PfiT - The Administrator's Guide

Documentation of the administration of the PfiT tools

August 19, 2020

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

Contents

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

Chapter 1

Introduction

High-Performance-Computing (HPC) has become a standard research tool in many scientific disciplines. Research without at least supporting HPC calculations is becoming increasingly rare in the natural and engineering sciences. Additionally, new disciplines are discovering HPC as an asset to their research, for example in the areas of bioinformatics and social sciences. This means that more and more scientists, who lack a deep understanding of the architecture and the functioning of such systems, start using HPC resources. This knowledge gap is growing, since the complexity of HPC resources is increasing and gaining significant importance in the field of performance engineering.

Most scientists that are new to HPC run their applications on local Tier 3 systems and are satisfied if their research problem can be solved on an available system in an acceptable time frame. The missing knowledge with respect to performance measurements will often restrict the performance, so that they are not able to scale their calculations to a Tier 2 or Tier 1 compute resource. At the same time, Tier 3 compute centers typically lack sufficient human resources to work with each user individually on application performance. In order to increase awareness for performance issues and enable users to assess possible gains from performance improving measures, systematic, unified, and easily understood information on performance parameters should be provided across all scientific communities. This especially pertains to the performance parameters of HPC jobs and the importance of performance engineering, since HPC compute clusters are very expensive resources. This applies to the procurement of the hardware and software as well as for the service of the system (especially the power supply), in which for example the overall energy costs amount to several hundred thousand euros for a five year life cycle of a typical Tier 3 system. To reduce these costs, jobs should have, for example, lower run time to save energy or less waiting time until starting the job for better utilization of the cluster. An efficient usage can be achieved, if the users obtain insight on the runtime behaviour of their programs, for example, via different kinds of reports, including additional automatic evaluation, interpretation and presentation of the performance metric values.

Although the usage of many performance measurement tools is mostly straightforward, the serious disadvantage is the missing explanation of the results in a clear and simple way. Often the generated reports require expert knowledge to be understood – this makes the profiling for normal cluster users nearly useless, since they usually do not have the background to interpret the results. By automatically assembling all data provided by available tools into a single centrally organized framework, it will be much easier for the user (and for administrators, too) to identify potential performance bottlenecks or pathological cases. In addition, with the help of an all-in-one monitoring, job analysis and reporting tool, the user acceptance for code optimizations might be drastically increased, especially when the code tuning shows a considerable performance boost. Additionally, the interface may also help to overcome the gap of understanding and communication between experts and end users by incorporating all data into a shared documentation system.

In order to achieve these goals, a monitoring, job analysis and reporting tool set, partly based on existing solutions, was implemented in the scope of this project. This tool set will automatically collect performance metrics and present them to researchers in easy to understand summaries or as a timeline (in appropriate reports). The tool set is complemented by documentation and best practices information, detailing, as applicable, further steps to be taken regarding the investigation of the problem, recommended changes to the job submission, and promising performance engineering targets.

PfiT is the monitoring tool suite to collect the monitoring and performance data of single core applications as well as shared memory and distributed memory jobs. This tool suite was developed in the DFG project ProfiT-HPC (Profiling Toolkit for HPC Applications), and the contributing partners of this project are the universities of Göttingen, Hamburg, Hannover and Rostock as well as the Zuse Institute Berlin (ZIB) (the university of Hannover retired in the course of the project). *PfiT* is based on parts of the TICK-Stack, which consists of the metric collecting tool *Telegraf*, the time series database *InfluxDB*, which stores the metric values, the visualization tool *Chronograf* and *Kapacitor* as the system monitoring tool. Only *Telegraf* and *InfluxDB* are used in this project. Due to its great flexibility, the visualization tool *Grafana* has been used for realtime visualization of the metrics instead of *Chronograf*. The main application area of the TICK stack are for example clouds. In this project we want to introduce parts of this stack into the HPC world, in order to measure performance metrics and to store and process them. Furthermore it must be mentioned, that the administrator can choose between the two metric collectors *Telegraf* (with appropriate plugins) and *PfiTCollect*, which was written as part of the *ProfiT-HPC* project. Both tools are metric collecting agents, but *PfiTCollect* is a new ligthweight collector which uses for example just a fraction of disk and memory space of the *Telegraf* version and spawns only one thread instead of approximately 40 like in *Telegraf*. This toolsuite is supplemented by several different report generators, namely a text and PDF report generator and a realtime dashboard in *Grafana*. An aggregator acts as the data interface between the database, the report generators and a recommendation system.

This document presents the administrator's guide of the *PfiT* system. In this guide we will describe every tool of this package and how to download, install, configure and run it. Additionally, we documented selected aspects concerning the *InfluxDB* timeseries database, for example topics about security issues, retention policies, continuous queries, backing up and restoring the data. Furthermore, example on-site installations of the involved project members will be presented.

This document is similar to the original description of the tools *Telegraf*, *InfluxDB* and *Grafana* on the corresponding web sites, but represents a condensed form of the description with a different structure. Although single parts of the procedure were described in the appropriate manuals very well and precisely, we did this because we think that the original documentation was somewhat confusing for someone with the first contact with these tools. **Therefore, parts of the text were taken over literally. The citations have been marked in the text by** *Quote* **and** *End quote***, so as not to be confused with any other symbols within the citation itself. If we have overseen a quote somewhere, please write an email to: info@profit-hpc.de, so we can make a correction.**

In every section of this chapter, we will shortly present the tools *PfiT* is based on. *Telegraf*, *InfluxDB* and *Grafana* as well as *PfiTCollect*, the aggregator and the report generators.

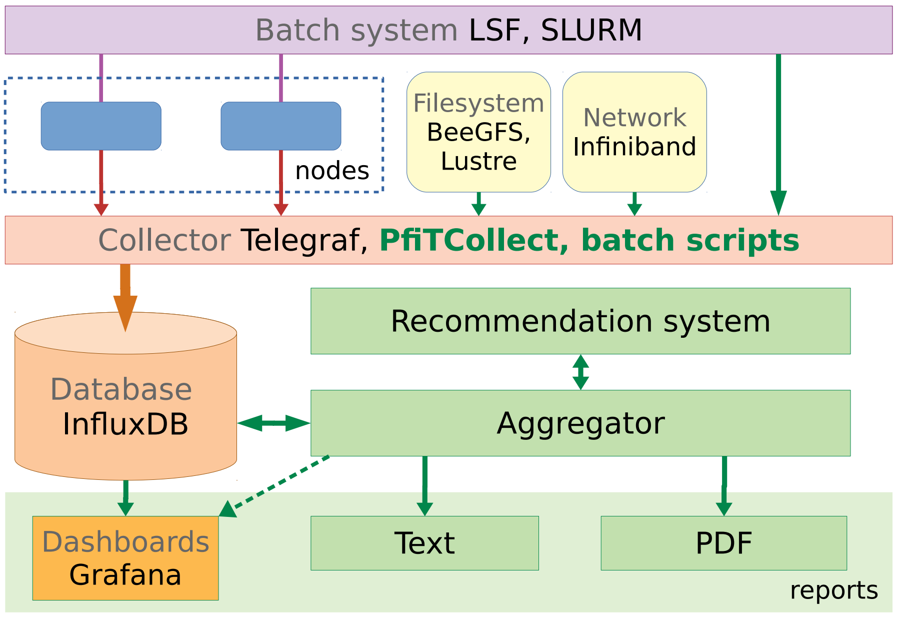


Figure 1.1: The architecture of the PfiT system

1.1 The architecture of the PfiT system

In figure 1.1 the architecture of the *PfiT* toolkit and its pricipal components is visualized. The main building blocks of this toolkit are:

- **Collecting** the metrics for every node in the cluster/subset of the cluster with the collecting agents *Telegraf* or *PfiTCollect* (with additional bash scripts and tools to collect IO, network and graphic cards metrices),
- **storing** the metrics in a time series database (in our case *InfluxDB*),
- **aggregating** the metrices stored in the database
- and job analysis within the **recommendation system**.
- **Report generation** will be done on the basis of the stored data of *InfluxDB* and the job analysis to formulate the recommendations.

1.2 The tools of the PfiT system, *not* written in the project

In this section *Telegraf*, *InfluxDB* and *Grafana* will be presented, that means all tools, which are not written within the *ProfiT-HPC*-project.

1.2.1 Telegraf

Telegraf is one of the two metric collectors of the *PfiT*-Stack and was written in *Go* by the *InfluxData Inc*. *Telegraf* is open source (except the scalable clustering component) and is realized as a server agent, which in general runs in the background on all (or a subset of) nodes in the cluster (exactly one *Telegraf* instance on one node) which sents metric data to the database in a two level hierarchy. Via the numerous plugins it is possible to collect many metrics of the system without measureable impact on the system performance (depending of the interval length). The plugin concept is also used to control the output and to write the collected metric values to time series databases (for example in our case *InfluxDB*, but other options are possible, for example, *Graphite*, *eXtremeDB*, *OpenTSB*, *ElasticSearch*) or data queues. In the *ProfiT-HPC* project the plugins that are used are listed here with some example metrics they collect:

- *Input/Collector plugins* (process and node based):
  - *procstat*, collects CPU, memory, swapping, IO metric values, etc. from *procfs*,
  - *uprocstat*, collects CPU, memory, swapping, IO metric values, etc. from *procfs* and labels the metrics with the jobid (this plugin was written in the ProfiT-HPC project to be able to visualize the metrics in *Grafana*),
  - *CPU*, collects CPU metric values (for example user, system, idle and iowait time),
  - *MEM*, collects free, used and available memory, etc.
  - *Process*, collects process metrics, such as the number of running, sleeping, stopped and zombie processes, etc.
  - *system*, collects the values of metrics of the load balance, the uptime of the system, etc.
  - *DiskIO*, collects the number of reads and writes, the number of read and written bytes, etc.
  - *Net*, collects the number of packages read and written, the number of read and written bytes, etc.
- *Processor plugins*,
  - *PfiT-JobID*, which adds the job ID's of a batch job to every metric passing through it.

Further metric classes are

- *Aggregate plugins*, to aggregate the data of the different jobs,
- *Output plugins*,

but they are not in use in our case.

A detailed introduction to the current version of *Telegraf* can be found on https://docs.influxdata.com/telegraf/. A more condensed documentation is included in the *PfiT* documentation.

1.2.2 InfluxDB

InfluxDB is a timeseries database, which was also written in *Go* by the *InfluxData Inc.*. It is designed to store time series data in an easy and efficient way. *InfluxDB* is mainly open source, except for the scalable clustering component and the enterprise solution of the database. In our case *InfluxDB* stores the metric values which were collected by *Telegraf* or *PfiTCollect*. Every measurement point, including the corresponding metrics, is labeled with a time stamp, which is internally represented in nanoseconds since the epoch. This time stamp is used to distinguish/label the measurements in the database. *InfluxDB* has a built in, SQL like query language which enables the request and retrieval of information from the databank to generate the user reports. Internally the time series data in *InfluxDB* is organized in measurements, which are conceptually analogous to tables in relational databases. All data in this table have a mandantory time stamp, (optional) tags, which are storing the metadata, fields to store the metadata and the values of specific metrics. In the tag components of the fields, the metadata or describing strings of the measured data are stored, while in the field values, the appropriate values of the metrics are stored.

A detailed tutorial of the current version of *InfluxDB* can be found at `https://docs.influxdata.com/influxdb/` and as in the case of *Telegraf* a more condensed documentation is included in the *PfiT* documentation.

1.2.3 Grafana

Grafana is an open source metric analytics and visualization suite, which runs as a web application and supports creating general purpose dashboards. *Grafana* supports several backends, such as *InfluxDB*, *Graphite* and *Elasticsearch*. The two main actions of *Grafana* are quering data from the time series database, which is defined in the dashboard, and visualizing the results of these queries. A detailed tutorial (installation, configuration, usage) can be found at `http://docs.grafana.org/` and a condensed version of the documentation is included in the *PfiT* documentation.

1.3 The tools of the PfiT system, *written* in the project

1.3.1 PfiTCollect

The *PfiT* monitoring tool suite provides the additional metric collector *PfiTCollect*, which can be used as a lightweigth alternative to the *Telegraf* collector, since it needs much less main memory, virtual memory and other resources. The collected raw metrics are described in the user's guide, Chapter 2.2. The data will be collected at predefined time stamps, for example every 30 seconds, and are then pushed into the *InfluxDB* database by the versital and widely used *curl* tool for transferring data to and from a server. Most of the raw metrics which are collected in *Telegraf* are also collected by *PfitCollect*. Some of them are not to stored explicitly to save database storage. However, in contrast to *Telgraf*, *PfiTCollect* does not support a plugin concept and all plugins are implemented in a special function and module.

The metric collector *PfiTCollect* can be installed wherever the administrator wants to install it. Preferably for example the directories /usr/local/bin or /opt/ can be taken. The compilation process can be done via adapting the delivered Makefile and then typing `make all` in the shell. This will compile the C program and after that, if no error occured, `make install` must be executed. It is important to set the ownership of this program accurately, as well as its rights. Finally, the metric collector is ready to be used in a job or as a cluster wide monitoring tool.

Why PfiTCollect

PfiTCollect is a light weight metric collector (written in C), designed towards the needs of the *ProfiT-HPC* project, which aims to collect metrices (at the moment only just via monitoring) which will be processed further and are used as the basis for the text and the PDF report generation. The text report lists mean values of CPU, memory and swap space, IO (work, NFS and scratch) and Infiniband usage, and the high water mark of the memory while running the job. Additionally pathological runtime cases will be detected and reported from the metrics (for example the used walltime is just a fraction of the requested walltime). The PDF report serves for further insight into the runtime behaviour of the appropriate jobs.

At the moment the collected data will be stored in the time series database *InfluxDB*, whereas a general interface will be implemented in the future for further time series database. Furtermore in a specified configuration file the collection intervall, the database, the .netrc file, the SSL/TLS certificate and the path for the commands of the batch system and the collector for IO and network traffic are listed.

This version of *PfiTCollect* is suited for non shared clusters and towards all batch systems, since all node specific metrics (and the NFS and scratch metrics) are read from the procfs filesystem. Furthermore IO, network (Infiniband) and GPU metrices can only be determined, if the system consists of *BeeGFS*, Infiniband network and Nvidia-GPUS. In that case the metrices of every part are read out by `beegfs-ctl`, `perfquery` and `nvidia-smi`.

Goals introducing PfiTCollect

- Providing a light weight metric collector in C with small needs according to system resources,
- installation and configuration should be simple,
- source code basis should be compact.
- It's an alternative to *Telegraf*, for those administrators, who prefer a small, simple and clean written collector tool in comparison to a powerful and flexible yet big and confusing collector.

1.3.2 Telegraf Plugins

1.3.3 Aggregator

1.3.4 Reportgenerator (Text)

1.3.5 Reportgenerator (PDF)

1.3.6 Grafana Dashboard for realtime visualization

Chapter 2

Administrator's Guide

In this chapter the administrator will find details about the installation, configuration and usage of the *PfiT* system. The first four sections of this chapter treat the download, installation, configuration and operation of *Telegraf*, *PfiTCollect*, *InfluxDB* and *Grafana*. In the case of *InfluxDB*, the communication with this database, security topics of the different tools, strategies to downsize the data (retention policy, continuous quering) and backup topics will be presented. On top the aggregator, the job analysis and recommendation system as well as the report generation will be described in the sections **??** until **??**.

2.1 Telegraf

2.1.1 Download and installation

Three variants of sources to download the *PfiT* tools exist. As the consequence the installation is accordingly different:

1. Download the tools as source,
2. download the tools as a RPM/DEB package,
3. download the tools as a binary.

In the following, the three kinds of installation packages for *Telegraf* are described.

Source

The sourcefiles of *Telegraf* can be downloaded from *GitHub* from the following address: `https://github.com/influxdata/telegraf`. Here one can find the project page of the *Telegraf* (sub-)project. These project files can be downloaded for example via cloning the project with *git clone https://github.com/influxdata/telegraf*. After that a local subfolder *telegraf* should appear in the current folder which contains the project files.

RPM/DEB and Binaries

Two other package/repo formats for downloading *Telegraf* are the RPM/DEB packages and the binary files from the following address: `https://portal.influxdata.com/`

downloads. One can choose between the installation of RPM (Redhat and CentOS) and Debian packages (Ubuntu and Debian) and precompiled binaries for Windows (64 Bit), OS X, Docker and Linux (32- and 64-Bit, ARM). Steps to download and unpack are described on the corresponding download page (address see above).

2.1.2 Configuration and operation

The configuration and configuration procedure of *Telegraf* will be explained here. As already mentioned, we often used the examples from the *Telegraf* documentation (see https://github.com/influxdata/telegraf/blob/master/docs/CONFIGURATION.md) and only changed the structure of them, to adapt them to our needs. Direct citations will be marked as described in the introductory chapter. In this subsection we will introduce the configuration of the metric collector *Telegraf* via the config file *telegraf.conf*, which could be located for example in */etc/telegraf/telegraf.conf* (global). The option of configuring *Telegraf* with environment variables will be omitted. A location of the configuration file(s) different from the default location can be set with the parameter

```
telegraf --config /path/to/file/file.config
```

Using the flag *–config-directory* allows including all *.conf* files of this directory into the *Telegraf* configuration.

To create a default configuration file, the following line solves this task:

```
telegraf config > telegraf.conf
```

All default configuration values, which are available in *Telegraf*, will be written to `stdout` and in this example redirected to the file *telegraf.conf*. Since there are a lot of parameters of a lot of plugins which will be written to this configuration file, it is possible to filter them by subject to the input or output filter. The following example should demonstrate this (please see https://github.com/influxdata/telegraf/blob/master/docs/CONFIGURATION.md):

```
telegraf --input-filter cpu:mem:net:swap --outputfilter influxdb
   config > telegraf.conf
```

In this example only the sections of the input plugins *cpu, mem, net, swap* and the output plugin *influxdb* will be listed in the configuration file as well as some additional information, like global tags, agent parameters, and parameters for the processor or other plugins.

In the following we will present the most important parameters of the sections for our task. It should be noted, that some parameters are commented out by default (line is beginning with a hash (#)). To activate them remove the hash (or more) from the beginning of the line.

General

We will start with the general paragraphs "global Tags" and "agent".

[global_tags]

- dc = "example_tag" will add the given tag to all collected metrices.
- user = "example_user" sets the user for *Telegraf*.

[agent]

- `intervall` sets the data collection intervall of *Telegraf* and of all of its plugins (default: 10 s).
- `round_intervall` adjusts the `intervall`, i.e. if the intervall length was set to 10 seconds, then the collection times are :00, :10, :20, etc. and not for example :04, :14, :24, etc.
- `hostname` sets the new hostname, if the default hostname should be overwritten. If this string is empty then `os.Hostname()` should be used.
- `omit_hostname` will omit the hostname in the `host` tag in *Telegraf*, if it is set to `true` (default: `false`).

Input plugins

In the case of *PfiT*, the following input plugins are installed and collects metrics about the appropriate topics:

- cpu (system cpu usage),
- disk (disk usage),
- diskio (disk IO traffic by device),
- kernel (kernel statistics from `/proc/stat` and `/proc/sys/kernel/random/entropy_-avail`, that are not covered by other plugin metrics),
- mem (system memory usage),
- processes (get the number of processes and process information),
- swap (swap memory usage),
- system (collects metrics about system activities like system load and uptime).

Now some of the example parameters to configure, which are important for the plugins listed above, will be explained. The description often is very close to or taken directly from the description of the plugins on https://github.com/influxdata/telegraf/tree/master/plugins/inputs. Some parts are skipped, which are not marked, to do not disturb the flow of reading.

[[inputs.cpu]]

- `percpu = true|false` indicates, if per cpu statistics should be collected or not (default `true`)
- `totalcpu = true|false` indicates, if total cpu system statistics should be collected or not (default `true`)

[[inputs.disk]]

- `mount_points = ["/"]` selects the mountpoints, where statistics should be generated (and vice versa where not) (default "/")
- `ignore_fs = ["tmpfs", "devtmpfs", "devfs"]` let *Telegraf* ignore the specified file system types.

[[inputs.diskio]]

- `devices = ["sda", "sdb"]` will restrict the generation of *Telegraf* statistics to the specified devices.

The following plugins do not need explicit configuration:

- inputs.kernel
- inputs.mem
- inputs.processes
- inputs.swap
- inputs.system

Output plugins

After presenting some configuration parameters of the input plugins, we will continue to describe the parameters of the output plugins which is limited to the *InfluxDB* plugin.

[outputs.http]

- `urls = [ "http://example.org:8086" ]` sets the name or IP address of the database server.
- `database = "databasename"` sets the databasename, to which *Telegraf* should post the metricvalues (default: "telegraf"). This information is mandantory!!
- `retention_policy = "retention_policy_name"` sets the retention policy, which should be used,
- Setting User credentials (if enabled)
  - `username = "username"` (default `username = "telegraf"`),

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

  - `password = "password"` (default `password = "metricsmetricsmetricsmetrics"`).

- Setting SSL/TLS certification (if enabled)
  - `ssl_ca = "/etc/telegraf/ca.pem"` stores the certification request (??)
  - `ssl_cert = "/etc/telegraf/cert.pem"` stores the certificate,
  - `ssl_key = "/etc/telegraf/key.pem"` stores the private key of the certificate,
  - `insecure_skip_verify = true|false`; if `true` then the verification process will be skipped.

Besides these specialized parameters related to the plugins, there are further filters (*selectors* and *modifiers*), which can be used by almost all plugin types. *Selector filters* are filters which includes or excludes measurement points and *modifier filters* remove tags and fields from a measurement point. For example, they can control which fields or tags should be ignored or which ones should be recognized (please see the following reference: https://github.com/influxdata/telegraf/blob/master/docs/CONFIGURATION.md).

- `namepass` defines an array of global string patterns and only those measurement points will be emitted by *Telegraf*, which measurement name matches these patterns (selector filter).
- `namedrop` defines an array of global string patterns and only those measurement points will *not* be emitted by *Telegraf*, which measurement name matches these patterns (the inverse of namepass; selector filter).
- `fieldpass` defines an array of global string patterns and only fields of measurement points will be emitted, which field key name matches these patterns (modifier filter).
- `fielddrop` defines an array of global string patterns and fields of measurement points will *not* be emitted, which field key name matches these patterns (the inverse of fieldpass; modifier filter).
- `tagpass` defines an array of global string patterns and only those measurement points will be emitted, which tag key matches one of the patterns (selector filter).
- `tagdrop` defines an array of global string patterns and those measurement points will *not* be emitted, which tag name matches one of the patterns (the inverse of tagpass; selector filter).
- `taginclude` defines an array of global string patterns and only those tags of the measurement points will be emitted, which matches the ones in the array.
- `tagexclude` defines an array of global string patterns and those measurement points won't be emitted, which matches the ones in the array.

To exemplifiy these filter parameters, we will have a look at two examples, which are taken from the *Telegraf* documentation (please see the following reference: `https://github.com/influxdata/telegraf/blob/master/docs/CONFIGURATION.md`).

In the first example the `fielddrop` and `fieldpass` keyword are presented:

```
# Drop all metrics for guest & steal CPU usage
[[inputs.cpu]]
  percpu = false
  totalcpu = true
  fielddrop = ["usage_guest", "usage_steal"]

# Only store inode related metrics for disks
[[inputs.disk]]
  fieldpass = ["inodes*"]
```

The second example demontrates the use of `tagdrop` and `tagpass`:

```
[[inputs.cpu]]
  percpu = true
  totalcpu = false
  fielddrop = ["cpu_time"]
  # Dont collect CPU data for cpu6 & cpu7
  [inputs.cpu.tagdrop]
    cpu = [ "cpu6", "cpu7" ]

[[inputs.disk]]
  [inputs.disk.tagpass]
    # tagpass conditions are OR, not AND.
    # If the (filesystem is ext4 or xfs) OR (the path is /opt
    # or /home) then the metric passes
    fstype = [ "ext4", "xfs" ]
    # Globs can also be used on the tag values
    path = [ "/opt", "/home*" ]
```

Further examples can be found at the above given address.

Additional Telegraf plugins written within the project

In this subsection we will describe the PfiT-JobID plugin, which is a processor plugin to append the JobID to every measurement point. It determines the batch job IDs from a file with the name and format

```
pfit_
```

ideally generated by the batch system via a prologue script. The path to this file and the delimiter within its name for the userid and the job ID's are configurable.

The following paragraph of the configuration file has to be added:

```
# Add the batch job ID's as tags.
[[processors.pfit-jobid]]
  ## directory which contains the job info file
  ## (default is "/run/pfit/")
  jobinfofileDir = "/run/pfit/"
  ## delimiter for the batch job information in the job info file
  ## (default is "_")
  delimiter = "_"
```

Tags: The job ID's are added as jobid1 and jobid2 tags (and as jobid3 and jobid4 tags and so on, if there are several job info files available in case of shared nodes) by this processor plugin.

After installing the new *Telegraf* binary, create a new *telegraf.conf* file, e.g.:

```
./telegraf config > ../etc/telegraf.conf.new
```

and activate the pfit-jobid plugin in the "PROCESSOR PLUGINS" section by removing the comment before "[[processors.pfit-jobid]]". If necessary, configure the `jobinfofileDir` and the `delimiter`. Do not forget to activate the other plugins you are using, as well.

2.1.3 Installation

After having described the download and configuration procedure of the *Telegraf* tool, in this section the different procedures to install the tools from source, RPM/DEB package and from binary file(s) will be introduced. Notice that we concentrate only on the installation on HPC clusters and leave the description of the installation on PCs, workstations, etc. aside (but it's similar). Furthermore, having administrator status as "root" is often necessary in order to install the tools successfully.

Before the installation process can start, two requirements have to be fulfilled:

- If using *Telegraf*'s service plugins it is often necessary to use additional custom ports, which can be configured in the configuration file of *Telegraf* (for example in /etc/telegraf/telegraf.conf).
- *InfluxData* strongly recommends the usage of the NTP (Network Timer Protocol) to synchronize times between hosts and to get unified time stamps for all hosts.

Installation of Telegraf

The installation process of *Telegraf* will be described by the example of a Redhat/CentOS distribution, because these distributions were stated in the survey of 80 % of the participating institutions (please see `https://profit-hpc.de/wp-content/uploads/2017/08/profithpc_deliverable_1-2_final.pdf`, p. 7). The following installation procedure for *Telegraf* is cited here from the *Telegraf* documentation on `https://docs.influxdata.com/telegraf/v1.5/introduction/installation/` for a *RedHat* and *CentOS* package.

Install the latest stable version of *Telegraf* using the *yum* package manager:

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

```
cat << EOF | sudo tee /etc/yum.repos.d/influxdb.repo
[influxdb]
name = InfluxDB Repository - RHEL \$releasever
baseurl  = https://repos.influxdata.com/rhel/$releasever/
    $basearch/stable
enabled  = 1
gpgcheck = 1
gpgkey   = https://repos.influxdata.com/influxdb.key
EOF
```

Once the repository is added to the yum configuration, install and start the *Telegraf* service by running:

```
sudo yum install telegraf
sudo service telegraf start
```

Or if your operating system is using systemd (CentOS 7+, RHEL 7+):

```
sudo yum install telegraf
sudo systemctl start telegraf
```

Installation of Telegraf plugins

The *PfiT-JobID processor plugin* simply adds the job ID's of a batch job to every metric passing through it. It determines the batch job ID's from a file with the name and format "pfit_" , ideally generated by the batch system via a prologue script. The path to this file and the delimiter within its name for the userid and the job ID's are configurable.

In the following lines we will describe the installation procedure of the *PfiT-JobID processor plugin*, but this procedure is a general installation guide for all plugins.

- First, add the PfiT-JobID directory and its content to the plugins/processors subdirectory in the *Telegraf* source code (workspace).
- Import the PfiT-JobID directory in plugins/processors/all/all.go (workspace).
- Rebuild *Telegraf* by running "make" in the *Telegraf's* main source directory. (For building a new RPM package, use the build.py script in the scripts subdirectory, e.g.: ./scripts/build.py –package –platform=linux –arch=amd64 –version=2017-11-23)

The new RPM packages can then be found in the telegraf/build subdirectory.

2.2 PfiTCollect

PfiTCollect is a lightweight implementation of a metric collector for (currently) only non shared clusters collecting metrics of the following areas:

- CPU,
- memory,
- swap,
- system (for example load),
- IO (scratch, NFS, BeeGFS (via beegfs-ctl)),
- Network/Infiniband (via perfquery),
- GPU (just Nvidia (via nvidia-smi)).

2.2.1 Download

In contrast to the other tools, *PfiTCollect* is provided only as source code. The downloadaddress is `https://profit-hpc.de/downloads/`. The installation of *PfiTCollect* will be described in the next section.

2.2.2 Installation

To install *PfiTCollect* the following requirements to install *PfiTCollect* are needed:

- GNU Make,
- gcc 4.8 or higher (or other C-Compiler),
- beegfs-ctl (tool to collect BeeGFS metric data),
- perfquery (tool to collect Infiniband metric data),
- nvidia-smi (tool to collect GPU metric data),
- mounted *procfs*,
- Running InfluxDB instance.

The *PfiTCollect* metric collector can be installed as every other program, which is installed on the cluster. No specific things have to be considered.

2.2.3 Configuration and operation

The configuration of the *PfiTCollect* program is very simple. The configuration will be done with the help of a configuration file, which overwrites the default values of the program and which is per default hardcoded as `/etc/pfitcollect.config`. If another path should be used, then *PfiTCollect* will have to be called in the following way:

```
pfitcollect --config /path/to/configfile
```

There are only some parameters which have to be set:

- *interval length*, which defines the temporal distance between two measurement points (in integer seconds),
- *address of influxdb node* is the fully qualified domain name or the IPv4 address of the *InfluxDB* server without the port number (at the moment just one *InfluxDB* server is supported). Examples are: influxdb.server or 112.56.56.23.
- *influxdb database*: Name of the database in *InfluxDB* which is used by *Telegraf* and *PfiTcollect*.
- *full path netrc file*: Credentials to login to *InfluxDB* with the following structure:
  - `machine` machinename
  - `login` loginname
  - `password` password
- *full path certificate file*: Full path to the certificate file (including the certificate file name).
- *full path to beegfs-ctl command*: Full path to the `beegfs-ctl` command file (without the `beegfs-ctl` command).
- *full path to perfquery command*: Full path to the `perfquery` command (without the `perfquery` command).
- *full path to nvidia-smi command*: Full path to the `nvidia-smi` command (without the `nvidia-smi` command).
- full path to temporary pfitcollect path: Full path to the path, where temporary files are stored.

2.3 InfluxDB

Analogously to *Telegraf*, *InfluxDB* can be downloaded as follows:

2.3.1 Download

Source

As in the case of *Telegraf*, the source files can be downloaded from *GitHub*: https://github.com/influxdata/influxdb. Cloning the InfluxDB project can be done by executing *git clone https://github.com/influxdata/influxdb*. After downloading, a subfolder *influxdb* will be created, which contains all project files.

RPM/DEB and Binaries

InfluxDB RPM/DEB packages and binary files can be downloaded from the following address: https://portal.influxdata.com/downloads. Again, one can choose between the download of RPM (Redhat and CentOS), Debian packages (Ubuntu and Debian) and precompiled binaries for Windows (64 Bit), OS X, Docker and Linux (32- and 64-Bit, ARM). For every selection the steps to download them are described on the above stated webpage.

2.3.2 Installation

Before the installation process can start, two requirements have to be fulfilled:

- The ports, which will basically be used, are
  - port 8086 (client server communication over the *InfluxDB* HTTP API)
  - and port 8088 (RPC service for backup and restore).

  If using *InfluxDB* with other plugins it is possible, that other ports in addition to these ones are required (custom ports).
- It is recommended by *InfluxData* to use the NTP (Network Timer Protocol) to synchronize times between hosts, to get unified time stamps for all hosts. If this synchronzation step is omitted, inaccurate time steps can be the result.

The installation process of InfluxDB will be described by the example of a Redhat/CentOS distribution, because these distributions were stated in the survey of 80 % of the participating institutions (please see https://profit-hpc.de/wp-content/uploads/2017/08/profithpc_deliverable_1-2_final.pdf, p. 7). Again, the following lines are citations from the *InfluxDB* documentation on https://docs.influxdata.com/influxdb/v1.7/introduction/installation/ for *RedHat* and *CentOS*.

Quote

Red Hat and *CentOS* users can install the latest stable version of *InfluxDB* using the *yum* package manager:

```
cat << EOF | sudo tee /etc/yum.repos.d/influxdb.repo
[influxdb]
name = InfluxDB Repository - RHEL \$releasever
baseurl  = https://repos.influxdata.com/rhel/$releasever/
    $basearch/stable
```

```
enabled  = 1
gpgcheck = 1
gpgkey   = https://repos.influxdata.com/influxdb.key
EOF
```

Once repository is added to the yum configuration, install and start the *InfluxDB* service by running:

```
sudo yum install influxdb
sudo service influxdb start
```

Or if your operating system is using systemd (CentOS 7+, RHEL 7+):

```
sudo yum install influxdb
sudo systemctl start influxdb
```

End quote

2.3.3 Configuration and Running

With respect to the hardware, *InfluxData* recommends using 2 SSD disks for *InfluxDB*, one for the Write Ahead Log (WAL) and the other one for storing the data. Furthermore, each machine should have at least 8 GB RAM.

There are three ways of configuring *InfluxDB*:

- Configure *InfluxDB* via the configuration file (in general *influxdb.conf*),
- configure *InfluxDB* via environment variables and
- configure *InfluxDB* via the default values of *InfluxDB*,

while the precedence is from top to bottom. In Linux the global location of the configuration file often is /etc/influxdb/influxdb.conf, but other paths are admissible.

To start *InfluxDB* and to read in the configuration file there are two possible ways:

- influxd -config /etc/influxdb/influxdb.conf (read in the configuration data from the file),
- set the environmentvariable INFLUXDB_CONFIG_PATH and start the *InfluxDB* service *influxd*.

In the following we will present the most important configuration parameters, which we will discuss using the configuration file as an example (please see https://docs.influxdata.com/influxdb/v1.7/administration/config/ and https://github.com/influxdata/influxdb/blob/master/etc/config.sample.toml. The important sections of the file for our concerns are:

- Metastore ([meta]; meta data settings),
- Data ([data]; data settings),
- Retention Policies ([retention policy]; retention policy settings),
- Continuous queries ([ontinuous_queries]; continuous queries settings),
- HTTP endpoints ([http]; HTTP endpoints settings),
- others.

The equaivalent environment variables are listed, too. It is important, that the user running influxd has write permissions to the data directories and to the Write Ahead Log (WAL) and permissions to update the configuration file for every *InfluxDB* instance in the cluster.

For every section of the configuration file, we list the most important parameters for our purposes with small explanations.

Metastore settings ([meta])

- `dir` = "`/var/lib/influxdb/meta`"
  - This directory contains the meta data of *InfluxDB* (including meta.db).
  - The location of the directory may be change.
  - Environment variable: `INFLUXDB_META_DIR`.
- `retention-autocreate=true`
  - On creating the database the default retention policy "autogen" will be created.
  - The data in the retention policy "autogen" has an infinite lifetime.
  - This retention policy will be applied if no other retention policy is prescribed at read or write actions to/from *InfluxDB*.
  - If this retention policy should not be applied, set above variable to false.
  - Equivalent environment variable: INFLUXDB_META_RETENTION_AUTOCREATE.

Data settings ([data])

This section of the configuration file of *InfluxDB* controls, where the data of *InfluxDB* will be stored and how it is flushed from the WAL (Write Ahead Log).

- `dir` = "`/var/lib/influxdb/data`"
  - This is the default linux directory where the *InfluxDB* database stores the data (where the TSM engine stores the TSM values). This may be change if another directory is needed.

  - Environment variable: `INFLUXDB_DATA_DIR`.

- `wal-dir = "/var/lib/influxdb/wal"`
  - This directory is the default directory to store the WAL data (Write Ahead Log).
  - Environment variable: `INFLUXDB_DATA_WAL_DIR`.

- `max-series-per-database = 1000000`
  - This parameter sets the maximum number of series in the database before series will be deleted from the database, in which the default value is 1 million.
  - Setting `max-series-per-database = 0` allows infinite number of series.
  - Environment variable: `INFLUXDB_DATA_MAX_SERIES_PER_DATABASE`.

- `max-values-per-tag = 100000`
  - This parameter defines the maximum number of tag values per tag key, in which the default value is 100000.
  - Setting this parameter to 0 allows infinite tag values per tag.
  - Environment variable: `INFLUXDB_DATA_MAX_VALUES_PER_TAG`.

Retention policy settings [retention]

In the *Retention policy settings* section the values concerning the parameters are stored, which control the deletion of old data in the database.

- `enabled = true`
  - Enables (true) or disables (false) the retention policy.
  - Environment variable: `INFLUXDB_RETENTION_ENABLED`.

- `check-interval = "30m0s"`
  - In the default setting, *InfluxDB* checks every 30 minutes the fulfillment of the Retention Policy.
  - Environment variable: `INFLUXDB_RETENTION_CHECK_INTERVAL`.

HTTP endpoint settings [http]

In this section enabling HTTPS and authentication will be described. More generally in this section the interplay between InfluxDB and the HTTP endpoints will be presented.

- `enabled = true`
  - Enables (true) or disables (false) the HTTP endpoint. Please be careful with disabling the HTTP endpoint, because the *InfluxDB* uses the HTTP API to connect to the database.
  - Environment variable: `INFLUXDB_HTTP_ENABLED`.

- `bind-address = ":8086 "`
  - InfluxDB uses the above given (bounded) port to communicate with the HTTP service.
  - Environment variable: `INFLUXDB_HTTP_BIND_ADDRESS`
- `auth-enabled = false`
  - The parameter *false* unsets the authentication over HTTP/HTTPS, while setting it to *true* will be enable it.
  - Environment variable: `INFLUXDB_HTTP_AUTH_ENABLED`.
- `log-enabled = true`
  - Setting this parameter enables or disables HTTP requests logging.
  - Environment variable: `INFLUXDB_HTTP_LOG_ENABLED`.
- `https-enabled = false`
  - Enables (true) or disables (false) HTTPS.
  - Environment variable: `INFLUXDB_HTTP_HTTPS_ENABLED`.
- `https-certificate = "/etc/ssl/influxdb.pem"`
  - Set the path to the SSL/TLS file when HTTPS is enabled.
  - Environment variable: `INFLUXDB_HTTP_HTTPS_CERTIFICATE`.
- `https-private-key = ""`
  - Use a separate private key location. If only the https-certificate is specified, the httpd service will try to load the private key from the https-certificate file. If a separate https-private-key file is specified, the httpd service will load the private key from the https-private-key file.
  - Environment variable: `INFLUXDB_HTTP_HTTPS_PRIVATE_KEY`.
- `max-connection-limit = 0`
  - This parameter sets the maximum number of HTTP connections (0 is unlimited)
  - Environment variable: `INFLUXDB_HTTP_MAX_CONNECTION_LIMIT`.
- `max-body-size = 25000000`
  - This number (in bytes) specifies the size of the message body of a client request.
  - Environment variable: `INFLUXDB_HTTP_MAX_BODY_SIZE`.

Logging settings [logging]

- `level = “info “`
  - The logging level parameter determines the verbosity of the output and the output cases. The possible logging level values are: error, warn, info, and debug, in which info is the default level.
  - It's possible to choose more than one level.
  - Environment variable: `INFLUXDB_LOGGING_LEVEL`.

Continuous queries [continuous_queries]

In this subsection the parameters concerning the continuous queries will be presented and illustrated.

- `enabled = true`
  - enables (true) or disables (false) continuous quering.
  - Environment variable: `INFLUXDB_CONTINUOUS_QUERIES_ENABLED`.
- `run-interval = “1s“`
  - Sets the run interval after which checking for continuous quering is needed. Set it to the lowest interval value a continuous quering should run.
  - Environment variable: `INFLUXDB_CONTINUOUS_QUERIES_RUN_INTERVAL`.

There are further configuration sections and parameters, but they are of minor importance for our goal. If the user is interested in those parameters, please have a look at https://docs.influxdata.com/influxdb/v1.7/administration/config/.

2.4 Grafana

Although *Grafana* is not a tool of the TICK-Stack, the download procedure is analogous to the foregoing tools.

2.4.1 Download

Source

Grafana is an open source project and the project page is located on *GitHub*: https://github.com/grafana/grafana. It can be downloaded, for example, via cloning the project using the following command: *git clone https://github.com/grafana/grafana.git*. As in the foregoing cases, we then get a subfolder *grafana* with the source and project files in it.

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

RPM/DEB and Binaries

In the case of Grafana the RPM/DEB and binary files can be downloaded from the following address: `https://grafana.com/grafana/download/`. On this page one can again choose between different operating systems (Linux, Windows, Max OS X), the download of an RPM (Redhat and CentOS) or a Debian package (Ubuntu and Debian) and precompiled binaries for Windows (64 Bit), OS X, Docker and Linux (32- and 64-Bit, ARM). For every selection the steps to download them, are described.

2.4.2 Installation

See installation instructions for the appropriate operating system at website:
https://grafana.com/docs/installation.

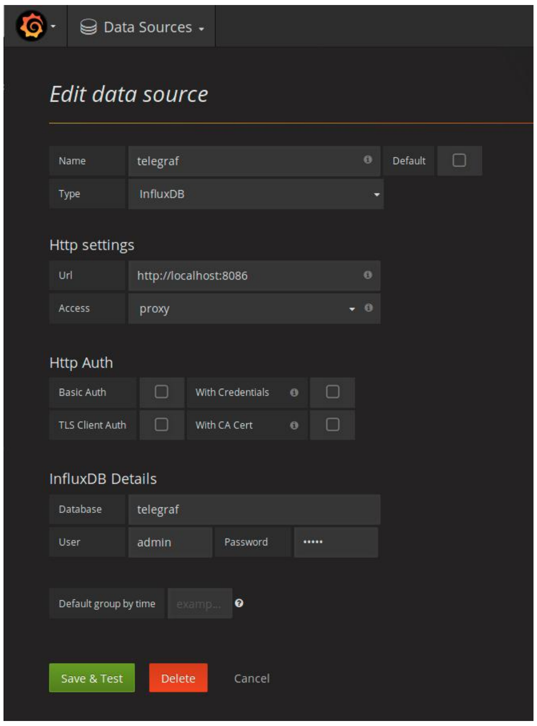
2.4.3 Configuration and Running

After installing *Grafana* on the designated server, the core configuration can globally be done for example in the file `/etc/grafana/grafana.ini` and this configuration is dependent upon your own security specifications. In our case we are using the default settings (for example the port number is 3000).

The dashboards have been developed using *Grafana* V7.0.5. No special configuration needs to be implemented for use with the *PfiT*-monitor.

To import the *pfit-influxdb* database, choose in the upper left menu the following item (please see picture 2.1):

- Data Sources → Add data source



Edit data source

Name: telegraf   Default: ☐
Type: InfluxDB

Http settings
Url: http://localhost:8086
Access: proxy

Http Auth
Basic Auth: ☐   With Credentials: ☐
TLS Client Auth: ☐   With CA Cert: ☐

InfluxDB Details
Database: telegraf
User: admin   Password: •••••

Default group by time: examp...

Figure 2.1: Interactive integration of pfit-database in Grafana

Here the settings have to be made according to the corresponding *InfluxDB* instance for the *PfiT* monitor (for example database name (here *telegraf*), database type (here *InfluxDB*), HTTP setting, user credentials). Especially notice the label *Name* in which you define the *PfiT* database. This name must be used when importing the *Grafana* json dashboards.

The *Grafana* json dashboard files can be installed interactivly from `http://docs.grafana.org/reference/export_import/` with *Grafana*.

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

Each of the 4 json's dashboards (available at https://gitlab.gwdg.de/profit-hpc/pfit-grafana)

1. ProfiT-HPC-General.json (Home dashboard)
2. ProfiT-HPC-Job.json (Job dashboard)
3. ProfiT-HPC-Node.json (Node dashboard)
4. ProfiT-HPC-User.json (User dashboard)

have to be imported manually and interactively with using the menu button (see picture 2.2, left top):

- dashboards → import → upload .json file,

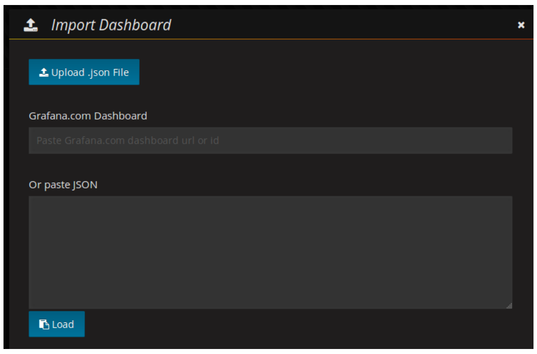
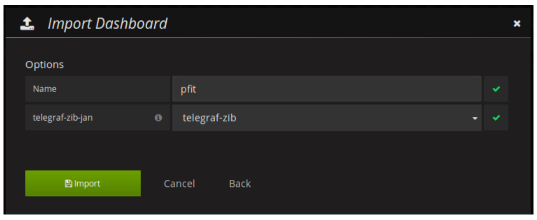


Figure 2.2: Interactive json import in Grafana

and the corresponding *PfiT-InfluxDB* database has to be included for each dashboard.

The json file can be uploaded and you must select the appropriate PfiT-InfluxDB data source, which has already been integrated into *Grafana*.

Please note that some dashboards are linked in tables and the links should be changed accordingly to the path which was assigned by Grafana after the import.



Options

Name	pfit
telegraf-zib-jan	telegraf-zib

Figure 2.3: Interactive integration of pfit-database in pfit dashboards

2.5 Aggregator

The tool aggregates data in a format which is suitable for generating reports.

This tool also contains the script to export jobinfo data into DB from the post execution script.

Prerequisites

The aggregator needs particular software packages to be installed along with data available in DB.

- Python 3
- InfluxDB. Currently the aggregator works only with InfluxDB, so it should be installed and the aggregator should be configured as shown in configuration section below.
- DB should contain all required data specified in DB spec(https://gitlab.gwdg.de/profit-hpc/aggregator/-/blob/master/docs/InfluxDBspec.md).

You should also install Python dependencies with:

```
python3 -m pip install -r requirements.txt
```

Configuration

Samples for configurations can be stored in the repository, please rename corresponding templates with an extension `.sample` to `.py` and change placeholder values accordingly. Real configs should be ignored in `.gitignore` file. Sample configs must not be used in the code. Example: `influxdb.sample` → `influxdb.py`

Usage

The main executable of the aggregator module is data.py. You can type `./data.py -h` for more help.

```
usage: data.py [-h] [-t {text,pdf}] JOBID

Gets the job information required for generating text or PDF
   reports
and outputs it in JSON format.

positional arguments:
JOBID                 job ID used in the batch system

optional arguments:
-h, --help            show this help message and exit
-t {text,pdf}, --type {text,pdf}
```

```
type of the output (default: text)
```

Get test output

In order to get json output from the test data, located at test/data, a docker container with InfluxDB instance and imported data should be running. In order to build the docker container with necessary test data, run the following script:

`test/docker/influxdb/build_influxdb.sh`

To run the InfluxDB container, simply execute:

`test/docker/influxdb/run_influxdb.sh`

After the influxDB instance is up you need to configure the aggregator to use it, by copying `influxdb.sample` → `influxdb.py` and editing it as:

```
IDB = {
    "username": "",
    "password": "",
    "database": "pfit",
    "api_url": "http://localhost:58086",
    "ssl": False,
}
```

As an example the following command will output the test data in the json format for a pdf report:

`./data.py -t pdf 1352076`

You can use other JOBIDs which you can find in test/data directory. Note: Test data with JOBID 2368599 doesn't include Infiniband metrics, therefore it is required to switch off InfluxDB support in the configuration file(`conf/config.py`) before using it.

Export job info

In order to export the job info with JOBID you should call `export.py`:

`export.py JOBID`

Then the aggregator will gather a job information with ID JOBID from the batch system configured in `/conf/config.py` and save it into the configured database as a pfit-jobinfo and additionally in pfit-jobinfo-alloc measurements.

2.6 Report generation

The ProfiT-HPC toolkit delivers several types of reports - for different audiences, with different detail levels and on the basis of different platforms. The *ASCII report* is the most basic report from, reporting monitoring metrics and batch system information on the given job. From these very basic measurements, first recommendations regarding the resource usage of the jobs are made.

Furthermore, several Grafana dashboards have been created for a *Job View*. These enable the user to monitor their jobs to a very fine-grained level of detail. At the moment, these dashboards are only available for the system administrator but will in future also be exported to the users. In addtion to these user dashboards, a more general *Admin View* has been created, that gives the administrator an overview on all finished jobs, from where he can go into the detailed job views. In future, there will be additional reports, e.g., a pdf report. Here, additional profiling information will be delivered to the user. This

2.6.1 Textreport

As described in the introduction, the ASCII report delivers information on basic monitoring metrics and batch system information with additional recommendations for the user by the analysis of the job. For this, several scripts are started in the postexecution phase of the job[1]:

- `pfit_postexec.sh`
  This shell script is executed in the postexecution phase of the job. Within this script, the following Python script are started to gather the collected metrics and then generate the Text report from of these metrics.

- `pfit_get_metrics.py`
  This script gathers the collected metrics from the desired database (currently only *InfluxDB* supported) and writes them into a temporary file `pfit_metrics_tmp_${PFIT_JOBID}`.

- `pfit_ascii_report_generator.py`
  This sript generates the Text report. It takes a metrics file and several options as input and writes the generated report to a configurable path or it will sent by E-Mail or to `stdout`.

Prerequisites Text report

For the Python scripts to work, the following prerequisites must be met:

- Python version 3.5 or higher,
- Python modules used:
  - requests (`pfit_get_metrics.py`),
  - argparse (`pfit_get_metrics.py`, `pfit_ascii_report_generator.py`),
  - time (`pfit_ascii_report_generator.py`),
  - sys (`pfit_get_metrics.py`, `pfit_ascii_report_generator.py`),
  - json (`pfit_get_metrics.py`, `pfit_ascii_report_generator.py`),
  - math (`pfit_ascii_report_generator.py`),
  - textwrap (`pfit_ascii_report_generator.py`).
- bash

[1]depending on the batch system you are using, this might also be the *Epilogue*

Configuration

Two files need to be edited in the configuration:

1. `pfit_postexecution.sh`, and
2. `pfit_config.sh`.

In the first, the path to the configuration file `pfit_config.sh` has to be inserted. In the latter, all other configuration details are entered:

- `PFIT_TBL_T`: threshold which defines the number of nodes for which the ASCII report view is switched from a tabular view to a summarized view.
- `PFIT_NODE_USAGE` defines whether the nodes on a cluster are `shared` or `non-shared`. This choice has influence on the generated ASCII report (see User's Guide)
- `PFIT_RESOURCE` defines in which format the compute resources are passed from and to the batch system. Possible values are:
  - `slot`,
  - `core`,
  - `node`,
  - `proc`,

  and it defaults to `proc`.
- `PFIT_BATCH_SYSTEM` which batch system is used on the cluster. Possible values are `LSF`, `TORQUE`, `SLURM`.
- `PFIT_TMP_PREFIX`: Path to temporary files written by the toolkit.
- `PFIT_REPORT_PREFIX`: Path to final reportfile written by the toolkit.
- `PFIT_PREFIX` path to pfit files (for example, python scripts for Textreport generation.
- `PFIT_IDB`: Boolean value whether *InfluxDB* is used.
- `PFIT_IDB_ADDRESS`: Query address of *InfluxDB* (for example https://url.of.database/query).
- `PFIT_IDB_UNAME`: User name for *InfluxDB*.
- `PFIT_IDB_PW`: Password for *InfluxDB* user, OR
- `PFIT_IDB_PW_FILE`: Path to the file with the InfluxDB password *(not completely implemented yet)*.
- `PFIT_IDB_NAME`: Name of *InfluxDB* database.
- `PFIT_FIELD_SEP`: Separator for fields in temporary files. Needs to be different from all symbols in stored metrics.
- `PFIT_NAME_SEP` Separator for file names in temporary files

Further configuration options are being implemented and will be added as applicable.

Setup

The batch system now needs to be instructed to run the script `pfit_postexecution.sh` in every postexecution phase of a job (epilogue).

2.6.2 PDF report

Similar to the ASCII report generator, the PDF report generator python executable script, `<pdfreport>/pdfgen/pfit_pdfreport.py`, creates a formatted and printable PDF-file (DINA4) for each job, where <pdfreport> denotes the home directory of the PDF report generator. The output file displays information on basic hardware, monitoring metrics, batch system and recommendations. It is divided into four parts. The first three include information concerning the specific job being analysed. The last part supplies general information.

Now we describe the parts in more detail. The first part contains information concerning the job execution in the section labeled "Job Overview" and about the hardware allocated for the job in the section labeled "Node Information". The section "Job Overview" lists the unique job identification number, the user's name, the name of the queue of the scheduler, the number of allocated nodes and requested number of cores and time information (duration requested, duration used, start time, completion time). The hardware information includes the CPU model, amount of total memory, number of sockets, number of CPU cores per socket, number of threads per core and a node name listing.

Also included in the first part of the report is a bar chart diagram called the "Global Summary of Resource Usage". On the left side, the name of the parameter of interest is listed. On the right side the actual values and units are printed. Each bar in the diagram compares two values. The walltime compares the requested time to the used time; the CPU usage compares the average CPU usage to a maximum value; the memory sum compares sum of maximum memory used during execution to the total allocated sum; swap sum shows only the maximum swap value. Further values are optional, meaning they appear only if the data are available. For example, the network traffic is compared (receiving and transmitting) and the I/O is compared (read and write).

At the bottom of the first page of the report, the recommendations from the recommendation system are listed.

The second part of the PDF report contains information about the node distribution of several metrics. The grafical representation of this information helps give a quick view of these distributions. The values to be displayed can be defined in the configuration file, along with the units of the recorded measurements in the json file and titles for the diagrams; see the next sections of this document. The units are then converted into human readable values by the generator and shown in the diagrams. In the present version, conversions are provided for the following units:

- Bytes to KB (Kilobytes), MB (Megabytes), GB (Gigabytes) or TB (Terabytes)

- Bytes/s to KB/s (Kilobytes), MB/s (Megabytes), GB/s (Gigabytes) or TB/s (Terabytes)
- s (seconds) to m (minutes), h (hours) or d (days)
- Counts to Counts*1.E-03, Counts*1.E-06, Counts*1.E-09 or Counts*1.E-12

Note that any key word not listed here will not be converted in any way.

The third part of the report contains the time series plots per node of all metrics which are included in the job's corresponding json file. The units of the metrics and the names of the plots can be defined in the configuration file, see section next sections of this document. The node names and color code are listed at the bottom of each page of plots.

The last part contains a so-called "key", where the user finds definitions of the parameters which appear in the report.

If GPU's are available on the allocated nodes, additional information is included on the global summary on the first page and on additional pages of the pdf report following the standard report. These additional pages include bo x plots of the statistical distributions on the GPU's as well as time series plots for each GPU. The last page contains a further "key" describing all G PU related parameters included in the report.

As input, the PDF report generator receives only a single json file for each job. This file is created for the PDF report generator in the toolkit by the aggregator in section 2.5.

Prerequisites

- Python, version 3.5 or higher
- Python modules used:
  - json
  - numpy, version 1.15.4 or higher
  - matplotlib, version 3.0.2 or higher
  - pandas, version 0.23.4 or higher
  - reportlab, version 3.3.0 or higher
  - svglib, version 0.9.0 or higher

Configuration

The configuration file, <pdfreport>/conf/pdfreport.conf, is used to customize the PDF report, for example, to define the units and names for metrics. If the file does not exist, default values are used. Use of the configuration file is described in the example below.

Example configuration file:

```
## This file contains the following:
##   Four string values using ';' as a separator
##   Notice that blanks are seen as characters, not separators
## <plot-type>;<variable-name>;<units>;<plot-label>
##
## 1.
##   List which node data statistics should be plotted as
   barcharts
##   (second part of pdf-report), corresponding units and caption

bar;cpu_usage;%;CPU usage
bar;mem_rss_max;Bytes;Memory MaxRSS
bar;mem_rss_avg;Bytes;Memory AveRSS
bar;mem_swap_max;Bytes;Memory Swap
bar;read_bytes;Bytes;I/O Read Size
bar;write_bytes;Bytes;I/O Write Size
bar;read_count;Counts;I/O Read Count
bar;write_count;Counts;I/O Write Count
bar;cpu_time_user;s;CPU time user
bar;cpu_time_system;s;CPU time system
bar;cpu_time_idle;s;CPU time idle
bar;cpu_time_iowait;s;CPU time iowait
bar;ib_rcv_max;Bytes/s;Max IB receive rate
bar;ib_xmit_max;Bytes/s;Max IB transmit rate

## 2.
##   Define units and caption of timeseries plots (third part of
##   pdf-report)
##   For values not listed here, an empty and the variable name
##   will be used, respectively.

timeseries;proc_cpu_usage;%;CPU usage
timeseries;proc_mem_rss_sum;Bytes;Memory RSS
timeseries;node_cpu_usage;%;Node CPU usage
timeseries;node_load;;Node Load1
timeseries;node_load5;;Node Load5
timeseries;proc_read_sum;Counts;Sum Read Requests
timeseries;proc_write_sum;Counts;Sum Write Requests
timeseries;node_ib_rcv_max;Bytes/s;IB max. receive rate
timeseries;node_ib_xmit_max;Bytes/s;IB max. transmit rate

## 3. Define number of time series plots per page
nsubpp;4
```

Program execution

After installation of all dependencies, `pfit_pdfreport.py` can be executed with the line command:

```
python3 <pdfreport>/pdfgen/pfit_pdfreport.py \
  <path-of-json-file> <tmp-dir> <pdf-report-dir>
```

where

- <pdfreport> is the home path of the PDF report generator
- <pdfreport>/pdfgen/`pfit_pdfreport.py` is the path to the python executable script
- <path-of-json-file> is the path to the json file containing the job-ID-data (see <pdfreport>/docs/spec-input)
- <tmp-dir> is the directory for temperary ouput files
- <pdf-report-dir> is the directory for the PDF report output file

Specifications and Example

An example input json file named <pdfreport>/testdata/`test-ib_1352076.json` has been included in the installation for testing purposes. This file has been created from a test influxdb databank using the aggregator in section 2.5 and according to the following specifications:

```
'''

# Input specification

Here is the input specification for PDF report.

    {
        "job": {
                "job_id": STR,
                "user_name": STR,
                "used_queue": STR,
                "submit_time": INT,
                "start_time": INT,
                "end_time": INT,
                "used_time": INT,
                "requested_time": INT,
                "requested_cu": INT,
                "num_nodes": INT,
                "recommendations": STR,
        },

        "nodes": [
              {
                "node_name": STR,
                "cpu_time_user": INT,
                "cpu_time_system": INT,
                "cpu_time_idle": INT,
                "cpu_time_iowait": INT,
                "write_bytes": INT,
                "write_count": INT,
                "read_bytes": INT,
                "read_count": INT,
                "mem_rss_max": INT,
                "mem_rss_avg": INT,
                "mem_swap_max": INT,
                "mem_vms": INT,
                "cpu_usage": NUM,
                "cpu_model": STR,
                "sockets": INT,
                "cores_per_socket": INT,
                "phys_threads_per_core": INT,
                "virt_threads_per_core": INT,
                "available_main_mem": INT,
                "num_cores": INT,

                "dynamic": {
                  "node_cpu_usage": {
                    "delta": INT,
                    "data": [NUM,]
                    },
                  "node_load": {
                    "delta": INT,
                    "data": [NUM,]
                    },
                  "proc_cpu_usage": {
                    "delta": INT,
                    "data": [NUM,]
                    },
                  "proc_mem_rss_sum": {
                    "delta": INT,
                    "data": [INT,]
                    },
                },
              }
        ],
    }

## job

Contains information about the job.

----------------------------------------------------------------
|  name           | type                    | comment            |
|  -------------  | ----------------------- | ------------------ |
|  job_id         | String, valid JOBID     | Job's ID           |
|  user_name      | String                  | username of the    |
|                 |                         | jobstarter         |
|  used_queue     | String                  | Queue in which job |
|                 |                         | was submitted      |
|  submit_time    | Integer, Unix timestamp | Submission date    |
|   ...               | ...                         | ...                |
|                     |                             |                    |
| recommendations     | List of strings from        | ["string(1)",...,  |
|                     |                             | "string(n)"] where |
|                     |                             | string(i) corre-   |
|                     | aggregator and              | sponds to a line of|
|                     |                             | text and has maxi- |
|                     |                             | mum of 98 charaters|
|                     | recommendation system       | Notice that this   |
|                     |                             | variable is not    |
|                     |                             | required           |
---------------------------------------------------------------------

## nodes

Contains data aggregated by node.

---------------------------------------------------------------------
| name                     | type                | comment              |
| ----------------------   | ------------------- | -------------------- |
| node_name                | String              | Node name            |
| cpu_time_user            | Integer, Seconds    | Amount of time CPU   |
| available_main_mem       | Integer, Bytes      | spent on user process|
| available_main_mem       | Integer, Bytes      | The memory size of   |
|                          |                     | each node in bytes   |
| ...                      | ...                 | ...                  |
---------------------------------------------------------------------

## nodes.dynamic

Contains timeseries data aggregated by node.

Measurements start at time `job.start_time` and `delta` is the
difference between timestamps of 2 consecutive Measurements.
`data` is a list which consists of consecutive measurements.

If measurement does not exist then there should stay `None` as
placeholder.

---------------------------------------------------------------------
| name                 | type            | comment                   |
| -------------------- | --------------- | ------------------------- |
| node_cpu_usage       | Float           | Average CPU usage of the  |
|                      |                 | node in percents. Calcu-  |
|                      |                 | lated as (100 -           |
|                      |                 | cpu_usage_idle)           |
| node_load            | Float           | Average values of load of|
|                      |                 | the last minute (load1)   |
|                      |                 | of the node               |
| proc_cpu_usage       | Float           | Average of the SUM of CPU|
|                      |                 | usages of all job's pro-  |
|                      |                 | cesses on the node        |
| proc_mem_rss_sum     | Integer, Bytes  | The sum of job's process  |
|                          |                     | RSS on the node            |
-----------------------------------------------------------------------
'''
```

In order to test the installation, execute the PDF report generator using the example configuration file of section 2.6.2 in combination with the example json file as follows.

Create a directory for the PDF reports:

```
mkdir PfiTPDFReport
```

Create a directory for temporary files:

```
mkdir tmppdf
```

Execute the PDF report generator:

```
python3 <pdf-report>/pdfgen/pfit_pdfreport.py <pdf-report>/
    testdata/test-ib_1352076.json ./PfiTPDFReport ./tmppdf
```

In the directory PfiTPDFReport, a file should be created with the name:

```
./PfiTPDFReport/pdf_report_1352076.pdf
```

This PDF file should contain six printable pages with information about the job with job identification number 1352076, 13 nodes and 40 requested cores. The example PDF report is located in the home directory of the report generator for comparison.

2.7 Statistics-Report

Until now, only the user can get an overview of its jobs and its metrics (with the text and PDF report). It would be advantageous, that administrators and Fachberater can also get an overview of the users jobstatistics in a certain time intervall. For this purpose, a statistics generator (and the appropriate documentation) was written in the course of the project extension. In this section, the installation, configuration and usage of the statisticsreportgeneraor will be described.

The statisticsgenerator was written in Python and is divided into two source files:

- `create_init_summary.py`,
- `create_report_from_summary.py`.

`create_init_summary.py` collects selected metrics of the job reports of a given time intervall and stores them in an intermediate format in the file `summary.txt`. This file can be run via cron job to collect the metrics since a given time until now. The workflow will be illustrated in the following:

- In the first step collect all metrics from the beginning of the collection process until now. The result of this initial step will be stored in the file `summary.txt`.
- In the following it is adviced to run `create_init_summary.py` every fixed time stamp and step (for example every day) using a cron job. `create_init_summary.py` collects all metrices from the test reports beginning after the job of the last entry of `summary.txt`. Every new job and its selected metrics will be attached to `create_init_summary.py` until the last job in the directory is reached. This procedure can be iterated until infinity. But it is also possible to run the script manually from the console.

These results in `summary.txt` can be evaluated with the python script `create_report_from_summary.py`. With this file an administrator can display a summary of all users over a specified time intervall with selected metrics and a ratio for all metrics in a table format. An example-output will be shown in ??, sorted by the total memory.

```
Startdate: Thursday, March 19, 2020 12:05:39
Enddate: Friday, May 08, 2020 13:05:39

Sorted by Nr jobs

|         |                    Walltime                 |           Jobs          |
|    User |     Requested |        Elapsed |  Ratio | #Jobs |    Avg runtime |
-------------------------------------------------------------------------------------  ...
| ******* |   00-12:00:00 |    00-12:00:08 |   1.00 |     1 |    00-12:00:08 |
| ******* |   00-01:00:00 |    00-00:01:39 |   0.03 |     1 |    00-00:01:39 |
| ******* |   01-16:00:00 |    00-10:40:18 |   0.27 |     1 |    00-10:40:18 |
| ******* |   00-00:30:00 |    00-00:11:27 |   0.38 |     1 |    00-00:11:27 |   ...

...

|         |          CPU                |          Memory          |          Swap            |
|    User |  Usage |  #Cores |  Ratio |   Hwm |    Total |  Ratio |  Used |    Total |  Ratio |
--------------------------------------------------------------------------------------------------  ...
| ******* |   0.00 |  256.00 |   0.00 |  1.20 |    62.80 |   0.02 |  0.00 |     1.90 |   0.00 |
| ******* |   0.90 |   16.00 |   0.06 |  1.20 |    62.80 |   0.02 |  0.00 |     1.90 |   0.00 |
| ******* |   0.00 |   16.00 |   0.00 |  1.10 |    62.80 |   0.02 |  0.00 |     1.90 |   0.00 |
| ******* |   0.00 |   16.00 |   0.00 |  1.30 |   251.80 |   0.01 |  0.00 |   128.00 |   0.00 |  ...

...

|         |          GPU            |
|    User |  Used |  Total |  Ratio |
---------------------------------------
| ******* |  0.00 |   0.00 |   0.00 |
| ******* |  0.00 |   0.00 |   0.00 |
| ******* |  0.00 |   0.00 |   0.00 |
| ******* |  0.00 |   0.00 |   0.00 |

...
```

These columns of the table have the following meaning:

- User: Name of the user of which
- Walltime:

  - Requested: The sum of all requested Walltimes over the time interval,
  - Elapsed: The sum of all elapsed walltimes over the time interval,
  - Ratio: The ratio of requested and elapsed walltime.
- Jobs:
  - #Jobs: The number of jobs which were started in the time interval,
  - Avg runtime: The sum of all elapsed walltimes over the time interval divided by the number of jobs.
- CPU:
  - Usage: Average CPU usage over the time interval,
  - #Cores: Average number of CPU cores over the time interval,
  - Ratio: The ratio of Usage and #Cores.
- Memory:
  - HWM: Average High water mark over the time interval,
  - Total: Average requested total main memory over the time interval,
  - Ratio: The ratio of HWM and Total.
- Swap:
  - Used: Average swap high water mark over the time interval,
  - Total: Average available swap memory over the time interval,
  - Ratio: The ratio of Used and Total swap space.
- GPU memory: Analoguous to main memory.

The prerequisites for using the statistics reporter are as follows:

- *python 3* (it was tested with version 3.6, but lower versions are possible runnable),
- the imported libraries given in the header of every file (if they are not existing, then install it for example with pip),
- the text report files must be sorted in the following way: n/nabcd.report. *n* is a folder with an increasing number (beginning from 0 or a value greater than 0) and every folder contains report files from n000 until n999, what means that in every folder theoretically should consists of 1000 report files.

Both scripts can be started in the following way:

- `python3 create_init_summary.py /path/to/config/files/`
- `python3 create_report_from_summary.py m 3 /path/to/summary/file/ sortby`

`create_init_summary.py` only needs the path to the config file `stats.conf`, the file, which holds the last superfolder n (`n_last_folder.txt`) and the summary file. The report generator file `create_report_from_summary.py TimeIntervallMeasurement NrTimeIntervalls /path/to/summary/file/ sortby` is the file, which creates the summary of a specified time interval for the administrator. To start the script we need 4 parameters:

- the time interval measure (possible values are y = years, m = months, d = days and h = hours),
- the number of for example months or days from now back into the past,
- the path to the config file `stats.conf`, the file, which holds the last superfolder n (`n_last_folder.txt`) and the summary file
- and the sorting parameter which controls, which columns should sorted. They are as follows:
  - user = sort by user,
  - requested | elapsed | ratiowalltime = sort by requested or elapsed walltime or by the walltime ratio,
  - njobs | aveelapsed = sort by the number of jobs or the average elapsed walltime,
  - cpuusage | cpucores | ratiocpu = sort by the average CPU usage, the average available total cores or the ratio of CPU usage and CPU,
  - memhwm | memtotal | ratiomem = sort by the average HWM, the average available total memory or by the ratio of HWM and total main memory,
  - swaphwm | swaptotal| ratioswap = Analoguously to main memory.
  - gpuhwm | gputotal | ratiogpu = Analogously to main memory.

Chapter 3

Special topics of InfluxDB administration

This chapter will take a closer look on special features of the timeseries database *InfluxDB*, especially the communication with the database, handling security topics as well as retention policy and continuous quereing. These topics will be explained in the next sections.

3.1 Communicating with InfluxDB

There are two main methods to communicate with the database *InfluxDB*:

- via the command line tool *influx*,
- via the HTTP API of *InfluxDB*.

Here we will only focus on the Command Line Interface (CLI) *influx*. The HTTP API will be used in the *PfiTCollect* metric collector via the `curl` command (see *Developer Guide*). For further information of the HTTP API, please have a look at `https://docs.influxdata.com/influxdb/v1.7/guides/writing_data/` and `https://docs.influxdata.com/influxdb/v1.7/guides/querying_data/`.

The CLI can be invoked by executing $INFLUXDB/bin/influx [OPTIONS], where the most often used options are (see also the manpage of `influx`):

- precision <rfc3339 | h | m | s | ms | u | ns>,
- username <username>,
- password <password>,
- unsafeSsl,
- ssl,
- host <host>,
- port <port>,

- database <database>.

precision defines the way of displaying the timestamps. *influx -precision rfc3339* displays the time in UNIX format, for example 2017-12-12T23:23:23. *influx -precision h* displays the time in hours, *m* in minutes, *s* in seconds, *u* in mikroseconds and *ns* in nanoseconds (which is the default setting in *InfluxDB*) since the 1st of january 1970 (since the epoch). The parameter *username* (possibly in connection with the *password* flag) allows the user to log in into the *InfluxDB* with user credentials. The flag *ssl* allows to use a SSL/TLS connection for HTTPS requests, while *unsafeSsl* (in combination with the *ssl* flag) allows the usage of unsafe connections (for example using a self signed certificate). To select a special database, host or port from the CLI, the use of database <database>, host <host> or port <port> is possible.

3.2 Security topics

Some of the most important goals to ensure the IT security are:

- confidentiality,
- integrity,
- authentification,
- authorization.

In the following subsections these goals will be presented for *Telegraf* and *InfluxDB*.

3.2.1 Authentication and Authorization

Authentication and authorization are crucial in *InfluxDB* to preserve the confidentiality of the data. Basically it is possible that all data stays public, but the following disadvantages should give attention in our application case:

- Every user of the database has read access to all performance data of all other users (especially access to the data of user behaviour, job and program names, etc.) which increases strongly the risk of misuse of the data.
- Every user of the database has write access to all performance data of all other users, which can lead to accidentaly deleting or changing data or to miscredit another user.
- Attacks to the database, misuse of the database.

Because of these threats it is meaningful to secure the *InfluxDB* database with authentication and authorization rights for the users to restrict the access to their own or selected data.

To set up the authentication process in *InfluxDB*, the first step is to create specified user accounts in *InfluxDB*. This could be done via the usage of the Command Line Interface (CLI) or via `curl`. By default authentication is disabled and all credentials are ignored. The consequence is, that every user has read and write access to all data. To give every user appropriate access to the data, users have to be created internally. In general at least one administrator user with all rights to all data (read and write) should be created and one or more ordinary users with restricted rights (for example only read access for the aggregator/report generator and only write access for the collector). If authentication is enabled, a user (especially the administrator user) has to be created and further users with limited access to the database. In the following the essential steps to setup authentication will be demonstrated.

3.2.2 Setting up authentication

Creation of administrator and ordinary users for authentication

The creation of a database administrator will be exemplified in the following line:

```
CREATE USER admin WITH PASSWORD 'zbycxd234Imgh' WITH ALL
   PRIVILEGES
```

This command line creates a user with the name *admin* and the password *zbycxd234Imgh*. All access rights are assigned to the user *admin* which means, that this user can read and write from/to all databases. A database can have more than one administrator which can be created in the same way for *InfluxDB*.

Non admin users can be created in an analogous way to that of an admin user. We will demonstrate this for the users *writer* and *reader*.

```
CREATE USER writer WITH PASSWORD '123azbyjhksg'
CREATE USER reader WITH PASSWORD 'azbyjhksg123'
```

In this case the user has not yet any read and write access. If there is at least one admin user *InfluxDB* will enforce authentication.

Enable authentication in the InfluxDB config file

By default the authentication is disabled in the configuration file. To enable it, in the first step please change the marked row (!!!!) in the `[http]` paragraph of the configuration file of *InfluxDB* from `false` to `true` (see https://docs.influxdata.com/influxdb/v1.7/administration/authentication_and_authorization/#authentication).

```
[http]
  enabled = true
  bind-address = ":8086"
  auth-enabled = true # !!!!!
  log-enabled = true
  write-tracing = false
  pprof-enabled = false
  pprof-auth-enabled = true
  debug-pprof-enabled = false
  ping-auth-enabled = true
  https-enabled = false
  https-certificate = "/etc/ssl/influxdb.pem"
```

Restart the InfluxDB process to establish authentication process

To finally establish authentication, the *InfluxDB* process has to be restarted (see below in case *systemd* and *InfluxDB* are running as system services). After that, authentication is enabled.

```
sudo systemctl restart influxdb
```

If `sudo` isn't enabled, a root login should be done.

3.2.3 Authenticate Requests

There are two ways to authenticate with *InfluxDB*:

1. Authentication with the CLI and
2. Authentication with the HTTP API.

Both will be described in the following subsections. We will begin with the description of the authentication via CLI.

Authentication with the command line interface (CLI)

In the case of authentication with the CLI, there are three possible ways to authenticate:

1. With the use of the *InfluxDB* environment variables INFLUX_USERNAME and INFLUX_PASSWORD.
2. Setting the flags *-username* and *-password* when starting the CLI.
3. After starting the CLI with *auth <username> <password>*.

Now we will give an example for each setting.

1. Authentication via environment variables:

```
> export INFLUX_USERNAME extractor
> export INFLUX_PASSWORD azbyjhksg123

> influx  # Start influx
  Connected to http://localhost:8086 version 1.4.x
  InfluxDB shell 1.4.x
```

2. Authentication via setting flags:

```
> influx -username reader -password azbyjhksg123

  Connected to http://localhost:8086 version 1.4.x
  InfluxDB shell 1.4.x
```

3. Authentication after starting the CLI:

```
> influx
  Connected to http://localhost:8086 version 1.4.x
  InfluxDB shell 1.4.x
> auth
  username: reader
  password: azbyjhksg123
```

Authentication with the InfluxDB API

The second possibility to authenticate to *InfluxDB* is the authentication via the HTTP API. In the following two examples, the authentication via basic authentication and via providing query parameters in the URL or in the request body will be presented (both using cURL):

- Authentication via basic authentication (described in RFC 2617, Section 2):

```
curl -G http://localhost:8086/query -u reader:azbyjhksg123
     --data-urlencode "q=SHOW DATABASES"
```

- Authentication by providing query parameters in the URL or request body
  In the following two lines it will be demonstrated how the user can authenticate by providing query parameters in the URL (first line; $query?u = reader\&p = azbyjhksg123$) or by providing them in the request body (second curl command; –data-urlencode "u=reader"–data-urlencode "p=azbyjhksg123")

```
curl -G "http://localhost:8086/query?u=reader&p=azbyjhksg123"
     --data-urlencode "q=SHOW DATABASES"

curl -G http://localhost:8086/query
     --data-urlencode "u=reader"
     --data-urlencode "p=azbyjhksg123"
     --data-urlencode "q=SHOW DATABASES"
```

Notice that the hostname may be different to "localhost".

Authorization

After the creation of the above two non admin users, no privileges were granted to them, that means no read and write permissions are given to these users for any database. Read and write access has to be set separately for every database. For example, write access can be granted to the user *writer* on the database *myTelegraf* with the following command line:

```
GRANT WRITE ON myTelegraf TO "writer"
```

This command line assigns the existing user *writer* write access for the given database *myTelegraf* but the user *writer* will not able to read the data from it. To allow an existing user *reader* to read data out of the database *myTelegraf*, the following command line is required:

```
GRANT READ ON myTelegraf TO "reader"
```

In this case the user *reader* has read permission for the database, but he has no write permission for *myTelegraf*.

If the user *reader* should get administrator privileges (that means, he has write and read access and some other privileges for *myTelegraf*), then the following line will help:

```
GRANT ALL PRIVILEGES TO "reader"
```

If the administrator of the database wants to revoke user privileges, then substitute GRANT with REVOKE and TO with FROM, for example:

```
REVOKE READ ON myTelegraf FROM "reader"
```

In the course of administration *InfluxDB*, it is possible that the admin wants to see all users and their admin privileges. This can be done via

```
SHOW users
```

which will provide in our case the following output:

```
user     admin
admin      true
writer     false
reader     false
```

In our installation/configuration we used this distinction between posting users (here the user *writer* to send metrices from *PfiTCollect* or *Telegraf* to the database) and the reading users (here the user *reader*), which reads the data from *InfluxDB* to create the Text or the PDF report. Furthermore the *writer* and*reader* user has no admin privileges.

Often it is necessary to change the password of the user or to delete a user from the *InfluxDB* (for example if the user is not a member of the organization any more). Setting the password for a user is shown in the following example. Only the user himself and the administrator can set the users password. It is recommended to use single quoting for the passwords and writing them in one line (a wordwrap between these single quotes could lead to a problem).

```
SET PASSWORD FOR "reader"='azbyjhksg123'
```

Dropping a user from the database can be done by using the keyword *DROP USER*. In the following example the user *reader* will be deleted using this keyword:

```
DROP USER "reader"
```

Managing user rights and permissions in PfiTCollect and Telegraf

To authenticate *Telegraf* requests to an *InfluxDB* instance with authentication enabled requires two steps. First, in *Telegraf's* configuration file, the username and password settings have to be uncommented and the appropriate username and password has to be entered: (The text was taken from and adapted to our needs.)

```
###############################################################
#                          OUTPUT PLUGINS                      #
###############################################################

OUTPUT PLUGINS

[...]

## Write timeout (for the InfluxDB client), formatted as a
## string.
## If not provided, will default to 5s. 0s means no timeout
## (not recommended).
timeout = 5s
username = reader
password = azbyjhksg123
(...)
```

After this first step restarting the database will activate the authentification process for *Telegraf*.

HTTPS Setup

To authenticate servers or clients and to secure the network traffic by cryptograhic security, the unencoded HTTP protocol was extended by an additional SSL/TLS layer (SSL = Secure Socket Layers, TLS = Transport Socket Layer), in which SSL is legacy. They are located on top of the Transport Layer in the TCP/IP protocol stack and their task is to secure the transport over the network through encoding and decoding the data, to ensure that the data will not be changed while tranporting over the network (managed by cryptograhical hashes (integrity of the document)) and to authenticate server/clients or users. Since TLS (especially the newest version TLS 1.3 (effective 04/2020)) is the up-to-date version, but SSL is widely in use (hazardous because of security problems in SSL!!), we will further speak of SSL/TLS. It is strongly recommended by *InfluxData* to secure the network traffic by using HTTPS.

To use HTTPS, certificates have to be created and in the following we will describe how to create a self-signed certificate (which often should be sufficient in a cluster and for our purposes) and to set up the HTTPS connection with a CA- or self signed certificate. *InfluxDB* supports "certificates composed of a private key file (.key) and a signed certificate file (.crt) file pair, as well as certificates that combine the private key file and the signed certificate file into a single bundled file (.pem) "(please see https://docs.influxdata.com/influxdb/v1.7/administration/https_setup/).

The following requirements are needed:

- Running an existing *InfluxDB* instance,
- a SSL/TLS certificate.

InfluxDB supports three kinds of certifications (the following three items are nearly literally taken from https://docs.influxdata.com/influxdb/v1.7/administration/https_setup/):

- Single domain certificates signed by a Certificate Authority (CA)
  These certificates provide cryptographic security to HTTPS requests and allow clients to verify the identity of the *InfluxDB* server. These certificates are signed and issued by a trusted, third-party Certificate Authority (CA). With this certificate option, every *InfluxDB* instance requires a unique single domain certificate.
- Wildcard certificates signed by a Certificate Authority
  Wildcard certificates provide cryptographic security to HTTPS requests and allow clients to verify the identity of the *InfluxDB* server. Wildcard certificates can be used across multiple *InfluxDB* instances on different servers.
- Self-signed certificates
  Self-signed certificates are not signed by a trusted, third party CA. Generate a self-signed certificate on your own machine. Unlike CA-signed certificates, self-signed certificates only provide cryptographic security to HTTPS requests. They do not allow clients to verify the identity of the *InfluxDB* server. We recommend using a self signed certificate if you are unable to obtain a CA signed certificate. With this certificate option, every *InfluxDB* instance requires a unique self signed certificate.

InfluxDB accepts certificates with private key files (*.key) and public key file / signed certificate file pairs (*.crt), as well as the combined version in the *.pem file.

HTTPS with a server certificate signed by a certificate authority

The following steps, which describe the process of generating certificates and enabling HTTPS, are strongly based on https://docs.influxdata.com/influxdb/v1.7/administration/https_setup/ „HTTPS with a CA signed certificate".

- Installation of the SSL/TLS certificate
  In the first step put the pair of *.key/*.crt files for example into the /etc/ssl directory (recommended for *InfluxDB*). The same is valid for the bundled *.pem file case.

- Change the permission(s) of the certificate file(s)
  Since root needs to have read and write access on these files, in the second step change the rights in the following way (example taken from https://docs.influxdata.com/influxdb/v1.7/administration/https_setup/):

```
> sudo chown root:root /etc/ssl/influxdb_signed.crt
> sudo chmod 644 /etc/ssl/influxdb_signed.crt
> sudo chmod 600 /etc/ssl/influxdb_private.key
```

  In the first step the ownership of the *influxdb_signed.crt* will be changed to root (for the user as well as for the group). In step two and three the access rights to the signed certificate and the private key will be set appropriately.

- Enable HTTPS in the *InfluxDB* configuration file
  Since HTTPS is disabled by default, the administrator has to activate it in the *HTTP* section of the configuration file in the following way (see section *Configuration of InfluxDB*; it is also possible to do that with the use of environment variables and the file names can be chosen arbitrarily):

  – Set *https-enabled = true.*
  – Set *https-certificate = /etc/ssl/influxdb_signed.crt*
    (or to /etc/ssl/influxdb_bundled_signed.pem) **OR**
  – set *https-private-key = /etc/ssl/influxdb_private.key*
    (or to /etc/ssl/influxdb_bundled_private.pem)

  (The *influxdb_signed* prefix of the above filenames are just example names and can be chosen arbitrarily.)

- Restart of the *InfluxDB* service
  If *InfluxDB* is running as a service under *Systemd*, the service has to be restarted, to account for the above changes:

```
sudo systemctl restart influxdb
```

- Verify HTTPS setup
  To check if the HTTPS setup works correctly, start the CLI tool of *InfluxDB* and then type in the following line:

  ```
  influx -ssl -host <domain_name>
  ```

  in which the domain often is a Fully Qualified Hostname.
  If the connection was successfully established, the following response will be created (in which the *InfluxDB* shell version can differ):

  ```
  Connected to https://localhost:8086 version 1.x.x
  InfluxDB shell version: 1.3.1
  >
  ```

  The important entry is the `https` in the first line. If this entry is just `http`, then something went wrong.

HTTPS with a self-signed server certificate

In contrast to certificates, which were signed by a certificate authority, a self signed certificate in general is not as secure as the ones signed by a certificate authority but in our cases they are sufficient. This paragraph is closely related to https://docs.influxdata.com/influxdb/v1.7/administration/https_setup/, in which the examples where slightly modified.

- Create a self signed server certificate
  This can be done for example like in the following line:

  ```
  sudo openssl req -x509 -nodes -newkey rsa:4096
    -keyout /etc/ssl/influxdb-selfsigned.key
    -out /etc/ssl/influxdb-selfsigned.crt -days 30
  ```

  First of all you need to be root to create a self signed certificate. In the following the *openssl* command was used to create the certificate, which will expire after 30 days (-days 30). This certificate is a X509 certificate with RSA encryption (4096 Bits) and both key pair files are stored into the directory /etc/ssl.

  After execution the above command, *openssl* asks for more information, for example country name, organizational name. You are free to fill in the desired things or not (it's recommended by *InfluxData* to fill in the information) but in both cases a valid certificate will be created. With `chmod` *InfluxDB* is allowed to read and write on the certificate.

- Enable HTTPS
  The following three steps are similar to the steps in the case of a CA certificate. In this step HTTPS have to be enabled. We will demonstrate that in the case of the configuration file but it is also possible to use environment variables to do that.
  - Set `https-enabled = true`
  - set `http-certificate = /etc/ssl/influxdb_selfsigned.crt` OR
  - set `http-private-key = /etc/ssl/influxdb_selfsigned.key`.
- Restart the *InfluxDB* service
  As in the above case, the *InfluxDB* service has to be restarted so that the changes can be regarded. If we are using *systemd* and *InfluxDB* is running as a service, the command looks like as follows:

```
sudo systemctl restart influxdb
```

  If `sudo` isn't enabled, try `su root` and then type (after logged in) `systemctl restart influxdb`.
- Verify the HTTPS setup
  In the last step of the HTTPS setup, the verification of the successful setup should be done. This can be verifyed with the CLI tool again, via typing

```
influx -ssl -unsafeSsl -host <domain_name>
```

  The setup of HTTPS was successful, if you got the same output, as in the above case.

Further security aspects

To protect the host, *InfluxData* recommended two more things (please see https://docs.influxdata.com/influxdb/v1.7/administration/security/):

- Managing ports:
  In the case, that only *InfluxDB* is running as a service on your server, then *InfluxDB* recommends, that all ports of the host, except port 8086, should be closed. Another possibility is, that a proxy can be used to port 8086. Furthermore port 8088 is used by the remote backup and restore process. It is recommended to close this port, too, and if performing a remote backup, giving specific permission only to the remote machine.
- Further recommendations
  InfluxData also recommends to implement on-disk encryption since InfluxDB does not offer built-in support to encrypt the data.

3.3 Storage and data administration

Since *PfiTCollect* and *Telegraf* can create many datapoints and a huge data volume to store in *InfluxDB*, it is important to implement a mechanism in *InfluxDB* to dump and/or downsize the data. This can be done either by dropping data after a specified time interval (*Retention Policy*) or by summarizing or aggregating the data over a specified time intervall and dropping the high precision raw data of that time interval (*Continuous Quering*), while storing the low precision data for a long(er) time. All two mechanisms serve to reduce the data volume.

Furthermore, a backing up and restoring of the database data is important, for example if a disk failure occurs and data will be lost if no copy of the data is available.

The *Retention Policy* and *Continuous Quering* will be presented in the next subsection, while the presentation of the *back up* and *restoring* features will follow the *retention policy* subsubsection.

3.3.1 Retention Policy and Continuous Qeuering

This documentation is a summary of the description of *Retention Policy* and *Continuous Quering* of the *InfluxDB* database (please see `https://docs.influxdata.com/influxdb/v1.7/guides/downsampling_and_retention/`). In this section both will be presented by example, in which we will choose the examples close to the *InfluxDB* description, but adapted to the needs of our project.

- Retention Policy
  A *retention policy* defines two things:

  - The duration of the data storage (DURATION). The default retention policy *autogen* has no time limit in storing the data. In the case of a retention policy with limited storage duration, the data points, which are older than the *retention policies* DURATION (that means their local time stamp) will be deleted.
  - The number of copies of the stored data in *InfluxDB* (REPLICATION). In the default retention policy *autogen* the replication factor is set to one.

  It is important to know, that a database can have more than one *Retention Policy* and that the data is bounded to the Retention Policy.

- Continuous Quering
  In contrast to a *Retention Policy* a *Continuous Query* is a *InfluxDB* query, which runs automatically and periodically to reduce and downsample the amount of data by summarizing and aggregating the data of the database.

In the following we will present some examples to show how to manage *Retention Policies* and *Continuous Quering*.

Retention Policy

In this paragraph we will present examples which will show how to create, alter or delete *Retention Policies*.

- Create a new default *Retention Policy*
  To create a default *Retention Policy* which is different to the default *Retention Policy autogen*, the administrator (no one else) have to explicitly create a new one. As an example we want to create a new default retention policy, that stores the data for seven days and after this period, the data will be deleted from the database.

  ```
  CREATE RETENTION POLICY "SevenDays" ON "myTelegraf" \
  DURATION 7d REPLICATION 1 DEFAULT
  ```

  In this example we have created the *Retention Policy* "SevenDays" in the database *myTelegraf*, in which the storage duration of the data is 7 days. Furthermore there are no more copies of independent data points in the database (just one; REPLICATION 1). The keyword DEFAULT defines this *Retention Policy* as the default one, which replaces the *autogen Retention Policy* as the default *Retention Policy*. Otherwise this new *Retention Policy* is not the default one. That means as a consequence, all reads and writes to *myTelegraf* will be done to the *autogen Retention Policy* and not to this one.

- Create a new non default retention policy

  ```
  CREATE RETENTION POLICY "TwentysixWeeks" ON "myTelegraf" \
  DURATION 26w REPLICATION 1
  ```

  Now we have created a new *retention policy* "TwentysixWeeks" for *myTelegraf* which has a storage duration of 26 weeks. If we want to have hours or days, then the suffix of the number has to be *h* (hours) or *d* (days), in which the minimum duration is 1 hour and the maximum limit infinity (INF). Because no DEFAULT keyword was appended this *Retention Policy* is not the default one and if it should be used it has to explicitly written down.

- Modifying a *retention policy*
  To modify features of a *retention policy* (for example duration or replication factor) the `ALTER RETENTION POLICY` command should be used (see https://docs.influxdata.com/influxdb/v1.7/query_language/database_management/#create-retention-policies-with-create-retention-policy). In the following example we will modify the DEFAULT *Retention Policy* "SevenDays" of the first example of this paragraph by changing DURATION from seven days to two weeks and the replication factor to 2.

```
ALTER RETENTION POLICY "SevenDays" ON "myTelegraf" \
DURATION 2w REPLICATION 2 DEFAULT
```

in which DURATION, REPLICATION and DEFAULT are optional keywords but at least one of them have to be used.

- Deleting a *retention policy*

```
DROP RETENTION POLICY "SevenDays" ON "myTelegraf"
```

If the *Retention Policy* "SevenDays" should be deleted from the database "myTelegraf", then the above line will manage this. If the deletion of a *Retention Policy* was successful then *InfluxDB* returns an empty response otherwise an error message (see https://docs.influxdata.com/influxdb/v1.7/query_language/database_management/#create-retention-policies-with-create-retention-policy).

Continuous Quering

Another feature to reduce the amount of data, is the downsampling of the data (for example by aggregation). In the *InfluxDB* terminology this is called *Continuous Quering*. Basically this feature checks every time intervall the specified field (key) values and after expiration of a specified time interval those values will be stored in another measurement of a specified retention policy by applying a function, which creates one value over the specified time intervall. The more detailed description of this aspect of *InfluxDB* is described in https://docs.influxdata.com/influxdb/v1.7/guides/downsampling_and_retention/ and https://docs.influxdata.com/influxdb/v1.7/query_language/continuous_queries/.

In the following we will only cover the basic features of *InfluxDB*s *Continuous Quering* and we will present examples on how to create a *Continuous Query* and how to delete it. It is important to have in mind, that *Continuous Quering* uses the local server's timestamp (which should be synchronized over all nodes of the cluster) and needs the GROUP BY time interval in the appropriate queries.

Create a Continuous Query

The general formulation of the creation of the *Continuous query* is (please see https://docs.influxdata.com/influxdb/v1.7/query_language/continuous_queries/:

```
CREATE CONTINUOUS QUERY <cq_name> ON <database_name>
BEGIN
  <cq_query>
```

```
END
```

In this general description *cq_name* is the newly created *Continuous Query* belonging to the database *database_name*. The *cq_query* must be a SELECT query, in which a function, an INTO clause and a group_by_time() must be formulated, to be a valid *Continuous Query*:

```
SELECT <function[s]> INTO <destination_measurement>
                     FROM <measurement> [WHERE <stuff>]
                     GROUP BY time(<interval>)[,<tag_key[s]>]
```

As an example we will present the creation of the *Continuous Query* "one_hour", in which the arithmetic mean of free and used memory of the measurement *mem* (database *myTelegraf*) will be taken over the time intervall one hour and stored into the new measurement "mem_free_used" of the retention policy "one_month" (see https://docs.influxdata.com/influxdb/v1.7/guides/downsampling_and_retention/.

```
CREATE CONTINUOUS QUERY "one_hour" ON "myTelegraf"
BEGIN
  SELECT mean("free") AS "mean_free", mean("used") AS "mean_used"
  INTO "one_month"."mem_free_used"
  FROM "mem"
  GROUP BY time(1h)
END
```

This example is similar to the one from https://docs.influxdata.com/influxdb/v1.7/guides/downsampling_and_retention/ but it is taylored towards our application scenario, the collection of system metrics data of *Telegraf*. Here, the *Continuous Query* "one_hour" will be created, which is assigned to the database "myTelegraf". The mean over the last one hour of the fields "free" and "used" of the measurement *mem* will be calculated and stored under the new field names "mean_free" and "mean_used" into the measurement "mem_free_used" of the retention policy "one_month" (which have to be created, if it does not exists).

To delete the above defined *continuous query*, use the following line:

```
DROP CONTINUOUS QUERY one_hour
```

And finally, to show all *continuous queries* this line will help:

```
SHOW CONTINUOUS QUERIES
```

3.3.2 Backup and Restore

Backing up a database (metadata as well as the data) and restoring it, are essential tasks in working with databases. Since *InfluxDB* version 1.5 the following features are provided (see https://docs.influxdata.com/influxdb/v1.7/administration/backup_and_restore/:

- Backing up and restoring of a single or multiple online/live database(s) (including optional filtering).
- Data imports from *InfluxDB* Enterprise clusters
- The format of the backup files of *InfluxDB* (legacy format) is now compatible to the backup format of the *InfluxDB Enterprise* database (uses i.a. less disk space).

In the older versions of *InfluxDB* (until version 1.4) the restore function could only be applicated to an offline instance of *InfluxDB*.

Online backup and restore (for InfluxDB OSS)

First we will have a look at the online backup and restore of the *InfluxDB* OSS data and metadata. The online backing up and restoring capabilities are renewed features of *InfluxDB* since version 1.5. To use them, some prerequisites have to be fulfilled which will be explained in the following (see this and the following at https://docs.influxdata.com/influxdb/v1.7/administration/backup_and_restore/).

- Both processes will be executed over a TCP connection to the *InfluxDB*.
- Uncomment on the remote node the following line in the root configuration file of *InfluxDB* (influxdb.conf): `bind-address = 127.0.0.1:8088`.
- Set `bind-address = <remote-node-IP>:8088` (port 8088 is the port specification for remote database access to *InfluxDB*).
- Every time a backup or restore command will be run, add `-host <remote-node-IP>:8088` to the command.

In the following we will describe the general form of the backup and restore commands and we will begin with the *backup* command (all descriptions and examples are generally taken from https://docs.influxdata.com/influxdb/v1.7/administration/backup_and_restore/ but with modifications (especially the examples)):

```
influxd backup
    [ -database <db_name> ] [ -portable ] [ -host <host:port> ]
    [ -retention <rp_name> ] |
    [ -shard <shard_ID> -retention <rp_name> ]
    [ -start <timestamp> [ -end <timestamp> ]|-since <timestamp> ]
    <path-to-backup>
```

The parameters have the following meaning, in which we just present for our needs the most important ones:

- [$-database < db\_name >$]: Specifies the database, which should be backed up (no given database leads to backing up all databases).
- [$-portable$]: Generates backup files in the newer *InfluxDB* Enterprise-compatible format (recommended for *InfluxDB OSS*).
- [$-host < host : port >$]: Specifies the host and the portnumber of the *InfluxDB OSS* instance. An example is given below and the default vaulue is 127.0.0.1:8088.
- [$-retention < rp\_name >$]: This argument specifies the Retention Policy to use for back up. If this parameter is set, then additionally the `-database` parameter has to be set. If no retention policy is set, all retention policies will be taken.
- [$-start < timestamp >$]: Start time of the backup intervall (RFC3339 format), which includes all time step points, which should be backed up. If no `-end` parameter is given, then all timesteps since the start intervall will be backed up until now.
- [$-end < timestamp >$]: End time of back up intervall (RFC3339 format); analoguous to `-start`. The combination of the `start` and `end` timestamp allows an incremental backup of a database.

To complete this explanation, a few examples will be given, to exemplify the parameters.

- `influxd backup -portable /tmp/PfiT-Backup` (back up all data in all databases to the specified path)
- `influxd backup -portable -start 2015-12-24T08:12:23Z /tmp/PfiT-Backup` (backup all databases since the given date and time until now)
- `influxd backup -portable -database myTelegraf /tmp/PfiT-Backup` (backup only the myTelegraf database)
- `influxd backup -portable -database myTelegraf -start 2017-04-28T06:49:00Z -end 2017-04-28T06:50:00Z /tmp/PfiT-Backup` (backup a database for the specified time interval)
- `influxd backup -portable -database myTelegraf -host 192.168.234.23:8088 /tmp/ PfiT-Backup` (backup *myTelegraf* using a romote connection to the by an IP given host).

The restore process can be done with the use of the following general command line:

```
influxd restore [ -db <db_name> ]
    -portable | -online [ -host <host:port> ]
    [ -newdb <newdb_name> ]
    [ -rp <rp_name> ] [ -newrp <newrp_name> ]
    [ -shard <shard_ID> ] <path-to-backup-files>
```

The following parameters of the restore commands are

- $[-portable]$: Generates backup files in the newer *InfluxDB* Enterprise-compatible format (recommended for *InfluxDB OSS*).
- -online: Only use this legacy backup format if the newer *InfluxDB* Enterprise-compatible format cannot be used.
- $[-host < host : port >]$: See backing up.
- $[-db < db\_name >]$: See parameter [-database <db_name>] in backing up.
- $[-newdb < newdb\_name >]$: A new database, into which the archive will be imported on the target system. If it is not set, than `-db` flag will be used.
- $[-rp < rp\_name >]$: See parameter [-retention <rp_name>] in backing up.
- $[-newrp < newrp\_name >]$: The retention policy, which is created on the target system (`-rp` must be set). If this one is not set, then `-rp` will be used.

Chapter 4

Miscellaneous

In this closing chapter all metrices collected by *Telegraf* and *PfiTCollect* with their datatypes will be presented. Furthermore an example installation of all tools is included.

4.1 Details

4.1.1 Collected metrics

Table 4.1: Collected metrics of *Telegraf* and *PfiTCollect*

Measurement	Field Key	Datatype
CPU	usage_user	float (percent)
	usage_system	float (percent)
	usage_idle	float (percent)
	usage_iowait	float (percent)
	usage_active	float (percent)
	usage_irq	float (percent)
	usage_softirq	float (percent)
	usage_guest	float (percent)
	usage_guest_nice	float (percent)
	usage_stael	float (percent)
	cpu_usage	float (percent)
MEM	active	integer
	inactive	integer
	available	integer
	buffered	integer
	cached	integer
	free	integer
	total	integer
	slab	integer
	swapcached	integer
	used	integer
	used_percent	float (percent)
	writeback_tmp	integer
	kernelstack	integer
	pagetables	integer
swap	free	integer
	in	integer
	out	integer
	total	integer
	used	integer
	used_percent	float

Table 4.2: Collected metrics of Telegraf and PfiTCollect

Measurement	Field Key	Datatype
system	load1	float
	load5	float
	load15	float
	n_cpus	integer
	n_users	integer
	uptime	integer
processes	total	integer
BeeGFS	beegfs_ops_read	integer
	beegfs_ops_write	integer
	beegfs_read	float
	beegfs_write	float
diskio	bandwidth_read	float
	bandwidth_write	float
	iops_in_progress	integer
	read_bytes	integer
	read_time	integer
	reads	integer
	weighted_io_time	integer
	write_bytes	integer
	write_time	integer
	writes	integer
nfs	nfs_read	integer
	nfs_write	integer
net	portrcvdata	integer
	portrcvpkts	integer
	portxmitdata	integer
	portxmitpkts	integer

4.1.2 Influence on performance

4.1.3 Data protection/Data security

- Which metrics will be collected?
  Please see subsection 4.1.1.

- Why were the metrics collected?
  The goal of collecting the metric values is to generate different performance reports at the end of every job to inform the user, which performance bottlenecks in the job were identified and how to eliminate them (or how to lower their performance impact). Furthermore this data will be evaluated by another report for system administrators and Fachberater to identify users with a conspicuous runtime behaviour and to inform them. No other persons are allowed to have access to the performance metrics, i.e. the administrators and the process which will read the data to create the reports and the process, which collects and stores the metric data. The reports will be delivered only to the owner of the job.

- How long the data will be stored?
  This depends on the *Retention Policy* of *InfluxDB* where the data is stored. If the default *Retention Policy autogen* will be used, the data will be stored infinitely or until the database can store no more data (because of the disk is full). Otherwise the metric data will be stored for the specified time interval of the *Retention Policy*. If additionally a *Continuous Query* is defined aggregated data can be stored for longer periods, but in general the origin of this data cannot be identified.

4.1.4 Limitations

4.1.5 Misc

4.2 Example Installations and Configurations of Telegraf and InfluxDB

4.2.1 Installation from Scratch in the cluster (Universität Hamburg)

In this subsection we will present the installation and configuration of *Go*, *Telegraf* and *InfluxDB* from scratch, all of which are running on the *Hummel cluster* of the *Regionales Rechenzentrum der Universität Hamburg (RRZ)*.

The *Hummel cluster* is a Tier 3 cluster consisting of 396 nodes, in which each of the 54 nodes possess two additional GPUs. Every node is a two socket node with eight cores per socket (Intel(R) Xeon(R) CPU E5-2630 v3). The nodes are equipped with 64 GiB and 256 GiB (big nodes) main memory. *QDR Infiniband* is used to connect the compute nodes and the parallel filesystem is *BeeGFS*.

The *Telegraf* and *PfiTCollect* instances are running on every node (one per node) and the *InfluxDB* database is hosted on a single server with 1 TiB storage space for the *InfluxDB* data and metadata. Concerning security aspects, the installed system is based on HTTPS (with a self signed certificate) and uses user credentials for the specified users *admin*, *reader* and *writer* to have read or write acccess to the data of the database. Since the data collected by *Telegraf* and *PfiTCollect* will strongly extend the data volume of *InfluxDB*, a *Retention Policy* is activated, which deletes selected data after a specified period (in our case after seven days). In combination with *Continous Quering*, which is a method to reduce data by aggregating it, this can reduce the data volume in a structured manner (no *Continuous quering* is activated). Backup and restore will take place ...

The basic idea behind this kind of installation is, that the system build is fully under control of the administrator. That means that all release source files of the *Go* and *Influx* project will be downloaded first to the predefined download/repository server. After this step these files will be downloaded into a predefined workspace, and after that, all dependencies that the *Go* Dependency Manager recognizes will be resolved and downloaded into a special subfolder. This ensures that the administrator has a complete overview over all files of the project and the dependencies of *Go* which have to be downloaded.

In the next step, the build process of *Go* will start. Although *Go* (and the *Go* tools) are available as binaries, we don't use these binaries. *Go* is open source and we want to build *Go* from scratch. In the first step we do a bootstrap step, in which *Go* 1.4 was build by the GCC compiler 4.9 (or by any C compiler). In the second step *Go* 1.9 (the newest version is 1.13.4) was build by using the *Go* compiler in the version 1.4 (which was formerly build in the bootstrapping process). After this step, we used *Go* 1.9 to build the *gdm* (*Go* Dependency Manager) which is used to download the release files and to resolve dependencies. In the last two steps, *InfluxDB* and *Telegraf* will be installed and configured, after downloading the release files and resolving the dependencies (downloaded by gdm, compiled by *Go*). *PfiTCollect* was built with the GCC.

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

Installing *Go*

Go can be installed in a directory, which can be freely choosen (it's not necessary to build *Go* in the *Telegraf* or *InfluxDB* workspace).

There are eight basic steps in all build scripts to carry out. (This is valid for downloading and bootstrapping *Go* 1.4, building *Go* 1.9 from 1.4, building *Telegraf* and *InfluxDB*.):

1. Declaration and initialization of variables;
2. loading modules, which are necessary for the build process
3. downloading repository from download server,
4. patching repository if necessary,
5. building the binaries,
6. removing traces and files,
7. building the modules(*Go* 1.9, *Telegraf*, *InfluxDB*),
8. configuring (if necessary).

In the following, we will present these steps by the example of building *Go* by bootstrapping and building *Go* from 1.4 to 1.9. We will begin with building *Go* by bootstrapping, which will be done with the help of an arbitray C compiler (in our case gcc 4.9). There are some self written utilities in the following build script, but since they are specially taylored to the Hummel cluster, we will not list them here.

```
#!/bin/bash

set -e
set -x
test "$(rrz-build-tools-version)" -ge 7

bootstrap_build=true

package=go
whatis="The Go programming language."
url=https://golang.org/
tarname=$package
version=1.4.3
version_suffix=
modtypes='tool'
prereq=
helptext="This is the compiler and runtime for Go."
configure_flags=""

source_file=${tarname}$version.src.tar.gz
source_hash=sha512sum:12bade4bce9aa
```

```
source_dir=$tarname
source_url=https://storage.googleapis.com/golang/$source_file


. $(most-deep-existing $PWD/rrz-build-env.sh)
. $(most-deep-existing $PWD/config/$RRZ_BUILD_CONFIG/config.sh)
test -e config-local.sh && . ./config-local.sh

if $bootstrap_build; then
  helptext="This is a minimal Go compiler for bootstrapping."
  export GCO_ENABLED=0
  # just get it over with
  build_script=make.bash
else
  module load go/1.4.3
  export GOROOT_BOOTSTRAP="$(dirname "$(dirname "$(type -p go)")")
   "
  # including test suite
  build_script=all.bash
fi

# ----- end of manual configuration -----

builddir=$PWD
fullversion=$version$version_suffix

# Begin logging.
eval "$(rrz-sh-autolog \
  $builddir/$package-$version.build.log \
  $RRZ_PREFIX/$package/$package-$version.build.log.bz2)"

for n in $prereq
do
  module load $n
  module list -t 2>&1 | grep -q $n
done
export LDFLAGS="$(rrz-build-linkpath -R)"
unset LD_LIBRARY_PATH

distfiles=${RRZ_DISTFILES:-$builddir/tar}
workprefix=${RRZ_WORKDIR:-$builddir/work}
workdir=$workprefix/$package/$fullversion

env > build.env
mkdir -p "$workdir"

printf "%s\t%s\t%s\n" $source_file $source_url $source_hash \
  > urls.txt

rrz-download -d=$distfiles

rm -rf "$workdir"
mkdir -p "$workdir"
```

```
 74 cd "$workdir"
 75 tar -xf $distfiles/$source_file
 76
 77 echo "RRZ: Baue in $PWD/$source_dir"
 78 cd $source_dir
 79
 80 for i in "$builddir"/patches/*.patch
 81 do
 82   if test -e "$i"; then
 83     echo "RRZ:Patch: $i"
 84     patch -Np0 < "$i"
 85   fi
 86 done
 87
 88 prefix=$RRZ_PREFIX/$package/$fullversion
 89 export GOROOT_FINAL=$prefix
 90
 91 $CC --version
 92
 93 cd src
 94 ./$build_script
 95
 96 # Cleanup traces of earlier install.
 97 rm -rvf $prefix
 98 rm -rvf $RRZ_PREFIX/$package/$package-$fullversion.*
 99
100 cd $workdir
101 mkdir -p $(dirname $prefix)
102 mv "$source_dir" "$prefix"
103
104 cd $builddir
105 mv -v build.env $RRZ_PREFIX/$package/$package-$fullversion.
        buildenv
106
107 for t in $modtypes
108 do
109   mod=$RRZ_PREFIX/$package/$package-$fullversion.$t.module
110   echo "Erzeuge Module-Datei: $mod"
111   whatadd=
112   test $t = tool && whatadd=" (tools only)";
113   rrz-build-modulefile \
114     --type=$t --prereq="$(rrz-module-list $prereq)" \
115     --program="$package" --modversion="$fullversion" \
116     --prefix="$prefix" --whatis="$whatis$whatadd" \
117     --modhelp="$helptext" > $mod
118 done
```

In lines 9 to 26 initialization of variables is done, e.g. the URL, the path for downloading the InfluxDB source (line 11), the package and version name of InfluxDB (line 9 and 13) and the source download address of the local server (line 22). In lines 30 to 40 the decision will be made, which kind of GO building should be done. If the variable bootstrapbuild is set to TRUE, then a bootstrap build will be made and the build_script variable is set to make.bash (this Makefile is part of the package). Otherwise the build_script variable is set to all.bash and the GO module is loaded, to build GO 1.9 with GO 1.4. After this initialization and configuration phase the existing GO and git modules are loaded, which will be needed, if GO 1.9 is build with GO 1.4. Otherwise the variable prereq is unset(line 52 to 58). In line 57 the linker environment variable LDFLAGS will be set and in the next line the variable LD_LIBRARY_PATH will be unsetted.

The download section starts with line 67. In line 67 f. the source tarball, the URL, from where to download the tarball and the hash of this tarball will be redirected into urls.txt, which serves as the source for the download with rrz-download in line 70. In line 72 f. the current GO workspace will be created, in which, in the first step, a possible old workspace tarfile will be deleted and recreated. Changing into the new workspace and untaring the downloaded tar file will complete the creation of the temporary workspace. The patch and build process begins with changing into the source directory of GO. Patching files is done in lines 80 to 86 and the build process of. ?I don't understand this sentence. And what is meant by f. which is used several times in this paragraph? In line 98 f. the core build will be done. A change into the directory GO/src and the start of the build script are the actions for this case. In this build script the build of GO 1.4 by bootstrapping or the build of GO 1.9 with GO 1.4 will be performed. After cleaning up of temporary files and directories of earlier installs, from line 107 to 118 the appropriate GO module files with the additional information will be created.

Creating the workspace for Telegraf and InfluxDB

According to the GO recommendations, the GO workspace has to be created, which has in general the following structure (GO recommendations):

- $HOME/go:
  - bin: Directory, which contains the binaries of the source files in src,
  - pkg: directory, which contains packages and libs,
  - src: contains GO sources of the different projects.

The src subfolder consists of (possibly multiple) version control repositories (e.g. Git) that consist of one or more source packages (as subfolders of the version control repositories) of GO source files (see https://golang.org/doc/code.html).

In the case of Telegraf and InfluxDB on the cluster Hummel, this recommendation was slighly modified. In the case of Telegraf, the workspace folder is Telegraf-1.x.y.wrk with the subfolders src and bin. The version control repositiory was named github.com/InfluxData/ with the package Telegraf (subfolder of the version control repository). The sources of Telegraf are in those folders. Additional dependency directories exist in the version control repository. This structure is visualized in the following itemize „graph":

- $HOME/Telegraf-1.x.y.wrk:
  - bin

  - src
    - github.com/InfluxData (the version control repository)
      - Telegraf (contains the realease and dependency files of Telegraf)

The workspace structure of InfluxDB looks similar to the Telegraf workspace. (Substitute Telegraf with InfluxDB.) After compiling and building Telegraf with the release and dependency source files (and possibly using libraries and additional packages), the appropriate binaries are present in the bin folder of the Telegraf workspace.

To build this workspace we are using the following bash script:

```
 1  #!/bin/sh
 2
 3  set -e
 4
 5  package=$1; shift
 6  version=$1; shift
 7
 8  wrk=$PWD/$package-$version.wrk
 9  tar=$PWD/$package-$version.tar.gz
10  if ! test -e $package-$version.wrk; then
11    mkdir $package-$version.wrk
12  fi
13  cd $package-$version.wrk
14  export GOPATH=$PWD
15
16  if ! test -e src/github.com/influxdata; then
17    mkdir -p src/github.com/influxdata
18    cd       src/github.com/influxdata
19    tar -xf $tar
20    mv $package-$version $package
21    cd $wrk
22  fi
23
24  cd $wrk/src/github.com/influxdata/$package
25  gdm restore --parallel=false
26  cd $wrk/..
27  tar --exclude=.git -czf $package-$version.wrk.tar.gz $package-
        $version.wrk
```

In lines 10 to 12, this script file creates the workspace of the Telegraf project, if it does not exist. The folder name consists of the package name and the version number, which are both parameters of the bash script. In this case the name of the workspace directory is Telegraf-1.x.y.wrk. After changing into this directory, the GOPATH is set to GOPATH=Telegraf-1.x.y.wrk (see lines 13 and 14). If the version control directory does not exist, it will be created in line 16. After getting into the directory $wrk/src/github.com/influxdata/$package (line 17), untaring the Telegraf tarball with the release files (line 18) and renaming the received folder, the download of the dependency source files with gdm restore take place in line 25. The flag –parallel=false supresses the start of more than one thread, because then the download from github fails. gdm will „checkout the dependencies in your current GOPATH. This will modify repos in your GOPATH“(see https://github.com/sparrc/gdm). After changing into the $wrk directory the last step is the creation of our workspace tarball (excluding the .git directory, because it is too large). This procedure downloads the influxdata release tarball and transforms it into the workspace tarball. With this workspace tarball we can build the packages of with the shell scripts of the RRZ. The downloads are separated from binaries (archived).

Installing and configuring InfluxDB

In this subsection we will briefly describe those steps on how to download the release and dependency files of InfluxDB and installing the InfluxDB database. Since a lot of steps a similar to the GO installation procedure, we only consider those lines which contain new aspects of the installation process. Furthermore, we will have a look at the the configuration of the InfluxDB, in which we only consider those lines, which are different to the default configuration the file in section ??.

The following bash script listing shows the installation process of InfluxDB.

```
#!/bin/bash

set -e
set -x
test "$(rrz-build-tools-version)" -ge 7

package=influxdb
whatis="Scalable datastore for metrics, events, and real-time
    analytics"
url=https://portal.influxdata.com/
tarname=$package
version=1.4.3
version_suffix=                     # typically set in config-local.
    sh.
modtypes='tool'
prereq=
helptext=""
configure_flags=""

source_file=$tarname-$version.wrk.tar.gz
source_hash=sha512sum:861b5a5a36a347
source_dir=$tarname-$version.wrk
source_url=http://src.rrz.uni-hamburg.de/extra/gostuff/
    $source_file

. $(most-deep-existing $PWD/rrz-build-env.sh)
. $(most-deep-existing $PWD/config/$RRZ_BUILD_CONFIG/config.sh)
test -e config-local.sh && . ./config-local.sh

module load go
module load git
```

```

# ----- end of manual configuration -----

builddir=$PWD
fullversion=$version$version_suffix

# Begin logging.
eval "$(rrz-sh-autolog \
  $builddir/$package-$version.build.log \
  $RRZ_PREFIX/$package/$package-$version.build.log.bz2)"

for n in $prereq
do
  module load $n
  module list -t 2>&1 | grep -q $n
done
export LDFLAGS="$(rrz-build-linkpath -R)"
unset LD_LIBRARY_PATH

distfiles=${RRZ_DISTFILES:-$builddir/tar}
workprefix=${RRZ_WORKDIR:-$builddir/work}
workdir=$workprefix/$package/$fullversion

env > build.env

# _create_ urls.txt and download files
printf "%s\t%s\t%s\n" $source_file $source_url $source_hash > urls
    .txt

rrz-download -d=$distfiles

rm -rf "$workdir"
mkdir -p "$workdir"
cd "$workdir"
tar -xf $distfiles/$source_file

echo "RRZ: Baue in $PWD/$source_dir"
cd $source_dir

export GOPATH=$PWD
PATH=$GOPATH/bin:$PATH

for i in "$builddir"/patches/*.patch
do
  if test -e "$i"; then
    echo "RRZ:Patch: $i"
    patch -Np0 < "$i"
  fi
done

prefix=$RRZ_PREFIX/$package/$fullversion
export PREFIX=$prefix

```

```
$CC --version

( cd src/github.com/influxdata/$package && go install -ldflags="-X main.version=$version" ./... )

# Cleanup traces of earlier install.
rm -rvf $prefix
rm -rvf $RRZ_PREFIX/$package/$package-$fullversion.*
# New install. Might install to a DESTDIR first and package things.
# Hm, that's a thought. I remember my old packinstall scheme.
# But that did not have a separate directory per $package/$version,
# so the add-on value is limited. --ThOr
mkdir -p "$prefix/bin"
cp -v "$workdir/$source_dir/bin/"* "$prefix/bin"

cd $builddir
mv -v build.env $RRZ_PREFIX/$package/$package-$fullversion.buildenv

for t in $modtypes
do
  mod=$RRZ_PREFIX/$package/$package-$fullversion.$t.module
  echo "Erzeuge Module-Datei: $mod"
  whatadd=
  test $t = tool && whatadd=" (tools only)";
  rrz-build-modulefile \
    --type=$t --prereq="$(rrz-module-list $prereq)" \
    --program="$package" --modversion="$fullversion" \
    --prefix="$prefix" --whatis="$whatis$whatadd" \
    --modhelp="$helptext" \
  > $mod
done
```

After initialization in the lines 27 and 28, the GO and Git modules are loaded, which will be needed for the following build process. And these lines are the only new aspects in this script with respect to the GO build script. The following logging, module loading, downloading, patching, building and module creation sections are basically the same as in the GO script. In the following we will list the contents of the config file of InfluDB and will briefly describe the differences to the default values (see section ...)

```
reporting-disabled = true
bind-address = "127.0.0.1:8088"

[meta]
  dir = "/srv/influxdb/db/meta"
  retention-autocreate = true
  logging-enabled = true

[data]
```

```
  dir = "/srv/influxdb/db/data"
  index-version = "inmem"
  wal-dir = "/srv/influxdb/db/wal"
  wal-fsync-delay = "0s"
  query-log-enabled = true
  cache-max-memory-size = 1073741824
  cache-snapshot-memory-size = 26214400
  cache-snapshot-write-cold-duration = "10m0s"
  compact-full-write-cold-duration = "4h0m0s"
  max-series-per-database = 1000000
  max-values-per-tag = 100000
  max-concurrent-compactions = 0
  trace-logging-enabled = false

[coordinator]
  write-timeout = "10s"
  max-concurrent-queries = 0
  query-timeout = "0s"
  log-queries-after = "0s"
  max-select-point = 0
  max-select-series = 0
  max-select-buckets = 0

[retention]
  enabled = true
  check-interval = "30m0s"

[shard-precreation]
  enabled = true
  check-interval = "10m0s"
  advance-period = "30m0s"

[monitor]
  store-enabled = true
  store-database = "_internal"
  store-interval = "10s"

[subscriber]
  enabled = true
  http-timeout = "30s"
  insecure-skip-verify = false
  ca-certs = ""
  write-concurrency = 40
  write-buffer-size = 1000

[http]
  enabled = true
  bind-address = "172.25.18.204:8086"
  auth-enabled = true
  log-enabled = false
  write-tracing = false
  pprof-enabled = true
  https-enabled = true
  https-certificate = "/etc/influxdb/cert2018.pem"
  https-private-key = "/etc/influxdb/key.pem"
  max-row-limit = 0
  max-connection-limit = 0
  shared-secret = ""
  realm = "InfluxDB"
  unix-socket-enabled = false
  bind-socket = "/var/run/influxdb.sock"
  max-body-size = 25000000

...

[continuous_queries]
  log-enabled = true
  enabled = true
  query-stats-enabled = false
  run-interval = "1s"
```

In the meta and data section of this configuration file, the path of the metadata, data and WAL directories where changed. The next changes to the default values were made in HTTP section. Here the InfluxDB instance was binded to the appropriate database server IP address and TCP port (172.25.18.204:8086). In contrast to the default value the authentication over HTTP was enabled (auth-enabled=true), as well as HTTPS (https-enabled=true). Authentification using HTTPS needs server sided certificates, which are located in *https-certificate=/etc/influxdb/cert2018.pem* and *https-private-key=/etc/influxdb/key.pem*. There are no further changes made in comparison to the default values.

In the following the service defaults are listed

```
[Unit]
Description=Influxdb timeseries database
After=network.target

[Service]
User=influxdb
ExecStart=/opt/sw/env/system-gcc/influxdb/1.4.3/bin/influxd -
   config /etc/influxdb/influxdb.conf
Restart=always
RestartSec=60

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

Installation and configuration of Telegraf

Since the steps to install Telegraf are similar to the installation steps of InfluxDB, we will refer to this part of InfluxDB and to the following bash script.

```
#!/bin/bash

set -e
set -x
test "$(rrz-build-tools-version)" -ge 7

debugstep=$1

package=telegraf
whatis="The plugin-driven server agent for collecting & reporting
    metrics with ProfiT-HPC plugins"
url=https://portal.influxdata.com/
tarname=$package
version=1.5.1
version_suffix=-pfit-20180123
modtypes='tool'
prereq=
helptext=""
configure_flags=""

source_file=$tarname-$version.wrk.tar.gz
source_hash=sha512sum:6abf81e59a976487876037edd15d6c22bbb9afdb057
    ...6bc92334c831a17bbb3b48b2b24bc7ce61
source_dir=$tarname-$version.wrk
source_url=http://src.rrz.uni-hamburg.de/extra/gostuff/
    $source_file

. $(most-deep-existing $PWD/rrz-build-env.sh)
. $(most-deep-existing $PWD/config/$RRZ_BUILD_CONFIG/config.sh)
test -e config-local.sh && . ./config-local.sh

module load go
module load git

# ----- end of manual configuration -----

builddir=$PWD
fullversion=$version$version_suffix

# Begin logging.
eval "$(rrz-sh-autolog \
  $builddir/$package-$version.build.log \
  $RRZ_PREFIX/$package/$package-$version.build.log.bz2)"

for n in $prereq
do
  module load $n
  module list -t 2>&1 | grep -q $n
done
export LDFLAGS="$(rrz-build-linkpath -R)"
unset LD_LIBRARY_PATH

```

```
distfiles=${RRZ_DISTFILES:-$builddir/tar}
workprefix=${RRZ_WORKDIR:-$builddir/work}
workdir=$workprefix/$package/$fullversion

export GOPATH="$workdir/$source_dir"
PATH=$GOPATH/bin:$PATH
export -p > build.env
mkdir -p "$workdir"

# _create_ urls.txt and download files
printf "%s\t%s\t%s\n" $source_file $source_url $source_hash > urls
    .txt

rrz-download -d=$distfiles

rm -rf "$workdir"
mkdir -p "$workdir"
cd "$workdir"
tar -xf $distfiles/$source_file

echo "RRZ: Baue in $PWD/$source_dir"
cd $source_dir

for i in "$builddir"/patches/*.patch
do
  if test -e "$i"; then
    echo "RRZ:Patch: $i"
    patch -Np1 < "$i"
  fi
done

prefix=$RRZ_PREFIX/$package/$fullversion
export PREFIX=$prefix

$CC --version

if test -e $builddir/sourcehack.sh; then
  $builddir/sourcehack.sh
fi

if test "$debugstep" = debug_build; then
  bash
else
  ( cd src/github.com/influxdata/telegraf && make VERSION=$version
    telegraf go-install)
fi

# Cleanup traces of earlier install.
rm -rvf $prefix
rm -rvf $RRZ_PREFIX/$package/$package-$fullversion.*

mkdir -p "$prefix/bin"
cp "$workdir/$source_dir/bin/telegraf" "$prefix/bin"
```

```
if test -e "$workdir/$source_dir/src/github.com/influxdata/
    telegraf/pfit-exec-scripts"; then
  cp "$workdir/$source_dir/src/github.com/influxdata/telegraf/pfit
    -exec-scripts/"*.sh $prefix/bin
  chmod +x $prefix/bin/*
fi

cd $builddir
mv -v build.env $RRZ_PREFIX/$package/$package-$fullversion.
    buildenv

for t in $modtypes
do
  mod=$RRZ_PREFIX/$package/$package-$fullversion.$t.module
  echo "Erzeuge Module-Datei: $mod"
  whatadd=
  test $t = tool && whatadd=" (tools only)";
  rrz-build-modulefile \
    --type=$t --prereq="$(rrz-module-list $prereq)" \
    --program="$package" --modversion="$fullversion" \
    --prefix="$prefix" --whatis="$whatis$whatadd" \
    --modhelp="$helptext" \
  > $mod
done
```

Enabling HTTPS

In the last step, HTTPS have to be enabled.

```
openssl ecparam -genkey -name prime256v1 -out key.pem
openssl req -new -sha256 -key key.pem  -out csr.csr
openssl req -x509 -sha256 -days 1460 -key key.pem -in csr.csr -out
     cert2018.pem
```

In the first line the private key, which is located on the InlfuxDB server, will be created. In this case an elliptic curve private key of the size 256 will be generated (parameter ecparam, -name prime256v1 and genkey). This will be stored in the file key.pem, which is in the pem format. In the second step, a Certificate Signing Request (CSR) will be created and stored into the file csr.csr. In the third and last step, the foregoing CSR have to be signed. This can be done like in the last line, in which the certificate signing request input file csr.csr is the input and the output of this process is the file cert2018.pem. This certificate has a validity of 4 years.

Gefördert durch die Deutsche Forschungsgemeinschaft (DFG)
KO 3394/14-1, OL 241/3-1, RE 1389/9-1, VO 1262/1-1, YA 191/10-1

List of Figures