Tango Workshop @ ICALEPCS 2021
PyTango and jupyTango

October 2021

Anton Joubert
NRF/SARAO

Nicolas Leclercq
ESRF

Acknowledgements

Sergi Rubio (ALBA) https://github.com/sergirubio

Vincent Michel https://github.com/vxgmichel

Karoo Team (SARAO)

Agenda

Introduction

Docker compose environment

Simple Tango device servers

~~API: Low-level vs. High-level~~

ITango for easy client access

JupyTango

~~Events and polling~~

~~Miscellaneous~~

~~How to test?~~

Additional resources

~~Strikethrough~~ items: in slide deck, but won't be covered today

Introduction

What is PyTango?

Python library

Binding over the C++ tango library

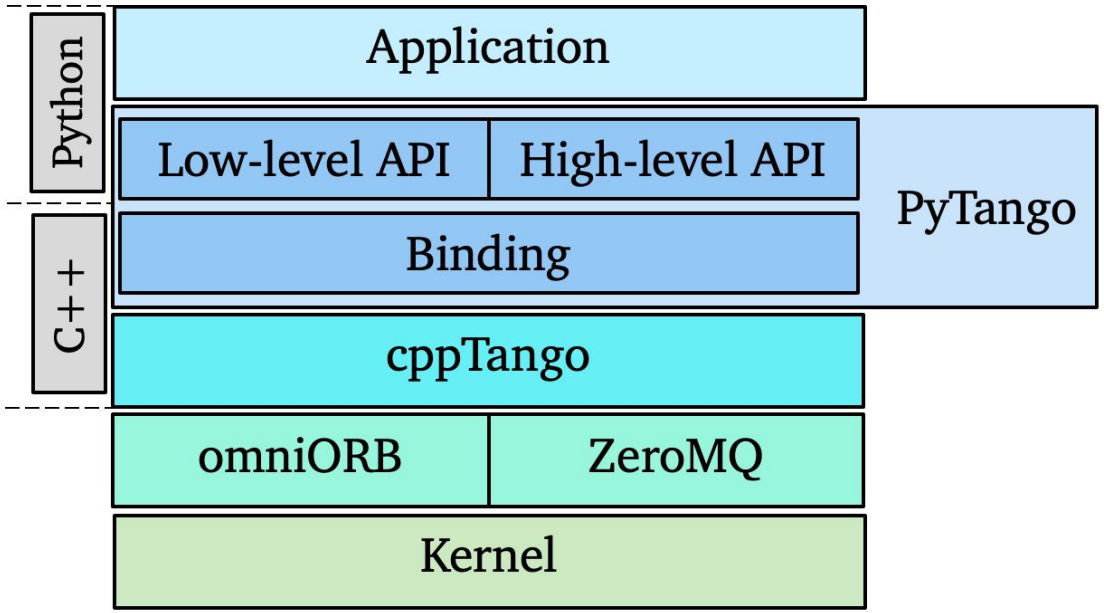
... using boost-python (future: pybind11)

Does not use omniorb Python library

Relies on numpy

Multi OS: Linux, Windows, MacOS (sort-of)

Works on Python 2.7, 3.5+



What is PyTango?

Plus some extras:

- Pythonic API
- asyncio and gevent event loop
- ITango (moved to a separate project)
- Experimental TANGO Database server (sqlite backend)
- DeviceTestContext for unit testing

Dependencies

OS dependencies:

- libtango >= 9.3, and its dependencies: omniORB4 and libzmq
- Boost.Python >= 1.33

Python dependencies:

- numpy >= 1.1
- six >= 1.10

Build dependencies:

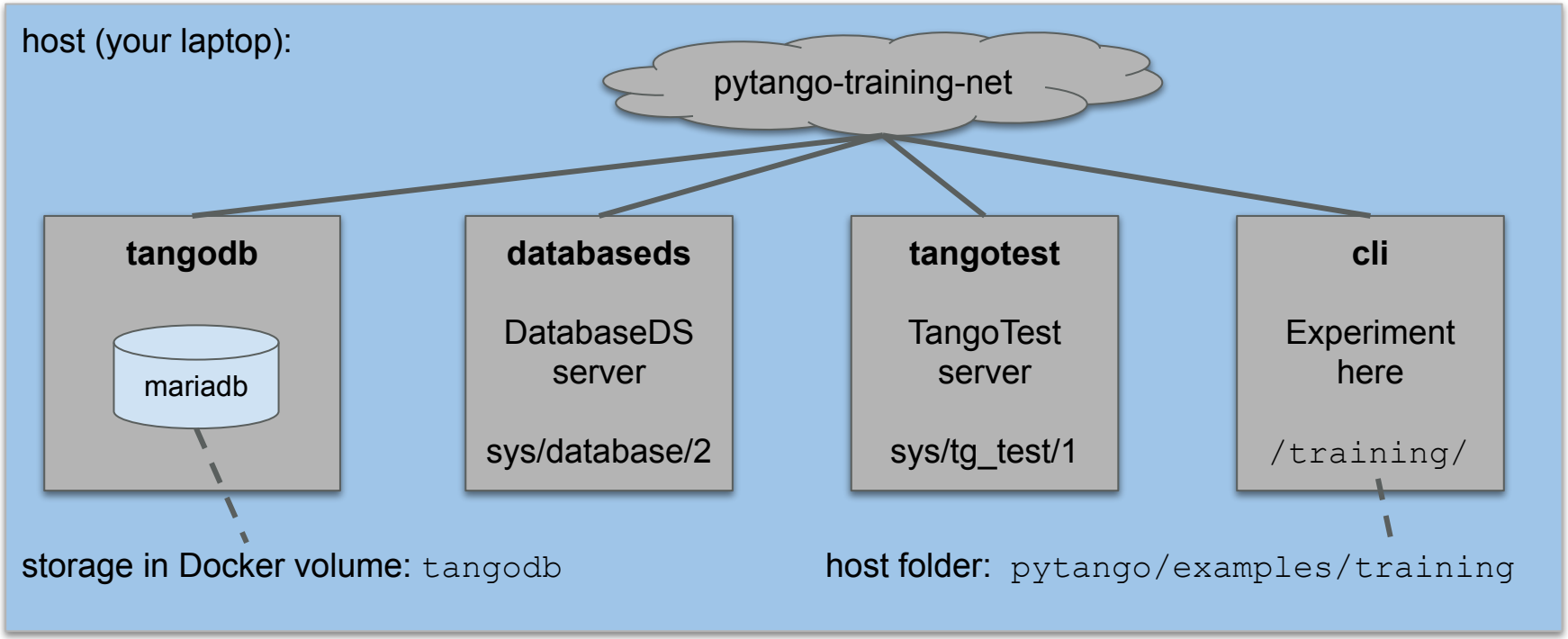
- Setuptools
- Sphinx

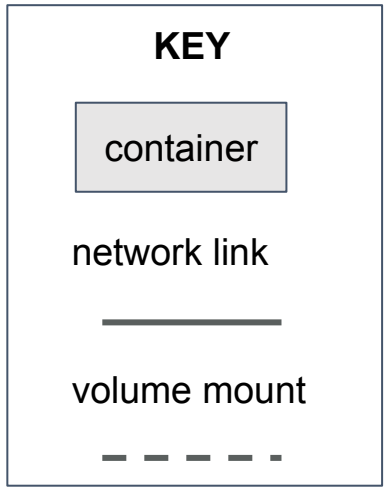
Optional dependencies:

- futures
- gevent

Docker compose environment

Docker compose setup





Repo URL: https://gitlab.com/tango-controls/pytango/-/tree/develop/examples/training

Start the Docker compose services

New Docker network required (once off):

```
➜  training git:(develop) ✗ docker network create pytango-training-net
3a55881054809b74546982482dcca9f90aecf4271f3abbf58fb43b8f7bca2311
```

Start services:

```
➜  training git:(develop) ✗ docker-compose up
Starting tangodb ... done
Starting databaseds ... done
Starting ipython    ... done
Starting tangotest  ... done
Attaching to tangodb, databaseds, tangotest, ipython
databaseds     | wait-for-it.sh: waiting 30 seconds for tangodb:3306
tangodb        | 2021-06-30 11:18:58+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.5.10+maria~focal star
ted.
databaseds     | wait-for-it.sh: tangodb:3306 is available after 0 seconds
tangodb        | 2021-06-30 11:18:58+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
databaseds     | main(): arrived
tangodb        | 2021-06-30 11:18:58+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.5.10+maria~focal star
ted.
tangodb        | Warning: World-writable config file '/etc/mysql/conf.d/sql_mode.cnf' is ignored
databaseds     | main(): export DataBase as named servant (name=database)
tangodb        | 2021-06-30 11:18:58 0 [Note] mysqld (mysqld 10.5.10-MariaDB-1:10.5.10+maria~focal) starting as process 1 ...
databaseds     | Ready to accept request
```

Run ipython session in container

```
[➜  training git:(add-training-examples) ✗ docker-compose exec cli ipython3
impPython 3.7.3 (default, Jan 22 2021, 20:04:44)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.21.0 -- An enhanced Interactive Python. Type '?' for help.

[In [1]: import tango

[In [2]: print(tango.utils.info())
PyTango 9.3.3 (9, 3, 3)
PyTango compiled with:
    Python : 3.7.3
    Numpy  : 1.19.2
    Tango  : 9.3.4
    Boost  : 1.67.0

PyTango runtime is:
    Python : 3.7.3
    Numpy  : 1.19.2
    Tango  : 9.3.4

PyTango running on:
uname_result(system='Linux', node='7ee8862308bd', release='4.19.121-linuxkit', version='#1 SMP Thu Jan 21 15:36:34 UTC 2021',
machine='x86_64', processor='')


[In [3]: import PyTango

[In [4]: PyTango is tango
Out[4]: True
```

Simple Tango device servers

Trivial *PowerSupply* device

A device server with a single device:

```python
#!/usr/bin/env python3

"""
Trivial power supply device with no external connection or behaviour.
"""

from time import sleep
from tango.server import Device, attribute, command


class PowerSupply(Device):

    @attribute(dtype=float)
    def voltage(self):
        return 1.5

    @command
    def calibrate(self):
        sleep(0.1)


if __name__ == '__main__':
    PowerSupply.run_server()
```

File: training/server/ps0a.py

Try to run it

```
[→  training git:(add-training-examples) ✗ docker-compose exec cli bash
[tango@7ee8862308bd:/training$ cd server/
[tango@7ee8862308bd:/training/server$ ./ps0a.py --help
 usage :  PowerSupply instance_name [-v[trace level]] [-file=<file_name> | -nodb [-dlist <device name list>] ]
[tango@7ee8862308bd:/training/server$ ./ps0a.py test
 The device server PowerSupply/test is not defined in database. Exiting!
 tango@7ee8862308bd:/training/server$
```

```
[tango@7ee8862308bd:/training/server$ tango_admin --help

Usage:
 --help                  Prints this help
 --ping-database         [max_time (s)] Ping database
 --check-device <dev>    Check if the device is defined in DB
 --add-server <exec/inst> <class> <dev list (comma separated)>   Add a server in DB
 --delete-server <exec/inst> [--with-properties]   Delete a server from DB
 --check-server <exec/inst>   Check if a device server is defined in DB
 --server-list  Display list of server names
 --server-instance-list <exec>   Display list of server instances for the given server name
 --add-property <dev> <prop_name> <prop_value (comma separated for array)>    Add a device property in DB
 --delete-property <dev> <prop_name>   Delete a device property from DB
 --tac-enabled Check if the TAC (Tango Access Control) is enabled
 --ping-device <dev> [max_time (s)] Check if the device is running
 --ping-network [max_time (s)] [-v] Ping network
```

Register (once-off) and run it

```
tango@7ee8862308bd:/training/server$ tango_admin --add-server PowerSupply/test PowerSupply train/ps/1
tango@7ee8862308bd:/training/server$ ./ps0a.py test
Ready to accept request
```

Start another shell and connect to the device as client:

```
→  training git:(add-training-examples) ✗ docker-compose exec cli ipython3
Python 3.7.3 (default, Jan 22 2021, 20:04:44)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.21.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import tango

In [2]: dp = tango.DeviceProxy("train/ps/1")

In [3]: dp.ping()
Out[3]: 936

In [4]: dp.voltage
Out[4]: 1.5

In [5]: dp.calibrate()

In [6]:
```

Less trivial *PowerSupply* device

Device connects to external hardware via TCP.

Need to install `gevent` in container to run simulator:

```
docker-compose exec cli pip install gevent
```

Configuration via properties (can be overridden in the Tango Database)

```python
from time import sleep
from socket import create_connection

from tango.server import Device, attribute, command, device_property


def connect(host, port):...


def write_readline(conn, msg):...


class PowerSupply(Device):

    host = device_property(str, default_value='localhost')
    port = device_property(int, default_value=45000)

    def init_device(self):
        super().init_device()
        self.conn = connect(self.host, self.port)

    @attribute(dtype=float)
    def voltage(self):
        return float(write_readline(self.conn, b'VOL?\n'))

    @command
    def calibrate(self):
        write_readline(self.conn, b'CALIB 1\n')
        while int(write_readline(self.conn, b'stat?\n')):
            sleep(0.1)
```

File: training/server/ps1.py

API: Low-level vs. High-level

Low-level server

```python
#!/usr/bin/env python3

import sys
import tango


class Motor(tango.Device_5Impl):
    def __init__(self, cl, name):
        tango.Device_5Impl.__init__(self, cl, name)
        Motor.init_device(self)

    def delete_device(self):
        pass

    def init_device(self):
        self.get_device_properties(self.get_device_class())
        self.attr_position_read = 1.0

    def always_executed_hook(self):
        pass

    def read_position(self, attr):
        attr.set_value(self.attr_position_read)

    def read_attr_hardware(self, data):
        pass
```

```python
class MotorClass(tango.DeviceClass):

    class_property_list = {}
    device_property_list = {}
    cmd_list = {}
    attr_list = {"position": [[tango.DevDouble, tango.SCALAR, tango.READ]]}

    def __init__(self, name):
        tango.DeviceClass.__init__(self, name)
        self.set_type(name)


def main():
    try:
        py = tango.Util(sys.argv)
        py.add_class(MotorClass, Motor, "Motor")
        U = tango.Util.instance()
        U.server_init()
        U.server_run()
    except tango.DevFailed as exc:
        print(f"-------> Received a DevFailed exception: {exc}")
    except Exception as exc:
        print(f"-------> An unforeseen exception occurred: {exc}")


if __name__ == "__main__":
    main()
```

High-level server

```python
#!/usr/bin/env python3

from tango.server import Device, attribute


class Motor(Device):

    def init_device(self):
        super().init_device()
        self.attr_position_read = 1.0

    @attribute(dtype=float)
    def position(self):
        return self.attr_position_read


def main():
    Motor.run_server()


if __name__ == "__main__":
    main()
```

```python
    @attribute(dtype=float)
    def position(self):
        return self.attr_position_read, time.time(), tango.AttrQuality.ATTR_CHANGING
```

Returning a 3-tuple allows the reading's time and quality to be modified.
If omitted, default is current time, and `ATTR_VALID`

Low-level client

```
[In [36]: reading = dp.read_attribute("voltage")

[In [37]: reading.value
 Out[37]: 1.5293084051287036

[In [38]: reading
 Out[38]: DeviceAttribute(data_format = tango._tango.AttrDataFormat.SCALAR, dim_x = 1, dim_y = 0, has_failed = False, is_empty
 = False, name = 'voltage', nb_read = 1, nb_written = 0, quality = tango._tango.AttrQuality.ATTR_VALID, r_dimension = Attribute
 Dimension(dim_x = 1, dim_y = 0), time = TimeVal(tv_nsec = 0, tv_sec = 1625170117, tv_usec = 137365), type = tango._tango.CmdAr
 gType.DevDouble, value = 1.5293084051287036, w_dim_x = 0, w_dim_y = 0, w_dimension = AttributeDimension(dim_x = 0, dim_y = 0),
  w_value = None)

[In [39]: dp.command_inout("calibrate")
```

High-level client

Simpler attribute reading, but cannot access details like time and quality.

```
In [40]: dp.voltage
Out[40]: 1.4703845668655005

In [41]: dp.calibrate()
```

Server comparison

High-level or "HL" API:

- Much less overhead on writing device servers and clients.
- Python properties and decorators allow a much more compact syntax.
- More readable code allows a better understanding of the behaviour.

Low-level or "classic" API:

- Dynamic Attributes are easier to implement, as "attr" argument allows attributes to be identified by name.
- Dynamic Commands implemented only on classic API.
- DeviceClass init allows to separate creation of variables that should be done only once from those done at each `Init()` call (but you can still write your own custom init if you want).

Client comparison

High-level or "HL" API:

- More readable code.
- Can accidentally use incorrect name for attribute writing.
- Cannot access attribute quality or timestamp when reading.

Low-level or "classic" API:

- Less readable.
- Allows fine-grain access, e.g., for attribute reading.
- Many functions only available this way, e.g., `command_inout_asynch`.

It is fine to mix the two APIs for client access.

ITango for easy client access

Connect to device

```
[→  training git:(add-training-examples) ✗ docker-compose exec cli itango3
ITango 9.3.3 -- An interactive Tango client.

Running on top of Python 3.7.3, IPython 7.21 and PyTango 9.3.3

help       -> ITango's help system.
object?    -> Details about 'object'. ?object also works, ?? prints more.

IPython profile: tango

hint: Try typing: mydev = Device("<tab>

[In [1]: # Device is an alias for tango.DeviceProxy

[In [2]: dev = Device("sys/tg_test/1")

[In [3]: # or can use class name (limits <tab> search space)

[In [4]: dev = TangoTest("sys/tg_test/1")

[In [5]: dev.ping()
Out[5]: 563
```

Commands and attributes

```
[In [6]: # send a command (low-level way)

[In [7]: dev.command_inout('DevShort', 1234)
 Out[7]: 1234

[In [8]: # send a command (high-level way)

[In [9]: dev.DevShort(1235)
 Out[9]: 1235

[In [10]: # read an attribute

[In [11]: dev.long_spectrum
 Out[11]:
 array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)

[In [12]: # write to it

[In [13]: dev.long_spectrum = (1, 2, 3, 4)

[In [14]: dev.long_spectrum
 Out[14]: array([1, 2, 3, 4], dtype=int32)
```

Built-in event monitor: `mon` command

```
In [15]: dev.poll_attribute('State', 3000)

In [16]: mon -a sys/tg_test/1/State
'sys/tg_test/1/State' is now being monitored. Type 'mon' to see all events

In [17]: dev.SwitchStates()

In [18]: mon
ID   Device                         Attribute          Value               Quality      Time
---- ------------------------------ ------------------ ------------------- ------------ ----------------
   0 sys/tg_test/1                  state              RUNNING             ATTR_VALID   21:04:55.149842
   1 sys/tg_test/1                  state              RUNNING             ATTR_VALID   21:04:58.149351
   2 sys/tg_test/1                  state              FAULT               ATTR_VALID   21:05:04.151308

In [19]: dev.SwitchStates()

In [20]: mon
ID   Device                         Attribute          Value               Quality      Time
---- ------------------------------ ------------------ ------------------- ------------ ----------------
   0 sys/tg_test/1                  state              RUNNING             ATTR_VALID   21:04:55.149842
   1 sys/tg_test/1                  state              RUNNING             ATTR_VALID   21:04:58.149351
   2 sys/tg_test/1                  state              FAULT               ATTR_VALID   21:05:04.151308
   3 sys/tg_test/1                  state              RUNNING             ATTR_VALID   21:05:16.148329

In [21]: mon -d sys/tg_test/1/State
Stopped monitoring 'sys/tg_test/1/State'
```

Run `mon?` for more details

End of ITango demo

More info: https://itango.readthedocs.io

It can also be used from a Jupyter notebook

jupyTango - Nicolas Leclercq

Build docker image

```
[➜  tango-src git clone git@gitlab.com:tango-controls/jupyTango.git
Cloning into 'jupyTango'...
remote: Enumerating objects: 212, done.
remote: Counting objects: 100% (125/125), done.
remote: Compressing objects: 100% (70/70), done.
remote: Total 212 (delta 66), reused 113 (delta 55), pack-reused 87
Receiving objects: 100% (212/212), 4.89 MiB | 949.00 KiB/s, done.
Resolving deltas: 100% (105/105), done.
[➜  tango-src cd jupyTango/docker
[➜  docker git:(develop) docker build -t jupytango:1.0.0 .
[+] Building 264.5s (15/15) FINISHED
 => [internal] load build definition from Dockerfile                                                                0.0s
 => => transferring dockerfile: 1.78kB                                                                              0.0s
 => [internal] load .dockerignore                                                                                   0.0s
 => => transferring context: 2B                                                                                     0.0s
 => [internal] load metadata for artefact.skao.int/ska-tango-images-tango-itango:9.3.4                              0.0s
 => [ 1/11] FROM artefact.skao.int/ska-tango-images-tango-itango:9.3.4                                              0.2s
 => [ 2/11] RUN DEBIAN_FRONTEND=noninteractive apt-get update && apt-get install -y procps git                     31.7s
 => [ 3/11] RUN python3 -m pip install opencv-python jupyterlab ipywidgets jupyter_bokeh                          218.5s
 => [ 4/11] RUN cp -Rf $HOME/.ipython/profile_default $HOME/.ipython/profile_jupytango                              0.3s
 => [ 5/11] RUN echo "config = get_config()" > $HOME/.ipython/profile_jupytango/ipython_config.py                   0.3s
 => [ 6/11] RUN echo "config.InteractiveShellApp.extensions = ['jupytango']" >> $HOME/.ipython/profile_jupytango/  0.4s
 => [ 7/11] RUN python -m ipykernel install --user --name jupyTango --display-name "jupyTango" --profile jupytango  1.0s
 => [ 8/11] RUN git clone -b master https://gitlab.com/tango-controls/jupyTango.git $HOME/jupyTango                 7.3s
 => [ 9/11] RUN cp $HOME/jupyTango/resources/logo/* $HOME/.local/share/jupyter/kernels/jupytango                    0.4s
 => [10/11] RUN export PYTHONPATH=$HOME/jupyTango                                                                   0.3s
 => [11/11] RUN export JUPYTER_CONTEXT=LAB                                                                          0.3s
 => exporting to image                                                                                              3.6s
 => => exporting layers                                                                                             3.5s
 => => writing image sha256:bb3e658008440fa9b2809129ee1d0d7e6896dfa923e9cd824c19f059bfb2ce5e                        0.0s
 => => naming to docker.io/library/jupytango:1.0.0                                                                  0.0s
```

Readme: https://gitlab.com/tango-controls/jupyTango/-/tree/develop/#giving-jupytango-a-try-using-docker

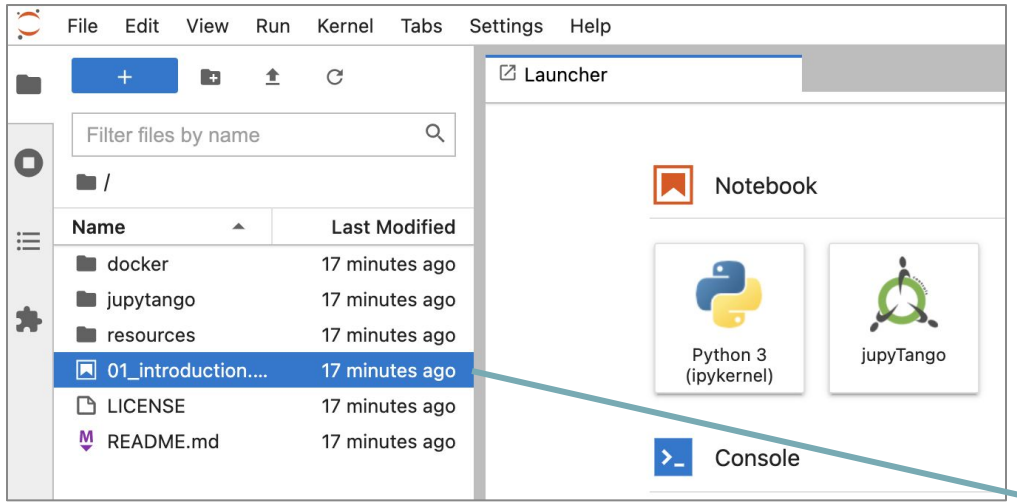
Run docker-compose

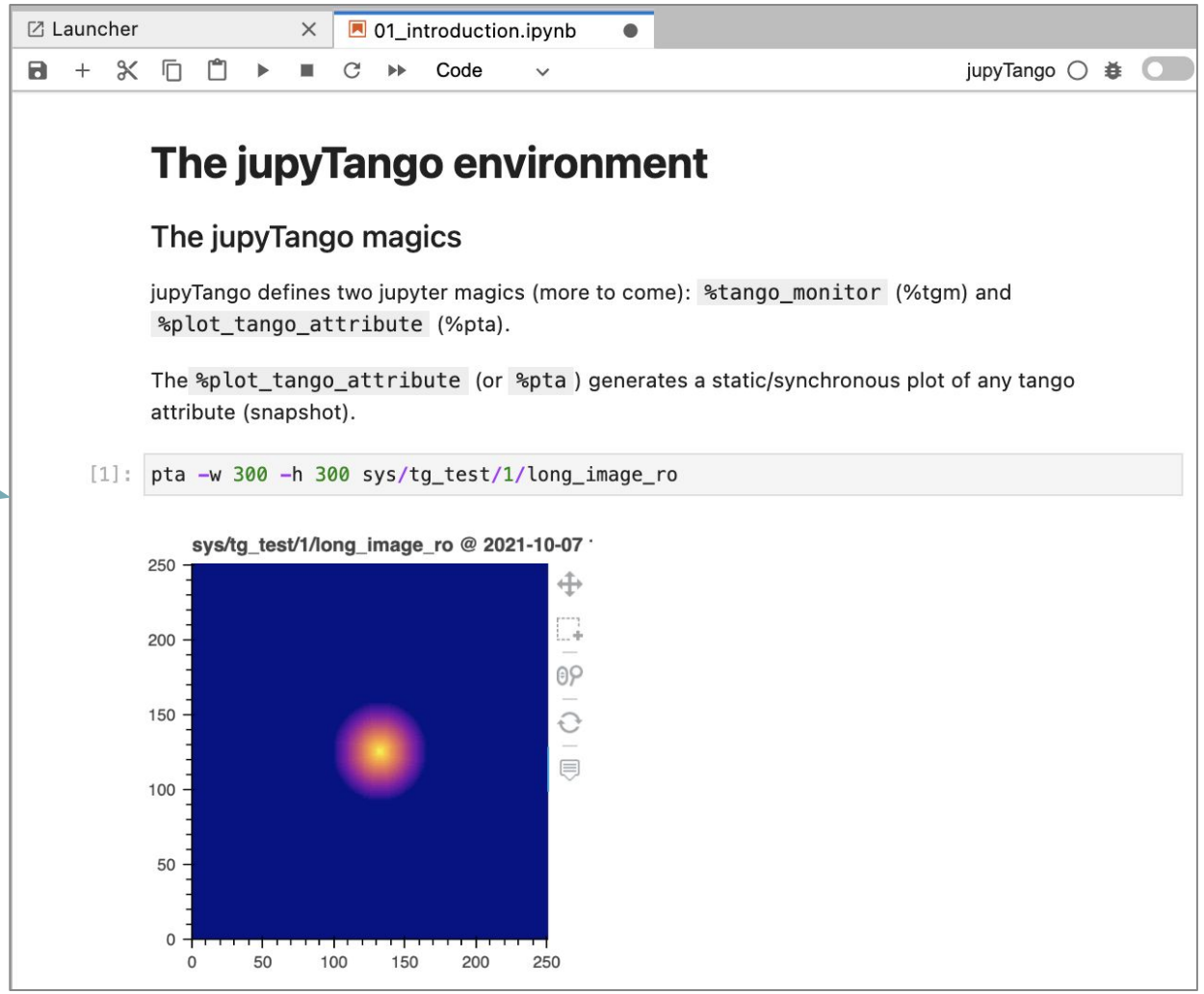
```
[→  docker git:(develop) docker network create jupytango-net
e9690e04aac610379c75c8200766df4dd8f285e31deb5715529224728d7835e2
[→  docker git:(develop) docker-compose up
Starting tangodb ... done
Creating databaseds ... done
Creating tangotest  ... done
Creating jupytango  ... done
Attaching to tangodb, databaseds, tangotest, jupytango
databaseds  | wait-for-it.sh: waiting 30 seconds for tangodb:3306
tangodb     | 2021-10-07 10:00:30+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.5.11+maria~focal started.
tangotest   | Can't build connection to TANGO database server, exiting
tangodb     | 2021-10-07 10:00:30+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
tangodb     | 2021-10-07 10:00:30+00:00 [Note] [Entrypoint]: Entrypoint script for MariaDB Server 1:10.5.11+maria~focal started.
tangodb     | 2021-10-07 10:00:30+00:00 [Note] [Entrypoint]: Initializing database files
```

```
jupytango   | [I 2021-10-07 10:00:32.741 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
jupytango   | [W 2021-10-07 10:00:32.745 ServerApp] No web browser found: could not locate runnable browser.
jupytango   | [C 2021-10-07 10:00:32.746 ServerApp]
jupytango   |
jupytango   |     To access the server, open this file in a browser:
jupytango   |         file:///home/tango/.local/share/jupyter/runtime/jpserver-1-open.html
jupytango   |     Or copy and paste one of these URLs:
jupytango   |         http://046c6790decb:8888/lab?token=afd379cd2d2a3a1d48b6e7010940a96d4c57d43b84f696d1
jupytango   |     or http://127.0.0.1:8888/lab?token=afd379cd2d2a3a1d48b6e7010940a96d4c57d43b84f696d1
```

Open URL with token in your browser

Connect to Jupyter notebook





NOTE:
Dynamic updates to plots (`tgm` and `tango_monitor`) don't work with docker-compose under MacOS!

Events and polling

Client polling vs. server polling vs. pushing events

A completely lazy implementation will consist of letting the client to ask periodically for the values it wants.

- Advantages, nothing is done if nobody wants it
- Disadvantage, many clients will easily saturate the device

Using Tango server polling, clients only access cached values or receive events

- Advantages, most efficient, you protect the device from external clients
- Disadvantages, the device is running continuously, any memory leak will be magnified, the device becomes sensitive to serialisation issues (if not giving time to process all pollings)

Pushing events manually from the device code

- Typically updating attribute values from a single command execution, `Update()`, or a background thread.
- It can be combined with the previous approach
- Not as easily configurable from Jive
- Even if using a background thread, it does not escape serialization completely (event pushing is blocking)

Try to subscribe to an attribute

```
[In [1]: import tango

[In [2]: dp = tango.DeviceProxy("train/ps/1")

[In [3]: cb = tango.utils.EventCallback()

[In [4]: eid = dp.subscribe_event("voltage", tango.EventType.CHANGE_EVENT, cb)
---------------------------------------------------------------------------
DevFailed                                 Traceback (most recent call last)
<ipython-input-4-6e3a6b1eba5e> in <module>
----> 1 eid = dp.subscribe_event("voltage", tango.EventType.CHANGE_EVENT, cb)
```

tango.utils.EventCallback docs

```
   1296     with self.__get_event_map_lock():

DevFailed: DevFailed[
DevError[
    desc = The polling (necessary to send events) for the attribute voltage is not started
  origin = DServer::event_subscription
  reason = API_AttributePollingNotStarted
severity = ERR]

DevError[
    desc = Failed to execute command_inout on device dserver/powersupply/test, command ZmqEventSubscriptionChange
  origin = Connection::command_inout()
  reason = API_CommandFailed
severity = ERR]

DevError[
    desc = Device server send exception while trying to register event
  origin = EventConsumer::connect_event()
  reason = API_DSFailedRegisteringEvent
severity = ERR]
]
```

Add polling

Could enable polling from client side, or in device server code.

Example of server code:

```
10    import ...
13
14
15    class PowerSupply(Device):
16
17        @attribute(
18            dtype=float,
19            polling_period=3000,  # milliseconds
20            rel_change=1e-3)
21        def voltage(self):
22            noise = -0.05 + 0.1 * random.random()
23            return 1.5 + noise
24
25        @command
26        def calibrate(self):
27            sleep(0.1)
28
29
30    if __name__ == '__main__':
31        PowerSupply.run_server()
```

File: [training/server/ps0b.py]

Subscribe to an attribute

Kill the **old** device and start the **new** device: `ps0b.py`

```
tango@7ee8862308bd:/training/server$ ./ps0a.py test
Ready to accept request
^Ctango@7ee8862308bd:/training/server$ ./ps0b.py test
Ready to accept request
```

From the ipython client session:

```
In [7]: eid = dp.subscribe_event("voltage", tango.EventType.CHANGE_EVENT, cb)
2021-07-01 20:45:57.439197 TRAIN/PS/1 VOLTAGE CHANGE [ATTR_VALID] 1.5406642504211145

In [8]: 2021-07-01 20:46:00.439279 TRAIN/PS/1 VOLTAGE CHANGE [ATTR_VALID] 1.5001033159371053
2021-07-01 20:46:03.440569 TRAIN/PS/1 VOLTAGE CHANGE [ATTR_VALID] 1.5087277294230272
2021-07-01 20:46:06.443265 TRAIN/PS/1 VOLTAGE CHANGE [ATTR_VALID] 1.4603678396472277
In [8]:

In [8]: dp.unsubscribe_event(eid)
```

Immediate callback, using a synchronous attribute read

Other events via ZeroMQ, every 3 seconds, in this case

Custom event callback function

```
In [18]: def callback(evt):
    ...:     if evt.err:
    ...:         print("Bad event!")
    ...:     else:
    ...:         print(f"Event:  {evt.attr_value}")
    ...:

In [19]: eid = dp.subscribe_event("voltage", tango.EventType.CHANGE_EVENT, callback)
Event:  DeviceAttribute[
data_format = tango._tango.AttrDataFormat.SCALAR
      dim_x = 1
      dim_y = 0
 has_failed = False
   is_empty = False
       name = 'voltage'
    nb_read = 1
 nb_written = 0
    quality = tango._tango.AttrQuality.ATTR_VALID
r_dimension = AttributeDimension(dim_x = 1, dim_y = 0)
       time = TimeVal(tv_nsec = 0, tv_sec = 1625151148, tv_usec = 923497)
       type = tango._tango.CmdArgType.DevDouble
      value = 1.457084791428074
    w_dim_x = 0
    w_dim_y = 0
w_dimension = AttributeDimension(dim_x = 0, dim_y = 0)
    w_value = None]
```

If we kill the device server:

```
In [20]:

Bad event!
Bad event!
Bad event!
Bad event!
```

Custom event callback class

```
In [27]: class MyCallback:
    ...:     def push_event(self, evt):
    ...:         if evt.err:
    ...:             print(f"bad event! {evt.errors[0]}")
    ...:         else:
    ...:             print(f"good event: {evt.attr_value}")
    ...:

In [28]: eid = dp.subscribe_event("voltage", tango.EventType.CHANGE_EVENT, MyCallback())
good event: DeviceAttribute[
data_format = tango._tango.AttrDataFormat.SCALAR
      dim_x = 1
      dim_y = 0
 has_failed = False
   is_empty = False
       name = 'voltage'
    nb_read = 1
 nb_written = 0
    quality = tango._tango.AttrQuality.ATTR_VALID
r_dimension = AttributeDimension(dim_x = 1, dim_y = 0)
       time = TimeVal(tv_nsec = 0, tv_sec = 1625167673, tv_usec = 138439)
       type = tango._tango.CmdArgType.DevDouble
      value = 1.5379521553502165
    w_dim_x = 0
    w_dim_y = 0
w_dimension = AttributeDimension(dim_x = 0, dim_y = 0)
    w_value = None]
```

If we kill the device server:

```
bad event! DevError[
    desc = Event channel is not responding anymore, maybe the server or event system is down
  origin = EventConsumer::KeepAliveThread()
  reason = API_EventTimeout
severity = ERR]

bad event! DevError[
    desc = Event channel is not responding anymore, maybe the server or event system is down
  origin = EventConsumer::KeepAliveThread()
  reason = API_EventTimeout
severity = ERR]
```

Pushing events manually

In `init_device`:

implemented: True: pushing is enabled

detect: False: Tango ignores change-event criteria

```
self.set_change_event("voltage", True, False)
self.set_archive_event("voltage", True, False)
```

In your methods:

```
self.push_change_event("voltage", value)   # can include [timestamp, quality]
self.push_archive_event("voltage", value)   # can include [timestamp, quality]
```

Can also push a `tango.DevFailed` exception object

Miscellaneous

Other generic clients

Single attribute only:

tango.AttributeProxy

```
In [1]: import tango

In [2]: voltage_attr = tango.AttributeProxy("train/ps/1/voltage")

In [3]: voltage_attr.name()
Out[3]: 'voltage'

In [4]: voltage_attr.read().value
Out[4]: 1.5
```

Many devices simultaneously:

tango.Group

can build complex hierarchy (nested groups)

can read/write attributes

can send commands with same or different params

```
In [5]: gr = tango.Group("sys")

In [6]: gr.add("sys/*")

In [7]: gr.get_device_list()
Out[7]: ['sys/database/2', 'sys/tg_test/1']

In [8]: replies = gr.command_inout("State")

In [9]: replies
Out[9]: [GroupCmdReply(), GroupCmdReply()]

In [10]: replies[0].get_data()
Out[10]: tango._tango.DevState.ON

In [11]: replies[1].get_data()
Out[11]: tango._tango.DevState.RUNNING
```

Note: These clients don’t support the high-level API

Database client

"Friendly" client to the DatabaseDS: [tango.Database]

```
In [22]: db = tango.Database()

In [23]: dev_info = db.get_device_info("train/ps/1")
    ...: print(f"name: {dev_info.name}")
    ...: print(f"class_name: {dev_info.class_name}")
    ...: print(f"ds_full_name: {dev_info.ds_full_name}")
    ...: print(f"exported: {dev_info.exported}")
    ...: print(f"ior: {dev_info.ior}")
    ...: print(f"version: {dev_info.version}")
    ...: print(f"pid: {dev_info.pid}")
    ...: print(f"started_date: {dev_info.started_date}")
    ...: print(f"stopped_date: {dev_info.stopped_date}")
name: train/ps/1
class_name: PowerSupply
ds_full_name: PowerSupply/test
exported: 1
ior: IOR:010000001700000049444c3a54616e676f2f4465766963655f353a312e3000000100000000000000ad00000001010200c0000003139322e31363
82e302e350047c400000e000000fe63d4e160000000cf0000000000000030000000000000080000001000000005454410100000001c00000010000000100
0010001000000010001050901010001000000090101000254544141000000010000000d0000003534323730386636313836390000000025000002f746d702
f6f6d6e692d74616e676f2f303030303030303230372d313632353431323730370000
version: 5
pid: 207
started_date: 4th July 2021 at 15:31:47
stopped_date: 4th July 2021 at 15:30:28
```

Default is TANGO_HOST env var, but can provide `(host, port)`, or even a file name.

Enumerated types - device

```python
#!/usr/bin/env python3

"""Trivial power supply device with no external connection or behaviour...."""

import ...


class TrackingMode(enum.IntEnum):
    INDEPENDENT = 0  # must start at zero!
    SYNCED = 1  # and increment by 1


class PowerSupply(Device):

    _tracking_mode = TrackingMode.SYNCED

    @attribute(dtype=TrackingMode, access=AttrWriteType.READ_WRITE)
    def output_tracking(self):
        return self._tracking_mode

    @output_tracking.write
    def output_tracking(self, value):
        self._tracking_mode = value
```

Use enum class directly, rather than `dtype="DevEnum", enum_labels=["INDEPENDENT", "SYNCED"]`

File: training/server/ps0c.py

Enumerated types - client

```
In [1]: import tango

In [2]: dp = tango.DeviceProxy("train/ps/1")

In [3]: tracking = dp.output_tracking

In [4]: tracking
Out[4]: <output_tracking.SYNCED: 1>

In [5]: type(tracking)
Out[5]: <enum 'output_tracking'>

In [6]: tracking == tracking.INDEPENDENT
Out[6]: False

In [7]: tracking.name
Out[7]: 'SYNCED'

In [8]: tracking.value
Out[8]: 1

In [9]: tracking.__class__.__members__
Out[9]:
mappingproxy({'INDEPENDENT': <output_tracking.INDEPENDENT: 0>,
              'SYNCED': <output_tracking.SYNCED: 1>})
```

```
In [10]: dp.output_tracking = tracking.INDEPENDENT

In [11]: dp.output_tracking
Out[11]: <output_tracking.INDEPENDENT: 0>

In [12]: dp.output_tracking = 1

In [13]: dp.output_tracking
Out[13]: <output_tracking.SYNCED: 1>

In [14]: dp.output_tracking = "INDEPENDENT"

In [15]: dp.output_tracking
Out[15]: <output_tracking.INDEPENDENT: 0>

In [16]: dp.output_tracking = "INVALID"
---------------------------------------------------------
KeyError                                  Traceback (
/usr/local/lib/python3.7/dist-packages/tango/device_p
    296             try:
--> 297                 value = enum_class[value]
    298             except KeyError:

/usr/lib/pyt
    351
--> 352
    353

KeyError: 'I
```

```
/usr/local/lib/python3.7/dist-packages/tango/device_proxy.py in __set_att
ribute_value(self, name, value)
    299                 raise AttributeError(
    300                     'Invalid enum value %s for attribute %s. Vali
d ones: %s' %
--> 301                     (value, name, [m for m in enum_class.__member
s__.keys()]))
    302     return self.write_attribute(name, value)
    303

AttributeError: Invalid enum value INVALID for attribute output_tracking.
 Valid ones: ['INDEPENDENT', 'SYNCED']
```

Logging decorators

```
14  from tango import AttrWriteType, DebugIt, InfoIt
```

```
23  class PowerSupply(Device):
24
25      _tracking_mode = TrackingMode.SYNCED
26
27      @attribute(dtype=TrackingMode, access=AttrWriteType.READ_WRITE)
28      def output_tracking(self):...
30
31      @output_tracking.write
32      def output_tracking(self, value):...
34
35      @attribute(
36          dtype=float,
37          polling_period=3000,  # milliseconds
38          rel_change=1e-3)
39      @DebugIt(show_args=False, show_kwargs=False, show_ret=True)
40      def voltage(self):
41          noise = -0.05 + 0.1 * random.random()
42          return 1.5 + noise
43
44      @command
45      @InfoIt()
46      def calibrate(self):
47          sleep(0.1)
```

Log entry and exit at debug level

Log entry and exit at info level

Others: `@WarnIt()`, `@ErrorIt()`, `@FatalIt()`, `@LogIt()`

https://pytango.readthedocs.io/en/latest/server_api/logging.html

Logging decorators

```
tango@542708f61869:/training/server$ ./ps0c.py --help
usage :  PowerSupply instance_name [-v[trace level]] [-file=<file_name> | -nodb [-dlist <device name list>] ]
tango@542708f61869:/training/server$
tango@542708f61869:/training/server$ ./ps0c.py test -v1
Ready to accept request
1625403441 [140266484774656] INFO train/ps/1 -> PowerSupply.calibrate()
1625403441 [140266484774656] INFO train/ps/1 <- PowerSupply.calibrate()

^Ctango@542708f61869:/training/server$ ./ps0c.py test -v3
Ready to accept request
1625403456 [140509251106560] DEBUG train/ps/1 -> PowerSupply.voltage()
1625403456 [140509251106560] DEBUG train/ps/1 1.4678693253449642 <- PowerSupply.voltage()
1625403459 [140509251106560] DEBUG train/ps/1 -> PowerSupply.voltage()
1625403459 [140509251106560] DEBUG train/ps/1 1.508858102070189 <- PowerSupply.voltage()
1625403461 [140509242713856] INFO train/ps/1 -> PowerSupply.calibrate()
1625403462 [140509242713856] INFO train/ps/1 <- PowerSupply.calibrate()
```

Show **info** logs on console

Show **debug** logs on console

```
tango@542708f61869:/training/server$ ./ps0c.py test -v5
Entering Logging::init
        TANGO_LOG_PATH is /tmp/tango-tango
        cmd line logging level is 5
        Logging::create_log_dir(/tmp/tango-tango/PowerSupply/test) returned -1
        added console target (logging level set from cmd line)
Leaving Logging::init
1625403547 [140483542640448] DEBUG dserver/PowerSupply/test Connected to database
1625403547 [140483542640448] DEBUG dserver/PowerSupply/test Entering Util::server_already_running method
1625403547 [140483542640448] DEBUG dserver/PowerSupply/test Leaving Util::server_already_running method
1625403547 [140483542640448] DEBUG dserver/PowerSupply/test calling Tango::NotifdEventSupplier::create()

1625403547 [140483542640448] DEBUG dserver/PowerSupply/test Failed to import EventChannelFactory notifd/factory/542708f61869 f
rom the Tango database
1625403547 [140483542640448] DEBUG dserver/PowerSupply/test calling Tango::ZmqEventSupplier::create()
1625403547 [140483542640448] DEBUG dserver/PowerSupply/test Heartbeat thread Id = 5
1625403547 [140483542640448] DEBUG dserver/PowerSupply/test Tango object singleton constructed
```

Show **all** logs, including all admin device CORBA calls, events, etc.

Green modes

Also called Asynchronous PyTango.

Checkout the docs: pytango.readthedocs.io/en/stable/green_modes/green.html

```
tango.GreenMode.Synchronous   # default
tango.GreenMode.Futures
tango.GreenMode.Gevent        # server serialisation disabled!
tango.GreenMode.Asyncio       # server serialisation disabled!
```

Serialisation model details:

https://tango-controls.readthedocs.io/en/latest/development/advanced/threading.html#serialization-model-within-a-device-server

Asyncio client example

```
[In [1]: from tango.asyncio import DeviceProxy as AsyncioProxy

[In [2]: device = await AsyncioProxy('sys/tg_test/1')

[In [3]: result = await device.read_attribute('double_scalar')

[In [4]: result.value
Out[4]: -66.25893320791678

[In [5]: await device.command_inout("DevDouble", 1.23)
Out[5]: 1.23
```

Low-level API

```
[In [6]: # HL API

[In [7]: await device.DevDouble(1.23)
Out[7]: 1.23

[In [8]: await device.double_scalar
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-8-c941e7940d71> in async-def-wrapper()

/usr/local/lib/python3.7/dist-packages/tango/device_proxy.py in __DeviceProxy__getattr(self, name)
    342     attr_info = self.__get_attr_cache().get(name_l)
    343     if attr_info:
--> 344         return __get_attribute_value(self, attr_info, name)
    345
    346     try:

/usr/local/lib/python3.7/dist-packages/tango/device_proxy.py in __get_attribute_value(self, attr_info, name)
    281 def __get_attribute_value(self, attr_info, name):
    282     _, enum_class = attr_info
--> 283     attr_value = self.read_attribute(name).value
    284     if enum_class:
    285         return enum_class(attr_value)

AttributeError: '_asyncio.Future' object has no attribute 'value'
```

High-level API
(fails for attributes - bug?)

Asyncio server example

```python
class AsyncioDevice(Device):
    green_mode = GreenMode.Asyncio

    async def init_device(self):
        await super().init_device()
        self.set_state(DevState.ON)

    @command
    async def long_running_command(self):
        self.set_state(DevState.OPEN)
        await asyncio.sleep(2)
        self.set_state(DevState.CLOSE)

    @command
    async def background_task_command(self):
        loop = asyncio.get_event_loop()
        future = loop.create_task(self.coroutine_target())

    async def coroutine_target(self):
        self.set_state(DevState.INSERT)
        await asyncio.sleep(15)
        self.set_state(DevState.EXTRACT)

    @attribute
    async def test_attribute(self):
        await asyncio.sleep(2)
        return 42
```

File:
asyncio_green_mode/asyncio_device_example

More details

General Asyncio overview

Slides: vxgmichel.github.io/asyncio-overview

Repo: github.com/vxgmichel/asyncio-overview

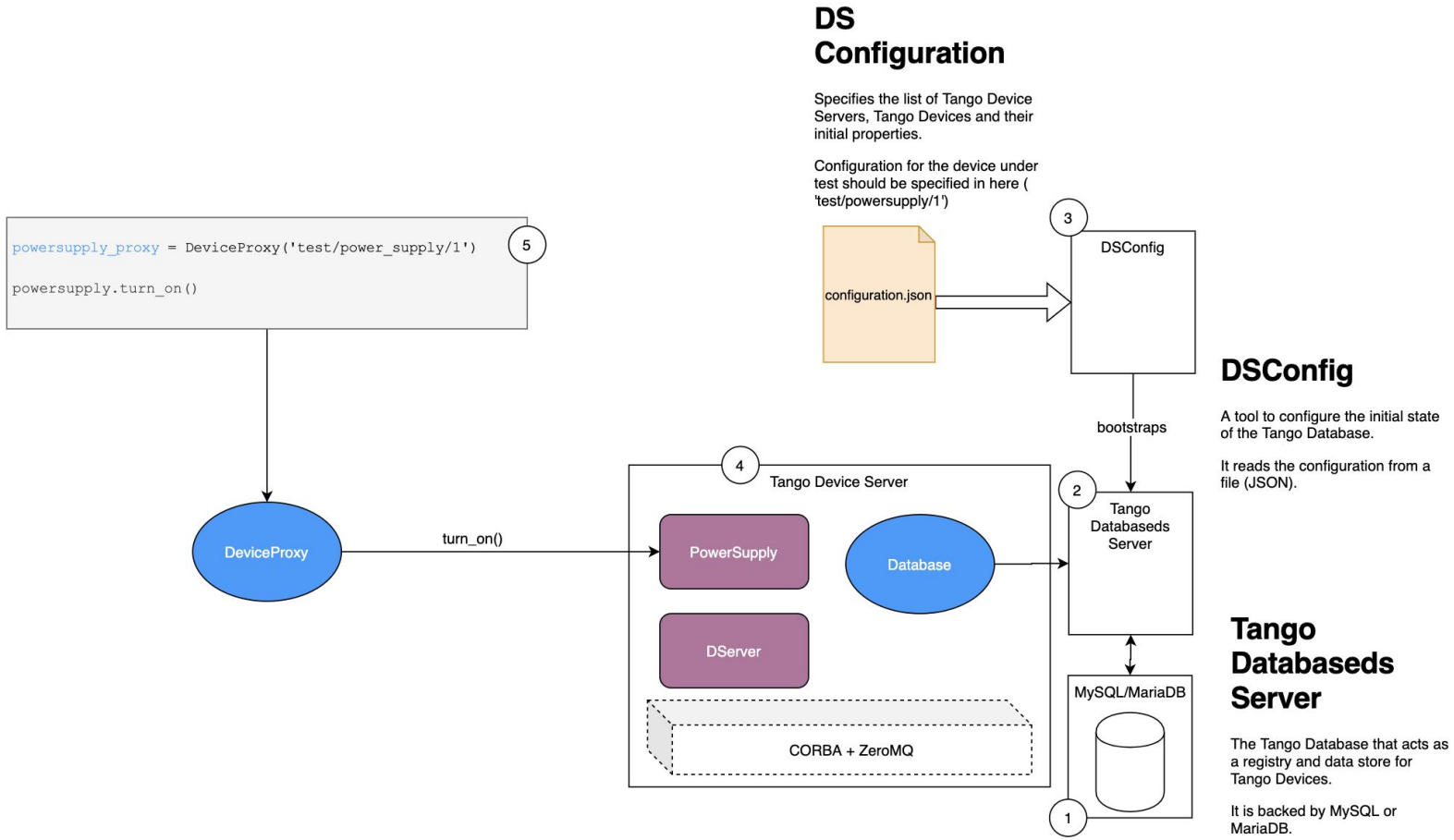
ICALEPCS 2017 PyTango workshop (notes on concurrency)

Slides: vxgmichel.github.io/icalepcs-workshop

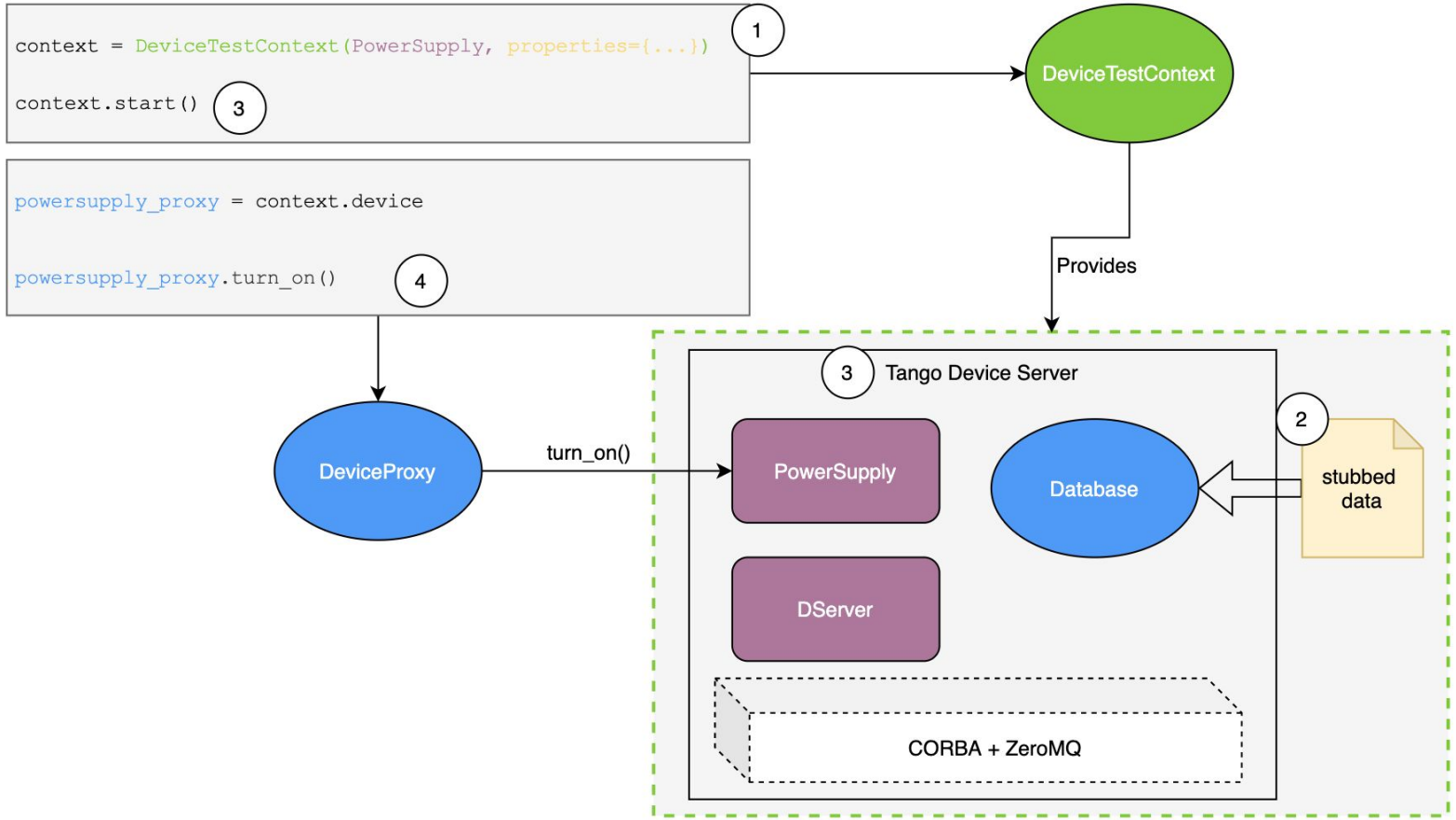
Repo: github.com/vxgmichel/icalepcs-workshop

How to test?

Testing with real Tango Facility



Testing with *DeviceTestContext*

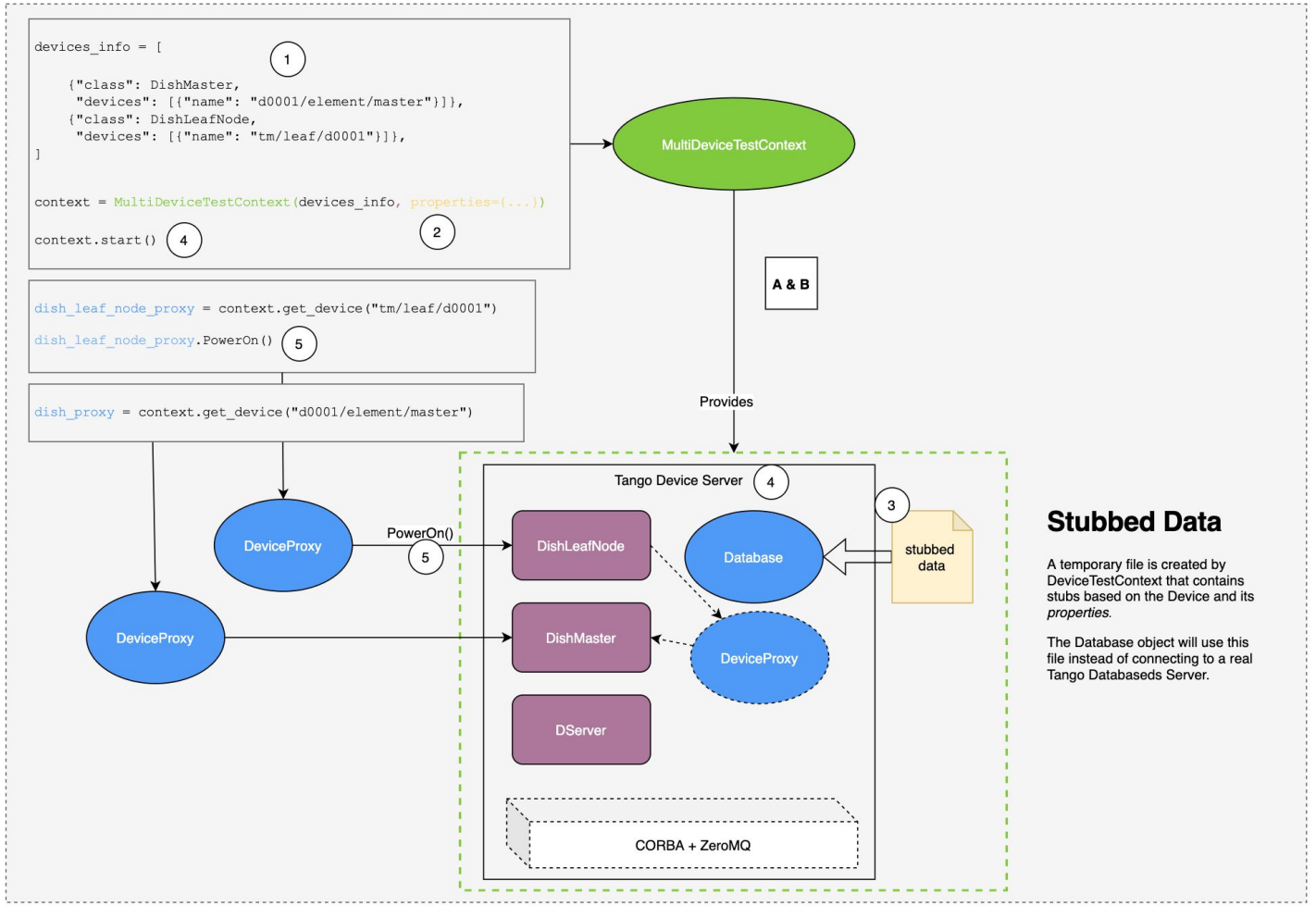


Stubbed Data

A temporary file is created by DeviceTestContext that contains stubs based on the Device and its *properties*.

The Database object will use this file instead of connecting to a real Tango Databaseds Server.

Testing with *MultiDeviceTestContext*



More details

General techniques were discussed at Tango 2020-11 Status Meeting

Slides here (backup)

Video recording here, starting at 1:02:22 mark

Take home: keep the business logic outside the Tango device

Additional resources

Useful links

Examples from this presentation

https://gitlab.com/tango-controls/pytango/-/tree/develop/examples/training

PyTango documentation

https://pytango.readthedocs.io

General Tango documentation

https://tango-controls.readthedocs.io

Tango community forum

https://www.tango-controls.org/community/forum/

SKAO Tango Dockerfiles:

https://gitlab.com/ska-telescope/ska-tango-images

SKAO artefact repository, for Docker images:

https://artefact.skao.int

Thanks!

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