ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP045

MONITORING THE SKA INFRASTRUCTURE FOR CICD

M. Di Carlo*, M. Dolci, INAF Osservatorio Astronomico d'Abruzzo, Teramo, Italy
U. Yilmaz, P. Harding, SKA Observatory, Macclesfield, UK
P. Osório, Atlar Innovation, Portugal
J. B. Morgado, CICGE, Faculdade de Ciências da Universidade do Porto, Portugal

Abstract

The Square Kilometre Array (SKA) is an international effort to build two radio interferometers in South Africa and Australia, forming one Observatory monitored and controlled from global headquarters (GHQ) based in the United Kingdom at Jodrell Bank. The selected solution for monitoring the SKA CICD (continuous integration and continuous deployment) Infrastructure is Prometheus with the help of Thanos. Thanos is used for high availability, resilience, and long term storage retention for monitoring data. For data visualisation, the Grafana project emerged as an important tool for displaying data in order to make specific reasoning and debugging of particular aspect of the infrastructure in place. In this paper, the monitoring platform is presented while considering quality aspect such as performance, scalability, and data preservation.

INTRODUCTION

The Square Kilometre Array (SKA) project has selected SAFe (Scaled Agile Framework) as an incremental and iterative development process. A specialized team – named System Team – is devoted to support the Continuous Integration, Continuous Deployment (CI/CD) [1], test automation, and the project components' quality. Building an infrastructure to support the CI/CD together with a monitoring solution, was one of the first goals of the team.

The SKA infrastructure consists of a standard footprint of VPN/SSH gateway, monitoring, logging, storage and Kubernetes [2] services tailored to support the GitLab [3] runner architecture that is shown in Fig. 1. Furthermore, this infrastructure is used to support Development and Integration testing facilities for the project's many subsystems. The selected logging solution is Elasticsearch [4], storage is handled through Ceph [5], while Prometheus [6] handles monitoring (see the Prometheus section below), and the (central) artefact repository (CAR) is Nexus [7]. It is important to realize that only artefacts produced by GitLab pipelines that have been marked for a release (i.e. triggered by a git tag), are allowed to be stored on the CAR. On all other cases, GitLab's own artefact repository is used.

The infrastructure shown in Fig. 1 is replicated in multiple locations spread over different continents, with specific hardware on top of OpenStack [8] or bare metal/virtual machine instances. Meaning, that for every infrastructure, the same components will be present – at a different scale depending on the available resources. Some components, such as Elasticsearch and Thanos, exist on a single location, centralizing the access to information of several data centres to allow aggregated analysis across different datacentres while reducing the maintenance overhead.

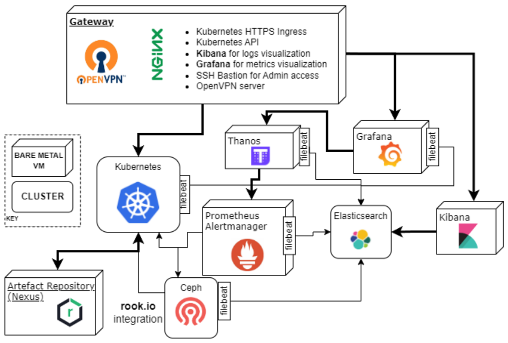


Figure 1: Simplified infrastructure.

PROMETHEUS

The selected monitoring solution, for every infrastructure, is Prometheus. It works in client-server architecture, where Prometheus acts as a client that reads (*scrapes* in its domain-specific language) timestamped information from multiple servers – called targets or exporters. The data is stored on a disk in TSDB format [9] and pushed to an object store. Figure 2 shows a detailed diagram that illustrates the monitoring architecture on SKA. The main components of the diagram are:

- **Prometheus server**, which is composed by the scraper, the TSDB storage, an HTTP API, and Web interface for data querying;
- **Thanos**, which is composed by many different components to help with high availability, data retention, retrieval and long term storage;
- **Jobs/exporters**, where Prometheus scrapes information as a time series;
- **Grafana**, as a data visualization and export tool that integrates with Prometheus/Thanos using PromQL – a specific query language;
- **Altermanager**, that delivers alarm notifications to end-user systems;

Each exporter provides a time series uniquely identified by its metric name and some optional key-value pairs – called labels. It is important to note that the exporter must give an

* matteo.dicarlo@inaf.it

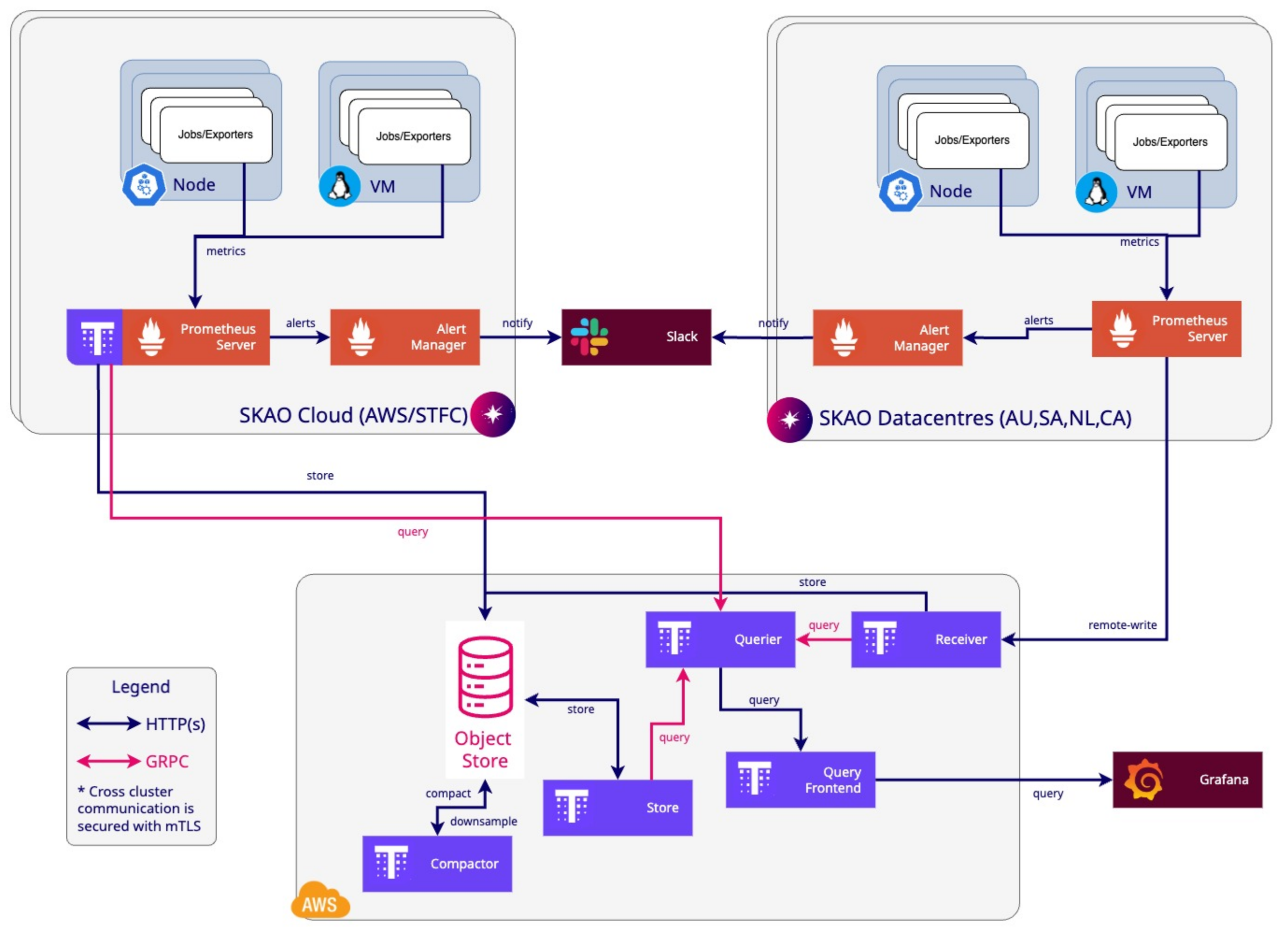


Figure 2: Detailed Monitoring Architecture.

extensive amount of information about the system that it is monitoring, usually under a single metrics endpoint.

Prometheus was adopted mainly due to its scalability, modifiability and optimal usage of resources – with memory allocation optimized to use the minimum amount required – as well as the very complete ecosystem of tools revolving around it. Because of the memory allocation optimizations, scaling is usually done vertically (i.e. adding more resources to the existing system) but it is also possible to have a federation of them, resembling horizontal scaling. This way, it is possible to have servers devoted to monitoring specific parts of the infrastructure, being a crucial aspect when dealing with multiple physical locations, and improving latency both when scraping and querying data.

Exporters

One of the most important quality of the Prometheus monitoring solution is its modifiability. This modifiability, makes it very easy to add new exporters since they are single-endpoint HTTP servers with well-structured data. The simplicity of exporter integration with the Prometheus solution is key to a consolidated monitoring platform. Aside from the extensive list of third-party exporters, it is possible to build custom ones (i.e. for an application), with multiple programming languages already having libraries to help on their development. This means we can monitor full-fledged clusters to simple parts of an application or a database, using the exact same resources and protocols.

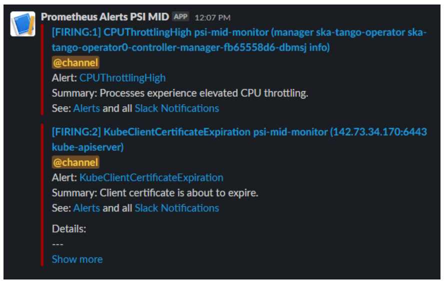
In the SKA, a set of well known exporters has been selected to be deployed – or enabled – on each node of the infrastructure. Most importantly, we make use of:

- **Node** exporter [10];
- **Container runtime** metrics: enabled depending on the runtime (containerd, docker or podman);
- **Kubernetes** [2] exporters using kube-state-metrics [11];
- **Elasticsearch** metrics [4];
- **Ceph** metrics [5];

The configuration of the exporters on the Prometheus server, happens with the help of a Python script (prom-helper) [12]. The script runs discovery queries on every node of the infrastructure inventory [13], creating JSON configuration files (targets in Prometheus domain-specific language) for each exporter type, including the discovered targets. Prometheus runs as a containerized application, using persistent storage for the data and configurations.

Alerts

A critical part of the monitoring solution selection is the ability to alarm the maintainers (System Team) of any problems in the infrastructure. Alertmanager and Prometheus work together to deliver detailed alarms on the multiple monitored infrastructures to communication channels. As multiple infrastructures are monitored, it is crucial that the alarm system allows for the de-duplication and grouping of alarms, as well as the capability to deliver the notifications to different communication channels. Community-developed alarming standards were adopted, as the *kubernetes-mixin* [14], enabling an in-depth alarming system of open-source products. Also, some custom exporters were developed to expose metrics on custom-built Kubernetes Operators. Figure 3 shows an example alarm notification.


Prometheus Alerts PSI MID APP 12:07 PM
[FIRING:1] CPUThrottlingHigh psi-mid-monitor (manager ska-tango-operator ska-tango-operator0-controller-manager-fb65558d6-dbmsj info)
@channel
Alert: CPUThrottlingHigh
Summary: Processes experience elevated CPU throttling.
See: Alerts and all Slack Notifications

[FIRING:2] KubeClientCertificateExpiration psi-mid-monitor (142.73.34.170:6443 kube-apiserver)
@channel
Alert: KubeClientCertificateExpiration
Summary: Client certificate is about to expire.
See: Alerts and all Slack Notifications

Details:

Show more

Figure 3: An alert message in Slack [15] for the Mid Prototype System Integration (PSI MID) cluster.

THANOS

Thanos is an open source project that provides global query views, high availability and long term data storage with historical querying capabilities [16]. Thanos is able to store data from multiple Prometheus instances, allowing queries to target multiple data-sources. It has several components targeted towards data replication, storage integration and centralised load-balanced query functions with support for encrypted data over public network – (see Fig. 2) for some highlights on how they are used in SKAO. The details of each thanos component is highligted in Fig. 4. Its main components and how they are used on SKA are explained below.

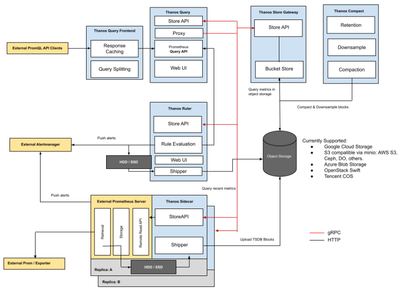


Figure 4: Thanos Architecture.

- **Thanos Sidecar** is the main component. It queries Prometheus, makes it available for querying, and uploads TSDB (Time series database) blocks to Thanos' target object store. This is deployed next to each Prometheus instance.
- **Thanos Receiver** is used if the sidecar cannot be deployed. This is mostly due to network rules in the datacentre. It accepts data from Prometheus using remote-write API, makes it available for querying, and uploads TSDB (Time series database) blocks to Thanos' target object store. This can be deployed anywhere.
- **Thanos Store Gateway** provides a store API on top of the historical data present on the object store;
- **Thanos Ruler** is the equivalent of Prometheus Ruler. Used to manage Thanos' rules. Note that this is not used in SKAO as the alerts are differentiated by their own alertmanagers in each datacentre;
- **Thanos Compactor** is responsible for down-sampling and data retention for efficient long term storage;
- **Thanos Querier** aggregates, de-duplicates the underlying data, and implements the Prometheus API;
- **Thanos Query Frontend** is a proxy to Thanos Queries, that improves range queries to the API by splitting and caching previous results;

The ability to use push or pull methods to collect Thanos Sidecar data, allows for different infrastructural and security requirements to be met, while maintaining the capability of accessing the data outside of premises. Being the SKA a globally distributed project, it is crucial to be able to access the data from multiple global locations, empowered by the replications features of the object store. Furthermore, the setup and access of user-level tools, like Grafana, to the stored data, is greatly simplified by these capabilities while the user experience is improved by using caching techniques while allowing aggregated dashboards to perform analysis from an single view.

GRAFANA

Prometheus API, uses the HTTP protocol, with a well-structured and simple format. Thus, it becomes simple to integrate with other systems using PromQL, which is particularly important for data analysis and visualization. The de facto visualization tool is Grafana [17]. which, by itself, can also have multiple data-sources other than Prometheus. It is remarkable the modularity and completeness of the Prometheus ecosystem as a fully-featured and open-source monitoring solution. Visualization is organized in dashboards, composed of reusable display panels. Dashboards usually target a particular source type (e.g., node metrics, kubernetes metrics), but the reusable panels can be used to build a global overview dashboard of the whole infrastructure. From an architectural point of view, it is built with a plugin architecture where a plugins can be:

- **Panels** (the basic visualization building block in Grafana), that transform raw data from a data-source into complex visualizations (e.g., maps, clocks, pie-charts);
- **Data-sources** that target external systems, retrieving the data from the upstream data-source, and reconciling the differences between the external and Grafana's data model;
- **Apps** that combine both data-sources and panels to visualize in a cohesive manner.

This architecture can be personalized extensively, correlating different data-sources, giving developers and testers the ability to diagnose system-wide problems. Figure 5 shows the resulting model in UML. Grafana represents an aggregation of plugins and multiple dashboards (a collection of panels). Data-sources can be Thanos or Prometheus instances that themselves collect data on several other data-sources, or directly other external, whenever historical storage is not required. In SKA, the TANGO Archiver [18] database or the Elasticsearch cluster are monitored this way.

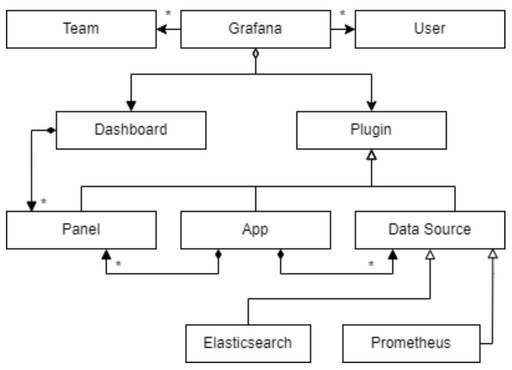


Figure 5: Grafana UML model.

DASHBOARDS

The core tools and services that support the SKA infrastructure have community-developed dashboards, as it has alarming standards, displaying information that empower health and performance analysis on the entire CI/CD infrastructure. In this section we present some of the dashboard that were adopted for the various data-sources: Figs. 6, 7, and 8 show some dashboards adopted from the Grafana community:

- **Node** exporter [19],
- **Elasticsearch** cluster [20],
- **Docker** containers [21],

Figure 9 shows a dashboard provided by the kubernetes-mixin [14] project.

Having alarms and visualizations on the same data, it becomes possible to track down causes for issues on

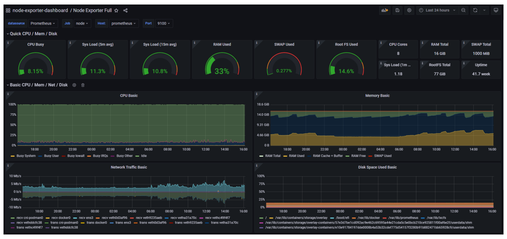


Figure 6: Node exporter dashboard.

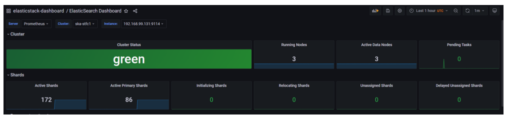


Figure 7: Elasticsearch dashboard for cluster health.

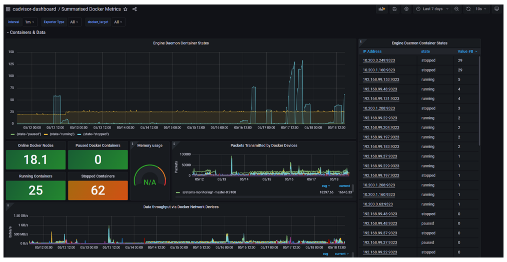


Figure 8: Docker metrics dashboard for container health.

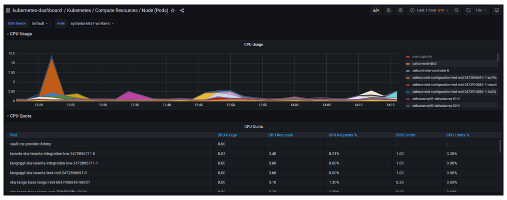


Figure 9: Kubernetes pods dashboard per node.

pieces of the infrastructure that might not apparently relate. Application-level components may malfunction due to system-level components misbehaving, and the capability to display performance-level metrics of multiple systems, can helps shed light on root-causes.

GLOBAL VIEW OF INFRASTRUCTURE OPERATIONS

The SKA has the particularity of being a globally-distributed project, as data might be needed in a different part of the world than it was collected or stored. Therefore, it becomes paramount that the monitoring system allows to deliver data across the globe, efficiently. Looking at the monitoring system as three main components - collection, storage and access/querying - the selected tools allow to distribute the workload to realize them in a global scale, maintaining high-performance on all planes.

- **Prometheus** collects metrics on several systems, locally or within public realms;

- **Alertmanager** delivers real-time alarm notifications on monitored systems, to multiple communication platforms;
- **Thanos** collect metrics from several infrastructures, consolidating the data for long-term storage securely. It also allows data to be replicated across locations and accessed with low latency;
- **Grafana** allows data to be visualized and correlated data between different systems of the same (or other) infrastructures;

Together these tools allow a global oversight on multiple pieces of infrastructure, while allowing for different infrastructural requirements to be met and integrated in a centralized (yet distributed) monitoring solution.

CONCLUSION AND FUTURE WORK

This paper presented an overview of the tools selected by the SKA project in order to monitor the performance of the infrastructure put in place for CI/CD purposes. In specific Prometheus with Thanos and Grafana have been selected for this scope. The main rationale for this choice is the modularity of the several tools withing the Prometheus ecosystem (custom exporters, data-sources and visualizations), together with performance efficiency (optimal usage of memory allocation) and scalability (vertical only at the moment). Another important aspect to consider in the selection, is the outstanding community around them, keeping them active a feature-rich.

The metrics gathered and displayed on the dashboards are available for a long periods of time (3 months at the moment) and are available from different parts of world to developers, scientists and engineers.

ACKNOWLEDGEMENTS

This work has been supported by Italian Government (MEF - Ministero dell'Economia e delle Finanze, MIUR - Ministero dell'Istruzione, dell'Università e della Ricerca).

REFERENCES

[1] M. Di Carlo *et al.*, "Ci-cd practices at SKA", *Proc. SPIE, Software and Cyberinfrastructure for Astron. VII*, vol. 12189, Aug. 2022. doi:10.1117/12.2620526

[2] Kubernetes, https://kubernetes.io/

[3] Gitlab, https://about.gitlab.com/

[4] Elasticsearch, https://www.elastic.co/

[5] Ceph storage, https://ceph.io/

[6] Prometheus, https://prometheus.io

[7] Nexus, https://www.sonatype.com/products/sonatype-nexus-repository

[8] OpenStack, https://www.openstack.org/

[9] Tsdb format, https://github.com/prometheus/prometheus/tree/release-2.22/tsdb/docs/format

[10] Node exporter, https://github.com/prometheus/node_exporter

[11] kube-state-metrics, https://github.com/kubernetes/kube-state-metrics

[12] Prometheus Helper Script, https://gitlab.com/ska-telescope/sdi/ska-ser-ansible-collections/-/blob/main/ansible_collections/ska_collections/monitoring/roles/prometheus/files/helper/prom_helper.py?ref_type=heads

[13] Ansible, https://www.ansible.com/

[14] Prometheus monitoring mixin for kubernetes, https://github.com/kubernetes-monitoring/kubernetes-mixin

[15] Slack, https://slack.com

[16] Thanos, https://thanos.io

[17] Grafana, https://grafana.com/

[18] Tango Archiver, https://gitlab.com/ska-telescope/ska-tango-archiver

[19] Node exporter full, https://grafana.com/grafana/dashboards/1860-node-exporter-full/

[20] Elasticsearch dashboard, https://grafana.com/grafana/dashboards/878-elasticsearch-dashboard/

[21] Docker and system monitoring, https://grafana.com/grafana/dashboards/893-main/