project-name Documentation

Release 0.1.0

author

Aug 23, 2023

CONTENTS:

This project defines a set of Docker images and Docker compose files that are useful for TANGO control system development.

CHAPTER

ONE

BUILDING THE DOCKER IMAGES

The following Docker images are built by this project:

Docker image	Description
pytango-builder	Extends ska/tango-cpp, adding PyTango Python bindings and other tools for building python libraries
pytango-runtime	Extends pytango-builder without any tools for development.
tango-admin	The TANGO tango-admin tool.
tango-cpp	Core C++ TANGO libraries and applications.
tango-databaseds	The TANGO databaseds device server.
tango-db	A MariaDB image including TANGO database schema. Data is stored separately in a volume.
tango-dependencies	A base image containing TANGO's preferred version of ZeroMQ plus the preferred, patched version of OmniORB.
tango-dsconfig	The TANGO MAXIV tool for managing the tango-db
tango-itango	itango, a Python shell for interactive TANGO sessions.
tango-java	As per ska/tango-cpp, plus Java applications and bindings.
tango-jive	The TANGO jive tool
tango-libtango	Same as tango-cpp.
tango-panic	The TANGO panic tool
tango-panic-gui	The TANGO panic tool with xfce4 and vnc.
tango-pogo	Image for running Pogo and displaying Pogo help. Pogo output can be persisted to a docker volume or to the host machine.
tango-pytango	same as pytango-runtime.
tango-rest	An image containing mtango-rest, which acts as a REST proxy to a TANGO system.
tango-test	The TANGO test device server.
tango-vnc	An image containing xfce4 and vnc in order to enable desktop application such as jive.

To build and register the images locally, from the root of this repository execute:

```
cd docker
# build and register TBC/tango-cpp, TBC/tango-jive, etc. locally
make build
```

Optionally, you can register images to an alternative Docker registry account by supplying the `CAR_OCI_REGISTRY_HOST` and `CAR_OCI_REGISTRY_PREFIX` Makefile variables, e.g.,

```
# build and register images as foo/tango-cpp, foo/tango-jive, etc.
make CAR_OCI_REGISTRY_PREFIX=foo build
```

1.1 Building with alternatives to Docker

You can use a daemon-less unpriveleged alternative to Docker to build container images using the dockerfiles hosted in this project by setting the `IMAGE_BUILDER` Makefile variable. This alternative image builder must be fully compatible with `docker build` options. Currently, Img and Podman were tested and they work without any major issues.

To use, e.g., Img:

```
# build and register images as TBC/tango-cpp, TBC/tango-jive, etc.
make IMAGE_BUILDER=img build
```

For more information about IMG, including installation:

https://github.com/genuinetools/img

For more information about Podman:

https://github.com/containers/podman

1.2 Pushing the images to a Docker registry

Push images to the default Docker registry located at https://docker.io by using the `make push` target.

```
# push the images to the Docker registry, making them publicly
# available as foo/tango-cpp, foo/tango-jive, etc.
make CAR_OCI_REGISTRY_PREFIX=foo push
```

Images can also be pushed to a custom registry by specifying a `CAR_OCI_REGISTRY_HOST` Makefile argument during the `make build` and `make push` steps, e.g.,

```
# build and tag the images to a custom registry located at
# http://test_registry:5000
make CAR_OCI_REGISTRY_PREFIX=foo CAR_OCI_REGISTRY_HOST=my_registry.org:5000 build

# Now push the images to the remote custom registry
make CAR_OCI_REGISTRY_PREFIX=foo CAR_OCI_REGISTRY_HOST=my.registry.org:5000 push
```

If your images were built with alternatives to Docker like Img or Podman do not forget to set the `IMAGE_BUILDER` variable accordingly.

CHAPTER

TWO

HELM CHARTS AVAILABLE ON SKA-TANGO-IMAGES REPOSITORY

There are two helm charts available on this repository: one is called `ska-tango-base` and the other is the `ska-tango-util`. There is another helm chart, called `ska-tango-images`, which is used only for testing purposes.

2.1 The ska-tango-base helm chart

The `ska-tango-base` helm chart is an application chart which defines the basic TANGO ecosystem in kubernetes.

In specific it defines the following k8s services:

- tangodb: it is a mysql database used to store configuration data used at startup of a device server (more information can be found here. If the `global.operator` is true then this won't be generated in favour of a databaseds resource type. More information available here
- databaseds: it is a device server providing configuration information to all other components of the system as well as a runtime catalog of the components/devices (more information can be found here.
- itango: it is an interactive Tango client (more information can be found here.
- vnc: it is a debian environment with x11 server and vnc/novnc installed on it.
- tangotest: it is the tango test device server (more information can be found here.

2.2 The ska-tango-util helm chart

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

In specific it defines the following helm named template:

- configuration (deprecated): it creates a k8s service account, a role and role binding for waiting the configuration job to be done and a job for the dsconfig application to apply a configuration json file set into the values file;
- deviceserver (deprecated): it creates a k8s service and a k8s statefulset for a instance of a device server;
- multidevice-config: it creates a ConfigMap which contains the generated dsconfig json configuration file, the boostrap script for the dsconfig application and a python script for multi class device server startup; if the `global.operator` is true then this won't be generated. More information available here;
- multidevice-job: it creates a job for the dsconfig application to apply a configuration json file set into the values file; if the `global.operator` is true then this won't be generated. More information available here;

- multidevice-sacc-role: it creates a k8s service account, a role and role binding for waiting the configuration job to be done; if the `global.operator` is true then this won't be generated. More information available here;
- multidevice-svc: it creates a k8s service and a k8s statefulset for a device server tag specified in the values file. If the `global.operator` is true then this won't be generated in favour of a DeviceServer k8s type. More information available here
- deviceserver-pvc: it optionally creates a volume for the deviceserver when it contains the dictionary *volume*. The subkeys are *name*, *mountPath* and *storage*. See example below.
- operator: it creates a k8s DeviceServer type of k8s resources.

With the introduction of the SKA TANGO Operator k8s controller the library is also able to generate DeviceServer type of resources. This can be activate by setting the parameter `global.operator`.

2.2.1 Dsconfig generation

Dsconfig is an application which configure the tango database with the help of a json file. With ska-tango-util a device derver is configurable using specifications in a values.yaml file of the chart instead of the dsconfig.json file, where all device servers have a configuration yaml block. Below there is an example of a values file that can be used with the ska-tango-util multi device definition:

```
deviceServers:
    theexample:
        name: "theexample-{{.Release.Name}}"
        function: ska-tango-example-powersupply
        domain: ska-tango-example
        instances: ["test"]
        polling: 1000
        entrypoints:
            - name: "powersupply.PowerSupply"
            path: "/app/module_example/powersupply.py"
            - name: "EventReceiver.EventReceiver"
            path: "/app/module_example/EventReceiver.py"
            - name: "Motor.Motor"
            path: "/app/module_example/Motor.py"
        server:
            name: "theexample"
            instances:
            - name: "test2"
                classes:
                - name: "PowerSupply"
                devices:
                - name: "test/power_supply/2"
                    properties:
                    - name: "test"
                    values:
                    - "test2"
            - name: "test"
                classes:
                - name: "PowerSupply"
                devices:
                - name: "test/power_supply/1"
                    properties:
                    - name: "test"
                    values:
```

(continues on next page)

(continued from previous page)

```
                - "test2"
            - name: "EventReceiver"
            devices:
            - name: "test/eventreceiver/1"
            - name: "Motor"
            devices:
            - name: "test/motor/1"
                properties:
                - name: "polled_attr"
                values:
                - "PerformanceValue"
                - "{{ .Values.deviceServers.theexample.polling }}"
                attribute_properties:
                - attribute: "PerformanceValue"
                properties:
                - name: "rel_change"
                    values:
                    - "-1"
                    - "1"
    class_properties:
        - name: "PowerSupply"
        properties:
            - name: "aClassProperty"
            values: ["67.4", "123"]
            - name: "anotherClassProperty"
            values: ["test", "test2"]
    depends_on:
        - device: sys/database/2
    image:
        registry: "{{.Values.tango_example.image.registry}}"
        image: "{{.Values.tango_example.image.image}}"
        tag: "{{.Values.tango_example.image.tag}}"
        pullPolicy: "{{.Values.tango_example.image.pullPolicy}}"
    volume:
        name: firmware
        mountPath: /firmware
```

Fields explained:

- **deviceServers** : contains a list of all device server defined
- **instances** : On this field the user can define which of the instances defined in the server tag are going to be created on the deviceServer.
- **entrypoints** : The number of entrypoints should correspond to the defined in the server tag field.
  - **name** : This is a **mandatory** field at entrypoints. The name field has to have a format like NameOfTheModule.NameOfTheClass.
  - **path** : This is a **optional** field at entrypoints. The path field is the path of the module that has the class of the device. This field may not be present **only** if the module is included in the list of directories that the interpreter will search, one example is if the modules are installed with pip.
- **server** : It's the equivalent of the dsconfig json file and define everything needed for a device server.
  - **intances** : A list of all instances for a device server. For each instance a number of devices can be defined together with the relative properties.
- **class_properties** : On this field you can list your class properties.

The device server configuration, like the above one, needs to be added to the values.yaml file. Below there is an example of how to add it (by splitting the definitions in different files):

```
deviceServers:
    theexample:
        instances: ["test2"]
        polling: 1000
        file: "data/theexample.yaml"
```

Fields explained:

- **file** : This field specifies the path of the device server configuration block as shown above. Note:. This file should be included in a data folder inside the chart.
- **polling** : This field is referenced in the above device server configuration block. In fact the ska-tango-util device server definition template some of the field composing it (like the properties). In the above example the *polled_attr* property of the *test/motor/1* device takes its value from this field. As a consequence, this field allows us to change the value of the *polled_attr* property in the parent chart.
- **instances** : If **instances** has values in the value file, this takes precedence over the data file **instances** field.

The use of the yaml file allows users to have a cleaner and more understandable view of the DeviceServer configurations compared to a json file configuration. The helm template multidevice-config creates a ConfigMap which contains the generated dsconfig that was loaded and converted to a json type file from the values.yaml file described above.

2.2.2 How to use the defined helm named template

A example on how to set up your k8s namespace with the helm named templates, described in the beginning of this *section*, can be seen on ska-tango-example repository. This templates are called by the below template present on the ska-tango-example repository:

```
{{ $localchart := . }}

{{- range $key, $deviceserver := .Values.deviceServers }}

{{- if hasKey $deviceserver "file"}}

{{- $filedeviceserver := $.Files.Get $deviceserver.file | fromYaml }}
{{- $_ := set $filedeviceserver "instances" (coalesce $localchart.Values.global.instances $deviceserver.instances $filedeviceserver.instances) }}
{{- $context := dict "name" $key "deviceserver" $filedeviceserver "image" $deviceserver.image "local" $localchart }}
{{ template "ska-tango-util.multidevice-config.tpl" $context }}
{{ template "ska-tango-util.multidevice-sacc-role.tpl" $context }}
{{ template "ska-tango-util.multidevice-job.tpl" $context }}
{{ template "ska-tango-util.multidevice-svc.tpl" $context }}
{{- $volume_context := dict "volume" $filedeviceserver.volume "local" $localchart }}
{{ template "ska-tango-util.deviceserver-pvc.tpl" $volume_context }}

{{- else }}

{{- $_ := set $deviceserver "instances" (coalesce $localchart.Values.global.instances $deviceserver.instances) }}
{{- $context := dict "name" $key "deviceserver" $deviceserver "image" $deviceserver.image "local" $localchart }}
{{ template "ska-tango-util.multidevice-config.tpl" $context }}
{{ template "ska-tango-util.multidevice-sacc-role.tpl" $context }}
```

(continues on next page)

(continued from previous page)

```
23 {{ template "ska-tango-util.multidevice-job.tpl" $context }}
24 {{ template "ska-tango-util.multidevice-svc.tpl" $context }}
25 {{- $volume_context := dict "volume" $deviceserver.volume "local" $localchart }}
26 {{ template "ska-tango-util.deviceserver-pvc.tpl" $volume_context }}
27
28
29 {{- end }}
30
31 {{- end }} # deviceservers
```

Tango-example template description:

- **Line 3** to **Line 29** : This template will iterate through each field under deviceServers on the values.yaml file.
- **Line 5** to **Line 15** : If the device server has a file field we will get that configuration file and use it. (**Best Practice**: Add the deviceServer configuration in the data folder and then pass the path of it in the file field of the deviceServer).
- **Line 17** to **Line 26** : If there is no file field it means that the configuration of this device was done inside the value.yaml. (**Note:** Making the configuration of the device inside the values.yaml makes this file bigger becoming harder to read and understand)
- **Line 7** : As discussed before it is possible to have a instances field in the values.yaml file and in the data file, it is also possible to have instances defined as a global field. It is being used a coalesced function that takes the first not null value of the list. The priority is, first it takes the instance value from the global variable if there is none it takes it from the values file and then from the data file.
- **Line 19** : Same as line 8 but without the possibility of having the instance field on the data file.
- **Line 9** and **Line 20** : Context is a list of variables that will passed as arguments to the templates.
- **Line 14** to **Line 15**: Use and set the context for persistent volume claims attached to teh deviceserver
- **Line 25** to **Line 26**: same as 14 to 15
- **Templates** : There are five templates already described before. Each template will be called for each deviceServer as they are inside the range loop (line 3).

CHAPTER

THREE

SKA TANGO-CONTROLS DOCKER IMAGES ON KUBERNETES

The following are a set of instructions of running the SKA TANGO-controls docker images made by SKA on Kubernetes, and has been tested on minikube v1.12.3 with k8s v1.18.3 Docker 19.03.8 on Ubuntu 18.04.

CHAPTER

FOUR

MINIKUBE

Using Minikube enables us to create a single node stand alone Kubernetes cluster for testing purposes. If you already have a cluster at your disposal, then you can skip forward to the section *Running the SKA TANGO-controls docker images on Kubernetes*.

The generic installation instructions are available at https://kubernetes.io/docs/tasks/tools/install-minikube/. A deployment of Minikube that will support the standard features required for the SKA is available at https://gitlab.com/ska-telescope/sdi/deploy-minikube.

Once you have finished the deployment you may need to fixup your permissions:

```
sudo chown -R ${USER} /home/${USER}/.minikube
sudo chgrp -R ${USER} /home/${USER}/.minikube
sudo chown -R ${USER} /home/${USER}/.kube
sudo chgrp -R ${USER} /home/${USER}/.kube
```

Once completed, minikube will also update your kubectl settings to include the context `current-context: minikube` in `~/.kube/config`. Test that connectivity works with something like:

```
$ kubectl get pods -n kube-system
NAME                                READY   STATUS    RESTARTS   AGE
coredns-86c58d9df4-5ztg8            1/1     Running   0          3m24s
...
```

4.1 Helm Chart

The Helm Chart based install of the SKA TANGO-controls docker images relies on Helm (surprise!). If your system does not have a running version of Helm the easiest way to install one is using the install script:

```
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash
```

4.2 Cleaning Up

Note on cleaning up:

```
minikube stop # stop minikube - this can be restarted with minikube start
minikube delete # destroy minikube - totally gone!
rm -rf ~/.kube # local minikube configuration cache
# remove all other minikube related installation files
sudo rm -rf /var/lib/kubeadm.yaml /data/minikube /var/lib/minikube /var/lib/kubelet /
↪etc/kubernetes
```

(continues on next page)

(continued from previous page)

4.3 Running the SKA TANGO-controls docker images on Kubernetes

The basic configuration for each component of the SKA TANGO-controls docker images is held in the `values.yaml` files.

We launch the SKA TANGO-controls docker images with:

```
$ make k8s-install-chart
```

To clean up the Helm Chart release:

```
$make k8s-uninstall-chart
```

4.4 Vault Secrets

When deploying to a remote cluster we may want to use the vault to fetch secrets.

The tango-base charts are configured to allow the use of vault in the **tangodb** and **databaseds** database containers.

When the vault is enable in your chart, vault annotations are added to the chart templates allowing the secrets to be injected in the container

This secret file, in the examples, are formatted as a key/value pairs allowing us the ability to source the file and consequently add the variables as environment variables. This is useful for database containers.

But be aware that sourcing the secret file, depending on your container specification, may disrupt its normal startup flow.

After sourcing the file you need to run the necessary scrips / commands so that your application starts correctly. This changes from application to application.

4.5 Enable vault secrets in the tango charts

To use vault configure in the values.yml (this is the tangodb example):

```
tangodb:

...

  vault:
    useVault: true
    secretPath: stfc
    role: kube-role
```

parameter|description :—–:|:—–: useVault| turn it on/off secretPath| starting path for the secret in the server role| vault role to use

If you are using **minikube** set the **useVault** parameter to false, remove it or remove the vault section entirely.