Experiences with the ALICE Mesos infrastructure

D Berzano¹, G Eulisse¹, C Grigoraş¹, K Napoli²

¹ European Organization for Nuclear Research (CERN), Genève, Switzerland
² University of Malta

E-mail: dario.berzano@cern.ch

Abstract. Apache Mesos is a resource management system for large data centres, initially developed by UC Berkeley, and now maintained under the Apache Foundation umbrella. It is widely used in the industry by companies like Apple, Twitter, and Airbnb and it is known to scale to 10 000s of nodes. Together with other tools of its ecosystem, such as Mesosphere Marathon or Metronome, it provides an end-to-end solution for datacenter operations and a unified way to exploit large distributed systems. We present the experience of the ALICE Experiment Offline & Computing in deploying and using in production the Apache Mesos ecosystem for a variety of tasks on a small 500 cores cluster, using hybrid OpenStack and bare metal resources. We will initially introduce the architecture of our setup and its operation, we will then describe the tasks which are performed by it, including release building and QA, release validation, and simple Monte Carlo production. We will show how we developed Mesos enabled components (called “Mesos Frameworks”) to carry out ALICE specific needs. In particular, we will illustrate our effort to integrate Work Queue, a lightweight batch processing engine developed by University of Notre Dame, which ALICE uses to orchestrate release validation. Finally, we will give an outlook on how to use Mesos as resource manager for DDS, a software deployment system developed by GSI which will be the foundation of the system deployment for ALICE next generation Online-Offline (O²).

1. Layout of the ALICE computing infrastructures at CERN

The ALICE Online and Offline currently operate a number of diversely sized computing clusters located at CERN.

The main ones used for software offline purposes are the Release Validation cluster [1] and the PROOF-based Virtual Analysis Facility [2]: both clusters are virtual and run on resources supplied by the CERN OpenStack instance [3]. Historically they both were bare HTCondor clusters with worker nodes dynamically provisioned by a tool monitoring the batch queue [2][4]. Such virtual resources count up to 400 cores split into four-core VMs having 2 GB of RAM per core.

A more powerful cluster hosting the ALICE central services for the Grid (namely AliEn [5]) on premises is also available. Its gross size is 400 cores split into nodes with a diverse processors layout. Most of those cores are statically bound to the central Grid services and no paradigm was initially foreseen to scavenge unused CPU cycles via dynamic allocation.

Another relevant facility hosted on premises is the DAQ cluster, purposed to test the next generation ALICE software framework for the upcoming LHC data taking period (“Run 3”).
framework, called O², combines online and offline operations [6], and the test cluster will sustain both at the same time.

A separate cluster is the High-Level Trigger [7], used to perform real time data filtering during experiment runs. The cluster is utilized opportunistically outside data taking periods for running Grid jobs using a pilot-based approach with Docker containers [8]. Initially a custom OpenStack setup was used for the same purpose [9].

1.1. Specifications of a modern datacenter
The setup of the aforementioned separate clusters has been carried on with some degree of dynamic provisioning only: long-running services were statically allocated whereas batch workers were started on demand as virtual machines using a IaaS approach.

The experience gathered in using IaaS in HEP computing showed that requesting new virtual machines does not always give a prompt response. On commercial clouds, with a high resources turnaround, the response time is considered acceptable [10], whereas on CERN OpenStack delays have to be expected [11].

In addition to that, automatically scaling farms by monitoring batch queues has become a limitation as ALICE O² is adopting a more elaborated, non-batch workflow model [6].

What we require from a modern ALICE computing facility are two features: the ability to natively manage and scale single tasks, as opposed as whole virtual machines, on a very large set of resources (10⁴ cores) seen as a single entity. Tasks part of different workflows should be also able to opportunistically scavenge unused resources allocated by other workflows.

Such a datacenter should be able to run both single-shot jobs and services: for the latter, having the ability to define availability zones and automatic service redeployment in case of faults are essential.

A final requirement is the ability to gracefully handle both production and development environments at the same time, and multiple users too, with an appropriate set of configurable policies.

2. Apache Mesos in production
We have decided to try to fulfill the above requirements by adopting Apache Mesos in production on part of our computing clusters.

Apache Mesos allows to “program against your datacenter like it’s a single pool of resources” [12]. Mesos works by taking control of your datacenter and constantly monitoring used and available resources. A Mesos-compatible application, which is called a framework using a Mesos-specific terminology, may register to the Mesos cluster and will periodically receive resource offers, which it then decides whether to accept or to reject. Rejected resources are made free for other frameworks to use. This mechanism is called two-level scheduling [13].

Mesos addresses the problem of optimal resources utilization by making sure that all frameworks get their fair share of resources, and by temporarily making a framework borrow the unused resources of another one. On a very busy datacenter this should make sure that all computing resources are always exploited as much as possible.

Mesos was originally developed as a UC Berkeley research project and it is currently being sponsored by the Apache foundation. Many major players in research and IT industry are using it, including Apple, Twitter, Netflix, NASA and Airbnb.

Our experiences of fitting existing and future ALICE applications into the Mesos architecture are covered in Section 3.

2.1. ALICE Mesos cluster setup at CERN
Mesos has a two-level architecture: a certain number of masters control the infrastructure and the registered frameworks, while a large number of agents, one per worker node, are ultimately in charge of deploying, running and garbage-collecting tasks. A single master leads at a given moment in time: the leader is elected using the ZooKeeper quorum mechanism.
Our current production setup counts three Mesos masters running inside CERN OpenStack virtual machines, running in three different availability zones: such placement tackles high availability and makes highly improbable that all three masters become unavailable at the same time.

Mesos masters are treated as “pets”, whereas the agents, disseminated both on CERN OpenStack virtual machines and on the ALICE offline cluster mentioned in Section 1, are treated as “cattle”, i.e. it is not critical for the infrastructure to lose an agent. Mesos functions as the glue making both our virtual machines and bare metal nodes work together as one.

Each service run on Mesos gets an entry in an internal DNS service within the local mesos domain. Since Mesos picks the service location, which is dynamic, a static frontend node had to be set up in order to map back to the sole Mesos services we want to expose to the public network. The next section will describe in detail how this mapping works.

3. ALICE applications running on Mesos

Mesos itself does not provide with a mechanism to natively “submit” existing applications: programs have to be adapted in order to interact with it (by becoming frameworks), or they ought to be executed by an existing framework.

ALICE has adopted three different approaches of running applications on Mesos.

The first and most straightforward way is to rely upon an existing framework. This approach requires no development but only some configuration. Popular existing frameworks include Marathon (covered in Section 3.1) and Jenkins (3.2), which we both use extensively as seen in the next sections.

The other popular option is to develop a framework on your own. This implies programming in one of the languages for which Mesos has bindings, such as Python, C or Java. No language restriction applies by using the Mesos HTTP RESTful API, which is considered easier to use than native bindings. The application now turned into a framework will receive resource offers from Mesos and will have to be structured to make use of them.

A third option is to write an application scheduler based on Mesos, i.e. an application that acts like a framework to Mesos but allows users to run custom structured tasks on top of it. This approach can be compared to writing your own Marathon. Apple has developed a Mesos framework called J.A.R.V.I.S. for driving the operations behind the popular Siri voice assistant [14]. Our custom scheduler is a Mesos plugin for the Dynamic Deployment System [15], used to schedule complex tasks for ALICE O² (section 4.1).

All three approaches with multiple frameworks can work together on a single set of Mesos resources, as Mesos was specifically designed to orchestrate different use cases at the same time. The common denominator of our approaches is completely delegating to Mesos the resource management and deployment part – Mesos becomes a plugin to our applications that bring them an abstract view of the available resources.

This can be compared to what an operating system’s kernel does with processes, which normally do not ask to be run exclusively on a certain CPU or on a certain memory address. For this reason, Mesos is frequently referred to as a “kernel for datacenters”.

Mesos tasks can be executed on several forms: we have decided to deploy all ALICE tasks as Docker containers for simplicity, even though Mesos could schedule them on the bare metal too.

3.1. Marathon

Marathon [16] is a Mesos framework developed by Mesosphere designed to operate long-running services. Marathon comes with a web interface making it easy to deploy new stateless microservices, which in our case are fully self-consistent Docker containers.

Marathon allows to specify the resource configuration allotted to each task in a very fine-grained fashion – even fractions of a CPU can be specified. The most interesting feature is the presence of configurable health checks: Marathon will rely on them to decide whether to automatically redeploy the service in case of failures. Scaling a service is also made easy by the web interface.
In case of multiple instances of a single service are running, they will all be registered under the same host name on the Mesos DNS internal server: clients will use the final addresses in a round-robin fashion by leveraging DNS load balancing features.

Our frontend node maps back to the services we expose to the public network by using a combination of Apache reverse proxies and Traefik [17], a dynamic reverse proxy instantly reacting to service changes in Mesos.

Since defining new production and test services has become very easy in terms of management we currently have several ones running. Examples include: a package repository web frontend, a task periodically generating RPMs out of newly built packages and a health monitoring service for our CVMFS replicas.

3.2. ALICE Continuous Integration

ALICE uses Jenkins [18] for performing several continuous integration tasks, notably automatic pull request checks and daily software builds. Jenkins comes with a Mesos plugin allowing to spawn disposable workers for running the builds on top of a Mesos cluster.

Jenkins has proven to be an excellent testbed for our Mesos infrastructure: given that pull requests can be opened and updated at any time by any user, Mesos has to sustain unpredictable load peaks quite swiftly. In this case the end user is the stakeholder: she has no knowledge of the backend testing her pull requests and she just want to get feedback on her code as fast as possible.

We use Docker containers to deploy Jenkins build tasks in this case as well: the clear advantage is that the build environment is completely decoupled from the runtime environment. As a consequence, we are still able to deliver SLC5 builds even if our infrastructure runs on mixed CentOS 7 and modern Ubuntu flavors. Apart from legacy platforms, new ones can be easily added by leveraging the same principle.

3.3. ALICE Release Validation and Work Queue on Mesos

ALICE runs a release validation for every core software release [1]. Our release validation is a set of interconnected tasks where the output of some jobs might become the input of others. Such dependencies are expressed using the Makeflow tool and language [16] whose syntax deliberately resembles that of Makefiles.

A Makeflow workflow can be submitted on Work Queue workers [20], which are commonly submitted in turn as jobs onto a batch system. In order to integrate the release validation workflow, we have developed the Mesos Work Queue framework [21]: it is a Mesos framework starting Docker containers running Work Queue dynamically based on the effective need. This framework, written in C++, has effectively made us capable of running the release validation on Mesos without needing a batch system at all.

The way Mesos Work Queue works is depicted in Figure 1 and can be described as follows. First off the user starts the usual Makeflow workflow: this spawns an interactive session that will exit only when the last job has completed and all output has been collected. The Makeflow interactive session acts as a Work Queue master: Work Queue workers will connect to it to fetch jobs in a pilot fashion.

The master advertises the number of waiting jobs on a service called Work Queue catalog. Our Mesos framework periodically queries the catalog, and in case there are waiting jobs it starts accepting offers from Mesos accordingly, by effectively starting the workers.

Given that all resource knowledge and deployment features are offered by Mesos, whereas all scheduling and resubmission policies are part of Work Queue, our framework is a mere 200 lines of code making the two tools interoperate. The development effort for such integration was extremely low, and by keeping the code concise we have made its management easy.

This specific use case clearly shows the benefits and costs in investing in Mesos as resource manager: we have moved the release validation from dynamically started virtual machines to an infrastructure that handles several applications at the same time without changing its structure at all (no development was required on the release validation itself), and this was possible by just a little
investment in coding. The release validation still partly runs on the same virtual machine quota as before, as seen in the cluster setup description (see 2.1), but with a better resource usage and prompt scalability.

3.4. Monte Carlo Grid jobs on Mesos
Even if our current production setup is a small fraction of what a Grid site might provide, we have invested some effort in making ALICE Monte Carlo jobs on Mesos, by effectively utilizing Mesos instead of a batch system as a submission backend for AliEn.

Given the simplicity of the Work Queue framework as described in the previous section, we have decided to invest our efforts in making Work Queue work as an AliEn backend by writing the AliEn Work Queue integration[22]: this means that ALICE Grid jobs executed by our Mesos Grid site are effectively submitted as Work Queue jobs, and the Mesos Work Queue framework takes care of orchestrating the provisioning of resources. This is depicted in Figure 2.

3.5. Managing the lifecycle of service containers
As a design decision, every microservice in our infrastructure runs inside a certain Docker container. This has the advantage of easing the development on user laptops before actually shipping the container on the production platform. Containerized services always run in a known and consistent environment and suffer less from compatibility problems.

We have defined a workflow for the development, testing and deployment of new services which is user-friendly, as automatic as possible and allows us to keep track of all changes occurred in a container definition. Our container manifests are stored on a GitHub repository [23] and all changes must come as pull requests: the whole workflow is illustrated in Figure 3.

Our build system automatically picks the pull request and attempts to build the container using Packer [24]. On success, the container is pushed to a Docker registry on a special test repository: this means that the container is available everywhere for downloading and testing.

When we merge the pull request, another automatic container build is triggered, this time pushing the artifact to the production Docker registry. In most cases, we can use the new definition of a service container by simply restarting it from Marathon: an image update attempt is made by default at startup.

4. ALICE Run 3 combined online and offline operations on Mesos
Our current operations serve as an excellent testbed for the Mesos use case foreseen for the next LHC data taking period. The ALICE O² (online/offline) project comprises both a single online/offline
software framework, currently under active development, and shared computing facilities where both online and offline operations are scheduled and executed.

\(O^2\) task definitions are expressed by job topologies and executed by DDS, the Dynamic Deployment System: an overview of DDS along with the integration effort with Mesos is described in section 4.1.

For more generic operation types involving replacing running components at runtime, managing several users and handling both production and development – all of it on the same infrastructure – we are investigating the use of Apache Aurora: this is covered in section 4.2.

4.1. Dynamic Deployment System
DDS, the Dynamic Deployment System [15] is a tool developed at GSI to define and run topologies of interconnected stateful tasks. DDS is also a distributed key-value store used for allowing tasks to share any data: it handles hundreds of values at the same time and propagates the updates throughout the whole topology. DDS exposes an API that allows tasks to subscribe to certain key-value updates by invoking a callback when this occurs. Such updates are known to work up to the scale of thousands of tasks and key-value pairs.

DDS offers a rich language to express the topologies and their configuration, and a set of command-line tools, and has a number of available backends allowing it to deploy tasks on a variety of resource management systems (e.g. Slurm) or even on the bare node (using simple SSH access). Given its plugin-based architecture, we have developed a novel adapter called Mesos DDS [25] that effectively enables DDS to run tasks on resources offered by Mesos.

The tool, written in C++, is composed of two components. The first is the Mesos framework: this service receives offers from Mesos and accepts them based on the submitted DDS tasks. The service exposes a RESTful API to receive tasks and commands. The second component is the Mesos RMS plugin for DDS, allowing DDS users to specify Mesos as resource management system: when submitting a DDS task on Mesos, the plugin talks to the service and asks for resources to be provisioned. A schema is presented in Figure 4.

**Figure 3.** Lifecycle management of our containerized microservices: changes proposals come as pull requests and are deployed to a development repository. Changes are pushed to a production repository upon pull request merge.

**Figure 4.** Interaction of the Dynamic Deployment System with Mesos. The framework accepts Mesos offers based on the number of active topologies submitted to it via a DDS plugin talking to a RESTful API.
Along with the Mesos Work Queue framework presented in section 3.3, this is an additional example of integration of an existing tool with Mesos without breaking the user space: DDS users will keep using the command-line tools as they know them and no modification is required on the task topologies.

4.2. Apache Aurora

Apache Aurora [26] is a Mesos framework suitable both for long running services and for periodic jobs, as well as single-shot ones.

Aurora is capable of handling multiple users and groups via access control lists, quotas and priorities on a single datacenter: important users developing a critical feature might be temporarily given more resources with ease.

Production and development resources and environments, can be handled too: production tasks are always given more priority, and development can opportunistically use part of the production resources.

Aurora comes with a web interface and a command-line tool which we have currently integrated with the CERN Single Sign-On.

Aurora is known to be used by Twitter as a production tool scaling up to 100,000 concurrent tasks running on a single datacenter [27].

5. Conclusions and outlook

We have been happily using Apache Mesos and its ecosystem in production since November 2015. Mesos is currently silently backing most of the offline software production operations: since its introduction we have been able to deliver ALICE software daily tags, release validation and pull request tests without any noticeable service interruption. Notably, in 2016 we have been finally able to deliver 100% of our daily software builds for analysis purposes, and the time to deployment of each build decreased on average by two hours.

The design decision of keeping several Mesos masters living in different availability zones has allowed us to seamlessly maintain Mesos running even when we suffered from network outages or other infrastructural problems. The same applies to the agents, living both on CERN OpenStack virtual machines and on powerful on premises nodes: in case of problems on either infrastructure Mesos has always been able to keep exploiting the other.

As extensively described in section 3 some considerable effort has been put into converting our services to microservices and adapting existing use cases to Mesos. We have tried to maintain the efforts we have made as simple as possible and always shared them with the community for potential feedback [21][22][23][25]. The initial adoption cost almost immediately paid off in terms of stability, reliability and efficient resource usage: the more use cases add, the more we appreciate the value added by Mesos.

Moving our services to Mesos has also been an occasion for rethinking them in a way that the end user never sees the underlying resource management: Mesos operates as a non-invasive thin layer in our infrastructure.

The confidence and experience we have gained by exploiting Mesos in production are proving useful for inspecting a new Mesos-based paradigm for the ALICE next generation software, O²: our efforts are currently being spent in analyzing the exploitation of Apache Aurora and Dynamic Deployment System to deliver infrastructure reliability for combined online and offline operations.

References

[1] D Berzano and M Krzewicki 2015 The ALICE Software Release Validation cluster J. Phys.: Conf. Ser. 664 022006 doi:10.1088/1742-6596/664/2/022006

[2] D Berzano et al. 2014 PROOF as a Service on the Cloud: a Virtual Analysis Facility based on the CernVM ecosystem J. Phys.: Conf. Ser. 513 032007 doi:10.1088/1742-6596/513/3/032007
[3] RM Llamas et al. 2014 Commissioning the CERN IT Agile Infrastructure with experiment workloads J. Phys.: Conf. Ser. 513 032066 doi:10.1088/1742-6596/513/3/032066
[4] https://github.com/dberzano/elastiq
[5] P Saiz et al. 2003 AliEn–ALICE environment on the Grid Nucl. Instr. Meth. Phys. Res. A 502 437-440 doi:10.1016/S0168-9002(03)00462-5
[6] P Buncic, M Krzewicki, P Vande Vyvre 2015 Technical Design Report for the Upgrade of the Online-Offline Computing System CERN-LHCC-2015-006 https://cds.cern.ch/record/2011297
[7] J Lehrbach, V Lindenstruth for the ALICE Collaboration 2017 ALICE HLT Cluster operation during ALICE Run 2 (submitted) https://indico.cern.ch/event/505613/contributions/2236598/
[8] M Concas, D Berzano et al. 2017 Plancton: an opportunistic distributed computing project based on Docker containers (submitted) https://indico.cern.ch/event/505613/contributions/2227997/
[9] M Krzewicki, D Berzano et al. The ALICE High Level Trigger: status and plans 2015 J. Phys.: Conf. Ser. 664 082023 doi:10.1088/1742-6596/664/8/082023
[10] R Taylor et al. 2016 Consolidation of Cloud Computing in ATLAS (poster at CHEP 2016) https://indico.cern.ch/event/505613/contributions/2230743/
[11] D Berzano et al. Lightweight scheduling of elastic analysis containers in a competitive cloud environment: a Docked Analysis Facility for ALICE 2015 J. Phys.: Conf. Ser. 664 022005 doi:10.1088/1742-6596/664/2/022005
[12] http://mesos.apache.org/
[13] B Hindman et al. 2011 Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center NSDI 2011 295-308 http://www.usenix.org/events/nsdi11/tech/full_papers/Hindman.pdf
[14] https://mesosphere.com/blog/2015/04/23/apple-details-j-a-r-v-i-s-the-mesos-framework-that-runs-siri/
[15] http://dds.gsi.de/
[16] https://mesosphere.github.io/marathon/
[17] https://traefik.io/
[18] https://jenkins.io/
[19] http://ecl.cse.nd.edu/software/makeflow/
[20] http://ecl.cse.nd.edu/software/workqueue/
[21] https://github.com/alisw/mesos-workqueue
[22] https://github.com/alisw/alien-workqueue
[23] https://github.com/alisw/docks
[24] https://www.packer.io/
[25] https://github.com/alisw/mesos-dds
[26] http://aurora.apache.org/
[27] https://blog.twitter.com/2016/overview-of-the-twitter-cloud-platform-compute