

Integrated automation for configuration management and operations in the ATLAS online computing farm

Artem Amirkhanov¹, Sergio Ballestrero², Franco Brasolin³,
Haydn du Plessis², Christopher Jon Lee²,
Konstantinos Mitrogeorgos⁴, Marco Pernigotti⁵,
Arturo Sanchez Pineda⁶, Diana Alessandra Scannicchio⁷,
Matthew Shaun Twomey⁸

¹ Budker Institute of Nuclear Physics, Russia

² University of Johannesburg, South Africa

³ Istituto Nazionale di Fisica Nucleare Sezione di Bologna, Italy

⁴ Aristotle University of Thessaloniki, Greece

⁵ CERN, Switzerland

⁶ INFN Gruppo Collegato di Udine and Università di Udine, Italy

⁷ University of California Irvine, United States of America

⁸ University of Washington, United States of America

E-mail: atlas-tdaq-sysadmins@cern.ch

Abstract. The online farm of the ATLAS experiment at the LHC, consisting of nearly 4000 PCs with various characteristics, provides configuration and control of the detector and performs the collection, processing, selection, and conveyance of event data from the front-end electronics to mass storage. Different aspects of the farm management are already accessible via several tools. The status and health of each node are monitored by a system based on Icinga 2 and Ganglia. PuppetDB gathers centrally all the status information from Puppet, the configuration management tool used to ensure configuration consistency of every node. The in-house Configuration Database (ConfDB) controls DHCP and PXE, while also integrating external information sources. In these proceedings we present our roadmap for integrating these and other data sources and systems, and building a higher level of abstraction on top of this foundation. An automation and orchestration tool will be able to use these systems and replace lengthy manual procedures, some of which also require interactions with other systems and teams, e.g. for the repair of a faulty node. Finally, an inventory and tracking system will complement the available data sources, keep track of node history, and improve the evaluation of long-term lifecycle management and purchase strategies.

1. Introduction

The online farm of the ATLAS [1] experiment at the LHC consists of nearly 4000 nodes with various characteristics. Due to the large scale of the farm and the variety of the systems, appropriate tools to address various requirements are needed to effectively manage [2] and monitor these nodes [3]. As a result, when experts are performing routine interventions, they are required to update a number of individual tools to remove a node from production, schedule downtime, etc. This is a time consuming process, and the expert must remember to update all the tools in the correct order (as per the defined procedures). Additionally, a procedure may



require the expert to constantly monitor the status of the node to determine when it is ready for an intervention and this results in an ineffective workflow.

2. Tools overview

Currently, some of the main tools the experts have to deal with to properly address an intervention on a node are:

- ConfDB [4] – in-house Configuration Database
- Icinga 2 [5]– monitoring and health reporting system
- OKS (Object Kernel Support) [6] – an object-based in-house database defining the resources available for ATLAS data taking

Each of these tools is needed to perform specific actions that will be shortly described in the following subsections.

In certain cases, a ticket needs to be opened to keep track of the intervention being performed. The current ticketing system in use by the team is Redmine [7].

2.1. Configuration Database

The Configuration Database (ConfDB) [4] uses a MySQL database to store configuration and state information of all the computer systems of the ATLAS Trigger and DAQ (TDAQ) Online computing farm, and provides a web interface for performing various operations.

The tool aggregates specific system administration information with data from external sources, such as the CERN central network database (LanDB) and the ATLAS physical locations database. The automatic synchronisation with these external sources guarantees consistency across systems and decreases the likelihood of human error.

All the information related to the node hardware, operating system, and specific settings for the Icinga 2 checks, (for example, to override default thresholds or disable a specific check) is stored in ConfDB.

Finally, ConfDB manages the status of the node (in production, maintenance, retired, etc.) and function (TDAQ, Sim@P1 [8], etc.). These two flags are used by Puppet [9], the configuration management tool in use, when configuring a node to determine which services should be installed (the function of the node), and if they should be enabled or disabled (depending on the status of the node). These flags are also used to determine if a node should be included in the monitoring system, and what health checks should be performed.

2.2. Monitoring

The status and health of every node must be constantly monitored to ensure the correct and reliable operation of the whole online system. The monitoring system is not critical for data taking, but it acts as an early-warning system as it warns of impending issues whenever possible and provides alerts in case of failure.

The monitoring system [2] is composed of Icinga 2 [5], for the active checks and alerting system, and Ganglia [10], a scalable monitoring system that records performance metrics which is used for debugging and complementing the Icinga 2 information.

The IcingaWeb2 [11] interface provides an overview of every monitored node and the status of the health checks. The interface makes it possible to interact with the monitoring system, and in the event of an intervention, allows for downtimes to be scheduled to prevent receiving unnecessary alarms. Once an intervention has been completed, the health checks can be re-run to verify that the initial issue is no longer present.

The very large number of nodes to be monitored and the variety of configurations and settings required the creation of an automatic mechanism to produce the Icinga 2 configurations files.

SQL queries and regular expressions, which take advantage of the adopted naming convention for the nodes, are used to extract data from ConfDB and select the correct template for the node, thereby ensuring the correct monitoring configuration is used for each node.

2.3. OKS - Object Kernel Support

OKS [6] [12] is a library to support a simple, active persistent in-memory object manager.

It is used as the frame of the configuration database to provide the overall description of the Data Acquisition (DAQ) system, the trigger and detectors software and hardware. Such descriptions cover the online configuration of all ATLAS processes running during data taking and provide configuration parameters for many of them.

The information provided refers to which parts of the ATLAS systems and detectors are participating in a given run, when and where processes shall be started, when and in what order to shut down running processes, etc.

In this respect the nodes, which cannot be used because of some needed intervention, have to be disabled and not used in the data taking. The database is usually reloaded at the next start of the run to take into account any updated information. Once the intervention on the node is completed it needs to be re-enabled in the database, so that it is included in the running configuration at the next reload of the database.

3. Implementation

Having introduced the various tools that are used on a regular basis, the implementation of the automation system can be considered. The system will be required to interface with these tools, and take care of automatically executing a series of actions as defined by an arbitrary workflow. To ensure the longevity of the planned system, a community-developed and supported tool was preferred (compared to a custom-developed in-house tool). After evaluating various solutions on the market, Rundeck [13], a system which automates jobs, was selected for implementation and further evaluation. A number of scripts and plugins were written to enable Rundeck to interact with the various systems in use at ATLAS. These developments include:

- Node source script - regularly updates the list of nodes that Rundeck can operate on with live production status;
- ConfDB plugin - uses the ConfDB API to read and modify the current production status of the node;
- Icinga 2 plugin - uses the Icinga 2 API to schedule and remove downtimes, as well as run health checks for a node;
- OKS plugin - interacts with the OKS database to request that a node be inserted or removed from data processing;
- Notify plugin - triggers Webhooks (i.e. arbitrary HTTP callbacks) and sends SMS, email, and Redmine notifications;
- Waitfor plugin - allows a Rundeck job to be paused¹ until a node is in a desired state or until a manual intervention has been completed.

To illustrate Rundeck's utility, Figure 1 shows a typical series of actions that need to be performed when intervening on a node. Figure 2 shows the Rundeck implementation of this workflow. With the implementation of Rundeck, the majority of these actions can be offloaded and handled by Rundeck, allowing the expert to continue with other work. This is especially useful when interacting with OKS, as OKS status changes can take a number of hours to come

¹ Rundeck itself does not support pausing a job. The plugin implements pseudo-pause functionality by calling a script that enters into a sleep state until some condition is met.

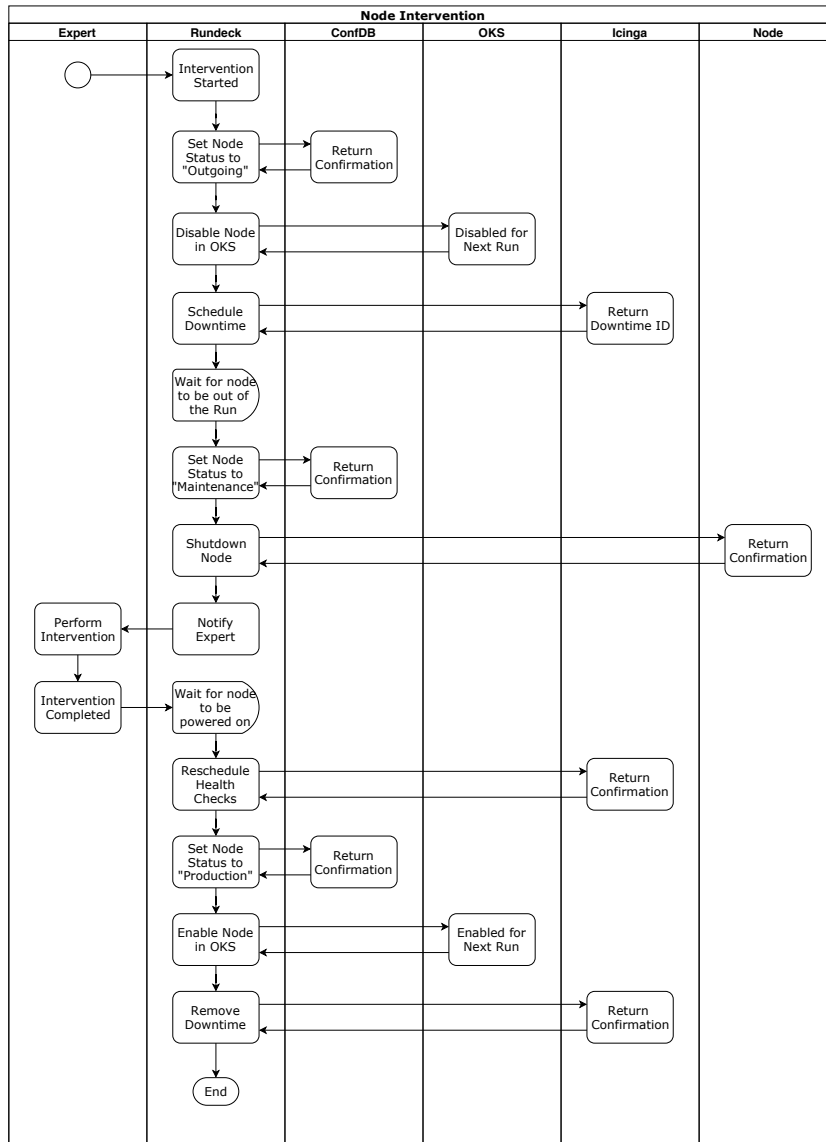


Figure 1: A typical workflow of actions performed with Rundeck while intervening on a node.

into effect. By using this Rundeck job, an expert can initiate an intervention and then be notified by Rundeck when the node is out of production and ready for maintenance. Once the intervention has been completed, Rundeck will handle all the steps required to bring the node back into production.

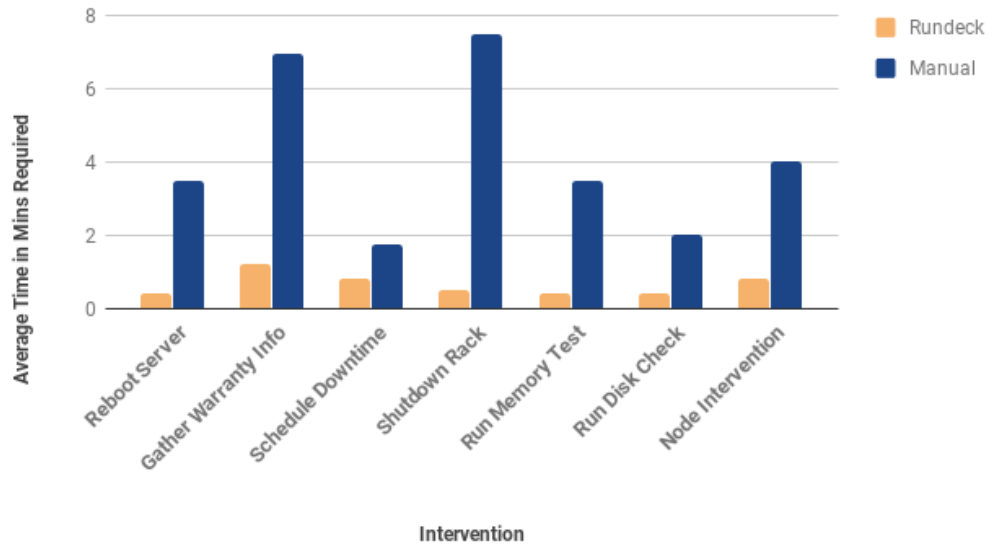
A useful feature of Rundeck is the centralised secrets storage facility, which allows keys and passwords to be securely stored, and then utilised by a Rundeck job. The usage of the key store can be seen in Figure 2, where secrets are injected into API calls and scripts when required through the use of `global.*` which are resolved to key store secrets at execution time (see steps 3, 5, 6, 14, 15, and 17).

This was tested first as a small pilot project in a lab environment containing approximately 400 nodes. Finally, the tool was deployed in mid-2018 within ATLAS where it is able to automate operations across 4000 nodes.

1. **OKS/Disable host**
[Disable a node from OKS \(checking the status\)](#)
 hostname: `${node.name}`
2. **Get Current ConfDB Status**
[ConfDB / Get Status](#) Fetches the current ConfDB status for a Node
 ConfDB API URL: `${globals.confdburl}`
3. **Set to Ongoing**
[ConfDB / Set Status](#) Sets the current ConfDB status for a Node
 New Status: `ongoing` ConfDB API URL: `${globals.confdburl}` API Key: `${globals.confdbuser}` API Key: `${globals.confdbpass}`
4. **OKS/Wait for OKS Status to change**
[Wait for OKS status to be 0](#)
 hostname: `${node.name}` delay: 30 wanted_status: 0 attempts: 48
5. **Schedule Downtime for 2 weeks**
[Icinga / Schedule Downtime](#) Schedules an Icinga Downtime for a Node
 Duration: 20160 Icinga API URL: `${globals.icingauri}` Icinga API Username: `${globals.icingauser}` Icinga API Password: `${globals.icingapass}`
6. **Set to Maintenance**
[ConfDB / Set Status](#) Sets the current ConfDB status for a Node
 New Status: `maint` ConfDB API URL: `${globals.confdburl}` API Key: `${globals.confdbuser}` API Key: `${globals.confdbpass}`
7. **Wait For / Create Lock File** Creates a lock file on the Rundeck server
 File: `/dsk1/rundeck-lock/${job.execid}-${node.name}`
8. **Send an email**
[Notify / Send Email](#) Sends an email
 Send To: `${job.username}` Send From: `rundeck@vm-atlas-rd-01.cern.ch` Subject: `TPU Intervention` Body: `The node ${node.name} is waiting for you. Delete the lockfile on the Rundeck server to resume the job: ${data.LOCKFILE}`
9. **Wait for a max of two weeks**
[Wait For / Lock File Removal](#) Waits until a lock file on the Rundeck server is removed
 File: `${data.LOCKFILE}` Check Interval (s): 300 Max Retries: 4032
10. **Wait 1 minute**
[*nixy / waitfor / sleep](#) wait for sleep to elapse, it will run locally once for each node
 Interval: 1m Cycles: 1
11. **Check Node is Up**
[Wait For / Remote Ping](#) Waits until a remote node is pingable from the Rundeck server
 Check Interval (s): 30 Max Retries: 20
12. **Check Port 22**
[Wait For / Remote Port Open](#) Waits until a remote port is open from the Rundeck server
 Remote Port: 22 Check Interval (s): 30 Max Retries: 20
13. **Wait 2 minutes**
[*nixy / waitfor / sleep](#) wait for sleep to elapse, it will run locally once for each node
 Interval: 2m Cycles: 1
14. **Return to Previous Status**
[ConfDB / Set Status](#) Sets the current ConfDB status for a Node
 New Status: `${data.PREVSTATUS}` ConfDB API URL: `${globals.confdburl}` API Key: `${globals.confdbuser}` API Key: `${globals.confdbpass}`
15. **Icinga / Reschedule Check** Reschedules Icinga Checks for a Node
 Icinga API URL: `${globals.icingauri}` Icinga API Username: `${globals.icingauser}` Icinga API Password: `${globals.icingapass}`
16. **Wait 1 minute**
[*nixy / waitfor / sleep](#) wait for sleep to elapse, it will run locally once for each node
 Interval: 1m Cycles: 1
17. **Icinga / Remove Downtime via Comment** Removes an Icinga Downtime for a Node
 Rundeck Execution ID: `${job.execid}` Icinga API URL: `${globals.icingauri}` Icinga API Username: `${globals.icingauser}` Icinga API Password: `${globals.icingapass}`
18. **OKS/Enable host**
[Enable node in OKS](#)
 hostname: `${node.name}`
19. **OKS/Wait for OKS Status to change**
[Wait for OKS status to be 2 \(enabled in OKS\)](#)
 hostname: `${node.name}` delay: 15 wanted_status: 2 attempts: 15

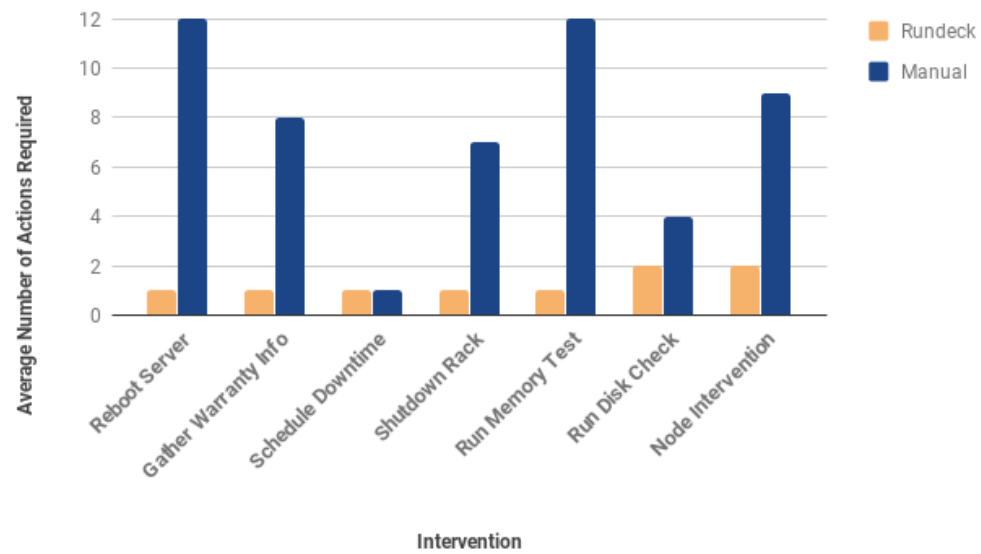
Figure 2: The resulting Rundeck workflow definition for a typical node intervention.

Time Required by Experts to Carry Out Interventions



(a)

Actions Required by Experts to Carry Out Interventions



(b)

Figure 3: Time (a) and number of actions (b) required to perform common interventions with and without Rundeck. Averages are based on the time and actions required by two independent experts.

4. Results

Since Rundeck has been deployed in both the test and production environments, a number of jobs have been created in order to simplify daily operations of the team. The time taken by an expert has been reduced, as shown in Figure 3a. This is because Rundeck is able to follow the execution of the intervention, leaving the expert free to continue with other tasks. Rundeck is also able to execute many commands almost instantly, whereas an expert would need to manually interact with various disparate systems. Additionally, Figure 3b shows that most operations have been reduced from many steps to simply launching the task from Rundeck.

Using Rundeck also improves security through the use of a very detailed activity log that provides audits for each action taken by users and Rundeck itself. As a result, it is always possible to know what actions were done on a node and by whom – something which was not always easy to determine. Moreover, the integrated key storage removes the need to store passwords in plain text within scripts. By using a shared platform for all interventions, experts are also able to see the status of interventions that were initiated by other colleagues, thereby preventing work from being repeated.

5. Inventory and tracking system

An inventory and tracking system would augment the current systems in use. Our survey up to now has not identified an existing open source tool suitable for our needs, so we expect to have to design and implement it.

This will keep track of the node history, thereby improving the evaluation of long-term lifecycle management and purchase strategies.

Further improvement of the overall automation and stability of the farm can be achieved through the design and implementation of a hardware tracking system. This would help the team by providing useful insights for the purchase of new hardware; for example, number of failures per vendor. The system would enable the team to handle spare parts more efficiently, which is becoming increasingly complex as the number of vendors and hardware configurations increase. This tool would also provide live availability information to the TDAQ infrastructure, ensuring that unstable nodes are not used for data taking. When integrated with Rundeck, it would be possible to alter the running job based on the history of the node.

6. Conclusion

Implementing an automation tool, like Rundeck, has reduced the number of manual processes that an expert must perform when carrying out an intervention. As a result, the expert's workflow would not be interrupted to regularly check the status of each step before proceeding with the next one. Although some plugins had to be developed to interact with various systems, this was easily accomplished using Rundeck's versatile plugin framework. Some effort was needed to work around Rundeck's shortcomings (such as not being able to pause a job, or not allowing branching within a job using conditional logic).

Rundeck has proven itself a valuable tool for improving and automating operations through its ability to interact with most systems used by the team. Further development will be needed to implement an inventory tracking system and subsequently integrate it with Rundeck. With its plugin framework and API, new integrations can be added with relative ease, further expanding its utility as new tools and systems are introduced.

Since its implementation, Rundeck has reduced the risk of human error by ensuring that correct procedures are followed for each intervention, whilst maintaining a clear audit trail of every action taken. With Rundeck handling many aspects of common interventions, experts are interrupted less, thereby allowing them to focus more on other tasks.

References

- [1] ATLAS Collaboration 2008 *The ATLAS Experiment at the CERN Large Hadron Collider JINST* **3** S08003
- [2] Ballestrero S, Brasolin F, Fazio D, Gament C, Lee C J, Scannicchio D A and Twomey M S 2017 *Journal of Physics: Conference Series* **898** 092008 URL <http://stacks.iop.org/1742-6596/898/i=9/a=092008>
- [3] Ballestrero S, Bogdanchikov A, Brasolin F, Contescu C, Dubrov S, Fazio D, Korol A, Lee C J, Scannicchio D A and Twomey M S 2015 *Journal of Physics: Conference Series* **664** 082024 URL <http://stacks.iop.org/1742-6596/664/i=8/a=082024>
- [4] Ballestrero S, et al 2012 *Centralized configuration system for a large scale farm of network booted computers J.Phys.Conf.Ser.* **396** 042004
- [5] Icinga2: <https://www.icinga.com/products/icinga-2/> accessed: 2018-11-22
- [6] Jones R, Mapelli L, Ryabov Y and Soloviev I 1998 *IEEE Transactions on Nuclear Science* **45** 1958–1964 ISSN 0018-9499
- [7] Redmine: <https://www.redmine.org/> accessed: 2018-11-22
- [8] Ballestrero S, et al 2014 *Design and performance of the virtualization platform for offline computing on the ATLAS TDAQ Farm J.Phys.Conf.Ser.* **513** 032011
- [9] PuppetLabs <https://puppet.com/products/how-puppet-works> accessed: 2018-11-22
- [10] Ganglia: <http://ganglia.sourceforge.net/> accessed: 2018-11-22
- [11] Icinga Web 2: <https://www.icinga.com/products/icinga-web-2/>
- [12] Almeida J, Dobson M, Kazarov A, Miotto G L, Sloper J, Soloviev I and Torres R *IReal-Time Conference, 2007 15th IEEE-NPSS; April 29 2007-May 4 2007 Page(s): 1 - 8* URL <https://ieeexplore.ieee.org/document/4382770>
- [13] Rundeck: <https://rundeck.org/> accessed: 2018-11-22