18th Int. Conf. on Acc. and Large Exp. Physics Control Systems ICALEPCS2021, Shanghai, China JACoW Publishing
ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV025

TangoGraphQL: A GraphQL BINDING FOR TANGO CONTROL SYSTEM WEB-BASED APPLICATIONS

J.L. Pons, European Synchrotron (E.S.R.F), Grenoble, France

Abstract

Web-based applications have seen a huge increase in popularity in recent years, replacing standalone applications. GraphQL provides a complete and understandable description of the data exchange between client browsers and back-end servers. GraphQL is a powerful query language allowing API to evolve easily and to query only what is needed. GraphQL also offers a WebSocket based protocol which perfectly fits to the Tango event system. Lots of popular tools around GraphQL offer very convenient way to browse and query data. TangoGraphQL is a pure C++ http(s) server which exports a GraphQL binding for the Tango C++ API. TangoGraphQL also exports a GraphiQL web application which allows to have a nice interactive description of the API and to test queries. TangoGraphQL has been designed with the aim to maximize performances of JSON data serialization, a binary transfer mode is also foreseen.

INTRODUCTION

Today, at the ESRF [1], we use mainly Java standalone applications for the accelerator control system. These applications are built on top of the Java Swing ATK framework [2] and Tango java APIs. Today, regarding GUI technologies, we see almost only development around web technologies such as React, Angular, Vue.js, Bootstrap, Material UI, etc... It is natural that we migrate our GUIs to web based applications. Java ATK is based on the Model View Controller model. React (Facebook) offers a very convenient way to implement this model using hooks. GraphQL [3], initially developed by Facebook in 2012, was moved as open source to the GraphQL foundation. A JavaScript Tango Web ATK built on top of React and GraphQL is currently under development at the ESRF. This framework is designed in order to ease as much as possible the migration of our Java applications.

ARCHITECTURE

MVC Model using React

React function components offer hooks that can be used to implement the MVC model. The key idea is to use the state hook and to pass the setState function handle as a dispatcher to the model. A layoutEffect hook handles the subscription in the listener list of the model as shown in Fig. 1.

The model (the GraphQL client) makes access to Tango devices using the TangoGraphQL server either through basic HTTP requests or through WebSocket as shown in Fig. 2.

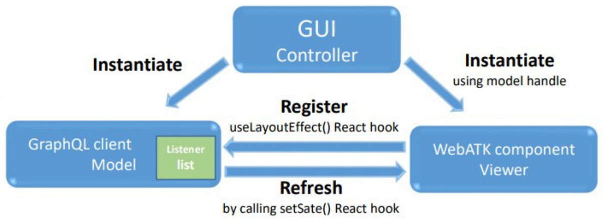


Figure 1: MVC using React.

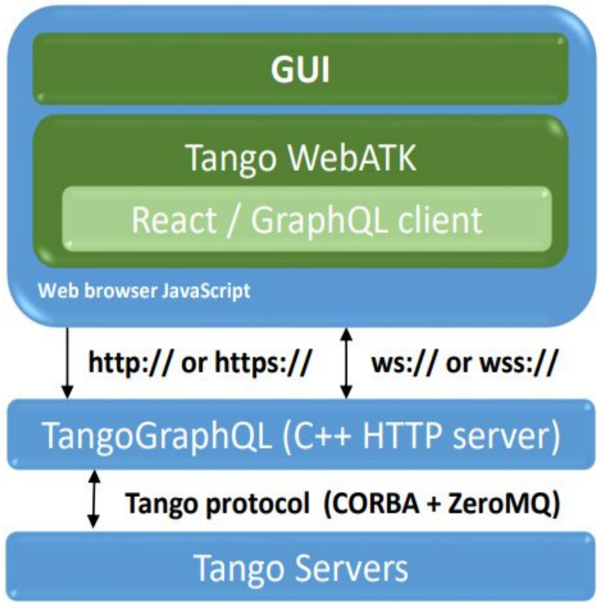
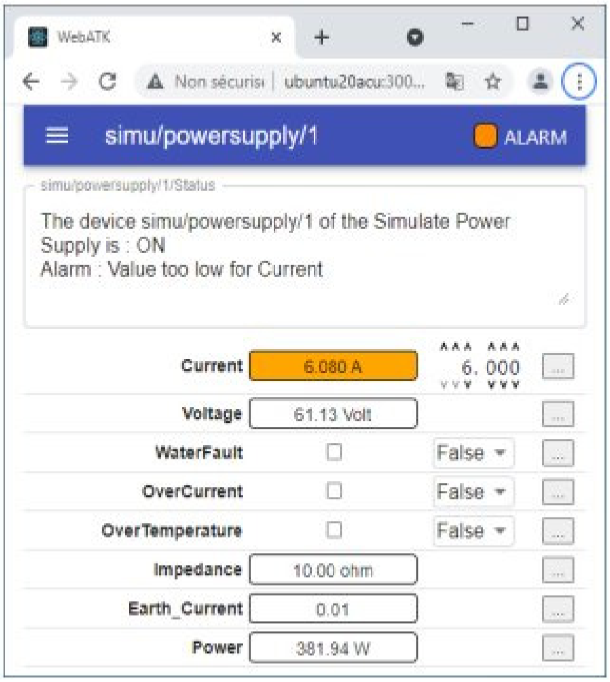


Figure 2: Architecture.

GRAPHQL

GraphQL vs REST

Compared to a REST API, GraphQL is a true query language allowing clients to query only what is needed in a single HTTP request. GraphQL also provides an introspection system giving information about the supported schema. GraphQL uses GraphQL query to perform introspection. The GraphQL foundation provides a web application called GraphiQL, based on this introspection system, enabling users to write query using modern tools such as completion, syntax checking and documentation browsing. It uses the Tango GraphQL schema definition [4] to provide all these features to the Tango GraphQL API. TangoGraphQL C++ server also exports a GraphiQL interface.



simu/powersupply/1/Status

The device simu/powersupply/1 of the Simulate Power
Supply is : ON
Alarm : Value too low for Current

Current	6.080 A	6.000
Voltage	61.13 Volt	
WaterFault	☐	False
OverCurrent	☐	False
OverTemperature	☐	False
Impedance	10.00 ohm	
Earth_Current	0.01	
Power	381.94 W	

Figure 3: WebATK Panel.

GraphQL Tango API

The GraphQL Tango API must provide all calls needed to build any applications including generic applications such as ATKPanel. WEBAtkPanel shown in Fig. 3 makes an introspection of the Tango device to display all its attributes and commands.

GraphQL defines 3 types of request: query, mutation and subscription. Queries are read only requests (basically a Tango read_attributes call), mutations are write requests (write_attributes or command_inout) and subscriptions are used to register to Tango events. The requests are sent to the server via a HTTP POST request.

When registering to an event, the TangoGraphQL server will push JSON frames when a Tango event is sent by the Tango server. When using the GraphQL over WebSocket protocol [5], the client (and the server) use the socket in a bidirectional mode (full duplex), which means that the client (or the web server) can send or receive requests at any time.

PERFORMANCE

Error Management

The GraphQL error management standard is not very convenient. When a node cannot be returned, the GraphQL standard impose that the node must be null and an additional errors JSON block has to be filled with errors. With large queries which may contain lots of errors, the browsing of this additional error tree is a bit heavy. TangoGraphQL server handles Tango errors as queries for best performance and keep GraphQL standard for parsing errors.

Binary Transfer

A GraphQL internal parser has been written from scratch and designed to support a true binary JSON transfer currently under development. Binary transfer is not a part of the GraphQL and JSON standard but it can be added without breaking the GraphQL compatibility by using HTTP header. Despite the fact that TangoGraphQL uses fast floating point serialization algorithm Grisu2 [6] from Florian Loitsch, image and spectrum need to be transferred in binary format in order to reach good performance at both server and client side. On the client side, binary transfer can be easily achieved using native JavaScript DataViews and ArrayBuffer. Table 1 shows timing measurement of the TangoGraphQL C++ server for a spectrum of 16383 random double values, the JSON encoding in text format is done using Grisu2.

Table 1: TangoGraphQL Server Performance

	TEXT, 16digits	BIN, 64bits
Get HTTP request	1ms	0.9ms
Tango reading	0.9ms	0.9ms
JSON encoding	6ms (Grisu2)	0.3ms
Send HTTP response	0.4ms	0.3ms

Asynchronous Group Calls

When the client needs to retrieve data from several devices (typically the state of n devices), the client can construct a GraphQL request using labeling. The server can detect labeled read_attributes calls and launches parallel asynchronous calls using Tango Group calls.

Load Balancing

TangoGraphQL is also a Tango server and can be configured, monitored or instantiated using standard Tango tools. It also has attributes that indicates number of connected clients, number and type of connections, network transfer, etc… The client can use this information to select the less loaded instance among a set of running TangoGraphQL servers as shown in Fig. 4.

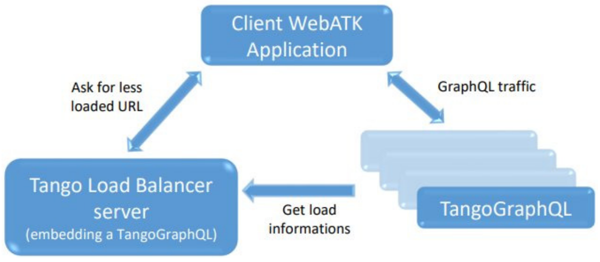

Client WebATK Application
Ask for less loaded URL
GraphQL traffic
Tango Load Balancer server (embedding a TangoGraphQL)
Get load informations
TangoGraphQL

Figure 4: Load balancing model.

TCP Connection

TangoGraphQL server uses multi-threaded connection handling and is HTTP 1.1 compliant. It uses the fact that in HTTP 1.1 the TCP connection can be persistent (keep-alive). The server keeps a "device factory" during all the TCP connection life cycle. This prevents from re-importing and re-introspecting each time Tango devices and avoids unwanted overhead with the Tango database and devices. When the connection is closed, TangoGraphQL releases all allocated resource associated to the connection.

CONFIGURATION

TangoGraphQL server can be configured using Tango properties:

- *Port*: HTTP server port (default is 8000)
- *IdleTimeout*: Timeout before closing a HTTP connection (default is 60s)
- *Authentication*: None or Basic
- *Users*: List of allowed users, (List of userName: sha1Password)
- *AllowMutation*: Allow mutation mode (ALWAYS, AUTHENTICATED, NEVER).
- *exportGraphiQL*: True to enable GraphiQL (available via http(s)://host:port/graphiql/)
- *httpsEnable*: Enable HTTPS server (need certificate)
- *httpsCertificate*: Link to the certificate file cert and its privKey file. (The server needs read access to this 2 files)
- *ResolveClientIP*: Resolve names in (connected) clients attribute

AUTHENTICATION AND ACCESS CONTROL

TangoGraphQL uses Open SSL for https with basic HTTP authentication scheme. Single SignOn authentication scheme is currently under development. Tango Access Control cannot be implemented using Tango 9 as it is not possible to set a username for a Tango client thread. This will require new feature of Tango 10.

CONCLUSION AND DISCUSSIONS

After an investigation done at the ESRF of the 4 present solutions (including the one presented in this paper) for a Tango backend, GraphQL appears to be the best alternative.

- Tango Rest API (java REST API on top of Apache) [7]
- RestDS (& RestDS2) (C++ REST API on top of boost HTTP server) [8]
- MaxIV TangoQL (Python GraphQL on top of AIOHTTP & Graphene) [9]

In this paper, a C++ implementation of a Tango GraphQL schema is presented and express the minimum requirements we have at the ESRF in order to make the transition from our Java applications to Web applications possible. The GraphQL schema is not fixed; it may evolve over time, using deprecations, for example. The schema proposed here is still open to discussion. The goal is for the community to converge on a common GraphQL specification which can be supported by multiple implementations.

REFERENCES

[1] European Synchrotron Radiation Facility, `https://www.esrf.fr`

[2] Java ATK framework, `https://gitlab.com/tango-controls/atk`

[3] GraphQL Foundation, `https://graphql.org/`

[4] Tango C++ GraphQL GitLab, `https://gitlab.com/tango-controls/TangoGraphQL`

[5] GraphQL over WebSocket Protocol, `https://github.com/enisdenjo/graphql-ws`

[6] Florian Loitsch, "Printing floating-point numbers quickly and accurately with integers", in Proc. 31st ACM SIGPLAN Conference on Programming Language Design and Implementation, June 2010, pp. 233-243.

[7] Java TANGO REST API, `https://tango-rest-api.readthedocs.io/en/latest`

[8] Sedykh, G.S *et al*., "The C++ TANGO REST API Implementation". Phys. Part. Nuclei Lett. 17, 604–606 (2020). `https://doi.org/10.1134/S1547477120040391`

[9] TangoGQL, MaxIV, `https://developer.skao.int/projects/web-maxiv-tangogql/en/latest/`