EPJ Web of Conferences **214**, 08016 (2019) https://doi.org/10.1051/epjconf/201921408016
CHEP 2018

Service monitoring system for JINR Tier-1

Ivan Kadochnikov[1], *Vladimir* Korenkov[1], *Valery* Mitsyn[1], *Igor* Pelevanyuk[1,*], and *Tatiana* Strizh[1]

[1]Joint Institute for Nuclear Research

Abstract. The JINR Tier-1 for CMS was created in 2015. It is important to keep an eye on the Tier-1 center all the time in order to maintain its performance. The one monitoring system is based on Nagios: it monitors the center on several levels: engineering infrastructure, network, and hardware. It collects many metrics, creates plots and determines hardware components states like HDD states, temperatures, loads and many other. But this information is not always enough to tell if the Tier-1 services are working properly. For that purpose, a service monitoring system was developed in order to collect data from different resources including WLCG monitoring services. The purpose of this system is to aggregate data from different sources, determine states of the services based on new and historical data and react according to some predefined instructions. The systems, general idea, and architecture are described and analyzed in this work.

1 Introduction

The Joint Institute for Nuclear Research(JINR) is an international intergovernmental scientific organization. The main areas of theoretical and experimental research at JINR are Particle Physics, Nuclear Physics, and Condensed Matter Physics. Laboratory of Information Technologies(LIT) is one of seven laboratories of the JINR. It is responsible for networks, computer and information resources, as well as mathematical support of a wide range of research at JINR in high energy physics, nuclear physics, condensed matter physics, etc.

In November 2011 it was decided to create a Tier-1 center for LHC experiments in Russia[1]. A proper infrastructure for the ATLAS, ALICE, and LHCb experiments has been launched on the basis of the NRC "Kurchatov Institute" (NRC KI, Moscow), while the Joint Institute for Nuclear Research (JINR, Dubna) hosted the CMS experiment.

The Tier-1 site at LIT JINR (T1_RU_JINR in CMS mnemonics) was officially put into operation in 2015[2]. It is providing the storage and processing of the CMS data, according to the computing model adopted by the CMS collaboration. The T1_RU_JINR is one of seven sites of such a scale in the world, involved in CMS data processing. In the current and the following LHC runs the volume of data is expected to increase by many times.

A monitoring system is a crucial part of the complex system like the Tier-1 center since it is not easily observable, characterizable, and could fall into the fault state. The robustness and extensiveness of the monitoring system determine the amount of work for system administrators.

*e-mail: pelevanyuk@jinr.ru

2 Monitoring Approaches

There are three main approaches to monitoring and different combinations of them: a complete monitoring software platform, a monitoring system composed of standard layers, and a custom developed monitoring systems. All approaches have both advantages and disadvantages so they should be chosen wisely to fit the tasks.

2.1 Complete monitoring platforms

There are several software packages implementing a complete monitoring solution. They all provide basic state checks for hardware and common services. They work well with hardware monitoring since they have many checks built-in or easily available. It is also possible to add custom checks which usually involves some amount of programming. Nagios platform was chosen for hardware monitoring in LIT JINR[3][4].

2.2 Composite monitoring system

In this approach, different systems are responsible for different aspects of monitoring activity and are integrated to provide the complete picture. Widely used set of components for monitoring is collectd[5] + InfluxDB[6] + Grafana[7]. They are responsible for collecting metrics, data storage, and data visualization respectively. The main idea behind this approach is to allow custom data flow, flexible visualization, and analysis. It is still possible to create a monitoring system without substantial software development investment with this approach. This approach is widely used nowadays[8].

2.3 Development of a custom monitoring system

This approach is used less due to the gradual improvement of two previous approaches but it is still a viable solution in a case where custom visualization and analysis are required. A good example is HappyFace[9]. The HappyFace Project is a meta-monitoring suit. It provides a modular software framework, designed to query existing monitoring sources to process the gathered information and to create an overview of the whole site and individual services. Development of a new system usually requires more effort and could be risky in terms of quality and supportability but it is a viable option when flexibility matters the most.

3 Service monitoring features

The Tier-1 center for the CMS experiment in JINR consists of services related to data transfer, data storage, and processing. It is an important task to present the state of all services in a comprehensive form. The monitoring data about services are scattered among many different sources. Some of them could be accessed via ssh or local web-interfaces, others could be gathered from central services in the form of JSON or CSV.

Some of the data are not simple metrics but rather representations of a state. For example: when checking the state of our computing resources it is not enough to know the number of completed jobs, but also the number of jobs completed by other Tier-1 centers. It helps to decide if a local problem causes the lack of completed jobs or they are absent at all Tier-1 centers. Thus more than a simple metric is needed to determine the state of the computing resources.

Another issue is that in many cases data received by a data collector represent only the current state and correct interpretation requires comparison with historical data. This task

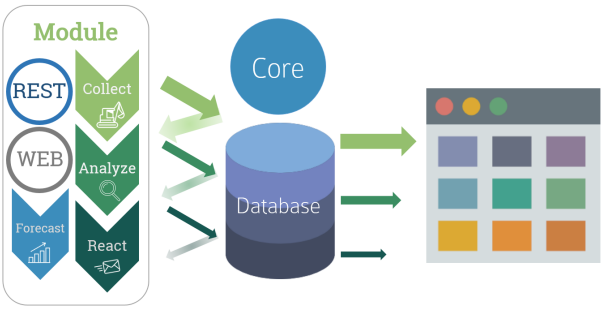

Figure 1. Architecture of service monitoring system

would be not trivial for some of the monitoring platforms and may require an additional database.

Finally, calculation of the state of a service may require some information from other services. For example, if we had no jobs for the past hour it could be either problem with computing service or with data transfer service, or with some central grid services, so we should check them before indicating that the state of computing service is bad or critical.

These are the reasons why we decided that standard monitoring systems may work not so well and would require some tricks in order to cope with these issues. With the development of a new monitoring system some additional work is required to develop the core of the system which would allow using different modules. Modules development for different data sources and services is required for any approach. That is why we decided to design and develop the monitoring system specifically for service monitoring.

4 Architecture of service monitoring system

The variety of monitoring information sources requires a targeted approach to every information source. That imposes a modular style on our architecture, where for each source of data we have a module which is responsible for retrieving, storage, analysis, reaction, and visualization. The schema of the system architecture is shown on the Figure 1. A module consists of two files. One file with source code written in Python where we keep the database structure, code for retrieving data, analysis, reactions and also the set of methods to expose data for the web application via a REST interface. The second file contains JavaScript code for the web application. In JavaScript code, we describe how the module will behave and will be displayed on a web page.

With this architecture it is possible to focus on modules: add and change them without affecting the whole system. This architecture also allows implementing modules which do not need to use any of the proposed parts of a module, for example: it is possible to make a module which has only the web and the REST components, so it does not collect anything, but it can display on the web page data from some other sources. That is how our journal of checks works: it retrieves results of all checks for the last 12 hours and puts this data to HTML table which allows sorting and filtering.

In this architecture, the core contains the code used by all modules. It is also responsible for creating the database tables described in the modules and updating them according to the changes in the source code of modules. We also include in the core a web application based on the Django[10] framework. This application is responsible for serving all REST functions listed in all monitoring modules.

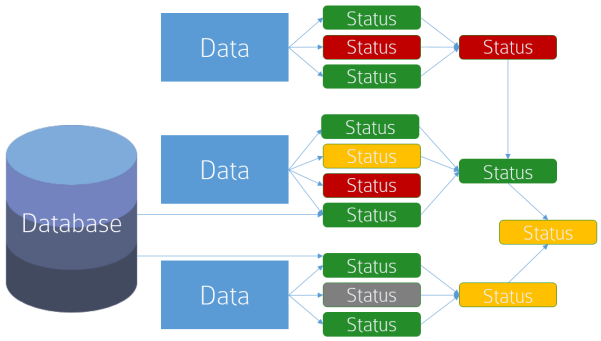

Figure 2. Example of states dependencies

A database was introduced to keep collected data and some additional information. We are using PostgreSQL[11] as a default for our modules. This means that if PostgreSQL is suitable for the task the developer could create all tables just describing them in the code of a module. But it is possible to use any source and storage of data. In case of a need, it is possible to override methods responsible for writing and reading data.

5 Events and states

Most of the time we receive raw data for service monitoring. This data are only practical for system experts. But being transformed to service states they can be useful for operators. Service state tells operators how good the performance of the service are, for example Excelent state or Bad state. State of a service may consist of several minor states and they could sum up into the state of the service. Depending on the service state an operator could take an actions. This idea was chosen to implement in the monitoring system.

In the monitoring system a module may have states defined. A single state consists of a name, a time of the last change, a small description, and a value. The value is a number from 0 to 50. These numbers have mappings to states: Excellent-10, Good-20, Normal-30, Bad-40, Critical-50, Undefined-0. This set of states was chosen to be default since it represents different operator actions. Excellent - service performance is high and could be used as an example. Good - service behaves good and does not need a tuning. Normal - service performance is OK, but some actions could be taken in order to improve it. Bad - service requires attention or fix, notification could be sent via e-mail. Critical - administrator intervention is highly demanded, notification could be sent via SMS.

It is the developer who decide how to rate the state of the service depending on the collected data. The state of the service may consist of several minor states and developer may choose the rules how to sum up them, for example take an average state value, choose the state worst, consider undefined as critical, etc.

For now, we are taking into consideration the worst minor state as the state of a service, but this could be adjusted. On Figure 2 is an example of possible state dependencies.

State is a characteristic of a component which exists all the time with different values. There may be some changes which do not necessarily lead to a state change but are still important for administrators. So, the notion of an event was introduced. Every state change is an event. Events could be also created depending on other events or data received by the module. This allows reacting to a set of different events. This also allows a high-level view of the whole system and changes in its behavior.

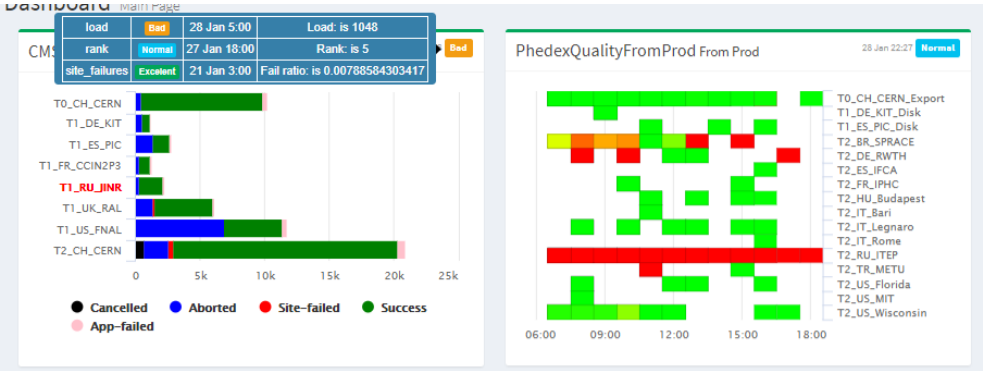

Figure 3. Detailed example of two modules

6 Web application - Dashboard

A web interface was chosen as the best way to present monitoring information to administrators and shifters. Users receive a pure html page with a list of modules for display. AngularJS[12] was chosen to instantiate web applications related to each module. The use of AngularJS allows easy configuration of the web application. The developer should create a list of modules in the form of HTML tags and AngularJS will spawn and launch all of them. An example of a module described in web application:

```
<smallbox box-app='PhedexQuality'
box-name='PhedexQualityFromDebug'
box-options='{"instance": "debug",
"delta": "12", "direction": "from"}'
box-description="From Debug"> </smallbox>
```

The *smallbox* tag is indicating the amount of space for the module. *box-app* is the name of the module. *box-name* is the caption for the module which will be displayed in the module header. *box-options* contains all options relevant for the module. And *box-description* is used as an additional caption for the module.

Modules are written in JavaScript. Developers can create new modules. There is functionality which allows overriding basic methods related to requesting data from the REST interface and data visualization. Developers could also provide the URL to get states and the framework will automatically download the data and update the state of a component. A simple indicator of the download in a form of spinning arrows was created to indicate that the module is downloading data right now. A small amount of space was provided to display the time of the last successful data download.

State information is downloaded separately. Received state information is applied to a small indicator in the top right corner of the module space. On mouse hover, the detailed information about minor states is displayed. Examples of this display for two modules are on the Figure 3.

Two modules displayed there as an example: CMS Job Status and Phedex Quality from JINR - Production. CMS Job Status shows the amount of Cancelled, Aborted, Failed and Successful jobs for last 8 hours in a form of a histogram. After hovering above module state the tooltip appears describing substates. It is possible to see that the module state is Bad due to the lack of load on worknodes: less than 1000 of jobs were completed during the last

period. Our Rank is Normal: we are fifth after T2_CERN, T0_CERN, T1_UK_RAL, and T1_US_FNAL. And Site_failures is Good: we have less than 1% of failed jobs.

7 Results

We were able to build a service monitoring system which is successfully working now. It can collect data from different sources, store it in a database and show on the web page. It was built with the Python language and the Django framework on the server side and HTML, CSS and Javascript for the web interface. The devised architecture allows modifying modules without affecting the entire system. This system is already providing the fastest way to get detailed information from 9 web-pages and there are more modules in development. It allows making notifications in case of problems leting the

References

[1] A. Berezhnaya, A. Dolbilov, V. Ilyin, V. Korenkov, Y. Lazin, I. Lyalin, V. Mitsyn, E. Ryabinkin, S. Shmatov, T. Strizh et al., *"LHC Grid Computing in Russia: present and future"*, Journal of Physics: Conference Series **513**, 062041 (2014)

[2] N.S. Astakhov, A.S. Baginyan, S.D. Belov, A.G. Dolbilov, A.O. Golunov, I.N. Gorbunov, N.I. Gromova, I.S. Kadochnikov, I.A. Kashunin, V.V. Korenkov et al., *JINR Tier-1 centre for the CMS experiment at LHC*, Physics of Particles and Nuclei Letters **13**, 714 (2016)

[3] A.S. Baginyan, N.A. Balashov, A.V. Baranov, S.D. Belov, D.V. Belyakov, Y.A. Butenko, A.G. Dolbilov, A.O. Golunov, I.S. Kadochnikov, I.A. Kashunin et al., *"Multy-level monitoring system for Multifunctional Information and Computing Complex at JINR"*, CEUR Workshop Proceedings (CEUR-WS.org) **2023**, 226 (2017)

[4] I. Kashunin, A. Dolbilov, A. Golunov, V. Korenkov, V. Mitsyn, T. Strizh, *"The monitoring system of Multifunctional Information and Computing Complex"*, CEUR Workshop Proceedings (CEUR-WS.org) **1787**, 256 (2016)

[5] *"collectd - the system statistics collection daemon"*, accessed: 2019-02-01, `https://collectd.org/`

[6] *"influxdata (influxdb) - time series database monitoring and analytics"*, accessed: 2019-02-01, `https://www.influxdata.com/`

[7] *"grafana - the open platform for analytics and monitoring"*, accessed: 2019-02-01, `https://grafana.com/`

[8] S. Bovina, D. Michelotto, *"The evolution of monitoring system: the INFN-CNAF case study"*, Journal of Physics: Conference Series **898**, 092029 (2017)

[9] V. Mauch, C. Ay, S. Birkholz, V. Büge, A. Burgmeier, J. Meyer, F. Nowak, A. Quadt, G. Quast, P. Sauerland et al., *"The HappyFace Project"*, Journal of Physics: Conference Series **331**, 082011 (2011)

[10] *"django - a python-based free and open-source web framework"*, accessed: 2019-02-01, `https://www.djangoproject.com/`

[11] *"postgresql - object-relational database management system"*, accessed: 2019-02-01, `https://www.postgresql.org/`

[12] *"angularjs — javascript-based open-source front-end web application framework"*, accessed: 2019-02-01, `https://angularjs.org/`