DEVELOPING A TANGO DEVICE IN A K8S CONTEXT

Matteo Di Carlo
INAF – Osservatorio Astronomico d'Abruzzo
39th Tango Community Meeting at INAF
Giulianova, Province of Teramo, Italy

CONTEXT - 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
- Heavy use of **Makefile** (i.e., building, testing, deployment, …)
- **Gitlab** for CI/CD

KUBERNETES IN FEW WORDS

- Microservice architecture
  - Normally includes more than one machine (virtual or not)
  - The fundamental concept is the pod (which is a group of container)
  - A service represent the point of contact of one or more pods distributed in one or more nodes
- In TANGO we assume a Kubernetes Service == TANGO Device Server
  - Even if stateful.

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

INSTALL MINIKUBE – MAINLY 2 STEPS

- 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 22/24 but works OK also in a WSL2 system

DETAILED STEPS – INSTALL DOCKER-CE UBUNTU

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install 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/
$ make
$ echo "MEM=6144" >> PrivateRules.mak
$ echo "DRIVER=docker" >> PrivateRules.mak
$ echo “KYVERNO=no" >> PrivateRules.mak
$ make all
```

```
Minikube Installed:         Yes!
Helm Installed:             Yes!
Profile:                    minikube
DRIVER:                     docker
RUNTIME:                    docker
HAPROXY_ENGINE:             docker
...
IPADDR:                     10.255.255.254
CLUSTER_IP:                 192.168.58.2
HOSTNAME:                   MattZenBook1.
CLUSTER_DOMAIN:             cluster.local
...
KUBERNETES_VERSION:         v1.31.4
KUBERNETES_SERVER_VERSION: v1.31.4
HELM_VERSION:               v3.16.4
HELMFILE_VERSION:           0.169.2
YQ_VERSION:                 4.44.6
KYVERNO_VERSION:            1.13.2
CILIUM_CLI_VERSION:         0.16.22
INGRESS:                    http://192.168.58.2
USE_CACHE:
CACHE_DATA:                 /home/ubuntu/.minikube/registry_cache
KIND_VERSION:               v0.26.0
KIND_DRIVER:                docker
Minikube status:
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

DETAILED STEPS – WSL2

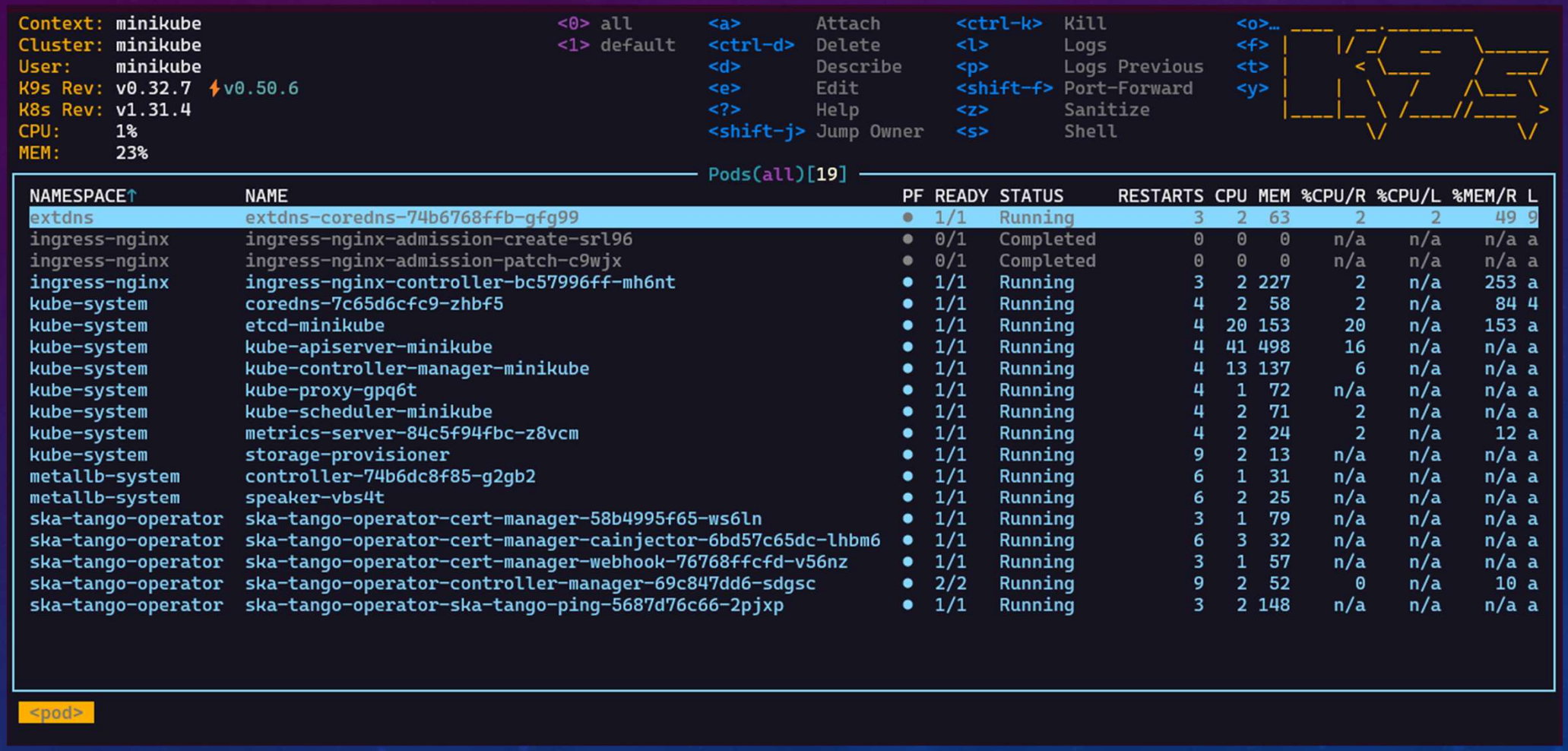
```
# Make sure system is configured and resolv.conf not automatically generated
$ cat /etc/wsl.conf
[boot]
systemd=true
[network]
generateResolvConf = false

$ sudo -s
$ echo 'nameserver 8.8.8.8' >> /etc/resolv.conf
$ echo 'nameserver 1.1.1.1' >> /etc/resolv.conf
$ echo 'nameserver $(kubectl get svc -n extdns  extdns-coredns -o
jsonpath='{.status.loadBalancer.ingress[0].ip}')' >> /etc/resolv.conf
$ chattr +i /etc/resolv.conf
```

WHAT EXACTLY DID WE INSTALL?

- Minikube https://github.com/kubernetes/minikube/releases
  - 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
- kubectl v1.31.4
- helmfile 0.169.2
- K9s v0.32.7
- jq 4.44.6
- Kind v0.26.0

K9S


Context: minikube
Cluster: minikube
User: minikube
K9s Rev: v0.32.7 v0.50.6
K8s Rev: v1.31.4
CPU: 1%
MEM: 23%
<0> all
<1> default
<a> Attach
<ctrl-d> Delete
<d> Describe
<e> Edit
<?> Help
<shift-j> Jump Owner
<ctrl-k> Kill
<l> Logs
<p> Logs Previous
<shift-f> Port-Forward
<z> Sanitize
<s> Shell
<o>…
<f>
<t>
<y>
Pods(all)[19]
NAMESPACE↑ NAME PF READY STATUS RESTARTS CPU MEM %CPU/R %CPU/L %MEM/R L
extdns extdns-coredns-74b6768ffb-gfg99 ● 1/1 Running 3 2 63 2 2 49 9
ingress-nginx ingress-nginx-admission-create-srl96 ● 0/1 Completed 0 0 0 n/a n/a n/a a
ingress-nginx ingress-nginx-admission-patch-c9wjx ● 0/1 Completed 0 0 0 n/a n/a n/a a
ingress-nginx ingress-nginx-controller-bc57996ff-mh6nt ● 1/1 Running 3 2 227 2 n/a 253 a
kube-system coredns-7c65d6cfc9-zhbf5 ● 1/1 Running 4 2 58 2 n/a 84 4
kube-system etcd-minikube ● 1/1 Running 4 20 153 20 n/a 153 a
kube-system kube-apiserver-minikube ● 1/1 Running 4 41 498 16 n/a n/a a
kube-system kube-controller-manager-minikube ● 1/1 Running 4 13 137 6 n/a n/a a
kube-system kube-proxy-gpq6t ● 1/1 Running 4 1 72 n/a n/a n/a a
kube-system kube-scheduler-minikube ● 1/1 Running 4 2 71 2 n/a n/a a
kube-system metrics-server-84c5f94fbc-z8vcm ● 1/1 Running 4 2 24 2 n/a 12 a
kube-system storage-provisioner ● 1/1 Running 9 2 13 n/a n/a n/a a
metallb-system controller-74b6dc8f85-g2gb2 ● 1/1 Running 6 1 31 n/a n/a n/a a
metallb-system speaker-vbs4t ● 1/1 Running 6 2 25 n/a n/a n/a a
ska-tango-operator ska-tango-operator-cert-manager-58b4995f65-ws6ln ● 1/1 Running 3 1 79 n/a n/a n/a a
ska-tango-operator ska-tango-operator-cert-manager-cainjector-6bd57c65dc-lhbm6 ● 1/1 Running 6 3 32 n/a n/a n/a a
ska-tango-operator ska-tango-operator-cert-manager-webhook-76768ffcfd-v56nz ● 1/1 Running 3 1 57 n/a n/a n/a a
ska-tango-operator ska-tango-operator-controller-manager-69c847dd6-sdgsc ● 2/2 Running 9 2 52 0 n/a 10 a
ska-tango-operator ska-tango-operator-ska-tango-ping-5687d76c66-2pjxp ● 1/1 Running 3 2 148 n/a n/a n/a a
<pod>

TANGO TOOLS WITH DOCKER

- Jive

```
docker run --network host --user 1000:1000 --volume="/home/ubuntu:/home/tango" --volume="/tmp/.X11-unix:/tmp/.X11-unix" --env="DISPLAY=:0" artefact.skao.int/ska-tango-images-tango-jive:7.36.3
```

- Pogo

```
docker run --network host --user 1000:1000 --volume="/home/ubuntu:/home/tango" --volume="/tmp/.X11-unix:/tmp/.X11-unix" --env="DISPLAY=:0" artefact.skao.int/ska-tango-images-tango-pogo:9.8.3
```

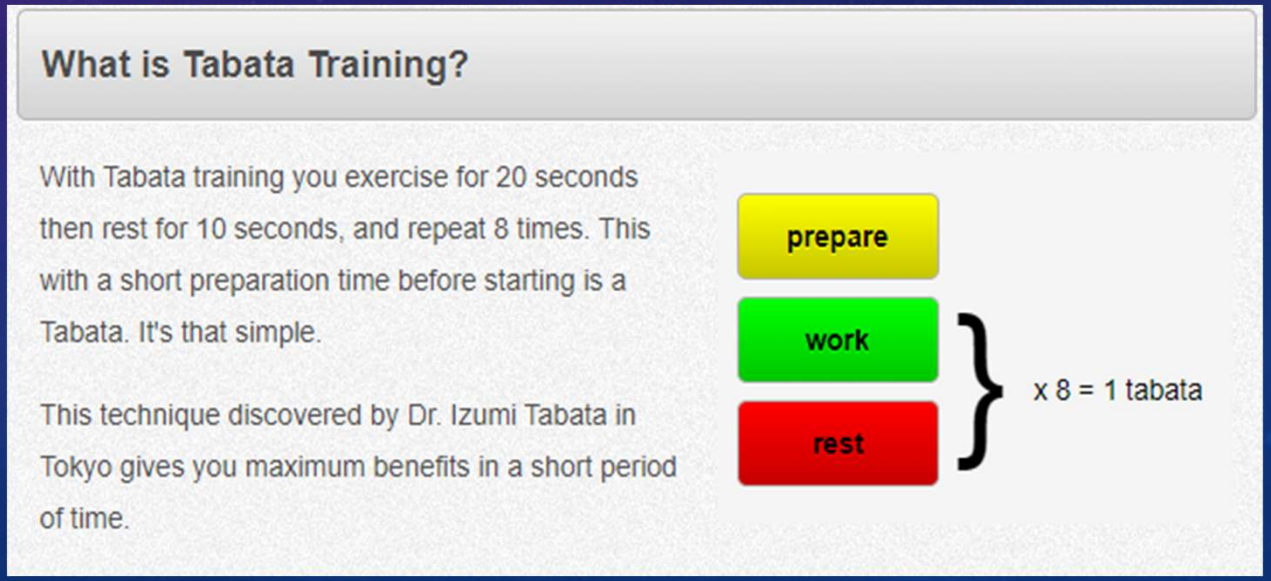
LET’S CREATE A SIMPLE TANGO DS

A MORE COMPLEX EXAMPLES: 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 used to use it for testing new versions of the framework

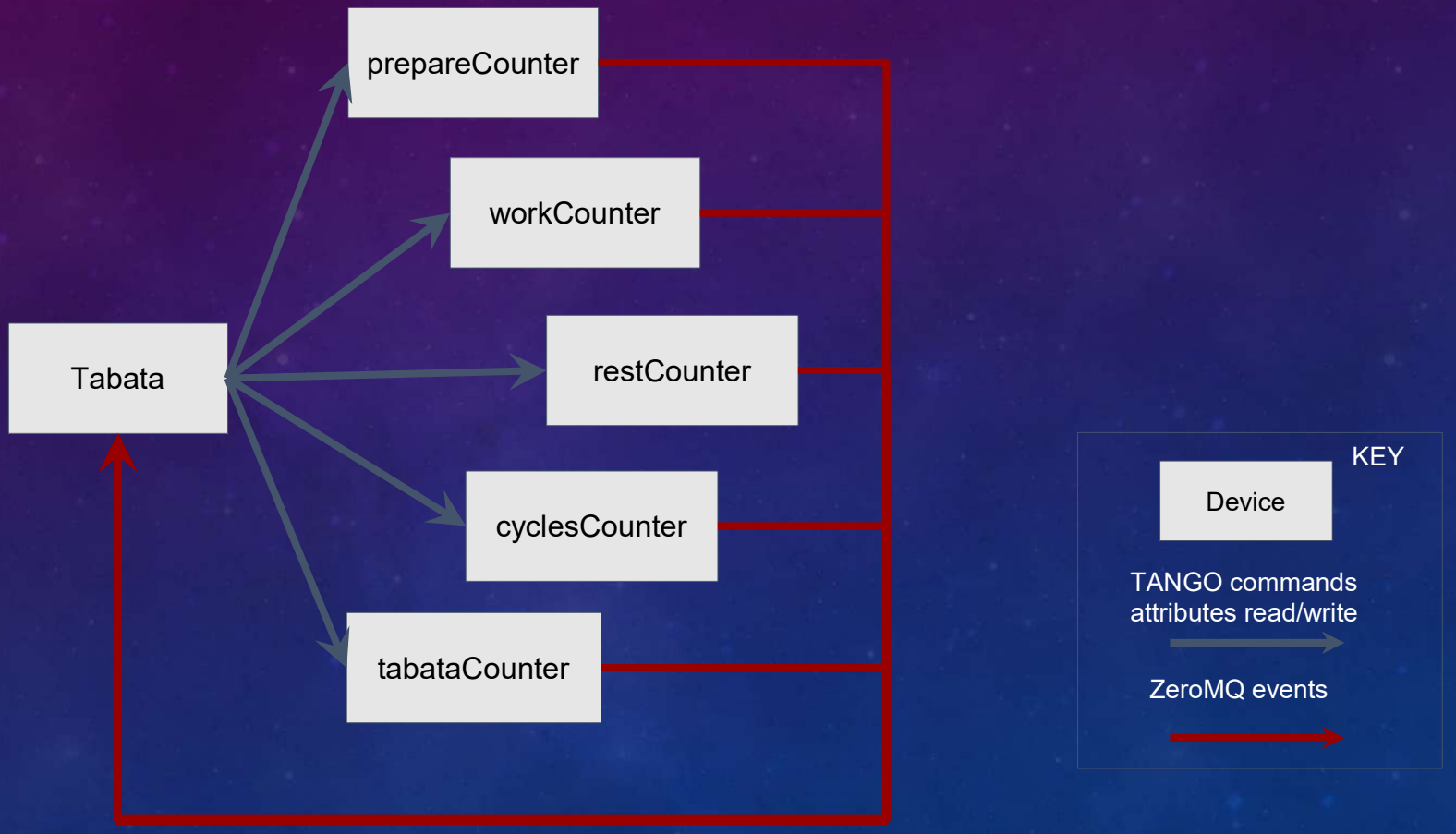
THE TABATA DEVICE

- 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

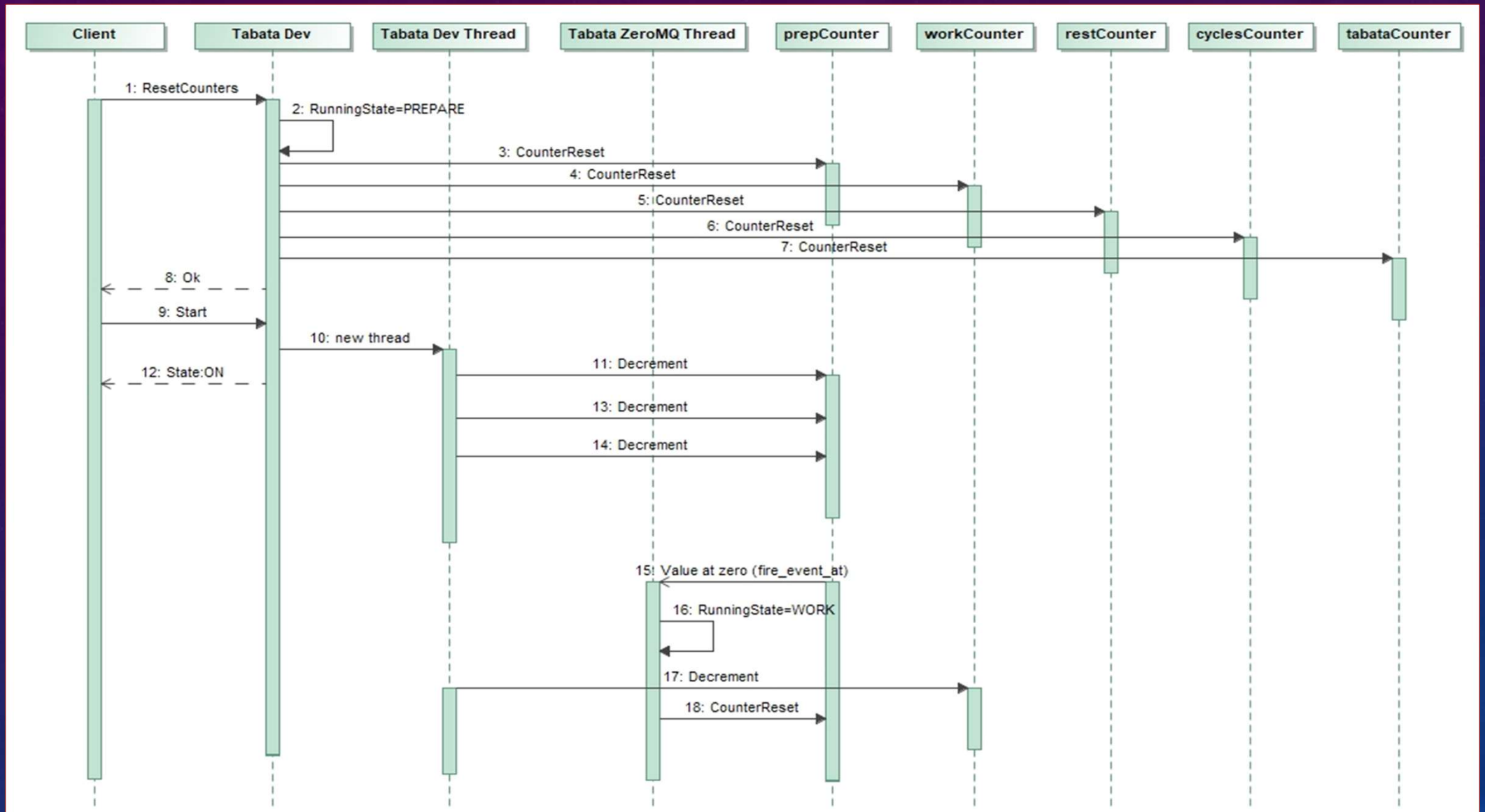
- Running_state (PREPARE, WORK, REST)
- State (ON, OFF)
- tabatas: for counter initialization
- cycles: for counter initialization
- rest: for counter initialization
- work: for counter initialization
- prepare: for counter initialization

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

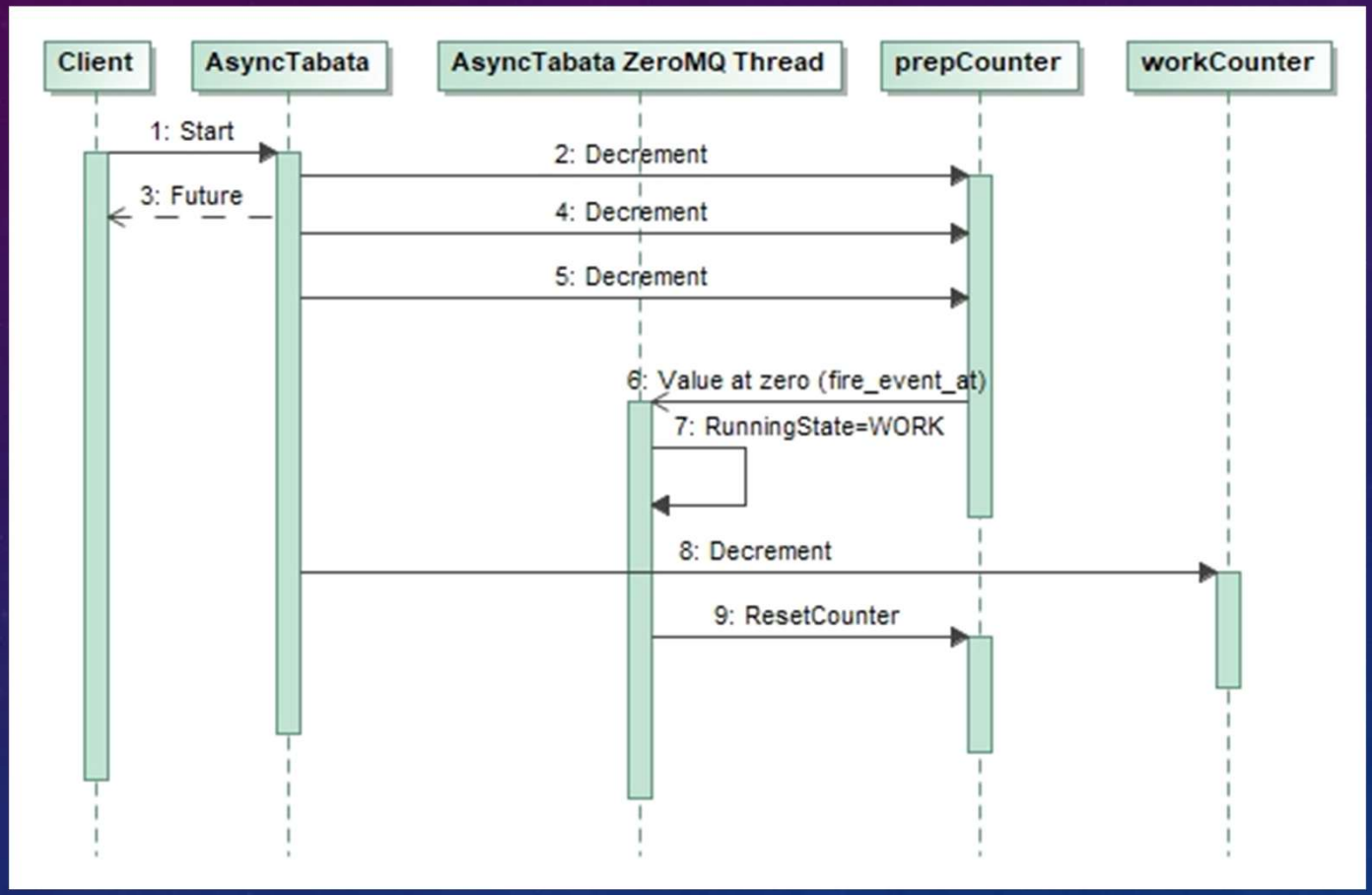
THE ASYNCTABATA DEVICE

- 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, 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 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.

```
[tool.poetry]
name = "ska-tango-examples"
version = "0.5.2"
description = "SKA Tango Examples"

[…]

[tool.poetry.dependencies]
python = ">=3.10,<3.13"
pytango = "^9.5.0"
numpy = "1.23.0"
debugpy = "^1.5.1"
```

OCI IMAGE - DOCKERFILE

```
FROM artefact.skao.int/ska-tango-images-tango-dsconfig:1.5.13 as tools

FROM artefact.skao.int/ska-build-python:0.1.3 as build

WORKDIR /app

COPY pyproject.toml poetry.lock ./

...

RUN poetry install --no-root

FROM artefact.skao.int/ska-python:0.1.4

...

COPY --from=build ${VIRTUAL_ENV} ${VIRTUAL_ENV}

ENV PYTHONPATH="/app/src:app/.venv/lib/python3.10/site-packages/:${PYTHONPATH}"
```

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
```

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: 0.4.19
  repository: https://artefact.skao.int/repository/helm-internal
- name: ska-tango-base
  version: 0.4.19
  repository: https://artefact.skao.int/repository/helm-internal
  condition: ska-tango-base.enabled,global.sub-system.ska-tango-base.enabled
```

Ska-tango-base 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

```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-exam les
  tag: 0.4.24
  pullPolicy: IfNotPr
livenessProb
  in
  perio
  time
  failu
readi
  initialDe on: 0
  periodSec s: 10
  timeoutSeconds: 3
  successThreshold: 1
  failureThreshold: 3
```

Partial set of parameters to set!

SKA-TANGO-EXAMPLES DEV WORKFLOW

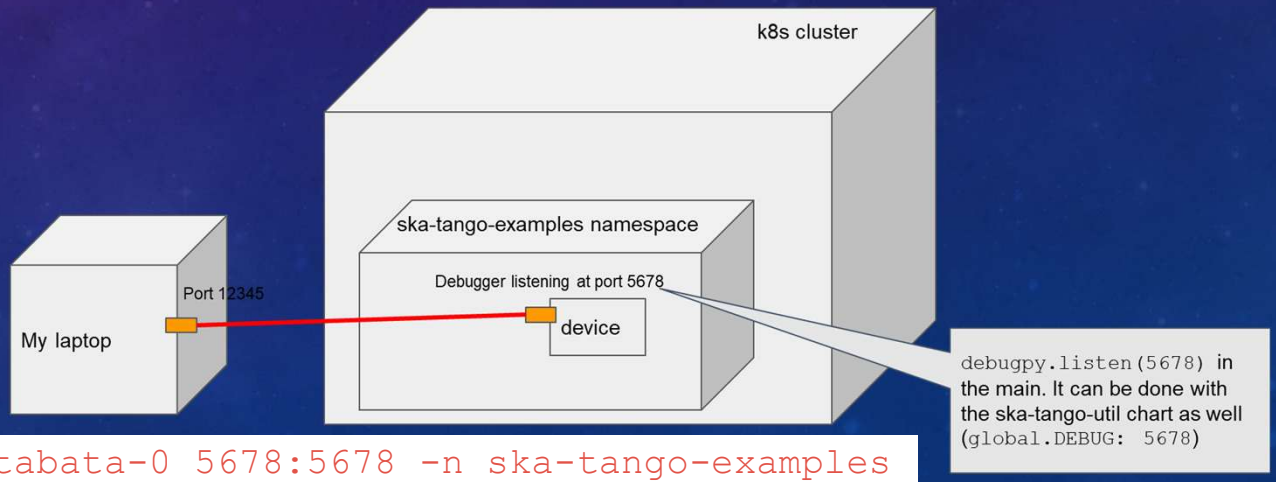
```
$ sudo apt update && sudo apt install pipx
$ pipx install poetry
$ git clone https://gitlab.com/ska-telescope/ska-tango-examples.git
$ cd ska-tango-examples
$ git submodule update --init --recursive
$ poetry install; eval $(poetry env activate)
$ poetry self add poetry-plugin-export
$ make python-test

$ eval $(minikube docker-env)
$ make oci-build
$ sudo apt install jq
$ echo 'CLUSTER_DOMAIN=cluster.local' >> PrivateRules.mak
$ make k8s-install-chart; make k8s-wait
$
$ make k8s-test

$ make k8s-uninstall-chart
```

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)




```
$ kubectl port-forward pod/ds-tabata-tabata-0 5678:5678 -n ska-tango-examples
```

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.

Thanks!

matteo.dicarlo@inaf.it

OSSERVATORIO ASTRONOMICO D'ABRUZZO
INAF ISTITUTO NAZIONALE DI ASTROFISICA