SKA TANGO Operator

M. Di Carlo, P. Harding et al.
INAF – Osservatorio Astronomico d'Abruzzo

ICALEPCS CAPE TOWN SOUTH AFRICA 2023

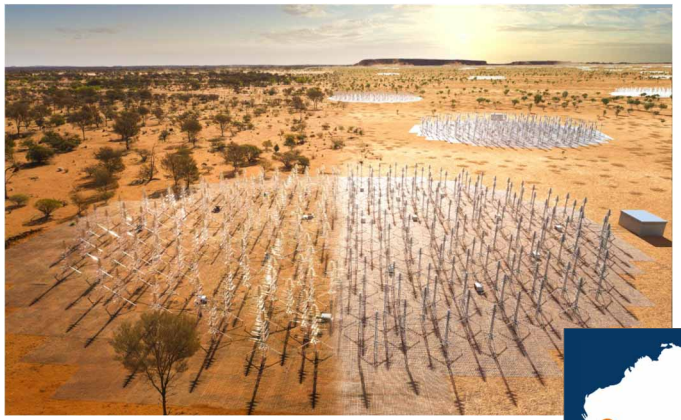
SKA Telescopes

Mid Telescope

Location: South Africa

350 Mhz to 15.3 GHz

197 dishes - max baseline: 150km(1)



Low Telescope

Location: Australia

50 MHz to 350 GHz

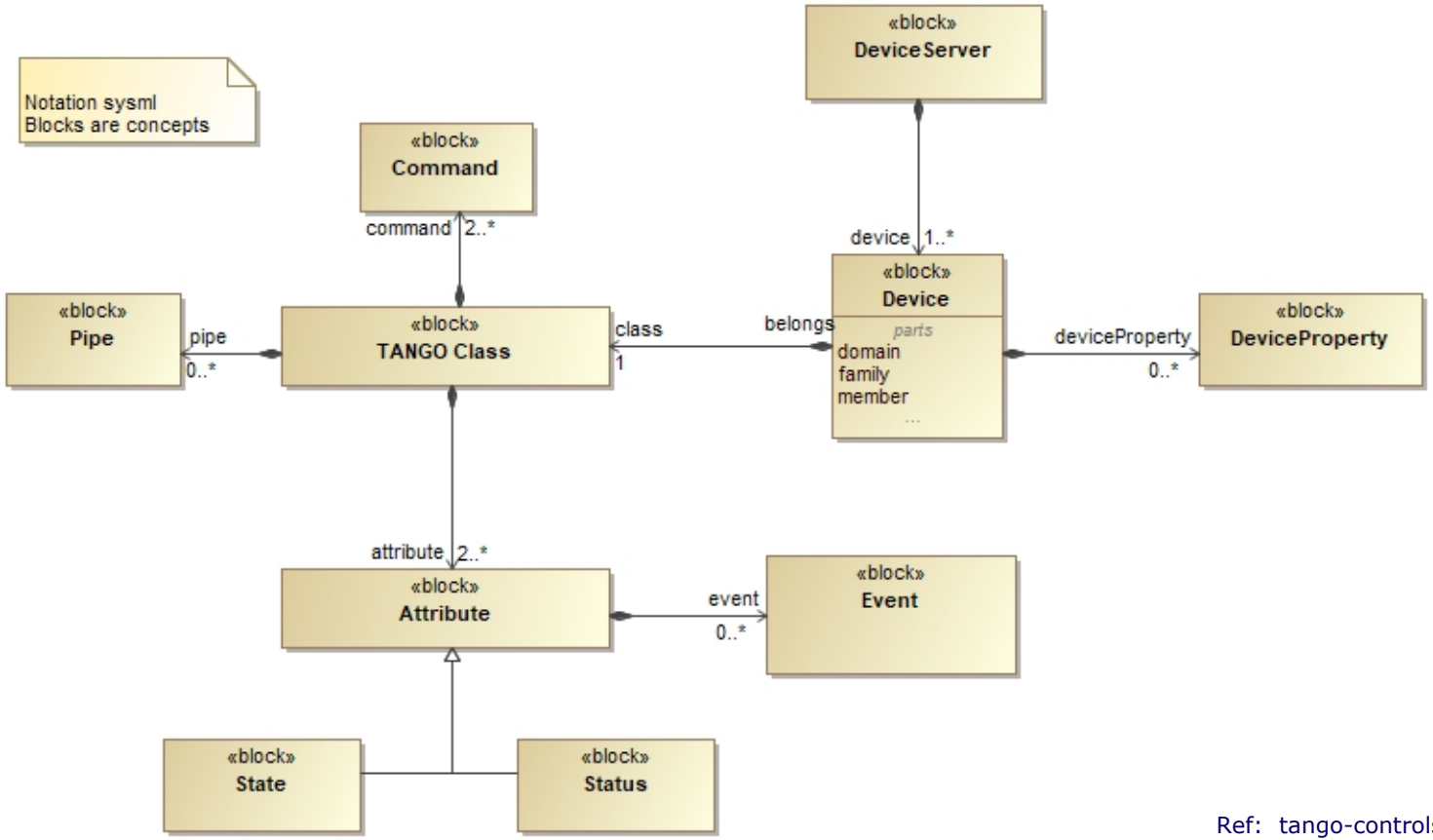
(1) Data for SKA1 implementation

Ref: skao.int

SKA Development and Deployment Practices

- Kubernetes (**k8s**) for container orchestration (kubernetes.io)
- **Helm** for packaging and deploying SKA Software *(helm.sh)*
  - Tool for managing charts, where a **chart** is a recipe to deploy several **k8s** resources (i.e., containers, storage, networking components, etc) required for an application to run
  - Works on templates, allows to adapt generic configurations to different environments (i.e., the different SKA datacentres)

TANGO-controls framework



Ref: tango-controls.readthedocs.io

Kubernetes

- Open-source system for **automating** the deployment, **scaling**, and **management** of containerized applications
  - Very extensible with multiple plugin points
  - Collection of primitives: an abstraction of compute, network, and storage
  - Supports common scheduling models: stateful, replicas (stateless), daemons and jobs
- Declarative “model” of operation, with various controller components managing the lifecycle of the declared resources

Imperative vs Declarative

- "Imperative" is a command
  - Add this device to the DB
  - Execute this python command
  - …
- "Declarative" is a statement of the desired state
  - Deploy these devices, with this configurations, in this specific order, with those dependencies…

Setting up a TANGO Device server (in general)

- For every **device server**:
  - Deploy the code (i.e. to a VM)
    - Configuration (i.e., environment variables)
    - Resources (i.e., persistent storage, networking)
  - Include all of the devices in the TANGO **database**:
    - Properties
    - Polling definition
    - …
  - Start devices in **order** (if there are dependencies, wait for each of them)

Device server in k8s

- A device server will be composed by the following k8s resources:
  - **Job** for the initialization of the TANGO database
  - **Service** (exposes the application to the network)
  - **Statefulset** (in k8s this is an object used to manage stateful applications)
    - Pod (in k8s groups one or more containers into a unit)
    - InitContainers (specialized containers that run before app containers in a Pod) to resolve the DS dependencies
      - Waiting for the configuration job to complete
      - Waiting on devices to be up in TANGO

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

```yaml
name: "name-used-in-k8s"
function: description-text
domain: description-text
legacy_compatibility: false
instances: ["01"]
depends_on:
  - device: sys/database/2
  - device: sys/motor/1
entrypoints:
- path: "<optional-path-to-python-file.py>"
  name: "module.ClassName"
command: "<optional: the python command to use>"
server:
  name: "server-name"
  instances:
  - name: "01"
    classes:
    - name: "ClassName"
      devices:
      - name: "test/mydevice/3"
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
```

Declaring a DS in k8s

Name is the k8s name of the resources

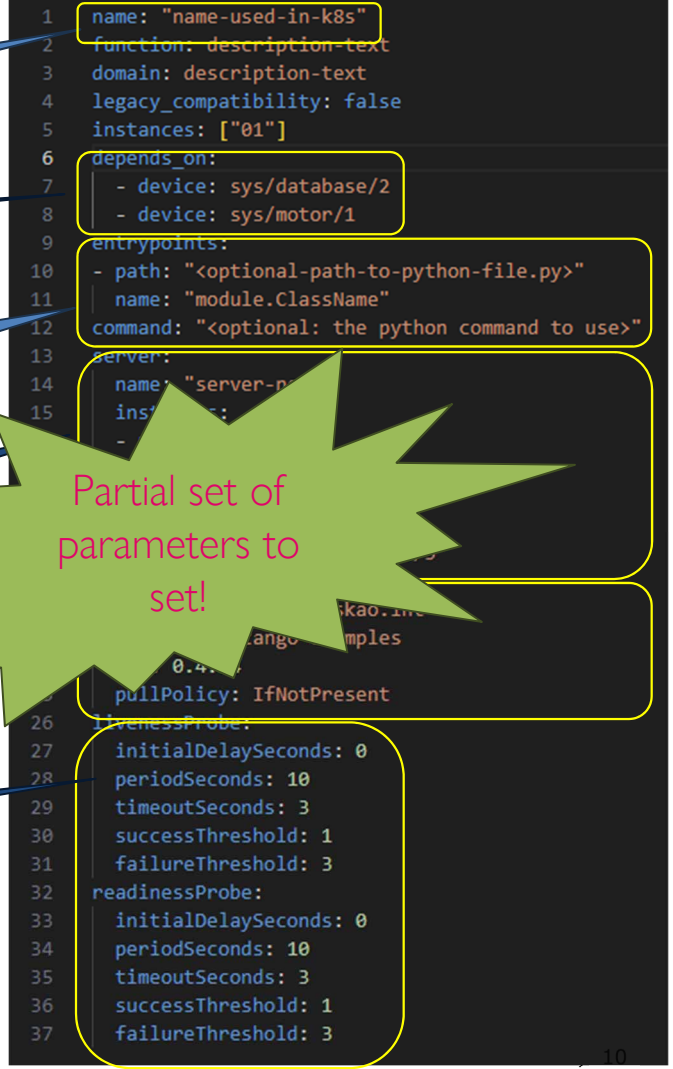
The list of dependencies: devices or simple host:port

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



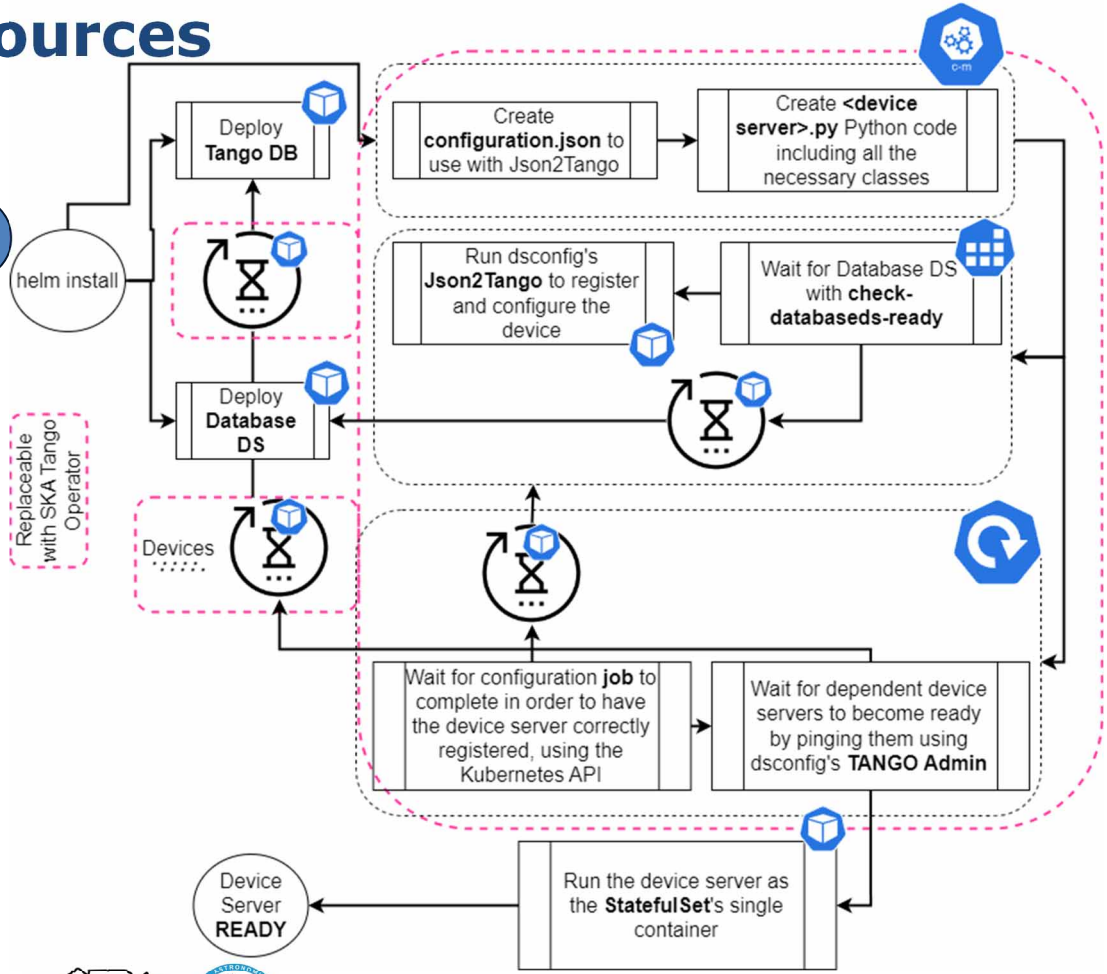
Partial set of parameters to set!

From YAML to k8s resources

2 **Helm** charts:

- ska-tango-base: which installs the databaseds and tangodb
- ska-tango-util library chart: Translates the yaml definition into k8s resources

https://gitlab.com/ska-telescope/ska-tango-images/-/tree/master/charts



Challenges with ska-tango-util helm library chart

- Creates **extra** resources in the Kubernetes cluster (i.e., extra init-containers to solve dependencies)
- Leaves behind **spent** resources (i.e., job pods that have completed)
- The Crash Loop Backoff behaviour that exists in the Kubernetes cluster can slow the Device Server startup in case of multiple dependencies
- An environment might end up with **different versions** of the k8s resource declarations, at the choosing of the developers (i.e. developers can select the version they like)

Extending k8s - The Operator pattern

- *"The operator pattern 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"*
  - Extends the **Control Plane** to give custom **behaviours**
  - Uses **Custom Resource Definitions** (extending the API) to create new manageable resources
  - Uses the **control loop** pattern (a non-terminating loop that regulates the state of a system) to reconcile the resources to their desired state

Ref: kubernetes.io/docs/concepts/extend-kubernetes/operator

The SKA TANGO Operator

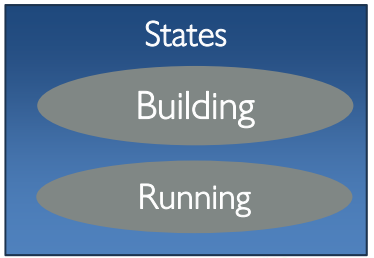
- Kubernetes Operator capable of managing TANGO resources (DeviceServer and DatabaseDS), controlling their lifecycle within the Kubernetes' native control/event loop
  - Allows for a **lighter** deployment, due to avoiding init-containers for dependency resolution
  - Optimised startup time for Device Servers, as the operator can directly tap into the TANGO environment and retrieve information on dependent devices and the TANGO **host** itself

Why introduce an Operator ?

- Developers know Device Servers, not StatefulSets and Pods (or any k8s resource for that matter), as those are components relevant to the **platform** and not the **application**
- Making TANGO components **first-class** citizens of Kubernetes
- Automating much of the tasks a human would do to operate a TANGO resource, quicker and more reliably
- Allows to collect custom metrics on the CRDs, giving information on the system in a TANGO domain instead of a k8s domain (i.e., Device servers instead of StatefulSets)

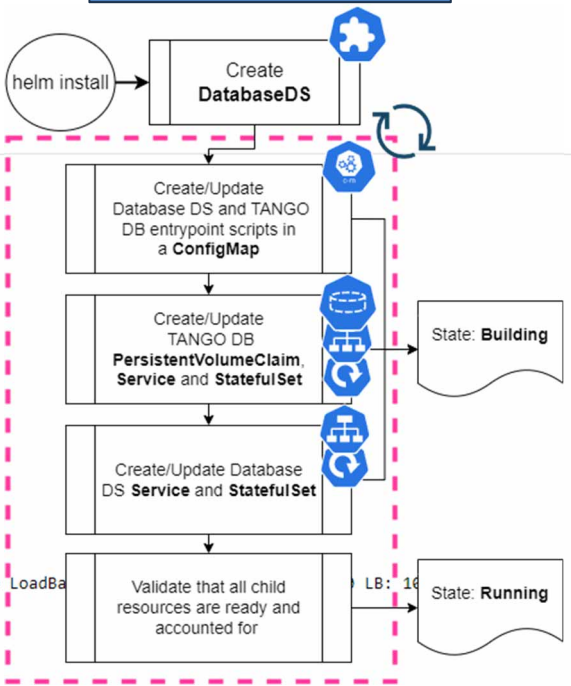
Custom Resource Definition (CRD) *databaseds.tango.tango-controls.org*

- TANGO DB StatefulSet, Service and PersistentVolume
- Database DS StatefulSet and Service



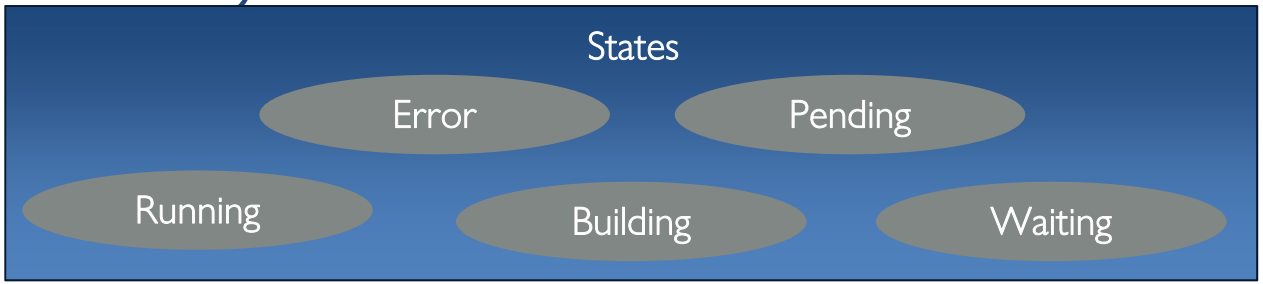

```
> kubectl -n ci-ska-skampi-3649827723-mid describe databaseds databaseds-tango-base
------------------------------------------------------------------------------------------------

API Version:  tango.tango-controls.org/v1
Kind:         DatabaseDS
Metadata:
  Creation Timestamp:  2023-01-24T09:26:16Z
Spec:
  Cluster Domain:         cluster.local
  Enable Load Balancer:   true
  Image Pull Policy:      IfNotPresent
  Orbport:                10000
  Tango DB Storage Class:  nfss1
Status:
  Ds:         1 of 1
  Replicas:   2
  Resources:  databaseds-tangodb-databaseds-tango-base := ClusterIP/10.98.139.14, mysql/3306 - databaseds-tango-base := LoadBa                LB: 10
  State:      Running
  Succeeded:  2
  Tangodb:    1 of 1
```

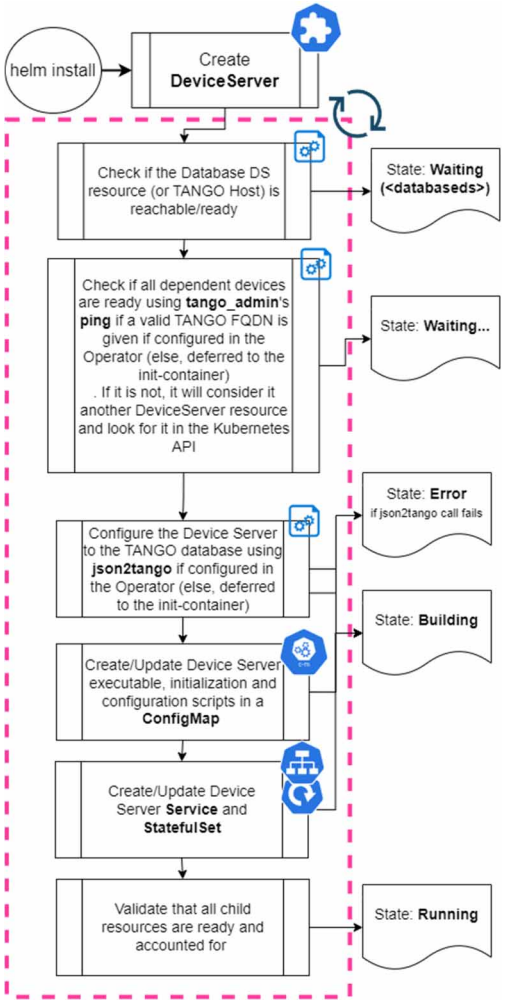


Custom Resource Definition (CRD): *deviceservers.tango.tango-controls.org*

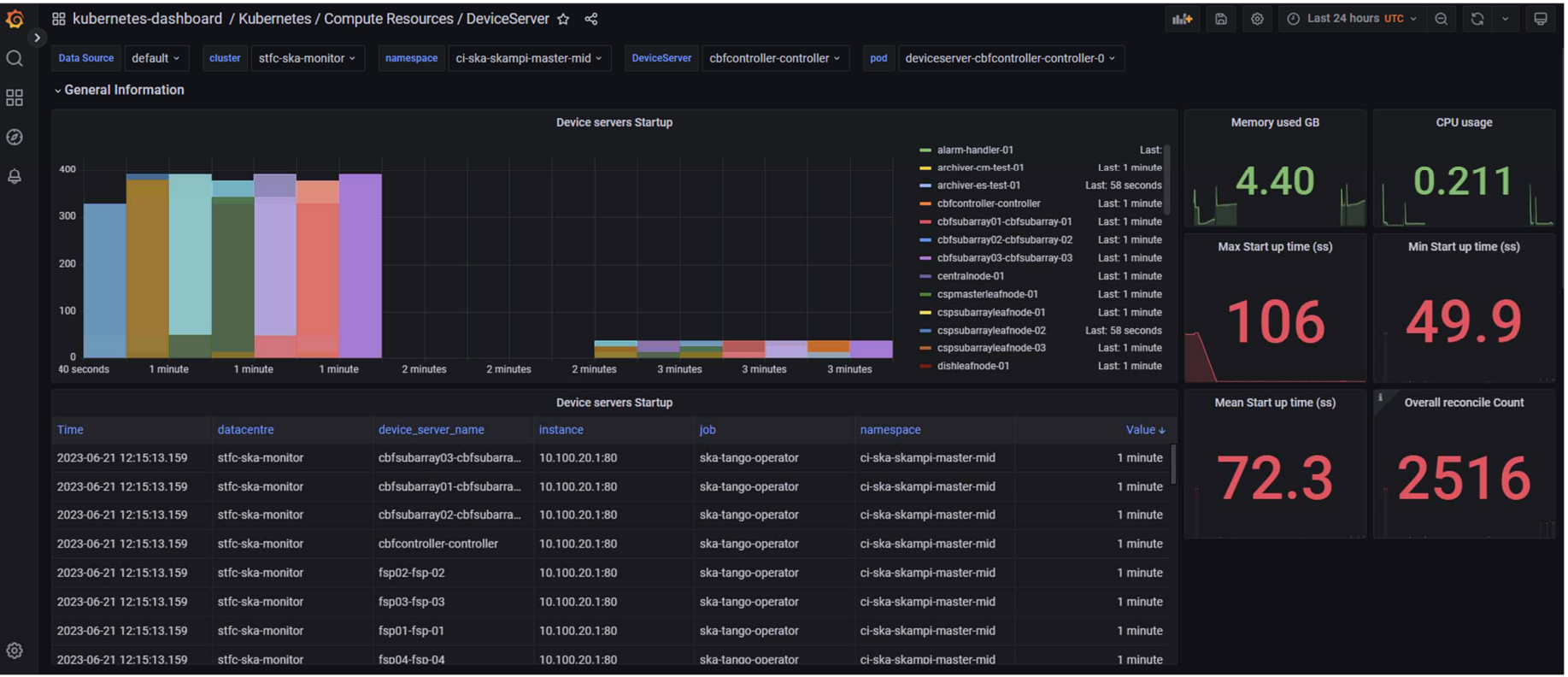
- Device Server StatefulSet and Service and Configurations (DS script used to run the device)



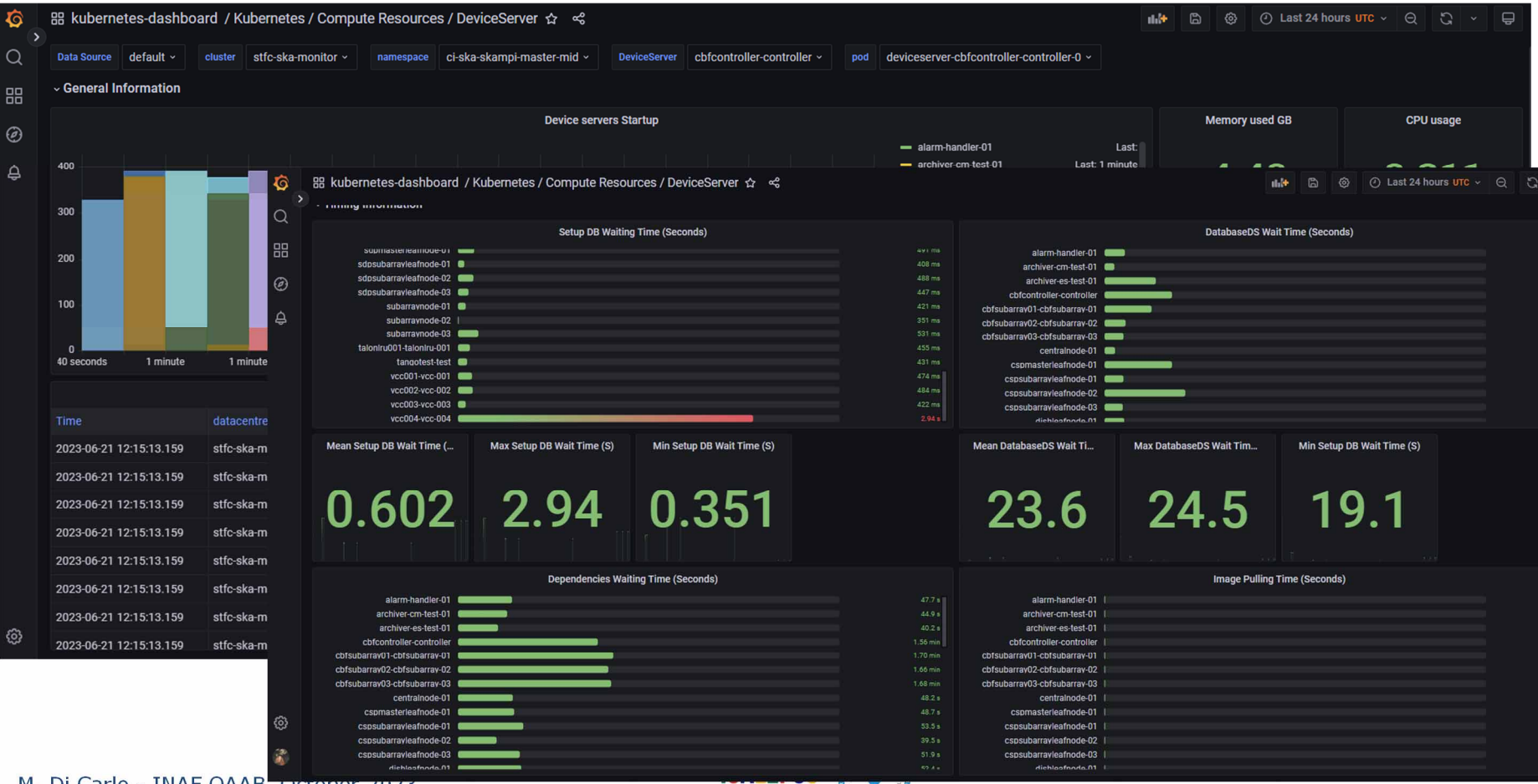

```
> kubectl get deviceserver --all-namespaces
-------------------------------------------------------------------------------------------------------------
NAMESPACE                       NAME                                COMPONENTS   SUCCEEDED   AGE     STATE
-------------------------------------------------------------------------------------------------------------
ci-ska-skampi-3650792176-mid    archiver-cm-archiver-test-01        2            2           2m30s   Running
ci-ska-skampi-3650792176-mid    archiver-es-archiver-test-01        2            2           2m30s   Running
ci-ska-skampi-3650792176-mid    cbfcontroller-controller            2            2           2m30s   Running
ci-ska-skampi-3650792176-mid    cbfsubarray01-cbfsubarray-01        2            2           2m30s   Running
ci-ska-skampi-3650792176-mid    cbfsubarray02-cbfsubarray-02        2            2           2m30s   Running
...
```



Grafana support for the Operator metrics



Grafana support for the Operator metrics

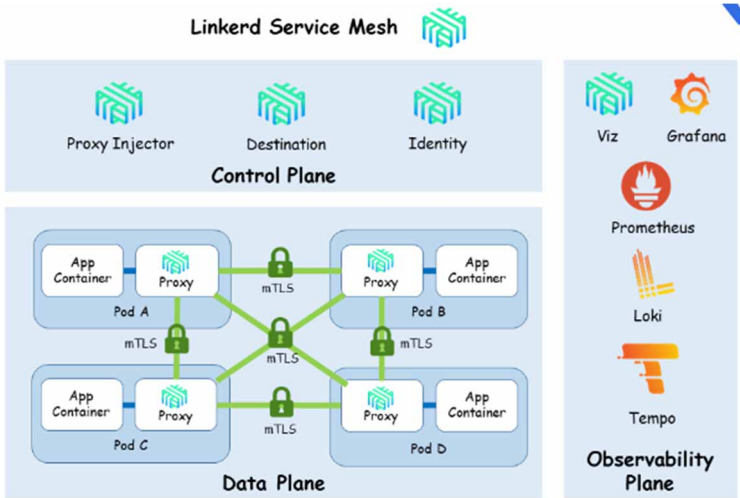


Summary

- Kubernetes managed device servers!
- Optimized initialization times and startup reliability
- Configuration delegated to a specialized service
- Less resources usage (memory, cpu, storage and even network)
- Better metrics (and dashboard) to evaluate our device servers
- Abstraction of TANGO domain-specific components from the platform ones
- And… Security (almost) for free!

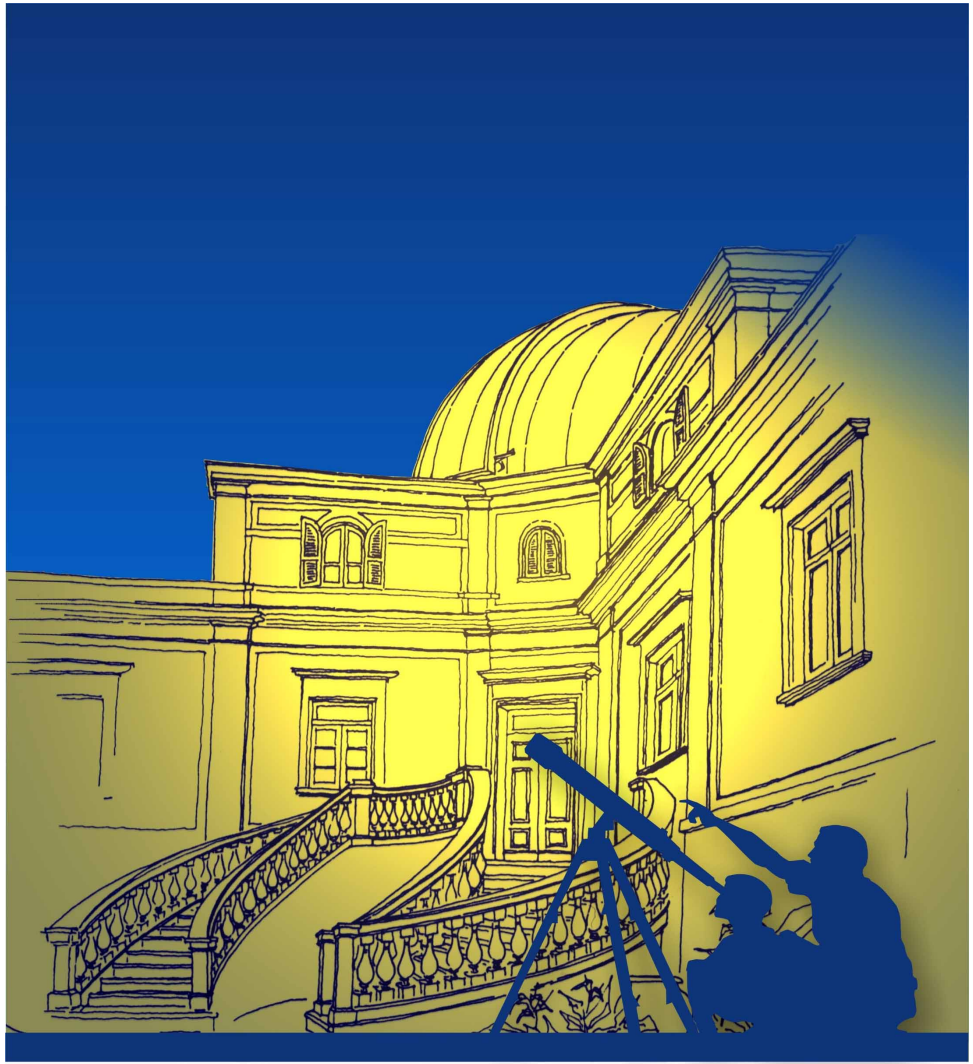
Securing Tango - With Linkerd

- Linkerd is a service mesh:
  - It provides a lightweight proxy that is attached as a SideCar to Pods
  - It captures in and outbound traffic
  - The traffic is encrypted/decrypted
  - Policy controlled
  - Traffic is monitored and logged



Securing Tango - Linkerd, a Universal Mesh

- Linkerd can be used in Cluster and between Clusters
- This approach enables the end to end security of the Tango Controls hierarchy - all without modification to the underlying application
- It can also be used for non-Tango traffic (any TCP), wherever it is challenging to retrofit mTLS



Thanks for you attention!

www.oa-abruzzo.inaf.it

matteo.dicarlo@inaf.it