# Containerized Batch System Monitoring

*Andreas* Gellrich[1], *Thomas* Hartmann[1]*, *Birgit* Lewendel[1], *David* Prelogović[2], and *Christian* Voss[1]**

[1]DESY, Notkestraße 85, D-22607 Hamburg, Germany
[2]University of Zagreb, Croatia

**Abstract.** Running a batch system for grid jobs and for local users, we investigate a generic solution to monitor the resource usages of jobs. We extend a standard toolkit for monitoring container performance metrics also for non-containerized applications, so that we can make easy use of a popular industry solution. By concentrating on the basic kernel features also used by containers frameworks, we envisage to use the same tools on non-containerized batch systems as well as container orchestrators.
For deployment, we encapsulate the tools as Singularity containers and distribute them via CVMFS.

## 1 Introduction

Since microservices in the form of containers have become en vogue, various monitoring tools have been developed. While such tools are primarily focused on Docker as the framework with the largest following, these tools can also be used for a more generic approach.
As Containers are based on standard Linux kernel features employed to encapsulate processes in Linux namespaces and cgroups, container monitoring tools can be reused more generically for any kernel resources in cgroups. While containers and container orchestration frameworks are becoming established as Local-Resource-Management-Systems (LRMS) alongside traditional batch systems, batch systems as HTCondor [1, 2] or SLURM [3] are prevalent in the HEP world, where workloads of the Grid computing communities have evolved along these. Since nowadays both, batch systems and container frameworks, use the same kernel features for resource management and control, it becomes feasible to refit and establish off-the-shelf monitoring tools for existing LRMSes. We chose Google's cAdvisor [4] as lightweight tool without further dependencies and a rich REST API.
We distribute cAdvisor and our Logstash-based aggregation script as Singularity [5] containers via CVMFS [6] for an easy and fast deployment along the lines of established system requirements in the grid world.
Aiming for portability, configuration parameters are controlled via environment variables.

*e-mail: thomas.hartmann@desy.de
**e-mail: christian.voss@desy.de

While the container for aggregating job statistics is primarily aligned to HTCondor, support for other batch systems, which use cgroups for resource management, could also be added to the aggregation container or handled by a separate aggregation tool. For easier interchangeability of database engines, we stick to JSON formatted data [11] to structure job details. Locally, we use ElasticSearch [14] for information storage and information retrieval, which itself uses Apache Lucene as back-end [13].

## 2 Motivation

DESY operates HTCondor[1] clusters with ∼15 000 cores dedicated to grid workflows as well as ∼9 000 cores for batch processing by users in the `National Analysis Facility (NAF)` [7, 8].
Where the grid aims for large scale bulk processing of data and Monte Carlo event simulations, the NAF is targeted on a more interactive work-flow with a fast turn-around of user jobs. Storage back-ends are based on several dCache instances[2] [10] for long-term storage and on a fast scratch space called `DUST` based on IBM's Spectrum Scale file system [9].
For both use cases, visual job profiling simplifies the analysis and debugging of performance and failures. For the bulk production in most of the Grid workflows, the local site admin can gather a quick overview of failing job tasks from the experiments by mapping the global job IDs in the experiment workload management systems to the local batch systems. E.g., figure 1 shows the job memory profiles of an active node, which were used to identify the jobs that sent their host node into swapping.
Equivalently in the NAF use case, individual users could generate an overview of their jobs spread over time as well as over batch system nodes instead of working through all jobs' logfiles to gather an overview of the jobs' performances (provided that some kind of pilot process had been logging the performance alongside the actual job).

## 3 Data Aggregation

We use cAdvisor to gather basic system statistics on process trees, which are confined in cgroups, i.e., batch jobs and their child processes for which the HTCondor daemon created separate cgroup sub-slices. cAdvisor itself can export the current system statistics as JSON objects. Per node, we have cAdvisor read the job statistics and add to cAdvisor's JSON object our own additional information, which we gathered from the Condor batch system. The expanded job statistics are then sent to our central Elastic Search[3] [14] cluster and are visualized with Kibana as UI.

[1]HTCondor on development version 8.7.8
[2]instances at versions 4.2.X, being updated on a regular basis to current release versions
[3]Elastic Search cluster on version 6.2.4

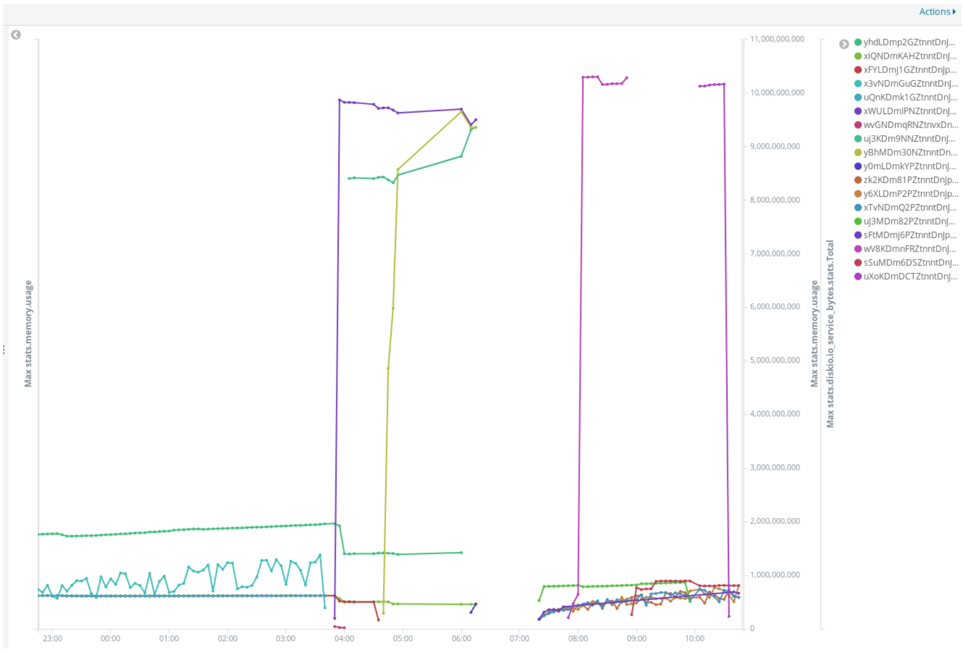


**Figure 1.** Memory usage profiles of grid jobs, identified here by their compute element's ARC ID, on a node, which started to swap. Kibana was used for visualization of the data from an ElasticSearch database.

## 3.1 cAdvisor: cgroup Statistics

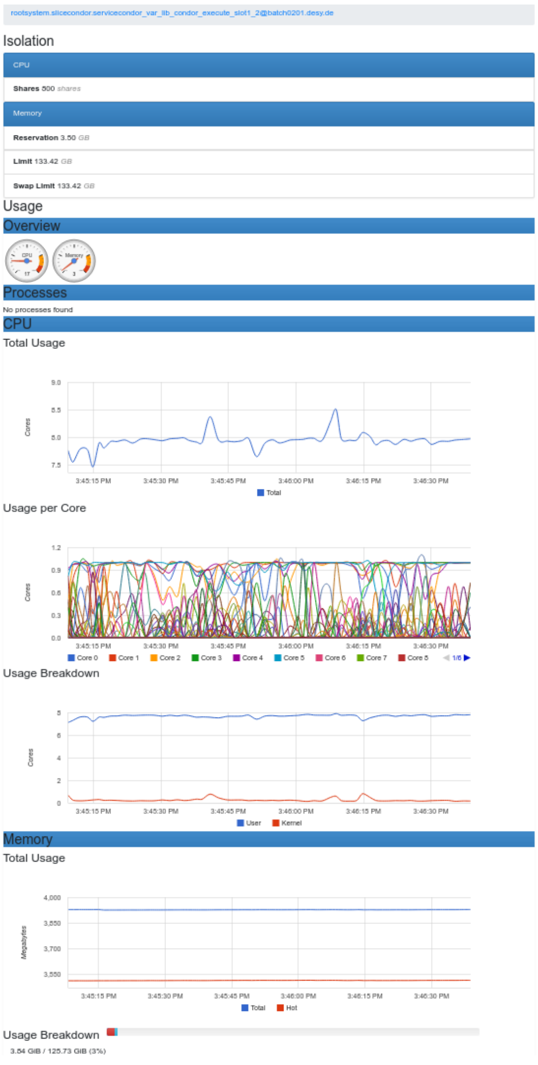
As cAdvisor already brings its own driver for exporting statistics to Elastic Search, in the most basic approach cAdvisor could gather and forward cgroup metrics to the database. But as we have potentially additional interesting knowledge at hand on a node, e.g., the batch system's job details, we intercept the metrics before storing them.
Since cAdvisor is primarily aimed at monitoring containerized applications, it is deployed itself as Docker [16] container. Since we do not necessarily use Docker as a container engine and consider the installation and maintenance of Docker on batch nodes an unnecessary potential security risk due to the Docker daemon running under the root context, we decided to deploy cAdvisor as Singularity container[4] due to its more lightweight approach.
The conversion of the cAdvisor docker container into a Singularity container is straight forward, as the actual cAdvisor application is a Go binary with hardly any dependencies on the container OS and environment[5]. cAdvisor runs as a Singularity container without problems and we applied only minor tweaks to the container's environment settings, optimizing it for our use case. Since cAdvisor has to have read access to all process information in the `/proc` and `cgroup` virtual file system mounts,

[4]Singularity version 2.6.1 installed on batch system nodes
[5]We based the Singularity container on the latest production release of cAdvisor on Dockerhub, which was v0.27.4 during writing

the container has to be run in the root context[6] with cgroup bind-mounted in the container's namespace[7].
In addition to its JSON export, cAdvisor can also publish the the current system status and cgroup resource usages via an integrated web server as shown in Figure 2. Thus, cAdvisor can also be used for a quick overview on a node's current utilization.
In production, we run the cAdvisor container as systemd service started after the network and HTCondor services as requirements.

### 3.2 Batch System Parameters

Since cAdvisor is agnostic towards container engines or other cgroup handlers, the raw cAdvisor statistics do not contain any batch system job specifics. Thus, we extend cAdvisor's raw cgroup statistics by batch job details.
We wrote a Python script to aggregate cAdvisor and batch system statistics. The script reads HTCondor batch job information details from the file system as well as the jobs process details created at start under `/proc/{PID}/`. This information is combined with the cAdvisor cgroup details. The script itself is executed by a Logstash service, reads cAdvisors's metrics for the aggregation script, and forwards the resulting enriched job infos to a Elastic Search instance.
We wrap Logstash and the aggregation script in another Singularity container. Executed regularly[8], the container will try to read the current statistics from the running cAdvisor container, process them and forward the extended JSON object. The container can be configured via environment variables, e.g., the address of the ElasticSearch cluster, so that no local configuration files are necessary and the container can be run out-of-the-box after setting the environment variables.


rootsystem.slicecondor.servicecondor_var_lib_condor_execute_slot1_2@batch0201.desy.de

Isolation

CPU

**Shares** 500 *shares*

Memory

**Reservation** 3.50 *GB*

**Limit** 133.42 *GB*

**Swap Limit** 133.42 *GB*

Usage

Overview

Processes

No processes found

CPU

Total Usage

Usage per Core

Usage Breakdown

Memory

Total Usage

Usage Breakdown

3.54 GIB / 125.73 GIB (3%)

**Figure 2.** cAdvisor view on a 8-core HTCondor batch job's current resource usage.

[6]When run as a native Docker container, cAdvisor needs to be run as exposed and privileged container as well to have access to all necessary information in the virtual file systems.
[7]Mount suggestions for cAdvisor as Docker container can be applied to its avatar as Singularity container.
[8]We use a `timectl` unit to run the container regularly and define dependencies on other systemd units, but any other cron-like scheduler as cron or condor_cron should be working as well.

To take the European Data Protection Rules (GDPR) [17] into account to ensure encrypted transfer of metrics, we modified our first approach adding an intermediate step via Filebeat.
Our first approach is sketched in Figure 3. Here, the aggregation container collects information from Condor and cAdvisor forwarding them directly to a Logstash node within our ELK Stack for further processing and storage in Elastic Search.
To comply with the European Data Protection rules, we allow for sensitive data only to be transmitted encrypted. As we use mainly Filebeat and other Beat shippers as log and information forwarders, we use our existing PKI infrastructure as certificate basis to establish encrypted channels from our batch nodes to Logstash parsing nodes or Elastic Search database nodes. Since using the existing PKI with Logstash proved to be cumbersome, we took a shortcut. Instead of sending the extended information directly from the aggregation container to ElasticSearch, we write the extended information in a file and attach a Filebeat to the file. Figure 4 sketches both possible ways for forwarding the data, directly or encrypted via the intermediate Filebeat.

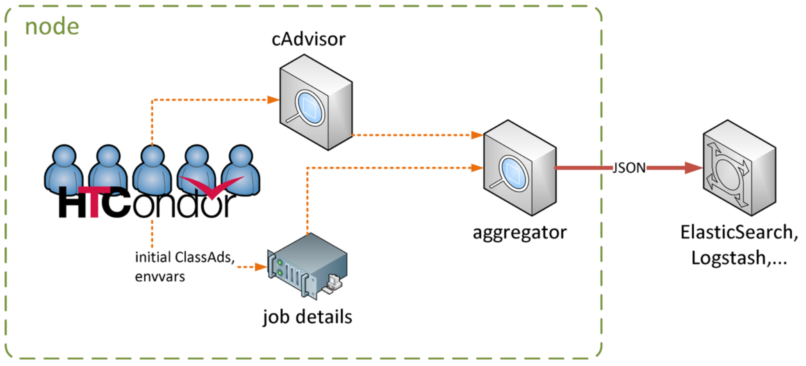


**Figure 3.** Combining job information from a cAdvisor container instance with batch system details in an aggregation container and forwarding the extended job information directly to Elastic Search or for further processing to a separate Logstash instance.

## 4 Deployment

We deploy the cAdvisor and the aggregation containers via CVMFS. As the containers are self-contained (except requiring a local installation of Singularity), we can avoid building packages for different distributions and can use the already existing distribution infrastructure.
Both containers need to be run with root privileges as they need to have access to the process and cgroup information in the virtual file systems as `/proc` and `/sys/fs/cgroup`. Since we mount the host file system only as a read-only bind mount, the security implications are assumed to be limited.
The cAdvisor container expects the host namespace mounted under `/rootfs`[9] and can be initiated directly or as Singularity instance [10]. We run the cAdvisor container

[9]as cAdvisor needs to read basic process information, virtual file systems as `/proc` or `/sys/fs/cgroup` (or other cgroup mounts) need to be readable, we bind / read-only onto "`/rootfs`".
[10]`/usr/bin/singularity run -bind /:/rootfs:ro /cvmfs/grid.desy.de/container/cadvisor`

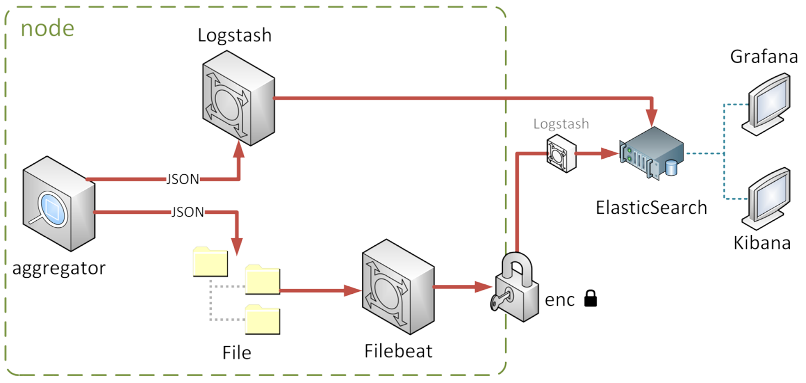


**Figure 4.** Paths from the aggregation container for sending the job details to the ElasticSearch instance. Either directly or taking a detour over a local file for an encrypted transport via the established Filebeat infrastructure to comply with the GDPR.

as a persistent systemd unit after a node's network start-up with the paths in CVMFS as requirement.
For the aggregation container, we bind mount only the necessary directories. To take different cgroup implementations/mount points of the different kernel generations into account, the aggregation container expects the cgroup virtual file system to be mounted on a dedicated mount point. For the operating mode of dumping the job information into a file for Filebeat, a writeable mount shared with the host namespace is necessary. If no output paths are given as parameters, it is assumed to send the job data directly to a end point [11]. A timectl unit regularly runs the aggregation container as a one-time systemd unit with the cAdvisor unit as dependency. Additional information to enable environment variables for changing the container's default behaviour can be obtained via `singularity help`.
Since we encountered a number of nodes with stalled CVMFS repository mounts when running the containers as services, we additionally deploy an auxiliary unit to prevent the repository mount point to be unmounted[12].

## 5 Outlook

Instead of reading HTCondor batch system details from the jobs' and PIDs' information in the (virtual) file systems, the HTCondor tool `condor_who` could be a more direct and less error prone way to collect the jobs' information on a node. Furthermore, implementing our own Beat [18] for encrypted transmission instead of using an intermediate Filebeat as stop-gap would be advisable.
Our approach could be generalized to other batch systems (apart from HTCondor) by rewriting and sourcing our code into separate classes, which would also make the code maintenance easier.

[11] `/usr/bin/singularity exec -bind /var/log/condor/jobmon:/var/log -bind /sys/fs/cgroup:/sys/fs/cgroup:ro -bind /sys/fs/cgroup:/cgroup:ro /cvmfs/grid.desy.de/container/condormon.d /usr/local/bin/condor_services_logstash.py -json /var/log/jobs.stats -stderr /var/log/jobs.err`
[12] `/cvmfs/cernvm-prod.cern.ch/extras/cvmfs_block /cvmfs/grid.desy.de`

## References

[1] D.Thain, T. Tannenbaum and M. Livny, Concurrency - Practice and Experience **17**, 323-356 (2015)
[2] HTCondor homepage[13] `https://research.cs.wisc.edu/htcondor/`
[3] M. Jette, A. Yoo and M. Grondona: "SLURM: Simple Linux Utility for Resource Management" Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) (2003), 44-60
[4] Google cAdvisor: `https://github.com/google/cadvisor`
[5] Sylabs.io Singularity: `https://www.sylabs.io/`
[6] CernVM File System: `https://cernvm.cern.ch/portal/filesystem`
[7] T. Finnern "Status of DESY Batch Infrastructures", HEPiX Fall 2015 at Brookhaven National Laboratory BNL (2015) `http://pubdb.desy.de/record/275834/files/DESY_Th.Finnern-_BatchInfrastructures.pdf`
[8] Andreas Haupt, Y. Kemp: "The NAF: National Analysis Facility at DESY", J.Phys.Conf.Ser. 219 (2010) 052007
[9] IBM Spectrum Scale technology (previously GPFS) `http://www-03.ibm.com/-systems/storage/spectrum/scale/index.html`
[10] dCache `https://www.dcache.org/`
[11] The JavaScript Object Notation (JSON) Data Interchange Format `https://tools.ietf.org/html/rfc8259`
[12] ElasticSearch: RESTfull, Distributed Search & Analytics `https://www.elastic.co/products/elasticsearch`
[13] Apache Lucene `https://lucene.apache.org/`
[14] cAdvisor: Exporting cAdvisor Stats to ElasticSearch `https://github.com/google/cadvisor/blob/master/docs/storage/-elasticsearch.md`
[15] Kibana: Explore, Visualize, Discover Data `https://www.elastic.co/products/kibana`
[16] Docker: Build, Ship, and Run Any App, Anywhere `https://www.docker.com/`
[17] European Comission regulation 2016/679: EU data protection rules `https://ec.europa.eu/commission/priorities/justice-and-fundamental--rights/data-protection/2018-reform-eu-data-protection-rules_en`
[18] Creating a New Beat `https://www.elastic.co/guide/en/beats/-devguide/current/new-beat.html`

[13]For all URLs memento snapshots have been preserved on `archive.org`. To get each URL's document version as seen during writing, call the original URL under the `20181029` timestamp, i.e., `https://web.archive.org/web/20181029/`*`original-url`*