Architecture for Highly Configurable Dashboards for Operations Monitoring and Support

André CARDOSO, Cláudio Jorge VIEIRA TEIXEIRA*, Joaquim SOUSA PINTO

Instituto de Engenharia Electrónica e Telemática de Aveiro, Departamento de Engenharia Eletrónica, Telecomunicações e Informática, Universidade de Aveiro, Campus Universitário de Santiago, Aveiro, 3810-193, Portugal
marquescardoso@ua.pt
claudio@ua.pt (**Corresponding author*)
jsp@ua.pt

Abstract: Nowadays, information systems make use of a series of different components and platforms. Being able to monitor the health status of such components and platforms, along with the overall health status of the operation supported by such information systems is of paramount importance. The immediate solution for these monitoring needs is to use vendor specific applications, which results in having to look and understand several different interfaces. As it will be explained, there are available solutions that integrate such needs on a single platform. However, these solutions failed to provide a monitoring dashboard for the entire system operation, with highly customizable dashboard interface, easily defined metrics and easy mobile application availability. This paper describes an architecture for configurable dashboards capable of presenting heterogeneous metrics, side by side, regardless of their origin. A prototype, based on such architecture, is presented as a proof of concept. This prototype is being used to monitor the Cape Verde's Justice Information System. The result, based on the proposed architecture and prototype, is a custom tool with a single configuration file that can be adapted to different thresholds, metrics and monitoring scenarios.

Keywords: System monitoring, Operations monitoring, Real-time monitoring, Web-based dashboard, Mobile dashboard.

1. Introduction

One definition applied to monitoring is the supervision of "*activities in progress to ensure they are on-course and on-schedule in meeting the objectives and performance targets*" [5]. It is one of the most prominent techniques of the IT industry, used in diverse categories ranging from network monitoring and physical machines status query, all the way up to applications performance and information systems high-level queries. This type of monitoring and observation enables the overall supervision of Quality of Service (QoS) of systems and applications [11]. By assessing such benchmarks, it is currently possible to put the service level in place for the monitored systems.

These are powerful tools but "*if they fail to tell you what you need to know in an instant, you'll never use it.*" [6]. This means that if a system is put in place for monitoring critical resources, the monitoring system must be reliable even when the critical resources fail. The monitoring system cannot fail.

The need for monitoring is to such an extent that most systems come with tools for assessing internal relevant metrics in the context of the system's operation. For instance, in the Windows [18] operating system we find both a device manager tool and a task manager tool. The first relates to the hardware/driver state in terms of basic functionality, whereas the latter relates to performance wise metrics (usage and consumption of resources such as CPU, disk and network), general and by process. Other example is VMware's dashboard [33], showcasing both physical resources (barebones servers information) and virtualized resources (virtualized storage, virtualized network and virtualized machines). These are three examples of monitoring solutions for specific purposes.

With such tools, the monitoring of complex systems is cumbersome. For complex systems with several components, a large amount of monitoring tools is required, each for monitoring specific aspects of the system's operation. In such scenarios, the adoption of monitoring tools able to provide a holistic view of the system proves to be

https://doi.org/10.24846/v27i3y201807

benefic. On this type of systems, a system failure may be the result of several small component failures. Finding and fixing them rapidly, with minimal user impact, is critical.

In this paper, this type of systems. within a single user interface, integrates and summarizes different metrics required for monitoring the health and operational status of complex systems.

This paper consists of six sections. Section 2 presents the case study system, for which we are required to deliver an informational dashboard. Section 3 presents the state of the art regarding application-monitoring tools. Section 4 discusses the design and architectural principles pertaining to the proposed monitoring tool. Section 5 presents the developed prototype and, finally, Section 6 presents the main conclusions of this work.

2. Case Study

The work described in this paper is motivated by the need of having a monitoring dashboard to be used by the technical staff of Cape Verde's Justice Information System (JIS). The JIS is in use on Cape Verde's Courts for the computerization and dematerialization of both criminal and civil case files. It is a centralized system, used nationwide, and designed for use by all justice actors: judges, prosecutors, attorneys, clerks and criminal and national police [21], [27].

The JIS is an in-house, custom development, with a multitude of applications and servers. The need to develop a holistic dashboard, with vertical and horizontal information regarding the current operational status of the JIS, came with the requirement of better assisting the technical team, responsible for the development, setup, deployment and maintenance of the JIS.

The development team needs to be aware of the issues concerning the production environment, together with the issues concerning the staging, testing and even the source code development.

Figure 1 depicts the overall monitoring scenario. It consists of four major environments: Production, Support, Staging and Development. Each environment has its own requirements and needs in terms of what is critical and what is monitored.

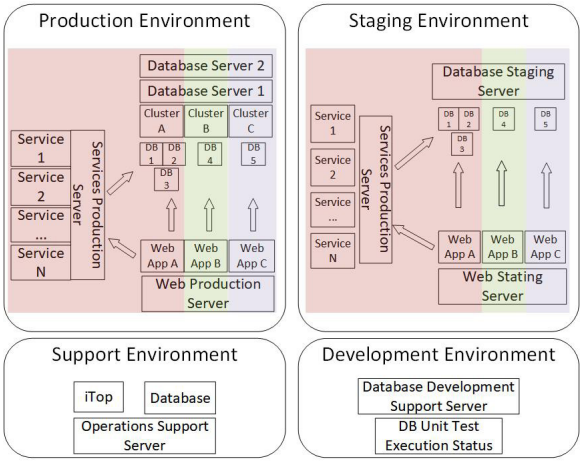

Figure 1. General System Architecture

In the Development Environment, the focus is on the result of batch-run unit tests, for early detection of regression issues, and for detecting new issues. This information is accessed with custom database queries against the database development support server.

The staging environment replicates the main architectural principles and servers' configurations and restrictions but simplifies the setup, in terms of redundancy of operation and performance.

The JIS application ecosystem consists of a series of web applications, each supported by a set of databases. There is also a set of internal, autonomous, services responsible for handling specific business logic. The scope of reaching each application is illustrated with a different color.

The major concerns in terms of staging environment relate to proper functionality and configuration. Despite being in a continuous integration development scenario, there are times where a disruptive change is required. As such, there is a need to attain metrics of the different modules operational status.

The support environment relates to the user support operations. As mentioned, JIS is a nationwide system for the courts. For user support, JIS has several helpdesk teams, with technicians ready to assist judges, prosecutors and clerks on a daily basis. These teams are also responsible for keeping the user's computers and peripherals up to date in terms of security updates and in proper working order. They are also the first responders to assist in any issues regarding the use of the JIS. To better perform on their function, and

to keep track of issues and solutions, the team uses an Information Technology Infrastructure Library (ITIL) [28] system, in this case iTop [3]. Generally, the helpdesk also brings some metrics, to better assess the team's performance, such as number of pending support tickets and average age of support tickets.

Finally, the production environment brings performance and high-availability issues on top of the staging environment. This environment has more servers, with high-availability clusters put in place, and more internal services (for parallelization purposes). The required metrics include proper functionality, number of active users, performance of request (page load times, systems response time, etc.), database backups, clusters' health, servers' resources status, etc. In addition, some operational metrics are of interest: number of case files filled, number of documents produced, etc. These metrics may indicate if there is some issue within the application, hampering users from fulfilling the desired tasks.

During their lifetime, information systems evolve. Despite setting up continuous integration and a huge set of unit, functional, performance and integration tests, in complex systems we easily lose track of current state of each module or service. Ensuring that all applications are running optimally all times is mandatory. In this specific case, the development and support teams require the monitoring of the metrics from the different environments. This means having a considerable amount of monitoring tools put in place, each with its own specificities.

The focus of this work was on researching the proper way of having these metrics aggregated and summarized, and on drawing the attention of the support team when something unexpected occurs.

The purpose of the present paper is the creation of a highly configurable and dynamic solution that can aggregate in the same interface metrics of different systems, with minimal user effort, in terms of configuration and maintenance.

As a proof of concept, a prototype capable of showing simultaneously the monitoring information will be developed from the previously presented environments.

3. State of Art

As stated by [31], "*most businesses across the whole spectrum combine at least two or three different tools to monitor and run their IT infrastructure*".

There are, however, solutions that enable the monitoring of the whole spectrum. These solutions will be the focus of the state of art, since they relate better with the goal of this work.

The solutions analyzed were selected based on online ranks for monitoring tools, cross-referenced with trending ranks and literature review. We analyzed ranks of monitoring tools described in several online reviews: a top 10 review [31], a top 20 review [19], a top 40 review [4] and a top 8 specific for 2018 [25]. Table 1 shows the results of the most mentioned tools, with the number of occurrences, among the articles, excluding the tools mentioned only once.

Table 1. Summary of the tools mentioned

Solutions	Number of Cites
AppDynamics	2
Bluestripe	2
Boundary	2
CopperEgg	2
Datadog	2
WhatsUp Gold	2
LogicMonitor	3
Nagios	3
New Relic	3
Zabbix	3
Icinga	4
Solarwinds (View-Trace)	4

The tools mentioned twice were ruled out, resulting a top six most cited tools. Thus, the most cited solutions on such online posts were cross-referenced with search trends statistics, using Google Trends. As described on the Google Trends website, data collected is *"... anonymized (no one is personally identified), categorized (determining the topic for a search query) and aggregated (grouped together). This allows us to measure interest in a particular topic across search, from around the globe, right down to city-level geography."* [26].

Chart 1 shows the most searched terms in Google search engine relevant for the selected tools, during a one-year time span. Google Trends indicator ranges from 0 to 100, where 100 is the maximum search interest reached.

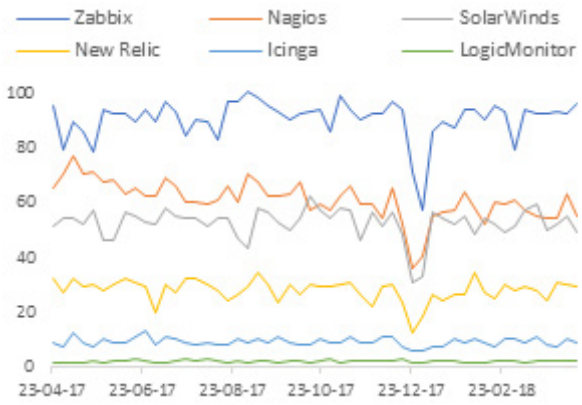

Chart 1. Google Trends Indicator

As expressed in Chart 1, Zabbix [34] is the most trended search. Despite some fluctuations, Nagios [22] is the second most trended search, followed by Solarwinds [30], New Relic [23], Icinga [20] and finally LogicMonitor [13].

Furthermore, we use the search engine at IEEE Xplore [9] to inquire which of these tools are most mentioned. The quoted tool name was used as the search phrase. The results obtained show that LogicMonitor is not mentioned at all. Solarwinds, Icinga and New Relic are mentioned twice, Zabbix seven times and Nagios, with thirty articles is by far the most mentioned tool.

Table 2 summarizes the presented information. It takes into account the number of citations in Table 1, the average of the inverse position of the Chart 1 and the result of IEEE search. These indicators, when normalized and added, total the score of each tool.

Table 2. Summary analysis

	Cites	Trends	IEEE Explore	Total
Nagios	0.150	0.238	0.682	1.070
Zabbix	0.150	0.286	0.159	0.595
Solarwinds	0.200	0.190	0.045	0.436
Icinga	0.200	0.095	0.045	0.341
New Relic	0.150	0.143	0.068	0.361
LogicMonitor	0.150	0.048	0.000	0.198

We selected the three tools with highest score in Table 2 for a more detailed analysis and comparison: Nagios, Zabbix, and Solarwinds.

3.1 Nagios

Nagios is one of the oldest and most mature open-source monitoring tools. Due to its versatility and extensibility through a plug-in mechanism, we can find extensions to almost every equipment or metric. It has a straightforward installation and configuration. We can extend its base configuration with custom scripts, alerts via emails or Short Message Service (SMS).

It is based on a Master/Slave architecture, with a central node running the Nagios Core component, capable of performing basic node analysis, and the slaves being different plug-ins installed on the monitored client machines, enabling the master to perform remote checks. There are two methods to monitor host and services: 1) Monitoring via Nagios Remote Plugin Executor (NPRE) agent where, the agent monitors the local resources and sends the data to Nagios Server. 2) Monitoring via public services, meaning it is accessible via protocols such as ICMP, SNMP and SSH. This method is useful for servers where we cannot or do not want to install an agent on. In the case of windows servers (as JIS), the Nagios Core offers a protocol "check_nt" that communicates with the predefined machines to be monitored. Figure 2 illustrates the communication channel established between the Monitoring Host (Nagios) and the windows clients, via the NSClient++ daemon.

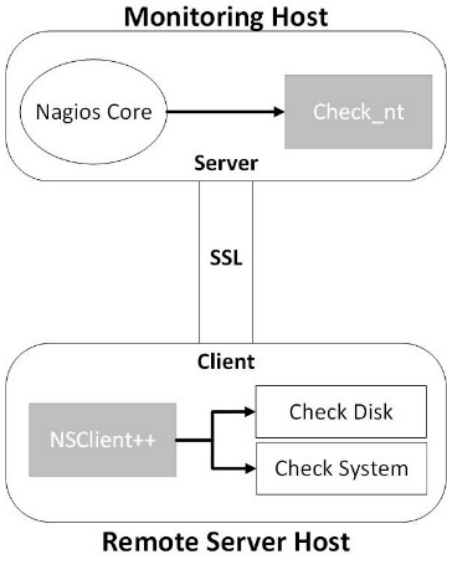

Figure 2. Nagios Communication Architecture

The "check_nt" is an old legacy protocol that only has some basic local system resource checks, with security being provided by SSL protocol.

In [12] authors conclude that Nagios is a good package and is being used by the masses. In [14] authors enhance Nagios for cloud computing monitoring and in [24] Nagios is integrated with OpenStack. Finally, in [10] authors conclude that Nagios is "*a powerful network monitoring software",* enabling *"independent developers to extend functionalities without modifying the Nagios core"*.

We can find it widely used in healthcare, education, retail and financial industries [8]. In terms of community content and support, the official forum has 65584 members and the YouTube channel (with 5900 subscribers) holds a bigger focus on tutorials.

3.2 Zabbix

Zabbix is an open source enterprise-monitoring tool for networks and applications that runs on a Linux based operating system. It offers a free public license and a paid service for custom solutions. It has a simple architecture composed of a server that deals with monitoring agents' information, database queries and notifications due to events, as shown in Figure 3. The agents are services that run on the target devices.

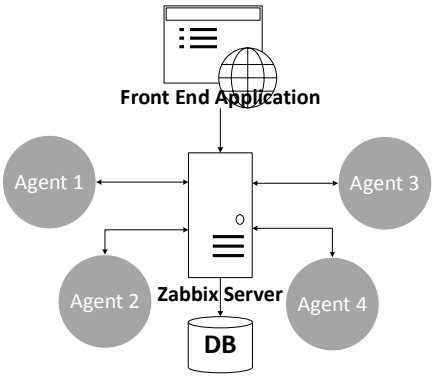

Figure 3. Zabbix architecture

As explained in [34], the key features provided are report metrics and automated alerts for devices, networks, databases, websites and report metrics. It is capable of monitoring services such as HTTP, ICMP and SMTP without requiring the installation of specific monitoring agents. Based on SNMP, Zabbix provides a network discovery tool to search for new Zabbix agents or new file systems, network interfaces, CPUs and Simple Management Protocol Object Identifiers (SNMP OIDs). When found, it will generate discovery events that automatically initiate pre-determined actions.

In [16]the Internet telecommunication companies are growing rapidly and now are based on the cloud computing environments. Management of a big distributed production infrastructure with multiple business services requires a centralized control system. This paper describes how the Zabbix enterprise-class monitoring system can be used as an adaptive solution for the purpose of real-time control of cloud computing resources, auto-detection of critical anomalies in advance and, when possible, auto-restore production services using a predefined workaround procedure. Real-world company examples are provided.","author":[{"dropping-particle":"","family":"Mescheryakov","given":"Serg","non-dropping-particle":"","parse-names":false,"suffix":""},{"dropping-particle":"","family":"Shchemelinin","given":"Dmitry","non-dropping-particle":"","parse-names":false,"suffix":""},{"dropping-particle":"","family":"Efimov","given":"Vadim","non-dropping-particle":"","parse-names":false,"suffix":""}],"container-title":"2014 6th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT Zabbix is used for real-time control of cloud computing resources and it is concluded that Zabbix is *"a powerful tool for effective control of cloud computing resources in a big distributed virtualized infrastructure"*.

In terms of community content and support, the official forum has 79688 members and the YouTube channel (with 1400 subscribers) holds a bigger focus on conference videos than tutorials.

3.3 Solarwinds

Solarwinds is a proprietary software that can be used to monitor a large variety of applications from a single dashboard that comes preconfigured. The number of modules and features available is extensive, and it supports a set of control actions on such nodes, being able to reboot server, terminate processes and fix issues remotely.

To assist on monitoring management, it features a user interface for adding and removing monitored nodes.

In terms of support, it hosts a training and certification academy, suitable for a company that needs a first response team when the core of their business depends on the availability of the system. Solarwinds is used by companies such as Accenture [1] consulting company and Lockheed Martin [15] an advanced technologies company.

In terms of community content and support, the number of members registered on Solarwinds forum and YouTube channel is not public. The YouTube channel is found to have a good balance between conferences and tutorials videos.

3.4 Concluding Remarks

Table 3 shows the summary of the described monitoring solutions. The major features compared relate to the type of architecture in place, the multiple systems monitoring and the support type these solutions offer.

Table 3. Monitoring Solutions Comparison

	Architecture Type	Support Type
Zabbix	Agent / Server	Professional and Training
Nagios	Agent / Server	Professional and Community
Solarwinds	Agent / Server	Professional, Community, Training and Certifications

As shown, all analyzed solutions use an Agent/ Server architecture. This approach enables a quick adaptation to new scenarios, by means of deployment of new, specific agents.

Documentation, support line and community forums are critical for the adoption and success of any product. The reviewed solutions follow this rule and offer such tools to end-users. Furthermore, we find the proprietary solutions with training and certifications.

Table 4 shows how each system compares to others in term of user interaction. In this comparison, the research is particularly interested in the type of interface supplied, and for customization allowed.

Table 4. User interaction comparison

	Interface Type	Interface Customization
Zabbix	HTML5, Mobile	No
Nagios	HTML5	Yes
Solarwinds	HTML5, Mobile	Yes

Zabbix and Nagios, being open source, enable a deep source code change. They also enable the use of themes that allow for HTML customization and mobile support. Additionally, on Nagios the community provides many unofficial features ready to use and customize.

Solarwinds is the most complete solution in our review. It contains several modules, easy interface customization and provides mobile applications for on-the-go monitoring. Solarwinds also allows us to add and remove nodes dynamically using a user's interface, in contrast to Nagios, which uses text files configurations.

In the process of finding a suitable monitoring tool, it was concluded that all reviewed applications feature the required functionalities for monitoring the case study system. They all announce monitoring of basic server resources and network. They support all the major protocols for communication and programing languages for agents' configuration or development. They all provide a Restful API [29], enabling agents to communicate with any server, if authorized. In addition, all solutions claim to be adaptable, scalable and versatile.

4. Design and Architecture

Despite all solutions presented being suitable for multiple systems monitoring, it was found that none could indeed support the present case study in full: deploy a monitoring dashboard for the entire system operation with highly customizable dashboard interface, easily defined metrics and easy mobile application availability.

These requirements demanded that regardless of the chosen solution, and considering the solutions reviewed, there would be the need to perform custom development.

As presented, the JIS comprehends a series of servers, services and applications. Within the applications, there is also the need to be aware of certain performance counters, related to business rules.

To accommodate such different metrics, the definition of the categories is proposed. Each category aggregates the information considered pertinent to it. Taking into account the JIS as the case study, we define seven categories, according to the case study operational

requirements: 1) Server resources; 2) Database status; 3) Case file metrics; 4) Websites status; 5) User activity benchmarks; 6) Tests performance benchmarks; 7) Helpdesk performance.

The **Server Resources** category congregates metrics related to the servers' operational status (topics): used/available CPU, used/available RAM, and used/available storage. These metrics are shown grouped by topic.

The **Database Status** category holds metrics related to the operational status of databases, namely information about the last successful backups (both full and transactional).

The **Case file metrics** shows information regarding the usage of the system: how many case files were consulted in the last days, how many requirements have been filled, etc.

The **Website status** shows metrics regarding the connectivity, availability, issues and performance of the several web sites of the JIS ecosystem.

The **User activity benchmarks** shows metrics concerning the user's performance on these sites: logged in count, password mismatch count.

The **Tests performance benchmarks** sums up metrics concerning the results of the latest automated test batch-run status (number of passed, failed and inconclusive tests).

Finally, the **Helpdesk performance** holds metrics on the helpdesk team: number of open tickets, average response time, average close time of such tickets.

4.1 Overall Architecture

To retrieve the information required for such metrics there was the need to develop specific agents that could access the data, generate and deliver the metrics to the monitoring solution. For the basic server monitoring solution, any solution with a RESTful API will be enough. Custom metrics are produced using additional monitoring endpoints. To better encapsulate the entire monitoring structure, a dashboard server was devised, whose responsibility is to get proxy information from the basic metric RESTful system and from the custom agents into the dashboard. Figure 4 illustrates the proposed architecture for the monitoring solution. It depicts the dashboard interface, the dashboard API and the Basic metric RESTful system. These three components enable the monitoring of any application ecosystem.

The Dashboard Interface is the main output of this solution. Conceptually, the interface must be responsive and easily adapted to fit on mobile devices as an installed application, with no additional setup or application configuration.

The Dashboard API is the driver/proxy of information between the interface calls and the application ecosystem. For security reasons, most of the resources monitored are in a protected intranet, meaning that direct access to them is strictly off limits. This proxy must act as a filter of requests and as a driver of information. The API defined within the Dashboard API server must be flexible enough to support any kind of screening on any kind of system inside the protected intranet, but must be objective, so that the call is correctly proxied to the right system. The Dashboard API can perform direct queries to internally available databases, check access times on web sites and proxy requests to the basic metric RESTful system.

The Basic Metric Restful System is an existing system, which can be installed and used as monitoring solution for the basic server resources (CPU, RAM, Storage, connectivity). The sole requirement of such system is its interoperability using a RESTful API.

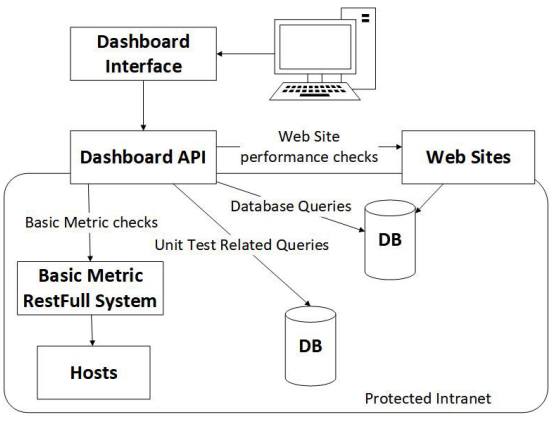

Figure 4. Proposed architecture for the monitoring solution

The Dashboard API also features a RESTful API for communication.

4.2 Dashboard Interface Architecture

As mentioned, the Dashboard interface is the most visible output of this project. One of the most important requirements of the interface is to deliver a highly configurable view where categories and topics can be added, removed, formatted and plotted on-demand. On top of that, we intend to use the same interface and logic to deploy mobile applications for on the go monitoring, screening and alerts.

Figure 5 illustrates the dashboard interface's main components. As expressed, the outputs of the dashboard may be visible using a regular web browser or within a mobile application, by using a mobile application generator.

The Dashboard application consists of a series of rules for querying information on the Dashboard API, and for displaying such information.

The Configuration file is the main piece of the interface, and the only one that needs changes to monitor different systems. It stores which categories to display, how to display them, which topics to display on the categories, how to display the collected metrics and which thresholds are relevant in the interface.

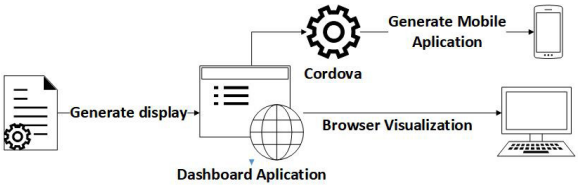

Figure 5. Dashboard Interface main components.

Once the application is ready for production, there is an intention of using a framework that creates a container to deploy the application to the different mobile platforms.

5. Implementation

With the architecture defined, implementing the monitoring solution was the next step. The basic metric RESTful system, the custom agents, the dashboard API and the dashboard interface were the main components to implement.

5.1 Monitoring Backend

From the list of candidates, the Nagios Core was chosen as the basic metric RESTful system. The main reasons were its free availability, good support forums and easy extensibility. Nagios Core 4.0.6 was installed on a server running Ubuntu 16.04 [2], using the default configurations. For Nagios assessment, the Nagios client was installed on the servers. On Windows Servers, the NSClient++ [17] was installed, more specifically, the version NSCP-0.5.2.33, with the necessary permissions to check the intended resources.

For the custom metrics check, a specific logic was developed to communicate with the systems. Considering the JIS case study, direct database queries were used on most systems. When this was not possible, the development team implemented RESTful based custom endpoints, within the required systems, to provide the required information. The Dashboard API triggered the database and endpoint queries.

The Dashboard API was developed as a .NET Restful service. It provides the interface defined in Code 1.

As shown, the interface was defined as a very broad and generic service proxy call. The parameters on the interface, *"type", "resource", "component", "datemin"* and *"datemax"* enable the retrieval of all the required information, on different formats.

```
http://IP/dashboard/api/{type}/{resource}
http://IP/dashboard/api/{type}/{resource}/
{component}
http://IP/dashboard/api/{type}/{resource}/
{datemin}/{datemax}
```

Code 1. Dashboard API definition.

The "type" parameter expects value "summary", "list", "table", "gauge", "linechart", "piechart", "infos" or "ping". With this API, the same information can be received in different formats. The parameter "type" enables this behavior. Values "list" and "table" are self-explanatory. Value "summary" will return a summary of the information and value "infos" will return detailed information. Finally, values "gauge", "linechart" and "piechart" return data properly formatted to feed a gauge chart, and value "ping" is used for checking connectivity, returning the response code of such connectivity request.

The "resource" parameter identifies the monitored resource in the IT ecosystem.

The "component" parameter identifies the single server's component that should be returned. It expects value "cpu", "ram", "drive" or "uptime".

Code 2 features some practical examples.

```
http://IP/dashboard/api/summary/db1
http://IP/dashboard/api/table/db1
http://IP/dashboard/api/gauge/srv1
http://IP/dashboard/api/summary/srv1/ram
http://IP/dashboard/api/ping/srv2
```

Code 2. Dashboard API Practical Examples

Parameters "datemin" and "datemax" are to be used in cases where a specific time window of events is required. This is the case of case file metrics and website status, mentioned before. In the case file metrics category, the "resource" parameter corresponds to the subcategory proposed within the case files, "procACN" for archived, queried or new case files, "procCPR" for civil, criminal and appeal case files and lastly "procPend" for pending orders and applications. For the second mentioned category, website status, the "resource" parameter can be one of "eventLogDetails", "eventLog" or "action". Correspondingly, the information received is represented by the last errors in error log, with the number of occurrences of each, last recorded events and last recorded actions. Code 3 presents some practical examples of such API calls.

```
http://IP/dashboard/api/lineChart/procACN/
{datemin}/{datemax}

http://IP/dashboard/api/table/action/{date-
min}/{datemax}
```

Code 3. Dashboard API with date interval

On Code 3 example, the last two variable fields are used to define the date range to which the query refers. This information is retrieved from the dashboard's configuration file.

5.2 Monitoring Dashboard

The monitoring dashboard has two main components: the configuration file and the user interface.

As mentioned, the configuration file holds all the information on which categories and topics will be in display, and where to get the information. In this file, each Panel represents one category. Code 4 shows the configuration properties of a Panel.

Most properties of the Panel are self-explanatory. It has an internal id (panelId) and a name. To assist on the visual configuration and interface, each panel may have a default background color, defined as a CSS property, and an icon from the font awesome dataset, or from an image URL.

Conceptually, each panel may have a brief summary of the metrics within the related topics. This summary of information should be available and visible if the panel is also visible. To achieve this, each Panel defines a set of summary objects. Each object has three properties, a property with the URL that provides the data, and two for displaying information, one aligned to the right and other to the left, both able to display the data received from a URL or a text label.

```
{   "panelId": number
    "name": string
    "defaultColor": string (css),
    "fontAwsomeIcon":
{font-awesome object}, opcional
    "imgSrc": string, URL, optional
    "summaryInterval": number,
    "summary":
[{"serviceURL": string (url),
"display": string,
"subdisplay": string}, …]
    "threshold": {threshold object},
    "views": {view object} }
```

Code 4. Panel Example

The threshold set enables the customization of warnings and errors within the dashboard monitor. The defined thresholds are specific to the dashboard monitor.

Finally, each panel defines a set of views. Each view relates to a topic from a specific category. The view displays detailed information regarding a given topic. Each view consists of a set of information for independent components and charts. Namely, each component defines its particular endpoint for getting information, the type of chart to display and the information refresh interval. An example of such component is expressed in Code 5.

```
{…
  "linearChart":
  {
    "type": "line",
    "dateConfig":
    {
      "type": "month"
      "value": 6
    },
    "xAxisModel":"{m}, {y}",
    "serviceUrl":"http://…",
    "serviceInterval": 30000,
    "legend": true
  },…
  "tables":
  [{ "title": string,
     "serviceURL": string,
     "serviceInterval": number,
     "footer": string, },… ] }
```

Code 5. Example of a view's configuration

With this configuration file, all the required metrics could be retrieved and displayed. In terms of information display, a prototype for a permanent operations control room display was required to be presented. Furthermore, there was also the need to present a mobile version of such dashboard.

The mobile version represents a summarized view of the dashboard, showing only the panel's information. In this version, and using devices with restricted displays, the views will not be shown.

The Dashboard interface was developed using JavaScript, HTML5 and CSS. For this development, the AngularJS [7] version 4 was used.

Finally, the mobile application is built using Cordova [32], that wraps all the application's resources and acts as a container for a running web application in the mobile device.

5.3 Look & Feel

In this section, the paper presents some of the visualization components of the developed dashboard prototype.

This application is mainly for visualization, with no user interaction, so the information of views and panels is automatically switched at configurable intervals. The top row contains the different components mentioned before and the side panels contain critical information about server resources that are always in display.

Figure 6 presents an integrated view concerning usage of server resources of two servers.

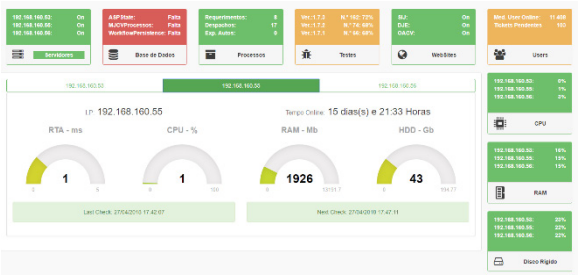

Figure 6. Server resources layout

In the overhead panels and side panels, a brief overview of other areas is present.

Figure 7 illustrates the responsive layout proposed for a quick mobile view of the application. The main information panel is not visible in this case.

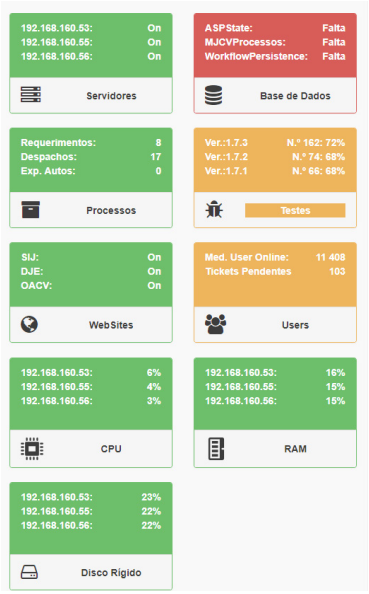

Figure 7. Simplified layout for mobile applications

Figure 8 presents a customized view concerning metrics related to the overall operation. In this case, the amount of new case files is inserted in the system.

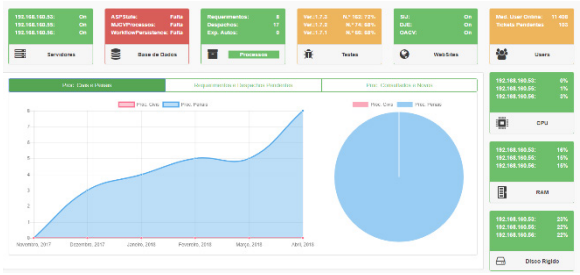

Figure 8. Case file metrics layout

Figure 9 describes the test runs on the development and staging phases.

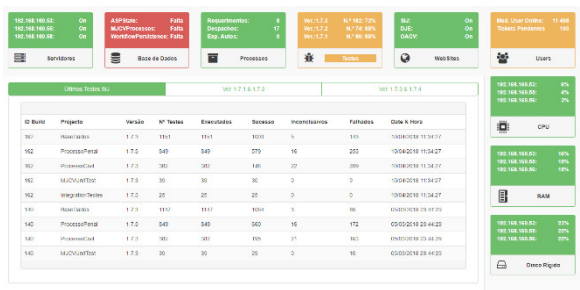

Figure 9. Tests performance layout

Figure 10 presents the overall status of the different web applications used in the case study.

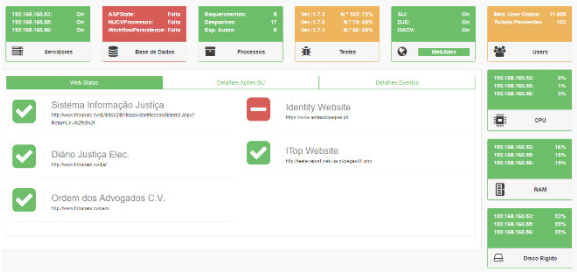

Figure 10. Websites status layout

6. Conclusion

In this paper, we presented a prototype of a customizable solution for monitoring complex systems. A monitoring solution was delivered, based on a single configuration file that can be adapted to different metrics and monitoring scenarios.

As a proof of concept, the prototype was configured to monitor the operational status of the Justice Information System of Cape Verde.

The JIS has been a good case study to develop and test this type of solutions because is an environment with different components and specific high-level metrics.

Being a web-based solution, it can be displayed on any HTML5 capable browser. Currently, it is under test as the intended use: with information displayed on a smart-TV.

REFERENCES

1. Accenture, *Accenture*. [Online]. Available: <https://www.accenture.com/us-en/new-applied-now>. [Accessed: 02-May-2018].

2. Canonical Ltd, *Ubuntu*. [Online]. Available: <https://www.ubuntu.com/>. [Accessed: 02-May-2018].

3. COMBODO Inc, *iTop Definition*. [Online]. Available: <https://www.combodo.com/itop-193>. [Accessed: 01-May-2018].

4. Demers, T. *40 Application Performance Management Tools*. [Online]. Available: <https://blog.profitbricks.com/application-performance-management-tools/>. [Accessed: 01-May-2018].

5. Dictionary, B. (2015). *Monitoring Definition*. [Online]. Available: <http://www.businessdictionary.com/definition/monitoring.html>. [Accessed: 01-Feb-2018].

6. Few, S. (2006). Clarifying the vision, *Information Dashboard Design The Effective Visual Communication of Data*, 223.

7. Google (2018). *Angular*. [Online]. Available: <https://angular.io/>. [Accessed: 02-May-2018].

8. iDatalabs (2017). *Companies using Nagios*. [Online]. Available: <https://idatalabs.com/tech/products/nagios>. [Accessed: 02-May-2018].

9. IEEEEXplore (2015). *IEEE Explore*. [Online]. Available: <http://ieeexplore.ieee.org/Xplore/home.jsp>. [Accessed: 01-Jun-2018].

10. Issariyapat, C. (2012). Using Nagios as a groundwork for developing a better network monitoring system. In *2012 Proceedings of PICMET '12: Technology Management for Emerging Technologies* (pp. 2771-2777).

11. Katsaros, G., Kübert, R. & Gallizo, G. (2011). Building a service-oriented monitoring framework with REST and nagios. In *Proceedings - 2011 IEEE International Conference on Services Computing, SCC 2011* (pp. 426-431).

12. Kaushik, A. (2010). Use of Open Source Technologies for Enterprise Server Monitoring Using Snmp, *Technology*, *2*(7), 2246-2252.

13. LogicMonitor Inc, *LogicMonitor*. [Online]. Available: <https://www.logicmonitor.com/>. [Accessed: 02-May-2018].

14. Luchian, E., Docolin, P. & Dobrota, V. (2016). Advanced monitoring of the OpenStack NFV infrastructure: A Nagios approach using SNMP. In *2016 12th International Symposium on Electronics and Telecommunications, ISETC 2016 - Conference Proceedings*, *no. 7* (pp. 51-54).

15. Martin, L. (2016). *Lockheed Martin*. [Online]. Available: <https://www.lockheedmartin.com/en-us/index.html>. [Accessed: 01-May-2018].

16. Mescheryakov, S., Shchemelinin, D. & Efimov, V. (2014). Adaptive control of cloud computing resources in the Internet telecommunication multiservice system. In *2014 6th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)* (287-293).

17. Medin, M. *NSClient++*. [Online]. Available: <https://www.nsclient.org/>. [Accessed: 02-May-2018].

18. Microsoft, *Windows OS*. [Online]. Available: <https://www.microsoft.com/en-us/windows/>. [Accessed: 01-May-2018].

19. Monitoring, A. P. (2014). *20 Top Server Monitoring & Application Performance Monitoring (APM) Solutions*. [Online]. Available: <https://haydenjames.io/20-top-server-monitoring-application-performance-monitoring-apm-solutions/>. [Accessed: 03-Jun-2018].

20. Monitoring Icinga Open Source, *Icinga*. [Online]. Available: <https://www.icinga.com/>. [Accessed: 02-May-2018].

21. Morais, R., Pinto, J. S. & Teixeira, C. (2014). Sistema de Informação da Justiça de Cabo Verde, *Rev. do Ministério Público*, *137*, 261-273.

22. Nagios Enterprises (2009). *Nagios Core - Features*. [Online]. Available: <https://assets.nagios.com/datasheets/nagioscore/Nagios Core - Features.pdf>. [Accessed: 01-May-2018].

23. New Relic Inc (2018). *New Relic Documentation*. [Online]. Available: <https://docs.newrelic.com/>.

24. OpenStack Foundation, *OpenStack*. [Online]. Available: <https://www.openstack.org/>. [Accessed: 20-May-2018].

25. Putano, B. *Top Server Monitoring Tools for the New Year*. [Online]. Available: https://stackify.com/top-server-monitoring-tools/. [Accessed: 01-May-2018].

26. Rogers, S. (2016). What is Google Trends data — and what does it mean ?, *Google News Lab*.

27. Rosa, J., Teixeira, C. & Sousa Pinto, J. (Jul. 2013). Risk factors in e-justice information systems, *Gov. Inf. Q.*, *30*(3), 241-256.

28. Rouse, M. (2014). *ITIL Definition*. [Online]. Available: <http://searchdatacenter.techtarget.com/definition/ITIL>. [Accessed: 01-May-2018].

29. Rouse, M. (2016). *RESTful API*. [Online]. Available: <http://searchmicroservices.techtarget.com/definition/RESTful-API>. [Accessed: 02-May-2018].

30. SolarWinds Worldwide LLC, *Solarwinds*. [Online]. Available: <https://www.solarwinds.com/>. [Accessed: 02-May-2018].

31. Sotnikov, A. *Top 10 Server & Application Monitoring Tools*. [Online]. Available: <https://www.acronis.com/en-us/blog/posts/top-10-server-application-monitoring-tools>. [Accessed: 01-May-2018].

32. The Apache Software Foundation, *Cordova*. [Online]. Available: <https://cordova.apache.org/>. [Accessed: 02-May-2018].

33. VMware Inc, *VMware Software*. [Online]. Available: <https://www.vmware.com/>. [Accessed: 01-May-2018].

34. Zabbix LLC, *Zabbix*. [Online]. Available: <https://www.zabbix.com/>. [Accessed: 01-May-2018].