INTEGRATION OF BESPOKE DAQ SOFTWARE WITH TANGO CONTROLS IN THE SKAO SOFTWARE FRAMEWORK — FROM PROBLEMS TO PROGRESS

A. J. Clemens[*], Observatory Sciences Ltd., Cambridge, United Kingdom
D. Devereux[†], CSIRO, Australia
A. Magro[‡], Institute of Space Sciences and Astronomy University of Malta, Malta

Abstract

The Square Kilometre Array Observatory (SKAO) project is an international effort to build two radio interferometers in South Africa and Australia to form one Observatory monitored and controlled from the global headquarters in the United Kingdom at Jodrell Bank. The Monitoring, Control and Calibration System (MCCS) is the "front-end" management software for the Low telescope which provides monitoring and control capabilities as well as implementing calibration processes and providing complex diagnostics support.

Once completed the Low telescope will boast over 130,000 individual log-periodic antennas and so the scale of the data generated will be huge. It is estimated that an average of 8 terabits per second of data will be transferred from the SKAO telescopes in both countries to Central Processing Facilities (CPFs) located at the telescope sites.

In order to keep pace with this magnitude of data production an equally impressive data acquisition (DAQ) system is required. This paper outlines the challenges encountered and solutions adopted whilst incorporating a bespoke DAQ library within the SKAO's Kubernetes-Tango ecosystem in the MCCS subsystem in order to allow high speed data capture whilst maintaining a consistent deployment experience.

INTRODUCTION

The Square Kilometre Array Observatory (SKAO) represents a significant advancement in our pursuit of understanding the universe through radio astronomy. This scientific endeavor requires a complex toolchain in which each component plays a crucial role. At the core of this toolchain lies Docker [1], a well-known technology for containerization, which serves as a key element for packaging and deploying various components of the SKAO project. Additionally, Minikube [2] and Kubernetes [3] take on important roles, managing local development environments and ensuring robust production-level deployments to support the scalability and resilience necessary for a project of this scale. Helm [4], with its templating capabilities, simplifies the deployment process, while Make acts as a unifying force to streamline the intricate interactions between these components, ensuring an efficient deployment workflow.

[*] ajc@observatorysciences.co.uk
[†] drew.devereux@csiro.au
[‡] alessio.magro@um.edu.mt

Amidst this technological tapestry a significant challenge emerged: the seamless integration of third-party software, including xGPU [5] and NVIDIA's CUDA [6] (Compute Unified Device Architecture) along with their dependencies into the framework of Tango Controls [7]. Tango Controls, the chosen control framework for the SKAO project, forms the backbone upon which our astronomical endeavors rely.

This paper documents the journey undertaken to bridge the gap between the SKAO deployment toolchain and third-party data acquisition (DAQ) software [8]. Its purpose is to provide a comprehensive account of the strategies employed, the complexities encountered, and the solutions devised during this process. The goal is to offer valuable insights, not just as a record of achievements but as a resource for engineers and scientists facing similar integration challenges.

We will explore two distinct phases: the first phase "The Quest for Data" delves into the incorporation of third party software into the SKAO's data acquisition system. The subsequent phase "The Correlator Saga" explores the challenges and solutions encountered in the integration of these critical components.

THE QUEST FOR DATA

In our pursuit of data acquisition the first milestone was the creation of a containerized Tango device server to drive the DAQ (data acquisition) software. This step laid the foundation for our data acquisition endeavors within the project. As we embarked on this quest we encountered a series of formidable challenges, each demanding resourcefulness and persistence to surmount.

Inheriting Capabilities

The initial challenge arose from the limitations of configuring capabilities solely within the Dockerfile. While the container itself possessed the necessary capabilities, a crucial nuance emerged: the Kubernetes pods failed to inherit these essential capabilities.

To address this challenge we informed Helm about the specific capabilities required by appending them to the `securityContext::capabilities::add` field within the values file. This ensured that Kubernetes pods inherited the critical capabilities, aligning both the container and pod with the requisite permissions.

Selective Capability Application

Expanding upon the prior solution, the challenge of selective capability application came to the forefront. Despite the initial solution's effectiveness in ensuring proper permissions for the device server, it unintentionally introduced unnecessary capabilities to all other device servers within the team's ecosystem.

In response, we opted for a more targeted approach. To selectively apply capabilities only where necessary we established a dedicated DAQ repository complete with its own OCI (Open Container Initiative) image. This strategic isolation allowed us to maintain precise control over capability assignment, mitigating the undesired side effects.

Data Reception

As we ventured further into the quest for data, a perplexing issue surfaced. Despite specifying `EXPOSE <port>/udp` in the Dockerfile our device server was not receiving data as anticipated. An important, yet surprising, revelation was that the `EXPOSE` command in Dockerfiles held no direct influence on network functionality.

In response to this challenge, refinements were made to our Helm templates. These templates were adjusted to incorporate enhancements that would generate a load balancer service for each receiver in order to expose an external IP address to which data could be routed. This adjustment ensured that each service efficiently routed traffic to the appropriate port of its respective receiver's pod, thereby allowing data ingress.

Network Interface Access

Persisting on our journey, we encountered an enduring challenge - reliable data reception from external sources within the cluster remained elusive. This was because because Minikube, our local development Kubernetes cluster, lacks the capability to grant raw network interface access, resulting in complete packet loss at the cluster boundary.

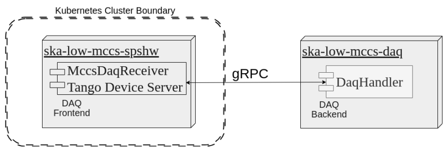

Figure 1: First Evolution of DAQ Architecture.

In reaction to this challenge we initiated a transformation of the DAQ device's architecture (Fig. 1). The device was partitioned into two distinct components: a frontend Tango device server and a backend DAQ server. These components communicated via gRPC [9] facilitated by a Kubernetes service. This strategic division afforded us the flexibility to deploy the frontend Tango device server within the cluster using familiar SKAO tools. Simultaneously, the backend DAQ server could be manually deployed externally to the cluster using Docker. This external deployment allowed us to grant the requisite access to raw network interfaces thus resolving the packet loss issue at the cluster boundary.

Transition to Hardware Sites

Our path toward data acquisition reached an important crossroads as we transitioned from successful data capture in Minikube using simulated data to hardware sites with a complete Kubernetes setup. Old challenges resurfaced, primarily concerning gaining access to the host's network interfaces.

To address this we embarked on the incorporation of a Container Network Interface (CNI) meta-plugin known as MULTUS into our setup. MULTUS was an important piece in our solution by enabling the simultaneous loading of our primary CNI, Calico, and subsequently granting a pod access to an additional network interface by moving it into the pod's network namespace. This strategic integration empowered our data capture efforts at hardware sites, demonstrating our adaptability and resolve.

In our pursuit of data acquisition, each challenge became an opportunity for innovation. These hurdles, though formidable, were navigated with determination and resourcefulness. Our commitment to overcoming these obstacles reflects our dedication to advancing radio astronomy, one step at a time.

THE CORRELATOR SAGA

As we progressed further into our integration journey, a new chapter unfolded - the compilation and integration of the correlator, a critical component in the SKAO's data processing arsenal. This chapter, marked by its own set of challenges and innovative solutions, showcases our unwavering commitment to building a robust and adaptable system.

Compatibility Complexities

Creating an OCI image compatible with the SKAO toolchain, Tango Controls and CUDA, along with their intricate dependencies proved to be a formidable puzzle. Complex compatibility matrices and version disparities hindered development, leaving us searching for an alternative approach.

In light of this problem, we embarked on a strategic pivot to decouple the Tango Controls and CUDA requirements (Fig. 2). The first step involved relocating the frontend Tango device server from the DAQ repository, effectively removing the Tango dependency. This architectural shift allowed us to employ an official NVIDIA base image, pre-loaded with CUDA and its essential dependencies. Simultaneously, we separated the communication protocol between the DAQ frontend and backend into its own dedicated repository. This deliberate separation ensured protocol flexibility, preserving adaptability in case of a transition from gRPC to other communication methods. This streamlined deployment approach paved the way for uniform deployment of both DAQ halves, each originating from their respective repositories and eliminating the requirement of deploying the backend externally to the cluster with Docker.

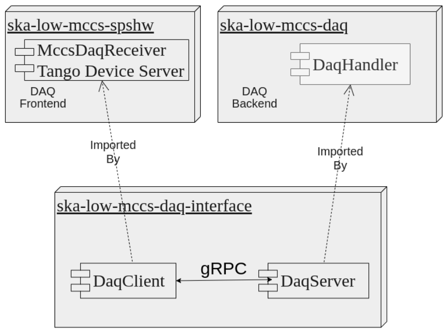

Figure 2: Final Evolution of DAQ Architecture.

Unlocking GPU Resources

Upon deployment, the DAQ server backend encountered initial difficulties in establishing communication with the available GPU resources on the host. This challenge stemmed from intricacies in deployment configuration and the overall cluster configuration.

To surmount this challenge, a multi-faceted solution emerged. The first critical requirement was to ensure that a GPU was not only available on the host machine but also registered with the cluster as an allocatable resource. This registration enabled the GPU to be requested by a pod under Helm's `resources::limits::nvidia.com/gpus` key. Furthermore, the host machine needed to have the correct GPU driver installed, as well as the NVIDIA Container Toolkit which provides a specialised container runtime. To ensure that the Docker daemon was able to provide GPU hardware access to applications running in containers we utilized the NVIDIA Container Runtime and specifying it in the Helm template's `runtimeClassName` field. This ensured that the DAQ server backend could harness the power of available GPU resources, unlocking the potential of the correlator.

In the unfolding saga of the correlator integration, each challenge met and overcome reinforces our commitment to building a resilient and adaptable system. These solutions reflect our dedication to precision and innovation as we push the boundaries of technology in the pursuit of groundbreaking radio astronomy research.

CONCLUSION

The journey we embarked upon, as recounted in this paper, is not merely a documentation of technical challenges and their solutions but a testament to the resilience, innovation and unwavering dedication of the teams behind the SKAO project. In our quest to integrate third-party DAQ software into the Monitoring, Control, and Calibration System, we encountered formidable hurdles, each requiring a unique approach and a steadfast commitment to success.

The integration of third-party software and its dependencies demanded ingenuity and adaptability. Challenges, such as configuring capabilities and managing compatibility between Tango Controls and CUDA prompted us to devise innovative solutions. These solutions, from refining Helm templates to streamlining deployment pipelines and separating communication protocols, not only addressed immediate obstacles but also laid the foundation for a more robust and adaptable system.

As we delved deeper into data acquisition and the complexities of GPU integration within the correlator, our resolve remained unshaken. We adapted to new realities, ensuring that the system would be well equipped to process and analyze the vast volumes of data it will soon encounter. Our journey was marked by the pursuit of excellence and the relentless pursuit of solutions that align with the goals of this groundbreaking project.

While this paper illuminates the path we traversed, it it important to recognize that our work is part of a much larger tapestry of scientific discovery. The SKAO project represents a testament to human curiosity and ambition as we endeavor to unlock the mysteries of the universe. Our contributions, though significant, are but a single thread in this grand narrative, and we stand alongside countless others who share our passion for exploration.

In conclusion, the challenges we faced and the solutions we crafted are a reflection of our commitment to advancing radio astronomy and technology integration. They exemplify the spirit of collaboration and innovation that drives projects of this magnitude. The SKAO is poised to make groundbreaking discoveries, and we are proud to have played a role in laying the technical groundwork for its success. As we look to the stars, we are reminded that the pursuit of knowledge knows no bounds and the SKAO stands as a testament to the power of human ingenuity and the enduring quest for understanding.

REFERENCES

[1] Docker, `https://www.docker.com`

[2] Minikube, `https://minikube.sigs.k8s.io`

[3] Kubernetes, `https://kubernetes.io`

[4] Helm, `https://helm.sh`

[5] M. A. Clark, P. C. La Plante, and L. J. Greenhill, "Accelerating Radio Astronomy Cross-Correlation with Graphics Processing Units", in *Int. J. High Perform. Comput.*, vol.27, 2013, pp. 178–192. `doi:10.1177/10943420124447`

[6] CUDA, `https://developer.nvidia.com/cuda-zone`

[7] Tango Controls, `https://www.tango-controls.org`

[8] A. Magro, K. Bugeja, R. Chiello, and A. DeMarco, "A High-Performance, Flexible Data Acquisition Library for Radio Instruments", in *2019 IEEE-APS Top. Conf. Antennas Propag. Wirel. Commun. (APWC)*, Granada, Spain, 2019, pp. 069-074. `doi:10.1109/APWC.2019.8870490`

[9] gRPC, `https://grpc.io`