Using elasticsearch for archiving with TANGO-controls framework

M. Di Carlo, M. Dolci

M. Di Carlo, M. Dolci, "Using elasticsearch for archiving with TANGO-controls framework," Proc. SPIE 12189, Software and Cyberinfrastructure for Astronomy VII, 121891P (29 August 2022); doi: 10.1117/12.2627030

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2022, Montréal, Québec, Canada

Using Elasticsearch for archiving with TANGO-controls framework

Di Carlo M.[a] and Dolci M.[1]

[a]INAF Osservatorio Astronomico d'Abruzzo, Teramo, Italy

ABSTRACT

The TANGO controls framework community has put a lot of effort in creating the HDB++ software system that is an high performance, event-driven archiving system. Its design allows storing data into traditional database management systems such as MySQL as well as NoSQL database such as Apache Cassandra. The architecture allow also to easily extend it to other noSql database like, for instance, Elasticsearch. This paper describes the extension for Elasticsearch made and how to use it alongside its graphical tool called Kibana.

Keywords: TANGO, HDB++, Elasticsearch, ELK, noSql DB

1. INTRODUCTION TO TANGO-CONTROLS AND HDB++

The tango controls system is built on top of the CORBA (Common Object Request Broker Architecture[1]) standard, limiting the possibility to introduce new objects with the IDL (interface definition language) to the "device" concept, in order to build distributed control system. Therefore, a device is mainly an object with attributes within a process called "device server".[2] One of its core feature is the event model, which allows the communication between devices according to a predefined event. One of the event typology available is the archive event, that can be configured to be fired with an absolute or relative change in value (for a particular attribute of a device) or it can be configured to archive in a polling fashion.[3]

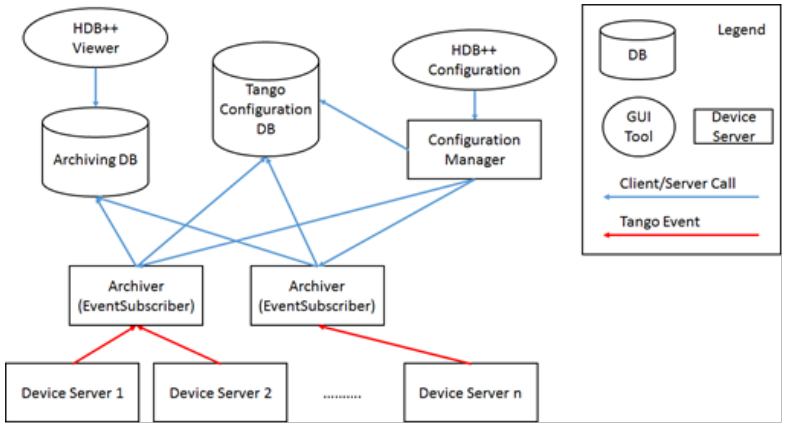

Figure 1. HDB++ Runtime View.

Fig. 1(available on the online documentation[3]) shows the main runtime components of the HDB+ system that are the Configuration Manager and the Event Subscriber (aka Archiver). The first one is a device that assists in adding, modifying, moving, deleting an attribute to/from the archiving system while the second one is the receiver of the event, the one that has to store the value of the configured attribute.

Further author information: Di Carlo M.: E-mail: matteo.dicarlo@inaf.it

Software and Cyberinfrastructure for Astronomy VII, edited by Jorge Ibsen, Gianluca Chiozzi, Proc. of SPIE Vol. 12189, 121891P · © 2022 SPIE
0277-786X · doi: 10.1117/12.2627030

1.1 Abstract DB

Fig. 2 shows the architecture of the HDB++ in term of modules (mainly c++ library). The two main module are the "Event Subscriber" and the "Configuration Manager". Both of them use a third module, called "Database Abstraction Layer", that define two interfaces (namely DBFactory and AbstractDB) for extending the archiving system to other database system. Extending the HDB++ system, therefore, is very easy: it is only needed to implement the above two interfaces. The figure shows only two implementations of the abstraction layer that allows the interaction (for storage purpose) to the Mysql database system[4] and the Cassandra database system[5] but other implementation are available such as TimescaleDB.[6]

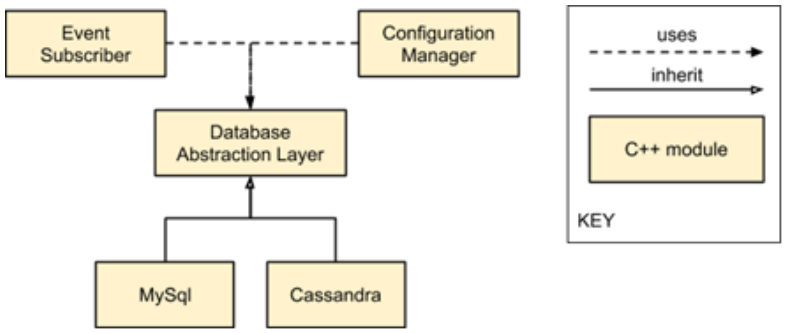

Figure 2. HDB++ Module View.

1.2 Data Model

The tradeoff of the simplicity of extending the system is a fixed data model, which is the schema of the database. This means that the archiving interfaces (both the "Event Subscriber" and the "Configuration Manager") cannot be changed and what is archived must always have the same shape. In essence, it consists of five tables, shown in Figure 3, which stores the following information:

- AttributeConfiguration: for each attribute (to store in DB when needed) there is an entry in this table which associate it with a specific data type;
- AttributeConfigurationHistory: for each operation on attributes (add/remove/start/stop), a row is inserted with the timestamp of the operation;
- AttributeParameter: for each attribute (to store in DB when needed) there is an entry in this table which associate it with the list of parameter already stored in the TANGO database;
- AttributeEventData: for each event there is a row which stores the information associated with the event (mainly timing and quality factor);
- Value: for each event stored in the AttributeEventData, there is an entry in a Value table (double, long, string and so on).

2. INTRODUCTION TO ELASTICSEARCH

Elasticsearch[7] is a real-time distributed search and analytics engine. The term "real-time" refers to the ability to search (and sometimes create) data as soon as they are produced; traditionally, in fact, web search crawls and indexes web pages periodically, returning results based on relevance to the search query. It is distributed because its indices are divided into shards with zero or more replicas. But the main ability is the analytics engine which allows the discovery, interpretation, and communication of meaningful patterns in data. It is based on Apache Lucene,[8] a free and open-source information retrieval software library, therefore Elasticsearch is very good for full text search (like for logging data or text files in general). It is developed alongside a data-collection and log-parsing engine called Logstash, and an analytics and visualization platform called Kibana. The three products are designed for use as an integrated solution, referred to as the "Elastic Stack" (formerly the "ELK stack").

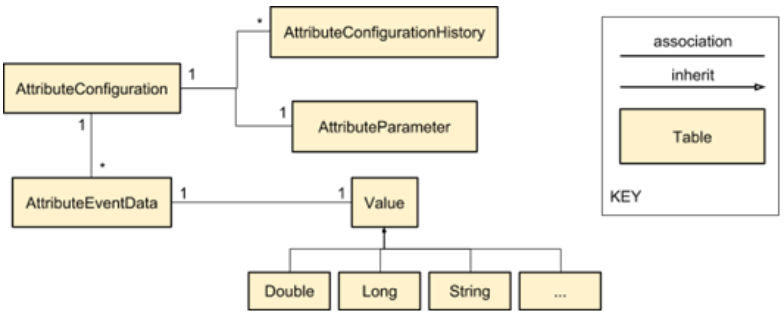

Figure 3. HDB++ Data Model.

2.1 Elasticsearch as noSQL database

Looking at the definition given in 7, it is not straightforward to assert that Elasticsearch is a noSql Database; the problem is the definition of noSql which it is not very precise: "Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable".[9] Since that definition is not exact, the only possibility is to summarize the main features of Elasticsearch in order to understand its qualities. They are:

- no transaction: no support for transaction;
- schema flexible: there is no need to specify the schema upfront;
- relations: denormalization, parent-child relations and nested objects;
- robustness: to properly work, elasticsearch requires that memory is abundant;
- distributed: it is a CP-system in the CAP (Consistency-Availability-Partition tolerance) theorem;[10]
- no security: there is no support for authentication and authorization.

When using a software like Apache Lucene or Elasticsearch, all the features must be well considered and, also, an important consideration deserves the relations. ELK see data as everything is flat: basically every document, stored in ELK, is independent and therefore every document should contain all of the information required to decide whether it matches a query. In particular, this helps in indexing, in searching and in scalability since documents can be spread across multiple nodes.

3. HDB++ LIBRARY FOR ELASTICSEARCH

The selected development language was C++ mainly because HDB++ is made in that language. It includes the ability to work with REST[11] and with Json data.[12] The details of the implementation can be found at 13 and 14.

Fig. 4 shows the class diagram of the implementation done. The central class is the ElasticDB which realizes the main methods of the interface AbstractDB (that uses the classes HdbEventDataType, EventData and AttrConfEventData belonging to the TANGO core library). The classes AttributeName and Log are used to extract the information needed from the full attribute name and to log data messages. The class DAL (refer to Data Access Layer[15]) provides to the ElasticDB class a simple access to the Elasticsearch functionalities. In fact, it implements four private methods which represents the basic operation (except for the delete operation) that can be done with a database (Insert, Update, Get single entity and search for entities).

With the help of those simple methods and for each information to store, there are one or more method to save and read the data. For example, if a new archive event arrives to the system (represented with the class EvenData), the HdbPPELK will create the object (in this case an instance of EventData) and simply pass it to the DAL which will store it.

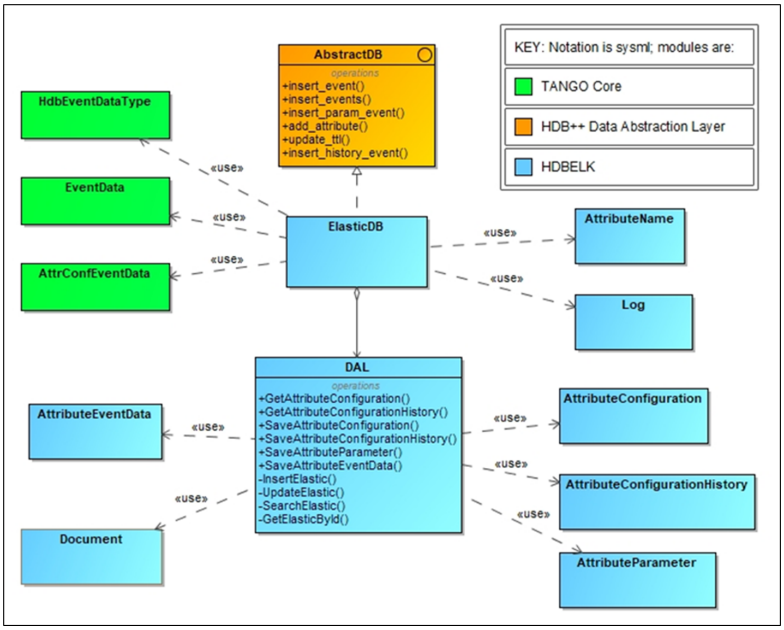

Figure 4. HDB++ library for Elasticsearch class diagram.

4. TESTING

Compared to the work done in 14, this work represents an improvement of the initial work in two aspects: testing and containerization. In specific, it has been evaluated two main options for testing. The first one is the same used for the HDB++ library for Apache Cassandra backend[5] called Catch2.[16] The second option is Google Test.[17] While both are good options for C++ testing, the selection was for Google Tests because it is based on the popular xUnit architecture. In specific, tests are independent, repeatable, well organized, portable, reusable and when it fails should provide as much information about the problem as possible. It is also important to consider that there is a different terminology:

- Google test case === test suite === group of tests;
- Google Test === Test === C++ macro TEST();
- Google test fixture class: when grouping tests that share common objects.

Fig. 5 and fig. 6 shows respectively an example of a test made with the selected testing framework and the configuration made for CMake.

```
TEST_F(DALTest, SaveAttributeEventData){
    bool result = false;
    if (!_DAL->GetAttributeConfiguration(*attr_conf)) {
        result = _DAL->SaveAttributeConfiguration(*attr_conf);
        EXPECT_EQ(result, true) << "Errors -> " << _DAL->GetErrors() << " QueryResult -> " << _DAL->GetQueryResultArray();
    }
    attr_event_data->SetAttributeConfigurationId(attr_conf->GetID());
    result = _DAL->SaveAttributeEventData(*attr_event_data);
    EXPECT_EQ(result, true) << "Errors -> " << _DAL->GetErrors() << " QueryResult -> " << _DAL->GetQueryResultArray();
}
```

Figure 5. Test examples.

5. CONTAINERIZATION

As said above, together with testing the main improvement of this paper to the one presented in 14 is the contanerization done. In specific, a Dockerfile has been developed which include all the necessary dependencies for the archiver device such as:

```
enable_testing()

add_executable(
    hello_test
    test/hello_test.cc
    test/attribute_name_test.cc
    test/dal_test.cc
    # test/elastic_db_test.cc
    # add here more tests
)

target_include_directories(hello_test
    PUBLIC ${INCLUDE_PATHS}
    PUBLIC ${CMAKE_CURRENT_LIST_DIR}/src
    PUBLIC "${PROJECT_BINARY_DIR}")

target_link_libraries(
    hello_test
    gtest_main
    ${LIBS_DIR}
    ${PROJECT_SOURCE_DIR}/${OUTPUT_DIR}/libElasticHDB.so
)

include(GoogleTest)
gtest_discover_tests(hello_test)
```

Figure 6. Google tests configuration.

- the Interface library for the HDB++,[18]
- the TANGO-controls framework dependencies,[2]
- the ReST client for C++,[19]
- JSON for Modern C++,[20]
- Tango device server for the HDB++ Event Subscriber,[21]
- Tango device server for the HDB++ Configuration Manager.[22]

Most of the dependencies are included with the live at head philosophy.[23] This means that this project refer to the main branch of the archiver libraries such as the Event subscriber and the configuration manager. This philosophy has been selected for mainly two reasons: the first one is to avoid the handling of multiple versions for those projects and secondary because that old versions appears to be not working perfectly compared to the main branches.

The first and most important result of the containerization work is that it is possible to have a reproducible environment in many different contest. For instance, thanks to Gitlab,[24] it was possible to realize a pipeline that build, test and publish a container image which garuantee a very good confidence in term of reliability. Fig. 7 and fig. 8 shows the gitlab pipeline made available for the project and the test job which shows how the image built in the previous steps of the pipeline is used in the test job.

Together with the Dockerfile it has been developed an integration with kubernetes[25] and helm chart[26] in order to manage the installation of the HDB++ and the Elasticsearch database. In specific it has been used widely the resources available from the SKA repositories such as the SKA Tango charts utility[27] (ska-tango-base and ska-tango-util)

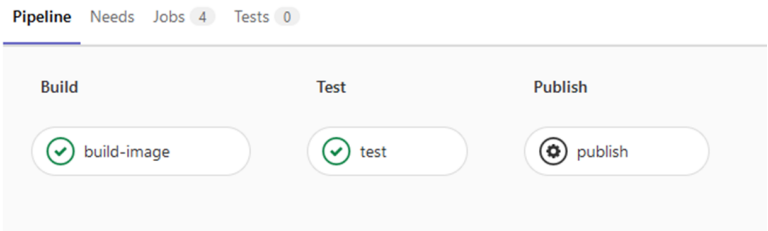

Figure 7. Gitlab Pipeline.

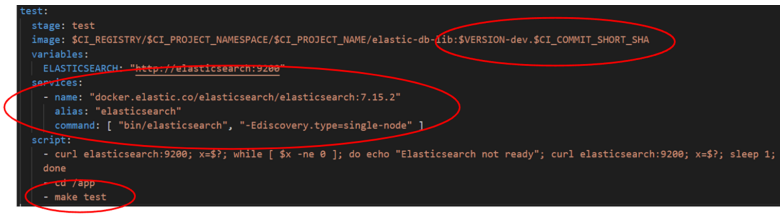

Figure 8. Gitlab Test Job.

In order to take the full advantage of the Gitlab pipeline, a Makefile has been developed which allows, with only one command, to:

- Build or test the C++ library locally;
- build and push the docker image of the project;
- run all tests in a container and start an elasticsearch database in a container;
- install, uninstall or reinstall the application using Helm.[26]

6. CONCLUSION

This work is mainly an extension of what presented in 14 for testing and containerization. In specific a new Docker image and helm chart have been created with a live at head philosophy;[23] tests have been made with google and the gitlab pipeline has been enabled so that the building, the testing and, in case of request, the publish can ben done.

ACKNOWLEDGMENTS

This unnumbered section is used to identify those who have aided the authors in understanding or accomplishing the work presented and to acknowledge sources of funding.

REFERENCES

[1] "Corba." `www.corba.org`. (Accessed: 25 May 2022).
[2] "Tango-controls framework." `https://www.tango-controls.org/`. (Accessed: 25 May 2022).
[3] "Hdb++ introduction." `http://tango-controls.readthedocs.io/en/latest/tools-and-extensions/archiving/HDB++.html`. (Accessed: 25 May 2022).
[4] "Library for hdb++ implementing mysql schema." `https://github.com/tango-controls-hdbpp/libhdbpp-mysql`. (Accessed: 25 May 2022).
[5] "Hdb++ library for apache cassandra backend." `https://github.com/tango-controls-hdbpp/libhdbpp-cassandra`. (Accessed: 25 May 2022).
[6] "Hdb++ backend library for the timescaledb extenstion to postgresql." `https://github.com/tango-controls-hdbpp/`. (Accessed: 25 May 2022).
[7] "Elk." `www.elastic.co`. (Accessed: 25 May 2022).

[8] "Apache lucene." https://lucene.apache.org. (Accessed: 25 May 2022).
[9] "nosql." nosql-database.org. (Accessed: 25 May 2022).
[10] "Cap theorem." en.wikipedia.org/wiki/CAP_theorem. (Accessed: 25 May 2022).
[11] "Rest." en.wikipedia.org/wiki/Representational_state_transfer. (Accessed: 25 May 2022).
[12] "Json." www.json.org. (Accessed: 25 May 2022).
[13] "Hdb++ library for elasticsearch." https://gitlab.com/tango-controls/hdbpp/libhdbpp-elk. (Accessed: 25 May 2022).
[14] Carlo, M. D., "Hdb@elk: another nosql customization for the hdb++ archiving system," *Proc. SPIE 10707* (2018).
[15] "Data access layer." https://en.wikipedia.org/wiki/Data_access_layer. (Accessed: 25 May 2022).
[16] "Catch2 - unit testing framework for c++." https://github.com/catchorg/Catch2. (Accessed: 25 May 2022).
[17] "Googletest primer." https://google.github.io/googletest/primer.html. (Accessed: 25 May 2022).
[18] "Interface library for the hdb++." https://gitlab.com/tango-controls/hdbpp/libhdbpp.git. (Accessed: 25 May 2022).
[19] "Rest client for c++." https://github.com/mrtazz/restclient-cpp. (Accessed: 25 May 2022).
[20] "Json for modern c++." https://github.com/nlohmann/json. (Accessed: 25 May 2022).
[21] "Tango device server for the hdb++ event subscriber." https://github.com/tango-controls-hdbpp/hdbpp-es.git. (Accessed: 25 May 2022).
[22] "Tango device server for the hdb++ configuration manager." https://github.com/tango-controls-hdbpp/hdbpp-cm.git. (Accessed: 25 May 2022).
[23] "Live at head." https://abseil.io/about/philosophy#upgrade-support. (Accessed: 25 May 2022).
[24] "Gitlab." https://about.gitlab.com/. (Accessed: 25 May 2022).
[25] "Kubernetes." https://kubernetes.io/. (Accessed: 25 May 2022).
[26] "Helm." https://helm.sh. (Accessed: 25 May 2022).
[27] "Ska tango charts." https://developer.skao.int/projects/ska-tango-images/en/latest/helm_charts.html. (Accessed: 25 May 2022).