Developing a Tango device in a k8s context

Matteo Di Carlo INAF – Osservatorio Astronomico d’Abruzzo

Tango Controls Workshop @ ICALEPCS 2023 South Africa

SKA Deployment Practices

- Kubernetes (**k8s**) for container orchestration (kubernetes.io)
- **Helm** for packaging and deploying SKA Software *(helm.sh)*
  - A **chart** is a recipe to deploy the 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)
- Heavy use of **Makefile** (i.e., building, testing, deployment, …)
- **Gitlab** for CI/CD

Development environments for Kubernetes

There are a number of competing Kubernetes development environments - eg:

- Minikube - https://kubernetes.io/docs/tasks/tools/install-minikube/
- Kind - https://kind.sigs.k8s.io/docs/user/quick-start/
- Microk8s - https://microk8s.io/

Minikube still remains the most comprehensive option for a personal Kubernetes development environment

- It is based on kubeadm, the core Kubernetes cluster deployment tool, and tracks around 1-3 months behind k8s cluster point releases

Install Minikube

- Install docker-engine
  - https://docs.docker.com/engine/install/ubuntu/
- Deploy a Minikube development environment using make
  - https://gitlab.com/ska-telescope/sdi/ska-cicd-deploy-minikube
  - This repo is intended for ubuntu OS but works OK also in a WSL2 system

Detailed steps – install docker ubuntu

```
$ sudo apt-get update
$ sudo apt-get install -y ca-certificates curl gnupg make
$ sudo install -m 0755 -d /etc/apt/keyrings
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
$ sudo chmod a+r /etc/apt/keyrings/docker.gpg
$ echo \
  "deb [arch="$(dpkg --print-architecture)" signed-
by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
$ sudo apt-get update
$ sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-
plugin docker-compose-plugin
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
$ newgrp docker
$ sudo service docker start
$ docker run hello-world
```

Detailed steps – install minikube

```
$ git clone https://gitlab.com/ska-telescope/sdi/ska-cicd-deploy-minikube/
$ cd ska-cicd-deploy-minikube/
$ git submodule update --init --recursive
$ echo "MEM=6144" >> PrivateRules.mak
$ echo "DRIVER=docker" >> PrivateRules.mak
$ make all
```

```
Minikube Installed: Yes!
Helm Installed:     Yes!
DRIVER:             docker
RUNTIME:            docker
ADDONS:             --addons=logviewer --addons=metrics-server --addons=ingress
CPUS:               4
MEM:                6144
OS_NAME:            linux
OS_ARCH:            x86_64
OS_BIN:             amd64
EXE_DIR:            /usr/local/bin
SUDO_FOR_EXE_DIR:   sudo --preserve-env=http_proxy --preserve-env=https_proxy
IPADDR:             172.19.24.208
MINIKUBE_IP:        192.168.49.2
HOSTNAME:           MattZenBook1.
FQDN:               MattZenBook1..local.net
MOUNT_FROM:         /srv
MOUNT_TO:           /srv
PROXY_VERSION:      2.8
PROXY_CONFIG:       /home/ubuntu/.minikube/minikube-nginx-haproxy.cfg
MINIKUBE_VERSION:   v1.30.1
KUBERNETES_VERSION: v1.27.3
KUBERNETES_SERVER_VERSION: v1.27.3
HELM_VERSION:       v3.12.1
HELMFILE_VERSION:   0.155.0
YQ_VERSION:         4.34.1
INGRESS:            http://192.168.49.2
USE_CACHE:
CACHE_DATA:         /home/ubuntu/.minikube/registry_cache
Minikube status:
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

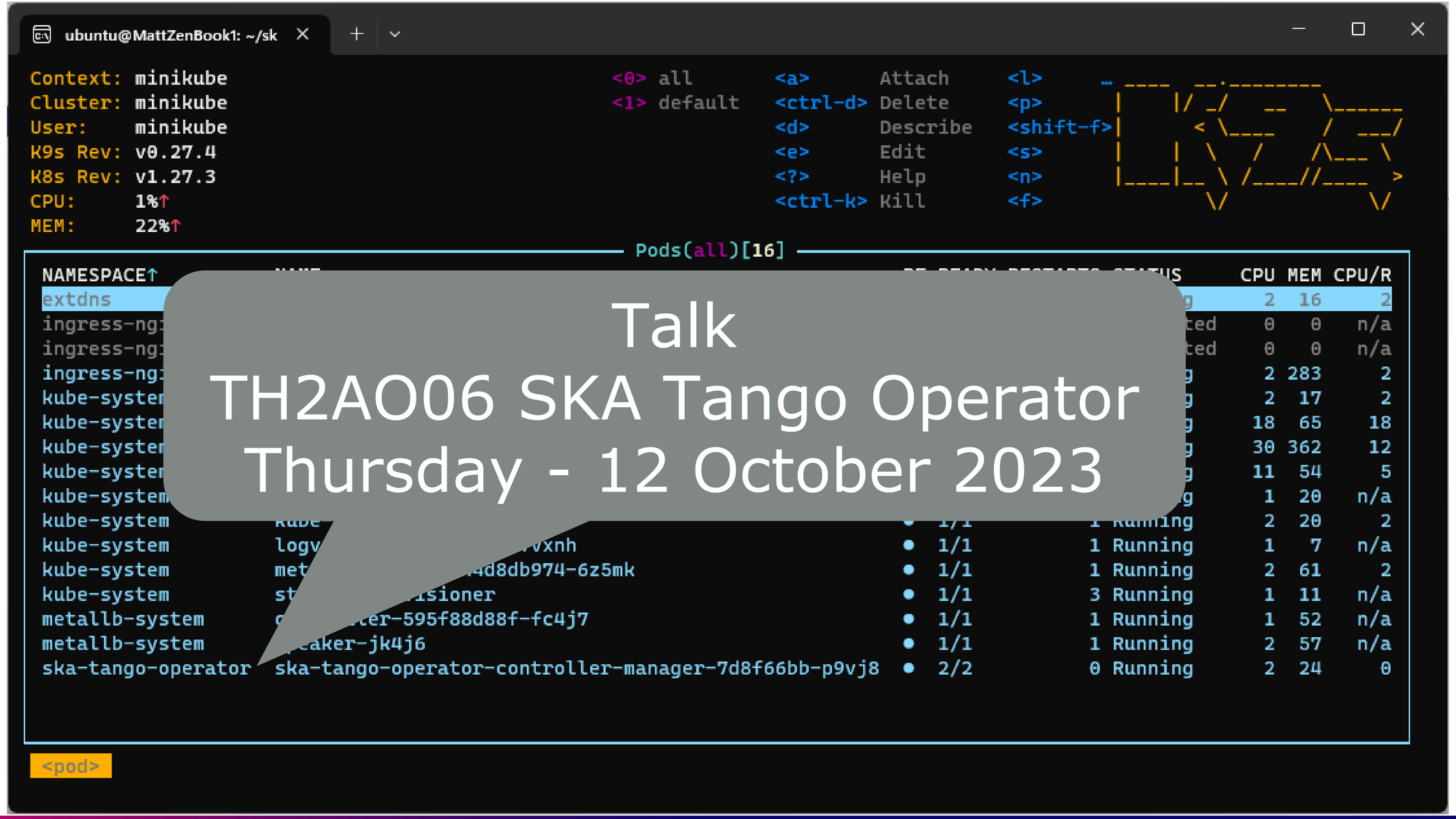
Detailed steps – WSL2

```
# Make sure system is configured and resolv.conf not automatically generated
$ cat /etc/wsl.conf
[boot]
systemd=true
[network]
generateResolvConf = false

# setup name server
sudo -s
apt update -y
systemctl disable --now systemd-resolved
rm -rf /etc/resolv.conf
echo "nameserver 8.8.8.8" > /etc/resolv.conf
apt install dnsmasq dnsutils ldnsutils -y
echo "server=8.8.8.8" >> /etc/dnsmasq.conf
echo "server=1.1.1.1" >> /etc/dnsmasq.conf
echo "server=1.0.0.1" >> /etc/dnsmasq.conf
echo "server=/svc.cluster.local/$(kubectl get svc --namespace extdns extdns-coredns -o
jsonpath='{.status.loadBalancer.ingress[0].ip}') # minikube" >> /etc/dnsmasq.conf
rm -rf /etc/resolv.conf
ln -s /run/resolvconf/resolv.conf /etc/resolv.conf
systemctl restart dnsmasq
```

What exactly did we install?

- Minikube https://github.com/kubernetes/minikube/releases
  - logviewer - simple logview available on port 32000 - browse with `sensible-browser http://$(minikube ip):32000`
  - metrics-server - simple metrics server
  - ingress - NGINX based Ingress Controller for exposing HTTP/HTTPS services
  - metallb - enable creation of `LoadBlancer` type `Service` resources to expose application ports out of Kubernetes. This is deployed in conjunction with a DNS responder (`extdns`) that can be integrated with the local users DNS settings to have automatic name resolution for these services.
- Helm https://github.com/helm/helm/releases
- Haproxy https://registry.hub.docker.com/_/haproxy
- K8s tools https://kubernetes.io/releases/

k9s

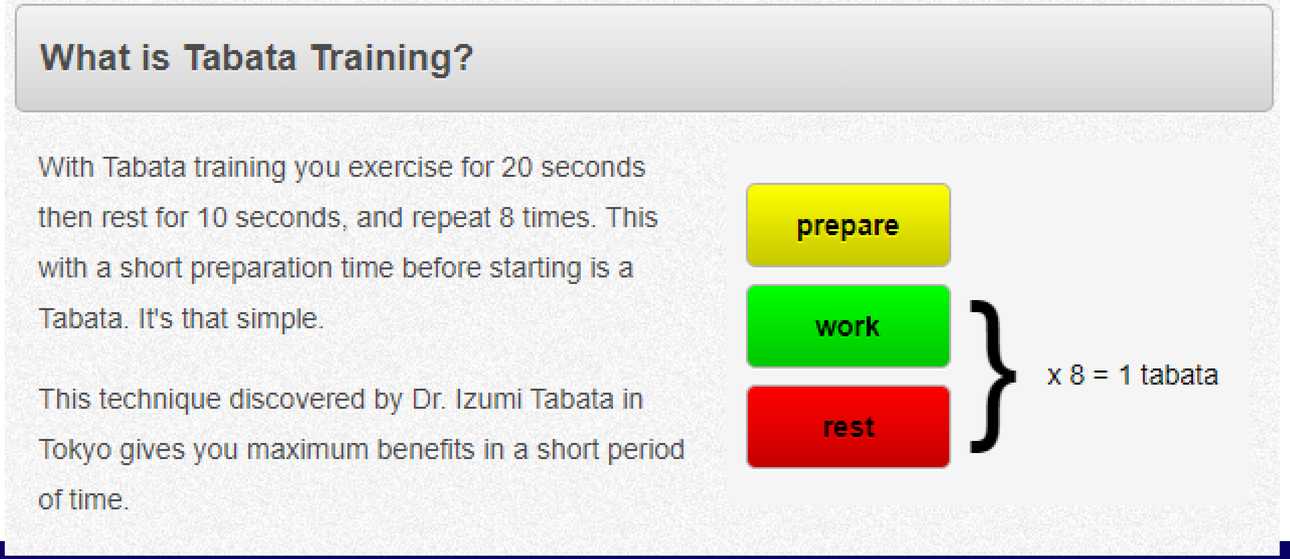

ubuntu@MattZenBook1: ~/sk
Context: minikube
Cluster: minikube
User: minikube
K9s Rev: v0.27.4
K8s Rev: v1.27.3
CPU: 1%↑
MEM: 22%↑
<0> all
<1> default
<a> Attach
<ctrl-d> Delete
<d> Describe
<e> Edit
<?> Help
<ctrl-k> Kill
<l>
<p>
<shift-f>
<s>
<n>
<f>
Pods(all)[16]
NAMESPACE↑
CPU MEM CPU/R
extdns
kube-system
metallb-system
ska-tango-operator
ska-tango-operator-controller-manager-7d8f66bb-p9vj8 ● 2/2 0 Running 2 24 0
<pod>
Talk
TH2AO06 SKA Tango Operator
Thursday - 12 October 2023

ska-tango-examples repository

- Demonstrates how to structure a project that provides some simple Tango devices coded in PyTango in k8s.
- Use k8s for development and testing so that the build environment, test environment and test results are all completely reproducible and are independent of host environment.
- List of TANGO examples that demonstrate some features of the framework as starting point for SKA developers in learning it
- We also use it for testing new version of the framework

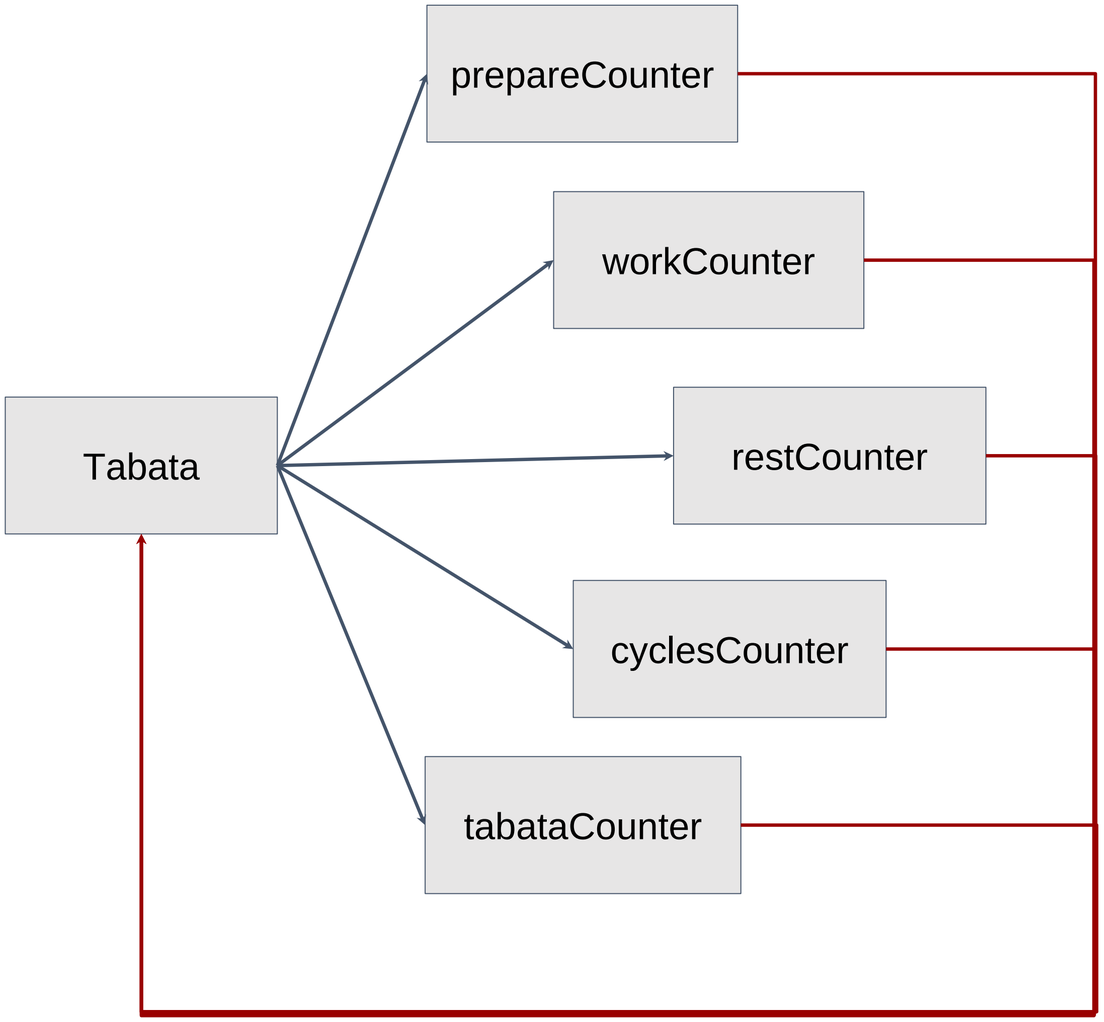
The Tabata device

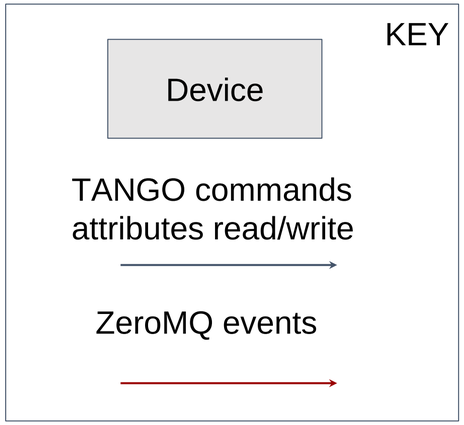
- One of the most complete example is the tabata
- It is a realization of a gym workout
  - more information at https://en.wikipedia.org/wiki/High-intensity_interval_training.
  - https://www.tabatatimer.com/



The Tabata device

The Tabata device





The counter device

- This Device demonstrate the use of the TANGO event mechanism to send change events to clients.
- There's also a device attribute in polling so that events for that attribute are sent automatically.
- Commands:
  - *Increment*
  - *Decrement*
  - *CounterReset*
- Attributes:
  - *value (only read)*
  - *polled_value (only read)*
  - *fire_event_at (read/write)*

The Tabata device: attributes

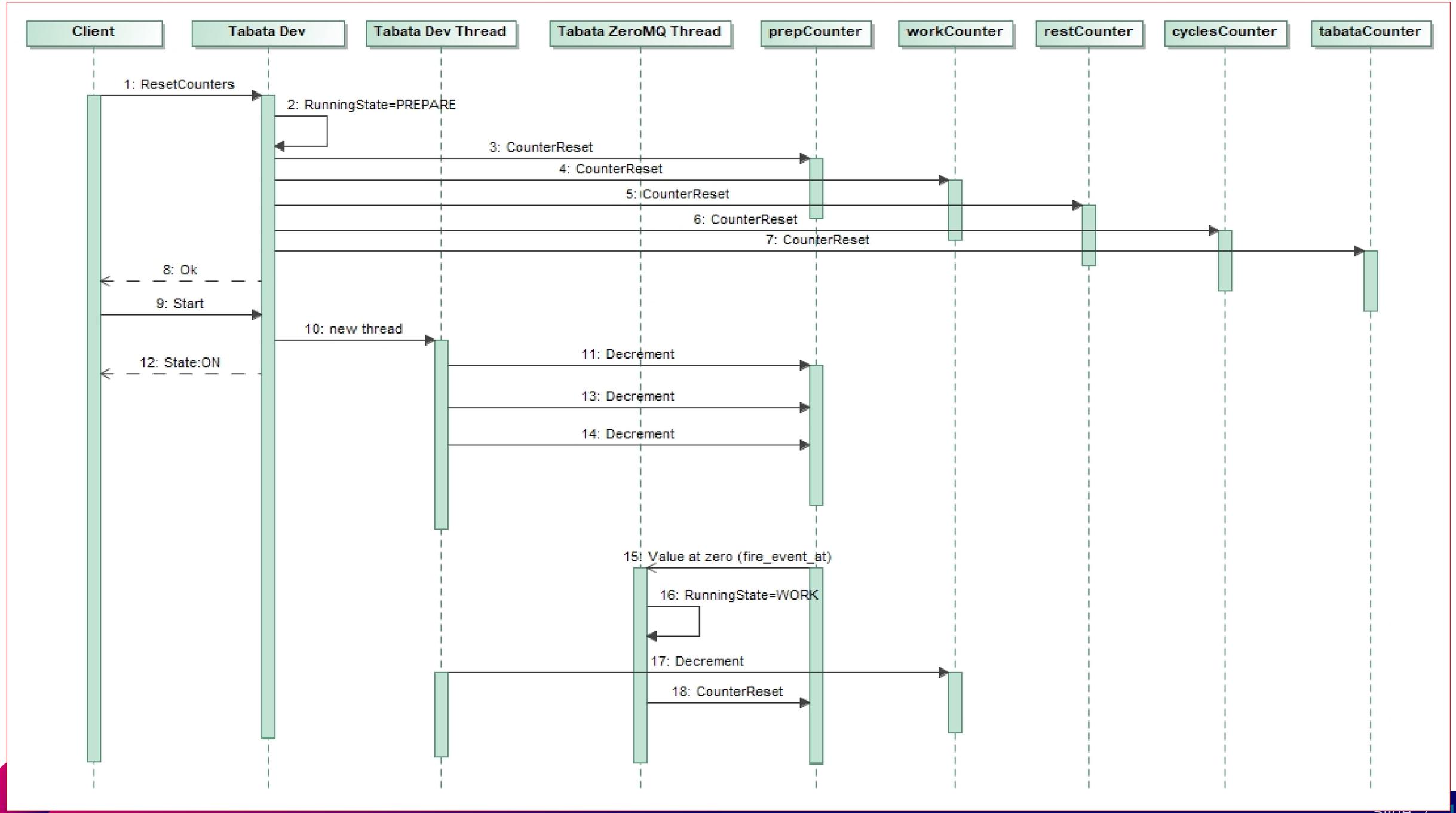
- Running_state (PREPARE, WORK, REST)
- State (ON, OFF)
- tabatas: for counter initialization
- cycles: for counter initialization
- rest: for counter initialization
- work: for counter initialization
- prepare: for counter initialization

The Tabata device: properties

- prepCounter: device name for the prepare counter
- workCounter: device name for the work counter
- restCounter: device name for the rest counter
- cyclesCounter: device name for the cycles counter
- tabatasCounter: device name for the tabatas counter
- sleep_time: to speed the execution during tests

The Tabata Device: commands

- Start: start a python thread for interacting with the counters
- Stop: stop the python thread for interacting with the counters
- ResetCounters: reset the counters to the related attributes


Client
Tabata Dev
Tabata Dev Thread
Tabata ZeroMQ Thread
prepCounter
workCounter
restCounter
cyclesCounter
tabataCounter
1: ResetCounters
2: RunningState=PREPARE
3: CounterReset
4: CounterReset
5: CounterReset
6: CounterReset
7: CounterReset
8: Ok
9: Start
10: new thread
11: Decrement
12: State:ON
13: Decrement
14: Decrement
15: Value at zero (fire_event_at)
16: RunningState=WORK
17: Decrement
18: CounterReset

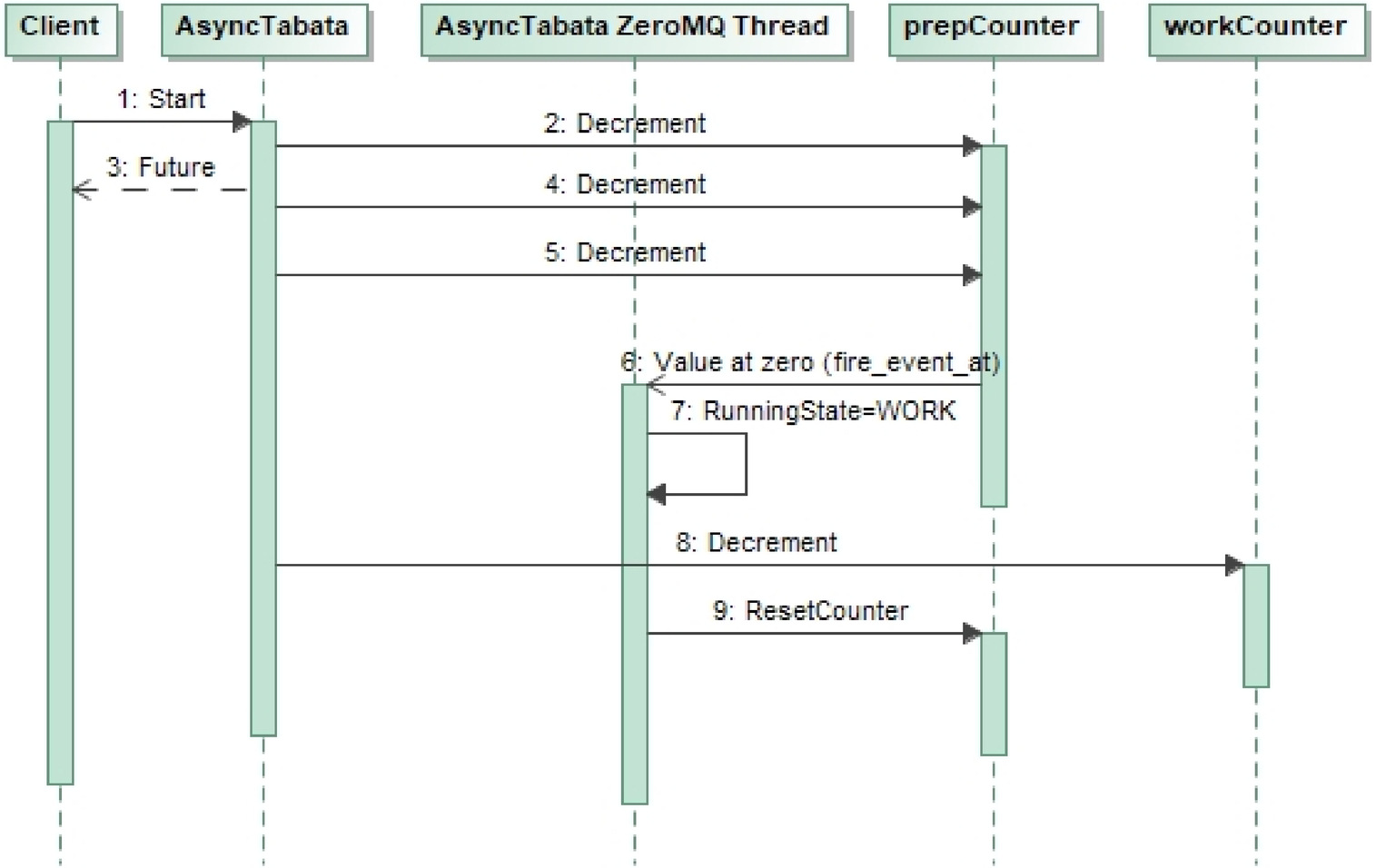
The AsyncTabata Device

- Same as Tabata but the realization is asynchonous.
- The tabata device has 2 commands: Run and Stop.
- The run executes the entire job so it's not possible to use it without an async command.

Notes on serialization model

- It is not common to change the default behaviour, usually commands are always very quick and, in case a long job must be run, a thread can be used for the execution with the necessary lock mechanism.
- Anyway, while the Tabata device uses the default serialization model, the AsyncTabata changes the default to no synchronization.
- https://pytango.readthedocs.io/en/stable/green_modes/green_modes_server.html
- https://tango-controls.readthedocs.io/en/latest/development/advanced/threading.html#serialization-model-within-a-device-server

The AsyncTabata Device sequence diagram



ska-tango-examples structure

- Folders:
  - src: source code, i.e. src/ska_tango_examples/tabata/Tabata.py
  - tests: test code, i.e. tests/integration/test_tabata.py
  - charts: helm charts for installing into k8s
  - docs: documentation
  - .make: ska makefile submodule for automation
- In the root folder:
  - Dockerfile
  - pyproject.toml

Poetry - pyproject.toml

- Poetry is a tool for dependency management and packaging in Python. It allows you to declare the libraries your project depends on and it will manage (install/update) them for you. Poetry offers a lockfile to ensure repeatable installs, and can build your project for distribution.

Poetry - pyproject.toml

- Poetry is a tool for depend
packaging in Python. It al
libraries your project depe
(install/update) them for y
ensure repeatable installs
for distribution.

```
[tool.poetry]
name = "ska-tango-examples"
version = "0.4.28"
description = "SKA Tango Examples"

[…]

[tool.poetry.dependencies]
python = "^3.9"
pytango = "^9.4.2"
ska-tango-base = "^0.12.0"
ska-ser-log-transactions = "*"
numpy = "1.23.0"
debugpy = "^1.5.1"
```

OCI image - Dockerfile

```
ARG BUILD_IMAGE="artefact.skao.int/ska-tango-images-pytango-builder:9.4.3"
ARG BASE_IMAGE="artefact.skao.int/ska-tango-images-pytango-runtime:9.4.3"
FROM $BUILD_IMAGE AS buildenv
FROM $BASE_IMAGE
USER root

WORKDIR /app

COPY --chown=tango:tango pyproject.toml poetry.lock ./
RUN poetry export --format requirements.txt --output poetry-requirements.txt --
without-hashes && \
    sed -i '/pytango/d' poetry-requirements.txt && \
    sed -i '/numpy/d' poetry-requirements.txt && \
    pip install -r poetry-requirements.txt && \
    rm poetry-requirements.txt

COPY --chown=tango:tango src ./

USER tango
```

Testing

- Encapsulated in the Makefile
- It uses pytest with no bdd
- It uses pytest fixture and a factory pattern for creating the right device context
- Unit (no install required) testing with
  - `$ make python-test`
- Integration (install required) testing with
  - `$ make k8s-test`

DevFactory class

- It is a factory class which provide the ability to create an object of type DeviceProxy.
- When testing the static variable _test_context is an instance of the TANGO class MultiDeviceTestContext (done with pytest fixture).
- More information on tango testing can be found at the following link: https://pytango.readthedocs.io/en/stable/testing.html

conftest.py

Dictionary fixture present in the test files

```
@pytest.fixture
def tango_context(devices_to_load, request):
    true_context = request.config.getoption("--true-context")
    logging.info("true context: %s", true_context)
    if not true_context:
        with MultiDeviceTestContext(devices_to_load, process=False) as context:
            DevFactory._test_context = context
            yield context
    else:
        yield None
```

Install the ska-tango-examples

- Kubernetes (k8s) for container orchestration (kubernetes.io)
  - Kubernetes Service == TANGO Device Server
- Helm for packaging SKA k8s applications (helm.sh)
  - Each SKA element provides an helm chart for running it in k8s
  - Helm has the concept of dependency: a chart can have one or more sub-charts

ska-tango-examples dependencies

The ska-tango-util helm chart is a library chart which helps other application chart defines TANGO device servers.

```
dependencies:
- name: ska-tango-util
  version: 0.4.7
  repository: https://artefact.skao.int/repository/helm-internal
- name: ska-tango-base
  version: 0.4.7
  repository: https://artefact.skao.int/repository/helm-internal
  condition: ska-tango-base.enabled,global.sub-system.ska-tango-base.enabled
```

Application chart which defines the basic TANGO ecosystem in kubernetes.

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: it is an interactive Tango client
tangotest: it is the tango test device server

Declare the Device Servers

Name is the k8s name of the resources

The list of dependencies: devices or simple host and port

Command or entry points of the device servers (if more than one entry point is specified, we are referring to a multi-devices DS)

The server definition, it can indicates the list of instances, devices, classes, etc

The container image to use

How to check when the DS is ready or in failure

```
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-exam les
  tag: 0.4.24
  pullPolicy: IfNotPr
livenessProbe:
  in
  perio
  tim
  s
  failu
readi
  initialDe ion : 0
  periodSec s: 10
  timeoutSeconds: 3
  successThreshold: 1
  failureThreshold: 3
```

Partial set of parameters to set!

ska-tango-examples dev workflow

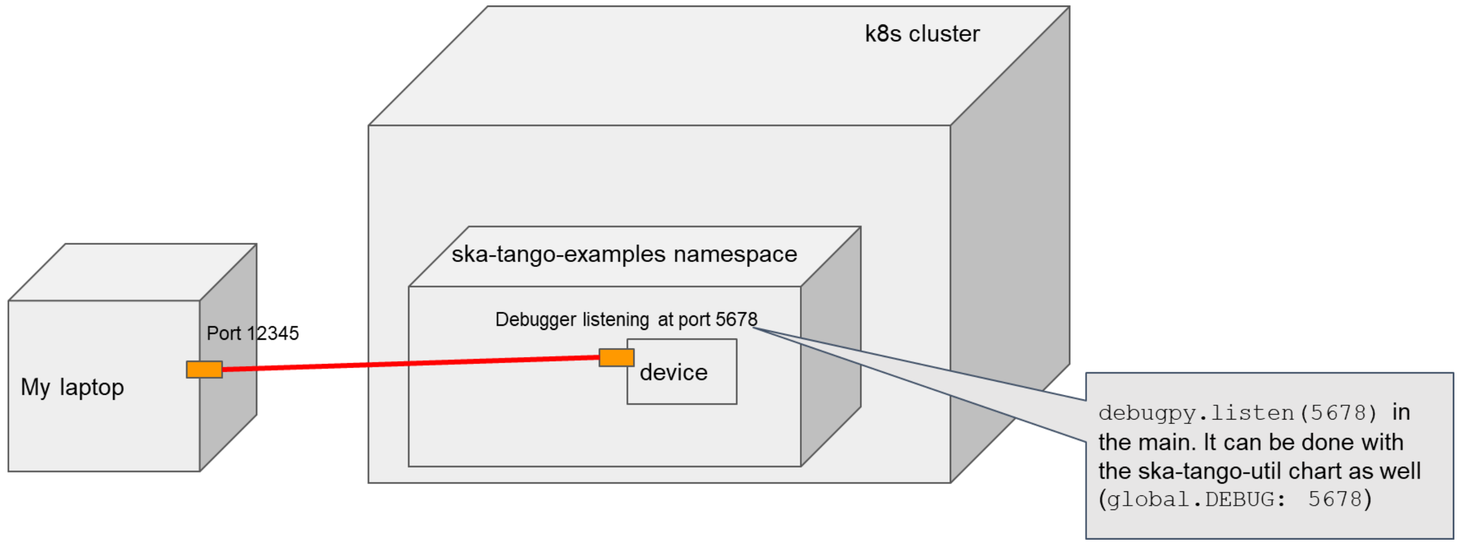
```
$ git clone https://gitlab.com/ska-telescope/ska-tango-examples.git
$ cd ska-tango-examples
$ git submodule update --init --recursive
$ eval $(minikube docker-env)
$ curl -sSL https://install.python-poetry.org | python3 -
$ poetry install; poetry shell

$ make python-test

$ make oci-build

$ make k8s-install-chart

$ make k8s-watch; make k8s-wait

$ make k8s-test

$ make k8s-uninstall-chart
```

Common tools available

- Jive
- Pogo
- Logviewer
- …

Debugging - debugpy library

- It is an adapter of the pydevd used in PyCharm: https://github.com/microsoft/debugpy
- How it works:
  - CLI: python3 -m debugpy --listen localhost:5678 mydevice.py
  - From code:
    - import debugpy
    - debugpy.listen(5678)



debug_this_thread

- A TANGO Device server does not use the python threads so they are not debuggable unless we make them aware of the debugger.
- https://github.com/microsoft/debugpy/wiki/API-Reference#debug_this_thread
- Makes the debugger aware of the current thread, and start tracing it. Must be called on any background thread that is started by means other than the usual Python APIs (i.e. the threading module), in order for breakpoints to work on that thread.

We recognise and acknowledge the Indigenous peoples and cultures that have traditionally lived on the lands on which our facilities are located.

SKAO

www.skao.int