PyCharm - Docker configuration

This tutorial shows how to configure PyCharm Professional for SKA control system development using the SKA Docker images.

PyCharm can be configured to use the Python interpreter inside a Docker image, which allows:

* development and testing without requiring a local Tango installation;
* the development environment to be identical to the testing and deployment environment, eliminating problems that occur due to differences in execution environment.

Follow the steps below to configure PyCharm to develop new code and run tests for the tango-example project using the Docker images for the project.

Prerequisites-

Make sure that the following prerequistes are met:

 Docker is installed, as described on the page Docker Docs.

 PyCharm Professional must be installed. PyCharm Community is not sufficient!

 You have basic familiarity with PyCharm. If this is the first time you have used PyCharm, follow the First Steps tutorials so that you know how to use PyCharm to develop, debug, and test a simple Python application using a local Python interpreter.

Clone the tango-example project

PyCharm allows you to check out (in Git terms clone) an existing repository and create a new project based on the data you’ve downloaded.

1. From the main menu, choose VCS | Checkout from Version Control | Git, or, if no project is currently opened, choose Checkout from Version Control | Git on the Welcome screen.

2. In the Clone Repository dialog, specify the URL of the tango-example repository (you can click Test to make sure that connection to the remote can be established).

3. In the Directory field, specify the path where the folder for your local Git repository will be created into which the remote repository will be cloned.

4. Click Clone, then click Yes in the subsequent confirmation dialog to create a PyCharm project based on the sources you have cloned.

Build the application image-

With the source code source code checked out, the next step is to build a Docker image for the application. This image will contain the Python environment which will we will later connect to PyCharm.

Begin a terminal session in the cloned repository directory and build the image:

The last line of terminal output displays the name and tags of the resulting images.

Configure the remote interpreter-

Following the official PyCharm documentation, configure Docker as a remote interpreter using the image you just created.

Click ‘OK’ to apply your changes.

You can now navigate through the project. As an exercise, open the source code for the PowerSupply class , which is defined in powersupply/powersupply.py. Notice that the IDE notifications and intellisense / code completion are now based on information gathered from the remote Docker interpreter. Below an import statement, try typing from tango import and activate code completion (ctrl+space). Notice how the tango packages installed in the Docker image are suggested to complete the statement.

Whenever you change the Python environment, for example by adding or removing dependencies in Piplock, after rebuilding the Docker image you should regenerate the project skeletons to make PyCharm aware of the changes. To do this, select File | Invalidate Caches / Restart… from the main menu.

Running unit tests-The tango-example project illustrates two types of unit test:

1. Self-contained unit tests that execute within the confines of a single Docker container. These tests use the Tango class DeviceTestContext, which provides a mock connection to a Tango database. In the tango-example project, these tests are found in tests/test\_1\_server\_in\_devicetestcontext.py.

2. Unit tests that exercise a device in a real Tango environment, with connections to a Tango database and other devices. utilise require a connection. In the tango-example project, these tests are found in tests/test\_2\_test\_server\_using\_client.py.

This tutorial illustrates how to run the self-contained unit tests described in 1.

From the main menu, choose Run | Edit Configurations… and click on the ‘+’ button to add a new configuration. From the menu that appears, select Python tests | pytest to add a new pytest test configuration.

1. Change the Target radio button to ‘Module Name’. Click ‘…’ to select the target, choosing test\_1\_server\_in\_devicetestcontext as the module to be tested.

2. Select ‘Project Default’ as the Python interpreter for this configuration.

Click ‘OK’ to accept your changes.

From the main menu, choose Run | Run…, then from the Run dialog that opens, select the configuration you just created. The unit tests will execute, with the results displayed in PyCharm’s Run panel.