

- deploy the code into a node (i.e., Host, VM, container, etc...);
- configure the application before running (i.e., set environment variables, configuration files, etc...);
- configure any needed resources like persistent storage or network aspects;
- declare the device(s) in the TANGO database of devices;
- set any needed properties in the TANGO database of devices (i.e., polling definition for events, etc...);
- start devices in order (if there are dependencies, wait for each of them to be fulfilled before proceeding).

Without container orchestration, the automation of the above steps can be complex, depending on the software automation tool used. With the help of k8s, it is possible to package the TANGO-controls framework into a set of base container images so that the final product is a containerised application that will run in a container orchestrator. In SKA, the repository *ska-tango-images* [8] contains the definitions of all TANGO-controls components in a set of container images together with two helm charts: *ska-tango-base* and *ska-tango-util*. The first one defines the basic TANGO ecosystem – consisting mainly of the TANGO database – while the second one is a helm library chart which helps developers define TANGO device servers through configurable template macros. Figure 2 shows how to define a device server using the *ska-tango-util* helm chart library – please note that only a partial set of parameters are shown.

Declaring a DS in k8s

- Name is the k8s name of the resources
- The list of dependencies: devices or simple host/port
- Command or entry points of the device servers (if more than one entry point is specified, we are referring to a multi-device DS)
- The server definition, containing the list of instances, devices and classes to be ran
- The container image to use
- How/when to check if the device server is ready or in failure

```
name: "timer-dev-server"
function: description: test
domain: description: test
instances: ["timer"]
entrypoints:
- device: tps/database/2
  device: tps/factor/2
server:
name: "server-name"
instances:
- name: "a"
  classes:
  - name: "Classname"
  devices:
  - name: "test/device/1"
image:
registry: artefact.skao.int
image: ska-tango-examples
tag: 0.4.24
pullPolicy: IfNotPresent
livenessProbe:
initialDelaySeconds: 0
periodSeconds: 10
timeoutSeconds: 3
successThreshold: 1
failureThreshold: 3
readinessProbe:
initialDelaySeconds: 0
periodSeconds: 10
timeoutSeconds: 3
successThreshold: 1
failureThreshold: 3
```

Figure 2: Declarative definition of a device server.

The following code shows an example of a device server containing two instances (basically two processes) starting from the same container image.

```
name: timer-dev-server
function: timer
domain: timer-app
instances: ["counters", "timer"]
image:
  registry: artefact.skao.int
```

```
image: ska-tango-examples
tag: 0.4.24
pullPolicy: IfNotPresent
entrypoints:
- name: "Timer.Timer"
  path: "/app/src/Timer.py"
- name: "Counter.Counter"
  path: "/app/src/Counter.py"
server:
instances:
- name: "counters"
classes:
- name: "Counter"
devices:
- name: "srv/counter/minutes"
- name: "srv/counter/seconds"
- name: "timer"
classes:
- name: "Timer"
devices:
- name: "test/timer/1"
properties:
- name: "LOGGING_LEVEL"
  values:
  - "DEBUG"
livenessProbe:
  initialDelaySeconds: 0
  periodSeconds: 10
  timeoutSeconds: 3
  successThreshold: 1
  failureThreshold: 3
readinessProbe:
  initialDelaySeconds: 0
  periodSeconds: 10
  timeoutSeconds: 3
  successThreshold: 1
  failureThreshold: 3
```

Given the above definition, the helm chart library - *ska-tango-util* - will translate it into the following k8s resources:

- a k8s job for the initialization of the TANGO database;
- a service that exposes the application to the network;
- a Statefulset (in k8s this is an object used to manage stateful applications) which contains:
 - a Pod (k8s object that groups one or more containers into a unit);
 - one or more InitContainers (specialized containers that run before app containers in a Pod) to resolve the device server dependencies in order to:
 - * wait for the configuration job to complete and
 - * wait for device dependencies.

Figure 3 shows the workflow of the start of a device server in k8s, starting from the `install` command.

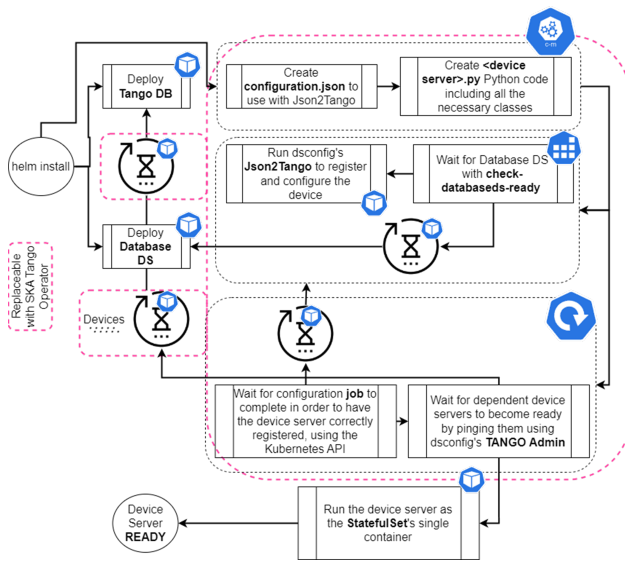


Figure 3: Workflow of Device Server installation using ska-tango-util Helm chart.

EXTENDING K8S - THE OPERATOR PATTERN

The above solution to set up a Device Server in Kubernetes works, but also comes with some challenges. Particularly, we can highlight:

- creates extra resources to setup the Device Server (i.e., the InitContainers to wait for the dependencies);
- leaves behind spent resources in the API (i.e., Job pods that have completed do not get deleted immediately);
- CrashLoopBackoff (happens when a container fails to start, taking exponentially longer to restart the failed container) of InitContainers can compound, making the startup of an environment with multiple Device Servers take very long;
- wastes API resources with all the pooling that needs to happen to wait for dependencies, either the TANGO database or other devices;
- the cluster might end up with multiple versions of Kubernetes resources created by ska-tango-util, as there is no centralised control over the user chart versions.

To solve these challenges, it is possible to use the k8s operator pattern, which aims to capture the key aim of a human operator who is managing a service or set of services. Human operators who look after specific applications and services have deep knowledge of how the system ought to behave, how to deploy it, and how to react if there are problems [9].

Technically speaking, the operator:

- extends the Control Plane to describe custom behaviours;
- uses Custom Resource Definitions (extending the API) to create new manageable resources and
- uses the control loop pattern (a non-terminating loop that regulates the state of a system) to reconcile the resources to their desired state.

SKA TANGO Operator

The SKA TANGO Operator is a k8s Operator capable of managing TANGO resources (DeviceServer and DatabaseDS), controlling their lifecycle within the k8s native control/event loop. Specifically:

- allows for a lighter deployment, due to avoiding the creation of InitContainers to resolve dependencies;
- optimises the startup time for Device Servers, as the operator can directly tap into the TANGO environment and retrieve information on dependent devices and the TANGO host itself and
- relieves the Kubernetes API of the management of extra resources (and their disposal).

Together with solving the above challenges, the main rationale to introduce the operator is to help developers to think directly in terms of TANGO-controls and not in terms of k8s resources, as those are components relevant to the platform and not the application. Introducing the operator also allows automating many of the tasks a human would do to operate a TANGO resource, in a quicker and more reliable manner. It also facilitates the collection of custom metrics on the CRDs, giving information on the system in a TANGO domain instead of the k8s domain (i.e., Device Servers instead of StatefulSets) therefore making TANGO components first-class citizens of k8s.

The resources added with the SKA TANGO Operator are the `databases.tango.tango-controls.org` and `deviceservers.tango.tango-controls.org`. The workflow for these is represented in Fig. 4 and Fig. 5.

One of the main advantages – together with the aforementioned aspects – of using the SKA TANGO Operator is the ability to extract very detailed information in the form of *Prometheus* [10] metrics and display them in *Grafana* [11]. Figure 6 shows a deployed application in SKA and general statistics about the start-up time of a Device Server. Figure 7 shows a more detailed view of the timing information, showing an example of how much time a specific server waits for the configuration to happen or for a particular dependency to be available. Figure 8 shows another refinement for the timing information, showing what the operator is doing for a selected device server.

CONCLUSION

In this paper, we discussed an approach to make k8s directly manage the TANGO-controls framework Device

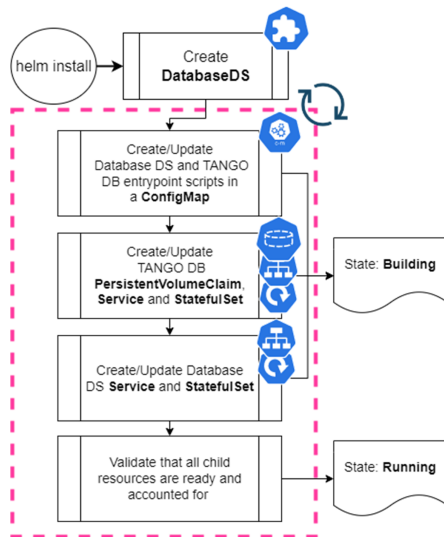


Figure 4: DatabaseDS workflow.

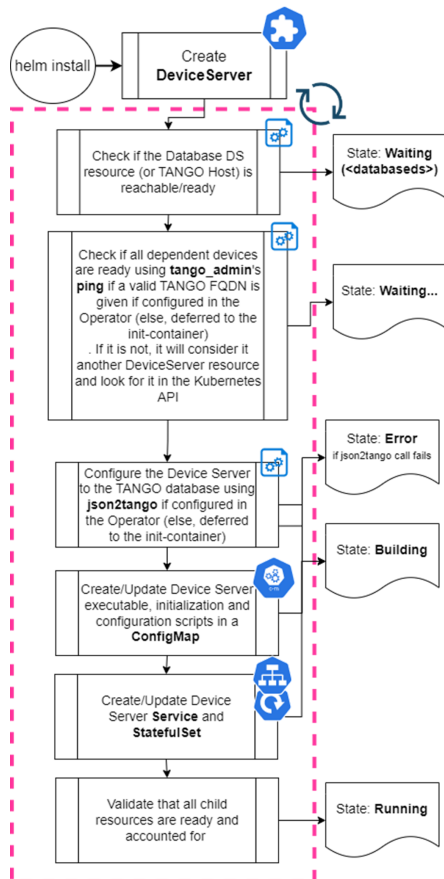


Figure 5: Device Server workflow.

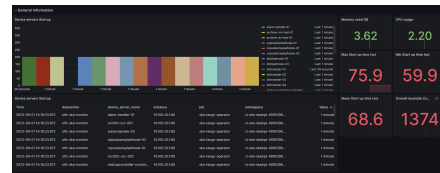


Figure 6: Device Server start up time.



Figure 7: Device Server detail timing information.

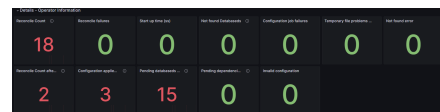


Figure 8: Device Server detail operator statistics.

detailed information in the form of Prometheus metrics so that it is possible to present those on a Grafana dashboard. Furthermore, it is important to consider an aspect that too often is underestimated in the context of monitoring and control system, such as security. This is an aspect that, for now, belongs to future work, but it is worth mentioning that with technologies such as *Linkerd* [12], it is possible to secure the communication of the Device Servers within and between k8s clusters.

Linkerd

Linkerd is a security focused service mesh that provides a lightweight proxy that is attached as a SideCar to Pods. This captures in and outbound traffic and adds encryption/decryption, policy controls and traffic monitoring and logging. Linkerd can be used both inside a Cluster and between Clusters, and can provide an end to end security solution for the TANGO-controls hierarchy. This can be achieved without modifications to the underlying application. Linkerd, can also be used for non-TANGO traffic (as long as it uses the TCP protocol), whenever retrofitting mTLS presents a sizeable effort.

Zero Trust Networking and Securing Control Systems

Zero Trust Architecture (or a Zero Trust Security Model [13]) is the principle of not implicitly trusting interactions between users, devices and services within a networked topology, and instead implementing steps to verify all connections. This approach acknowledges the drift of the modern networked environment towards a state where the traditional controlled network perimeters have become porous due to the highly connected nature of the working environment

Servers. This method allows for a number of optimizations, such as fast initialization time and startup reliability. The configuration of any device server is delegated to a specialised service so that less resources are used (in terms of memory, CPU, storage and even network utilization). It basically realises an abstraction of TANGO domain-specific components from the platform ones. It also provides very

and the variety of actors that require varying degrees of access to systems throughout any organisation. For example: a developer could work from a public Wi-Fi in an airport lounge, publish code and deploy critical services within an Observatory over VPN. A Telescope Operator could visit colleagues in another institution, discuss and amend observation details carried out over an Eduroam connection back to the Observatory. With these changes to what constitutes the secure boundary of modern systems, and in order to improve the overall security of system operations, it is necessary to rethink how we can secure highly connected systems when the chances of the traditional outer network perimeter being compromised is now greatly increased. This change in the status quo is demonstrated by recent attacks that have been carried out on research projects (recently ALMA [14], and NOIRLab [15]). One way of addressing these threats is to apply the principles of ZTA to key parts of the infrastructure, such as the Controls Systems, Monitoring and Logging. By taking this approach, it will reduce the inner attack surface area, and even when a system is compromised, the ability to still retain trust in the system observability components is more likely to remain - something that is not possible in a compromised perimeter based security model. Linkerd is one such tool that can be used to provide a ZTA oriented inner layer of protection throughout a system landscape, that can encompass modern containerised applications as well as legacy application suites.

ACKNOWLEDGEMENTS

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

REFERENCES

- [1] Tango-controls framework, <https://www.tango-controls.org/>
- [2] CORBA, <https://www.corba.org/>
- [3] ZeroMQ, <https://zeromq.org/>
- [4] Kubernetes, <https://kubernetes.io/>
- [5] Helm, <https://helm.sh/>
- [6] Gitlab, <https://about.gitlab.com/>
- [7] 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
- [8] Ska-tango-images repository, <https://gitlab.com/ska-telescope/ska-tango-images>
- [9] Kubernetes Operator Pattern, <https://kubernetes.io/docs/concepts/extend-kubernetes/operator/>
- [10] Prometheus, <https://prometheus.io>
- [11] Grafana, <https://grafana.com/>
- [12] Linkerd, <https://linkerd.io/>
- [13] Linkerd, https://en.wikipedia.org/wiki/Zero_trust_security_model
- [14] Linkerd, <https://www.eso.org/public/announcements/ann22014/>
- [15] Linkerd, <https://noirlab.edu/public/announcements/ann23022/>