



NOVA

IMS

Information
Management
School

MAAA

Mestrado em Métodos Analíticos Avançados
Master Program in Data Science and Advanced Analytics

**Centralised Monitoring and Alerting Solution for
Complex Information Management Infrastructure**

Yuri Ivanov Binev

Internship report presented as partial
requirement for obtaining the Master's degree in
Data Science and Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

**CENTRALISED MONITORING AND ALERTING SOLUTION FOR COMPLEX
INFORMATION MANAGEMENT INFRASTRUCTURE**

by

Yuri Binev

Internship report presented as partial requirement for obtaining
the Master's degree in Data Science and Advanced Analytics

Advisor: prof. Flávio Pinheiro

Co Advisors: Frederico Carvalho and Filipe Camelo

September 2020

ABSTRACT

Monitoring and alerting receive increasing attention over the last years together with growth of the information generated and managed from one side and increasing of the computing power and capacity of the computing hardware. Monitoring solutions have been tightly linked to the software being monitored making the broad view of the performance of the all interlinked services too complex and ineffective. In this work a monitoring and alerting solution based on “Prometheus” is developed. Continuous collection of metrics from various different services is performed and organized for visualization and monitoring in several layers of precision. Different exporters for various systems were evaluated and many of them enhanced. Alerting logic in response of detected performance problems and function irregularities has been developed and implemented with “Alert Manager”. Another software is developed for recording and visualizing current or past alerts and also is used as debugging tool of the alerts configuration. Visualization is implemented for Grafana with several dashboards. All tools and software packages used for implementing this monitoring and alerting solution are open source and free to use.

KEYWORDS

Monitoring and Alerting, Visualization, Big data, Time series, Prometheus, Grafana, Alert Manager

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
CPU	Central Processing Unit, Main Processor
DAG	Directed Acyclic Graph: Collection of tasks to be run in Airflow
ELK	Elasticsearch, Logstash, Kibana: The components of Elastic Stack. To the same stack belongs Beats even is not included in the abbreviation
ERD	Entity-relationship diagram
HDFS	Hadoop Distributed File System
HDP	Hortonworks Data Platform (HDP®): Hadoop distribution by Hortonworks
GC	Garbage Collection
JMX	Java Management Extensions
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
PromQL	The query language of Prometheus
SMTP	Simple Mail Transport Protocol
TICK	Telegraf, InfluxDB, Chronograf, Kapacitor: The 4 main components of InfluxData platform.
TSDB	Time series database
YAML	Yet Another Markup Language or YAML Ain't Markup Language
YARN	Yet Another Resource Negotiator, Apache YARN

TABLE OF CONTENTS

1	Introduction.....	1
2	Overview of Existing Monitoring Solutions.....	2
2.1	Push vs. Pull Architecture.....	2
2.2	Ambari.....	2
2.3	Elasticsearch.....	3
2.4	InfluxData Platform.....	4
2.5	Prometheus.....	5
2.6	Grafana.....	6
2.7	Comparison between evaluated generic solutions.....	6
3	Installing and Configuring monitoring and Alerting solution with Prometheus.....	8
3.1	Prometheus and Alert Manager.....	8
3.2	Exporters.....	8
3.2.1	Exporting metrics from Hadoop services.....	8
3.2.2	Exporters for Elasticsearch Ecosystem.....	10
3.2.3	NiFi Exporters.....	11
3.2.4	Exporters for specific services.....	12
3.3	LogAlerts Program.....	13
3.4	Grafana and additional panel plugins.....	14
4	Final Centralized Monitoring and Alerting Solutions.....	16
4.1	Testing Cluster.....	16
4.1.1	Monitoring Hadoop based services.....	16
4.1.2	Monitoring Elastic Stack.....	17
4.1.3	Monitoring Specific Services.....	17
4.1.4	Graphical representation of the Testing Cluster Monitoring.....	18
4.2	Production.....	18
4.2.1	Monitoring HDP clusters.....	19
4.2.2	Monitoring Database services.....	21
4.2.3	Airflow Cluster and Celery.....	22
4.2.4	Monitoring NiFi cluster.....	23
4.2.5	Monitoring Kafka clusters.....	23
4.2.6	Monitoring services from Elastic stack.....	24
4.2.7	Monitoring NginX.....	26
4.2.8	Monitoring the Operating Systems.....	26
4.2.9	Defining alerts and configuring Alert Manager.....	27
4.2.10	Prometheus self-monitoring.....	29
4.2.11	Structure of the Monitoring solution.....	29
4.3	Computational Chemistry Specialized Cluster.....	31
4.3.1	Additional Exporters.....	31
4.3.2	Building the monitoring solution.....	32
5	Conclusions.....	35
6	Supplementary Material.....	35
7	Limitations and recommendations for future works.....	36
8	References.....	37
9	Index.....	39

LIST OF FIGURES

Figure 1: Architecture of Ambari Metric System.....	3
Figure 2: Structure of monitoring solution based on Beats, Elasticsearch and Kibana.....	3
Figure 3: Structure of TICK Stack.....	4
Figure 4: Representation of generic monitoring and alerting solution with Prometheus, Alert Manager and Grafana....	5
Figure 5: Comparison of the three approaches for monitoring Hadoop services with Prometheus: JMX Exporter (left), Hadoop Exporter (middle) and Ambari Exporter (right).....	10
Figure 6: ERD of the database used in LogAlerts program.....	13
Figure 7: Representation of the Monitoring Solution for the Testing Cluster.....	18
Figure 8: Choosing ports for exporting metrics without (left) and with (right) using various IPs from loopback network	19
Figure 9: Common metrics for some services from both HDP clusters.....	20
Figure 10: Panel displaying HDFS usage and some temporal data storage rate and metrics.....	20
Figure 11: Internal Hadoop Exporter metrics: CPU and memory usage, left and scrape delays, right.....	20
Figure 12: Metrics from Presto server associated with HDP2.....	21
Figure 13: Fragment of a dashboard with Redis metrics.....	21
Figure 14: Some PostgreSQL Metrics.....	22
Figure 15: Airflow temporal DAGs and Tasks Performance (up) and status overview (down).....	22
Figure 16: Celery dashboard example.....	23
Figure 17: Fragment of a dashboard displaying status and performance of Apache NiFi cluster.....	23
Figure 18: Example of simple dashboard for Kafka.....	24
Figure 19: Tabular representation of common metrics from some Logstash instances.....	24
Figure 20: Temporal plots for incoming events rate (left) and average delay (right).....	24
Figure 21: Example for Metricbeat dashboard.....	25
Figure 22: Tables with Elasticsearch cluster-wide (up) and node-specific (down) status metrics.....	25
Figure 23: Health status (up) and plots of filesystem transfer rate (left) and documents rate (right).....	26
Figure 24: Some metrics from nodes of HDP2 cluster.....	26
Figure 25: Tables with alerts about filesystem , memory and CPU overuse.....	27
Figure 26: Service statuses collected by Runit (left), Systemd (top, right) and Supervisor (bottom, right) collectors..	27
Figure 27: Screen shot of the web interface of LogAlerts program displaying some active and resolved alerts together with alert labels and annotations.....	28
Figure 28: Panels with metrics from Prometheus and plot with current and estimated maximum TSDB size.....	29
Figure 29: Panel displaying various active (red) and pending (yellow) alerts.....	29
Figure 30: Final structure of the Centralised Monitoring solution for Production.....	30
Figure 31: Snapshot of table panel with some of the operating system and generalised smartctl (the last 4 columns) metrics.....	32
Figure 32: IPMI metrics and generalised sensor status.....	33
Figure 33: Plots of some metrics from Apcupsd Exporter.....	33
Figure 34: Dashboard with status overview of the Cluster.....	34
Figure 35: Structure of the monitoring solution for the Cluster.....	34

1 INTRODUCTION

This scientific report describes development and implementation of centralized monitoring and alerting solution for information management services and infrastructure in Network Analytics department in a big telecommunication company^a. The department employs diverse *big* and classic data solutions with complex interconnections between them. Partial monitoring solutions already existed however they are specific for the various services and Big Data ecosystems. Implemented and constantly enhanced and reinforced data and infrastructure security policies and personal data protection rules^b impose additional restrictions for monitoring solutions and alerting routes.

Centralized monitoring solution would solve inconsistencies between the various ecosystems and will make effective response and troubleshooting possible without circumvent or loosen the information and infrastructure security.

Programs and services that are monitored include various services from Hadoop ecosystem bundled in two of HDP distributions, two separated Kafka clusters, Airflow cluster, three different versions of Elasticsearch, multiple instances of Logstash, Filebeats, Apache, two separated Celery instances, NiFi cluster, various Postgres servers, three separated Redis servers, two Presto servers, an internal Gitlab installation. Also operating system and hardware metrics are monitored from multiple virtual and real hosts.

^a NOS, Portugal

^b Segment network access, Principle of Least Privilege among the others

2 OVERVIEW OF EXISTING MONITORING SOLUTIONS

An overview of existing and well known monitoring solutions was done. They were evaluated with the five primary facets of a monitoring system: Data collection, Data storage, Visualization, Analytics and reporting, Alerting [1]. Additional attention was paid to the ease of implementation and support, resiliency, integration, cost.

2.1 PUSH VS. PULL ARCHITECTURE

Two main paradigm exist in monitoring architectures, accordingly to the site of data collecting component [1]: *Push* and *Pull*^a.

In *Push* architecture the data collection component is passive and the metrics and events are sent on a regular basis or on event by the monitored program or the metric forwarding component. So the monitored services, machines or infrastructure are *emitters* that send data to a central collector [2]. In the most cases the emitters send the data to the collectors as soon it is available or the event happened, avoiding the delay of time based collection and the risk of losing rapidly evolving metrics or events. The collection of the metrics and events are distributed over the monitored entities therefore it is horizontally scalable.

In *Pull* architecture the data collecting component is the active part. It emits requests to the monitored targets to collect the metrics and is the central part that is configured for the monitored targets. The requests and data collection are performed on regular intervals and the result suitable for storage and management as time series, but is prone to loss of short lived events.

Monitoring solutions with combined Push/Pull architectures are also studied and described [3] as effort to combine positive sides of both approaches and mitigate their shortcomings. Earlier a specific purpose monitoring solution implementing such hybrid model was implemented and described [4], motivated by complementary properties of each simpler model.

Both types of architectures have advantages and disadvantages making them suitable for one environment or another. For example *Push* would be better choice for event-driven monitoring while *Pull* is for continuous metric collection of time series [5].

2.2 AMBARI

Apache Ambari [6] is management and monitoring software for Hadoop clusters. It may be installed separately however often comes bundled with proprietary Hadoop distributions such as HDP [7]. It has simple but powerful web interface for managing the services. Monitoring is implemented over JMX exposed to the network and has alerting ability on pre-defined events. Monitoring with Ambari is an example of *Pull* monitoring architecture. Simple representation of the Ambari metric system is shown on Fig. 1.

^a Still referenced as *white box* and *black box* respectively [2]

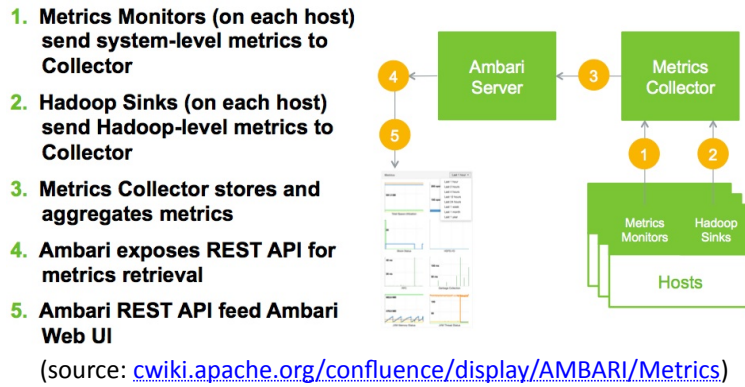


Figure 1: Architecture of Ambari Metric System

2.3 ELASTICSEARCH

Performance of Elasticsearch is monitored by its companion visualisation tool Kibana which is part of Elastic Stack [8] or Open Distro [9] packages.

Elastic Stack offers extension pack for monitoring and alerting of the components of ELK. Metrics include performance, system and CPU load, disk space used and others. Visualisation and alerting is performed in Kibana.

Open Distro offers options to build similar metrics and configure alerts based on selected conditions or detected anomalies.

More universal monitoring system to include external metric sources may be build after including “beats” to collect metrics and events and to send them to the Elasticsearch cluster. “Beats” are usually small native programs, in some cases extendible with plugins to support specific metric and event sources. Beats are the active components sending the metrics to Elasticsearch directly or through instances of Logstash. Several Beats implementations are available [10] and officially supported. Fig. 2 displays model of monitoring solution based on Elasticsearch and Beats also showing its *Push* type architecture.

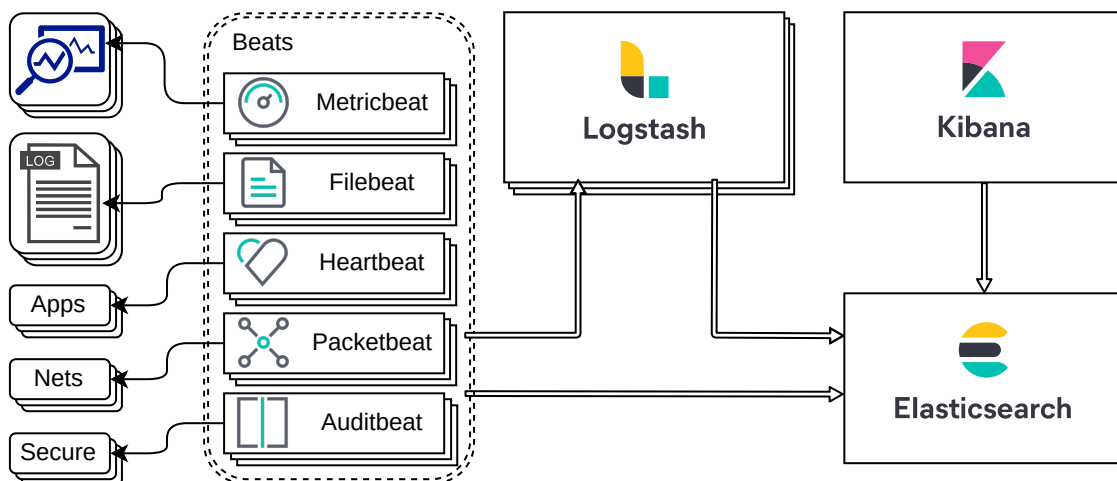
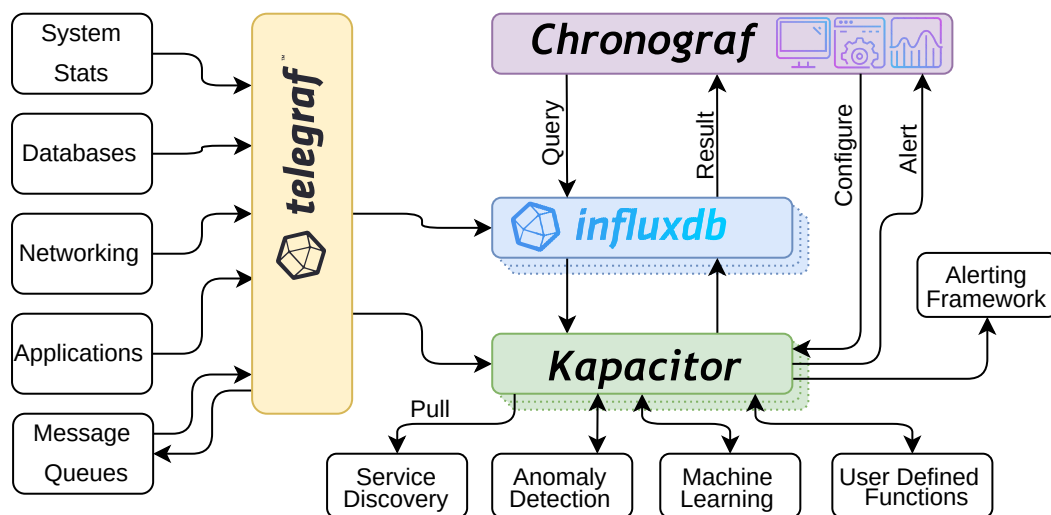


Figure 2: Structure of monitoring solution based on Beats, Elasticsearch and Kibana

2.4 INFLUXDATA PLATFORM

InfluxData platform, also known as TICK Stack [11], is a widely used generic monitoring platform, mostly free and open source, designed for collecting and storing time series capable of real time monitoring, alerting and machine learning based analytics. Among the free platform InfluxData promotes two more proprietary solutions commercialised as Influx Cloud and Influx Enterprise. Both are based on TICK stack and are managed by InfluxData, former is run on Amazon’s AWS and the latter – on any internal or external platform chosen by the customer.

The platform consists of four main components: Telegraf, InfluxDB, Chronograf and Kapacitor. All the components are developed in Go [12] resulting in great performance and portability. Each component has specific tasks and communicate with the others by HTTP based API allowing them to work independently. The next Fig. 4 shows the structure of TICK Stack. Similarly to ELK Stack solution it has a mostly *Push* type architecture.



(Recreated from www.influxdata.com/time-series-platform/kapacitor/)

Figure 3: Structure of TICK Stack

Telegraf is the component collecting data from the monitored services [13]. It is designed with plugin architecture with plugins grouped in 4 main types – inputs, outputs, aggregators and processors. Input plugins collect statuses and metrics from the operating system and hardware of the monitored machines and running services. Large number of input plugins are already available and new ones may be developed [14]. Output plugins send the collected and processed data to databases and aggregation systems typically InfluxDB and Kapacitor but there are options for external services and APIs like Openmetrics, Elasticsearch, Kafka among others. Multiple destinations can be defined.

InfluxDB is the central component of a monitoring solution. It is a specialized time series database capable of high data and query load [15]. Paid version also exists which includes horizontal scalability and clustering support. InfluxDB receives data from collecting agents such as Telegraf. Multiple levels of compression and granulation and retention policies are implemented. Queries are done in SQL based language called “InfluxQL” or more recently developed and more specialized query language called “Flux”.

Chronograf is the visualisation and configuration tool of the InfluxData platform[16]. It is a web-based tool with support for custom dashboards for various types of data and alerts visualisations and multilevel access controls and various authentication schemes. Configuration utilities include managing InfluxDB and Kapacitor.

Kapacitor is the data processing component [17] bearing various modules with advanced statistical and machine learning algorithms or user defined functions and capable for querying and storing back to InfluxDB. Kapacitor is also responsible for defining and sending alerts through various messaging and alerting frameworks. The paid version has clustering support for horizontal scaling.

2.5 PROMETHEUS

Prometheus [18] is a time series database integrated with metrics collection and storage capabilities. It uses own query language PromQL [19] permitting the users to select and aggregate data together with functions for conversion and manipulations of the metrics. Defined are 2 basic types of metrics: Counter and Gauge. Counter is a metric that always increase, not necessary by integer numbers. Examples include CPU time used by some process, network traffic passed through some network interface. A decrease of a Counter is considered as “Reset” and is ignored by functions calculating rates. A Gauge is a metric that may both increase and decrease. Examples for gauges may be temperature or memory usage.

Prometheus includes simple visualization web-based “Expression Browser” used for preparing queries and exploring time series. It is not suitable for general purpose monitoring and visualisation [20]. To create visualisation dashboards external tool such as Grafana should be used.

To collect metrics Prometheus relies on “Exporters”. Exporters generally are small programs that collect metrics from the monitored system and to translate to simple text format ready to be collected by Prometheus on regular intervals, typically 30 seconds to 4 minutes. Software developers are encouraged [21] to implement exporting metrics in Prometheus format such as Prometheus be able to collect metrics directly, without the need of intermediate exporter. A notable example is NginX with its VTS module [22].

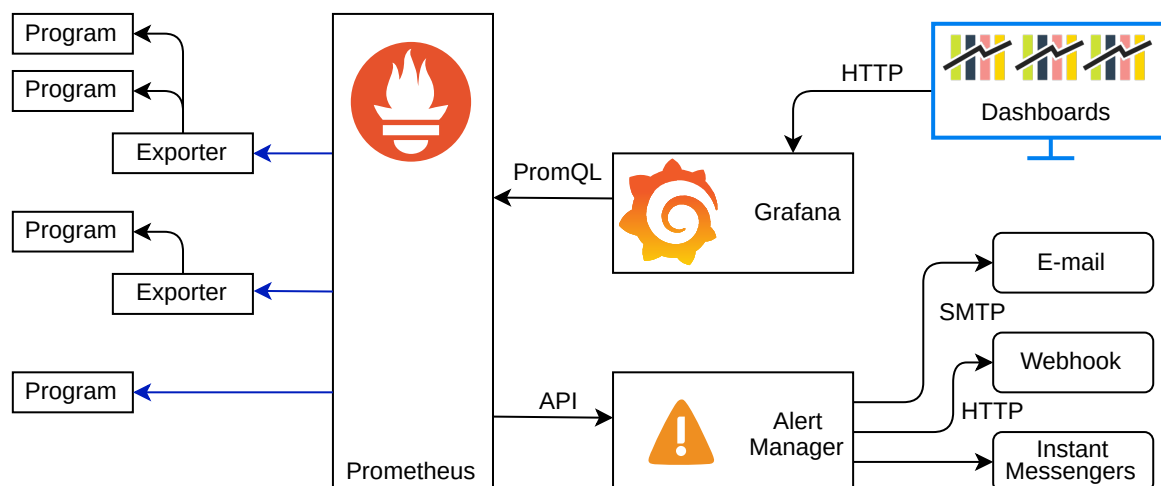


Figure 4: Representation of generic monitoring and alerting solution with Prometheus, Alert Manager and Grafana

There are many “official” exporters available [23]. Additionally libraries for most major languages are developed making relatively easy creating new or custom exporters.

There is internal alerting component for defining and sending alerts to the companion software Alert Manager or any other program implementing the appropriate API. According to the configuration Alert Manager will group alerts and forward through the some of the supported channels. Fig. 4 shows an example of simple monitoring solution with Prometheus and Grafana.

2.6 GRAFANA

Grafana [24] is a free and open source visualization tool and is widely used. It is written in Go [12] making its native executables compatible with wide range of hardware, operating systems, distributions and versions.

Many data sources are available through plugins, including for Elastic Search, InfluxDB and Prometheus which are have are internally supported as “Native plugins” [25]. Visualization is implemented as dashboards and panels presented in web based application. Huge collection of panel plugins exist making visualisation in Grafana very flexible.

Grafana has internal alerting solution configurable on metric values through visualization panels. Alerts can be send by e-mail, several instant messaging networks or external APIs and Webhooks. Alert Manager from Prometheus is also supported.

Being free and open source, together with it’s resilience and flexibility make Grafana the preferred choice for many monitoring and visualisation solutions.

2.7 COMPARISON BETWEEN EVALUATED GENERIC SOLUTIONS

Following the criteria listed on page 2, Influxdata (TICK Stack), Elasticsearch (ELK Stack) and Prometheus/Grafana based solutions are compared on Table 1.

Table 1: Comparison between InfluxDB, Elasticsearch and Prometheus based monitoring solutions

	TICK Stack	ELK Stack	Prometheus/Grafana
Source	Open source	Mostly Open source	Open source
Licensing	Mostly free	Free/Optional paid ^a	Free
Design/Purpose	Monitoring and Alerting	Search Engine	Monitoring and Alerting
Alerting	Kapacitor	Not by design	Alert Manager
Architecture type	Push	Push	Pull
Installation	Moderate	Straight forward	Easy
Configuration	Moderate	Simple/Moderate	Moderate
Diversity of sources	Many, Possible to develop new	Few	Many, Easy to develop new
Scalability	Clustering, paid	Clustering	Federation

^a Open Distro [9] is Free and Open source alternative

	TICK Stack	ELK Stack	Prometheus/Grafana
Development language	Go	Java	Go
Operation requirements	Moderate	High	Low

Although Elasticsearch is installed and used in the department the other two solutions – InfluxDB and Prometheus – have as advantage being designed for the purpose of monitoring and alerting. Additionally both have support for many services and being developed in Go is expected to exhibit good stability and performance.

Prometheus has the advantage having completely free and open source license, including the advanced features. The *Push* architecture makes Prometheus based solution more suitable for the purpose as monitored data consists mostly of metrics and logs and events were considered lower priority.

Prometheus based solution was selected for further exploration and implementation.

3 INSTALLING AND CONFIGURING MONITORING AND ALERTING SOLUTION WITH PROMETHEUS

3.1 PROMETHEUS AND ALERT MANAGER

Prometheus and Alert manager are downloadable [26] as binary packages for various operating systems including Linux. The binaries are static and independent of the distribution and version of the operating system. Neither need elevated access be run as regular or preferably as dedicated user with restricted access for improved security. The processes are managed either by supervisord or systemd and are configured as system or user services.

The configuration is done with command line options and one or more configuration files written in YAML. To create notification texts Alert Manager uses template files based on Go templating system [27].

Prometheus may need significant amount of disk space to store the database. In the main monitoring solution described in this work Prometheus database grew up to 25GB.

3.2 EXPORTERS

In this section will be reviewed existing exporters for the various services used in the Department. Will be taken in account running requirements, available metrics, stability and implications.

The most basic exporter is Node Exporter [28] which is stand-alone program. It collects metrics about operating system and hardware status and performance for Linux and collected metrics are exported with embedded http server for Prometheus. It has modular structure permitting enabling or disabling metrics collectors depending on needs.

Node Exporter must be installed and run on every monitored node. The executable is single file and there is no configuration files – the configuration is done by command line only and is managed by the generic Linux startup managers. In this work both systemd and supervisord were used.

By default Node exporter does not need to be run with elevated privileges. In some specific cases such as monitoring system manages special privileges may be needed.

3.2.1 Exporting metrics from Hadoop services

Many Java-based services use MBeans to communicate statuses and metrics. JMX Exporter [29] which is part of Prometheus ecosystem use MBeans to export metrics from the JVM. It is used as “Java agent” and must be included in the JVM command line that starts the monitored Java service:

```
java -javaagent://fullpath/jmx_prometheus_javaagent.jar=8080://fullpath/config.yaml -jar theService.jar
```

With above example metrics will be collected from MBeans and will be served by embedded http server listening on port 8080. Metrics will include generic ones available in any JVM, such as

memory and CPU usage and specific to the monitored application. The configuration file is used to redefine names of the exported metrics and optionally to exclude some.

Alternatively JMX Exporter may be used as stand-alone application but is discouraged as it is more complicated to configure and will not export the generic metrics [29].

With Ambari environment, such as HDP, inclusion of JMX Exporter as Java agent requires modification of the startup files `*_env.sh` of every service.

Including Java agent in the JVM has several implications:

- Adding or modifying JMX Exporter configuration in the JVM command line requires restart of the monitored service. Even in redundant character of Hadoop based services such restart may affect the stability of the whole ecosystem.
- Re-using a port number of the embedded http server or error in the configuration file will fail the inclusion of the Java agent and will prevent the JVM to start. Additionally there is no error message related to the JMX or Java agent on the output making such error difficult to track and resolve.
- Including multiple `-javaagent` options in the JVM command line is an error. This is particularly important in Ambari environment where the `*_env.sh` files are reused for multiple services and options are added cumulatively, so the `-javaagent` option should be only added if not exists already.
- The `jmx_exporter_javaagent.jar` and its configuration files must be copied to all nodes in the cluster.
- Re-configuration of the JVM command line of the services usually requires additional privileges to all the nodes of the cluster or when Ambari is used for management full access to Ambari configuration.

For services from Hadoop ecosystem instead if JMX Exporter the stand-alone Hadoop Exporter [30] may be used as alternative. It is a Python program designed to scrape metrics from multiple Hadoop based services and export them to Prometheus. The program relies on JMX web service to be enabled in the services, which is by default in the used HDP distributions. It does not require to be installed on the Hadoop nodes and can collect metrics remotely. Using this Hadoop Exporter avoids most of the difficulties implied by the JMX Exporter.

The original version of Hadoop Exporter has several shortcomings:

- Scrapping is done in single thread making it to take long time – from few seconds to tens of seconds.
- To get a list of services to be scrapped Hadoop Exporter relies on command line where only single service of a type can be specified or on a service by Ambari which is not enabled by default

To overcome those shortcomings Hadoop Exporter was heavily modified and enhanced^a:

- The command line options code was reworked
- There is no limit for number of services monitored of the same type
- Scrapping is made multi-threaded making the whole process much faster
- Multiple bug fixes and optimisations were done
- The list of services can be loaded from file avoiding the need of long command line
- The program is ported from Python 2.x to Python 3.5+

Another option for monitoring Hadoop services managed by Ambari is Ambari Exporter [31]. It collects the metrics from Ambari instead of directly as in Hadoop Exporter. This may have advantage of more centralized metrics transfer – all metrics are collected from the same server, but makes it too dependent of Ambari itself.

The Ambari Exporter is developed using .NET framework and designed to run in Microsoft Azure environment and compiling and running it in Linux requires additional efforts for installing new development tools and would introduce additional overhead for running the exporter in Wine and Mono [32].

Fig. 5 shows the difference between the above described approaches for monitoring Hadoop based services.

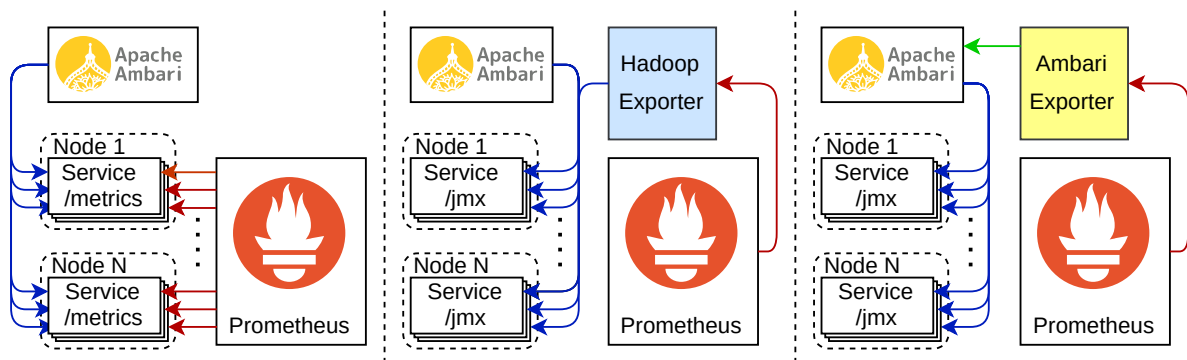


Figure 5: Comparison of the three approaches for monitoring Hadoop services with Prometheus: JMX Exporter (left), Hadoop Exporter (middle) and Ambari Exporter (right)

3.2.2 Exporters for Elasticsearch Ecosystem

Exporters for Elasticsearch, Beats and Logstash exist and they have similar functionality. All of them are stand-alone programs capable of collecting metrics remotely.

^a A fork of the of Hadoop exporter with those enhancements included is made public, keeping the original Open Source license, at https://gitlab.com/y1b/hadoop_exporter

Elasticsearch exporter [33] is a stand-alone application written in Go. Can be compiled from sources or used pre-compiled binaries. One instance per monitored node is required to collect node specific metrics both for Elasticsearch and the operating system. Cluster-wide metrics such as number of online nodes and cluster status are repeated from all exporter instances.

Configuration is done with command line and environment variables. For setting the credentials for access Elasticsearch environment variables should be used instead of command-line for improved security.

Several different Logstash Exporter forks exist, but the one published by “Bonnie News” [34] was chosen for further investigation. Similarly to Elasticsearch Exporter it is a standalone single executable program, but with much simpler configuration.

Logstash after version 5 has the necessary option “Web API” enabled by default however the listening port by default will take consecutive values starting from 9600, which leads to difficulties of the configuration of the exporter. It is advisable to set the “Web API” port to some fixed number ,using the setting `http.port`, specific for each Logstash instance running on the same machine.

The original sources were patched^a to implement some enhancements that were found necessary but missing:

- Add labels “name” and “id” to `Logstash_info_node` metric – this permits associating the Logstash instance with the Exporter instance even without setting the fixed number of “Web API” port;
- Add `Logstash_info_status` metric with possible values 0, 1 and 2 corresponding to status “green”, “yellow” and “red” if available;
- Add `start_time` metric.

Beats Exporter[35] is another standalone, single executable program written in Go. It has very few command-line options. It has full support for Filebeat and Metricbeat and partial for Packetbeat and Auditbeat.

Unlike Logstash Beats do not have the necessary “Web API” enabled by default. The options `http.enabled` and `http.port` must be set in the respective *beat configuration files.

3.2.3 NiFi Exporters

Several different exporters exist for Apache NiFi:

- `nifi_exporter` [36] is stand-alone program developed in Go configurable with single configuration file. It requires the authentication in NiFi to be enabled;
- NiFi Exporter [37] is a Python implemented exporter for NiFi to Prometheus;

^a `logstash_exporter.patch` in Supplementary Material

- nifi-prometheus-reporter [38] is a reporting task for NiFi to send statistics to Prometheus Pushgateway [39];
- Solution based on JMX Exporter [29] together with appropriate Grafana dashboard is available [40].

The Go implementation was compiled from sources without modifications and used for the monitoring solution of the testing cluster (see p. 16 below).

The second option – the Python implementation was modified^a to enable monitoring NiFi without authentication and is used for production.

3.2.4 Exporters for specific services

PostgreSQL instances can be monitored by the stand-alone PostgreSQL Exporter [41]. To permit remote monitoring and avoid using the exporter with elevated privileges both for the operating system and the database server it is required to create a dedicated database account with limited privileges together with special views with database-wide access.

PostgreSQL Exporter instances are managed by available Linux startup managers. The database connection string which contains the credentials for the dedicated database user is defined in environment variable for improved security.

Stand-alone dedicated Presto exporter with small set of metrics is available [42]. Run by the standard Linux startup managers and can be used remotely. Alternatively generic JMX Exporter based monitoring is also possible providing equivalent set of metrics.

Redis Exporter is available [43]. It was compiled from original sources without modifications. The exporter is capable of collecting metrics remotely over the network or locally through Unix socket.

Kafka metrics may be exported from Kafka brokers by JMX Exporter and the special configuration file included distribution.

As alternative stand-alone Kafka Exporter is available [44]. It may be used remotely no configuration changes or restart are needed but credentials for access are required.

NginX is widely used web-server implementation. The default setup has “ngx_http_stub_status_module” that exposes limited set of metrics that can be scrapped by “nginx-prometheus-exporter” [45]. The "status module" is not enabled by default in most binary distributions. Enhanced version of the module with more metrics is implemented in commercial "Nginx Plus". Independent module "nginx-module-vts"[22] exists and was selected as better alternative. The module is capable of exporting metrics directly in Prometheus format and may be compiled as plugin and enabled in the NginX main configuration file without the need of re-compilation of NginX itself. For compilation of the module, source code of exactly the same version of NginX is required.

^a NiFi_Exporter_patch in Supplementary Material

To monitor Airflow the Prometheus exporter[46] which is implemented as plugin in airflow. It defines a DAG and a Task to export metrics directly scrapable by Prometheus. Installation is done by common Python tools such as “pip install airflow-exporter” executed inside Airflow Python virtual environment.

Celery is another Python based service and it’s activity monitoring is performed on its database back-end by Celery Prometheus Exporter[47] which is stand-alone program developed in Python. It is run inside Python virtual environment with Celery and database back-end installed.

3.3 LOGALERTS PROGRAM

Alert Manager from Prometheus ecosystem has extensive features for receiving, grouping and forwarding or silencing alerts send by Prometheus or other software implementing it’s API. However it does not keep history of past inactive or resolved alerts which was found necessary for the monitoring solution. To meet this requirement the LogAlerts program was developed and published^a with open source license

LogAlerts implements partially Alert Manager API used by Prometheus (and eventually others) to send alerts. The API is at /alerts/v1/alerts and is accepting only POST requests with JSON describing active alerts. LogAlerts keeps track of active and resolved(not active, passed) alerts and stores them in SQLite3 database. The database is automatically created of startup if does not exists. The schema is hard-coded in the program. Fig. 6 shows the ERD of the database.

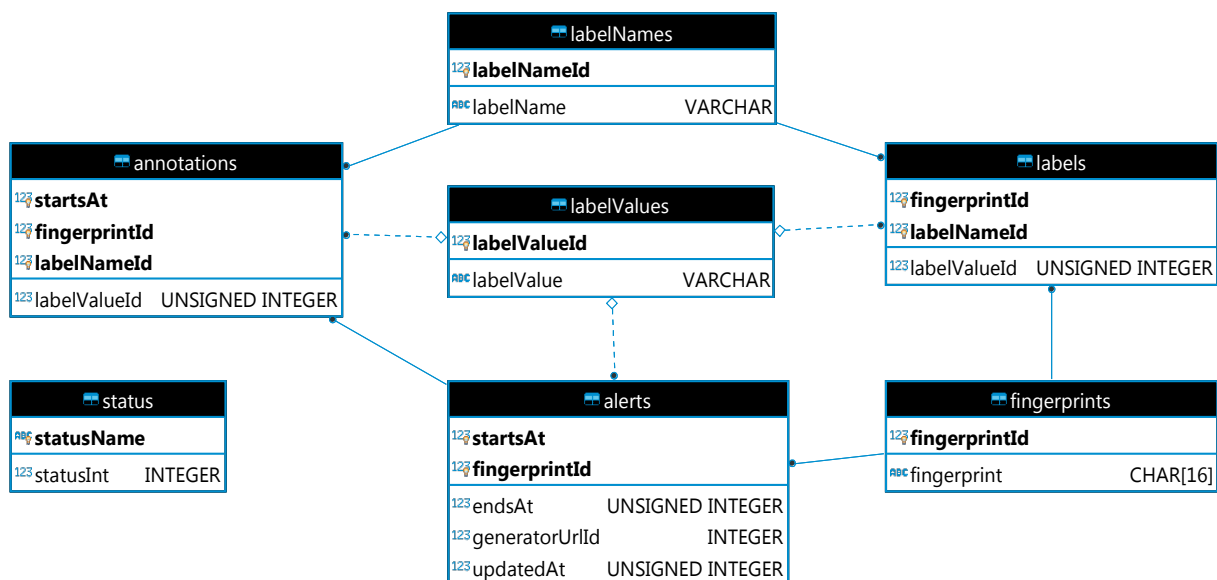


Figure 6: ERD of the database used in LogAlerts program

LogAlerts implements also simple web server, by default on port 9131, to serve the single index page and the list of alerts in JSON format, which is then rendered to a table by JavaScript. The page displays list of all alerts (active and resolved) started last 24 hours by default and optionally may be set to show active alerts only. Screen shot of the web interface is displayed on Fig. 27 on page 28.

^a YB, LogAlerts, <https://gitlab.com/y1b/logalerts>

Simple pagination is implemented to move back and forward by 24 hours. Also is possible to select manually day and hour. Active alerts are always displayed.

Automatic refresh every 60 seconds is implemented and enabled by default but it can be disabled. Columns Name, Description and Query are clickable to display more extensive information of the alert.

Lack of connectivity, down time, misconfiguration may cause LogAlerts to miss the API message that an alert was resolved. Then the alert is listed always as active, even is not shown in Prometheus.

The bug is mitigated by manually changing the "updatedAt" timestamp to be equal or bigger by few seconds to the "endsAt" timestamp for the alerts where "updatedAt" is older than (2 hours in the example) and endsAt is bigger than updatedAt. The following shell command may be used:

```
$ sqlite3 logalerts.db "update alerts set updatedAt=endsAt where endsAt>updatedAt and updatedAt<strftime('%s', 'now')-2*60*60;"
```

The API to update the alerts is allowed only from localhost, and so on the same host where Prometheus is running. No other access control is implemented. Everyone who has access to the listening port and any IP of the machine can read the list of alerts. Internal SQL queries are designed to be secure against common SQL threads (i.e. SQL injections).

To collect alerts from Prometheus it is necessary to include its local IP address and port in the "alerting" section of the Prometheus configuration file:

```
#alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
      - targets:
        - 127.0.0.1:9093 #AlertManager
        - 127.0.0.1:9193 #LogAlerts
```

AlertManager(s) and LogAlerts work in parallel and independently of each-other.

LogAlerts consists of single python file (logalerts.py) and 3 files with the web page code (index.html, index.css and index.js). Those files are never served directly, but they are combined by the program and then sent to the client. The program requires python3 but no special modules are required. The program is started by common startup managers and a restricted account may be used. A part from SQLite3 database logalerts.db and it's journal logalerts.db.wal no other write access is needed.

3.4 GRAFANA AND ADDITIONAL PANEL PLUGINS

Installing Grafana is straightforward – several distributions are available including generic Linux one [24]. It is uncompressed in the file system and can be run from there. The distribution includes various data source plugins including for Prometheus.

Despite of variety of included visualisation panels some non-standard panel plugins were used.

BoomTable [48] is a panel plugin for rendering tables in Grafana with multiple metrics. The plugin is still in development with frequent breaking changes. The version used is tagged "3c1ad62" in Github and downloadable as version 1.3 from Grafana.com.

This version orders columns produced by single query randomly and changes the order on every time change or refresh. It can be mitigated by performing multiple queries with different set of labels for each column – The columns will be displayed in the order of queries.

Changing the order of the rows is possible only by sorting by value of some column, which is not practical in most of the cases. By default no column is selected and rows are displayed by order served by Prometheus which can't be controlled. This was solved by small modification^a of the plugin code which will sort the rows by row name by default, when no sorting column is selected. There is no control that can make or reverse order of the rows – it is possible only by editing the "sorting_props" in the code of the panel. The ordering bugs are resolved by latter versions but new ones are introduced.

Statusmap is panel plugin to create temporal status maps [49]. It is frequently developed but no breaking changes are noticed and current version (r142, 4b7d111) works correctly.

To create pie charts panels special plugin is used [50] which is seldom updated and stable.

^a boomtable-sortbyrow-r260.patch in Supplementary Material

4 FINAL CENTRALIZED MONITORING AND ALERTING SOLUTIONS

The Department uses two separated clusters – one for testing and initial development and another for production. Additionally, as voluntary project, monitoring solution was implemented for a cluster dedicated to computational chemistry in external academic organization^a.

4.1 TESTING CLUSTER

The testing cluster consisted of 4 virtual nodes running Linux. It was running HDP v.2 distribution with HDFS, HBase, Hive and Kafka managed by Ambari. There there was Apache NiFi instance over all 4 nodes, Elastic Stack which was later replaced with Open Distro and a PostgreSQL server running on one of the nodes.

Prometheus, Alert Manager and Grafana were installed manually on one of the nodes.

Grafana was configured to use port 3009 instead of the default port 3000. Additionally the “Prometheus AlertManager Datasource” plugin for Grafana [51] was installed.

Alert Manager was configured to send alerts by e-mail through internal e-mail server.

The operating system and hardware metrics on all 4 nodes are exported by `node_exporter` which was installed and managed by `supervisord` without any special options enabled.

4.1.1 Monitoring Hadoop based services

All Hadoop based services were monitored by JMX Exporter installed as `javaagent`. For this purpose `jmx_prometheus_javaagent.jar` was copied to all 4 nodes and re-used for the monitored Hadoop services. The command line changes were done in Ambari administrator console.

For HDFS services the `javaagent` command line options were added in `hadoop-env.sh` after variables `HADOOP_..._OPTS` are defined. The modification avoids multiple inclusion of `-javaagent` option as explained shortly on page 9. With the code^b snipped below JMX exporter was configured^c to serve metrics on ports 7003 for Primary and Secondary Name Nodes and 7004 for the Data Nodes. The Name Nodes were on different machines which allowed to use the same port number for both.

```
[ -z "${HADOOP_DATANODE_OPTS##*-javaagent:*}" ] \
|| HADOOP_DATANODE_OPTS="$HADOOP_DATANODE_OPTS \
-javaagent:/-/jmx_prometheus_javaagent.jar=7004:/-/datanode_jmx.yaml "
```

```
[ -z "${HADOOP_NAMENODE_OPTS##*-javaagent:*}" ] \
|| HADOOP_NAMENODE_OPTS="$HADOOP_NAMENODE_OPTS \
-javaagent:/-/jmx_prometheus_javaagent.jar=7003:/-/namenode_jmx.yaml "
```

```
[ -z "${HADOOP_SECONDARYNAMENODE_OPTS##*-javaagent:*}" ] \
|| HADOOP_SECONDARYNAMENODE_OPTS="$HADOOP_SECONDARYNAMENODE_OPTS \
-javaagent:/-/jmx_prometheus_javaagent.jar=7003:/-/namenode_jmx.yaml "
```

HBase had Master Server running on one node and there were Region servers and Query servers on the other 3 nodes. The lines below were added to `hbase-env.sh` in HBase advanced configuration in Ambari:

^a DQ-FCT-UNL

^b Here and thereafter with “/-/” is denoted the full path to the file

^c Configuration files were taken from [Wikimedia Foundation Puppet repo](#)

```
[ -z "${HBASE_MASTER_OPTS##*-javaagent:*}" ] \
|| HBASE_MASTER_OPTS="$HBASE_MASTER_OPTS \
-javaagent:/-/jmx_prometheus_javaagent.jar=7001:/-/hbase_jmx.yaml "

[ -z "${HBASE_REGIONSERVER_OPTS##*-javaagent:*}" ] \
|| HBASE_REGIONSERVER_OPTS="$HBASE_REGIONSERVER_OPTS \
-javaagent:/-/jmx_prometheus_javaagent.jar=7002:/-/hbase_jmx.yaml "

[ -z "${PHOENIX_QUERYSERVER_OPTS##*-javaagent:*}" ] \
|| PHOENIX_QUERYSERVER_OPTS="$PHOENIX_QUERYSERVER_OPTS \
-javaagent:/-/jmx_prometheus_javaagent-0.12.0.jar=7010:/-/hbase_jmx.yaml "
```

Configuration file that may be used for this approach is available [52] and is the same for the 3 services.

After restarting the HBase services, metrics from HBase Master were served on port 7001, from the Region Server – on port 7002 and from Query Server – on port 7010.

Zookeeper's configuration file^a was distributed on the 3 nodes where Zookeeper was running and the line was added to the environment configuration:

```
[ -z "${SERVER_JVMFLAGS##*-javaagent:*}" ] \
|| SERVER_JVMFLAGS="$SERVER_JVMFLAGS \
-javaagent:/-/jmx_prometheus_javaagent.jar=7005:/-/zookeeper.yaml "
```

After restarting Zookeeper JMX Exporter was serving metrics on port number 7005

Similarly to above the configuration file^b was copied to the two Kafka broker nodes and the line was added to the Kafka environment configuration such as the metrics to be served on port 7006:

```
[ -z "${KAFKA_OPTS##*-javaagent:*}" ] \
|| KAFKA_OPTS="$KAFKA_OPTS \
-javaagent:/-/jmx_prometheus_javaagent.jar=7006:/-/kafka-0-8-2.yml "
```

4.1.2 Monitoring Elastic Stack

Elasticsearch Exporter [33] and Logstash Exporter [34] were installed on all nodes and configured to monitor the local Elasticsearch and Logstash instances. The metrics were served on ports 9098 and 9097 respectively.

Later the Elastic stack was replaced by Open Distro but no changes of the exporter configurations were necessary.

4.1.3 Monitoring Specific Services

To monitor the NiFi installation nifi_exporter [36] was installed on a single node and was configured to collect metrics from the local NiFi node.

On one of the nodes was running PostgreSQL server and PostgreSQL exporter was installed on the same node as described on page 12 above.

Prometheus, Alert Manager export metrics that can be collected by Prometheus. This option is enabled by default.

Grafana also exports Prometheus metrics but this option is not enabled in the default configuration. It was enabled^c in the modified configuration file.

^a JMX Exporter [Zookeeper example](#)

^b JMX Exporter [Kafka example](#)

^c enable=true in [metrics] group as [described](#)

4.1.4 Graphical representation of the Testing Cluster Monitoring

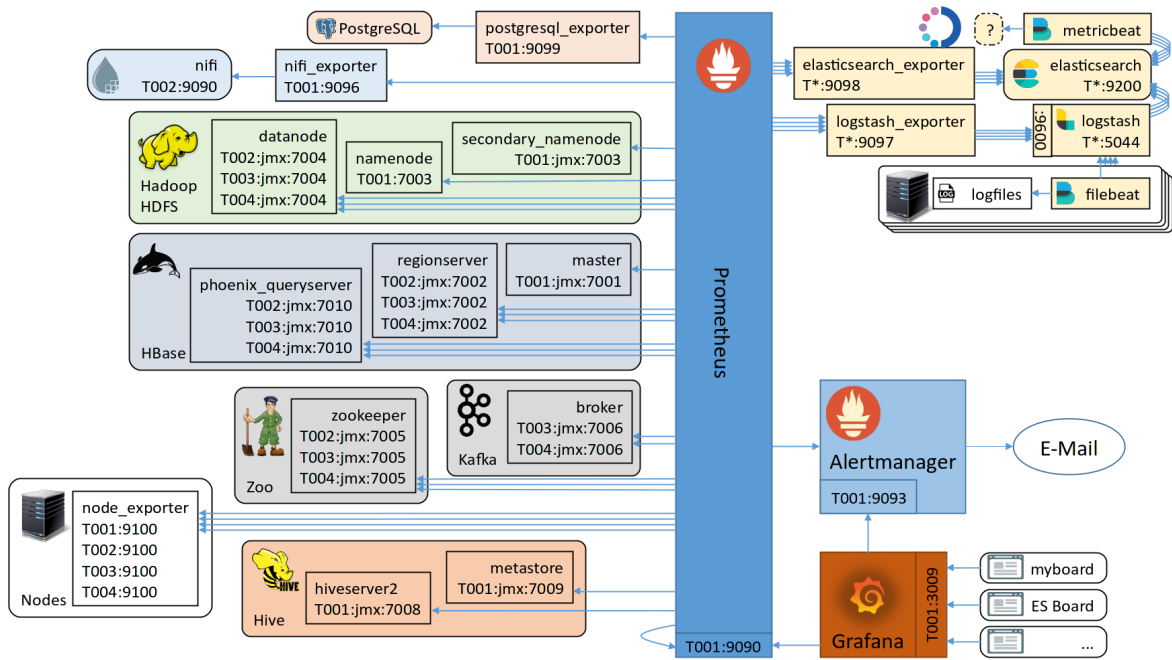


Figure 7: Representation of the Monitoring Solution for the Testing Cluster

4.2 PRODUCTION

Production environment includes of two separated HDP clusters, one full Elastic stack cluster, two Kafka clusters, an Apache NiFi cluster, Airflow cluster and various independent machines. In this section will be described metrics scrapping solutions for the monitored components, conversion and graphical representation of the collected metrics and some of the defined alerts.

Prometheus, Alert Manager, LogAlerts and most of the exporters are running on single machine^a. Same machine runs Gitlab installation which is bundled own monitoring solution based on Prometheus and Grafana. Prometheus, Alert Manager, Node Exporter and Grafana from Gitlab were disabled in favour of the ones part of the centralized monitoring solution and theirs functionality was replicated. The additional panel plugins described in section 3.4 were installed in Grafana.

Most of the stand-alone exporters are concentrated on the Monitoring Node. With very few exceptions one exporter instance is required for each monitored service making the number of running exporters significant. Each exporter occupies different port to serve the Prometheus metrics. The large number of used ports may be difficult to manage and may introduce conflicts between the services disrupting their functionality. This problem was mitigated by allocation various IPs from the loopback network together with range of port numbers. Such approach is graphically represented on the next Fig. 8.

^a Will be referenced as “Monitoring Node”

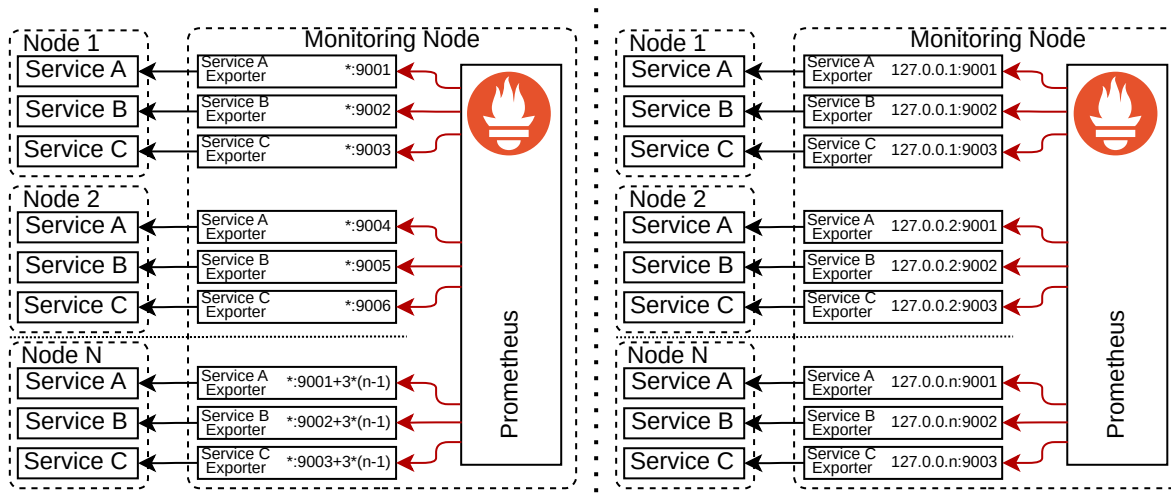


Figure 8: Choosing ports for exporting metrics without (left) and with (right) using various IPs from loopback network

Additionally renaming the instances in Prometheus configuration would be much simpler. The Prometheus configuration fragment below makes label “instance” correspondent to the nodes where each service is running for the hypothetical set-up shown on right side of Fig. 8:

```

- job_name: 'ServiceA'
  static_configs:
  - targets:
    - '127.0.0.001:9001'
    - '127.0.0.002:9001'
  - '127.0.0.00n:9001'
  relabel_configs:
  - target_label: instance
    source_labels: [address]
    regex: '127\.0\.0\.(\d+):9001'
    replacement: 'Nodes${1}'
- job_name: 'ServiceB'
  static_configs:
  - targets:
    - '127.0.0.001:9002'
    - '127.0.0.002:9002'
  - '127.0.0.00n:9002'
  relabel_configs:
  - target_label: instance
    source_labels: [address]
    regex: '127\.0\.0\.(\d+):9002'
    replacement: 'Nodes${1}'
- job_name: 'ServiceC'
  static_configs:
  - targets:
    - '127.0.0.001:9003'
    - '127.0.0.002:9003'
  - '127.0.0.00n:9003'
  relabel_configs:
  - target_label: instance
    source_labels: [address]
    regex: '127\.0\.0\.(\d+):9003'
    replacement: 'Nodes${1}'

```

4.2.1 Monitoring HDP clusters

Two HDP clusters – versions 2 and 3 are monitored. Metrics from both clusters are scrapped by the modified Hadoop Exporter described on page 10. Single instance of the exporter is used to export metrics from all services from both clusters and is running on the Monitoring Node. In single configuration file are listed all monitored services with their type, cluster and node which includes the cluster name and JMX location with simple text format:

```

SERVICE_TYPE      HDPx%NODEn  http://n.n.n.n:port/jmx
...

```

The monitored services are: HDFS (Name, Data and Journal Nodes), YARN (Resource and Node Managers), Hive (Server and LLAPDaemon), HBase (Master and Region Servers) and MapReduce (Job History Server).

Metrics exported by scrapped by Prometheus. Special care was taken in Prometheus configuration to define additional labels in the metrics to distinguish the clusters and the nodes.

Some of the common metrics are suitable for displaying in tables. Fig. 9 shows example of tables of common metrics Up Time, CPU usage, GC Load^a, Heap Usage. Colours of the cells are chosen to draw attention to metrics with non-optimal values.

^a GC Load is a measure for the fitness of memory management; Defined as portion of time or computing power used for GC; Values above 10% may be symptom for memory allocation problems and poor performance.

Service	Instance	Uptime	CPU	GCLoad	HeapUse
yarn nodemanager	NADN006	2.0 week	0.1	1.8	97.7%
hdfs namenode	NANN001	2.3 day	1.2	0.6	88.5%
yarn nodemanager	NADN005	2.1 week	0.1	1.5	83.5%
hdfs namenode	NANN002	11.6 hour	1.2	1.0	75.6%
yarn nodemanager	NADN004	2.5 week	0.1	2.3	73.6%
yarn nodemanager	NADN003	2.1 week	0.1	2.5	72.8%

Service	Instance	Uptime	CPU	GCLoad	HeapUse
yarn nodemanager	NADN203	1.4 day	0.2	1.5	41.2%
yarn nodemanager	NADN201	20.0 hour	0.1	1.3	50.7%
yarn nodemanager	NADN202	1.4 day	0.1	1.0	79.5%
hdfs namenode	INEC210	2.2 week	0.0	0	62.1%
hdfs datanode	NADN202	2.2 week	0.0	0	21.1%
hive llapdaemon	INEC210	1.1 week	0.0	0	70.1%

Figure 9: Common metrics for some services from both HDP clusters

Snapshot of HDFS usage and temporal storage metrics is shown of Fig. 10 as example of specific metric graphical representation.

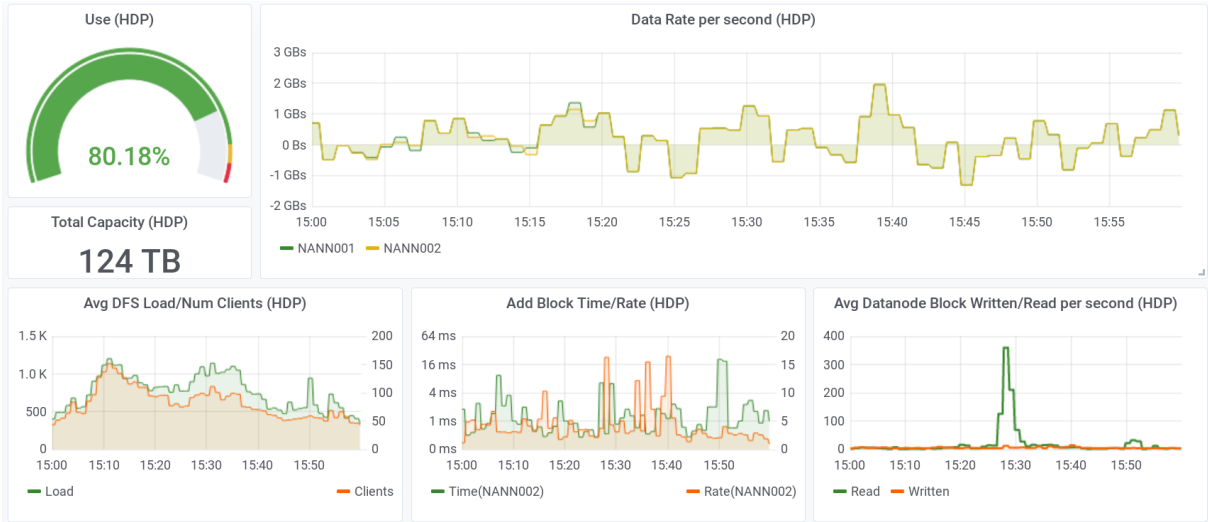


Figure 10: Panel displaying HDFS usage and some temporal data storage rate and metrics

Hadoop exporter has very low resource requirements despite large set of managed metrics. Fig. 11 has graphical representation of CPU and memory usage and scrapping delays for period of about 30 minutes.

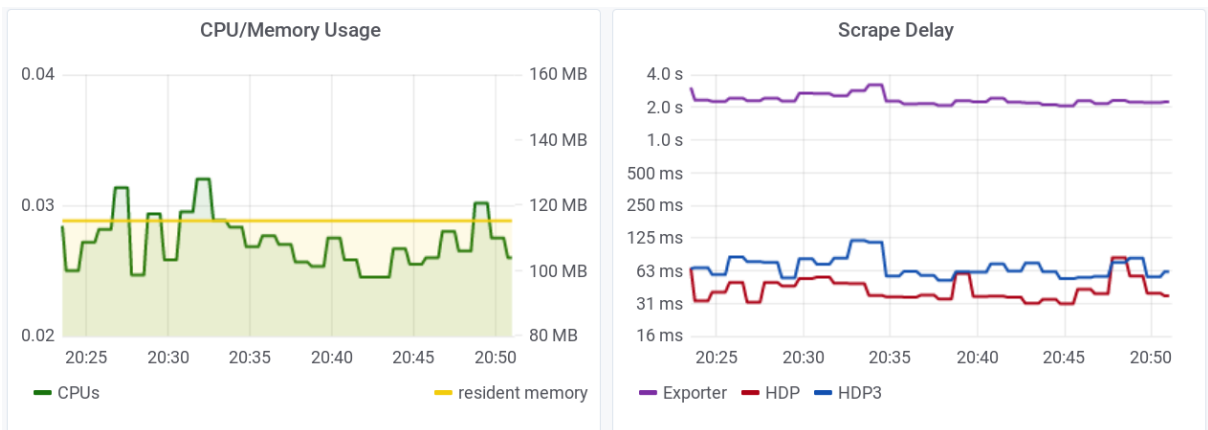


Figure 11: Internal Hadoop Exporter metrics: CPU and memory usage, left and scrape delays, right

4.2.2 Monitoring Database services

Two Presto servers associated to each HDP cluster are monitored. Metrics are collected remotely by two instances of Presto Exporter installed on the Monitoring Node. The metrics exported to Prometheus are limited in number but enough for monitoring of status and performance of the server. Fig. 12 shows panels with some of the collected metrics from one Presto server.

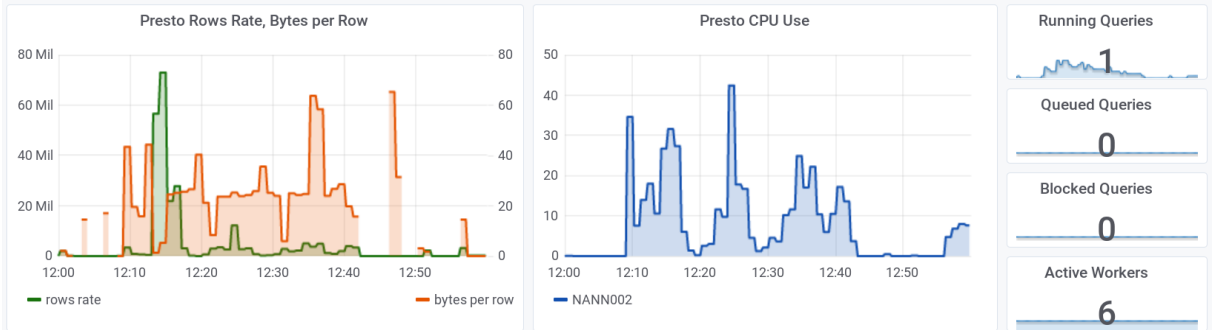


Figure 12: Metrics from Presto server associated with HDP2

Two separated Redis servers are monitored together with another one part of Gitlab installation. All three exporters are running on the Monitoring Machine. On the Fig. 13 below some of the metrics collected from one of the Redis servers are represented.



Figure 13: Fragment of a dashboard with Redis metrics

Four separated PostgreSQL servers are monitored remotely by PostgreSQL Exporter instances all running on Monitoring Machine. On Fig. 14 are shown some basic metrics collected from one of the servers.

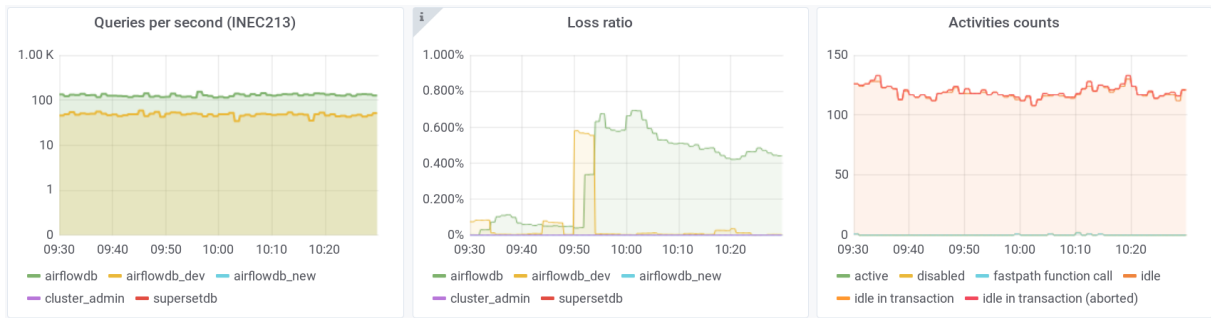


Figure 14: Some PostgreSQL Metrics

4.2.3 Airflow Cluster and Celery

Airflow cluster has 3 nodes. The exporter [46] is installed in the Airflow environment and the metrics are collected from one of the nodes. The metrics are limited – only DAG and Task last run statuses and Time Running are available but with more sophisticated queries much more performance information can be obtained. Fig. 15 shows example for temporal performance metrics and status overview for last 24 hours.

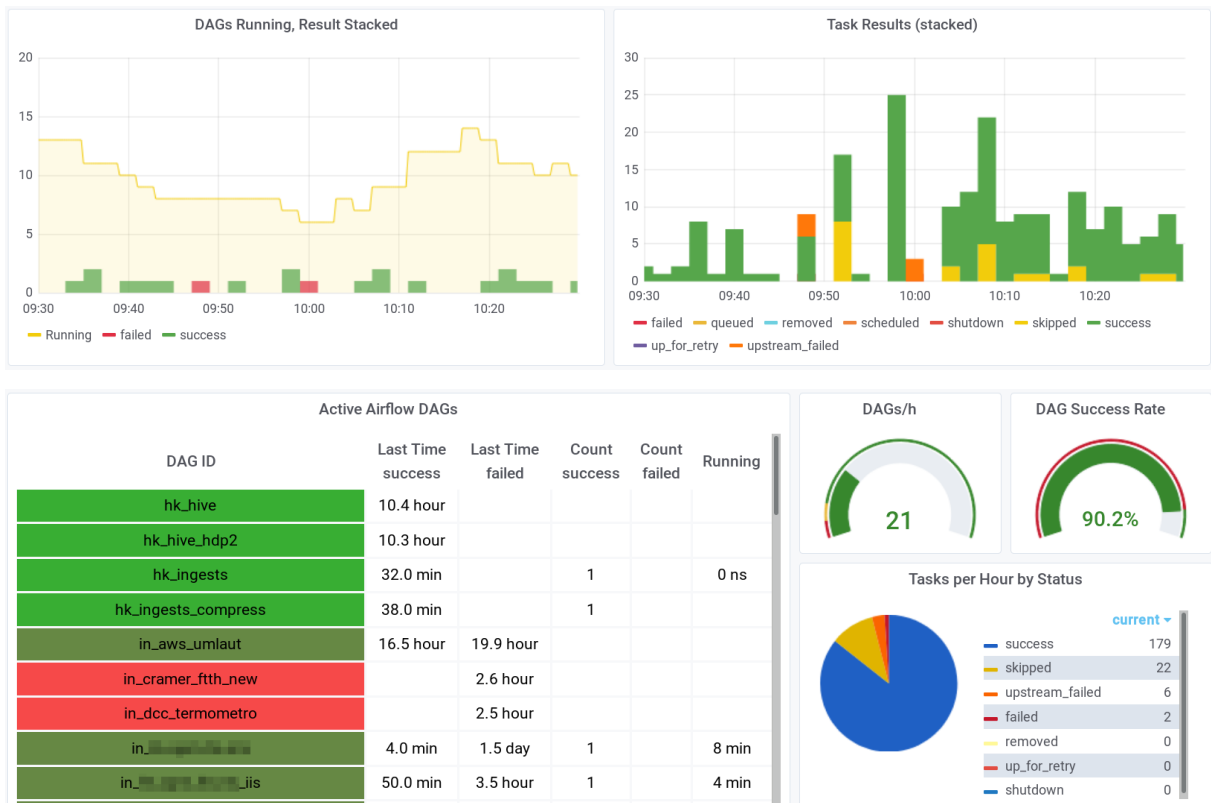


Figure 15: Airflow temporal DAGs and Tasks Performance (up) and status overview (down)

There are two instances of Celery configured with backed Redis. Monitoring is done by two instances of Celery Exporter installed on the Monitoring Node and configured to monitor the corresponding Redis database. Metrics are limited but enough to monitor the service. Fig. 16 shows snapshot of the simple Celery dashboard.

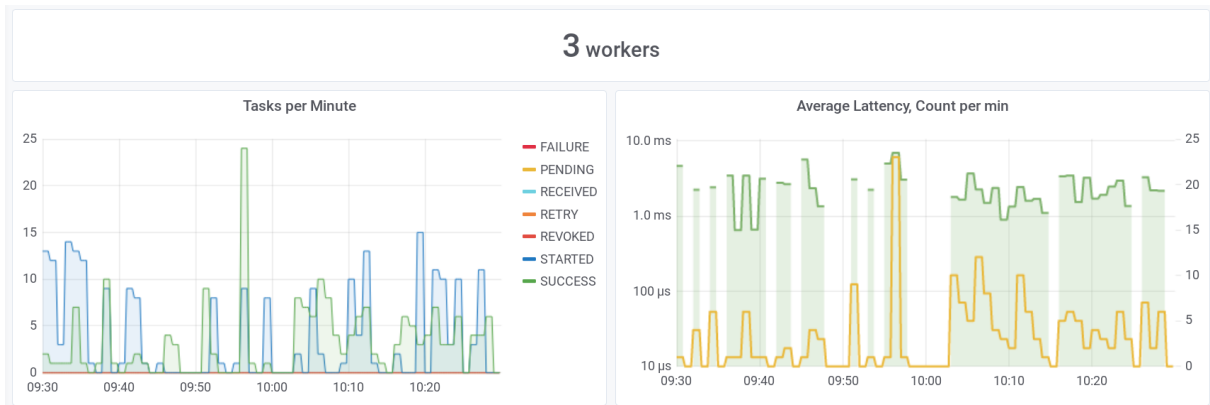


Figure 16: Celery dashboard example

4.2.4 Monitoring NiFi cluster

Apache NiFi cluster is monitored by the implemented in Python NiFi Exporter shortly described on p. 11. Because of security concerns Monitoring Node does not have direct access to the NiFi cluster and the exporter was installed on different node and metrics were collected remotely by Prometheus. Next Fig. 17 has panels showing current status of the NiFi cluster together with some performance metrics.

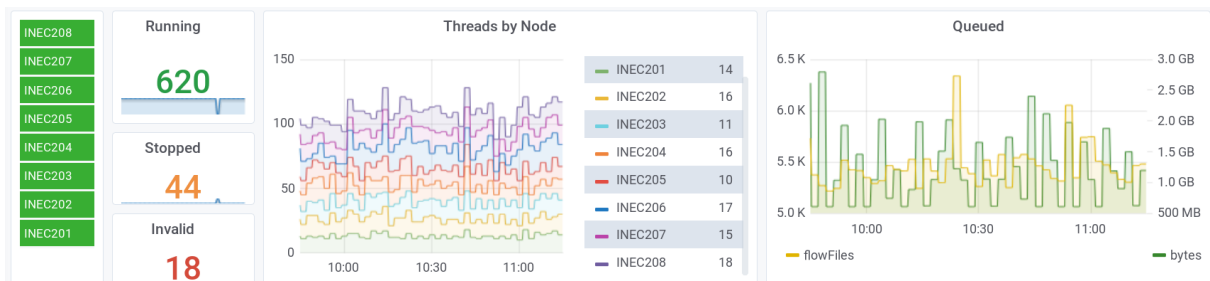


Figure 17: Fragment of a dashboard displaying status and performance of Apache NiFi cluster

4.2.5 Monitoring Kafka clusters

Two separated clusters are monitored. Two instances of the stand-alone Kafka Exporter [44] are running on Monitoring Node. Available metrics permit precise monitoring of the overall performance and individual topics and consumer groups. Fig. 18 has example of dashboard containing table with topics and numbers of messages for the last 24 hours and plots with message rate and lag for selected topics.

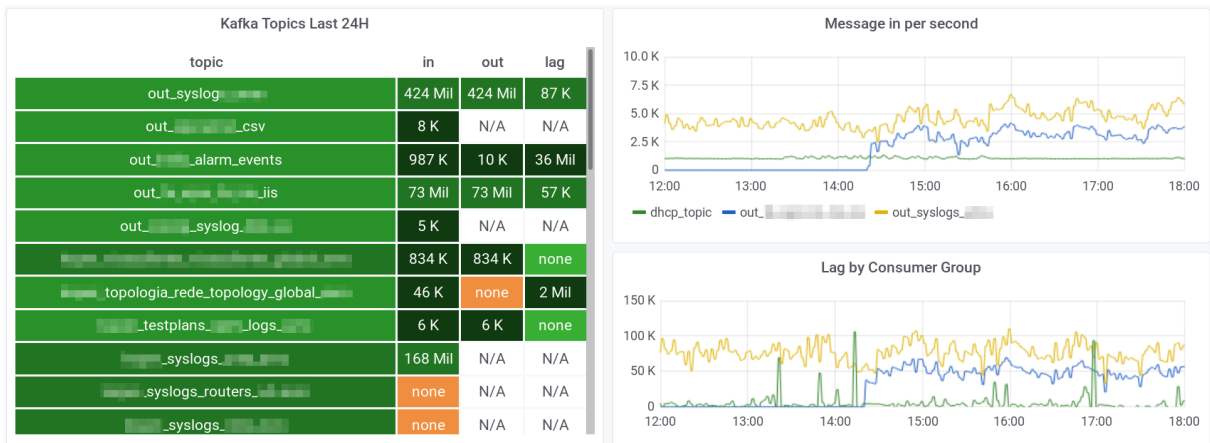


Figure 18: Example of simple dashboard for Kafka

4.2.6 Monitoring services from Elastic stack

In total about 50 Logstash instances distributed over various nodes are monitored. The modified version of Logstash Exporter described on page 11 was used. All associated exporter instances are running on the Monitoring Node. Fig. 19 shows fragment of the table with common

Node	Stream	Name	Uptime	Logstash In rate	Last In	Logstash Out rate	Delay	CPUs use	GC Load
INEC004		collector_004	22.5 week	2 s ⁻¹	now	2 s ⁻¹	none	0.08	0.02%
INEC004	HDFS	streaming_hdfs	1.2 week	3628 s ⁻¹	now	3657 s ⁻¹	4.75 s	0.68	0.45%
offline INEC004			unknown	unknown		unknown	unknown	unknown	unknown
INEC004		_events	22.5 week	none	12 hour	none	none	0.02	none
INEC004	IIS	_iis	1.6 week	none	5 min	none	none	0.02	none
INEC004		syslog_...	25.4 week	3625 s ⁻¹	now	3625 s ⁻¹	4.00 s	0.53	0.26%
INEC004	PortalHDFS	_portal_hdfs	1.6 week	45 s ⁻¹	now	45 s ⁻¹	0.02 s	0.03	0.27%
INEC015	m9606 down		N/A	N/A	>48h	N/A	N/A	N/A	N/A
INEC019		syslog_...	3.2 week	6658 s ⁻¹	now	6658 s ⁻¹	10.37 s	2.93	0.13%

Figure 19: Tabular representation of common metrics from some Logstash instances

metrics and statuses for various Logstash instances including some special situations such as exporter is offline, target Logstash instance cannot be connected. The colour of the third column corresponds to the status of the Logstash instance when available. On Fig. 20 temporal plots of incoming events rate and average delay of the same Logstash instances are presented.

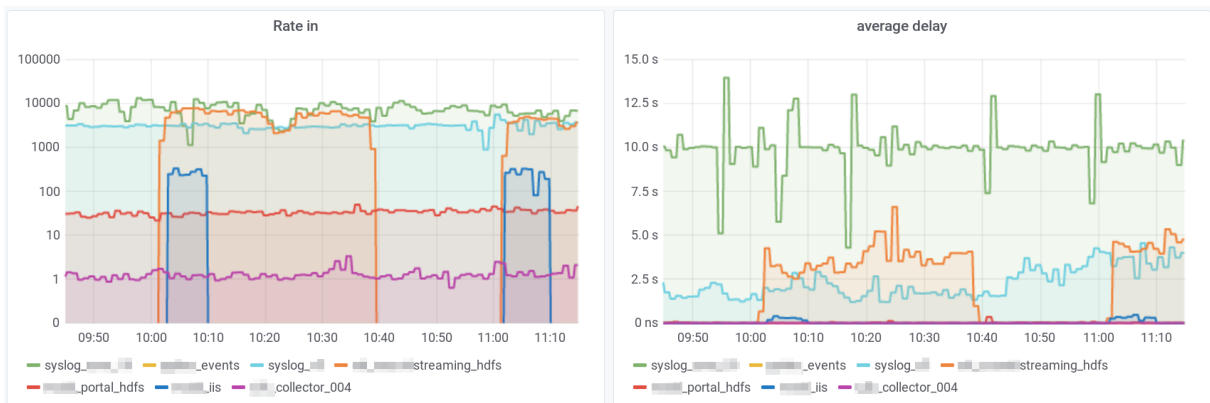


Figure 20: Temporal plots for incoming events rate (left) and average delay (right)

Single instance of Metricbeat is monitored remotely with Beats Exporter [35] installed on Monitoring Machine. On Fig. 21 is shown simple dashboard with the basic metrics collected.

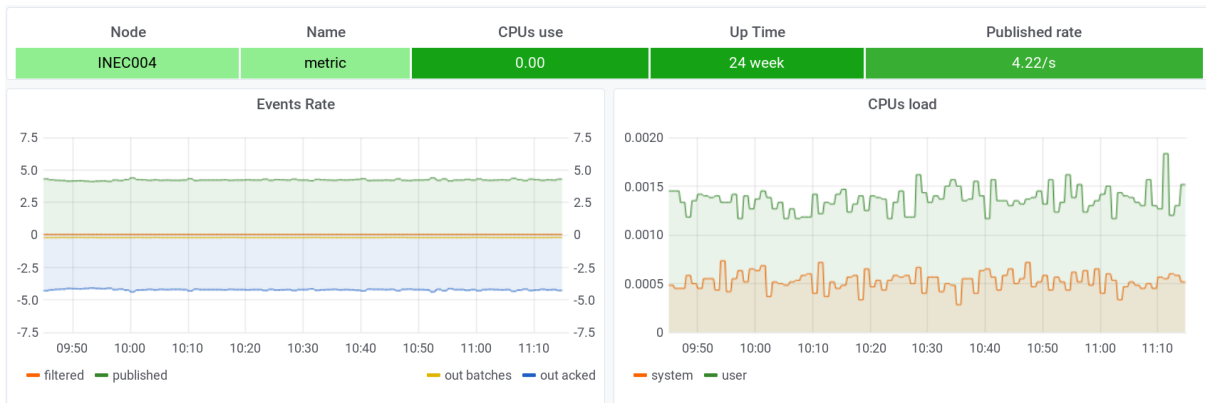


Figure 21: Example for Metricbeat dashboard

Two separated Elasticsearch instances are used and monitored: A single node Elasticsearch instance running on Monitoring Node and a multi-node cluster. Elasticsearch Exporter [33] is used and one instance for each node is used to export metrics. Cluster-wide metrics such as cluster healths, numbers of shards, number of nodes are repeated in every metric set. All Elasticsearch Exporter instances are running on the Monitoring Node with Prometheus metrics served on the same port but with different IPs from the loopback network corresponding on the nodes as described on page 18.

Fig. 22. shows status snapshot of the cluster both for cluster-wide and node specific metrics. Note the cluster-wide table includes status for both monitored Elasticsearch installations.

Cluster Name, Health	Nodes any	Nodes data	Pending Tasks	Max CPU use	Max GC Load	Index Rate	Doc Rate	Query Rate	Avg Query Latency	Max Space use	Primary shards	Active shards	Init shards	Reloc shards	Delay shards	Unass shards
Main	10	6	none	72%	0.85%	22832 s ⁻¹	22124 s ⁻¹	4 s ⁻¹	77.96 μs	71%	1924	3367	0	0	0	0
monitoring-cluster	1	1	none	3%	0.78%	82 s ⁻¹	76 s ⁻¹	1 s ⁻¹	79.37 μs	5%	498	498	0	0	0	199

elasticMain	Role	Uptime	Index rate	docs rate	Query Rate	Query Latency	CPUs use	GC Load	CPU OS load	Disk used
INEC003	Master+Data	3 day	3630 s ⁻¹	3120 s ⁻¹	0.53 s ⁻¹	125 μs	5.40	0.6%	55%	67%
INEC006	Data	5 day	3796 s ⁻¹	4233 s ⁻¹	0.75 s ⁻¹	44 μs	2.45	0.9%	15%	71%
INEC009	Data	3 day	3806 s ⁻¹	3895 s ⁻¹	0.37 s ⁻¹	45 μs	3.98	0.7%	34%	71%
INEC010	Data	2 week	3468 s ⁻¹	3348 s ⁻¹	0.53 s ⁻¹	63 μs	3.80	0.5%	35%	68%
INEC011	Data	2 week	4332 s ⁻¹	3654 s ⁻¹	0.92 s ⁻¹	73 μs	7.31	0.6%	61%	71%
INEC012	Data	3 day	3801 s ⁻¹	3875 s ⁻¹	0.57 s ⁻¹	118 μs	4.53	0.6%	20%	69%
INEC013	Monitoring	2 week	not data	not data	not data	not data	0.11	0.3%	7%	63%
INEC014	Monitoring	3 day	not data	not data	not data	not data	0.13	0.2%	1%	3%
INEC015	Master	23 week	not data	not data	not data	not data	0.10	0.3%	12%	18%
INEC019	Master	19 week	not data	not data	not data	not data	0.01	none	25%	1%

Figure 22: Tables with Elasticsearch cluster-wide (up) and node-specific (down) status metrics

On the next Fig. 23 temporal plots of filesystem transfer rates and document rates and overall cluster health status are displayed.

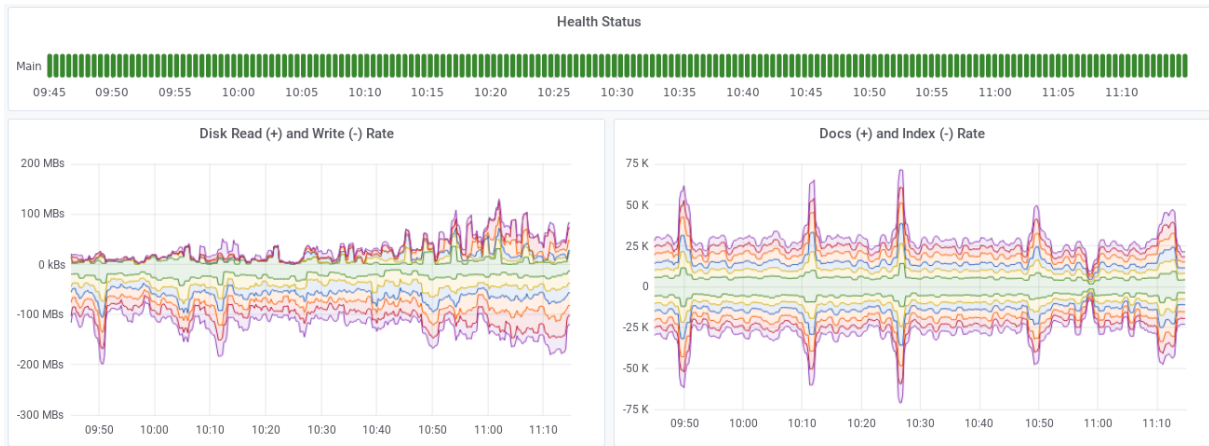


Figure 23: Health status (up) and plots of filesystem transfer rate (left) and documents rate (right)

4.2.7 Monitoring NginX

Several instances NginX servers are used, among them one used in Gitlab installation and another used as main gateway to the internal services. In all cases the NginX VTS module [22] was compiled and used to serve metrics for Prometheus. Metrics include number of requests and latency by result code, virtual server and back-end.

4.2.8 Monitoring the Operating Systems

To monitor the status of the operating system of the nodes Node Exporter is used. It was installed on about 30 nodes, including the ones of HDP2 and ELK clusters. Most of the exporters are used with default configuration, which already gives rich set of metrics including CPU load, memory usage, filesystem status, network usage.

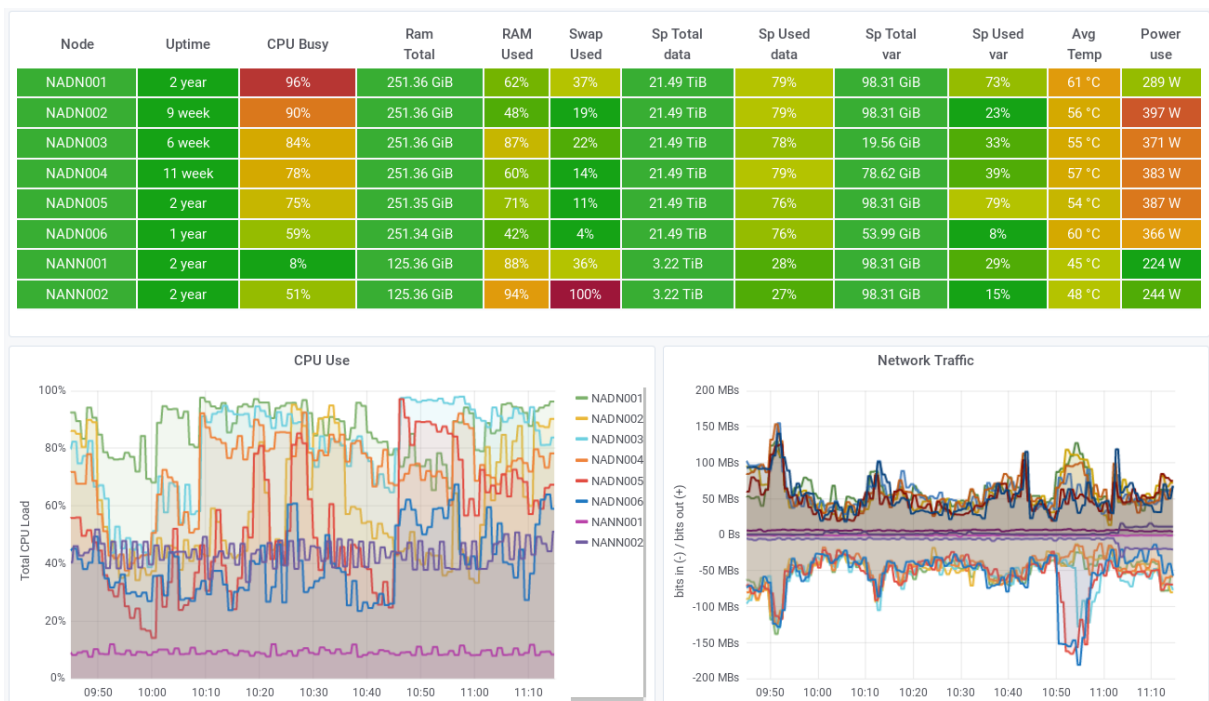


Figure 24: Some metrics from nodes of HDP2 cluster

On the Fig. 24 is given example of tabular representation of some operating system and hardware metrics together with plots of CPU load and network performance.

Using tables with coloured cells with metrics for large numbers of nodes is useful for getting broad overview on the status of the infrastructure but may have too much information transmitting to the viewer and some unusual situation may be ignored. As alternative were created table panels where will be shown only metric values that deserve attention. Such panel with filesystem overuse, CPUs excessive load and and memory overuse is displayed on Fig. 25.

cluster	instance	/	/usr	/var	/opt/commvault
ELK	INEC017			87.3	
ELK	INEC019		89.2		
HDP	NADN001		92.2		
HDP	NADN004				85.4
HDP	NANN001	90.7			

cluster	instance	RAM Use
ELK	INEC016	86.4
ELK	INEC019	93.3
HDP	NADN003	87.4
HDP	NANN001	87.9
HDP	NANN002	94.2

cluster	instance	CPU Use
HDP	NADN001	95.4
HDP	NADN002	88.5
HDP	NADN003	83.1
HDP	NADN004	77.1

Figure 25: Tables with alerts about filesystem , memory and CPU overuse

Among the others optional metric collectors, Node Exporter is capable collecting statuses from start startup managers such as systemd, supervisord and runit. That permits monitoring status of the services without metrics capabilities or with no known exporters.

Metrics exported by those collectors are limited – if the service is running, type of the service and for some – startup time. Fig. 26 has part of a dashboard displaying statuses of some services managed by the monitored startup managers. On the right in the table are services managed by runit from the Gitlab installation where is visible that instances of Alert Manager, Node Exporter and Grafana that are bundled with it are configured but disabled. On the right is temporal status of few services managed by systemd and down various service statuses managed by supervisord on the Monitoring Node.

instance	service	desired	normal	Time
INEC005	alertmanager	down	up	166 day
INEC005	gitaly	up	up	138 day
INEC005	gitlab-monitor	up	up	138 day
INEC005	gitlab-workhorse	up	up	138 day
INEC005	grafana	down	up	179 day
INEC005	logrotate	up	up	9 min
INEC005	nginx	up	up	125 day
INEC005	node-exporter	down	up	37 day
INEC005	postgres-exporter	up	up	138 day
INEC005	postgresql	up	up	138 day

Services Status

LS Exporters	Prometheus	Node Exporter	Reporting NA
57	2 day	5 week	20 week
LS Exporters Down	Alert Manager	Superset HDP... is Down	Reporting ELK
6	5 week		20 week

Figure 26: Service statuses collected by Runit (left), Systemd (top, right) and Supervisord (bottom, right) collectors

4.2.9 Defining alerts and configuring Alert Manager

Alerts are defined in Prometheus configuration. PromQL is used to describe the conditions that will trigger an alert and may be defined delays before the alert to be sent and additional labels and descriptive annotations.

The example below defines alert that is triggered when some filesystem is used more than 85% and forwarding the alert to the configured alert managers is delayed by 2 minutes. The labels of the

alert is inherited from the metrics used in the query and new label “severity” is added. Two annotations are included which texts are generated from some of the labels.

```
- alert: PartitionFull
  expr: 1-node_filesystem_avail_bytes/node_filesystem_size_bytes>0.85
  for: 2m
  labels:
    severity: warning
  annotations:
    summary: 'Home dir at {{$labels.instance}} from {{$labels.cluster}} is overused'
    description: 'Filesystem mounted at {{$labels.mountpoint}} at {{$labels.instance}} from
  {{$labels.cluster}} is more than 85% used'
```

Multiple alerts were defined with broad sources and conditions and some groups are defined as well 4 levels of severity. Some of them are:

- Alerts from Elasticsearch cluster:
 - disk space used for data is more than 92% used (warning);
 - a node from ELK cluster is unreachable (error);
 - a node is reporting unhealthy status (warning);
 - queries are delayed by more than 5 seconds (warning).
- Alerts from the operating systems:
 - A node has rebooted (info);
 - A partition is more than 85% full (warning);
 - A partition is more than 98% full (error).
- Logstash streams:
 - A Logstash instance has stopped receiving events (warning);
 - A Logstash instance has not received any event for some period (warning).
- Alerts from Redis servers:
 - Cache misses are more than 80% (warning);
 - Redis is rejecting client connections (warning).
- Alerts from Airflow:
 - Airflow is down or unreachable (critical);
 - No DAGs were run for more than 5 minutes.

While Prometheus defines the alerts and sends them on the configuration of the Alert Manager depends how the alerts are forwarded. Various “receivers” were defined forward alerts to different e-mail addresses depending of the alerts grouping. The content of the e-mails are generated from the labels and annotations defined in the alerts. The configuration of the e-mail receivers permits also modifying the headers of the e-mails including “From:” header that is used to identify the alert source.

The LogAlerts program was also installed and configured to work together with Alert Manager it is used to keep history of past and resolved alerts and also as a debugging tool when defining the alerts in Prometheus and configuring the Alert Manager. On the next Fig. 27 screen shot of the web interface of LogAlerts used in Production is shown.

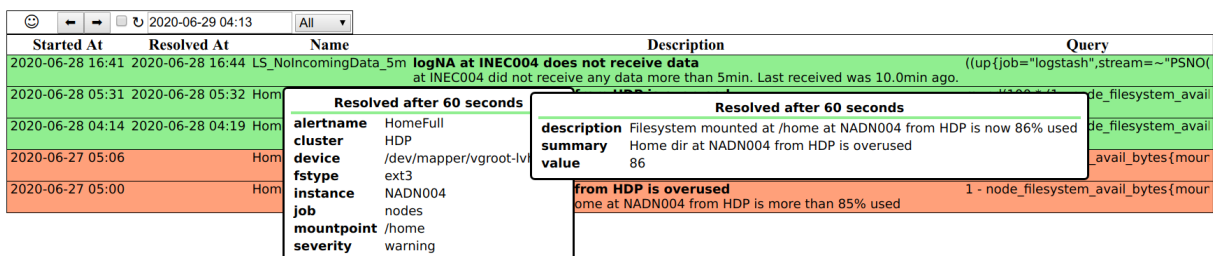


Figure 27: Screen shot of the web interface of LogAlerts program displaying some active and resolved alerts together with alert labels and annotations.

4.2.10 Prometheus self-monitoring

Prometheus exports and was configured to store own extensive metrics. They include CPU and memory usage, number of collected time series, query engine state, database status and size.

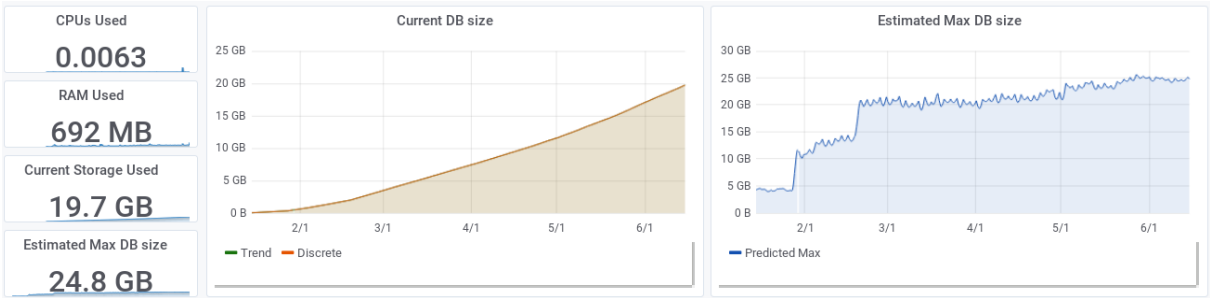


Figure 28: Panels with metrics from Prometheus and plot with current and estimated maximum TSDB size

Prometheus automatically deletes older than some pre-configured period, here equal to 6 months. The database size metric helps to estimate disk space that would be required at the end of the period, assuming no metrics are added or removed, by extrapolating the data size over some reasonable period to the time when old metrics start to be deleted. On the Fig. 28 are displayed current CPU, memory and TSDB disk space usage together with plots of current disk usage and estimated maximum disk space required for the TSDB for a period of 6 months. The steps correspond to the moment when large sets of metrics were included.

Prometheus has special internal metrics for current pending or active alerts. This allows creating panels to display list of current alerts interdependently of Alert Manager. This functionality does not replace the Alert Manager or LogAlerts program. On Fig. 29 is displayed snapshot of a table panel listing two active and two pending alerts of various types.

Started	Active	AlertName	Instance	Extra	Severity
2020-07-16 02:15:13	3 min	PartitionFull	INEC003	/var/crash	error
2020-07-16 02:15:13	3 min	NodeRebooted	INEC003	-	info
2020-07-16 02:12:44	5 min	LS_NoIncomingData_Always	INEC004	syslog_	warning
2020-07-16 02:17:44	16 s	LS_NoIncomingData_5m	INEC004	_events_staging_raw	warning

Figure 29: Panel displaying various active (red) and pending (yellow) alerts

4.2.11 Structure of the Monitoring solution

The graphical representation of the full Monitoring Solution is presented on the Fig. 30 on the next page 30. In the centre is the Monitoring Node with Prometheus, Grafana, Alert Manager and LogAlerts together with most of the exporters. On the top are the ELK cluster and the two HDP clusters. In the bottom are Airflow cluster and both Kafka clusters. All Node Exporters are represented separately, on the bottom left.

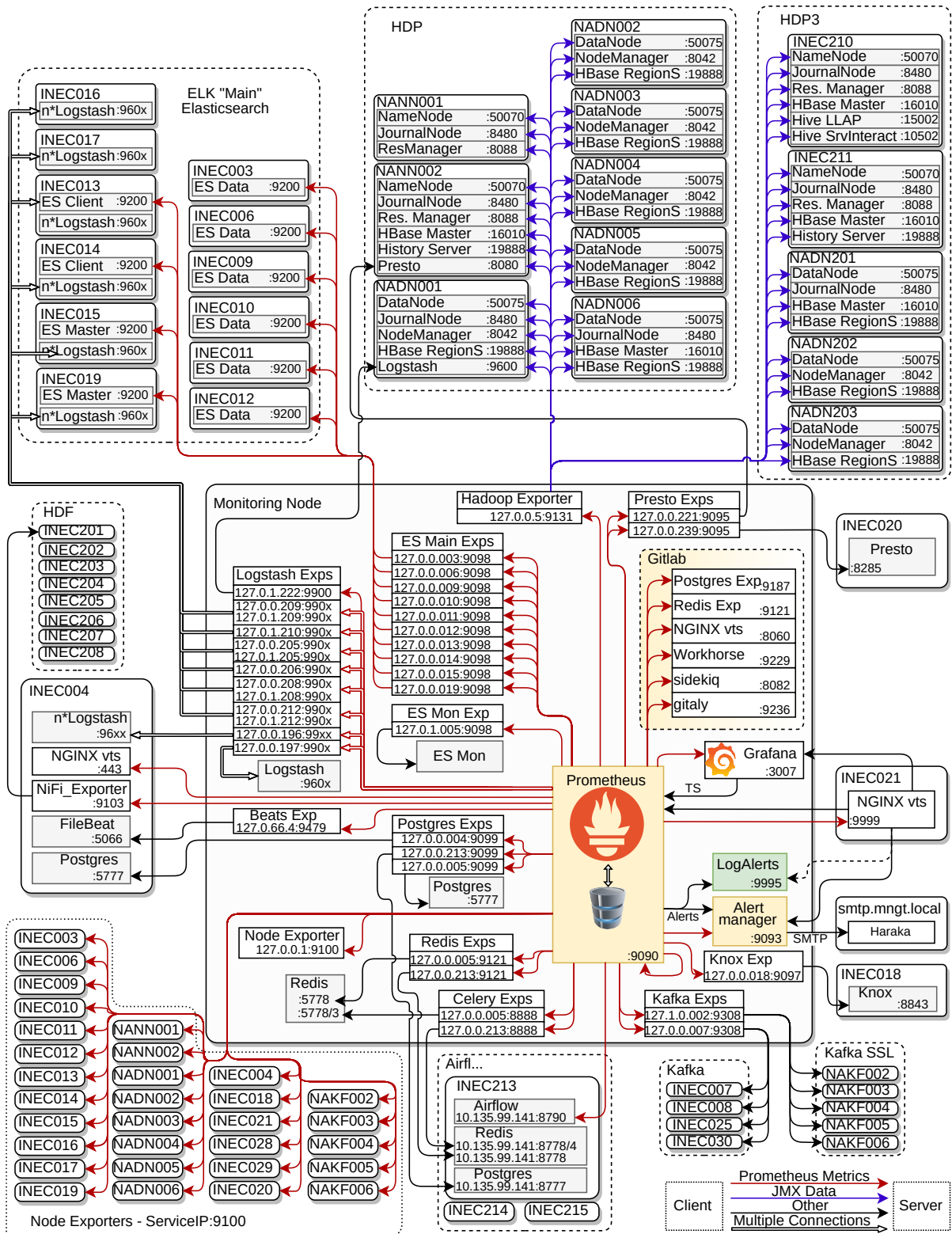


Figure 30: Final structure of the Centralised Monitoring solution for Production

4.3 COMPUTATIONAL CHEMISTRY SPECIALIZED CLUSTER

With the experience acquired developing the previous two centralised monitoring solutions similar solution for specialised cluster was developed and implemented as a voluntary project which will be shortly described in this section.

The cluster consists of about 80 computing nodes from various generations, one file server and one Command and Control server^a. Computing nodes have the same operating systems installed and the software is equivalent with few exceptions. The job queueing system is implemented with SLURM^b and the nodes are grouped in several partitions by hardware configuration.

4.3.1 Additional Exporters

From the exporters discussed in 3.2 only Node Exporter is used. To monitor the specific services and hardware additional exporters were evaluated and installed.

IPMI Exporter [53] is a program developed in Go. It relies in on external binaries from FreeIPMI^c package to collect hardware status information remotely. Single instance of the exporter is capable to collect metrics from all the nodes in the cluster. Metrics include hardware component temperatures, cooling fans performance, power consumption. IPMI Exporter does not follow the recommendations for creating exporters because it starts external programs to collect the metrics.

Smartctl Exporter [54] uses smartctl program from S.M.A.R.T. Monitoring Tools to collect metrics from hard disks and export them to Prometheus. Similarly to IPMI Exporter Smartctl exporter executes external program to collect the metrics. The exporter together with recent version of smartctl need to be installed and run on all monitored nodes and Prometheus collects the metrics remotely. Metrics include internal temperature, age, restart and reload counts, number of relocated sectors and others that help detecting pre-failing status to replace failing hard disks before loosing data or disrupting the while cluster in case of catastrophic failure.

SLURM Exporter [55] is used to collect metrics from the queueing system. Metrics are allocation statuses of CPUs and nodes, number of jobs by status and scheduler information.

Apcupsd Exporter [56] is a stand-alone Go program queries apcups daemon to collect metrics about status of UPSs made by APC . Metrics include battery charge, power line status, load, estimated available time and ambient temperature.

Some of the nodes have installed GPUs produced by NVIDIA. Node Exporter does not collect status of GPUs and for that purpose special NVIDIA Exporter [57] is used. It is a stand-alone program that uses NVIDIA Management Library to access the devices and is installed on the computing nodes. It does not execute any external binary to collect metrics.

^a Will be referenced as CC

^b Simple Linux Utility for Resource Management, [Slurm Documentation](#)

^c Tools and library to access IPMI, <https://www.gnu.org/software/freeipmi/>

4.3.2 Building the monitoring solution

Prometheus is installed on the File Server to have direct access to the main storage for its TSDB. Grafana is installed and running on CC. Because of security consideration connections from and to the cluster are restricted – only CC can be accessed from Internet and only by SSH and so visualising dashboards from Grafana and accessing Prometheus web console is possible through SSH tunnel only.

Alerts were not defined and Alert Manager was not installed.

Node Exporter is installed on all nodes and the two servers, running the default configuration and is managed by Systemd.

Smartctl Exporter and recent smartctl executable is installed on all nodes and CC. Configuration files that include list of monitored hard disks are specific for each node. Scrapping interval in Prometheus was chosen to be 4 minutes – larger than the generic one of 60 seconds.

On Fig. 31 is shown table combining some metrics from Node Exporter and Smartctl Exporter. Only small portion of the nodes is included. Selected are examples for fully and partially loaded, idle and offline nodes from the different partitions. The right 4 columns are smartctl metrics.

Node	Uptime	CPU Busy	Average Temp	Ram Total	RAM Used	Swap Used	Total root	Used root	Total tmp	Used tmp	A 5 Max	A 197 Max	A 187 Max	Temp Avg
CC	12 week	0%	36 °C	7.70 GiB	13%	1%	146.29 GiB	4%	635.99 GiB	0%	0	0		38 °C
FileServer	12 week	1%	30 °C	92.88 GiB	4%	1%	19.10 GiB	15%	9.49 GiB	6%				
node01	39 week	100%	57 °C	3.92 GiB	14%	24%	3.81 GiB	63%	224.13 GiB	1%	0	0		39 °C
node03	39 week	0%	31 °C	3.92 GiB	11%	19%	3.81 GiB	63%	224.13 GiB	1%	0	0		35 °C
node08														
node71	43 week	67%	48 °C	35.35 GiB	14%	3%	3.87 GiB	83%	2.67 TiB	1%	14	0	0	22 °C
node72	43 week	100%	51 °C	35.35 GiB	15%	4%	3.87 GiB	62%	2.67 TiB	1%	90	0	13	22 °C
nodebig01	14 week	75%	63 °C	93.04 GiB	28%	0%	39.37 GiB	11%	839.83 GiB	3%	0	0	0	23 °C
nodegpu01	12 week	100%	53 °C	62.81 GiB	9%	0%	3.87 GiB	70%	429.90 GiB	2%	0	0	0	22 °C
nodegpu02	39 week	100%	51 °C	62.81 GiB	6%	0%	3.87 GiB	70%	429.90 GiB	2%	0	0	0	23 °C

Figure 31: Snapshot of table panel with some of the operating system and generalised smartctl (the last 4 columns) metrics

Single instance of IPMI Exporter and freeipmi executables are installed on the FileServer and it collects metrics from all nodes. On Fig 32 below is displayed table with some of the collected hardware metrics.

node	power	Ambient	CPU	rpm	sens warn	sens crit	power warn	power crit	fan warn	fan crit	temp warn	temp crit
CC		33 °C		9272	✓	✓			✓	✓	✓	✓
FileServer	184 W		31 °C	4800	✓	✓	✓	✓	✓	✓	✓	✓
node01		32 °C		9075	✓	✓			✓	✓	✓	✓
node03		30 °C		8784	✓	✓			✓	✓	✓	✓
node08					✓	✓						
node71	240 W	18 °C	51 °C	7550	✓	✓	✓	✓	✓	✓	✓	✓
node72	276 W	19 °C	56 °C	8000	✓	✓	✓	✓	✓	✓	✓	✓
nodebig01	250 W		67 °C	12250	✓	✓	✓	✓	✓	✓	✓	✓
nodegpu01	448 W		57 °C	13989	✓	1	✓	✓	✓	✓	✓	✓
nodegpu02	466 W		55 °C	16278	✓	1	✓	✓	✓	✓	✓	✓

Figure 32: IPMI metrics and generalised sensor status

NVIDIA Exporter was installed on the nodes with GPUs.

The cluster is powered by UPS which has network interface used by the own service to control the nodes in response of power failures. Same service is monitored by Apcupsd Exporter to collect and export metrics about the UPS status. Apcupsd Exporter is installed and running on the FileServer. Fig. 33 shows plots of some of the metrics for a period of 15 days.



Figure 33: Plots of some metrics from Apcupsd Exporter

Monitoring the running jobs and status of the queueing system is done by Slurm Exporter. It is stand-alone program installed and running on the File Server.

A cockpit-like dashboard shown on Fig. 34 was created to give overview on the whole cluster status - number of CPUs and nodes by allocation status reported by Slurm Exporter. As cooling system is of concern environment temperature reported by the UPS is included together with some averaged temperatures reported by Node, IPMI and Smartctl Exporters. Below are portions of CPU load by partition as reported by Node Exporter.

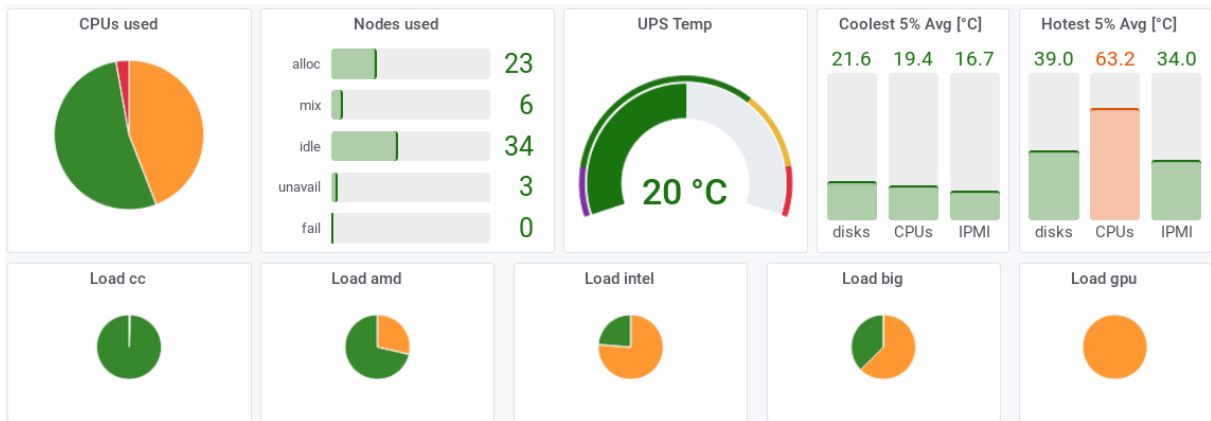


Figure 34: Dashboard with status overview of the Cluster

The diagram on Fig. 35 shows the structure of the monitoring solution.

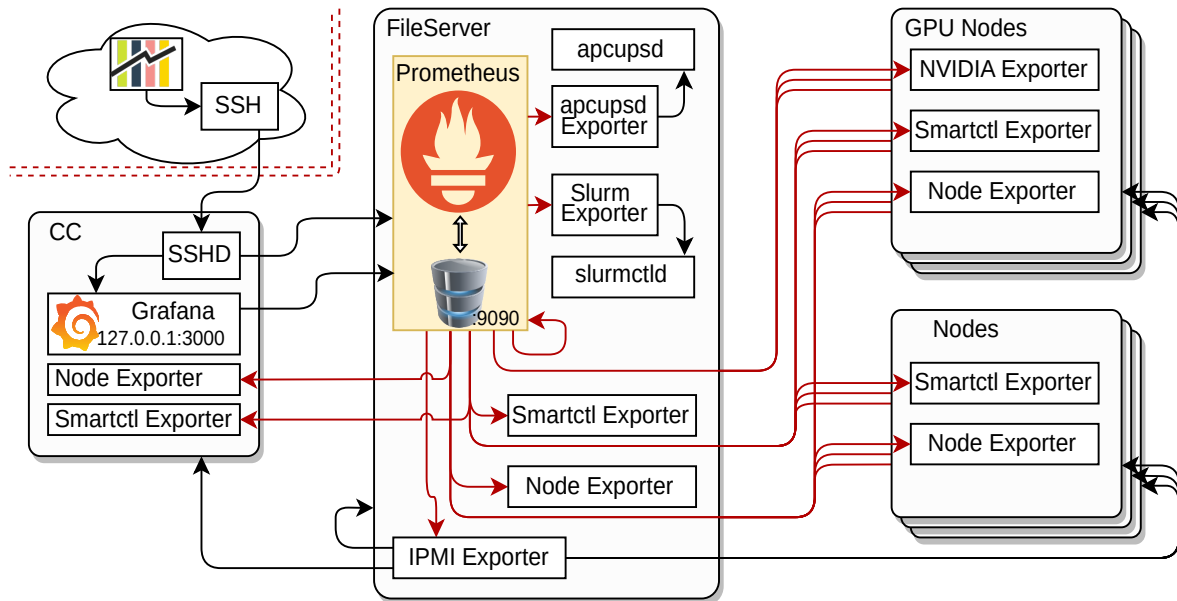


Figure 35: Structure of the monitoring solution for the Cluster

5 CONCLUSIONS

For this work extensive research for existing centralized monitoring solutions was done and various approaches and software packages for designing and building monitoring solutions were studied. Evaluated were complexity, versatility, cost of development and support, hardware and infrastructure requirements. Three major approaches – InfluxDB, Elasticsearch and Prometheus based – were compared further. Considering many advantages of Prometheus based solution it was selected for development and implementation.

To implement monitoring infrastructure around Prometheus large number of metric exporters were evaluated, some were modified and enhanced to fulfil better the requirements. Existing Hadoop Exporter was extensively enhanced to improve performance and add new features. Performance and behaviour of the monitored services were studied to define alerts and configure the Alert Manager. Additionally companion program LogAlerts was developed to store and display history of active and resolved alerts. It has own web interface and is used as debugging tool for defining alerts and alert routes.

Complex centralised monitoring and alerting solution was developed and implemented covering most of the services used in the Department and multiple dashboards for visualising statuses and metrics were developed. Based on the experience earned, another monitoring solution for infrastructure with different architecture and purpose was developed and implemented as a voluntary project.

6 SUPPLEMENTARY MATERIAL

`boomtable-sortbyrow-r260.patch`: Modifications done to Boom Table plugin for Grafana described on page 15. Solves sorting misbehaviour of rendered tables.

`elasticsearch_exporter.patch`: Modification to Elasticsearch Exporter to add Uptime metric.

`logstash_exporter.patch`: Patch for Logstash Exporter described on page 11. Adds Uptime, Status and Name metrics.

`NiFi_Exporter.patch`: Modification to NiFi Exporter to enable scrapping metrics without authentication.

`Testing.json`: Combined portable dashboard for Grafana that demonstrates part of the metrics and queries included in this report

`AirflowCeleryDashboard.json`: Combined Dashboard for Airflow and Celery similar to Figs. 22 and 23.

`logalerts.zip`: LogAlerts program described in section 3.3. It is also published at <https://gitlab.com/y1b/logalerts>.

`hadoopexporter.zip`: Modified version of Hadoop Exporter [30] enhanced for speed, multitasking and features as described on page 10. Modifications are tracked at https://gitlab.com/y1b/hadoop_exporter.

7 LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

Even when designed to be resilient a complex monitoring infrastructure need to be constantly supported and developed following (or anticipating) the changes and growth of the monitored services. Open Source nature of Prometheus ecosystem promotes development of new exporters and enhancements of the existing and implementing integrated Prometheus metrics in the services.

The described in this report solutions may benefit of further development to include more services in the monitoring pool. For example the “textfile” collector from Node Exporter may be used to collect metrics from services that do not have monitoring endpoints but are capable creating logs and files with metrics.

Some of the used exporters may need improvements or even re-implementations. A notable example is IPMI Exporter which calls external executables to collect metrics. It is against the recommendations and may introduce significant system load. Alternative IPMI Exporter may be developed based on “freeipmi” library and respective bindings.

Alerting logic are routes must be carefully designed. Approaches to send everything to everybody is contra-productive for generating unnecessary noise and causing the alerts being ignored.

Single Prometheus instance solution is not redundant – in case of failure of Prometheus it self or the machine where it is running no alerts will be generated and it may remain unnoticed. To mitigate such problems federated Prometheus instances on different machine may be configured. Federating may be also used for combined monitoring of multiple separated informational infrastructures.

8 REFERENCES

- [1] Mike Julian, *Practical Monitoring*, O'Reilly, 2018
- [2] James Turnbull, *The Art of Monitoring*, Turnbull Press, 2016
- [3] Hassan Jamil Syed, Abdullah Gani, Raja Wasim Ahmad, Muhammad Khurram Khan and Abdelmuttlib Ibrahim Abdalla Ahmed, *CloudMonitoring: A Review, Taxonomy, and Open Research Issues*, Journal of Network and Computer Applications
- [4] He Huang and Liqiang Wang, *P&P: a Combined Push-Pull Model for Resource Monitoring in Cloud Computing Environment*, in: Cloud Computing(CLOUD), 3rd International Conference on Cloud Computing, IEEE, 2010
- [5] James Turnbull, *Monitoring With Prometheus*, Turnbull Press, 2018
- [6] Apache Software Foundation, *Apache Ambari*, 2019, <https://ambari.apache.org>
- [7] Cloudera, *Hortonworks Data Platform (HDP®)*, <https://www.cloudera.com/downloads/hdp.html>
- [8] Elastic, *Elastic Stack*, <https://www.elastic.co/elastic-stack>
- [9] Amazon, AWS, *Open Distro for Elasticsearch*, <https://opendistro.github.io/for-elasticsearch/>
- [10] Elastic, *Beats: Data Shippers for Elasticsearch*, <https://www.elastic.co/beats/>
- [11] InfluxData, *Introduction to the InfluxData platform*, <https://docs.influxdata.com/platform/>
- [12] Go, <https://golang.org>
- [13] Influxdata, *Telegraf 1.15 documentation*, <https://docs.influxdata.com/telegraf/>
- [14] David G. Simmons , *How to Write a Telegraf Plugin for Beginners*, 2017, <https://www.influxdata.com/blog/how-to-write-telegraf-plugin-beginners/>
- [15] Influxdata, *InfluxDB 1.8 documentation*, <https://docs.influxdata.com/influxdb/v1.8/>
- [16] Influxdata, *Chronograf 1.8 documentation*, <https://docs.influxdata.com/chronograf/v1.8/>
- [17] Influxdata, *Kapacitor 1.5 documentation*, <https://docs.influxdata.com/kapacitor/v1.5/>
- [18] Cloud Native Computing Foundation, *Prometheus*, <https://prometheus.io/>, <https://prometheus.io/docs/introduction/overview/>
- [19] *PromQL: Prometheus Query Language*, <https://prometheus.io/docs/prometheus/latest/querying/basics/>
- [20] Brian Brazil, *Prometheus: Up & Running*, O'Reilly, 2018
- [21] *Openmetrics*, <https://github.com/OpenObservability/OpenMetrics>, <https://openmetrics.io>
- [22] YoungJoo Kim (김영주), *Nginx virtual host traffic status module*, <https://github.com/vozt/nginx-module-vts>
- [23] *Exporters and integrations | Prometheus*, <https://prometheus.io/docs/instrumenting/exporters/>
- [24] Grafana Labs, *Grafana*, <https://grafana.com/grafana/>
- [25] Grafana Labs, *Prometheus data source*, <https://grafana.com/docs/grafana/latest/features/datasources/prometheus/>
- [26] *Download | Prometheus*, <https://prometheus.io/download/>
- [27] *Notification template reference | Prometheus*, <https://prometheus.io/docs/alerting/latest/notifications/>
- [28] Prometheus, *Exporter for machine metrics*, https://github.com/prometheus/node_exporter
- [29] Prometheus, *JMX Exporter*, https://github.com/prometheus/jmx_exporter
- [30] Wu Lixuan (吴立轩), *Hadoop exporter for Prometheus*, https://github.com/cauwulixuan/hadoop_exporter

- [31] Microsoft, *Ambari Based Hadoop Exporter*, <https://github.com/microsoft/AmbariBasedHadoopExporter>
- [32] Mono, <https://wiki.winehq.org/Mono>
- [33] JustWatch, *Elasticsearch stats exporter for Prometheus*, https://github.com/justwatchcom/elasticsearch_exporter
- [34] Christoffer Kylvåg, *Prometheus exporter for Logstash metrics*, https://github.com/BonnierNews/logstash_exporter
- [35] Audrius Karabanovas, *Elastic beat-exporter for Prometheus*, <https://github.com/trustpilot/beat-exporter>
- [36] Mikołaj Siedlarek, *Apache NiFi metrics exporter for Prometheus*, https://github.com/msiedlarek/nifi_exporter
- [37] *NiFi Exporter*, https://github.com/mormor83/NiFi_Exporter
- [38] Matthias Nguyen, *Nifi Prometheus Reporter*, <https://github.com/mkjoerg/nifi-prometheus-reporter>
- [39] Prometheus, *Push acceptor for ephemeral and batch jobs*, <https://github.com/prometheus/pushgateway>
- [40] Hemantha Kumara, *NiFi JMX exporter*, <https://grafana.com/grafana/dashboards/9544>
- [41] Will Rouesnel, *PostgreSQL Server Exporter*, https://github.com/wrouesnel/postgres_exporter
- [42] Yahoo! JAPAN, *Presto Exporter*, https://github.com/yahoojapan/presto_exporter
- [43] *Prometheus Redis Metrics Exporter*, https://github.com/oliver006/redis_exporter
- [44] Daniel (Shijun) Qian, *Kafka exporter for Prometheus*, https://github.com/danielqsj/kafka_exporter
- [45] NGINX Inc., *NGINX Prometheus Exporter*, <https://github.com/nginxinc/nginx-prometheus-exporter>
- [46] *Airflow prometheus exporter*, <https://github.com/epoch8/airflow-exporter>
- [47] Horst Gutmann, *Celery prometheus exporter*, <https://github.com/zerok/celery-prometheus-exporter>
- [48] Sriramajeyam Sugumaran, *Boom Table Panel for Grafana*, <https://github.com/yesoreyeram/yesoreyeram-boomtable-panel>
- [49] *Statusmap panel for Grafana*, <https://grafana.com/grafana/plugins/flant-statusmap-panel>, <https://github.com/flant/grafana-statusmap>
- [50] Grafana Labs, *Pie Chart Panel Plugin*, <https://github.com/grafana/piechart-panel>, <https://grafana.com/grafana/plugins/grafana-piechart-panel>
- [51] Camptocamp SA, *Grafana datasource for Prometheus Alertmanager*, <https://grafana.com/grafana/plugins/camptocamp-prometheus-alertmanager-datasource>, <https://github.com/camptocamp/grafana-prometheus-alertmanager-datasource>
- [52] Bas Harenslak, *Monitoring HBase With Prometheus*, 2017, <https://godatadriven.com/blog/monitoring-hbase-with-prometheus/>
- [53] Soundcloud, *Prometheus IPMI Exporter*, https://github.com/soundcloud/ipmi_exporter
- [54] Maxim Gorlov (Максим Горлов), *Smartctl Exporter*, https://github.com/Sheridan/smartctl_exporter
- [55] Victor Penso, *Prometheus Slurm Exporter*, <https://github.com/vpenso/prometheus-slurm-exporter>
- [56] Matt Layher, *NVIDIA GPU Prometheus Exporter*, https://github.com/mdlayher/apcupsd_exporter
- [57] Rohit Agarwal, *NVIDIA GPU Prometheus Exporter*, https://github.com/mindprince/nvidia_gpu_prometheus_exporter

9 INDEX

Ambari	2, 9, 10, 16	Presto	12, 21	Chronograf	5
Elastic Stack	3, 6	Redis	12, 21	InfluxDB	4
Beats	3	SLURM	31	Kapacitor	5
Elasticsearch	3	Smartctl	31, 32	Telegraf	4
Kibana	3	Grafana		JMX	2, 19
Logstash	3	Alerting	6	NginX	12
Exporters		Installation	14	Open Distro	3, 17
Airflow	13, 22	Plugins	6, 14	PostgreSQL	12
Ambari	10	BoomTable	14	Process Managers	
Apcupsd	31	Pie Chart	15	Runit	27
Beats	11, 25	Statusmap	15	SLURM	31
Celery	13, 22	Hadoop	9, 16	Supervisord	8, 27
Elasticsearch	11, 25	HBase	16, 19	Systemd	8, 27
Hadoop	9, 19	HDFS	16, 19	Prometheus	
IPMI	31	Hive	19	Alert Manager	6, 13, 28
JMX	8, 12, 16	Kafka	17	Alerts	28, 29
Kafka	12, 23	MapReduce	19	Exporter	5
Logstash	11, 24	YARN	19	PromQL	5, 27
NginX VTS	5, 12, 26	Zookeeper	17	Counter	5
NiFi	17, 23	InfluxData		Gauge	5
Node	8, 16, 26, 32	Flux	4	TSDB	29
NVIDIA	31	InfluxQL	4	SQLite3	13
PostgreSQL	12, 17, 21	TICK Stack	4, 6	Supervisord	16