Integration Of Bespoke DAQ Software With Tango Controls In The SKAO Software Framework

From Problems to Progress

A. J. Clemens, Observatory Sciences Ltd., Cambridge, U.K.

ABSTRACT

The Square Kilometer Array (SKA) 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 based 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 SKA 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 poster outlines the challenges encountered and solutions adopted whilst incorporating a bespoke DAQ library within the SKA's Kubernetes-Tango ecosystem in the MCCS subsystem in order to allow high speed data capture whilst maintaining a consistent deployment experience.

CONTACT

A. J. Clemens
Observatory Sciences Ltd.
Email:
ajc@observatorysciences.co.uk
Website:
www.observatorysciences.co.uk

Introduction

The SKA deployment toolchain encompasses a comprehensive suite of technologies:

- Docker
- Kubernetes/Minikube
- Helm
- Make

The challenge was to integrate third party software and its dependencies with this ecosystem in a Tango Controls based framework.

The Quest for Data

The initial phase in integrating the DAQ software with MCCS involved the creation of a containerized Tango device server to drive the DAQ software.

To achieve high-rate data logging the DAQ software requires additional capabilities which includes raw network interface access among others. To enable these essential capabilities in our OCI image we can utilize the `RUN setcap <capabilities>` within our Dockerfile.

CHALLENGE: Configuring capabilities solely in the Dockerfile is insufficient when deploying to Kubernetes. Whilst the container itself receives the necessary capabilities, the pod does not inherit them.

SOLUTION: Inform Helm about the capabilities the pod requires by appending them to the `securityContext` field in the values file.

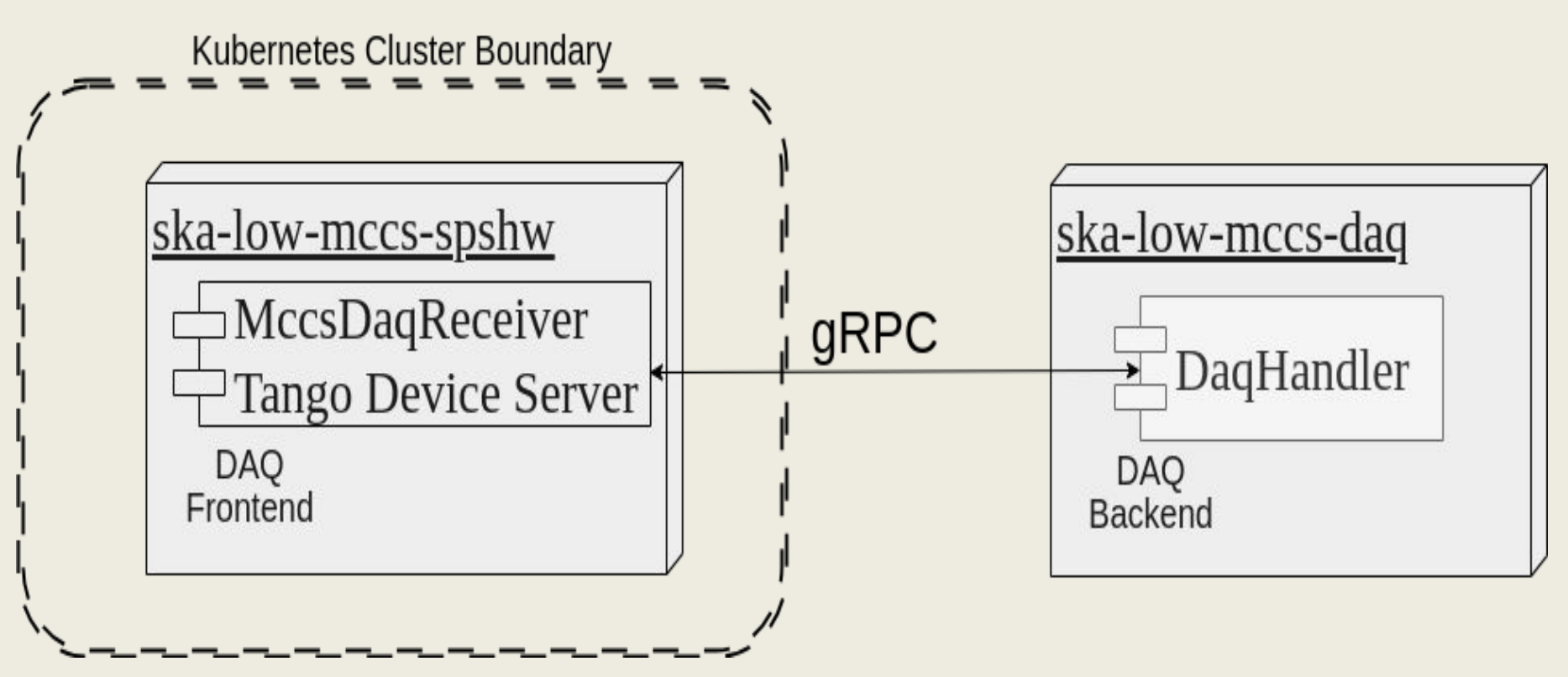
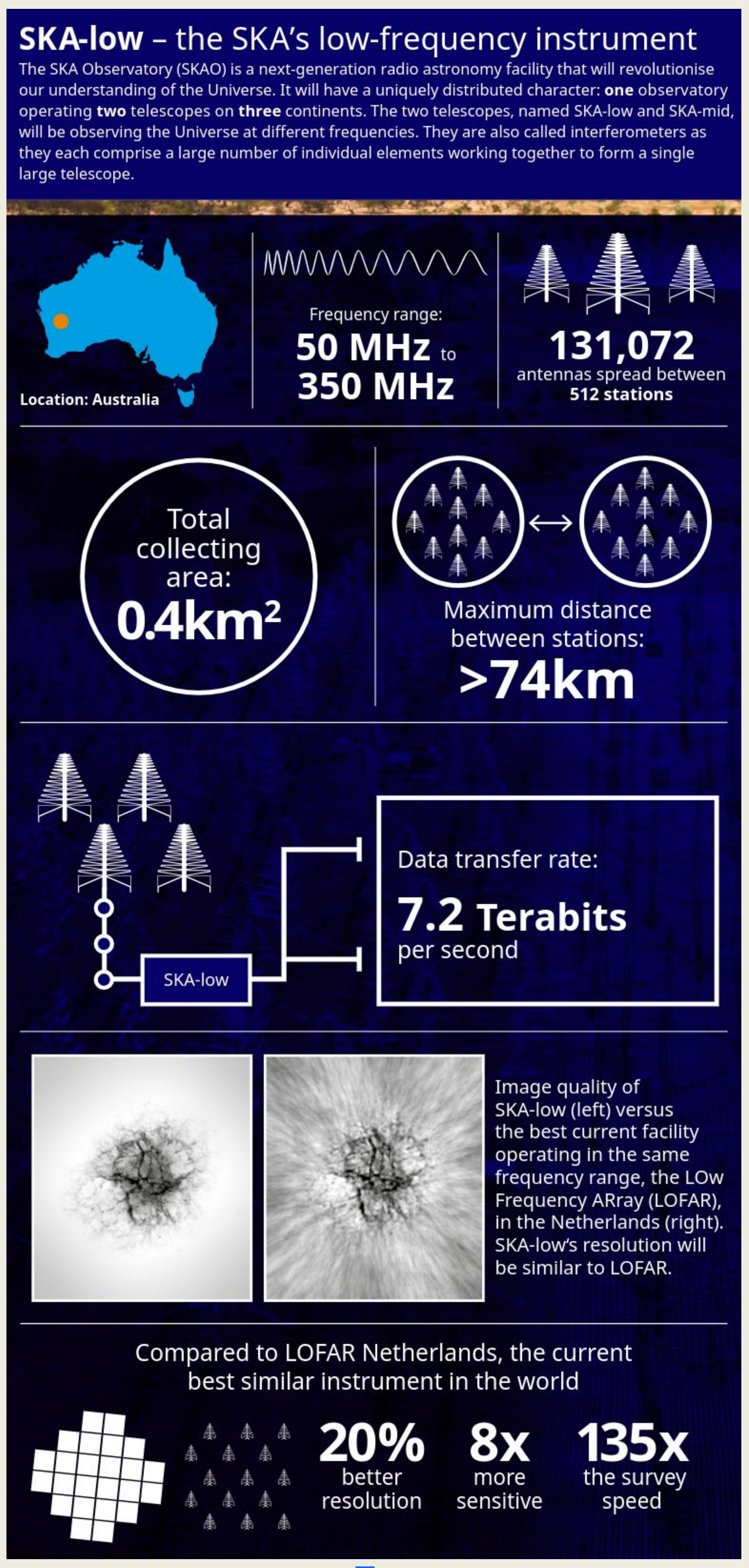


Figure 1. First Evolution of DAQ Architecture

CHALLENGE: Running the device server with correct permissions introduced an unintended issue - unnecessary capabilities were added to all team device servers.

SOLUTION: To selectively apply capabilities only where necessary, we opted to isolate this particular device server within a dedicated DAQ repository, complete with its own OCI image.



SKA-low – the SKA's low-frequency instrument

The SKA Observatory (SKAO) is a next-generation radio astronomy facility that will revolutionise our understanding of the Universe. It will have a uniquely distributed character: **one** observatory operating **two** telescopes on **three** continents. The two telescopes, named SKA-low and SKA-mid, will be observing the Universe at different frequencies. They are also called interferometers as they each comprise a large number of individual elements working together to form a single large telescope.

Image quality of SKA-low (left) versus the best current facility operating in the same frequency range, the LOw Frequency ARray (LOFAR), in the Netherlands (right). SKA-low's resolution will be similar to LOFAR.

CHALLENGE: Despite specifying `EXPOSE <port>/udp` in the Dockerfile our device server was not receiving data as anticipated. Surprisingly the command does not directly affect network functionality.

SOLUTION: We refined the Helm templates to generate a load balancer service for each receiver, empowering it to efficiently route traffic to the appropriate port of its respective receiver's pod.

CHALLENGE: Despite having a data routing service in place receiving data proved challenging. This persisted because Minikube lacks the ability to grant raw network interface access resulting in packet loss at the cluster boundary.

SOLUTION: We split DAQ into a frontend and a backend connected via gRPC and a Kubernetes service. This allowed deployment of the frontend with SKAO tools and the backend via Docker for raw network interface access. (See Figure 1)

CHALLENGE: When transitioning from successfully capturing simulated data in Minikube to hardware sites with a complete Kubernetes setup we continued to face network interface access challenges.

SOLUTION: A Container Network Interface (CNI) meta-plugin, MULTUS, enabled us to load our primary CNI, Calico, and grant pods access to an additional network interface thus facilitating the capture of data.

The Correlator Saga

At this stage of development our DAQ device had the capability to capture data in all modes with the exception of correlated data which had additional dependencies including an NVIDIA GPU, CUDA and xGPU support.

CHALLENGE: Creating a compatible OCI image for both Tango Controls and CUDA proved highly challenging. Complex compatibility matrices and version disparities hindered development, highlighting the need for an alternative approach.

SOLUTION: To decouple these requirements, we split the DAQ device further. We eliminated the Tango dependency by moving the Tango device server out of the DAQ repository allowing use of an official CUDA base image.

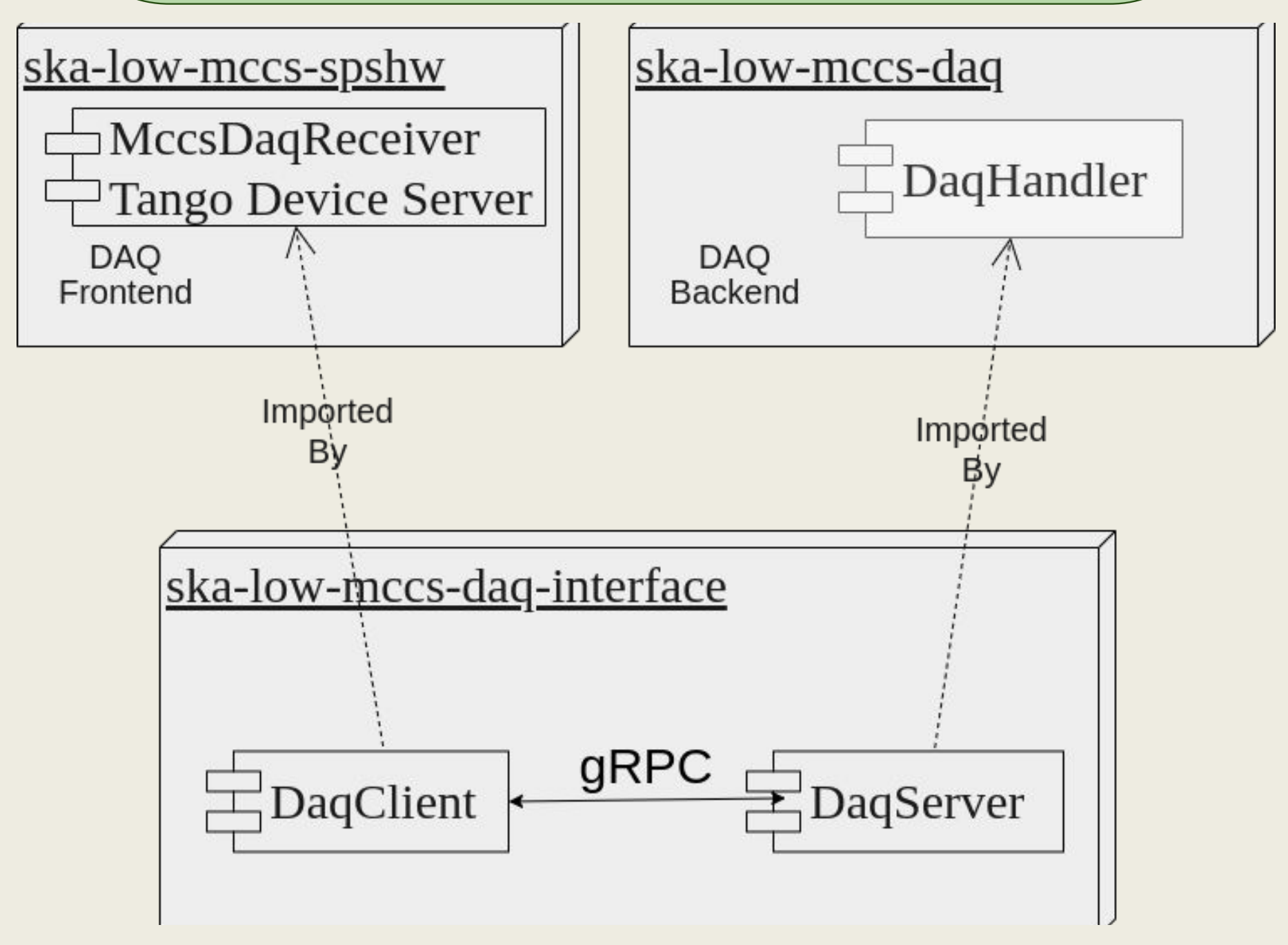


Figure 2. Final Evolution of DAQ Architecture

CHALLENGE: Upon deployment the DAQ server initially struggled to communicate with the host's GPU due partly to deployment configuration and partly to the overall configuration of the cluster.

SOLUTION: Register the GPU as a cluster resource and request via Helm's `resources::limits::nvidia.com/gpus` key and ensure the host has the right driver version, NVIDIA Container Toolkit and Docker daemon using the NVIDIA Container Runtime.