


# KUBERNETES: GENERAL CONCEPT AND ITS USE IN THE CONTEXT OF SKA

Matteo Di Carlo INAF - OAAB

# About me

- Matteo Di Carlo (matteo.dicarlo@inaf.it)
- Working for INAF-OAAB since 2014
- Since 2015 in the SKA project
- Software engineer, in SKA part of the system team and coordinator of the cop-tango community
- Certified Kubernetes Administrator The logo is a blue ship's wheel with the text "CERTIFIED kubernetes ADMINISTRATOR" in white and blue.
- <https://orcid.org/0000-0002-3903-9637>

# Kubernetes Concepts

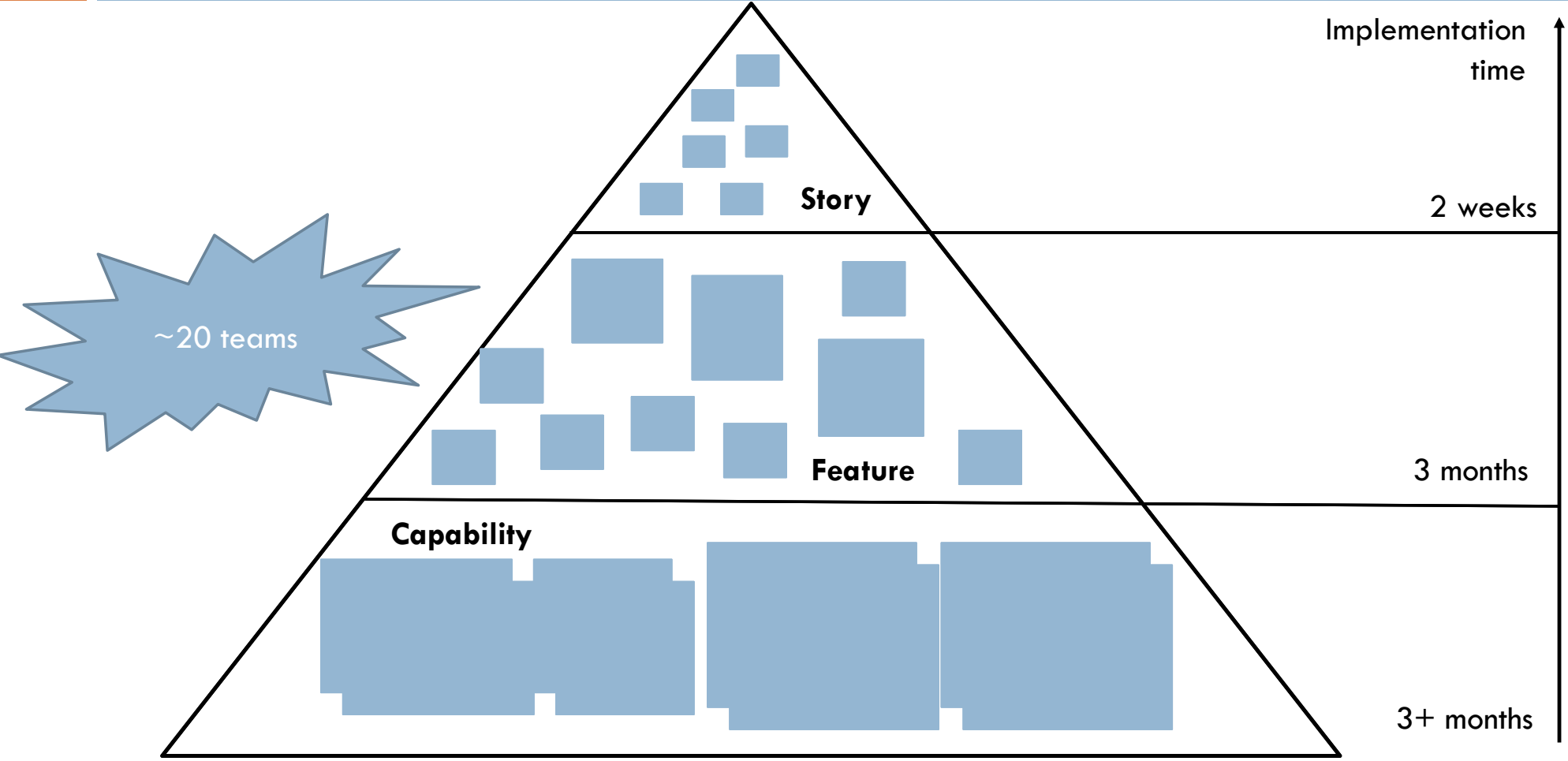
Open-source system for automating deployment, scaling, and management of containerized applications.

- ❑ Pod
- ❑ Deployment
- ❑ StatefulSet
- ❑ Service
- ❑ Ingress
- ❑ ConfigMaps
- ❑ Persistent volumes
- ❑ ...

# SKA Project

- International effort to build two radio interferometers in South Africa and Australia
- One Observatory monitored and controlled from the global headquarters (GHQ) based in the United Kingdom at Jodrell Bank
- Software development process is Agile
  - ▣ Mainly incremental and iterative
  - ▣ Many teams (17) including a specialized team (known as system team) devoted to support the continuous Integration, test automation and continuous Deployment.

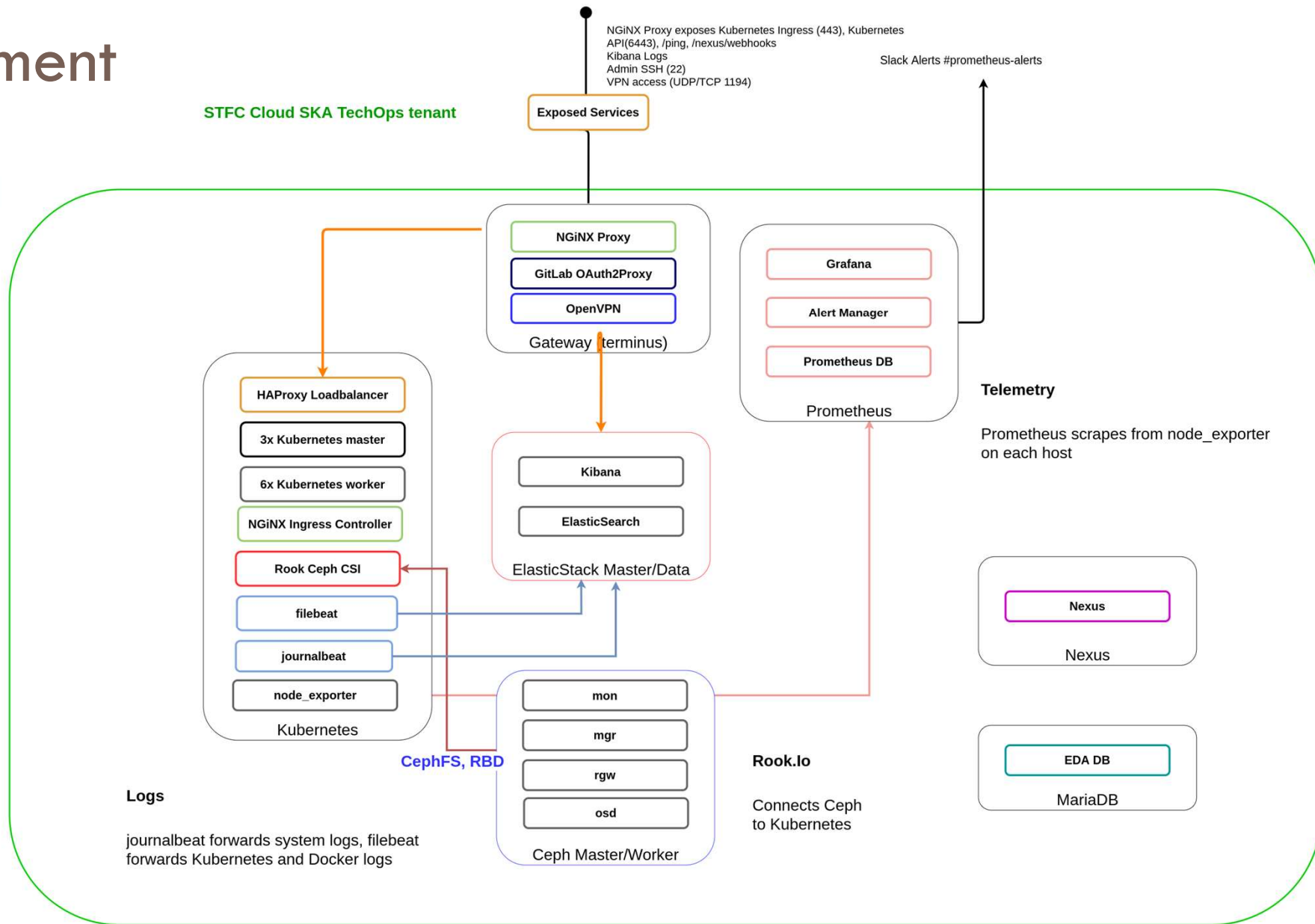
# Safe Agile development



# SKA infrastructure

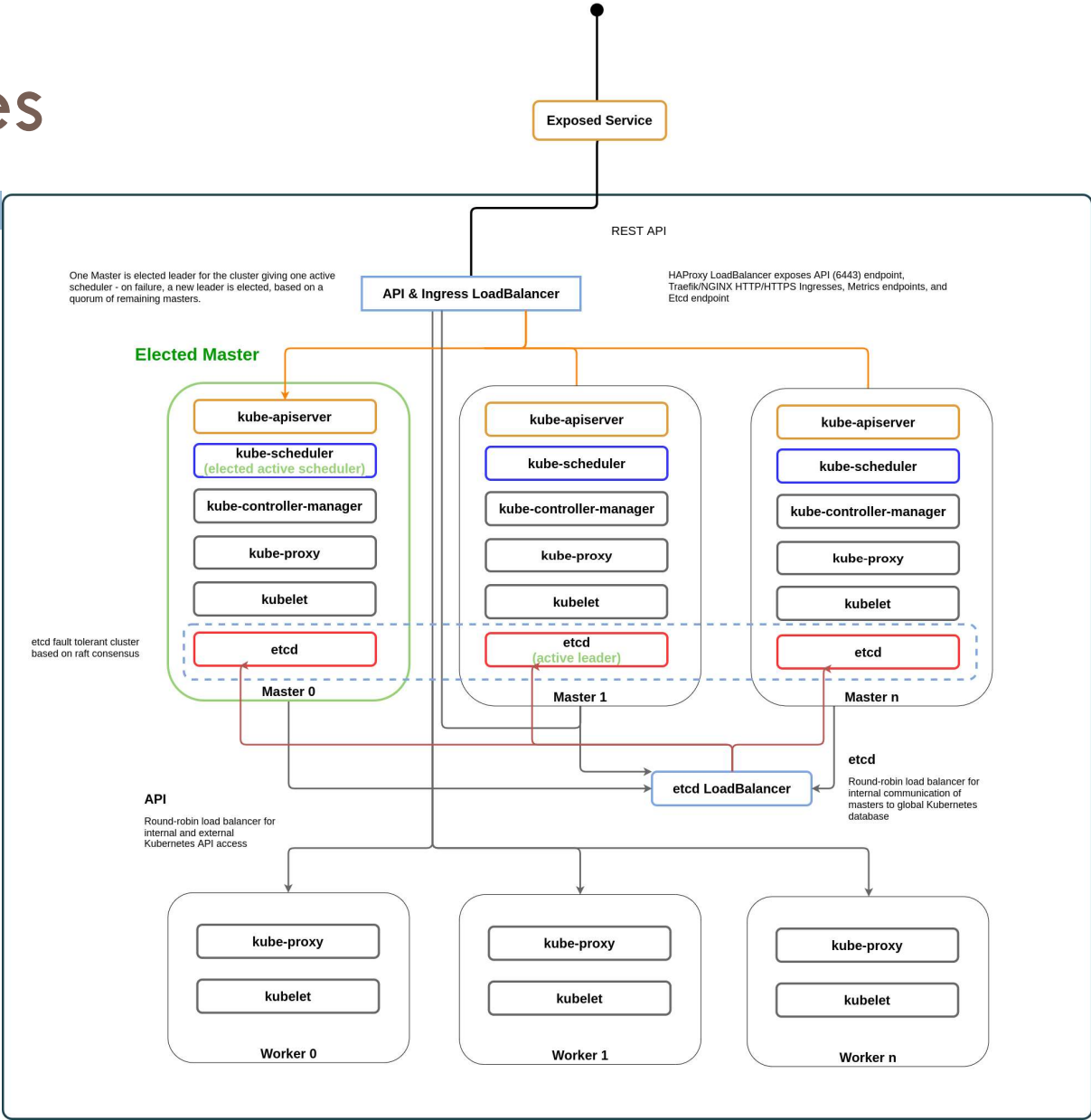
- Two openstack virtualization available, one in Portugal (EngageSKA) another one in UK (STFC)
- Composed by a number of «services»:
  - kubernetes
  - Prometheus and Grafana
  - Rock/ceph
  - Elasticsearch
  - Gateway and VPN
  - Nexus
  - Archiver DB (MariaDB/TimescaleDB)

# Deployment view



Credit: P.Harding (SKAO)

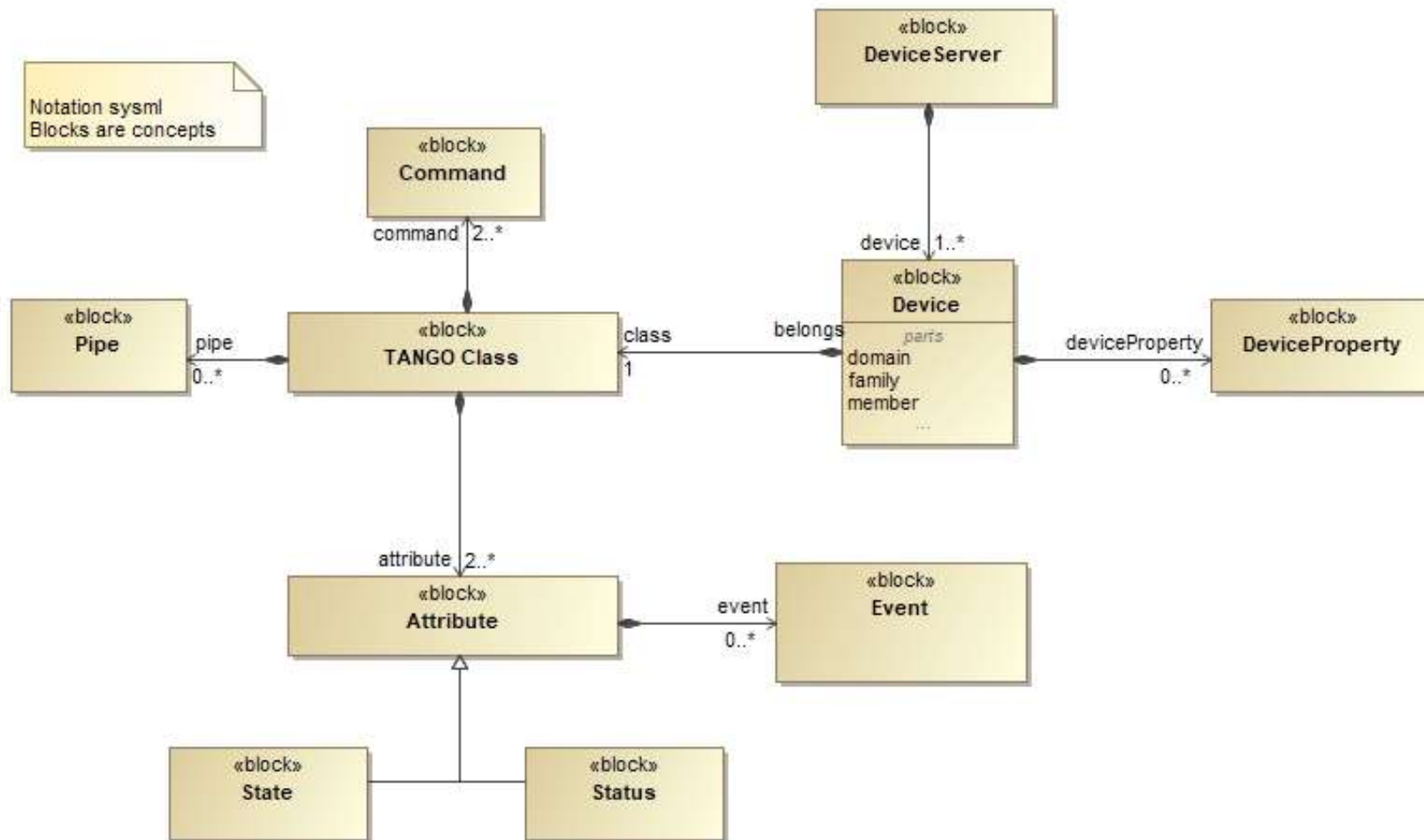
# Kubernetes



Credit: P.Harding (SKAO)



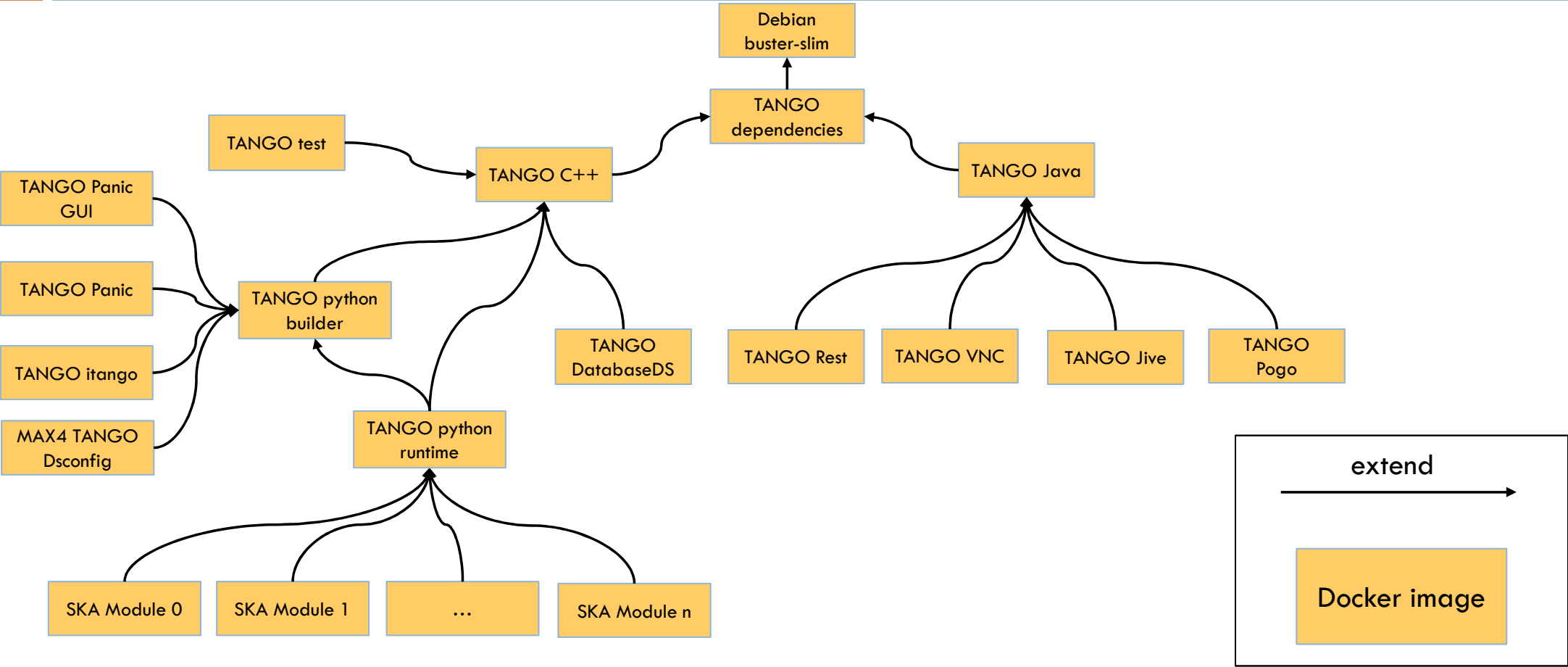
# TANGO-controls



# Containerization

- SKA == set of elements == a set software modules
- For each module there is one repository
- For each repository there is one docker image
  - ▣ convenient way to package up applications and preconfigured server environments

# ska-tango-images: containerized environment for TANGO-controls application



# Kubernetes and Helm

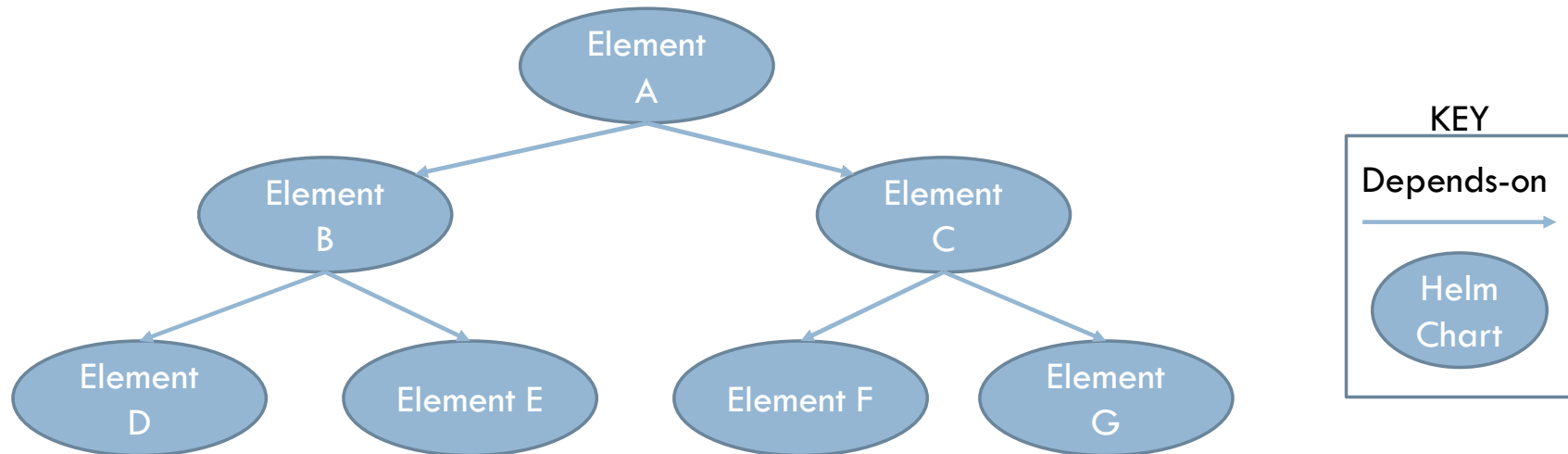
- Kubernetes (k8s) for container orchestration ([kubernetes.io](https://kubernetes.io))
  - ▣ Service == TANGO Device Server
- *Helm for packaging SKA k8s applications* ([helm.sh](https://helm.sh))
  - ▣ Tool for managing Kubernetes charts
  - ▣ Chart is a package of pre-configured Kubernetes resources (set of information for running a Kubernetes application)

**For each SKA element there must be an helm chart for running it in k8s!**

***Use of Makefiles for lifecycle management (one command for build images, start application using helm, test application and clean)!***

# Integration with Helm

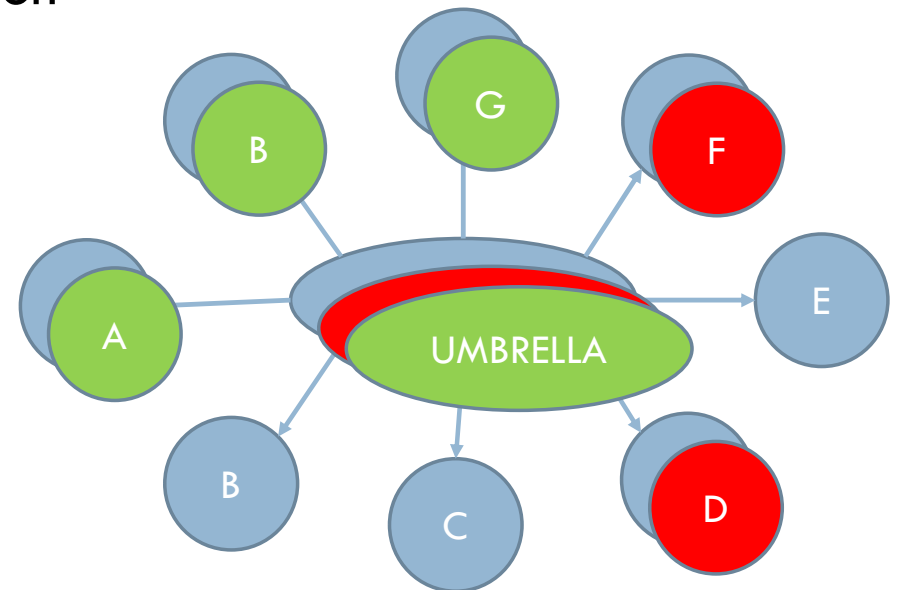
- Helm has the concept of dependency
  - ▣ An helm chart can have one or more sub-charts
- The integration of SKA elements can be done with this concept



# Helm sub-charts Architecture

- Operational aspects of using dependencies: the sub-charts are
  - ▣ aggregated into a single set; then
  - ▣ sorted by type followed by name; and then
  - ▣ created/updated in that order.

For every SKA element,  
there is at least an  
umbrella chart for  
integration testing



# ska-tango-base chart

- The ska-tango-base chart installs/defines the basic TANGO ecosystem in Kubernetes composed by the following services:
  - tangodb: mysql database used to store configuration data used at startup of a device server.
  - databaseds: device server providing configuration information to all other components of the system as well as a runtime catalog of the components/devices.
  - itango: an interactive Tango client.
  - vnc: debian environment with x11 server and vnc/novnc installed on it.
  - tangorest: rest api for the TANGO eco-system.
  - tangotest: TANGO test device server.

# ska-tango-util chart

- Library chart which helps other application chart defines TANGO device servers. In specific for each device server defined, it defines the following k8s resources:
  - a job for the initialization of the entry in the tangodb
  - a service
  - a statefulset
  - a role, rolebinding and a service account for waiting for the job to be finish in an init container



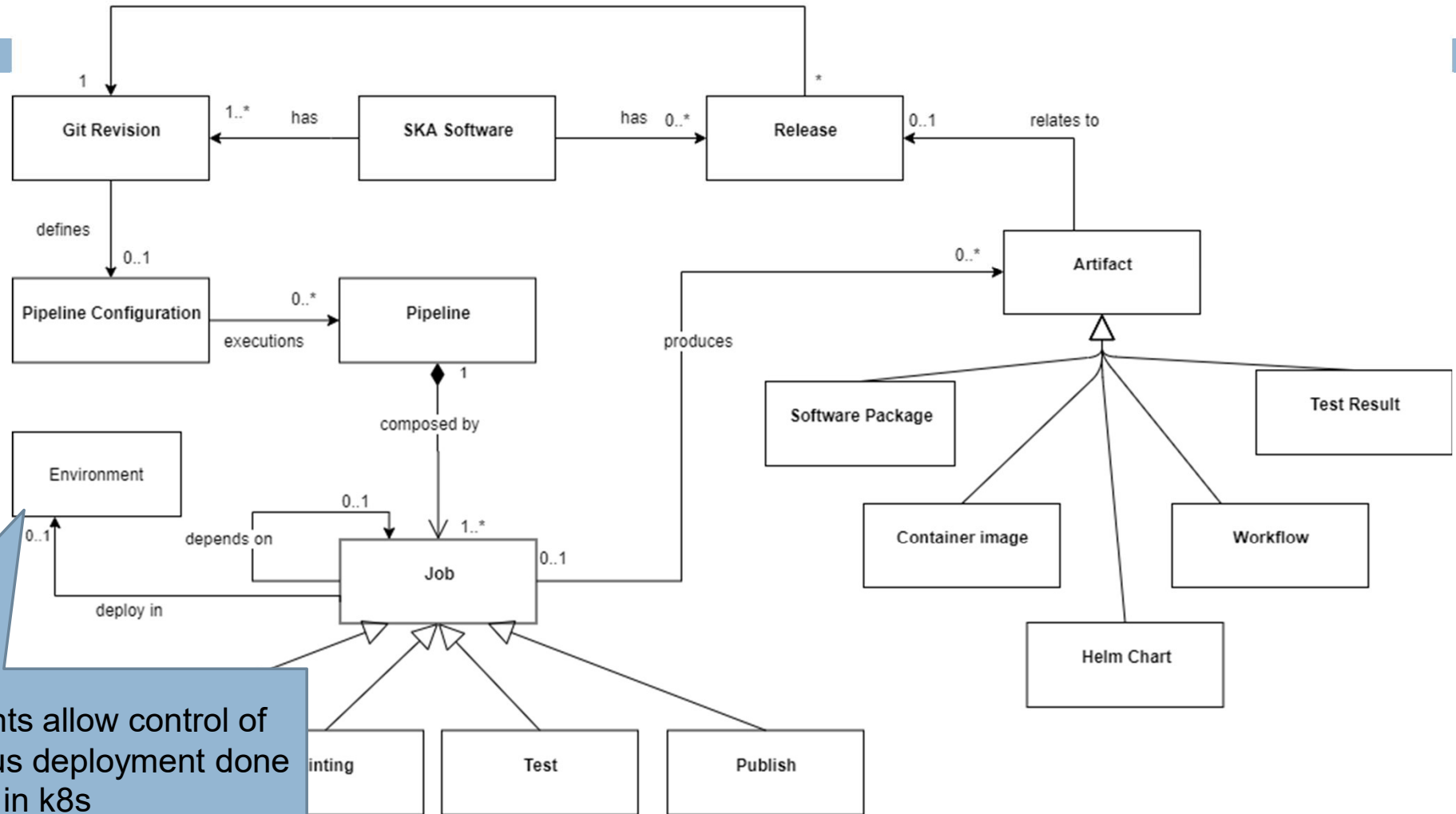
# CI-CD

- When many parts of the project are developed independently for a long period of time (weeks or longer),
- Code base and build environments diverges
- When changes are integrated
  - ▣ Weeks in verifying that everything works
  - ▣ Developers spend time in solving bugs introduced months earlier

# CI-CD

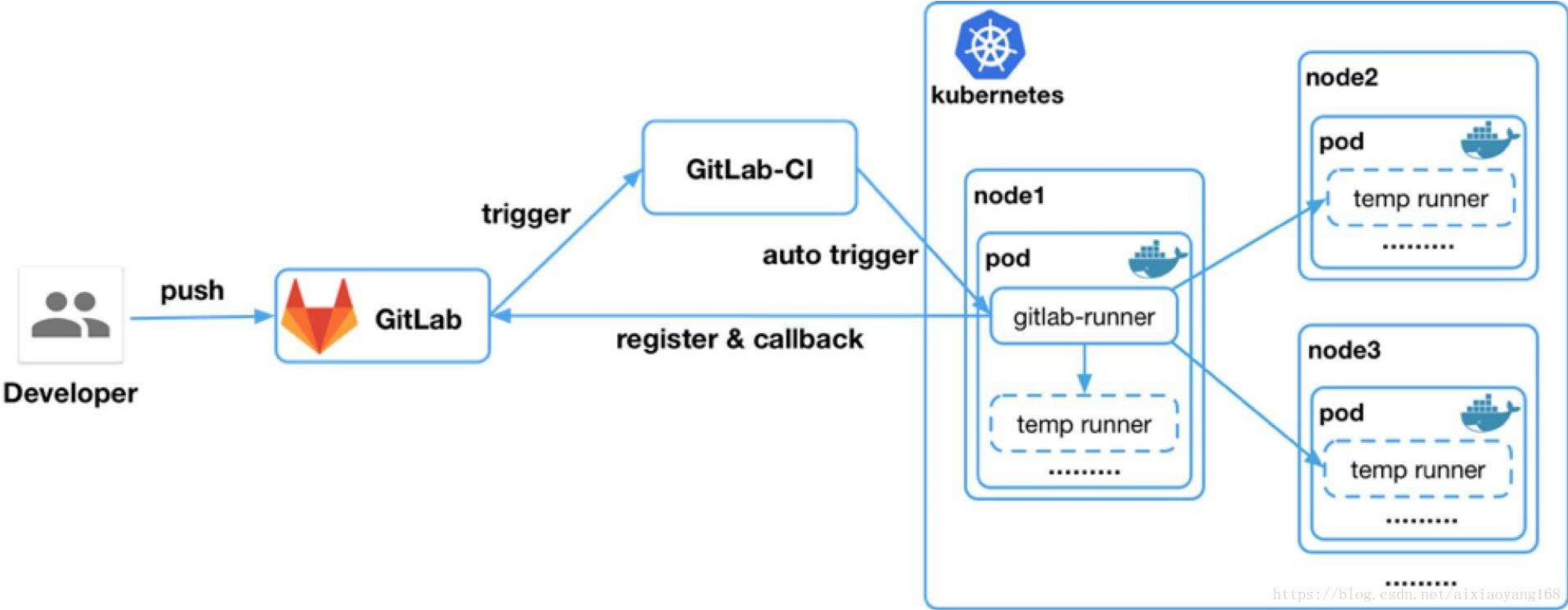
- Continuous integration (CI)
  - ▣ Set of development practices that requires developers to integrate code into a shared repository several times a day.
  - ▣ Each check-in is then verified by an automated build, allowing teams to detect problems early.
- Continuous delivery (CD)
  - ▣ Automate the delivery of new releases of software
  - ▣ Deployment has to be predictable and sustainable
    - The code must be in a deployable state
    - **Testing** needs to cover enough of your codebase.
- Continuous deployment (CD)
  - ▣ One step further: every single commit to the software that passes all the stages of the build and test pipeline is deployed into the production environment

# Gitlab Model



Environments allow control of the continuous deployment done in k8s

# Gitlab runner



Credit: P.Harding (SKAO)

# ska-tango-examples

- Demonstrates how to structure an project that provides some simple Tango devices coded in PyTango with CI/CD capabilities with Kubernetes and all SKA infrastructure
- Many authors:
  - ▣ Stewart Williams
  - ▣ Matteo Di Carlo
  - ▣ Matteo Canzari
  - ▣ Piers Harding
  - ▣ Anton Joubert
  - ▣ and many more <https://gitlab.com/ska-telescope/ska-tango-examples/-/graphs/master>

# unit-testing

- ❑ Encapsulated in the Makefile (make unit\_test)
- ❑ It uses pytest with no bdd
- ❑ It requires the TANGO-controls framework and pytango to work in a local laptop (tested on ubuntu 20.04 and windows wsl ubuntu 20.04)
- ❑ It is also possible to run them in a simple container (make pipeline\_unit\_test)
- ❑ It uses pytest fixture and a factory pattern for creating the right device context

# ska-tango-examples helm chart

- In order to install the examples, two charts have been created: one called ska-tango-examples which is the real application and the umbrella chart, called test-parent, used for testing.
- The ska-tango-examples uses the ska-tango-base chart for setting up the TANGO eco-system (only mysql database and databaseds device) and the ska-tango-util library chart which helps in the definition of the TANGO device servers

# Steps to installation

---

- ❑ Build the image with `make build`
- ❑ Install the chart with `make install-chart`
- ❑ Wait for the pods to be running with `make wait`
- ❑ Watch what's happening with `make watch`



# Development workflow with Makefile

- The usual workflow is:
  - start pogo (make start\_pogo) and create the skeleton in the right folder;
  - develop your device
  - unit-test very often with make unit\_test
  - once the device is ready, add its definition into a file in the data folder of the ska-tango-examples chart
  - Install it with make install-chart and check it's working with make wait
  - Test the new device with make test
  - In case check the device with jive with make install-chart JIVE=true
  - Uninstall when done

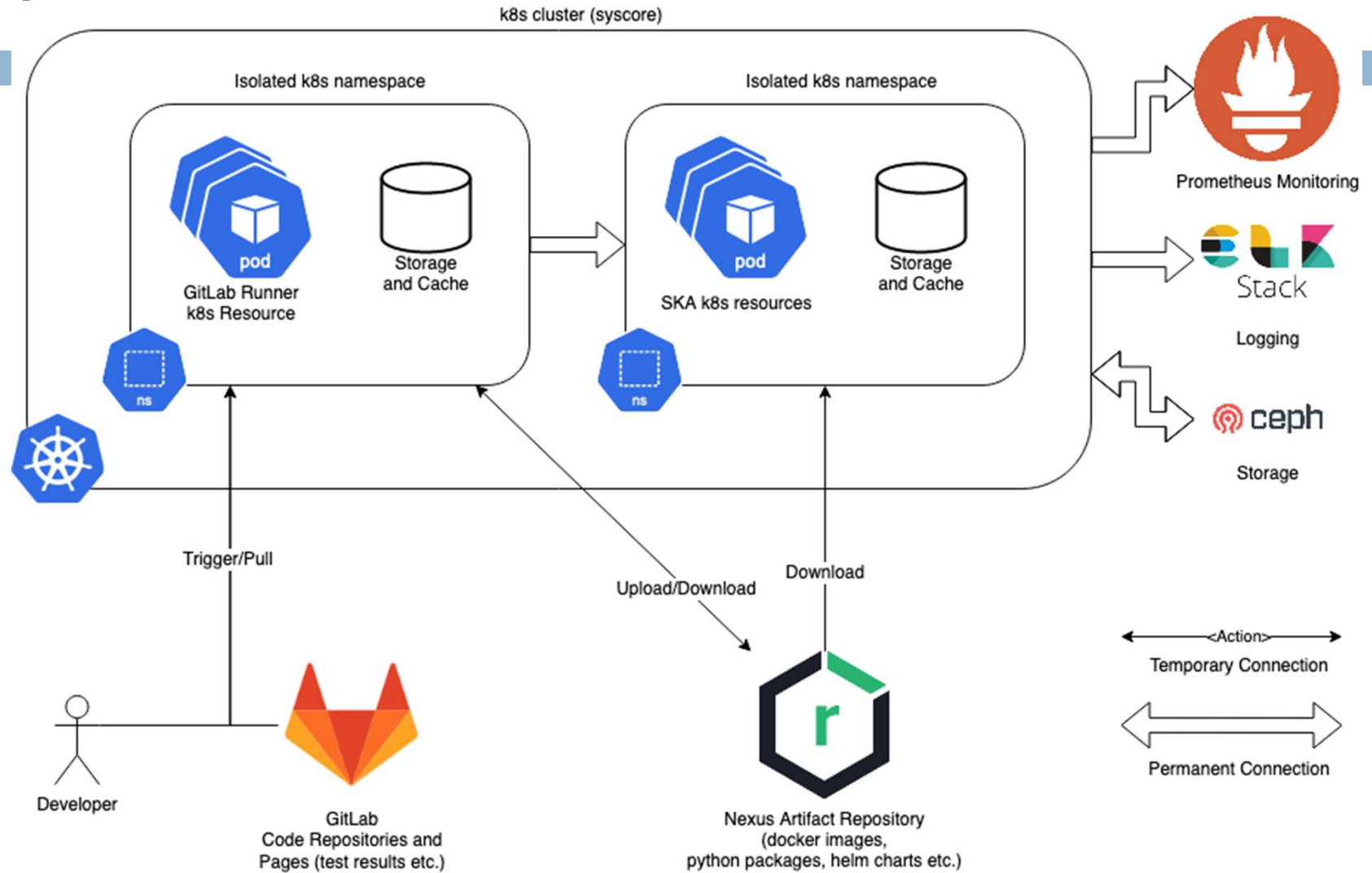
# ska-tango-examples gitlab pipeline

- It includes the following stages:
  - build (image and wheel)
  - lint (python source code and chart)
  - test (with and without the TANGO eco-system)
  - pages (for test information)
  - publish (for helm chart)
  - .post (ci-metrics, badges, etc.)

# Generic development workflow

- For each repo and for each commit (!):
  - install the (umbrella) chart in an isolated namespace
  - wait for every container to be running
  - For the tests:
    - Create a k8s pod (a container) in the isolated namespace
    - Run pytest inside the above pod
    - Return the tests results
  - uninstall the (umbrella) chart

# Gitlab pipeline - runtime



# Gitlab pages

- Store testing and coverage information and everything else important
- Pipeline artefacts are used to generate metrics and badges
- <https://developer.skao.int/en/latest/tools/ci-cd/continuous-integration.html#automated-collection-of-ci-health-metrics-as-part-of-the-ci-pipeline>

# Monitoring the performance of the devices

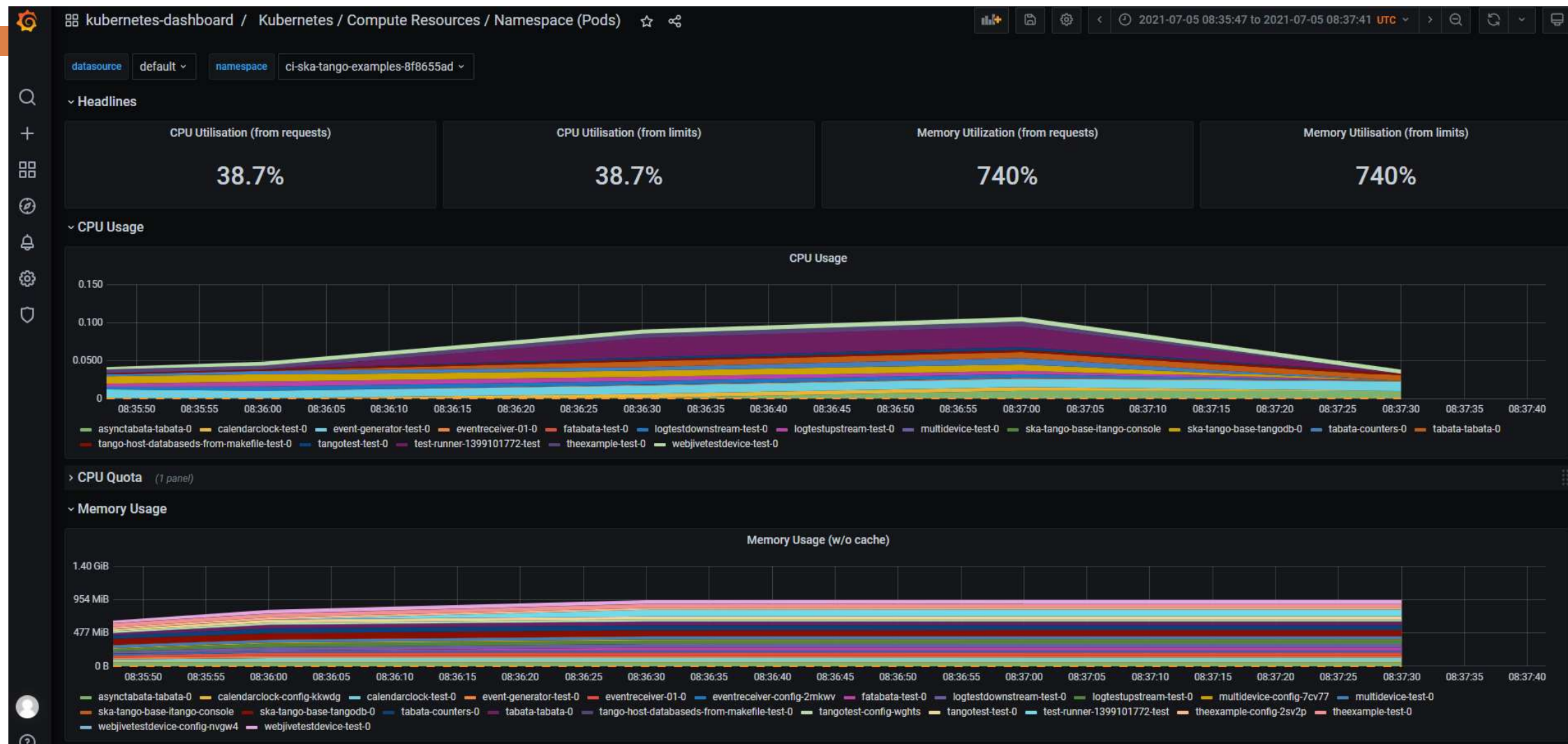
The screenshot displays the GitLab CI/CD interface for a project named 'ska-tango-examples'. The main content area shows a job titled 'test' in a 'running' state, triggered 16 seconds ago by Matteo Di Carlo. A message indicates that the job is creating a deployment to the 'test' environment using the 'stfc-k8s1' cluster and namespace 'ci-ska-tango-examples-8f8655ad', which will overwrite the latest deployment.

The job log shows the following steps:

- 1 Running with gitlab-runner 13.9.0 (2ebc4dc4)
- 2 on SKA-K8s-Runner fjHf7qKL
- 3 feature flags: FF\_GITLAB\_REGISTRY\_HELPER\_IMAGE:true
- 4 Resolving secrets (00:00)
- 5 (Collapsed)
- 6 Preparing the "kubernetes" executor (00:00)
- 7 Using Kubernetes namespace: gitlab
- 8 Using Kubernetes executor with image \$SKA\_K8S\_TOOLS\_DEPLOY\_IMAGE ...
- 9 (Collapsed)
- 10 Preparing environment
- 11 Waiting for pod gitlab/runner-fjhf7qkl-project-9673989-concurrent-14g66r to be running, status is Pending

The right sidebar provides details for the 'test' job, including a duration of 15 seconds, a timeout of 1h (from project), and the runner information: #7869064 (fjHf7qKL) SKA-K8s-Runner. It also shows the commit hash 8f8655ad and the pipeline #331771151 for master. Below the job details, a list of jobs is shown, including 'test' and 'unit-test'.

# Monitoring the performance of the devices



# Logging

- <https://k8s.stfc.skao.int/kibana/app/discover>
- Example query:
  - ▣ `kubernetes.namespace : "ci-ska-tango-examples-7ae62e9d" and kubernetes.labels.component: "theexample-test"`



# Elasticsearch - kibana

kubernetes.namespace: "ci-ska-tango-examples-7ae62e9d" and kubernetes.labels.component: "theexample-test" KQL Last 15 minutes Show dates Refresh

+ Add filter

filebeat-\*

Search field names

Filter by type 0

**Selected fields**

- message

**Available fields**

Popular

- @timestamp
- kubernetes.namespace
- kubernetes.pod.name
- log.file.path
- ska\_log\_message
- ska.application

agent.ephemeral\_id

agent.hostname

agent.id

agent.name

agent.type

agent.version

container.id

container.image.name

container.labels.org\_label-schema-build-\*\*\*

17 hits

Jul 5, 2021 @ 12:08:01.189 - Jul 5, 2021 @ 12:23:01.189 Auto


Time	message
Jul 5, 2021 @ 12:22:42.996	Ready to accept request
Jul 5, 2021 @ 12:22:42.996	1 2021-07-05T10:22:42.995Z INFO Dummy-7 always_executed_hook Motor.py#73  Connect to power Supply device
Jul 5, 2021 @ 12:22:42.996	1 2021-07-05T10:22:42.995Z INFO Dummy-7 get_device DevFactory.py#43  Creating Proxy for test/powersupply/1
Jul 5, 2021 @ 12:22:42.390	1 2021-07-05T10:22:42.389Z INFO MainThread init_device Motor.py#62  set_change_event on PerformanceValue
Jul 5, 2021 @ 12:22:42.384	1 2021-07-05T10:22:42.383Z INFO MainThread _update_state base_device.py#710 tango-device:test/eventreceiver/1 Device state changed from INIT to DISABLE
Jul 5, 2021 @ 12:22:42.384	1 2021-07-05T10:22:42.384Z INFO MainThread callbacks core.py#1108  Executed callback '<bound method _OpStateMachine._state_changed of <ska_tango_base.base.op_state_model._OpStateMachine object at 0x7faa3e350fd0>>'
Jul 5, 2021 @ 12:22:42.383	1 2021-07-05T10:22:42.295Z INFO MainThread enter core.py#125  Finished processing state DISABLE enter callbacks.
Jul 5, 2021 @ 12:22:42.295	1 2021-07-05T10:22:42.295Z INFO MainThread exit core.py#131  Finished processing state INIT_DISABLE exit callbacks.
Jul 5, 2021 @ 12:22:42.294	1 2021-07-05T10:22:42.294Z INFO MainThread do base_device.py#449 tango-device:test/eventreceiver/1 SKABaseDevice Init command completed OK
Jul 5, 2021 @ 12:22:42.294	1 2021-07-05T10:22:42.294Z INFO MainThread _call_do commands.py#366 tango-device:test/eventreceiver/1 Exiting command InitCommand with return_code ResultCode.OK, message: 'SKABaseDevice Init command completed OK'.
Jul 5, 2021 @ 12:22:42.293	1 2021-07-05T10:22:42.292Z INFO MainThread _update_state base_device.py#710 tango-device:test/eventreceiver/1 Device state changed from UNKNOWN to INIT

# Demo



# Quality aspects: Marvin


- Gitlab is able to send webhook when an event happen (i.e. a developer creates a branch)
- Marvin is an automation tool build with the FastAPI framework that is able to receive gitlab hook and add comment according to some checks performed.
- For example, for each branch we check that:
  - ▣ Documentation is updated,
  - ▣ There's an approval from a reviewer,
  - ▣ There's a like between the name of the branch and



Running inside  
k8s as any other  
ska applications!

# Nexus and the validation framework

- Based on celery (distributed system to process vast amounts of messages with tasks – processes always running), mongodb (for storing the validations made) and redis (for messages)
- When an artefact is pushed in Nexus, it triggers a web hook received by one of the celery workers that checks:
  - Naming Convention
  - Tag Convention
  - Metadata
  - We are currently working on a security/vulnerability



Running inside  
k8s as any other  
ska applications!

# Conclusion

- The SKA infrastructure developed provides a number of services such as:
  - horizontal scalability (thanks to k8s)
  - CI-CD (thanks to k8s and gitlab)
    - Automatic testing
    - Isolated environment for testing
    - Fast building and release docker images and helm charts only if tests passes
  - Monitoring (thanks to prometheus)
  - Logging (thanks to elasticsearch)
  - Storage (thanks to ceph)
  - Security
  - Central artefact repository
  - Quality aspect with Marvin and the validation framework