ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA139

SCALING UP THE DEPLOYMENT AND OPERATION OF AN ELK TECHNOLOGY STACK

S. Boychenko, P. Martel, B. Schofield CERN, Geneva, Switzerland

Abstract

Since its integration into the CERN industrial controls environment, the SCADA Statistics project has become a valuable asset for controls engineers and hardware experts in their daily monitoring and maintenance tasks. The adoption of the tool outside of the Industrial Controls and Safety Systems group scope is currently being evaluated by ALICE, since they have similar requirements for alarms and value changes monitoring in their experiment. The increasing interest in scaling up the SCADA Statistics project with new customers has motivated the review of the infrastructure deployment, configuration management and service maintenance policies. In this paper we present the modifications we have integrated in order to improve its configuration flexibility, maintainability and reliability. With this improved solution we believe we can propose our solution to a wider scope of customers.

INTRODUCTION

The scale and increasing complexity of modern SCADA applications deployed in a wide range of domains in CERN's accelerator complex urged the development of a range of support frameworks for the back-end infrastructure's operation and maintenance. Until recently, data from different sources was stored and processed independently; this made data analysis a tedious task, requiring the manual matching of information from multiple domains. In order to combine data from multiples sources, make it more accessible and provide seamless access to the underlying measurements, the SCADA Statistics service was developed [1].

There are around 200 controls applications maintained by the Industrial Controls and Safety systems group and most of those archive data from thousands of devices into a persistent storage. The equipment experts working with historical data are generally interested in analysing the alarms and value changes (for example to determine if there are misconfigured devices). Despite the fact that the database allows extraction of the required data, due to its internal structure and huge amount of persisted measurements, the task is tedious and time-consuming. The SCADA Statistics project was developed to solve this problem, enabling fast and easy access to alarms and value changes data. In addition to the historic data analysis, the system collects and processes the data from controls applications logs, distributed across many servers, preserving the information which was previously discarded after some time (log files on the servers are rolling, i.e. overwritten when determined threshold is reached).

The SCADA Statistics service (see Figure 1) uses the Elastic stack, an open source search and analytics engine. The Elastic stack is composed of Filebeat (a lightweight way to follow and centralize logs and files), Logstash (a data processing pipeline), Elasticsearch (the search and analytics engine) and Kibana (a visualization tool for Elasticsearch data). The system can be split into three separate layers: data source, backend and frontent. The data source layer is composed by a large number of servers running the controls applications and a centralized storage where historical data is being persisted. These services are hosted on the dedicated hardware cluster, which is operated by the department's system administrators. The frontend layer functionality is assured by the IT department, and most of the maintenance tasks are performed by CERN's Elastisearch experts. The backend layer is running on the virtual machines (provided through Openstack technology), having a limited support, generally constrained to network and basic Operating System issues.

After three years in production, the SCADA Statistics service has generated over 1 TByte of data with more than $3.7x10^9$ indexed measurements. The input rate increases on a continuous basis as new applications and use cases are integrated into the service. In addition to the wide user base within the same working group [2], external clients (from ALICE [3] and potentially other domains at CERN) become interested in running the service on their own infrastructure.

Even though the existing solution fulfils the initial requirements, the "community" (i.e. "free") version of Elastic stack (which SCADA Statistics is based on) lacks some functionality like monitoring and failure recovery. These features are required if we are to extend the scope of the project for external users. Additionally, in its initial implementation, the system configuration and deployment was based on manually edited, long shell scripts which made the maintenance and integration of new features very difficult.

In this paper, we describe the consolidation project that made the SCADA Statistics service more flexible, maintainable and configurable.

SCADA STATISTICS INFRASTRUCTURE MONITORING

The initial experience with the maintenance of the SCADA Statistics stack proved that the system requires a constant monitoring, as unexpected failures were observed in most of the system components. The deployment of the conventional Metricbeat-based solution was not an option, since the "community" version of Elasticsearch, which was used to make the monitoring information persistent, did not allow the definition and configuration of alarms. Furthermore, this solution does not support an automated failure recovery mechanism, requiring the person in charge to perform

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA139

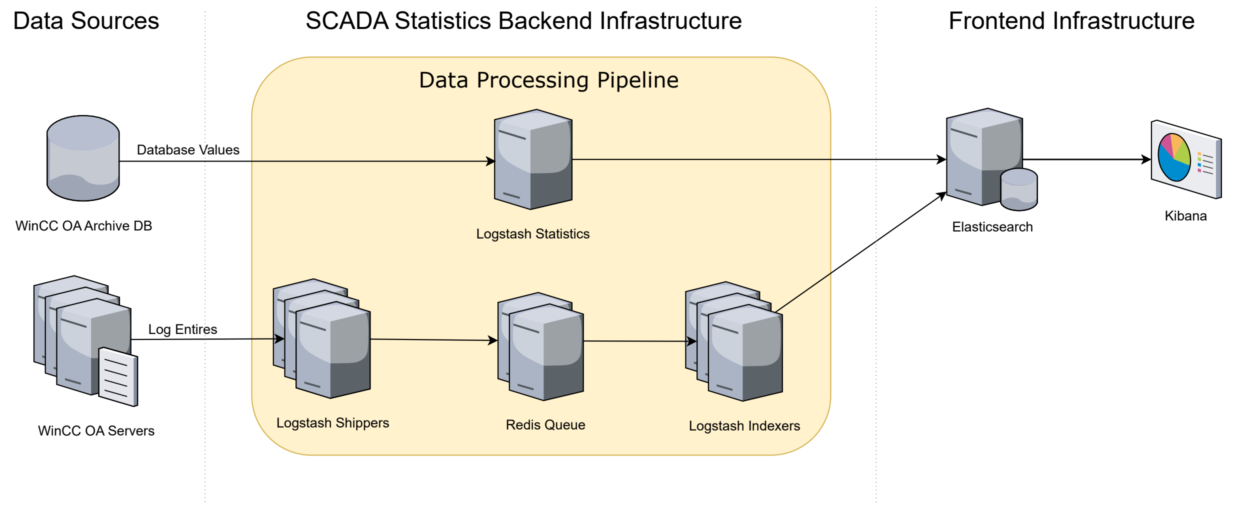

Figure 1: The initial infrastructure.

a manual intervention, even in the most simple cases (i.e. when some service has crashed and needs to be restarted).

The observed problems allowed us to define the requirements for the future monitoring system. First of all, it had to be pro-active, meaning that the number of manual interventions should be minimal. Secondly, the framework should notify the system administrators when an anomaly occurred, both when it was related to the system resources (e.g. high CPU or memory usage observed during a continuous time period) and to service behaviour (e.g. is the service was running and could be reached by other components in the data processing pipeline). Finally, the monitoring system had to be flexible, since the network where the solution was to be deployed is heterogeneous, with some of the cluster nodes having restricted access.

Based on the desired features, a few systems were studied as potential candidates for integration into the SCADA Statistics project infrastructure, namely COSMOS [4], Nagios [5] and Monit [6]. We started by evaluating the integration with COSMOS system, which looked very promising. Since it is being developed at CERN, we could count on strong support from our colleagues both during its setup and maintenance. On the other hand, at the time COSMOS was in an intense development phase, meaning that its stability could be affected and its API could undergo some changes over time. That was the main limitation and the reason why we proceeded with the evaluation of external monitoring systems. Nagios, despite fulfilling all of the desired features, required a steep learning curve at the beginning, and would therefore require significant man-hours investment, even for a simple setup. Finally, because it is able to supply most of the desired functionality with a simpler configuration, Monit became our choice for the monitoring tool, providing a good balance between supported features and setup complexity.

The configuration of the monitoring system for SCADA Statistics infrastructure follows the master-less approach; even though an additional daemon is running on one of the support machines to check the network status. The application on the the support machine is configured to periodically check whether all cluster nodes are reachable and that there are services listening on the pre-defined set of ports. The others, individual Monit instances, are configured in first place to check the status of the determined services, depending on the type of the machine where the daemon is running. Whenever an unexpected application termination is detected, an e-mail notification is sent to the configured list of recipients and an attempt to restart the failed component is performed. The previously described behaviour can be easily achieved by adding the following "human-readable" Monit instructions into a respective configuration file:

```
check process logstash
   matching "logstash"
   start program = "/bin/systemctl start logstash"
   stop program = "/bin/systemctl stop logstash"
   if does not exist then alert
```

Additionally, a simple configuration, similar to the one presented above, was added to track the system resources' usage and configuration file changes. A slightly more complex customization was required to setup arbitrary system metrics monitoring, like queue size. In this case, program status check (a Monit feature), was used to define a specific logic to assess those metrics.

CONFIGURATION MANAGEMENT PROCESS CONSOLIDATION

The configuration management consolidation addresses a different range of project issues. The revision of the current process was triggered by the need to upgrade the backend infrastructure to the latest Elastic-stack version, as its current one (at the time), was approaching the end of lifetime (no more bugfixes nor updates). The upgrade policy of Elasticsearch states that between 2 major releases the backward compatibility is not guaranteed; in our case we have detected backwards compatibility issues requiring major configuration changes. In order to execute the upgrade operation successfully, we needed a development environment where the new implementations and configurations could be extensively tested and validated in order to be applied to the production environment. After preliminary analysis of the existing configuration management and deployment procedures, we realised that the task would require a considerable effort to be accomplished, as the shell-based solution would need a complete review and redesign.

Shell script-based cluster management process may worked well in a small scale, homogeneous infrastructure; at the same time, once the system evolves and requires a more sophisticated setup, dedicated configuration management tools tend to be more efficient. The declarative approach of the configuration management tools (the developer does not need to define how to perform a determined operation) allows the framework to abstract from the implementation details and provide the conditions where the developer does not need to think about such details like the operating system where the deployment will be performed. Additionally, most of the configuration management frameworks are idempotent, meaning that in case of some operation is executed multiple times on a determined machine, its state will only be changed once, during the first execution (unless explicitly instructed otherwise). This functionality is a huge advantage in comparison to the shell scripts, where implementing such logic is time-costly and error-prone.

Amongst many available configuration management systems, Puppet [7] and Ansible [8] were evaluated, due to their wide adoption in the industry (as well as at CERN) and considerable high number of desired features in their "community" versions. The initial comparison between the two, revealed that Puppet would have required more effort to be adopted due to the required learning of the complex Domain Specific Language and Command-Line Interface used for configuring the framework. Other then that, Puppet requires client components to be installed on every cluster node, which rely on the master to function (which is a single point of failure), while Ansible is based on a serverless architecture and does not require any agent installation on target nodes to execute the configured tasks (other than SSH server and Python), further reducing the complexity of the infrastructure setup. Additionally, it also provides a possibility to execute ad-hoc commands on the configurable set of nodes, allowing faster prototyping and development.

The results of the evaluation helped us to determine that due to its simplicity and some of its unique features, Ansible suited better our SCADA Statistics use case. The SCADA Statistics project configuration management process was split into multiple infrastructure-level operations (Ansible Playbooks) like cluster setup or configuration deployment. Furthermore, each of those was divided into roles, which are

```
Setup Cluster
├── Install Basic RPMs
│   ├── Install WGET
│   ├── Install Java
│   └── Install RNG Tools
├── Install Logstash
│   ├── Setup Elastic Stack Repository
│   └── Install Logstash Service
├── Configure Logstash Indexer
│   ├── Determine Installed Logstash Plugins
│   ├── Install Missing Logstash Plugins
│   └── Update Logstash Configuration
├── Setup Logstash Directories
└── Start Logstash Service
```

Figure 2: An example Ansible Playbook.

used to define node-level applications, like Logstash setup. Some definitions are re-used between different Playbooks, allowing to reduce the amount of code which needs to be maintained. Finally, each of the roles is split into multiple tasks, which are responsible for interacting with a single Ansible module and execute a very targeted action, e.g. install Java or open a port in the firewall. The execution targets are defined in an a separate file, which consists of cluster nodes category sections, with corresponding machines lists. An example of a slightly simplified "Setup Cluster" Ansible Playbook, executed for Logstash Indexer host group can be visualised in the Figure 2. The roles are all of the children of the "Setup Cluster" component, while the tasks are the tree nodes assigned to each of the roles[1].

RESULTING SCADA STATISTICS INFRASTRUCTURE

In order to finalize the setup, several additional enhancements were done. First of all, we have migrated the monitoring system setup to a separate Ansible playbook, which would set for us all required parameters and deploy the generated configuration on the corresponding nodes. All of the produced configuration management was put into a source code control repository (GIT), so that we could track the changes and rollback if needed. Finally, all of the Ansible Playbooks were incorporated into Jenkins [9] Continous Integration and Delivery (CI/CD) framework, allowing to define precisely the execution conditions, environment and parameters. In Figure 3 we depict the latest version of the whole infrastructure with the respective monitoring and configuration management implementations.

Some of the Jenkins execution workflows, like configuration generation, are triggered automatically once a change in the source code is detected, while others require a manual action from the developer to perform its execution. This is done as an additional measure to protect the infrastructure from accidents (in case the cluster installation script is miss-

[1] note that childless nodes are actually composed by a single task with a similar name to the parent

configured) and to allow the developers to control the actions which alter the cluster state (configuration deployment task for example). Once the workflow is triggered, the developer is asked (in most cases) to provide some parameters (like hosts or the version of software which should be installed) and the rest of the execution is handled by the CI/CD framework. At any time, the history of the execution can be checked on the Jenkins Web user interface, allowing to track breaking changes, in case some have been introduced.

The monitoring and configuration management domain can be further extended to the data source infrastructure, in a similar way to that of the Monit configuration which can be deployed to the nodes which provide the log files. This way we could monitor the status of the Filebeat service and in case of a failure on the data collection machines perform the required maintenance.

OPERATIONAL RESULTS

The evaluation of the implemented solution for configuration management process and monitoring system is mostly based on the observations collected during the one-year period since the changes started being deployed into the production environment. The monitoring setup efficiency analysis is based upon the comparison between the number of the detected failures and their impact during different operation periods. Before Monit integration, Logstash service crashed several times, with failure detection delay ranging from one day to approximate one month. The most critical outage occurred on Logstash statistics calculation node, which stopped performing database queries and missing data was only detected four days later by the user. The data recovery took around 10 hours and was only possible because the source retains the copy of the required information for a long period of time. Longer service interruptions had less impact, since the nodes with failures had their functionality replicated; the problem here was mainly the increased resource usage and service availability. If since the introduction of the monitoring system, we still continue to observe random Logstash service crashes, the failure detection and recovery now requires only minimal interventions from developers. The Logstash statistics calculation service has failed three times during the past year, requiring only one manual intervention. The outage was caused by an underlying infrastructure update, which triggered an installation of the incompatible Logstash version, causing failures in communication between different infrastructure components. In this case the failure recovery process was performed in time in order prevent the users from experiencing any disruptions.

FUTURE WORK

Based on the findings and observations presented in this work we were able to identify several potential improvements which would require an additional investigation. First of all, the configuration and deployment process could be further simplified by combining the current solution with a container technology (for example Docker [10]). The individual

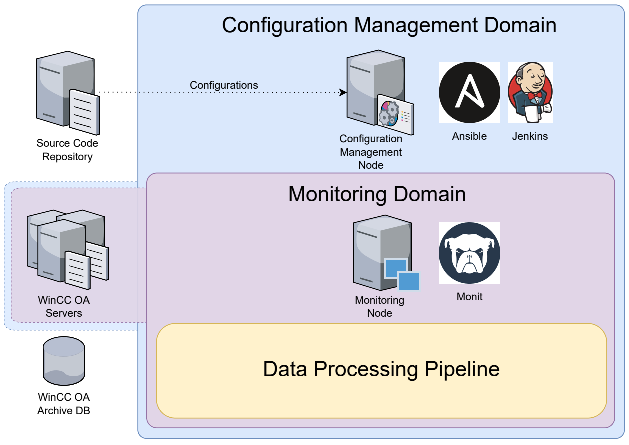

Figure 3: The resulting infrastructure.

infrastructure components, with respective dependencies and configuration, would be compiled into different Docker images. The images would be further instantiated through lightweight containers, serving as an abstraction from the underlying operating system, producing a consistent and uniform development environment (both for internal and external users). Amongst other possible additions which could improve some of the SCADA Statistics project infrastructure characteristics is the integration of different front-end solutions. Grafana [11] could bring several benefits, namely security enhancements. The "community" version of Kibana (native Elastic stack front-end) does not support any kind of authentication and authorization mechanisms, while Grafana provides a possibility for configuring role-based access, making it extremely useful to limit the access to the sensitive data.

CONCLUSION

The observed operational results have confirmed that the consolidation of the SCADA Statistics project infrastructure has made the system more maintainable, flexible and most importantly, reliable. The integration of the monitoring solution allowed us to reduce significantly the time required to detect failures and provided simple yet reliable self-healing mechanisms. Since its integration into the production environment, the number of manual interventions was minimal, in comparison to the previous experience. The improvements of the configuration and deployment management process have proven their efficiency both during the upgrade of the cluster and the setup of a similar infrastructure for external users. We could conclude that the chosen tools (Ansible, Jenkins and Monit) integrate well with the Elastic stack, and are viable options for use cases similar to the SCADA Statistics one.

REFERENCES

[1] J. Hamilton *et al.*,“SCADA Statistics monitoring using the elastic stack (Elasticsearch, Logstash, Kibana)”, in *Proc. ICALEPCS’17*, Barcelona, Spain, 2017.

[2] J. C. Tournier, *et al.*, “MONARC: Monitoring the Archiving Infrastructure of Control Systems", ICALEPCS’19, New York, USA, 2019, paper WEPHA019, this conference.

[3] K. Admodt *et al.*,“The ALICE experiment at the CERN LHC”, Journal of Instrumentation, vol. 3, num. 08 (2008).

[4] F. Locci, *et al.*, “CERN Controls Open Source Monitoring System", ICALEPCS’19, New York, USA, 2019, paper MOPHA085, this conference.

[5] Nagios - The industry standard in IT infrastructure monitoring, https://www.nagios.org/.

[6] A. Medina-González *et al.*,“Automated Configuration of Monitoring Systems in an Immutable Infrastructure”, *Proc. CIMPS’18*, Jalisco, Mexico, 2018 .

[7] Puppet - Unparalleled infrastructure automation and delivery, https://puppet.com/.

[8] J. Keating, “Mastering Ansible - Design, develop, and solve real world automation and orchestration needs by unlocking the automation capabilities of Ansible” (2015).

[9] M. Meyer, “Continuous integration and its tools”, IEEE Software Journal, vol. 32, pp. 14-16 (2014).

[10] D. Merkel, “Docker: lightweight linux containers for consistent development and deployment”, Linux Journal, vol. 2014, num. 239 (2014).

[11] Grafana - The leading open source project for visualizing metrics, https://grafana.com/.