

- Library patches
  - Set *rpath* to CVMFS library directory for `ld-linux` and `libnss_dns`
  - Overwrite "`LD_LIBRARY_PATH`" in `ld-linux` with a string that will not match any environment variable[11]

The 32-bit libraries placed in CVMFS were taken from a 64-bit CentOS installation, after installing the optional 32-bit compatibility libraries. The libraries were patched directly using `patchelf` and `sed`, without modifying the original source code. The `hexedit` utility was used for checking and determining where patches were needed.

With the above changes, the JAliEn JobAgent may run on arbitrary configurations using the 32-bit JDK, irrespective of the host OS. As Java relies on the kernel for spawning subprocesses, the payloads started by the JobAgent will retain the full 64-bit address space.

It must be noted that the shift towards providing all required system libraries in CVMFS introduces another significant change: not only are the containers provided centrally, but also all of the environment and dependencies needed to launch them, and subsequently the JobAgent and the payload. Combining everything, the following can now be provided directly through CVMFS:

- A dedicated, custom, container to run jobs in
- Singularity to start the container
- JAliEn for managing Singularity
- JAliEn dependencies

[11] To prevent libraries getting picked up from `LD_LIBRARY_PATH`.

- Java runtime for starting JAliEn
- Glibc & other libraries for launching the Java runtime

In other words, the changes introduced to universally support a 32-bit JDK can be said to provide its very own OS in CVMFS, used to bootstrap the environment needed for starting a JobAgent, and subsequently its Singularity container. Consequently, the JAliEn JobAgent can now run almost anywhere in the Grid, provided CVMFS and a recent Linux kernel (>=3.10-957, Section 10.1.4) are present. This deployment of a 32-bit JDK and libraries is completely transparent for sites, and will automatically be picked up by JAliEn once run.

10.3 Results as deployed

Combining the findings described in this chapter, together with the developments throughout Part II, provides everything needed to deploy a new Grid workflow. This section will describe and provide a full overview of the end results as deployed within the ALICE Grid, and view them in light of the initial draft of Section 5.4. Specifically, highlighting that three core concepts have emerged.

10.3.1 A containerised workflow

Figure 10.2 depicts the new workflow as deployed first at CERN, and later scaled across ALICE Grid sites. In many ways reminiscent of the draft from Section 5.4, it presents JAliEn as the managing instance, handling startup, containerisation and job execution. While similar, it also differs from the initial draft in several ways. As opposed to everything being in a unified container, mainly based on top of Singularity, there are instead two separate images, for two separate use-cases: job (worker node) and CE (VO-Box), where the latter is based on Docker, because of its more advanced networking stack. Furthermore, there is only one layer of containerisation within the JobAgent, in contrast to the draft where containers could be seen used for the JobWrapper, and again for the payload. That idea was put on hold for the time being because of instabilities experienced with nested containerisation. For further isolation, a site may launch each batch job inside its own (Docker or Singularity) container – a feature now supported by some batch systems.

10.3.2 Automatic build and distribution

Section 5.4 opens up the idea of better harnessing CVMFS for distribution, allowing for more automatic updates and deployment, without the need to involve site admins. In Figure 10.2, it can be seen that everything is now indeed taken directly from CVMFS, with changes pushed through Alidist. This applies not only to the JAliEn and Java binaries, as containers, libraries and dependencies are all taken from CVMFS as well. While not mentioned in Section 5.4 or its initial draft, Figure 10.2 also features versioning through the use of Alienv. This implies not only more transparency for users/admins, but also ensures having a consistent, up-to-date Grid, and in turn less concerns about old versions being used for too long.

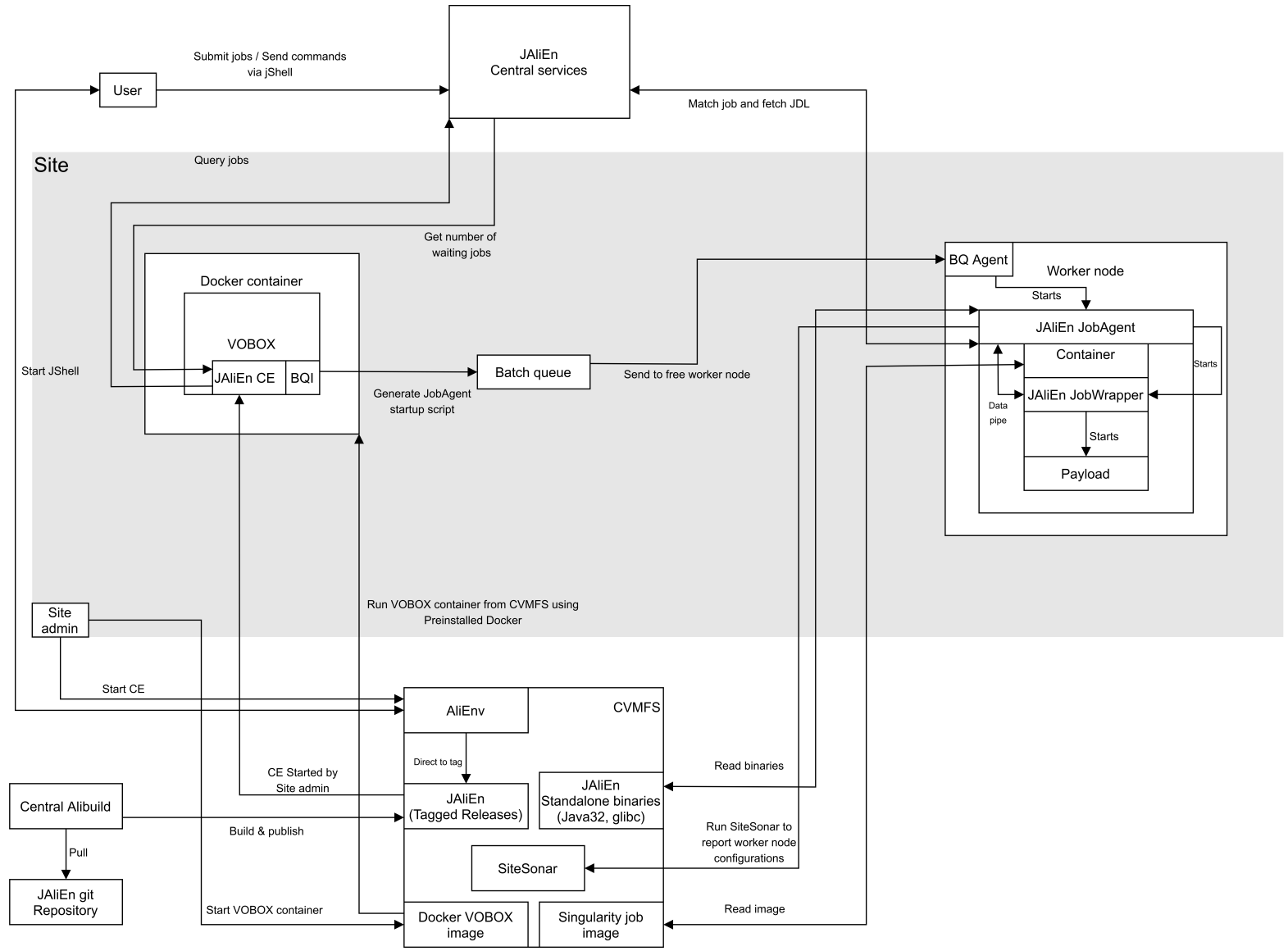


Fig. 10.2: Deployed version of the new Grid workflow. Both the VO-Box and payload (via JobWrapper) are run in their own containers, taken from CVMFS, and based on a central repository. The JobAgent has been split in two, with all libraries needed to start it at sites taken from CVMFS – meaning no installation is needed in advance.

10.3.3 A more homogeneous Grid

A common trend throughout the development is the increasing *homogeneity*, compared with the draft of Section 5.4. While jobs are executed in a standardised environment through the container, as seen in Figure 10.2, this also applies to site services, which have their own dedicated VO-Box container. Additionally, as seen in the previous subsection, it can be expected that JAliEn versions are likely to be the same across sites – including those on the VO-Boxes, as JAliEn is likewise fetched from CVMFS there, as illustrated in Figure 10.2 . Furthermore, as described in Section 10.2.2, for worker nodes that are not containerised, all the necessary system libraries for starting JAliEn are also provided through CVMFS, meaning the only requirement from the perspective of a worker node is a recent Linux kernel. All of these instances can be maintained centrally and distributed through CVMFS – shifting the responsibility away from the sites themselves.

CHAPTER 11

RESULTS AND EVALUATION

This chapter turns the focus back onto the original RQ of Section 1.4, aiming to examine the developments of Part II, as well as the consequences of deployment specifics from Chapter 10, in light of it. Specifically, to determine if the core of the RQ was met – making ALICE computing more flexible and easier to administer – by examining the direct outcomes, and determining if there are any corresponding gains.

11.1 Wider adoption

Chapter 10 described an initial deployment of the developments of Part II within a limited scope at CERN. This was initially done through a test site known as Juno[1], allowing for the testing of features and functionality, without disturbing production jobs submitted by others. As components became feature-complete, these were moved outside Juno, and deployed within "true" production environments at CERN – before being scaled and deployed on sites throughout the Grid.

11.1.1 *Container VO-Boxes*

Within CERN

The containerised VO-Box described in Chapter 6 was, as implied above, initially deployed only for the test site Juno at CERN. However, as the instabilities of existing VO-Boxes, as mentioned in Section 5.1, persisted, being able to migrate sites to an alternative became pressing – and multiple problematic CERN VO-Boxes were consequently converted to containerised instances. As described in Section 6.7, this initial migration was a success, with previously troubled VO-Boxes now showing stable performance and less load on the underlying host system.

The containerised VO-Box was initially based on Scientific Linux CERN (SLC) 6. This was partially done to mimic the "tried-and-tested" setup found within SLC 6, so as to avoid issues possibly related to CentOS 7 – but also to be able to re-use the existing init scripts. Once a full init replacement was completed for the containerised VO-Box (Section 6.4.1), the image was consequently updated to CentOS 7. With a large number of ALICE VO-Boxes still on SLC 6 at the time, the new containerised VO-Box thus provided an ample upgrade path.

[1] Short for "JAliEn Uno".

Given the observed instabilities of the VM VO-Boxes at CERN, with some still on SLC 6 – with a rapidly approaching EOL (End of Life)[2] – all remaining VM VO-Boxes were moved to the new containerised option. As of mid November 2020, all production VO-Box instances at CERN are containerised.

Outside CERN

For the majority of other ALICE sites, the matter of migrating/upgrading existing VO-Boxes was not as urgent as for CERN, where the VO-Boxes need to handle by far the largest numbers of concurrent jobs. Many had stable, already upgraded, setups, in turn giving little incentive to changes. Furthermore, the updated JAliEn CE can run directly from CVMFS, without requiring additional installations and configurations – and as will be seen in the next section, this is the core component for adapting to the new workflow. However, for new or redeployed VO-Boxes, the containerised option provides a quick path for achieving a working setup. Consequently, at a number of ALICE Grid sites, VO-Box VMs were replaced with containerised instances.

11.1.2 JAliEn

Migrating sites over to JAliEn is a very different process compared to the VO-Boxes. While the latter are mainly responsible for providing an entrypoint at an ALICE site, and letting the middleware submit jobs to the batch system of a site (through the CE) – JAliEn is a middleware choice that is not necessarily tied to the VO-Box implementation. A broken VO-Box may prevent new jobs from being able to run on a site, while any JAliEn faults may cause all jobs at the site to break and fail, damage files or possibly abuse system resources. In other words, care must be taken to ensure proper job execution within JAliEn itself, and also to update and adjust JAliEn to ensure compatibility with as many sites as possible. This entails migrating only a limited number of Grid sites over from AliEn to JAliEn at a time, to avoid potentially reducing the capacity of the Grid, and to avoid creating "black holes" for jobs – where incoming jobs would be accepted and started by JAliEn, but then fail due to JAliEn errors or site incompatibilities.

Despite the above concerns, the changes outlined in Chapter 9 provide key benefits from the adoption of JAliEn and a new workflow: everything needed can be taken directly from CVMFS, requiring only a working VO-Box, CVMFS and a recent Linux kernel (on the worker nodes). While care must still be taken, this should nevertheless drastically reduce the steps needed for migrating from AliEn to JAliEn.

Within CERN

As in Section 11.1.1 for the containerised VO-Boxes, CERN sites were the first to be migrated to JAliEn. Albeit in this case as a continuation of the deployment of the containerised VO-Boxes, and not because of specific issues/errors – instead pioneering new software within more controlled environments, informally referred to as (virtual) "sites", each submitting jobs to resources at their hosting site. The first production site to be migrated was CERN-ZENITH, and based on the experiences with this site,

[2] SLC 6 EOL was on November 30th 2020 [100].

adjustments were made and two further sites were migrated. As of mid January 2022, all CERN sites have been migrated to JAliEn.

Outside CERN

By only having to convert the VO-Box services, migrating to JAliEn has been made straightforward for site admins – not requiring any additional action on the worker nodes. Furthermore, thanks to the startup scripts from Section 9.1.2, this process is also streamlined and automated through CVMFS. Consequently, other ALICE sites have been gradually moved to JAliEn by the ALICE Grid team. MonALISA provides an overview of sites running JAliEn, together with their version tags [101]. As of November 2022, nearly all sites have been migrated, with the single remaining site expected to be completed soon.

Containerised JobAgents

Through the containerised JobAgents, no additional configuration is needed on worker nodes when migrating to JAliEn - as the startup script and JobAgent will bring the full environment directly from CVMFS. As stated in Section 10.1.5, the vast majority of worker nodes should be able to run the custom Singularity build provided in CVMFS - with SiteSonar likewise indicating that all sites should have an OS with user namespaces to be enabled, e.g. a recent version of CentOS[3]. Consequently, upon switching sites to JAliEn, nearly all ALICE grid jobs are now being run inside the preconfigured environment provided in the job container (Section 8.3). At any site, local conditions may hamper the use of the job container and would need to be investigated case by case.

11.2 Consequences and gains of wider adoption

The wider adoption described in Section 11.1, and in particular for JAliEn, brings the workflow depicted in Section 10.3 to life – and into full production use. This in turn comes with a number of gains, and also wider consequences:

- *Simplified distribution* – As new versions are tagged in Git, these may automatically be pulled, compiled and pushed to CVMFS, thus made available for all. This not only goes for the JAliEn middleware, but may likewise apply to containers, such as both the VO-Box and job container – which each have their own dedicated repositories.

- *Fast deployment of new middleware versions* – Having access to all binaries in CVMFS through Alienv allows for quick updates of the JAliEn middleware: unless a specific version is given, simply restarting the CE will have it pick up the very latest tag, launching the latest version of the JAliEn CE, which in turn will let the latest version of JAliEn be invoked by the JobAgent startup script.

[3] As everything needed to bootstrap the JobAgent is in CVMFS, this can be any Linux-based OS, including Ubuntu. The only requirement is having user namespaces enabled, and kernel >=3.10-957.

- *Fast OS updates* – As suggested in Section 8.3, containers provide means for quickly deploying OS updates. This applies to both VO-Boxes, and worker nodes – and in particular to the latter, where it suffices to simply put the updated image in CVMFS, and update the associated production symlink. Here, there is no intervention needed by site admins.

- *Towards a more homogeneous Grid* – Having close to all jobs run in the same containerised environment takes a big step towards a more homogeneous Grid. That said, as the underlying hosts remain heterogeneous, differences may still be observed – or some sites may simply run without using containerisation, for various reasons. Nevertheless, having the majority of all Grid jobs run in an identical environment brings more predictability in terms of how the payload will behave.

- *Changeable environments* – There are generally few options for accessing worker nodes on Grid sites. While the VO-Box serves as an entrypoint, such is mainly for accessing the batch queue system of the sites – which thereafter distributes submitted work onto worker nodes. By having most jobs run containerised, convenient means become available for tweaking the environment for the payloads on worker nodes. By modifying the container image in CVMFS, additional packages may be included, configurations may tweaked or environment variables forwarded, as needed.

- *Shift in responsibilities* – Having a single image for all jobs, a single image for VO-Boxes and all necessary libraries in CVMFS, effectively shifts responsibilities, earlier imposed on site admins, to the ALICE Grid team.

- *Performance improvements* – As mentioned in Section 6.7, going from a VM VO-Box to a container was observed to allow handling more jobs, while at the same time having less load on the underlying host. Additionally, upon wider deployment, it was noticed that JAliEn requires less time for both starting, and finalising jobs, compared to AliEn.

- *Multicore* – Having a new middleware distributed across the Grid, opens up more possibilities in terms of adding functionality. As will be seen in Section 12.4, this has allowed other features to be introduced, in particular multicore support.

While the above mainly outlines positive gains of the new containerised, and JAliEn-based, workflow, it must be noted that it also comes with caveats. These may largely be attributed to JAliEn – and specifically the new JobAgent. It behaves differently to that of AliEn in many respects, not only due to being rewritten from scratch, but also for accommodating the changes outlined in Chapter 8 – which in turn can cause problems for legacy job payloads still targeting the original AliEn JobAgent[4]. Furthermore, the switch to a new middleware framework, which is also containerised, adds unfamiliarity to users and site admins[5]. While largely intended to be a drop-in replacement, there

[4]E.g. expecting a directory structure as provided by AliEn.

[5]Several sites did not realise their ALICE jobs were now running containerised until noticing that the payloads started to behave differently.

are nevertheless some configuration changes required e.g. in central LDAP, as well as new options being available, which may be unknown.

11.3 Evaluating results

Having outlined the wider deployment of the new solution, and the gains and consequences of it, this section aims to turn the focus back towards the original RQ. Specifically, the RQ of Section 1.4 asks:

RQ: *How can a new Grid workflow be constructed, using new technologies, within the ALICE Grid infrastructure to make ALICE offline computing more flexible and easier to administer?*

In response to this RQ, a draft was made in Section 5.4, presenting a new, reorganised, Grid workflow – benefiting from both new container technologies, and new middleware solutions. This resulted in the final product as described in Section 10.3. Using this as a base, this section will attempt to resolve if the provided solution – the new Grid workflow – serves as an answer to the original question, and to what extent it may be considered both *flexible and easier to administer*.

In order to provide aid in terms of definition, Section 1.4.1 clarifies the target of the RQ is finding a workflow that reduces the steps required for both deployment, and utilisation, within ALICE offline computing, while also being able to scale freely (i.e. not constrained to an initial deployment, being able to scale both in terms of deployment size and features). This provides means for quantitative observations.

Flexibility

Flexibility, as defined earlier, entails being able to scale freely in terms of both size and features. As shown in Section 11.1, the new Grid workflow is now available on nearly all ALICE Grid sites. At a glance, this would suggest that, at the very least, it is not constrained in its ability to scale across sites. Examining this further, it can be accredited to a number of attributes of the new workflow.

Figure 11.1 can be seen to show the underlying Unix load for several central machines, hosting both AliEn and the JAliEn central services. On the 19th of March 2021, the first major site (CERN-ZENITH) was converted to JAliEn, with a major rollout across sites taking place at the beginning of 2022 – and the penultimate AliEn site converted in October 2022. This can likewise be observed in the figure. As migration progressed, a gradual decrease in occasional load spikes, as well as a decrease in overall load, became an emerging pattern. During this period, the amount of running jobs remained approximately the same throughout (Figure 11.2).

There are some discrepancies in the aforementioned pattern, most notably a disproportionate *increase* in load for some of the central servers, and in particular for aliendb4. Upon investigation, the cause is likely accredited to an uneven rotation of IPs in the front-end DNS alias for the central services, with more work being directed to aliendb4 instead of e.g. aliendb3. Nevertheless, upon a complete migration of around 70[6] active sites, the central services have remained as stable, if not more so, as for the legacy services.

[6] http://alimonitor.cern.ch/stats?page=proxies

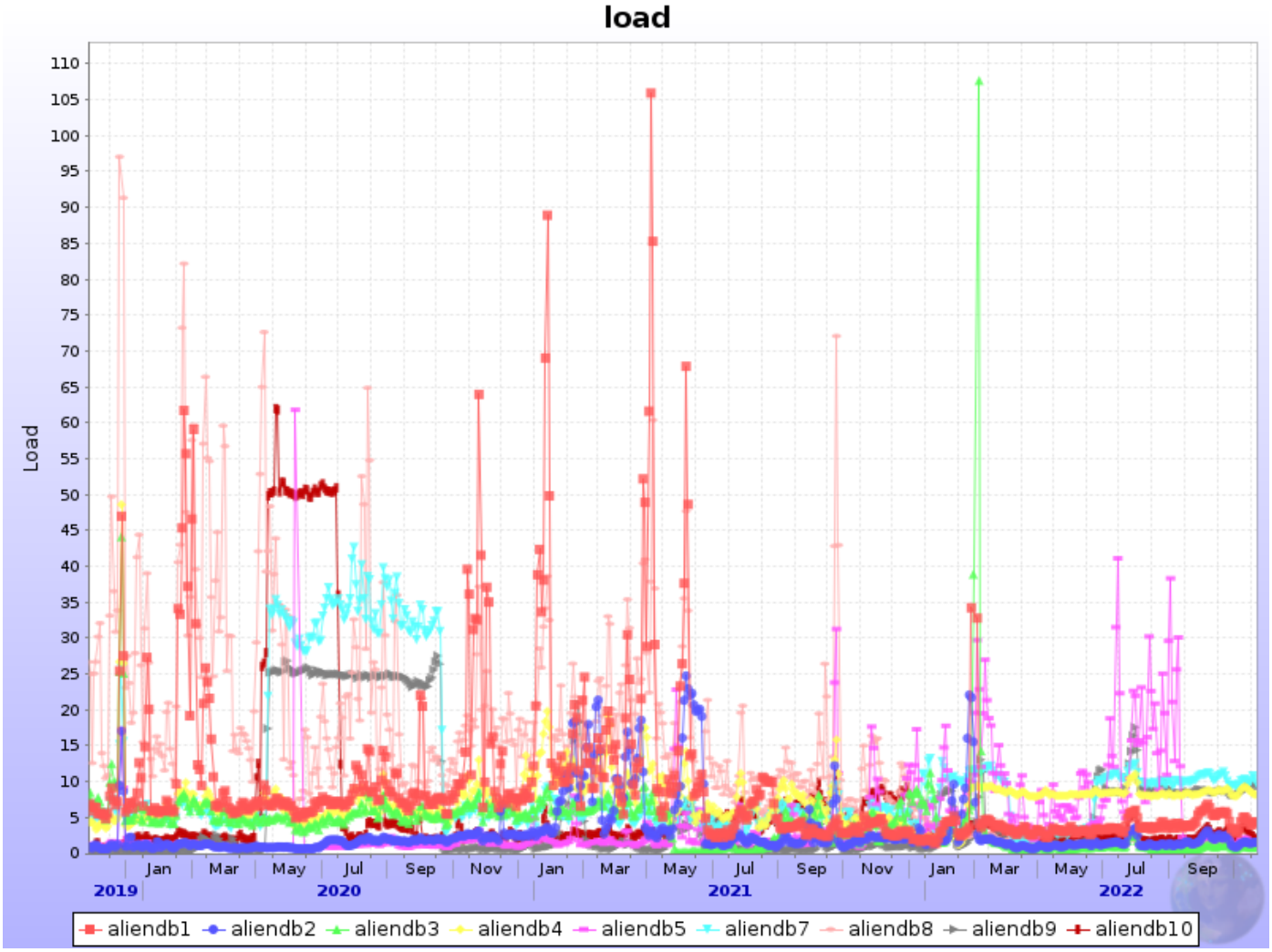


Fig. 11.1: Unix load on central machines over a three year duration, containing the time before, during and after migration. The prefix "aliendb6" is reserved for back-end database servers.

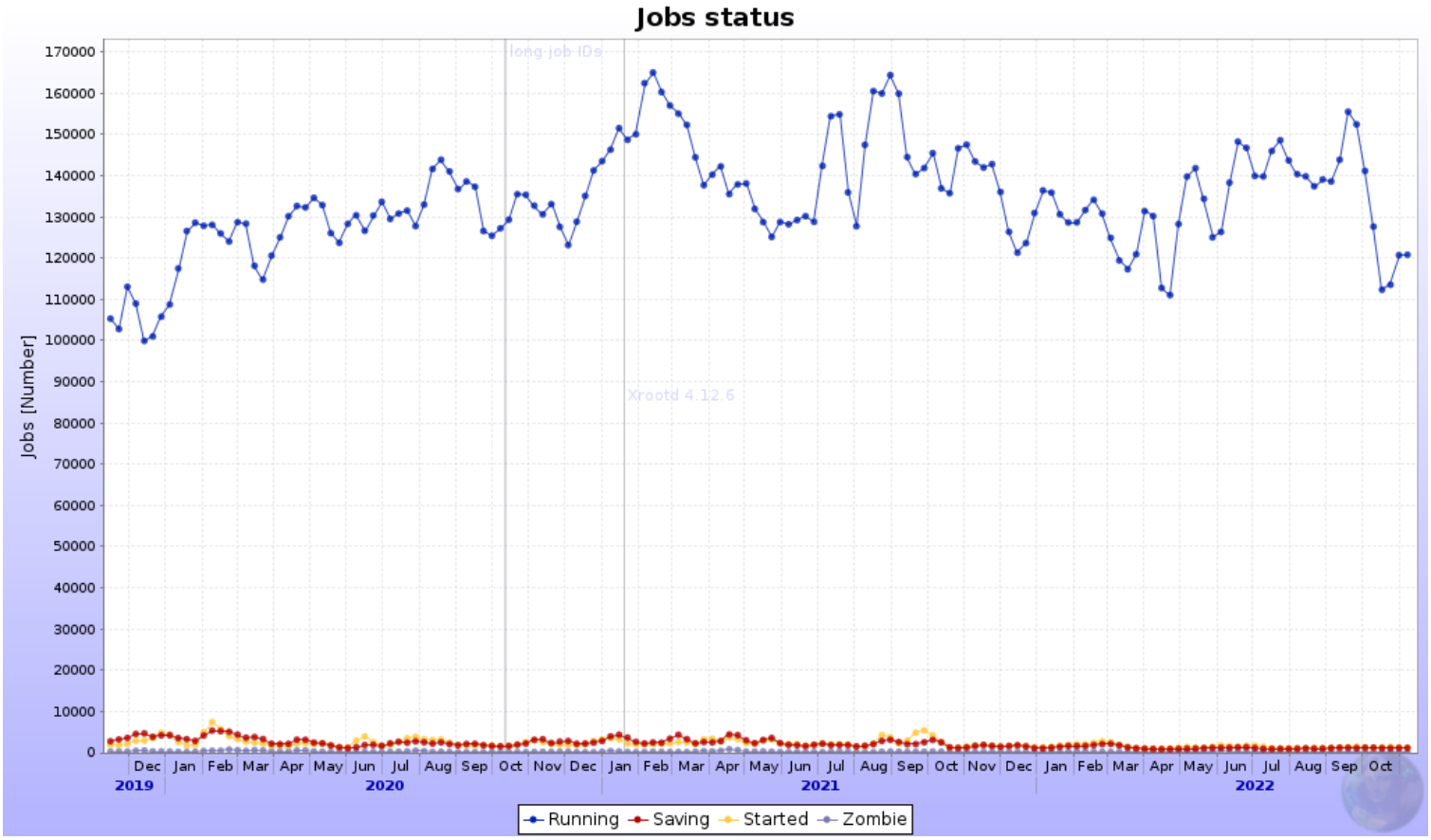


Fig. 11.2: Number of total active jobs across ALICE Grid sites, and their states, for the same duration as in Figure 11.1.

While both load, and the *number* of concurrent jobs, have remained stable upon the deployment of the new workflow, there has been no degradation to the jobs themselves. Figure 11.3 depicts the accumulated number of failed jobs per failure type – in the period before, during and after migration. While a gradual rise in the newly introduced ERROR_S state (Section 8.4) can be seen as adoption increases, there is no significant change or increase across the other error states. Furthermore, as legacy jobs are phased out and otherwise updated to account for possibly missing outputfiles, the number of ERROR_S states can also be seen to flatten over time.

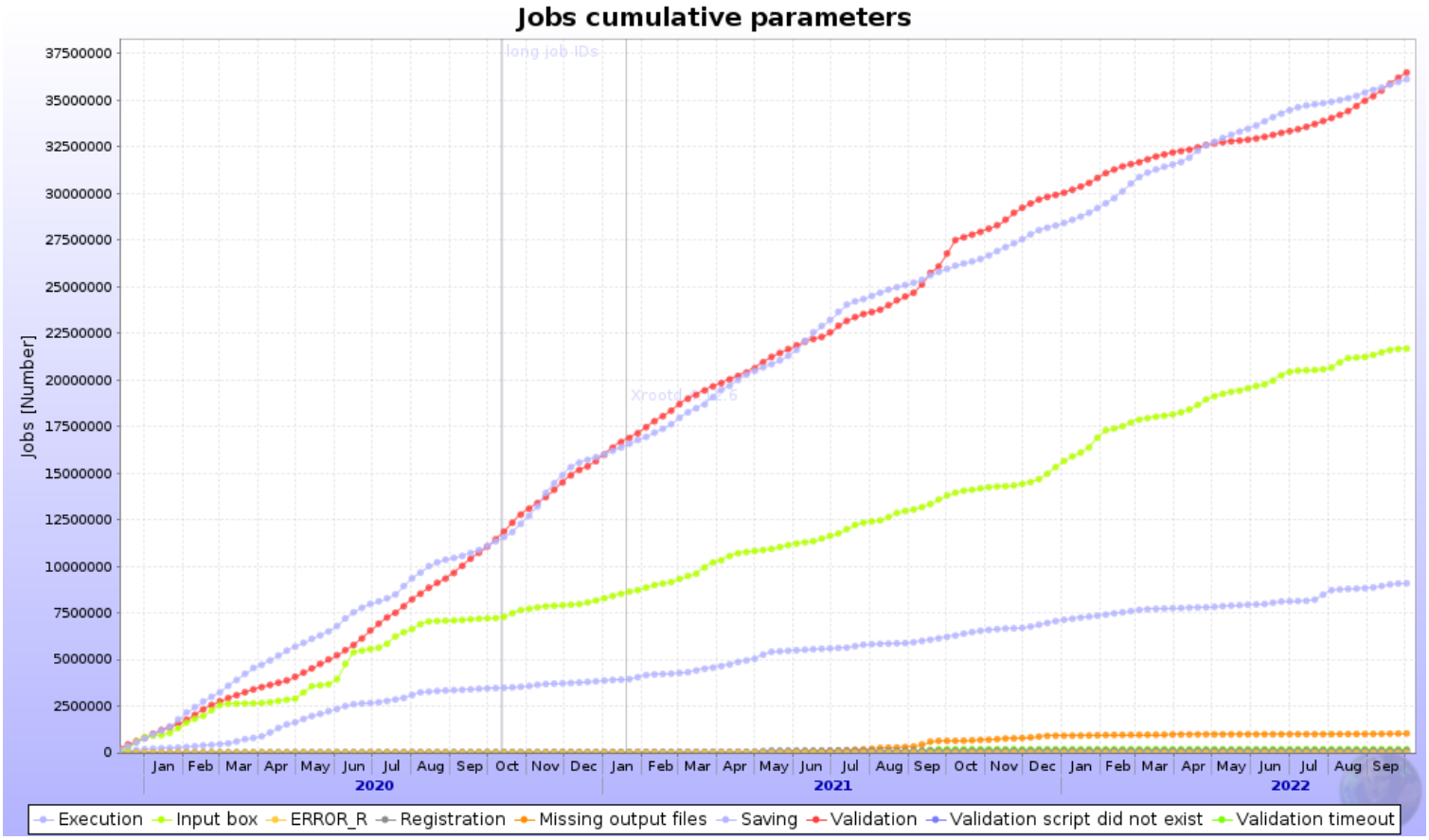


Fig. 11.3: Cumulative number of failed jobs per failure type (states), for the same duration as in Figure 11.1, and Figure 11.2.

Given the above, scalability has thus been a success. Likewise, the solution can be shown to be versatile to the addition of features – not only may new packages and features easily be provided to worker nodes through the use of containers, but the new JAliEn Grid workflow has already been host to multiple extensions of its feature set through parallel developments – such as the CE scripts of Section 9.1.2, Site Sonar of Section 10.1.5, and as will be seen in Section 12.4, support for multicore jobs. In other words, the new Grid workflow has shown to be highly *flexible*.

Administration

Being easier to administer has several implications. For this contribution, this has been defined as something that is simpler to:

- Deploy
- Configure
- Update
- Debug

Looking back at the results from Section 11.2, it becomes evident that the new Grid workflow provides easier administration when defined as above:

- ***Deploy*** – Deployment within an ALICE site now requires minimal action. Given preexisting LDAP entries for a site, the following are now available:
  - Preconfigured VO-Box container and job container
  - Automated CE scripts in CVMFS
  - Required binaries and other files in CVMFS

  In other words, given a recent Linux kernel, a Docker installation and CVMFS, there is a significant reduction in number of steps needed to deploy a full Grid site: launch a VO-Box container and start the CE using the scripts in CVMFS. The CE will thereafter generate JobAgent startup scripts that read and load everything else they need directly from CVMFS.

  The above comes in stark contrast to the previous solution based around AliEn, where the VO-Box setup comprised many more steps. As a rough estimate, such as setup would typically require the following:

  - Installation and initial setup of the VO-Box host
  - Configuration of VO-Box services
    - AliEn services
    - An intermediary HTCondor service, if AliEn will target HTCondor resources
    - A proxy renewal service, if AliEn will target HTCondor or ARC resources
    - VOMS client configuration, ditto
    - A GSISSH service, should the VO-Box be externally accessible
    - Various cron jobs
    - The VO-Box firewall

  In other words, considering each of the above points a "step", at least **8** unique steps (counting the configuration of each service) would be required. Only some of those steps could be done semi-automatically, starting from a VO-Box configuration file. For JAliEn, and the containerised VO-Box, this may now be reduced down to roughly **3** unique steps.

- ***Configure*** – One of the more demanding earlier configuration challenges was to gain a fully working VO-Box, as it requires the configuration of multiple services that need to work in unison. Within the new Grid workflow, this is greatly simplified, as a preconfigured container image is available. Instead of configuring each VO-Box service, there is now mainly need for providing a single configuration: for the container framework (Docker). In the rough estimate above, this reduces the number of VO-Box configuration steps from **7** to just one (**1**).

  A similar argument can also be made for the worker nodes, which use containers by default. To allow ALICE jobs to profit from containers, a worker node needs to allow Singularity to run either from a system installation with a suitable configuration, or from CVMFS, by enabling user namespaces – now done by default on later Linux kernels, including those found in recent versions of CentOS (Section 10.1.4). Once containers are able to launch, the responsibilities for configuration are shifted to the container maintainers. As the desired container image is determined via an invocation of Alienv, the worker nodes need to satisfy the package dependencies of the latter, as was already the case for AliEn.

- ***Update*** – As mentioned in Section 11.2, updating takes minimal effort, as new versions are automatically pushed to CVMFS. This allows the CE to automatically change to the latest version, simply by restarting (reloading Alienv). The same also holds for OS updates, as these are now tied to the container images. Updating the job container applies the change across all containerised worker nodes, which is also done automatically through CVMFS. This implies that where a site admin would previously need to maintain the underlying OS version and packages for the VO-Box and the site worker nodes, no more actions in this regard are now required, as these responsibilities are instead shifted to the ALICE Grid team. What used to be several items to maintain (e.g. **3** "steps" considering maintaining the underlying OS, packages and general configurations as one each, then times the number of WNs on the site), is now instead reduced to **0** – none[7].

- ***Debug*** – Faults, issues and errors, at some point, are arguably inevitable. However, moving towards more homogeneous environments for both VO-Boxes and worker nodes through containerisation, narrows down potential faults, as it can be assumed that the environments for jobs, and across containerised VO-Boxes, are largely the same between sites. This allows easier pinpointing of potential issues: whereas an issue could have arisen from any prior configuration step, and/or due to particularities in how the AliEn software stack behaves on individual WNs, errors are in this case mainly limited to site-specific configurations. Issues pertaining to the configuration of the VO-Box, the WN OS, execution environments and packages can now be debugged centrally. At the very least, these issues are roughly **4** sources of potential errors which need to be considered a lot less by site admins.

Figure 11.4 presents a summary of the number reduced steps/items from the above points, that come as a consequence of switching to the new workflow. It must be

[7] Assuming the underlying host OS will be supported beyond the lifetime of the hardware.

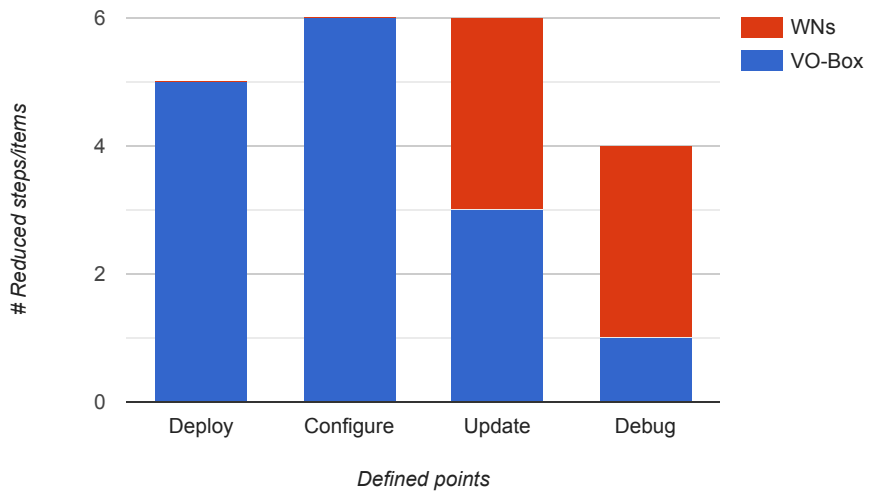


Fig. 11.4: Estimation of the number of reduced steps / items of responsibility for each of the defined points in Section 11.3. Blue represents the number of reduced unique steps / items of responsibility as needed for the VO-Box, while red has to be applied across the cluster / WNs.

emphasised that it is only a rough estimate, and the size and difficulty of each step / configuration task will vary. Items that must be repeated for each WN in the cluster, not just the VO-Box, are coloured red to highlight that these can be particularly demanding. While the numbers of reduced steps may only be estimations, they definitely help creating a Grid workflow that is *easier to administer*.

CHAPTER 12

THOUGHTS AND PERSPECTIVES

Having previously discussed the results and consequences in regards to the original RQ in Chapter 11, the current chapter aims to examine the implications of the achieved workflow within ALICE and beyond. Alternative approaches as achieved by other Grid collaborations will also be examined and put in perspective. Furthermore, related developments and future work will be discussed, before a conclusion is given.

12.1 ALICE in Run 3

As mentioned in Section 1.2.1 and Ref. [102], ALICE has been preparing for Run 3 of the LHC, which started in June 2022 [103]. With the needs of the ALICE collaboration set to greatly increase (Section 1.2), both in terms of storage and raw computing prowess, having a new and modern middleware framework available from the start has been essential.

Throughout the development of this thesis, JAliEn has been brought to a production-ready state, capable of seamlessly executing Grid jobs, and acting standalone, essentially without the need to rely on legacy AliEn services[1] – and has since had a successful deployment both initially within CERN, and throughout the Grid (Section 11.1.2). Post-deployment, it is now the default for job execution across the ALICE Grid, with the transition being largely transparent to end-users – and has also already shown to provide initial performance benefits (Section 11.2). As will be seen in Section 12.4, these benefits are already set to further increase thanks to future work.

12.2 Other novel Grid workflows

As was seen in Section 5.3, other Grid collaborations were likewise investigating means for tighter integration of containers and more novel technologies within their Grid workflows at the time of project start – and have consequently created their own takes. These approaches have their own benefits and drawbacks, with some being more similar to that of ALICE than others.

[1] The exception being in regard to the optimiser, still in development.

12.2.1 ATLAS PanDA

An approach akin to that of ALICE can be found within ATLAS and their Grid middleware PanDA, where the job pilot has likewise been completely rewritten from scratch [104]. Known as *Pilot 2*, it comes with a workflow mechanism that allows it to change run modes based on where the job is being executed, e.g. for HPC. It achieves this by being highly modular, where the provided pilot stack can be used as much, or as little, as needed. This is further simplified though a new component known as the *Harvester*, acting as an in-between service for the job-pilot for external communication and data staging [105]. The pilot will also provide a suitable environment for the payload through containerisation, which is in turn loaded from CVMFS and the CERN container registry – and set depending on the requirements for the job.

The usage of a new, rewritten, job-pilot is reminiscent of the approach taken by JAliEn. However, in-practice, it can likewise be considered the exact opposite: while the PanDA Pilot 2 uses its modular design and containerisation to better cater for a variety of configurations and setups, the new JAliEn JobAgent instead creates a more uniform environment across sites, by offloading core libraries into CVMFS, and thereafter using them to launch a container providing a familiar execution environment across all jobs.

12.2.2 LHCb DIRAC

Like both ALICE and ATLAS, LHCb and their DIRAC stack also provide a new, rewritten, job-pilot [106][107]. Known as *Pilot 3.0*, it launches and runs independent of DIRAC, with no prerequisites for the environment on the worker node. Instead, it will fetch all required DIRAC runtimes and set up the environment – which in turn will launch the JobAgent process responsible for job-matching and monitoring. While this is largely similar to the previous pilot, the new iteration of it can be enhanced through extensions, adapting to specific workflows as needed. The latter is, as with ATLAS and PanDA, intended to better adapt to opportunistic resources and HPCs. In this regard, it also introduces the concept of "inner CEs", which allows jobs to be wrapped in a different user ID, or a Singularity container. The latter not only for the purpose of payload isolation, but also to better adapt the environment – adjusting the OS and DIRAC versions as desired for the user job.

The approach taken by LHCb and DIRAC largely mirrors that of ATLAS and PanDA, with the aim of being able to better adjust jobs and their workflows to different resource setups.

12.2.3 CMS glideinWMS

The glideinWMS framework mainly exists as a thin layer built on top of the HTCondor batch system, and this is likewise reflected in its approach to containerisation. As opposed to any direct pilot or framework changes, Singularity is instead integrated as part of the pilot wrapper script [108]. As an analogue of JAliEn, this would be the startup script mentioned in Section 9.1, but with the logic of probing for compatibility and setting up the runtime, from Section 8.2.3. This provides convenient means for quick integration with Singularity, gaining access to its core features. Specifically, the isolation provided by containerisation is intended as a replacement for gLExec [76],

while its runtime environment facilitates backwards compatibility as sites upgrade to later distributions [109]. Furthermore, and not too unlike JAliEn, sites (and not individual jobs) are allowed to select their own container images, given they are available in CVMFS.

While there are major changes to neither the infrastructure nor the executing pilot, the approach of CMS is in many ways very similar to that of JAliEn: as opposed to adapting to a wide variety of configurations, the pilot instead fully relies on containerisation to provide for both isolation and compatibility – which it attempts to achieve by being both lightweight and having few requirements for running. Furthermore, as opposed to each job being able to select their environment, a smaller set of predefined containers are set on a site level.

12.2.4 *Thoughts on adapting to opportunistic resources*

Being more versatile in regard to the kind of configurations that are supported is a great strength of the approaches taken by both PanDA and DIRAC, in particular for HPC and supercomputers – which have limited compatibility with JAliEn, due to often having very specific configuration requirements. The same also applies for the software, which can be adjusted to always run in a compatible container. However, the approach also introduces more complexity, both in terms of initial setup and maintainability. While this can be offset by larger development teams, such an approach is not scalable for ALICE. Furthermore, giving users the ability to specify any container in a common registry, for each job, paves the way for continued use of deprecated analysis frameworks, in turn creating more fragmentation. While such an approach was considered for JAliEn and the new JobAgent, it was eventually abandoned because of these concerns.

12.3 Impact outside of ALICE

ALICE has designed and developed a new middleware with built-in containerisation from the very beginning, in turn removing the need for various ad-hoc solutions, and further moving the Grid into a more homogeneous direction, as well as reducing the work required from site admins. JAliEn itself is specifically designed to meet the particular needs of the ALICE collaboration, but as now seen in Section 12.2, other Grid collaborations have likewise developed their own takes of how to best incorporate novel technologies. While it remains to be seen which approach proves to be the most beneficial in the long-term, other virtual organisations may take inspiration from the ALICE (and other) experiences in the years to come – in a mutual effort to modernise the Grid as a whole.

12.4 Collaborations and future work

As discussed towards the end of Section 11.3, the new ALICE Grid workflow has been flexible in terms of extending its functionality. Specifically, the development pertaining to this thesis has not been done in a vacuum, but in active cooperation with several other ALICE collaboration members.

It must be noted that JAliEn has since been extended with support for multicore jobs, as required by the new data processing framework that will be used for Run 3. Furthermore, SiteSonar has become an essential tool for monitoring the properties of worker nodes across the ALICE Grid. It has also been extended with web front-ends for configuration of the tests (Ref. [110]) and visualisation of the results.

Within the core functionality, and as briefly mentioned in Section 12.1, the only remaining component not available within JAliEn is the optimiser – the service responsible for translating complex data processing tasks into multiple sub-jobs that in turn can be picked up by worker nodes at Grid sites. Because of its sheer size, developing the JAliEn counterpart is part of a separate project.

12.5 Conclusion and final remarks

Heterogeneous clusters, as found within Grid computing, create challenges by having to cater to multiple system requirements, configurations and deployment practices. Through the use of Cloud-related technologies, such as virtualisation, these challenges have to an extent been alleviated in the past – by "cloudifying" Grid Computing centres, allowing for the creation of more homogeneous environments within an otherwise heterogeneous Grid. This process is however not without its own caveats, requiring ad-hoc solutions and a larger deployment stack.

Containers, in part due to their lightweight aspects, provide means to avoid the above issues. By having them integrated within the next-generation Grid middleware for ALICE, JAliEn, a new Grid workflow has consequently been constructed. Utilising containers at the very core, together with other modern approaches and technologies such as CVMFS, past limitations and shortcomings have been overcome, in turn creating a new workflow that may, as seen in Section 11.3, aid in making ALICE offline computing both more flexible and easier to administer. In addition, it has also enabled the deployment of JAliEn in production in a timely manner – helping ALICE get ready for Run 3 and beyond!

BIBLIOGRAPHY

[1] "Scott McNealy - Quotation." [Online]. Available: http://quotesbox.org/authors/scott-mcnealy-15496_219556/ (Accessed 2022-11-28). I

[2] H. Simon and B. Wilkinson, *Grid Computing - Techniques and Applications*. New York: Chapman and Hall/CRC, 09 2009. [Online]. Available: https://doi.org/10.1201/9781420069549 1, 2.6

[3] B. Turner, *CERN — The European Organisation for Nuclear Research (The Statesman's Yearbook)*. London: Palgrave Macmillan UK, 2003, pp. 72–73. [Online]. Available: https://doi.org/10.1057/9780230271326_20 1

[4] I. Bird, "Computing for the Large Hadron Collider," *Annual Review of Nuclear and Particle Science*, vol. 61, no. 1, pp. 99–118, 2011. [Online]. Available: https://doi.org/10.1146/annurev-nucl-102010-130059 1, 1.2.1, 2.2

[5] K. Potter, "The Large Hadron Collider (LHC) project of CERN," *Proc. ICHEP 96, Warsaw, Poland, July 25-31*, 1996. [Online]. Available: https://doi.org/10.1142/3293 1

[6] The ALICE Collaboration: K. Aamodt et al., "The ALICE experiment at the CERN LHC," *Journal of Instrumentation*, vol. 3, no. 08, pp. S08 002–S08 002, aug 2008. [Online]. Available: https://doi.org/10.1088/1748-0221/3/08/s08002 1, 1.1, 1.2

[7] S. Bagnasco, L. Betev, P. Buncic, F. Carminati, C. Cirstoiu, G. Adriana, A. Hayrapetyan, H. Artem, A. Peters, and P. Saiz, "AliEn: ALICE environment on the GRID," *Journal of Physics Conference Series*, vol. 119, p. 062012, 07 2008. [Online]. Available: https://doi.org/10.1088/1742-6596/119/6/062012 1, 1.2.1, 2.6.1, 2.1

[8] A. Ashraf, M. Hartikainen, U. Hassan, K. Heljanko, J. Lilius, T. Mikkonen, I. Porres, M. Syeed, and S. Tarkoma, *Developing Cloud Software: Algorithms, Applications, and Tools*, 10 2013, ch. Introduction to Cloud Computing Technologies, p. 1–41. [Online]. Available: https://doi.org/10.13140/2.1.1747.8082 1, 3.3, 3.5

[9] V. Falfushinsky, O. Skarla, and V. Tulchinsky, "Integration of Cloud computing platform within Grid Infrastructure," *International Journal of Computing*, vol. 12, pp. 333–339, 2014. [Online]. Available: https://doi.org/10.1109/IDAACS.2013.6663018 1, 1.3.2

[10] D. Berzano, "A ground-up approach to High Throughput Cloud Computing in High-Energy Physics," Ph.D. dissertation, U. Turin, Exp. Phys., 3 2014. [Online]. Available: https://cds.cern.ch/record/2231054 (Accessed 2022-11-30). 1, 4.2, 5.3

[11] V. Silva, M. Kirikova, and G. Alksnis, "Containers for virtualization: An overview," *Applied Computer Systems*, vol. 23, pp. 21–27, 05 2018. [Online]. Available: https://doi.org/10.2478/acss-2018-0003 1, 1.3.2, 3.4

[12] C. Zampolli, "ALICE data processing for Run 3 and Run 4 at the LHC," 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2012.04391 1.2

[13] Beatriz Gay, Luis Gustavo Pereira and Rafael Peretti Pezzi. Computing systems for ALICE/LHC: run3 and beyond. ALICE. [Online]. Available: https://indico.ifsc.usp.br/event/6/contributions/871/ (Accessed 2022-03-27). 1.2

[14] J. Molina, A. Forti, M. Girone, and A. Sciabà, "Operating the Worldwide LHC Computing Grid: current and future challenges," *Journal of Physics: Conference Series*, vol. 513, p. 062044, 06 2014. [Online]. Available: https://doi.org/10.1088/1742-6596/513/6/062044 1.2.1

[15] CERN. The Worldwide LHC Computing Grid (WLCG). CERN. [Online]. Available: https://home.cern/science/computing/grid (Accessed 2022-03-27). 1.2.1

[16] G. von Laszewski and K. Amin, *Middleware for Communications*. John Wiley Sons, Ltd, 07 2005, ch. Grid Middleware, pp. 109 – 130. [Online]. Available: https://doi.org/10.1002/0470862084.ch5 1.2.1, 2.6

[17] H. Jin, *Advances in Web-Age Information Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, ch. Challenges of Grid Computing, pp. 25 – 31. [Online]. Available: https://doi.org/10.1007/11563952_3 1.2.2

[18] H. Lee, "Virtualization basics : Understanding techniques and fundamentals," 2014. [Online]. Available: http://grids.ucs.indiana.edu/ptliupages/publications/virtualization.pdf (Accessed 2022-11-29). 1.3.1, 3.1

[19] Jakob Blomer, Predrag Buncic and René Meusel, "The CernVM File System," CERN, Tech. Rep., 2013. [Online]. Available: https://jblomer.web.cern.ch/cvmfstech-2.1-0.pdf (Accessed 2022-11-29). 1.3.3, 2.5

[20] S. R. Walli, "The POSIX Family of Standards," *StandardView*, vol. 3, no. 1, p. 11–17, mar 1995. [Online]. Available: https://doi.org/10.1145/210308.210315 1.3.3

[21] A. Grigoras, G. Adriana, M. Pedreira, P. Saiz, and S. Schreiner, "JAliEn – A new interface between the AliEn jobs and the central services," *Journal of Physics: Conference Series*, vol. 523, p. 012010, 06 2014. [Online]. Available: https://doi.org/10.1088/1742-6596/523/1/012010 1.3.4, 2.6.3, 4.3.2, 7.1

[22] S.-H. Ha, D. Venzano, P. Brown, and P. Michiardi, "On the impact of virtualization on the I/O performance of analytic workloads," in *2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*, 2016, pp. 31–38. [Online]. Available: https://doi.org/10.1109/CloudTech.2016.7847722 1.3.4

[23] J. Shafer, "I/O Virtualization Bottlenecks in Cloud Computing Today," in *Proceedings of the 2nd Conference on I/O Virtualization*, ser. WIOV'10. USA: USENIX Association, 2010, p. 5. [Online]. Available: https://dl.acm.org/doi/10.5555/1863181.1863186 1.3.4

[24] I. Foster and C. Kesselman, "The history of the Grid," *Advances in Parallel Computing*, vol. 20, pp. 3–30, 01 2011. [Online]. Available: https://doi.org/10.3233/978-1-60750-803-8-3 2.1

[25] The Grid: A system of tiers. CERN. [Online]. Available: https://home.cern/science/computing/grid-system-tiers (Accessed 2022-03-27). 2.2.1

[26] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *The International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, 2001. [Online]. Available: https://doi.org/10.1177/109434200101500302 2.2.2

[27] R. Brun and F. Rademakers, "ROOT: An object oriented data analysis framework," *Nucl. Instrum. Meth. A*, vol. 389, pp. 81–86, 1997. [Online]. Available: https://doi.org/10.1016/S0168-9002(97)00048-X 2.4

[28] P. Buncic, P. Saiz, and A. J. Peters, "The AliEn system, status and perspectives," *CoRR*, vol. cs.DC/0306067, 2003. [Online]. Available: https://doi.org/10.48550/arXiv.cs/0306067 2.4

[29] P. Buncic, C. A. Sanchez, J. Blomer, L. Franco, A. Harutyunian, P. Mato, and Y. Yao, "CernVM – a virtual software appliance for LHC applications," *Journal of Physics: Conference Series*, vol. 219, no. 4, p. 042003, apr 2010. [Online]. Available: https://doi.org/10.1088/1742-6596/219/4/042003 2.5

[30] Pacman headquarters. [Online]. Available: http://atlas.bu.edu/~youssef/pacman/ (Accessed 2022-03-27). 2.5

[31] C. Grigoras and P. Saiz. Torrent-based software distribution. [Online]. Available: https://indico.cern.ch/event/62580/contributions/2064563 (Accessed 2022-03-27). 2.5

[32] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: the Condor experience," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005. [Online]. Available: https://doi.org/10.1002/cpe.938 2.6.1

[33] HTCondor™ Version 8.8.17 Manual. Center for High Throughput Computing, University of Wisconsin–Madison. [Online]. Available: https://htcondor.readthedocs.io/en/v8_8/ (Accessed 2022-03-27). 2.6.1

[34] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60. [Online]. Available: http://doi.org/10.1007/10968987_3 2.6.1

[35] G. Staples, "TORQUE Resource Manager," in *Proceedings of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing, November 11-17, 2006, Tampa, FL, USA*. ACM Press, 2006, p. 8. [Online]. Available: https://doi.org/10.1145/1188455.1188464 2.6.1

[36] (2015) TORQUE Resource Manager - Administrator Guide 4.2.10. Adaptive Computing Enterprises Inc. [Online]. Available: https://docs.adaptivecomputing.com/torque/4-2-10/torqueAdminGuide-4.2.10.pdf (Accessed 2022-03-27). 2.6.1

[37] M. Ellert, M. Grønager, A. Konstantinov, B. Konya, J. Lindemann, I. Livenson, J. Nielsen, M. Niinimäki, O. Smirnova, and A. Wäänänen, "Advanced Resource Connector middleware for lightweight computational Grids," *Future Generation Computer Systems*, vol. 23, no. 2, pp. 219–240, 2007. [Online]. Available: https://doi.org/10.1016/j.future.2006.05.008 2.6.1

[38] J. Mościcki, M. Lamanna, M. Bubak, and P. Sloot, "Processing moldable tasks on the grid: Late job binding with lightweight user-level overlay," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 725–736, 2011. [Online]. Available: https://doi.org/10.1016/j.future.2011.02.002 2.6.1

[39] Martinez Pedreira, M., Grigoras, C., and Yurchenko, V., "JAliEn: the new ALICE high-performance and high-scalability Grid framework," *EPJ Web Conf.*, vol. 214, p. 03037, 2019. [Online]. Available: https://doi.org/10.1051/epjconf/201921403037 2.6.2

[40] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," vol. 44, no. 2, p. 35–40, apr 2010. [Online]. Available: https://doi.org/10.1145/1773912.1773922 2.6.3, 7.3.2

[41] G. Costa. Real-Time processing of Big Data with ScyllaDB. Scylla. [Online]. Available: https://indico.cern.ch/event/730759/attachments/1678516/2695800/Big_Data_CERN.pdf (Accessed 2022-04-04). 2.6.3, 7.3.2

[42] A. Tsaregorodtsev *et al.*, "DIRAC - the distributed MC production and analysis for LHCb," in *14th International Conference on Computing in High-Energy and Nuclear Physics*, 2005, pp. 928–932. [Online]. Available: https://doi.org/10.5170/CERN-2005-002.928 2.6.4

[43] T. Maeno, K. De, T. Wenaus, P. Nilsson, G. A. Stewart, R. Walker, A. Stradling, J. Caballero, M. Potekhin, D. Smith, and (for The Atlas Collaboration), "Overview of atlas panda workload management," *Journal of Physics: Conference Series*, vol. 331, no. 7, p. 072024, dec 2011. [Online]. Available: https://doi.org/10.1088/1742-6596/331/7/072024 2.6.4

[44] I. Sfiligoi, "glideinWMS—a generic pilot-based workload management system," *Journal of Physics: Conference Series*, vol. 119, no. 6, p. 062044, jul 2008. [Online]. Available: https://doi.org/10.1088/1742-6596/119/6/062044 2.6.4

[45] WLCG Authorization Working Group, *WLCG Token Transition Timeline*, Aug. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.7014668 (Accessed 2022-11-29). 2.6.4

[46] T. Maeno, K. De, A. Klimentov, P. Nilsson, D. Oleynik, S. Panitkin, A. Petrosyan, J. Schovancova, A. Vaniachine, T. Wenaus, and D. Yu, "Evolution of the ATLAS PanDA workload management system for exascale computational science," *Journal of Physics: Conference Series*, vol. 513, no. 3, p. 032062, 06 2014. [Online]. Available: https://doi.org/10.1088/1742-6596/513/3/032062 2.6.4

[47] GlideinWMS VO Frontend Installation. [Online]. Available: https://osg-htc.org/docs/other/install-gwms-frontend/ (Accessed 2022-11-28). 2.6.4

[48] L. Arrabito *et al.*, "Application of the DIRAC framework to CTA: first evaluation," *Journal of Physics: Conference Series*, vol. 396, no. 3, p. 032007, dec 2012. [Online]. Available: https://doi.org/10.1088/1742-6596/396/3/032007 2.6.4

[49] G. Borges, I. Campos Plasencia, M. David, C. Sánchez, J. Gomes, A. Lopez Garcia, J. Lopez-Cacheiro, P. Orviz Fernandez, and A. Simón, "Provisioning of Grid Middleware for EGI in the framework of EGI-InSPIRE," in *Proceedings of the IBERGRID 2010*, 01 2010. [Online]. Available: https://digital.csic.es/bitstream/10261/41363/1/egi-global-tasks-mdavid-ibergrid10_v3.1.pdf (Accessed 2022-11-29). 2.6.4

[50] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Wuerthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. Mcgee, and R. Quick, "The Open Science Grid," *Journal of Physics: Conference Series*, vol. 78, no. 1, p. 012057, jul 2007. [Online]. Available: https://doi.org/10.1088/1742-6596/78/1/012057 2.6.4

[51] A. Filipčič, D. Cameron, O. Smirnova, A. Konstantinov, and D. Karpenko, "The Next Generation ARC Middleware and ATLAS Computing Model," *Journal of Physics: Conference Series*, vol. 396, no. 3, p. 032039, dec 2012. [Online]. Available: https://doi.org/10.1088/1742-6596/396/3/032039 2.6.4

[52] A. Dorigo, P. Elmer, F. Furano, and A. Hanushevsky, "XROOTD - A highly scalable architecture for data access," *WSEAS Transactions on Computers*, vol. 4, pp. 348–353, 04 2005. [Online]. Available: https://doi.org/10.37394/23205 2.6.5

[53] T. Mkrtchyan, K. Chitrapu, V. Garonne, D. Litvintsev, S. Meyer, A. Millar, L. Morschel, A. Rossi, and M. Sahakyan, "dCache: Inter-disciplinary storage system," *EPJ Web of Conferences*, vol. 251, p. 02010, 01 2021. [Online]. Available: https://doi.org/10.1051/epjconf/202125102010 2.6.5

[54] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," in *Proceedings of the 5th ACM Conference on Computer and Communications Security*, ser. CCS '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 83–92. [Online]. Available: https://doi.org/10.1145/288090.288111 2.6.6

[55] I. Foster and C. Kesselman, "Globus: a Metacomputing Infrastructure Toolkit," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, no. 2, pp. 115–128, 1997. [Online]. Available: https://doi.org/10.1177/109434209701100205 2.6.6

[56] R. Housley, *Public Key Infrastructure (PKI)*. John Wiley Sons, Ltd, 2004, vol. 3. [Online]. Available: https://doi.org/10.1002/047148296X.tie149 2.6.6

[57] B. Kileng and B. Wagner, "Dynamic virtualization tools for running ALICE grid jobs on the Nordic ARC grid," *NIK-2013*, 2013. [Online]. Available: http://www.nik.no/2013/4-4-dynamic_visualization_tools_for_running_ALICE_grid_jobs_on_the_nordic_ARC_grid.pdf (Accessed 2022-04-23). 2.8.2

[58] J. Surbiryala and C. Rong, "Cloud computing: History and overview," in *2019 IEEE Cloud Summit*, 2019, pp. 1–7. [Online]. Available: https://doi.org/10.1109/CloudSummit47114.2019.00007 3.1

[59] R. J. Creasy, "The Origin of the VM/370 Time-Sharing System," vol. 25, no. 5, p. 483–490, sep 1981. [Online]. Available: https://doi.org/10.1147/rd.255.0483 3.2

[60] J. Meyer and T. Downing, *Java Virtual Machine*. USA: O'Reilly Associates, Inc., 1997. [Online]. Available: https://dl.acm.org/doi/10.5555/265754 3.2

[61] D. Perri, M. Simonetti, and O. Gervasi, "Deploying Efficiently Modern Applications on Cloud," 2022. [Online]. Available: https://doi.org/10.3390/electronics11030450 3.4

[62] E. Casalicchio and S. Iannucci, "The state-of-the-art in container technologies: Application, orchestration and security," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 17, p. e5668, 2020, e5668 cpe.5668. [Online]. Available: https://doi.org/10.1002/cpe.5668 3.4.1

[63] J. McCabe and M. Friis, *Introduction to Windows Containers*. Redmond, Washington (USA): Microsoft Press, 2017. [Online]. Available: https://download.microsoft.com/download/A/1/3/A13A2B9E-D47C-4F93-B180-F7F9CD3382A7/Introduction_to_Containers_ebook.pdf (Accessed 2022-11-29). 1

[64] F.-X. Puig, J. Villalobos, I. Rodero, and M. Parashar, "Exploring the Potential of FreeBSD Virtualization in Containerized Environments," ser. UCC '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 191–192. [Online]. Available: https://doi.org/10.1145/3147213.3149210 1

[65] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux J.*, vol. 2014, no. 239, mar 2014. [Online]. Available: https://dl.acm.org/doi/10.5555/2600239.2600241 3.4.2, 5.2.1

[66] C. Organization. Podman. [Online]. Available: https://podman.io (Accessed 2022-11-29). 3.4.2

[67] D. Gannon and V. Sochat, *Singularity: A Container System for HPC Applications (Cloud Computing for Science and Engineering [Supplementary chapter])*, 06 2017. [Online]. Available: https://cloud4scieng.org/singularity-a-container-system-for-hpc-applications/ (Accessed 2022-04-23). 3.4.2, 5.2.1

[68] Linux Containers. [Online]. Available: https://linuxcontainers.org/ (Accessed 2022-11-29). 3.4.2

[69] C. Grigoras, R. Voicu, N. Tapus, I. Legrand, F. Carminati, and L. Betev, "MonALISA-based Grid monitoring and control," *The European Physical Journal Plus*, vol. 126, no. 1, p. 9, Jan 2011. [Online]. Available: https://doi.org/10.1140/epjp/i2011-11009-9 1

[70] Red Hat Enterprise Linux. [Online]. Available: https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux (Accessed 2022-03-27). 4.2

[71] Scientific Linux. [Online]. Available: https://scientificlinux.org/about/ (Accessed 2022-03-27). 4.2

[72] The end of Scientific Linux. [Online]. Available: https://lwn.net/Articles/786422/ (Accessed 2022-03-27). 4.2

[73] B. Bockelman. Moving CMS to a Container-based Infrastructure. [Online]. Available: https://www.nitrd.gov/nitrdgroups/images/c/c6/CMS_Containers_04042018.pdf (Accessed 2022-03-27). 4.2

[74] F. Megino, J. Albert, F. Berghaus, K. De, F. Lin, D. MacDonell, T. Maeno, R. Rocha, R. Seuster, R. Taylor, and M.-J. Yang, "Using Kubernetes as an ATLAS computing site," *EPJ Web of Conferences*, vol. 245, p. 07025, 01 2020. [Online]. Available: https://doi.org/10.1051/epjconf/202024507025 4.2

[75] Alan Perlis - Quotation. [Online]. Available: http://www.cs.yale.edu/homes/perlis-alan/quotes.html (Accessed 2022-11-29). II

[76] D. Groep, O. Koeroo, and G. Venekamp, "gLExec: gluing grid computing to the Unix world," *Journal of Physics: Conference Series*, vol. 119, no. 6, p. 062032, jul 2008. [Online]. Available: https://doi.org/10.1088/1742-6596/119/6/062032 4, 12.2.3

[77] V. F. Albor, M. S. Miguelez, T. F. Pena, V. M. Muñoz, J. J. S. Silva, and R. G. Diaz, "Cloud flexibility using DIRAC interware," vol. 513, no. 3, p. 032031, jun 2014. [Online]. Available: https://doi.org/10.1088/1742-6596/513/3/032031 5.3

[78] WLCG Containers Working Group Meeting, December 2017. CERN. [Online]. Available: https://indico.cern.ch/event/684575/ (Accessed 2022-03-27). 5.1

[79] GSI-Enabled OpenSSH. [Online]. Available: http://grid.ncsa.illinois.edu/ssh/ (Accessed 2022-11-29). 6.1

[80] (2006) VO-Box Security and Operations Questionnaires. [Online]. Available: https://indico.cern.ch/event/6311/attachments/1007127/1432828/VO-Box-security-policy_v05.pdf (Accessed 2022-03-27). 6.1

[81] Networking using a macvlan network. Docker. [Online]. Available: https://docs.docker.com/network/macvlan/ (Accessed 2022-11-29). 6.3.1

[82] M. Storetvedt, M. Litmaath, L. Betev, H. Helstrup, K. Hetland, and B. Kileng, "Grid services in a box: container management in ALICE," *EPJ Web of Conferences*, vol. 214, p. 07018, 01 2019. [Online]. Available: https://doi.org/10.1051/epjconf/201921407018 6.1, 6.4.2, 6.7, 6.3, 6.4, 6.5

[83] Mahesh Bandewar and Eric Dumazet, "IPVLAN – The beginning," *Proceedings of netdev 0.1, Ottawa, On, Canada*, 02 2015. [Online]. Available: https://legacy.netdevconf.info/0.1/papers/IPVLAN-The-beginning.pdf (Accessed 2022-11-29). 1

[84] Run multiple services in a container. [Online]. Available: https://docs.docker.com/config/containers/multi-service_container/ (Accessed 2022-03-27). 6.4

[85] Supervisor Developers, "Supervisor Documentation," 2018. [Online]. Available: http://supervisord.org/ (Accessed 2022-03-27). 6.4

[86] Migration Planning Guide - Upstart. [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/migration_planning_guide/sect-networking-upstart (Accessed 2022-03-27). 3

[87] Managing Services with SystemD. [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/chap-managing_services_with_systemd (Accessed 2022-03-27). 6.4.2

[88] Keep containers alive during daemon downtime. [Online]. Available: https://docs.docker.com/config/containers/live-restore/ (Accessed 2022-03-27). 6.6.2

[89] M. Pedreira, C. Grigoras, V. Yurchenko, and M. Storetvedt, "The Security model of the ALICE next generation Grid framework," *EPJ Web of Conferences*, vol. 214, p. 03042, 01 2019. [Online]. Available: https://doi.org/10.1051/epjconf/201921403042 7.2

[90] Storetvedt, Maxim, Betev, Latchezar, Helstrup, Håvard, Fanebust Hetland, Kristin, and Kileng, Bjarte, "Running ALICE Grid Jobs in Containers," *EPJ Web Conf.*, vol. 245, p. 07052, 2020. [Online]. Available: https://doi.org/10.1051/epjconf/202024507052 8.1

[91] D. vom Bruch, "Real-time data processing with GPUs in high energy physics," *Journal of Instrumentation*, vol. 15, no. 06, p. C06010, jun 2020. [Online]. Available: https://doi.org/10.1088/1748-0221/15/06/C06010 8.2.4

[92] Environment Modules. [Online]. Available: http://modules.sourceforge.net/ (Accessed 2022-03-27). 9.1.1

[93] G. Eulisse. Building ALICE software stack. [Online]. Available: https://indico.cern.ch/event/457365/#0-prototype-for-common-build-r (Accessed 2022-11-13). 9.1.1

[94] Alibuild. [Online]. Available: https://alisw.github.io/alibuild/ (Accessed 2022-03-27). 9.1.1

[95] K. Matthias and S. P. Kane, *Docker: Up & Running: Shipping Reliable Containers in Production*, 1st ed. O'Reilly Media, Inc., 2015. [Online]. Available: https://dl.acm.org/doi/book/10.5555/2846390 III

[96] A. Semjonov, "Security analysis of user namespaces and rootless containers," Bachelor Thesis, Technische Universität Hamburg, 2020. [Online]. Available: https://doi.org/10.15480/882.3089 10.1.4

[97] Install Singularity. [Online]. Available: https://opensciencegrid.org/docs/worker-node/install-singularity/ (Accessed 2022-03-27). 10.1.4

[98] 7.6 Release Notes. [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/7.6_release_notes/new_features_kernel (Accessed 2022-03-27). 10.1.4

[99] CentOS Linux: Upcoming discontinuation of updates and releases. [Online]. Available: https://access.redhat.com/announcements/6597981 (Accessed 2022-03-27). 7

[100] SL6 EOL. [Online]. Available: https://scientificlinux.org/downloads/sl-versions/sl6/ (Accessed 2022-03-31). 2

[101] Status of proxies and AliEn and LCG tests. [Online]. Available: http://alimonitor.cern.ch/stats?page=proxies (Accessed 2022-03-31). 11.1.2

[102] G. Eulisse, P. Konopka, M. Krzewicki, M. Richter, D. Rohr, and S. Wenzel, "Evolution of the ALICE Software Framework for Run 3," *EPJ Web of Conferences*, vol. 214, p. 05010, 01 2019. [Online]. Available: https://doi.org/10.1051/epjconf/201921405010 12.1

[103] Longer term LHC schedule. [Online]. Available: https://lhc-commissioning.web.cern.ch/schedule/LHC-long-term.htm (Accessed 2022-04-21). 12.1

[104] P. Nilsson, A. Anisenkov, D. Benjamin, D. Drizhuk, M. Lassnig, D. Oleynik, P. Svirin, and T. Wegner, "The next generation PanDA Pilot for and beyond the ATLAS experiment," *EPJ Web of Conferences*, vol. 214, p. 03054, 01 2019. [Online]. Available: https://doi.org/10.1051/epjconf/201921403054 12.2.1

[105] Barreiro Megino, Fernando Harald, Alekseev, Aleksandr, Berghaus, Frank, Cameron, David, De, Kaushik, Filipcic, Andrej, Glushkov, Ivan, Lin, FaHui, Maeno, Tadashi, and Magini, Nicolò, "Managing the ATLAS Grid through

Harvester," *EPJ Web Conf.*, vol. 245, p. 03010, 2020. [Online]. Available: https://doi.org/10.1051/epjconf/202024503010 12.2.1

[106] F. Stagni, A. McNab, C. Luzzi, W. Krzemien, and O. behalf of the DIRAC consortium, "DIRAC universal pilots," *Journal of Physics: Conference Series*, vol. 898, no. 9, p. 092024, oct 2017. [Online]. Available: https://doi.org/10.1088/1742-6596/898/9/092024 12.2.2

[107] A. Tsaregorodtsev. DIRAC Interware Project Status. CNRS. [Online]. Available: https://indico.egi.eu/event/5464/contributions/15909/ (Accessed 2022-11-14). 12.2.2

[108] GlideinWMS configuration – Run jobs under Singularity. Fermilab. [Online]. Available: https://glideinwms.fnal.gov/doc.prd/frontend/configuration.html#singularity (Accessed 2022-11-14). 12.2.3

[109] S. Gadrat. Singularity for CMS. CCIN2P3. [Online]. Available: https://indico.in2p3.fr/event/16538/contributions/57307/ (Accessed 2022-11-14). 12.2.3

[110] E. B. Sandvik, "Site Sonar - A monitoring tool for ALICE's Grid Sites," Master Thesis, The University of Bergen, Bergen, Norway, 2021. [Online]. Available: https://hdl.handle.net/11250/2838363 (Accessed 2022-03-27). 12.4