Managing software build infrastructure at ALICE using Hashicorp Nomad

Timo Wilken[1,*] and *Giulio* Eulisse[1]

[1]The ALICE Collaboration, CERN, 1211 Genève 23, Switzerland

Abstract. The ALICE experiment at CERN uses a cluster consisting of virtual and bare-metal machines to build and test proposed changes to the ALICE Online–Offline (O^2) software in addition to building and publishing regular software releases.
Nomad is a free and open-source job scheduler for containerised and non-containerised applications developed by Hashicorp. It is integrated into an ecosystem of related software, including Consul and Vault, providing a consistent interface to orchestration, monitoring and secret storage. At ALICE, it recently replaced Apache Mesos, Aurora and Marathon as the primary tool for managing our computing resources.
First, we will describe the architecture of the build cluster at the ALICE experiment. After giving an overview of the advantages that Nomad gives us in managing our computing workload, and our reasons for switching away from the Mesos software stack, we will present concrete examples of improvements in monitoring and automatic configuration of web services that we are already benefiting from. Finally, we will discuss where we see opportunities for future work in integrating the ALICE build infrastructure more deeply with Nomad, in order to take advantage of its larger feature set compared to Mesos.

1 Introduction

The ALICE experiment at CERN [1] develops the O^2 software suite [2, 3], which enables data-taking, detector calibration, reconstruction of events, Monte Carlo simulation and physics analysis [4] for LHC Run 3 and later. In addition, the Run 2 framework [5, 6] is still maintained to facilitate the analysis of older data. The ALICE central offline team is responsible for:

- automated compilation, unit testing and integration testing to provide Continuous Integration of proposed code changes to the maintained software
- releasing and managing the versioning of binaries for use in production and for daily integration tests
- Continuous Deployment, by publishing released builds for use in offline physics analysis and online data taking, built on multiple versions of CentOS and Alma Linux
- supporting members of the ALICE Collaboration by providing re-usable build artifacts to them, for use in local tests, on systems running CentOS, Alma Linux, Ubuntu, OSX and more

[*] e-mail: timo.wilken@cern.ch

- keeping relevant documentation up-to-date, in part by automatically regenerating documentation from source code comments

In order to make builds more reproducible and to avoid issues with differing versions of build dependencies on users' systems, an internally-consistent collection of these dependencies is provided [7]. This collection specifies the versions and sources of build dependencies to use when compiling O^2 software, in addition to compilation scripts. It is designed to be installed on top of an existing operating system supported by the ALICE Collaboration, such as CentOS or Alma Linux. It is versatile and well-tested enough to work on operating systems as diverse as Ubuntu and OSX, which makes it suitable for use by collaboration members on their machines.

The source code of the O^2 software suite is split across multiple repositories, most of them hosted on GitHub, with a small proportion on a GitLab instance hosted at CERN. This allows using GitHub "pull requests" in order to manage proposed code changes. Since the O^2 project's code was moved to its current location on GitHub in 2014, more than 14,000 pull requests have been received and processed in the most active repositories combined.

For each pull request, the applicable software is compiled and its built-in unit tests are run. In addition to this, integration checks are run in order to validate certain parts of the complete software stack, such as Monte Carlo simulation. One of these checks completes, on average, every two minutes. Additionally, 50 pull requests per hour are checked for code changes, so that updated pull requests can be rebuilt, and existing pull requests are always checked against the latest upstream code, even when this code changes.

These pull request checks run on five different platforms: CentOS, Alma Linux, Ubuntu, x86-based and ARM-based OSX. The diversity of Linux distributions being tested is facilitated through containerisation using Docker [8, 9].

2 Build cluster architecture

In order to satisfy the above use-cases, the ALICE central offline team administrates a cluster of 35 virtual machines, in addition to a handful of physical machines running various distributions of Linux and OSX. These resources total 600 CPU cores and 1.7 TiB memory.

The virtual machines are provided by an OpenStack instance at CERN [10] and managed centrally, with the ALICE Collaboration having full access to the machines themselves and being able to manage their configuration through standardised tools such as Puppet [11]. Physical machines are sometimes configured manually, but can also be centrally configured in a similar way.

Figure 1 gives an overview of the different software components in use as part of the ALICE Continuous Integration and Continuous Deployment infrastructure. We run different types of computing jobs as part of the infrastructure. In order to manage these jobs, allocate computing resources to them, and control their lifecycles, we use Nomad [12]. The different types of computing jobs are represented in the top-right of Figure 1 connected to the "Nomad" node with unlabelled arrows.

The largest proportion of these jobs in terms of computing resources are build jobs, which compile and test incoming pull requests as described in Sect. 1. In addition, we run web services for purposes such as user account administration, and web servers providing compiled application binaries to users on request, for use on their local machines. A few scheduled maintenance jobs also run, which control the size of the repository of compiled software, for instance.

As many of these jobs require access to passwords, API tokens and other secrets, we run Vault [13]. Vault provides this data to Nomad on request in an authenticated manner, such

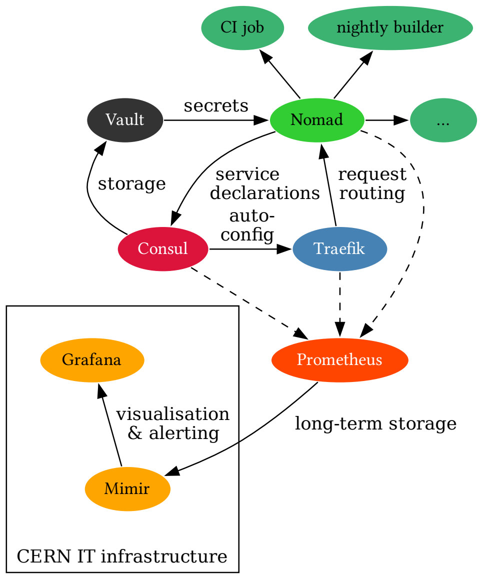

Figure 1. Architecture diagram. Solid arrows represent data and/or control flow; dashed arrows represent metrics extraction. Nomad, Consul and Vault are designed to work in tandem, with Nomad as the centre-point configuring other services (such as Træfik) through Consul. Metrics of the entire cluster are collected using Prometheus and routed to CERN infrastructure for visualisation and alerting.

that administrators only need to know their names, and Nomad will substitute their values where required.

In addition, we use Consul [14] as a DNS server, since Nomad jobs are automatically assigned internal domain names ending in `.service.consul`, and as a generic key/value store containing configuration data for other services, such as Træfik. Consul also has job monitoring features in the form of health checks, described in more detail in Sect. 3.2.2, and serves as the underlying data store for Vault.

Træfik [15] is an HTTP proxy, which can accept web requests from the internet and forward them transparently onto internal services. It can be configured through Consul, which avoids any manual work in configuring it for individual web services. Our use of it is further described in Sect. 3.2.2.

Nomad, Consul and Vault are open-source software developed by Hashicorp. They are designed to complement each other, and integrate deeply with each other, as shown above.

The entire cluster is monitored using Prometheus [16], which feeds metrics into monitoring and alerting infrastructure provided centrally by CERN [17]. Here, Mimir stores metrics and forwards them to the visualisation and alerting software Grafana.

3 Switching to Nomad

3.1 Previous architecture

Before switching to Nomad, the ALICE Continuous Integration infrastructure was based on Apache Mesos, Apache Aurora and Apache Marathon [18].

This architecture was starting to show its age, with Aurora especially not being intensively developed anymore, and thus still requiring Python version 2.7 to be installed both on servers and on administrators' machines. As this version reached its end of life in 2020, it became increasingly difficult to install and deploy, leading to unacceptable amounts of friction in administrating the build cluster.

The fact that development on Aurora had slowed meant that certain improvements were unlikely to be integrated in future. This includes better autoscaling support for Aurora jobs—

```
service {
  name = "${JOB}"
  port = "http"
  tags = [
  # Strip a /path prefix off the URLs passed to this service.
  "traefik.http.routers.job.rule=Host(`example.org`) && PathPrefix(`/path`)",
  "traefik.http.routers.job.middlewares=job-stripprefix",
  "traefik.http.middlewares.job-stripprefix.stripprefix.prefixes=/path",
  ]
}
```

Listing 1. Example of automatic configuration of Træfik through Nomad service definition.

the existing implementation required the restart of running instances in order to add new ones, which would delete any locally-cached build artifacts and slow down subsequent builds significantly. Since the software predated widespread adoption of monitoring standards like Prometheus, it was also difficult to set up monitoring and alerting for failures, and integrating it with the existing monitoring solution at CERN.

3.2 Improvements realised when switching to Nomad, Consul and Vault

Switching to a software stack based on Nomad, Consul and Vault allowed us to immediately realise several improvements, specifically in its deployment, its support for web services and monitoring. These are described in the following sections.

In addition to these improvements, we also benefit from more secure and better-integrated secrets management through Vault, since Mesos and Aurora have no integrated secrets management feature.

This only includes the features that we could adopt without any code changes to the ALICE Continuous Integration software itself. Potential improvements that would take more work to adopt are described in Sect. 4.

3.2.1 Deployment

The deployment of Nomad, Consul and Vault is simple: for Nomad and Consul, all that is needed on each build node is a static binary, a single configuration file for systemd (or launchd on OSX), and a configuration file for the software itself, primarily specifying how to connect to the rest of the cluster. Vault only needs a server node for Nomad agents to connect to; it is not deployed on each build node.

The deployment of the server nodes for Nomad, Consul and Vault is similarly straightforward, even with a High-Availability setup where multiple instances of each are running simultaneously. Each is deployed using the standard Puppet configuration management software in use at CERN.

3.2.2 Web service autoconfiguration

Out of all features of Nomad specifically intended for use by web services, we most rely on autoconfigured web request routing and health checks.

Listing 1 shows part of a Nomad configuration file for a computing job, which registers the job as a web service with Consul. This web service has “tags” attached to it, starting with

```
service {
  check {
    type = "http"
    port = "http"
    path = "/health"
    interval = "20s"
    timeout = "5s"
    initial_status = "warning"
  }
}
```

Listing 2. Example of automatic configuration of Consul health checks through Nomad service definition.

`traefik.`, which represent statements in Træfik's configuration syntax. This causes Træfik to read them and configure itself accordingly. In this particular case, these tags cause Træfik to route web requests for a particular hostname and an HTTP path starting with `/path` to the Nomad job that contains this code.

In this way, the host, IP address and port number of the running job can be entirely determined by Nomad. Nomad passes the information of the running job onto Consul, which stores it and passes it to Træfik. This allows web requests for a particular job to be routed transparently, with no manual intervention required from administrators, and with minimal downtime if the Nomad job fails and is rescheduled on another machine.

3.2.3 Monitoring improvements

Listing 2 shows an example of a Consul health check configured through a Nomad job definition. This health check runs periodically after the Nomad job definition is deployed. In this case, an HTTP request is made to the job's assigned IP address and port, with the path `/health`. This is expected to return a successful HTTP status; if it does not, this is signalled to Prometheus, and the monitoring stack sends an alert.

This arrangement allows monitoring web services that are not necessarily written to support detailed metrics extraction, since any endpoint that is always expected to return success can be used for such a check. Checks such as this allow downtime to be tracked in a fine-grained way, and operators to be alerted quickly when a service becomes unavailable.

Consul also has support for health checks that only connect to the job on a certain port to make sure that a server is bound there, or checks that run an arbitrary command on the host where a particular job is deployed – even inside the job's running container.

In addition to network-based health checks, Nomad also supports monitoring jobs in other ways. One of the most useful in our use-case, for build nodes, is to monitor their resource use in order to find inefficiencies or pinpoint resource bottlenecks during compilation.

3.2.4 Monitoring case study

Figure 2 shows a trace of some build nodes' processor use, expressed as a proportion of the processor resources allocated to them. This data shows that some build nodes, specifically running between 02:00 and 05:00, used fewer CPU resources than allocated, while other build nodes generally used 100% of their allocated resources.

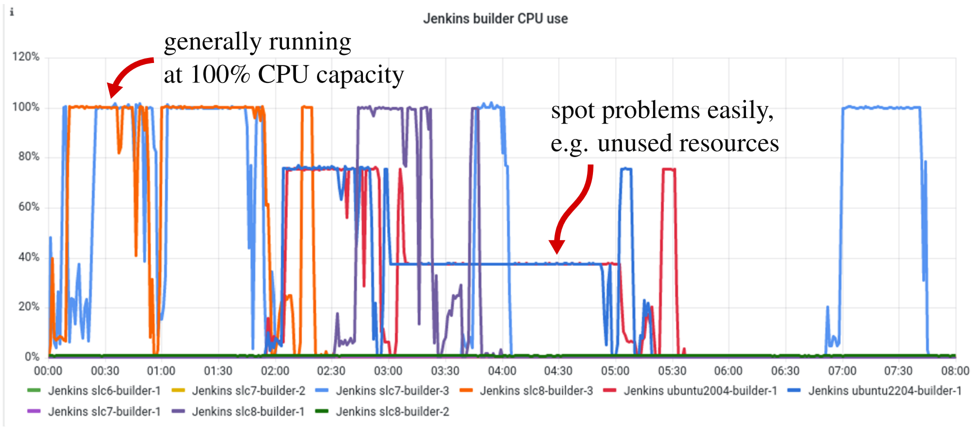

Figure 2. Processor (CPU) resource use, expressed as a proportion of CPU resources allocated to different build nodes, recorded during one night. 100% generally means 16 CPU cores. Some builders running between 02:00 and 05:00 use only part of their assigned CPU resources, which can be seen easily in metrics collected via Nomad.

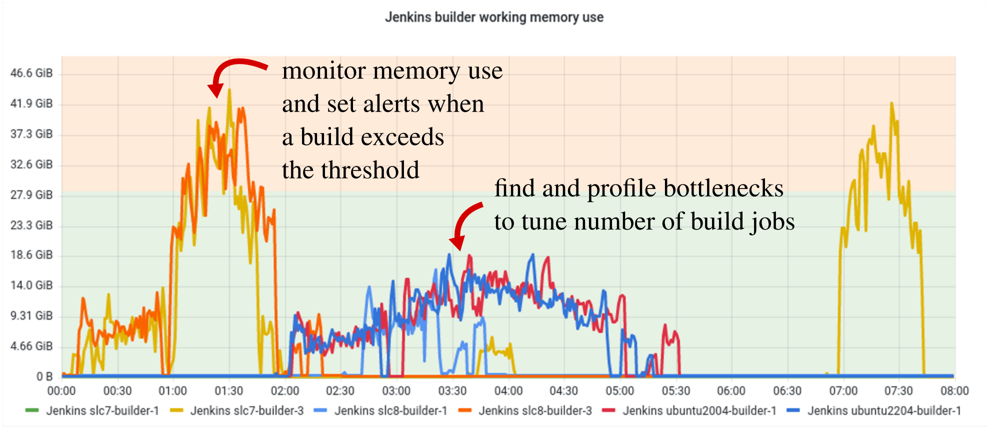

Figure 3. Total working memory use during the same time as in Figure 2. Standard build virtual machines have 28 GiB of memory. The build nodes that used only part of their assigned CPU quota (between 02:00 and 05:00) are using only part of their memory quota as well.

Metrics collected in this way allow us to easily find cases of suboptimal resource use, like the one mentioned above. In this particular case, these build nodes were intentionally using fewer processor cores than others, as their workload would exhaust the available memory otherwise.

Using the data in Figure 3, the memory bottleneck of the affected build nodes can be found – in this case, this was still below our virtual machines' memory size of 28 GiB. Using metrics in this way, we can scale up the number of CPU cores used slightly, until memory use is optimal, and adjust the remaining allocated resources to fit, leaving more unallocated resources for other jobs.

In particular, some of the first builds to run required more memory than was available on a typical virtual machine, but were still run at 16 cores. This was done because these builds had a higher priority and should finish sooner. Nomad allows us to change the allocated resources per builder, for instance depending on its platform, such that high-priority builds are assigned more, and are run on physical machines with more memory available.

In this case this memory overrun was intentional, but in case other builders ran out of memory assigned to them, the metrics collected from Nomad would allow us to set alerts when this happens, notifying cluster operators immediately.

4 Future work

While the Nomad setup as implemented is a clear improvement over the previous cluster setup, opportunities remain for further improvement in some areas.

Nomad does not track a job's disk space usage during the runtime of the job. This means that Nomad's accounting of allocated disk space can fall out of sync with actual disk usage, particularly when the controlling Nomad process is restarted on a build node. Future work may aim at changing this behaviour and offering this change to the upstream maintainers.

Nomad, Consul and Vault might be integrated with the Single Sign-On (SSO) solution in use at CERN. This would involve users signing in to Vault using their SSO credentials, with Vault then creating login tokens for the user, which they use to sign in to Nomad and Consul. This would be particularly useful if more people become cluster operators.

Autoscaling has also become much easier to implement with Nomad compared to Aurora. While manual scaling is already much better, causing much less downtime and cache loss, implementing autoscaling with a minimum of downtime still requires some cooperation from the Continuous Integration program launched by Nomad. Future work will aim at making this seamless, perhaps even automatic, allowing the number of build nodes for each platform to be scaled according to demand. Invalidation of large (multi-GiB) build caches is likely to be the main hurdle to this effort.

Consul, being a generic key/value store, may be used to store configuration data for build nodes. This would allow rolling out temporary deployments more easily, in order to test changes to the Continuous Integration program itself. Consul would provide a convenient web interface for changing any configuration values by operators.

Finally, by integrating more closely with Vault, the limited lifetime of secrets in the Continous Integration environment might be further reduced, with secrets only fetched on demand and removed from the environment immediately afterwards. This would represent a slight security improvement.

References

[1] The ALICE Collaboration, K. Aamodt, A.A. Quintana, R. Achenbach, S. Acounis, D. Adamová, C. Adler, M. Aggarwal, F. Agnese, G.A. Rinella et al., *The ALICE experiment at the CERN LHC* (2008), Vol. 3, p. S08002, `https://doi.org/10.1088/1748-0221/3/08/S08002`

[2] P. Buncic, M. Krzewicki, P. Vande Vyvre, Tech. Rep. CERN-LHCC-2015-006, ALICE-TDR-019 (2015), `https://cds.cern.ch/record/2011297`

[3] ALICE Collaboration, *AliceO2Group/AliceO2: 4 December 2023 async release, revision 4* (2023), `https://doi.org/10.5281/zenodo.10303764`

[4] A. Alkin, G. Eulisse, J.F. Grosse-Oetringhaus, P. Hristov, M. Kabus, *ALICE Run 3 Analysis Framework* (2021), Vol. 251, p. 03063, https://doi.org/10.1051/epjconf/202125103063

[5] D. Rohr, M. Krzewicki, H. Engel, J. Lehrbach, V. Lindenstruth, for the ALICE Collaboration, *Improvements of the ALICE HLT data transport framework for LHC Run 2* (IOP Publishing, 2017), Vol. 898, p. 032031, https://doi.org/10.1088/1742-6596/898/3/032031

[6] ALICE Collaboration, *alisw/AliPhysics: v5-09-59s-01 release* (2023), https://doi.org/10.5281/zenodo.10303830

[7] ALICE Collaboration, *alisw/alidist: 4 December 2023 async O2 tag, revision 4* (2023), https://doi.org/10.5281/zenodo.10303794

[8] D. Merkel, *Docker: Lightweight linux containers for consistent development and deployment*, https://dl.acm.org/doi/10.5555/2600239.2600241 (2014)

[9] Docker Inc, *Docker (version 24.0.5)* (2023), https://docker.com/

[10] R.M. Llamas, F.H.B. Megino, K. Kucharczyk, M.K. Denis, M. Cinquilli, *Commissioning the CERN IT Agile Infrastructure with experiment workloads* (2014), Vol. 513, p. 032066, https://doi.org/10.1088/1742-6596/513/3/032066

[11] Puppet Contributors, *Puppet (version 7.12.0)* (2021), https://github.com/puppetlabs/puppet

[12] Hashicorp, *Nomad (version 1.5.3)* (2023), https://nomadproject.io/

[13] Hashicorp, *Vault (version 1.12.3)* (2023), https://vaultproject.io/

[14] Hashicorp, *Consul (version 1.14.4)* (2023), https://consul.io/

[15] Træfik Labs, *Træfik (version 2.6.7)* (2022), https://traefik.io/traefik/

[16] Prometheus Authors, *Prometheus (version 2.36.2)* (2022), https://prometheus.io/

[17] A. Aimar, A.A. Corman, P. Andrade, S. Belov, J.D. Fernandez, B.G. Bear, M. Georgiou, E. Karavakis, L. Magnoni, R.R. Ballesteros et al., *Unified Monitoring Architecture for IT and Grid Services* (IOP Publishing, 2017), Vol. 898, p. 092033, https://doi.org/10.1088/1742-6596/898/9/092033

[18] D. Berzano, G. Eulisse, C. Grigoraş, K. Napoli, *Experiences with the ALICE Mesos infrastructure* (IOP Publishing, 2017), Vol. 898, p. 082043, https://doi.org/10.1088/1742-6596/898/8/082043