EOS Winston: Expert Systems for Automated Diagnosis and Remediation

August 2019

AUTHOR:

Ishank Arora

IT-ST-FDO

SUPERVISOR:

Luca Mascetti

CERN openlab

PROJECT SPECIFICATION

The IT-ST group at CERN runs and evaluates innovative cloud storage technologies for their application to big data problems in high-energy physics research. One of the entities it focuses on is EOS, the CERN multi-Petabyte disk-based storage service built from commodity hardware, heavily used as well by LHC and non-LHC experiments. The massive scale at which EOS runs leads to a room for multiple issues and anomalies to creep in. These need to be dealt with in real-time to ensure smooth operations.

The project aims to improve the current troubleshooting and diagnosis of the different components that compose the EOS infrastructure with the development of an Expert System that collects diagnostic information such as metrics, signals, and alerts, from each of the namespaces and assists engineers in reducing the time to debug these issues on the system, thus automating some of the troubleshooting tasks. This type of approach has been successfully used by Netflix and other companies to automate mitigation procedures on their systems.

ABSTRACT

This report describes EOS Winston, an event driven alerting and mitigation automation platform. Through the use of expert rules and online anomaly detection algorithms, it catches events which represent a departure from the normal behaviour and tries to automatically take corrective actions, often involving administrators in case it encounters events which it hasn't seen before and which haven't been described through the predefined rules.

The structure of the report is as follows: in the first section we describe what EOS is and the scale at which it operates, laying emphasis on the fact that operating at a peta-scale would involve anomalies and irregularities to creep in. We also describe the commonly occurring issues which previously involved manual intervention, and automatically mitigating of which was the main focus of this project. In the next section we discuss various alternatives for automation engines and anomaly detection algorithms, describing the pros and cons of each of these and explaining the rationale behind our choices.

The third section involves the description of the main platform - EOS Winston. After a brief overview of its architecture, we describe the Stackstorm pack, created as part of this project, which contains various corrective actions and the rules which link these with a variety of triggers. We also describe how the anomaly detection models were trained and how they are accessed through a REST API and the chatops integration which allows workflows to be executed from conversational channels.

Finally we discuss the various challenges faced during the project, how they were overcome and lay out some future possibilities which could further enhance the scope of the platform.

TABLE OF CONTENTS

1. INTRODUCTION

EOS [1] is a storage system developed at the IT department at CERN to facilitate the storage of extremely large amounts of raw as well as analysis physics data for experiments pertaining to the Large Hadron Collider as well as those coordinated externally. Secure access through multiple protocols with minimal latencies has made it indispensable to the scientific analysis and collaboration going on to further explain the standard model. But it often experiences certain repetitive errors which need to be mitigated in order to guarantee continued functioning. Handling these errors manually time and time again presents multiple problems which can affect the performance of the system as well as the efficiency of the administrators. Thus, there is a need for automated procedures to correctly identify these issues and initiate corresponding actions which can handle these without human intervention.

a. EOS: Exabyte Scale Storage

After being introduced as a file storage and management system exclusively for experimental data, EOS has grown to incorporate multiple use cases which span across all personnel at CERN. Other than providing a cloud synchronization platform, CERNBox, for users to store their personal as well as work-related data, it also allows the creation of project spaces where developers and physicists can collaborate on new enhancements and ideas.

To cater to the 5000+ users at CERN, the system needs to scale at an ever-increasing rate and hence there need to be continuous additions to the already existing entity. Ever since its inception in 2010 when it became operational for the ATLAS experiment with 2 PB of storage [2], it has been expanding at an exponential rate to result in the high-availability, low-latency storage pool that it is today. Some statistics which explain this staggering growth are as follows:

- EOS currently provides **280 Petabytes** of raw disk storage to its clients. These range from the individual LHC experiments (CMS, ATLAS, Alice and LHCB) to CERNBOX, the cloud sync utility and publically available data.
- **4.88 billion** files are stored which are available to their patrons without any measurable delays through multiple protocols.
- Each of the experiments and use cases has multiple namespaces accessible through metadata servers called MGMs and the actual data servers called FSTs. This results in a total of **25+ namespaces** accessible to users with varied authentication privileges.
- Each of these namespaces abstracts multiple nodes to ensure high availability in case of node failures. This results in a total of over **1200 nodes** currently deployed in production.
- **50,000 disks** are spread across these nodes to store such an enormous amount of raw and processed data. These are distributed across two data centres: the CERN center in Meyrin/Switzerland and the WIGNER center in Budapest/Hungary.

In addition to the scale, the multitude of features that are provided to the users makes EOS much more robust for use by different professionals. Some of the important features include:

- Access through **multiple protocols** including mounts, HTTP(S), and root, further make EOS more accessible through different devices and configurations.
- User authentication through **Kerberos certificates**.
- **Multi-user management** through access control and quota system

b. Recurring error codes and anomalies

Any system that operates at such a massive scale with so many functionalities is bound to face minor periodic failures, which can sometimes lead to much graver grievances. These can range from backend failures like expiration of file system draining or crashing of node daemons to those which might affect the end-users more directly, like unavailability of a particular node. Some commonly recurring examples which can be observed directly from the diagnostic information available from the MGMs are:

- The `nginx` daemons on a given node crashing.
- The `/var` mount on a given namespace instance running out of space. One of the reasons for this is the accumulation of log files created by multiple monitoring services.
- The failure in the boot process of multiple nodes in an instance. This could lead to downtime if a number of users try to access the instance simultaneously.
- A given user performing read or write operations at an alarmingly high limit for more than a predefined amount of time. This could also lead to unavailability for other users.
- The `heartbeatdelta` parameter on a node is a measure of its efficiency. Too high values indicate a bottleneck which affects its performance to perform the processing allotted to it.

While these errors have a fairly common occurrence and engineers are aware of measures to be taken in case these occur, some other anomalies and events which have not been previously observed might require a thorough diagnosis and thus, not have a straightforward solution. Catching these anomalies in early stages could prove to have extremely positive effects towards the performance of the system.

- Predicting the average load across the nodes of an instance could provide insights about whether the load is increasing in a continuous manner and after a certain point of time, might become too much for the instance to handle in its current state. If this situation could be predicted before it becomes a bottleneck, engineers can be notified about this and the situation can be handled before it has any major effects.
- A user writing to the disk at extremely high frequencies might continue to do so, causing a denial of request to other users who are also trying to access the same instance. Predicting this in the early stages can enable us to set appropriate limits on their behaviour.

2. POSSIBLE SOLUTIONS

The aim of the project was to develop and deploy a platform which could detect the onset of these errors by itself before they caused any major damage and initiate measures which could counter these. Specifically, the project had the following desirabilities:

- **Reduce the Mean Time to Repair (MTTR)** these issues. This includes the time passed since the issue occurred, is noticed by the administrator, and is corrected by the application of one of more measures.
- Repeated occurrences of the same issues and having to handle these again and again can often lead to **fatigue** among the administrators.
- There is also the need to figure out when the issue is actually a **false positive**. Reporting errors for a node which is actually supposed to be offline would not only create more spam but lead to the execution of actions which were needed to be paused for that particular situation.
- Building an infrastructure which could **easily deploy automated runbooks** which tackle these anomalies.
- For the addition of new runbooks, the team shouldn't have to focus on building the infrastructure from scratch again and again, but rather be able to **focus only on the business logic**.

Keeping these requirements in mind, two components were required which would build the backbone of the project: **an event-driven automation** engine which would detect and respond to such events and a **real-time anomaly detection algorithm** which would be plugged into this engine and govern the detection process. A number of candidates for both of these were analysed.

a. Candidates for Automation Engine

The emphasis was on searching for an open-source engine with a wide community support which possessed a majority of the features that might be desired from the platform at some point of time. Three candidates were evaluated for these:

i. ANSIBLE [3]

It is a platform where actions (e.g., BASH or Python scripts) are defined through playbooks, stored in static text files with no daemons or databases. Some of the pros are:

- Developed by RedHat, so good community support.
- Quite simple and fast.
- Has especially good networking support (supports Arista, Cisco routers).

On the other hand, it had a few disadvantages which more than outweighed the benefits of a large community support.

- The free version only supports CLI, no UI.
- Does not follow a server-client architecture, so can't be parallelized.
- Does not record any history of actions carried out.
- Does not support event-driven automation

ii. SALTSTACK [4]

It follows a master-slave architecture where minions run microservices on remote nodes, and communication takes place using ZeroMQ. Beacons fire events which are dealt by the reactor. Its advantages are as follows:

- Supports both agent-based and agent-less actions.
- High-performance for large deployments due to ZeroMQ.
- Used extensively by LinkedIn, hence is scalable.

And it suffers from the following limitations:

- Slow release cycle.
- Community provided modules are poorly tested.

iii. STACKSTORM [5]

It is a pluggable rule engine, comprising of triggers (activated by sensors) and actions - rules are defined using YAML or Python and it uses MongoDB for data storage and RabbitMQ for message broking. It possesses the following advantages:

- Supports a wide variety of interfaces, from Kubernetes to Slack to Twitter (available as packs on StackStorm Exchange, makes deployment lighter).
- Audit trail of all executed rules is recorded, can be accessed from a dashboard.
- Deployed by firms such as Netflix (Winston) and NASA.

At the same time, it has the following cons:

- Doesn't have the range of extension packs available compared with Salt and Ansible.
- No training solutions are offered.

b. Candidates for Anomaly Detection

Anomaly detection in time-series is a heavily studied area in machine learning. For our requirement, these algorithms should be able to predict in real-time if any given statistic is behaving differently from the norm based on past inference. State of the art Recurrent Neural Networks (RNNs) and algorithms implementing Hierarchical Temporal Memory (HTM) Networks were investigated for this.

i. DEEP AUTOENCODERS

With the advancements in deep learning across a variety of fields and modalities, it seemed to be the ideal candidate which could be explored to model the time series data we would be extracting from the EOS instances. However, the majority of recurrent networks which are currently employed across various applications either process data in batches or decompose the full time series to generate symbols. This makes them unsuitable for online prediction which is the requirement of our particular use case.

ii. NUPIC [6]

Nupic is an open-sourced implementation of Hierarchical Temporal Memory (HTM) networks developed by Numenta [7]. HTM is essentially a theoretical framework for sequence learning in the cortex and has been proven to work well in prediction tasks. Its application to anomaly detection in the online setting is achieved by the addition of two extra computations, as shown in Figure 1 [6].

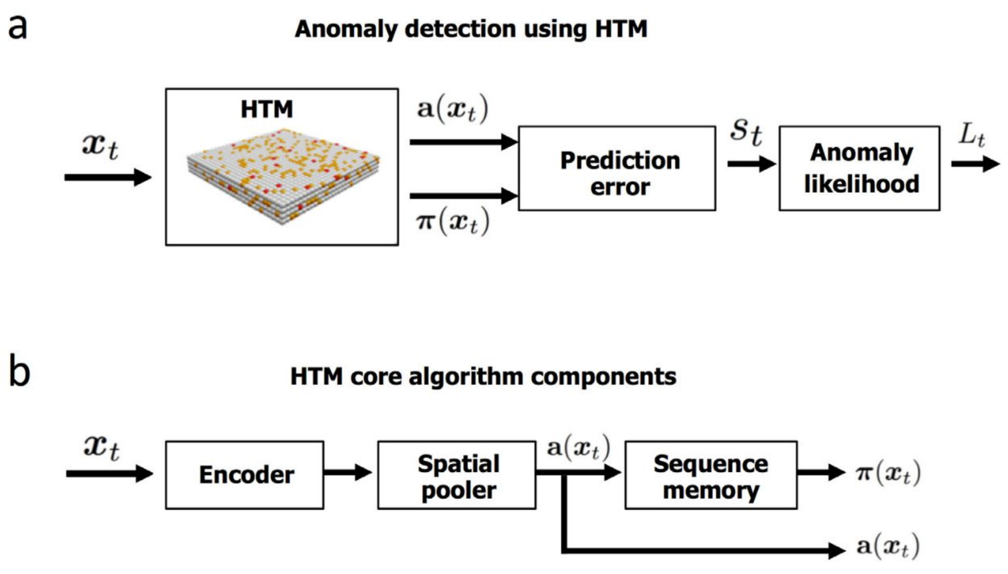
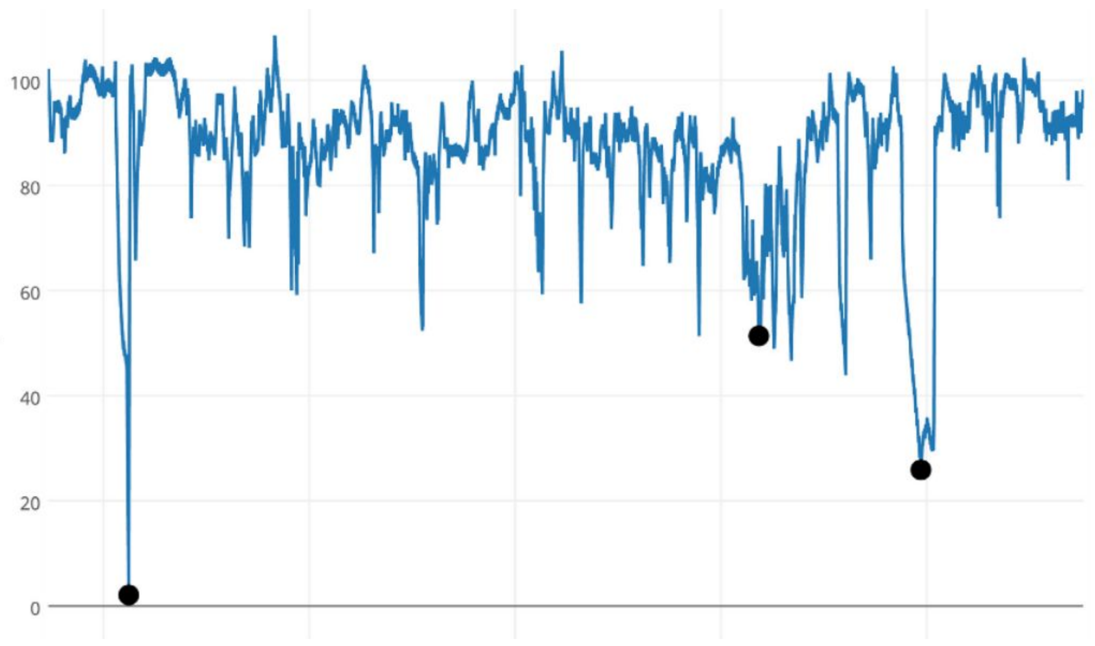


Figure 1. *HTM Networks for Online Anomaly Detection*

The current input is fed to the core HTM unit which consists of a spatial auto-encoder, followed by a pooler, which converts the input to a sparse binary vector which can then be represented in a much lower dimensional space. The sequence memory component is the essential component which then tries to model temporal as well as contextual patterns in this input data. $\pi(x_t)$ is the sequence memory's prediction of $a(x_t)$, for which the prediction error is calculated and based on that, the likelihood of the input being an anomaly.



3. EOS WINSTON

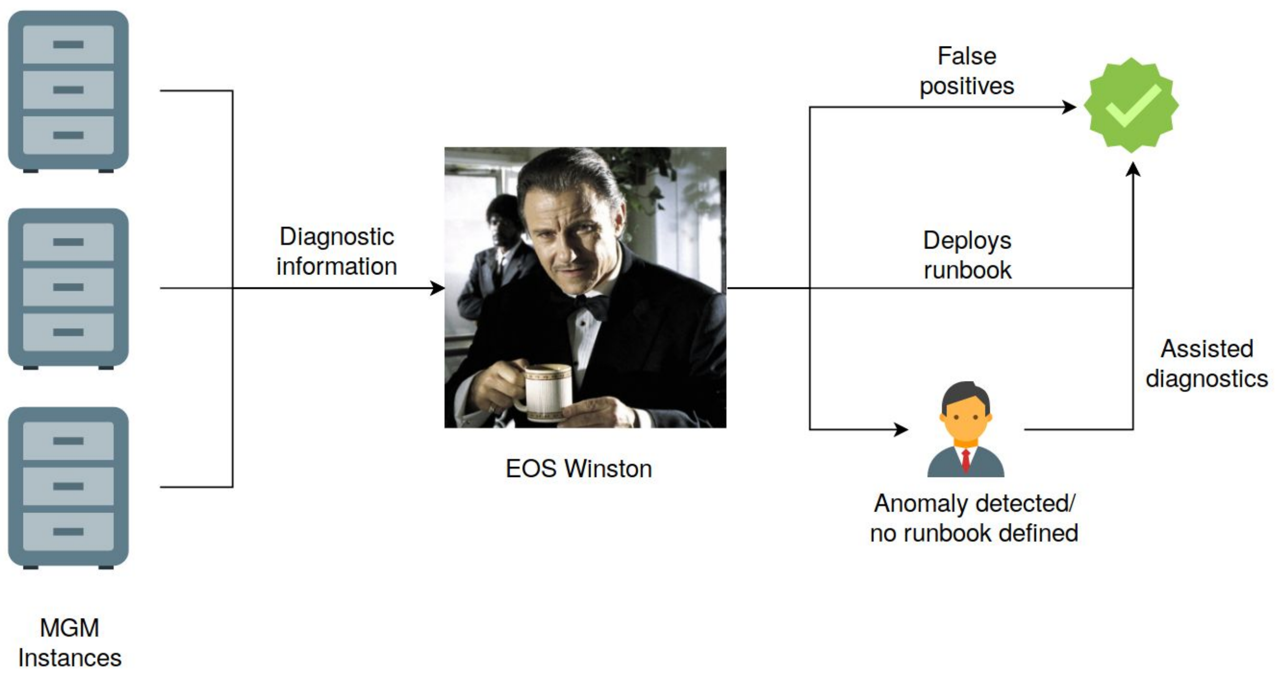
To cater to the requirements mentioned previously, a platform was developed which would utilize the described technologies to automate the error handling and mitigation procedure. Stackstorm was chosen as the choice for the automation framework for the multiple advantages it offers and owing to its ability to predict anomalies in real-time without the need to retrain on the whole of the data, Nupic was selected for the pattern recognition module. The resulting platform is named EOS Winston, which is called upon to automate the correction of recurring issues and the early detection of probable anomalous situations. To put it formally, EOS Winston is "*An event-driven alerting and mitigation automation platform, employing expert rules and streaming anomaly detection algorithms to categorise events and execute corresponding runbooks."*

a. The Architecture

Winston uses Stackstorm to gather statistics from the various MGMs. Stackstorm provides an end-to-end environment to read data and perform actions to make the appropriate changes. This is performed through various components:

- **Sensors** are plugins that receive callbacks or periodically poll to obtain data which is categorized into multiple events. External systems generate these events, which are used by Stackstorm to execute various triggers.
- **Triggers** are internal representations of the events received by the sensors. Examples include timers and webhooks. New Trigger can be defined and linked to an existing sensor or a user-defined one.
- **Actions** allow us to change the environment according to the data received by the sensors and processed by the triggers. Stackstorm provides a few generic actions such as shell commands and REST API calls, and can also be linked to multiple integrations such as Docker, Puppet and even social networking sites like Twitter.
- **Rules** map triggers to actions and thus provide a workflow through which the platform functions.
- **Packs** are used to group rules pertaining to a given entry together. So for our use case a new pack called EOS Winston was defined.

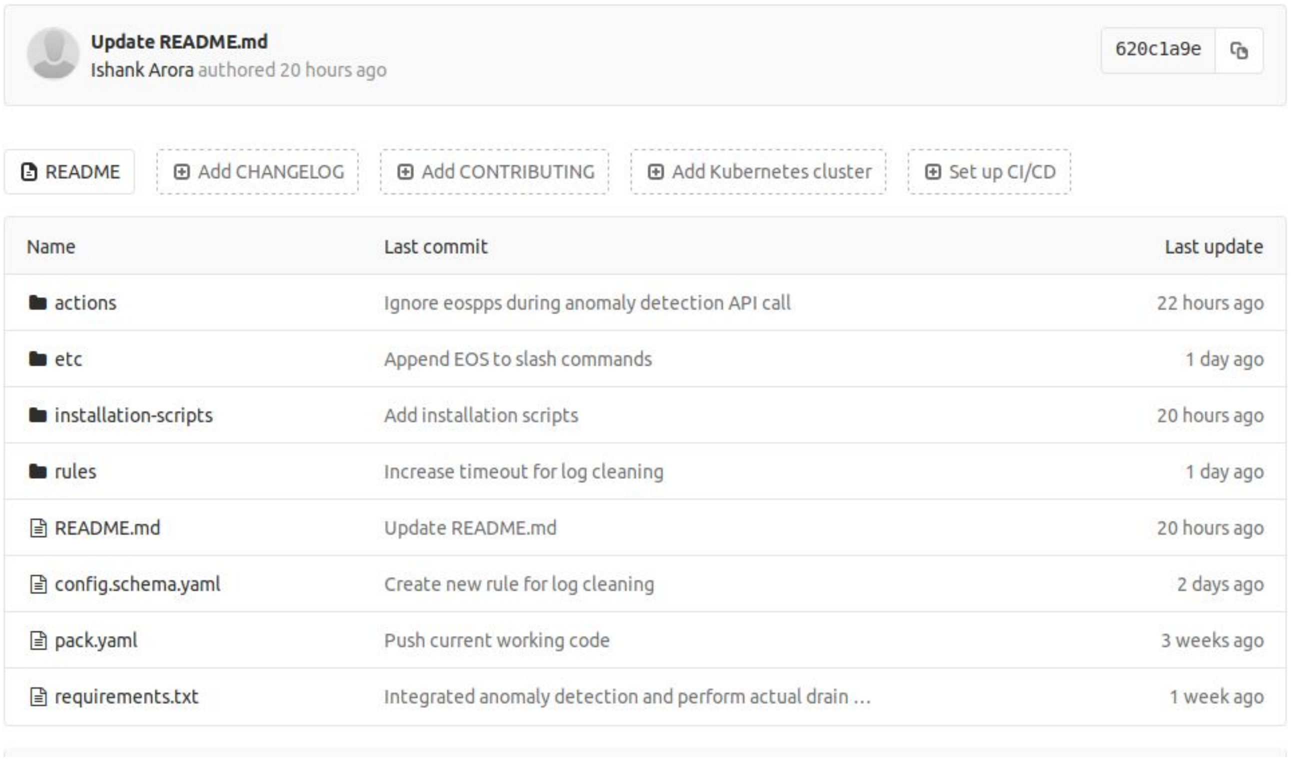
So using these existing entities within the EOS ecosystem we get diagnostic information from the MGM instances. This information includes various disk usage statistics, user activity, the EOS versions on the nodes and the load on various nodes in the instance. Once this data has been obtained by the corresponding sensors, Winston firstly tries to figure out false positives, and then processes the remaining events according to some expert rules as well as applies real-time anomaly detection to figure out cases for which predefined actions do not exist. Events for which rules have previously been defined are mitigated through runbooks which carry out appropriate responses on the various instances. The anomalies which are detected by Winston need help from the system administrator to be handled, so a corresponding alert in the Mattermost channel is posted and the relevant personnel are updated.



b. The EOS Winston Stackstorm pack

A new Stackstorm pack called EOS Winston was created for this project. The code can be found at https://gitlab.cern.ch/eos/eos-winston.The directory structure of the repository is as follows:

- The **actions** directory defines two actions - the first one, `diagnostic_platform`, for periodically polling the various instances and taking out the corresponding corrective measures and the other, `slash_commands`, for executing slash commands in the Mattermost channel to enable Chatops integration.
- The **rules** directory contains four integrating rules which facilitate the logical flow from sensors to actions. These carry out various functions like identifying the particular slash command which has to be executed, periodically renew the eosbot krb5 credentials for logging into the MGM instances, clean the outdated logs in case low space is reported in an instance, and periodically execute the diagnostic platform after every specified time interval.
- The **etc** directory contains the code to perform anomaly detection on the latest retrieved data, parse the individual system commands and a few other utility functions.
- **pack.yaml** contains the definition for the pack and **config.schma.yaml** lists the various user configurable parameters, examples of which are the various metrics which need to be retrieved, the corresponding commands for those metrics and various thresholds which govern the expert rules.
- The **installation-scripts** directory contains a few handy scripts with make installing Stackstorm and EOS Winston a single click operation on new systems.
- A snippet hosted at https://gitlab.cern.ch/snippets/916 enables easy installation across platforms.



Update README.md
Ishank Arora authored 20 hours ago
620c1a9e

README | Add CHANGELOG | Add CONTRIBUTING | Add Kubernetes cluster | Set up CI/CD

Name	Last commit	Last update
actions	Ignore eospps during anomaly detection API call	22 hours ago
etc	Append EOS to slash commands	1 day ago
installation-scripts	Add installation scripts	20 hours ago
rules	Increase timeout for log cleaning	1 day ago
README.md	Update README.md	20 hours ago
config.schema.yaml	Create new rule for log cleaning	2 days ago
pack.yaml	Push current working code	3 weeks ago
requirements.txt	Integrated anomaly detection and perform actual drain ...	1 week ago

i. eos_winston.yaml

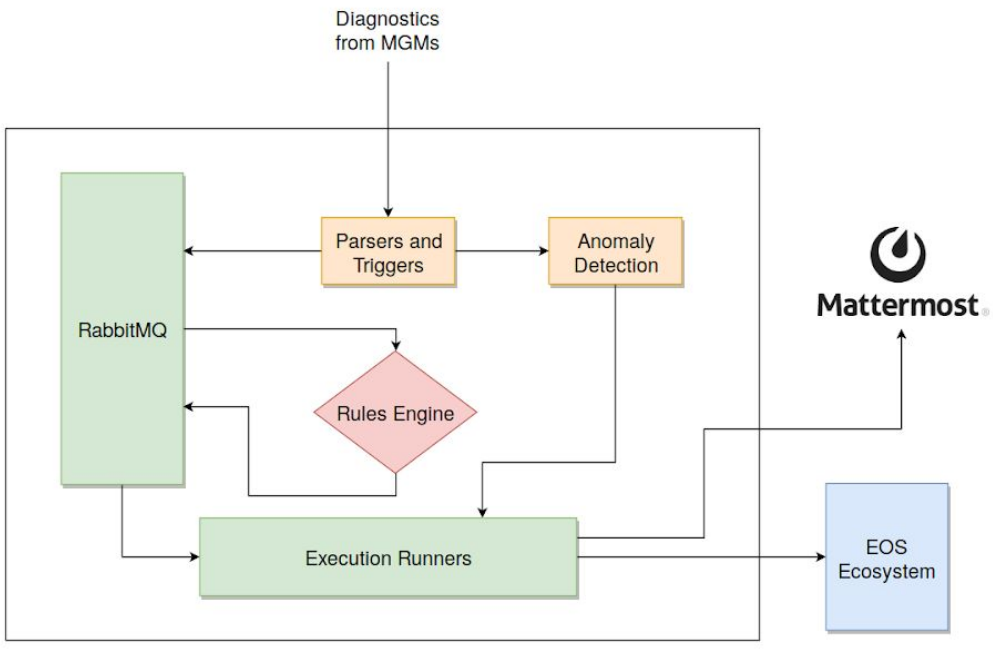
```
---
mattermost_webhook:
"https://mattermost.web.cern.ch/hooks/exdb7cypm7rxtb94d57wm5o4kh"
reports_dir: "root://eosproject.cern.ch//eos/project/e/eos-winston/report/"
anomaly_detection_api_url: "http://eosbackup-srv-w1.cern.ch"
log_cleaning_api_url:
"/api/v1/webhooks/log_cleaning/?st2-api-key=ZjdlNjA3NDE1ZWQxYTg4ZTEyMmQ1OTQ1YmQ0YWY
xMGVjMzdjYmI5YWQzMDJhNDY2M2NjNzYzOGU0YTc4NzNmOA"

update_interval: 6
instances_list:
  - "eosalice"
  - "eosalicedaq"
  - "eosatlas"
  - "eoscms"
  - "eoshome-i00"
  - "eoshome-i01"
  - "eoshome-i02"
  - "eoshome-i03"
  - "eoshome-i04"
  - "eoslhcb"
  - "eosmedia"
  - "eospps"
  - "eosproject-i00"
  - "eosproject-i01"
  - "eosproject-i02"
  - "eospublic"

metrics_list:
  - "nodels"
  - "fsls"
  - "nsstatuser"
  - "recycle"
  - "df"

metrics_commands:
  nodels: "eos node ls -m"
  fsls: "eos fs ls -m"
  spacels: "eos space ls -m"
  nsstat: "eos ns stat -m"
  nsstatuser: "eos ns stat -a -n -m"
  recycle: "eos recycle -m"
  df: "df"
  eosversion: "rpm -q eos-server"
  hostname: "hostname"
  accessls: "eos access ls -n -m"
  headnode: "rpm -q eos-server"
  clientls: "eos fusex ls -m"
  clientns: "eos ns"
temp_file_storage: "/root/report/"

threshold_config:
  disk_threshold_critical: 80
  disk_threshold_warning: 70
  heartbeat_threshold: 50
  node_uptime_threshold: 300
  daemon_start_time_threshold: 300
  recycle_volume_threshold: 80
  user_process_threshold: 1000
  load_anomaly_threshold: 0.6
  drain_expired_limit: 50
```

ii. ACTIONS

- `diagnostic_platform`:
    - Executes every one hour.
    - SSHs into the various MGM instances and retrieves the output of the specified commands.
    - Passes this data through the predefined expert rules.
    - Performs anomaly detection on the retrieved data using models trained on the previous data.
- `slash_commands`:
    - Executes whenever a user with appropriate access types a predefined slash command in the Mattermost channel.
    - Depending on the command, retrieves the corresponding information from the namespaces and sends it back through the incoming Mattermost webhook.

iii. RULES

- `monitoring_remediation`:
    - The trigger is a timer which executes every 1 hour.
    - The action executed is diagnostic_platform
- `krb5_renewal`:
    - The trigger is a timer which executes every 6 hours.
    - The action is a shell `kinit` command for the eosbot account.
- `log_cleaning:`
    - The trigger is a webhook which is called whenever a namespace runs out of disk space.
    - The action is executing a script which sorts the logs based on their creation dates and copies those older than a specified parameter to a backup destination.
- `exec_slash_commands` :
    - The trigger is a webhook which is called whenever an authenticated user enters a valid slash command.
    - The action is executing slash_commands with the corresponding parameters.



iv. ANOMALY DETECTION

The streaming anomaly detection implemented using Nupic is made accessible to Stackstorm through a REST API. Temporarily hosted on http://eosbackup-srv-w1.cern.ch, it is hosted using a Python Flask server and UWSGI. After each retrieval of information from the MGMs, an API call to the URL '/?instance={}&nodels_sum={}&timestamp={}' is made. This call first predicts whether the current statistic is an anomaly or not and then trains the existing models with the new information, does generating incrementally intelligent models. Thus, each call returns the confidence score of the current statistic being an anomaly. If this conference call is above a specified threshold then we report the incident to the mattermost channel and seek the help of the system administrator to adopt corrective measures. To tune the hyperparameters of these models, extensive swarming over a broad set of candidates was conducted, resulting in HTM models which best describe the pattern of the average load across the nodes of a namespace.

To extend the anomaly detection models to other training data the model definition and training procedures are in place. We just need to figure out other use cases for which early detection would have the most impact.

v. CHATOPS

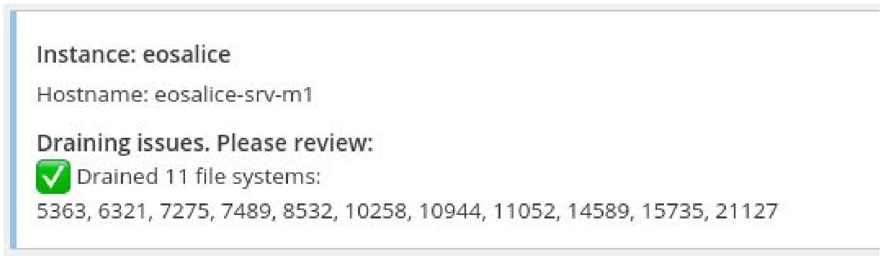
Chatops offers conversation driven development, allowing users to execute workflows as commands through Mattermost. It unifies the communication about what work should get done with the actual history of the work being done. Currently four chart are supported:

- `/eosversionsummary` : Get MGM nodes EOS versions summary.
- `/eosclientversionsummary` : Get clients EOS versions summary sorted into various categories.
- `/eosclientsummary` : Get MGM clients access summary.
- `/eosaccesssummary` : Get user limit access summary.

vi. IMPLEMENTED RUNBOOKS

Various runbooks have already been implemented and the business logic of others is in place. Adding new runbooks involves just defining a new python function and a corresponding YAML config file.

- Drain Expired:



Instance: eosalice

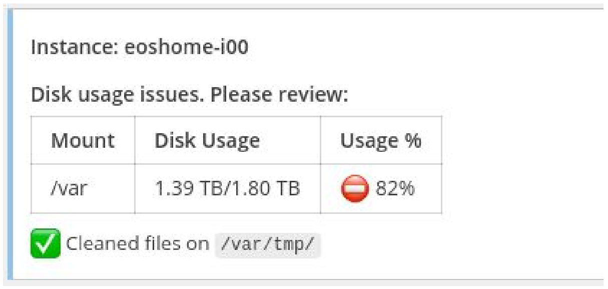
Hostname: eosalice-srv-m1

Draining issues. Please review:

✓ Drained 11 file systems:

5363, 6321, 7275, 7489, 8532, 10258, 10944, 11052, 14589, 15735, 21127

- /var usage > 80%:



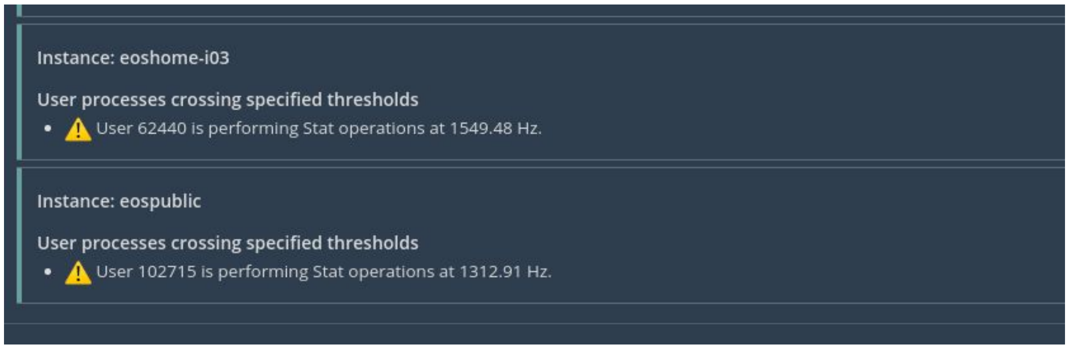
Instance: eoshome-i00

Disk usage issues. Please review:

Mount	Disk Usage	Usage %
/var	1.39 TB/1.80 TB	⛔ 82%

✓ Cleaned files on `/var/tmp/`

- Placing limits on user activity if continuously high activity by a user threatens to block other users:



Instance: eoshome-i03

User processes crossing specified thresholds

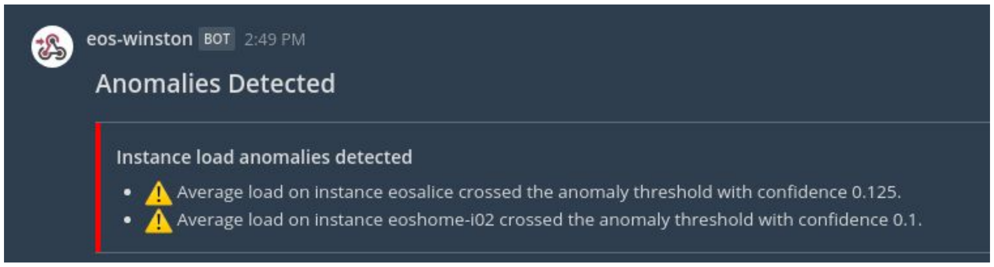
- ⚠ User 62440 is performing Stat operations at 1549.48 Hz.

Instance: eospublic

User processes crossing specified thresholds

- ⚠ User 102715 is performing Stat operations at 1312.91 Hz.

- Anomaly detection:



eos-winston BOT 2:49 PM

Anomalies Detected

Instance load anomalies detected

- ⚠ Average load on instance eosalice crossed the anomaly threshold with confidence 0.125.
- ⚠ Average load on instance eoshome-i02 crossed the anomaly threshold with confidence 0.1.

4. CHALLENGES FACED

- Issues with the HTCondor batch system: Getting Nupic to work involved executing a docker image on HTCondor. Despite its supporting docker, the system had problems configuring MYSQL, which was needed to enable model swarming.
- There were some installation issues with puppet, specifically involving the installation of repositories having a more updated version than the one supported by puppet.
- The nodes hosting Winston and the anomaly detection API all have self-signed certificates. Thus, these generate a warning on the client side when an HTTPS call is made to them, and Mattermost does not allow ignoring such warnings in slash command requests, while the Stackstorm webhook API is configured to run on HTTPS. So these had to be enabled for HTTP for the chatops integration.

5. FUTURE WORK

While the anomaly detection is currently enabled only for detecting unusually high average load across an instance, more use cases can be easily generated by identifying attributes which are known to have inconsistent peaks and troughs. Also joint models involving system performance and the user activity might prove to have interesting insights.

6. REFERENCES

[1]: https://iopscience.iop.org/article/10.1088/1742-6596/664/4/042042

[2]: https://iopscience.iop.org/article/10.1088/1742-6596/331/5/052015

[3]: https://www.ansible.com/

[4]: https://www.saltstack.com/

[5]: https://stackstorm.com/

[6]: https://github.com/numenta/nupic

[7]: https://numenta.com/neuroscience-research/research-publications/papers/hierarchical-temporal-memory-white-paper/