Tartu University

Faculty of Science and Technology

Institute of Technology

Nikita Naumov

Visualizing the actual job resource consumption in HPC clusters

Bachelor's thesis (12 EAP)

Computer Engineering

Supervisor:

Sander Kuusemets

Tartu 2024

Abstract

Visualizing the actual job resource consumption in HPC clusters

This thesis researches possibilities for creating tools, which provide users of HPC clusters with a way to discover actual resource consumption of jobs running on clusters.

Elastic Stack family of software is one way of creating such tools. It provides extensibility, integration among packages of different functionality and a way to visualize data. Elastic software packages were successfully used to develop a tool for collecting and visualizing data on job resource consumption in HPC cluster utilizing Slurm job scheduler.

CERCS: T120 Systems engineering, computer technology

Keywords: HPC, cluster, Slurm

HPC klastrite tööde kasutatavate ressursside tegeliku tarbe visualiseerimine

See lõputöö uurib võimalusi tööriista loomiseks, mis annaks HPC klastrite kasutajatele võimalust teada klastritel käivitatavate tööde tegeliku ressursside tarbe.

Elastic Stack tarkvara perekond on üks võimalus selliste tööriistade arendamiseks. See pakub laiendatavust, erineva funktsionaalsusega tarkvarapaketide omavahelist integratsiooni ja võimalust andmete visualiseerimiseks. Elastic tarkvara oli edukalt kasutatud Slurm plaanuri kasutatava HPC klastri tööde ressursside tarbe andmete kogumiseks ja visualiseerimiseks.

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia

Märksõnad: HPC, klaster, Slurm

Contents

Definitions

HPC - high-performance computing - computer systems which provide significantly higher computing speeds and power compared to majority of other computer systems [1]

daemon - program running as a background process [2]

UID - user identifier - numerical value identifying a user in Linux and other Unix-like operating systems [3]

Introduction

Today HPC clusters are becoming increasingly more common for performing various tasks such as data processing, operations of various companies and organizations from wide range of fields, scientific calculations and others. [1] Usually, such tasks use noticeably more resources than tasks performed by general-use personal computers. Among other things, this means that HPC tasks require more energy and financial resources. HPC users, whose tasks are being performed (such as scientists, businesses), often they would fail in realistically determining the amount of needed resources for their tasks. This can cause difficulties both for users (for example, having to pay more for requesting too much resources for a task) and cluster administrators and operators (such as, forming longer queue because of too much unneeded resources requested and thus completing less tasks than is possible over a certain time period) [4: page 1-2]. Therefore, both HPC cluster operators and their users have the need to find a way for collecting and predicting the resource consumption for different kinds of tasks.

The problem for this thesis is the absence of flexible software tools, which would allow both for gathering necessary HPC metrics and for further processing data for other purposes, including its visualization. It should be noted that HPC systems use may vary by used software (including operating systems, job schedulers and others), architecture and hardware, making it difficult to develop universal software to be used in clusters.

To solve aforementioned problem, author will research software used in HPC clusters, existing tools and libraries and develop metrics-gathering software meeting the requirements. For this thesis all work was done on the infrastructure of Tartu University HPC center.

1 State of the Art

HPC (*high-performance computing*) term is used to describe various computing practices, which, compared to standalone general-use PCs, provide much more power for performing the tasks. [5] In order to provide such power for tasks, HPC differs from general computing in several aspects. The first one is parallel computing: unlike general PCs, in which instructions are executed consecutively, in HPC many tasks are often executed in parallel at the same time. The second aspect is resources (for example, CPUs, memory). For HPC either supercomputers or clusters are used, but both provide significantly more resources than PCs. While supercomputers themselves have far more powerful hardware, clusters consist of many computer servers called nodes, which work together by being connected to each other via fast interconnect. [6] Because of this, HPC clusters are frequently used for such use cases, which require executing large number of jobs. HPC clusters use job schedulers, which control the execution of jobs. [1]

While job scheduler controls the technical side of job execution, it remains for the cluster's user to request the correct amount of resources needed. Since the user may not be proficient enough to request only as much resources as needed by the job, it is necessary to create a tool, which would give users understandable overview on the amount of resources used by jobs. There are different approaches for creating such tools depending on the objective. For example, in the Evalix project metrics were collected from all jobs performed [7: page 2]. Another approach is to only collect info about the biggest jobs and ignore smaller ones. [4: page 6]. This way could be better in certain circumstances, since jobs with bigger resource usage also have bigger energy consumption and consequently bigger cost of their execution. [4: page 2].

1.1 Rocket HPC

For this thesis Tartu University Rocket HPC cluster is used. Rocket cluster utilizes the Slurm scheduler [8] for allocating resources for jobs run by users. Tartu University HPC clusters are used by many academic and commercial users, and thus there is certainly a need to develop such tool like described earlier.

1.2 Slurm

As mentioned before, for the purposes of this thesis Slurm job scheduler is utilized. Slurm consists of several parts providing functionality, including allocating resources for jobs and user-facing tools for starting, controlling and monitoring their execution. [9]

Slurm user submits their job for the execution either using *srun* or *sbatch* tools. *srun* begins to run jobs in parallel, while also diving one job into job steps, which themselves could either be run sequentially or in parallel. *srun* also allows user to set various resource requirements, for example, which CPUs this job can and cannot use. [10][11] *sbatch* submits batch scripts to run sequentially. [10]

Slurm can control submitted job's resource limits by utilizing Linux kernel *cgroup* technology. *cgroup* (control group) is a grouping of tasks (processes) associated with parameters of *subsystem(s)*. A subsystem here means controller which can apply certain resource limits on *cgroups*. This allows for accounting and limiting the resource usage for groups of processes. For example, *cpuset* subsystem can limit tasks in certain *cgroup* to executing only on certain CPU(s). [12][13]

1.3 Elastic Stack

For this thesis, software from Elastic Stack was chosen to work with. Elastic Stack is a family of software packages which provide various search, visualization, monitoring and logging tools among others. It was chosen because packages from Elastic Stack are designed to work with each other and provide wide variety of features. Elastic packages that are going to be used are the following:

- Elasticsearch, a search engine, which is able to collect and store data provided by other tools (such as Elastic Beats) [14];
- Kibana, a visualization tool, which can take data from Elasticsearch (among other sources) and provide interface for managing and analyzing it [15];
- Metricbeat, a tool for collecting metrics from various sources and sending them according to user's needs (for example, to Elasticsearch) [16]. Metricbeat is one of Elastic Beats, the collection of programs for capturing and sending various types of data from servers. [17]

Another important reason is that Elastic Stack includes Elastic Beats platform, which is extensible and for example allows developers to create new Filebeat and Metricbeat modules or entirely new Beats. Overall, Elastic Stack was chosen over alternatives because of its integrated nature, meaning that developers/users do not need to put much work into figuring out how to send or process captured data or how to visualize it using other tools.

There are other possible integrated alternatives, such as Prometheus for collecting the metrics and Grafana for visualizing it. However, they have other drawbacks. For example, the freely available version of Grafana lacks role-based access control, so it isn't possible to set up software such way that user could only see data on their own jobs. [18]

2 Methodology

For developing metrics-collecting tool, Elastic Beats platform was chosen. Elastic Beats are collection of programs for collecting various types of data, processing it, combining the data with system-based metadata, and converting it all to JSON-based representation, before sending it to Elasticsearch for further use. [17] One of the available Beats is Metricbeat, which allows to collect metrics from various sources (such as *nginx*, Apache HTTP Server, overall system metrics). [16] Metricbeat collects metrics data using modules, each of which contains code for fetching specific data from source(s). After fetching requested metrics, Metricbeat generates an event, containing fields defined in a module. In case of failure, Metricbeat generates an error event. [19] Metricbeat is extensible by creating new modules. [20] In this thesis author has created a module named *'slurm'* for receiving job metrics from Slurm.

For development and testing, CentOS virtual machine with Slurm was created and set up. There, testing instances of Elasticsearch and Kibana were also set up.

2.1 Preparation

Before creating a new module, we need to install and configure necessary Elastic software. For this thesis author chosen self-hosted installation option (i.e., non-Docker installation). Since official Elastic repositories are no longer available, all packages were installed by manually downloading and installing them via package manager.

2.1.1 Elasticsearch

First, Elasticsearch software was installed by running *wget* command with location of the package (contained in official documentation) and the installing the downloaded packages using rpm package manager:

```
wget remote location of the package
sudo rpm --install local location of downloaded package
```

Figure 2.1: commands for downloading and installing Elasticsearch and Kibana

After installation, we can start Elasticsearch. When running for the first time, Elasticsearch would generate a password for default *elastic* user and output it to terminal. In addition, Elasticsearch will generate a CA certificate for using HTTPS. This certificate will be stored at */etc/elasticsearch/certs/http_ca.crt* and will be used to verify the HTTPS certificate validity of Kibana and Metricbeat instances. [21]

Elasticsearch comes mostly pre-configured after installation but can be configured further before using if needed. In case of this thesis, author will use the defaults.

In most cases, Elasticsearch will need to be run as a daemon. In most Linux systems, that means running it as a *systemd* service. When running for the first time or after changing the service configuration file, it is necessary to run the following command:

```
sudo systemctl daemon-reload
```

Figure 2.2: *systemd* command for reloading the *systemd* manager configuration [22]

In case it is needed to start Elasticsearch service at a startup, the following should be run:

```
sudo systemctl enable elasticsearch
```

Figure 2.3: *systemd* command for starting Elasticsearch service at startup

The service can be started manually with the following:

```
sudo systemctl start elasticsearch
```

Figure 2.4: *systemd* command for starting Elasticsearch service

And stopped with this command:

```
sudo systemctl stop elasticsearch
```

Figure 2.5: *systemd* command for stopping Elasticsearch service

The current status of a service could be checked with the following:

```
sudo systemctl status elasticsearch
```

Figure 2.6: *systemd* command for checking current status of Elasticsearch service

2.1.2 Kibana

Similarly to Elasticsearch, Kibana was installed by running *wget* and *rpm* tools to download and install the package.

Considering that we already run Elasticsearch after its installation, we already have aforementioned CA certificate for HTTPS and password for *elastic* user. Now it's possible to connect Kibana instance with Elasticsearch by generating enrollment token. It can be done by running the following command:

```
bin/elasticsearch-create-enrollment-token -s kibana
```

Figure 2.7: command for generating enrollment token for Kibana instance. Note that this command should be run from Elasticsearch installation directory [23][24]

The enrollment token will then be output to the terminal and should be copied. Now, we can start Kibana and enroll it using generated token. This is done by running the following commands from Kibana installation directory:

```
bin/kibana
bin/kibana-setup --enrollment-token enrollment token
```

Figure 2.8: commands for enrolling Kibana using an enrollment token [25]

After these steps are done, Kibana is now ready to be used. Similarly to Elasticsearch, it usually will be run as a *systemd* service and thus previously mentioned commands also apply here:

```
sudo systemctl daemon-reload
```

Figure 2.9: *systemd* command for reloading the *systemd* manager configuration

```
sudo systemctl enable kibana
```

Figure 2.10: *systemd* command for starting Kibana service at startup

```
sudo systemctl start kibana
sudo systemctl stop kibana
sudo systemctl status kibana
```

Figure 2.11: *systemd* commands for starting, stopping and checking current status of Kibana service

While Kibana's web interface is preconfigured and accessible locally via port 5601, it's usually needed to also access it remotely. In order to be able to do that, we need to edit Kibana's configuration file *kibana.yml* accordingly. In this case, it's located in */etc/kibana* directory. There, *server.host* field needs to be changed from default *localhost* value to remotely accessible IP address of Kibana instance. To apply changes in file, it needs to be saved and Kibana reloaded. [26]

2.1.3 Metricbeat

While Metricbeat is also available in packaged form for installation, for this thesis author decided to build it from source code instead. The reason for this is that new metricsets and module for Metricbeat will be developed, which both require building new Metricbeat binaries with new module code included.

First, Elastic Beats source code repository needs to be cloned to the machine:

```
git clone https://github.com/elastic/beats
```

Figure 2.12: command for cloning the Elastic Beats repository

Since Elastic Beats software is written in Go language, it's necessary to install Go language tools (compiler and others) and Mage build tool. In this case those were installed by downloading Go language tools as an archive from official website and unpacking it. Afterwards Mage was installed by cloning its repository and running installation script contained in it. Required commands were the following: [27][28]

```
wget location of required archive from Go website
sudo tar -C /usr/local -xzf name of downloaded archive
git clone https://github.com/magefile/mage
cd mage
go run bootstrap.go
```

Figure 2.13: commands for installing Go language tools [27][28]

When all needed tools are ready, we can proceed with building the Metricbeat. It only requires running one Mage command from inside *metricbeat* directory in repository:

```
cd repository/metricbeat
mage build
```

Figure 2.14: commands for building Metricbeat

After building, *metricbeat* binary will be located in *metricbeat* directory inside repository. This directory also contains configuration file called *metricbeat.yml*. This file will need some changes before it’s possible to run Metricbeat and connect it to Elasticsearch and Kibana:

- *setup.dashboards.enabled* was set to true; this will enable Kibana dashboards.
- *setup.kibana.host* was uncommented and set to “http://localhost:5601”: this points to the address of Kibana host.
- *output.elasticsearch.username* and *output.elasticsearch.password* were uncommented; username was set to “elastic” and password was set to “password” generated by Elasticsearch on its first run.
- *output.elasticsearch.ssl.certificate_authorities* was set to "*/etc/elasticsearch/certs/http_ca.crt*".
- *output.elasticsearch.allow_older_versions* was set to true; normally that would not be needed as usually both Elasticsearch and Beats have same versions installed. However, since here Metricbeat is compiled from the latest source and other packages were installed as their stable versions, this is done to allow Metricbeat communicate with Elasticsearch.
- *seccomp.enabled* was added and set to false; this is needed for created metricsets to be able access system files and execute commands.

Since we also want to be able to run Metricbeat as a daemon, we need to create a service file. Such file for Metricbeat could look like this:

```
[Unit]
Description=Metricbeat is a lightweight shipper for metrics.
Documentation=https://www.elastic.co/beats/metricbeat
Wants=network-online.target
After=network-online.target

[Service]

UMask=0027
Environment="GODEBUG='madvdontneed=1'"
Environment="BEAT_LOG_OPTS="
Environment="BEAT_CONFIG_OPTS=-c /etc/metricbeat/metricbeat.yml"
Environment="BEAT_PATH_OPTS=--path.home /usr/share/metricbeat --path.config
/etc/metricbeat --path.data /var/lib/metricbeat --path.logs /var/log/metricbeat"
ExecStart=location of metricbeat executable  --environment systemd $BEAT_LOG_OPTS
$BEAT_CONFIG_OPTS $BEAT_PATH_OPTS
Restart=always

[Install]
WantedBy=multi-user.target
```

Figure 2.15: example of *systemd* service file for Metricbeat

Metricbeat comes with many modules pre-installed, but only '*system*' is enabled by default. User is able to enable or disable any installed module by running the following commands:

```
./metricbeat modules enable module name
./metricbeat modules disable module name
```

Figure 2.16: commands for enabling and disabling Metricbeat modules [29]

It is also possible to get a list of currently enabled and disabled modules by running the following:

```
./metricbeat modules list
```

Figure 2.17: command for listing enabled and disabled Metricbeat modules [29]

2.2 Metricbeat module

Metricbeat modules consist of at least one metricset. Each metricset contains code for receiving one or multiple related metrics. This allows for a user to choose to receive only needed set of metrics instead of fetching everything. [19] When creating a new module, developer simultaneously creates the first metricset in this module and afterwards more metricsets are added. When creating a '*slurm*' module, the author created two metricsets in it – '*job_cpu*' for getting CPU-related metrics and '*job_mem*' for memory-related ones.

Each module and metricset contain definitions of fields present in an event. There can be two kinds of fields in a Metricbeat module – module fields and metricset fields. Metricset fields are specific to one metricset and only present in events generated by it, while module fields can be present in events generated by all metricsets in a module. [19] In '*slurm*' module, author defined the following fields:

- Module fields:
    - '*job_user*' – username whose job is being executed;
    - '*jobid*' – job ID;
    - '*step*' – job step
- Metricset fields:
    - *job_mem* fields:
        - '*memusage*' – memory usage of a job;
        - '*memreq*' – amount of memory job can use (requested memory amount)
    - *job_cpu* fields:
        - '*cpuutil*' – CPU utilization of a job (in % - CPU time/job running time);
        - '*cpuused*' – CPU usage (number (fraction) of CPUs used);
        - '*cpureq*' – CPU to processor threads ratio

The definitions for fields are contained in *fields.yml* file in corresponding metricset or module directory.

Each metricset contains source code files written in Go language, which are programs executed by Metricbeat. Each of them contains a metricset type definition, which has fields related to this particular metricset, *New()* function, which is called only the first time and creates new metricset instance, and *Fetch()* function, which is called periodically by Metricbeat and contains logic for getting the data. [30]

Creating new metricsets starts with running the following command in Metricbeat source code directory: [30]

```
make create-metricset
```

Figure 2.18: command for creating new metricset

This will prompt user for names of a module and metric set and create required directories and files. If a module with such name does not yet exist, it will additionally create required files for a new module.

In order to build Metricbeat with a new module or metricset, the following should be run: [30]

```
mage update
mage build
```

Figure 2.19: commands for updating and building Metricbeat

The first command will update some generated files according to changes made by developer and the second will build Metricbeat.

All source code which was written for new module is publicly available here along with instructions for installation: https://github.com/NikitaNaumov98/metricbeat-slurm

2.2.1 Memory metricset

In memory metricset we're getting the needed data from reading appropriate files in control groups (*cgroup*) directories associated with slurm jobs. Slurm specifies that default root directory for

Slurm control groups is */sys/fs/cgroup*. [31] Therefore, needed files are located in subdirectories of */sys/fs/cgroup/memory/slurm*. Inside this directory the following directory tree exists:

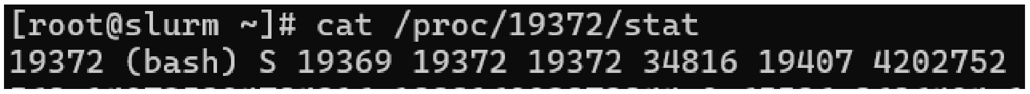
- */uid_*[UID]
    - */job_*[JobID]
        - */step_*[job step]
            - */memory.usage_in_bytes*
            - */memory.limit_in_bytes*
            - and other files..

Here "UID" refers to Linux UID, who launched the job; "JobID" refers to job ID; and "job step" refers to job step(s), creation and allocation of which depends on how the job was launched (for example, if the job was launched using *sbatch* command, then it only involves a single step named *batch*). [32]

Information we're interested in is located in various files in these directories. Since in real usage there will likely be multiple jobs, belonging to many different users, as well as multiple job steps per job, metricset's fetching function iterates over all three levels of directories, beginning with */uid_*, collecting info about all currently running jobs and job steps of all users. UID of a user, who launched the job, is used to determine the actual username, which will be included in an event as well. Once function arrives to */step_* directory, it also reads information contained in *memory.usage_in_bytes* and *memory.limit_in_bytes* files. These files contain memory usage of a job and maximum amount of memory a job can use, respectively. [33] If there aren't any */step_* directories in */job_* directory, then it reads *memory.max_usage_in_bytes* and *memory.limit_in_bytes* of that directory instead. Once everything is collected without errors, the Metricbeat event is generated containing all relevant fields and the information is sent to Elasticsearch. Possible errors are handled by Metricbeat's built-in logger. Metricbeat event is generated for every running job step or entire job, if there's no job step.

2.2.2 CPU metricset

For CPU metricset, the process of getting info is more complex than for memory metricset. The reason for this is, unlike memory usage information, we cannot get CPU info by simply reading files. Instead, we begin with using existing Go library "*go-ps*" for getting information about running processes. We iterate over the process list and look for process(es) with executable of *slurmstepd*. *Slurmstepd* is a Slurm job step manager, which is running during the entire execution of a job step. [34] This is useful as we can get information about what job steps are running right now. For *slurmstepd* processes we get their PIDs (process IDs), which are then used for getting additional information about processes from a */proc* directory. On Linux systems */proc* directory contains subdirectories for every process currently running, using PIDs as their names. These subdirectories contain various files with process-related information as their content. Among these files here we're going to read the contents of *status*, *cmdline* and *stat*. *Status* contains various status information; we're only interested in getting the UID of a process, so we're only using line 8 of the output. *Cmdline* contains the full command of the process, which also includes job ID and name of the job step for Slurm jobs. *Stat* file contains many various values associated with process, out of which we're using session ID value. [35] For example, this is how first 9 fields of a *stat* file of some process look like:



```
[root@slurm ~]# cat /proc/19372/stat
19372 (bash) S 19369 19372 19372 34816 19407 4202752
```

Figure 2.20: example of several first fields of a *stat* file

The sixth field of *stat* file is called '*sid*' and contains a session ID. [35]

The only time we're using information from Slurm *cgroup* directories is to get information about what CPUs this job is allowed to use, provided by a *cpuset.cpus* file. [36] This information is used to get how many CPUs the job is using, which is needed for computing CPU utilization.

To get the information about CPU utilization itself, the author decided to use *os/exec* Go package. This may cause problems with functioning of a module, since by default Metricbeat has Linux Secure Computing Mode (*seccomp*) enabled, which disallows for the process to make system calls. [37] In such cases, the easiest way to deal with a problem is to disable *seccomp* entirely by adding *seccomp.enabled: false* line to a Metricbeat configuration file. This, however, may increase potential impact of zero-day vulnerabilities and thus should be done with caution. It is also possible to configure the default *seccomp* policy (*allow* or *deny*) and the exception list of system calls that could or could not be made. These actions are similarly done in a Metricbeat configuration file.

Also it should be noted that at the time of writing *seccomp* was described by Elastic as a beta feature, which is likely to change and is not supported officially. [38]

The forenamed *os/exec* package is used to execute Linux commands within a Go program. [39] In this metricset it is used to execute *ps* Linux command to get CPU utilization values of processes related to job or job step. It is executed with the following arguments:

- *-o pcpu=* - only outputs CPU utilization values without the header line [40].
- *-g* [session ID] - elects only processes with a certain session ID. As mentioned before, we got the session ID from a *stat* file. Session ID allows to group several related processes and here we are using it to identify processes related to a job or a job step. This is needed because during the execution of a job, several processes are spawned and we need to get the utilization metrics for all of them.

After receiving the output for this command, we sum all values contained in it to get the overall CPU utilization for all processes of a job or job step. However, this number needs to be divided by a CPUs to threads ratio to be a meaningful metric. Getting the number of CPUs used by a job was described before and getting the number of threads for each of CPUs is done by using *os/exec* package to call *lscpu* command and parsing its output.

The last metric we want to get is the actual number (fraction) of CPUs utilized by a job. This is done by multiplying CPU utilization metric we compute previously by a CPUs to threads number and then dividing by 100.

3 Results

After building the Metricbeat with the new *slurm* module, it can be run according to an option specified in its configuration file and with chosen modules. Metricbeat configuration file *metricbeat.yml* also specifies where Metricbeat sends its output, which for example can be Elasticsearch or a simple text file.

3.1 Events

Running Metricbeat with *job_cpu* and *job_mem* metricsets enabled will periodically generate events containing fields associated with them. If some job is running along with Metricbeat, resulting events will roughly be the following:

```
//Output omitted – various technical information
   "event":{
      "dataset":"slurm.job_mem",
      "module":"slurm",
      "duration":317688
   },
   "metricset":{
      "name":"job_mem",
      "period":10000
   },
   "slurm":{
      "job_mem":{
         "memusage":4259840,
         "memreq":524288000
      },
      "job_user":"root",
      "jobid":134,
      "step":"batch"
   },
   "service":{
      "type":"slurm"
   }
}
```

Figure 3.1: Example of an event generated by Metricbeat for *job_mem* metricset

```
//Output omitted - various technical information
   "event":{
      "dataset":"slurm.job_cpu",
      "module":"slurm",
      "duration":171235091
   },
   "metricset":{
      "name":"job_cpu",
      "period":10000
   },
   "slurm":{
      "job_cpu":{
         "cpuutil":0.2,
         "cpuused":0.004,
         "cpureq":2
      },
      "job_user":"root",
      "jobid":134,
      "step":"batch"
   },
   "service":{
      "type":"slurm"
   }
}
```

Figure 3.2: Example of an event generated by Metricbeat for *job_cpu* metricset

If Metricbeat is configured to send information to Elasticsearch, then it is possible to use Kibana to visualize gathered information via dashboards.

3.2 Visualizing

By default, Metricbeat provides an index pattern, which is used by Kibana to discover and display Metricbeat fields. It means Kibana should be able to use fields created within new modules and metricsets. If they weren't added automatically for any reason (as was the case for the author), then the Kibana administrator can add them manually in *Stack Management* -> *Data Views*.

After module fields are made available, it's possible to create new dashboards using Kibana web interface (*Dashboards* -> *Create dashboard*). Author created a simple dashboard which can filter jobs by user(s) and job IDs, and display gathered metrics for all jobs within this range. Resulting dashboard with gathered information is pictured here:

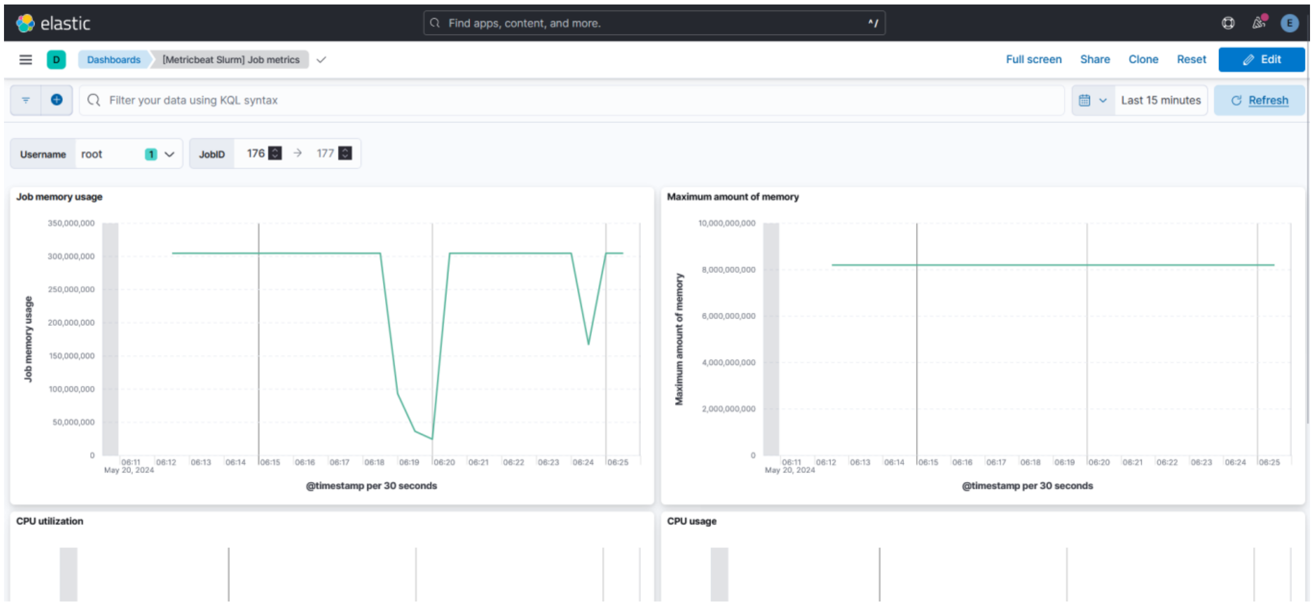


Figure 3.3: Screenshot of a Kibana dashboard showcasing gathered job metrics

For testing author used stress testing package '*stress-ng*' and run its various options as sample jobs. While it's still not testing on actual cluster jobs, it still managed to provide some insight into how created tool could behave in real situations.

4 Analysis and Discussion

In the process of writing this thesis, author has studied Slurm job scheduler used in HPC clusters, researched various software for achieving stated goal and created a tool, which allows for collecting data on utilized resources by executing jobs. Elastic Stack turned to be appropriate way to develop on for several reasons. One is extensibility, which allows developers to add new functionality by writing Go programs working alongside existing tools. Another is integration between various packages so that developers do not need to add ways to move data between various tools themselves. Then, there is also built-in error handling (for example, in Metricbeat), which means that it's not difficult to discard invalid data. Created tool also allows for HPC cluster administrators to create necessary Kibana dashboards and grant access to users to them via web interface, thereby enabling users to monitor the execution of their jobs.

5 Conclusion

The reason for writing this thesis was the relative absence of user-friendly tools for getting the information on resource consumption of HPC cluster jobs. To change that, author first studied the Slurm job scheduler, Linux kernel *cgroups* and some other technologies currently utilized in HPC clusters on the example of Tartu University HPC centre. Later, author researched which tools could aid the development of such software and decide on them. As a result, initial version of software was developed and it can now be said that development of such tools is possible by utilizing already existing software packages.

Considering all difficulties with this work, it can still be said it achieved at least a usable initial solution, which is suitable for further development. Possible areas for improvement include moving from relying on external Linux utilities for collecting the data, including more information about jobs and creating more user-friendly dashboards and visualization possibilities. Further testing in real-world-like situations would similarly be useful.

There are some obvious shortcomings with this module. Relying on calling external Linux commands within Go code to retrieve information is not an elegant solution and can also be a security risk. It also means that if Metricbeat is for any reason configured to prohibit making most system calls, this module cannot be used at all.

Bibliography

[1] IBM 2022. - What is high-performance computing (HPC)? https://www.ibm.com/topics/hpc 28.10.2022, 22:27 (UTC).

[2] Pat Brans 2022. - What is a daemon? https://www.techtarget.com/whatis/definition/daemon 08.2022

[3] Abhishek Prakash 2022. - Everything Important You Need to Know About UID in Linux https://linuxhandbook.com/uid-linux/ 26.05.2022

[4] X. Li, N. Qi, Y. He ja B. McMillan, "Practical Resource Usage Prediction Method for Large Memory Jobs in HPC Clusters", Lecture Notes in Computer Science, 11416, 2019, lk 1-2, 6, DOI: https://doi.org/10.1007/978-3-030-18645-6_1.

[5] Oracle 2022. - What is High-Performance Computing (HPC)? https://www.oracle.com/cloud/hpc/what-is-hpc/ 28.10.2022, 22:10 (UTC).

[6] The University of Sheffield 2024. - What is High Performance Computing? https://docs.hpc.shef.ac.uk/en/latest/hpc/what-is-hpc.html 20.05.2024

[7] J. Emeras, S. Varrette, M. Guzek ja P. Bouvry, "Evalix: Classification and Prediction of Job Resource Consumption on HPC Platforms", Lecture Notes in Computer Science, 10353, 2017, lk 2, DOI: https://doi.org/10.1007/978-3-319-61756-5_6.

[8] UT HPC Center 2022. - Rocket https://hpc.ut.ee/services/HPC-services/Rocket 30.10.2022, 02:07 (UTC).

[9] Slurm 2021. - Slurm 20.11.9 Workload Manager Overview https://slurm.schedmd.com/archive/slurm-20.11.9/overview.html 09.06.2021

[10] SchedMD 2021. - Slurm Workload Manager - Quick Start User Guide https://slurm.schedmd.com/archive/slurm-20.11.9/quickstart.html 16.03.2021

[11] SchedMD 2023. - Slurm Workload Manager - srun manual page https://slurm.schedmd.com/archive/slurm-20.11.9/srun.html 04.05.2023

[12] Linux kernel developers 2024. - Control Groups https://docs.kernel.org/admin-guide/cgroup-v1/cgroups.html 18.05.2024

[13] SchedMD 2022. - Slurm Workload Manager - Control Group in Slurm https://slurm.schedmd.com/cgroups.html 16.06.2022

[14] Elastic 2024. - Elasticsearch Guide - What is Elasticsearch? https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html 19.05.2024

[15] Elastic 2024. - Kibana Guide - Kibana—your window into Elastic https://www.elastic.co/guide/en/kibana/current/introduction.html 19.05.2024

[16] Elastic 2023. – Metricbeat overview https://www.elastic.co/guide/en/beats/metricbeat/master/metricbeat-overview.html 19.05.2023

[17] Elastic 2023. - What are Beats? https://www.elastic.co/guide/en/beats/libbeat/current/beats-reference.html 19.05.2023

[18] Grafana Labs 2024. - Grafana documentation - Role-based access control https://grafana.com/docs/grafana/latest/administration/roles-and-permissions/access-control/ 19.05.2024

[19] Elastic 2023. - How Metricbeat works https://www.elastic.co/guide/en/beats/metricbeat/master/how-metricbeat-works.html 19.05.2023

[20] Elastic 2023. - Extending Metricbeat https://www.elastic.co/guide/en/beats/devguide/current/metricbeat-developer-guide.html 19.05.2023

[21] Elastic 2024. - Elasticsearch Guide - Install Elasticsearch with RPM https://www.elastic.co/guide/en/elasticsearch/reference/current/rpm.html 12.05.2024

[22] systemd developers 2023. - systemctl manual page https://www.man7.org/linux/man-pages/man1/systemctl.1.html 22.12.2023

[23] Elastic 2024. - Kibana Guide - Install Kibana with RPM https://www.elastic.co/guide/en/kibana/current/rpm.html 12.05.2024

[24] Elastic 2024. - Elasticsearch Guide - Command line tools - elasticsearch-create-enrollment-token https://www.elastic.co/guide/en/elasticsearch/reference/current/create-enrollment-token.html 12.05.2024

[25] Elastic 2024. - Elasticsearch Guide - Start the Elastic Stack with security enabled automatically https://www.elastic.co/guide/en/elasticsearch/reference/current/configuring-stack-security.html 12.05.2024

[26] Elastic 2024. - Kibana Guide - Configure Kibana https://www.elastic.co/guide/en/kibana/current/settings.html 12.05.2024

[27] Go developers 2024. - Documentation - Download and install https://go.dev/doc/install 13.05.2024

[28] Mage developers 2024. - Installation https://magefile.org/ 13.05.2024

[29] Elastic 2024. - Metricbeat Reference - Metricbeat quick start: installation and configuration https://www.elastic.co/guide/en/beats/metricbeat/current/metricbeat-installation-configuration.html 19.05.2024

[30] Elastic 2023. - Creating a Metricset https://www.elastic.co/guide/en/beats/devguide/current/creating-metricsets.html 19.05.2023

[31] SchedMD 2023. - Slurm Workload Manager – cgroup.conf https://slurm.schedmd.com/cgroup.conf.html 04.05.2023

[32] SchedMD 2022. - Slurm Workload Manager – Job Launch Design Guide https://slurm.schedmd.com/job_launch.html 01.08.2022

[33] RedHat 2023. - Red Hat Enterprise Linux 6 Resource Management Guide - memory https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/sec-memory 19.05.2023

[34] SchedMD 2023. - Slurm Workload Manager - slurmstepd https://slurm.schedmd.com/slurmstepd.html 04.05.2023

[35] Linux kernel development community 2023. - The /proc filesystem https://www.kernel.org/doc/html/latest/filesystems/proc.html 19.05.2023

[36] RedHat 2023. - Red Hat Enterprise Linux 6 Resource Management Guide - cpuset https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/sec-cpuset 19.05.2023

[37] Elastic 2023. - Use Linux Secure Computing Mode (seccomp) https://www.elastic.co/guide/en/beats/metricbeat/master/linux-seccomp.html 19.05.2023

[38] Elastic 2024. - Filebeat Reference - Use Linux Secure Computing Mode (seccomp) https://www.elastic.co/guide/en/beats/filebeat/current/linux-seccomp.html 19.05.2024

[39] Go developers 2023. - os/exec https://pkg.go.dev/os/exec 19.05.2023

[40] procps-ng developers 2022. - ps manual page https://www.man7.org/linux/man-pages/man1/ps.1.html 12.12.2022

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Nikita Naumov,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose Visualizing the actual job resource consumption in HPC clusters, mille juhendaja on Sander Kuusemets, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 4.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Nikita Naumov

20.05.2024