Journal of Physics: Conference Series **396** (2012) 052019 doi:10.1088/1742-6596/396/5/052019

DIRAC RESTful API

A. Casajus Ramo[1], R. Graciani Diaz[1], A. Tsaregorodtsev[2]

[1] ICC, University of Barcelona
[2] CPPM Marseille

E-mail: adria@ecm.ub.es

Abstract.

The DIRAC framework for distributed computing has been designed as a flexible and modular solution that can be adapted to the requirements of any community. Users interact with DIRAC via command line, using the web portal or accessing resources via the DIRAC python API. The current DIRAC API requires users to use a python version valid for DIRAC.

Some communities have developed their own software solutions for handling their specific workload, and would like to use DIRAC as their back-end to access distributed computing resources easily. Many of these solutions are not coded in python or depend on a specific python version. To solve this gap DIRAC provides a new language agnostic API that any software solution can use. This new API has been designed following the RESTful principles. Any language with libraries to issue standard HTTP queries may use it. GSI proxies can still be used to authenticate against the API services. However GSI proxies are not a widely adopted standard. The new DIRAC API also allows clients to use OAuth for delegating the user credentials to a third party solution. These delegated credentials allow the third party software to query to DIRAC on behalf of the users.

This new API will further expand the possibilities communities have to integrate DIRAC into their distributed computing models.

1. Introduction

The DIRAC[18] project started in 2002 as a software tool to manage LHCb Monte Carlo simulation jobs in an efficient manner. DIRAC functionality has been extended since then to include data management, user analysis, workflows... to manage all LHCb's computing activities. Other communities started to use and adapt DIRAC to fit their own needs.

In the last years there has been an increasing attention to DIRAC. DIRAC is now used by several user communities as their grid management tool. As more and more people start interacting with DIRAC it has been clear that the DIRAC portal cannot accommodate all use cases. Each community has their own needs and problems that need solving.

Communities can extend the DIRAC web portal[4] to meet their needs. But some of them started projects on their own before deciding on using DIRAC or are using other already existing products. Those communities would like to connect their own projects with DIRAC. DIRAC is almost entirely coded in Python[17] and provides an API for third party applications to interact with it. Many of these solutions are not coded in Python. Migrating these projects to Python is not feasible in most cases.

Up until now, applications that wanted to interact with DIRAC that could not use the standard DIRAC API had to use the command line tools and parse the result. This solution

Codification	Requires schema	Human readable	Native types
XML	Yes	Yes	No
YAML	Yes	Yes	No
JSON	No	Yes	Yes
BSON	No	No	Yes
MessagePack	Yes	No	Yes
Protocol Buffers	Yes	No	Yes
Thrift	Yes	No	Yes

Table 1. Comparison of serializers

has proven troublesome time and again. Every time a command line tool changes the output format or a functionality is added or modified the applications have to be patched. Even if this approach can be used as a proof of concept, a more robust solution is needed.

This paper is organized as follows. Section 2 introduces the possibilities on how to make a language agnostic API. Section 3 describes the API implemented and what components are necessary to provide the required functionality. Section 4 summarizes this paper and explores future work.

2. Language agnostic APIs

A API is required to allow third party applications that are not written in the native language of a given software. In the case of DIRAC, its native language is python. To achieve the widest possible coverage APIs should be language agnostic. A language agnostic has to use standard mechanisms for:

- Codification of requests and responses
- Credential delegation
- Query protocol

At the same time, it is desirable that the chosen solution can be easily understood, maintained and debugged.

2.1. Request codification

There are plenty of serializers that are available for lots of languages. A few examples are *XML*[5], *YAML*[22]. *JSON*[11], *BSON*[2], *MessagePack*[13], *Protocol Buffers*[16], *Thrift*[1] and many more. A comparison of these serializers is included in table 1. *XML* and *YAML* were discarded because they do not have any knowledge of native types. The serialization has to be defined by the user to define what each object is. On top of that, the serialized data *XML* produces is overloaded with repetitive patterns that make the data too big. *MessagePack*, *ProtocolBuffers* and *Thrift* where discarded because they need a schema to be compiled before using the serializer. Defining a schema for the serializer can help maintain the interfaces once an API is mature. But it adds a lot of complexity to it. *BSON* is a binary form of *JSON*. Although *BSON* is an open specification there seems to be only one company behind it and it is not widely used. *JSON* is a more mature standard than *BSON* and has been used for a long time.

2.2. Credentials delegation

Whenever an application uses the API to interact with DIRAC, the user has to have granted privileges to the application previously as shown in figure 1. Without credentials delegation DIRAC would not know what the application is allowed to access.

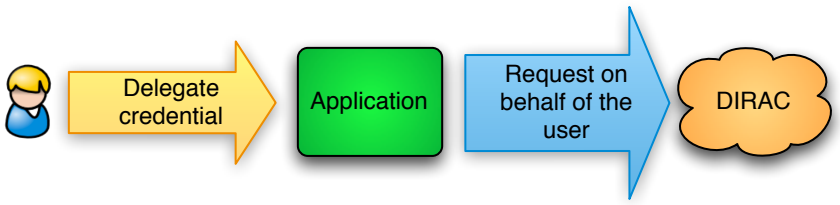

Figure 1. Credentials delegation

DIRAC uses x509 certificates [9] and grid proxies [3] to authenticate connections and can make delegate to the different components that make use of them. Grid proxies are not an RFC standard. There is an ongoing effort to update DIRAC to understand X509 proxies [19] in the near future. But using proxies is not easy to do through a web page. Although it would be possible to create a grid proxy in the browser if it had access to the user credentials, browsers are not able to retrieve the user key from the certificate store.

The usual solution for this problem has been Kerberos[12]. Kerberos is a network authentication protocol. Kerberos uses strong cryptography so that clients and servers can prove their identity across an insecure network connection. Once their identity has been proved, they can also encrypt all their communication. Although Kerberos would solve the problem, it is quite complicated and difficult to understand. On top of that Kerberos has multiple authentication plug-ins that make it difficult to be properly configured.

Ideally the delegation mechanism should be simple enough that it can be implemented from scratch with the basic modules available to any programming language. This requirement implies that it is simple enough for a wide user community to understand it and therefore many people will be able to contribute to spot possible flaws in the mechanism. This follows the open source approach to improve the quality of the software.

OAuth[8] is an alternative that lately has been gathering strength from the web community. Multiple web pages such as Twitter[15], Google Apps[20], and Flickr[21] support it. It is simple enough so that anyone can understand how it works. There is a second version of the protocol at the *OAuth* web page[14] that further simplifies the protocol but it is still a work in progress.

OAuth is a mechanism to grant privileges to tokens that are only known to certain parts. It starts with the application requesting a *request token* to DIRAC. In *OAuth* language, DIRAC is called provider and the application is called client. To get the *request token* the client must use his *client token* that has been granted before. That means that only known clients can request tokens, and that each token is tied to a single client. Once the provider grants the *request token* to the client, the client gives the token to the user so it can grant privileges to it. The user logs-in to the provider and grants the required privileges to the token. As soon as the token has the privileges granted, the user notifies the client that the *request token* has the requested privileges. Finally the client exchanges the *request token* for an *access token* with the provider. This last step is to ensure that not even the user know the tokens the client is using. An example flow is shown in figure 2.

Using this mechanism the client will never know the password or require access to the user certificate. On top of that, if a user does not want an application to access DIRAC anymore, it just needs to revoke the privileges to the token it granted and the application will not have access any more until a new token is validated.

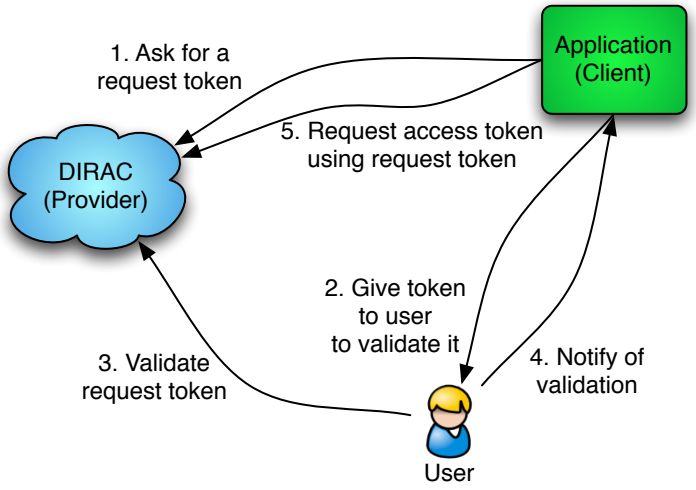

Figure 2. OAuth flow

2.3. Query protocol

There are lots of standard protocols that allow querying a remote server. Those using *XML* for their message format have been discarded due to the length of the messsages and the cost of processing *XML* serialized data. Not many remain standard protocols that are completely language agnostic and easy to use remain after discarding those that use *XML*. *REST* over *HTML*[6] architecture has been chosen as the way to expose DIRAC's language agnostic API. By using a *HTML* as transport layer it allows an easy integration with any language. *REST* has several key points that make it a perfect fit:

- It is so simple so that even if there are not any bindings for a specific language, one can be coded easily.
- By being stateless, *REST* allows for a scalable deployment. If the API server is overloaded, another can be set up thus increasing capacity is really easy.
- Allows cache management. Clients can know if a result if cacheable and for how long.

3. RESTful API

To provide the new language agnostic API several components have been added to DIRAC as shown in figure 3. A new server has been created to serve the *REST* requests. This server can also generate *OAuth* tokens. A credential store has also been added to store the tokens and privileges granted.

3.1. Service architecture

Any client (any community application) that wants to use the *REST* API requires an *OAuth* client token. Applications can request tokens to the *REST* server. Once a *request token* has been generated, the application has to redirect the user to a DIRAC web portal. Users can validate the *request token* in the web portal. By accessing the web portal users are authenticating themselves with their certificate so the web portal will know who they are automatically. Permissions are also be granted on a per group basis following the same authorization schema that DIRAC already provides. When accessing the web portal, users automatically have an active group. They can change group using the standard DIRAC web portal mechanism before granting privileges to the token.

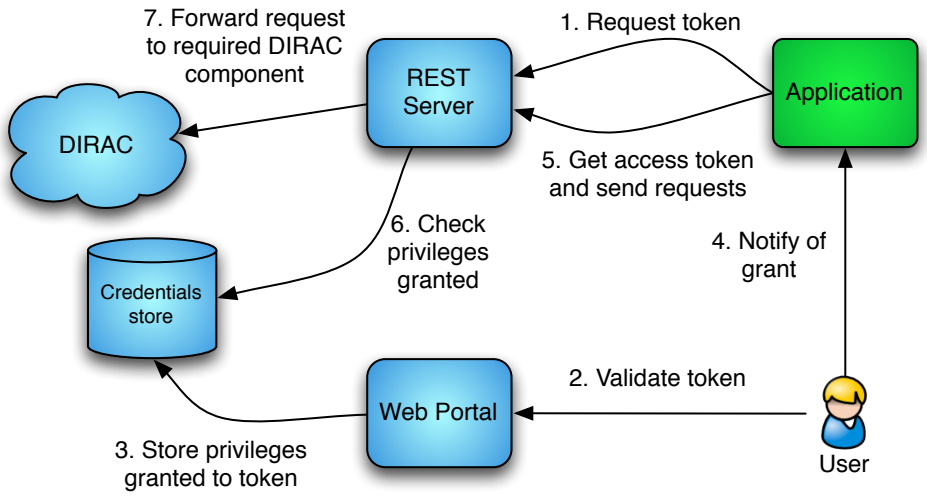

Figure 3. RESTful DIRAC architecture

Once the user has authenticated to the web portal and has granted privileges to the token. The web portal will redirect the user back to the application. The application can then finally exchange the *request token* for the final *access token* that will be used to sign the requests.

This flow is really useful if the client is a web application. In case the application is a graphical non-web one or has command-line interface there is an alternative for validating the token. DIRAC also provides a command-line tool to validate tokens. Users only need to generate a grid proxy using either DIRAC or any grid proxy generation tool and execute the command line script to validate the token.

In any case the credentials are stored in a new service. This service acts as a credential store. All the tokens, identities and privileges are stored there. This is a critical component that should be installed in a host where the access rules are strict.

3.2. API requests

Any request has to be signed with an *OAuth* token. The *REST* server will know on behalf on which user the request has to be executed based on the token that signs the request. Each token is linked to one user, one set of privileges and one client. The Credential store is queried by the *REST* server to retrieve the information for each new token. That information can be queried by the *REST* server until the cache time expires or the Credential store informs the *REST* server that those privileges have been revoked.

Each request has to follow the *REST* style. *REST* defines several guidelines on how to create a uniform interface:

- Each object has to have a unique and persistent identity. In web-based *REST* systems objects can be identified by *URIs*. Objects have to be conceptually separate from the representation that is sent to the client. For example, servers do not send the database they have but a *JSON* representation of the relevant records.
- When a client holds a represention of a object given by a server. It has enough information to modify or delete it, if it has permission to do so. That means that clients do not need to know the internal representation of a object.
- Each response has enough information to describe how to process it. Responses should also indicate their cacheability.

For instance, to retrieve information about a single job, an application can issue a *GET* request to the job URI. An example of the URI could be:

`http://restserver.domain.net/api/job/jobid`

The server will return a *JSON* encoded structure containing all info related to the job. The headers of the response will contain also the caching information for the response. If the application wants to kill a job, it can issue a *DELETE* request to the job URI. The server will reply whether it has permission or not... This mechanism can be extended for any object that DIRAC handles.

4. Summary

DIRAC has been extended to provide the previously described language agnostic API. This new API follows the *REST* style over *HTML* using *JSON* as the serialization format. *OAuth* is used as the credentials delegation mechanism to the applications. All three technologies are widely used and have bindings already made for most of today's modern languages.

Although the current implementation is at the prototype stage, several applications have shown interest in testing it such as *gUSE/ WS-PGRADE*[7] and *InSilicoLab*[10]. Once the API has been tested, credentials delegation using *OAuth* version 2 will be added to the service.

By providing this new API DIRAC can now be interfaced to any component written in most of today's modern languages.

Acknowledgments

The presented work has been financed by *Comisión Interministerial de Ciencia y Tecnología (CICYT)* (project FPA2010-21885-C02-01 and CPAN CSD2007-00042 from *Programa Consolider-Ingenio 2010*), and by *Generalitat de Catalunya* (AGAUR 2009SGR01268).

References

[1] *Apache Thrift*. URL: `http://thrift.apache.org/`.

[2] *Binary JSON*. URL: `http://bsonspec.org/`.

[3] A. Casajus and R. Graciani. "DIRAC Distributed Secure Framework". In: *Computing in High-Energy Physics and Nuclear Physics 2009*. 2009.

[4] A. Casajus and M. Sapunov. "DIRAC Secure Web User Interface". In: *Computing in High-Energy Physics and Nuclear Physics 2009*. 2009.

[5] *Extensible Markup Language*. URL: `http://en.wikipedia.org/wiki/XML`.

[6] R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616 (Draft Standard). Updated by RFCs 2817, 5785, 6266, 6585. Internet Engineering Task Force, June 1999. URL: `http://www.ietf.org/rfc/rfc2616.txt`.

[7] *grid User Support Environment*. URL: `http://www.guse.hu/`.

[8] E. Hammer-Lahav. *The OAuth 1.0 Protocol*. RFC 5849 (Informational). Internet Engineering Task Force, Apr. 2010. URL: `http://www.ietf.org/rfc/rfc5849.txt`.

[9] R. Housley et al. *[RFC3280] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. United States, 2002.

[10] *InSilicoLab*. URL: `http://insilicolab.grid.cyfronet.pl/`.

[11] *JavaScript Object Notation*. URL: `http://www.json.org/`.

[12] *Kerberos: The Network Authentication Protocol*. URL: `http://web.mit.edu/kerberos/`.

[13] *MessagePack*. URL: `http://msgpack.org/`.

[14] *OAuth*. URL: `http://oauth.net/`.

[15] *OAuth for Twitter*. URL: `https://dev.twitter.com/docs/auth/oauth`.

[16] *Protocol Buffers*. URL: `http://code.google.com/p/protobuf/`.

[17] *Python programming language*. URL: `http://www.python.org/`.

[18] A. Tsaregorodtsev et al. "DIRAC: A community grid solution". In: *Computing in High-Energy Physics and Nuclear Physics 2007*. 2008.

[19] S. Tuecke et al. *[RFC3820] Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile*. United States, 2004.

[20] *Using OAuth 2.0 to Access Google APIs*. URL: `https://developers.google.com/accounts/docs/OAuth2`.

[21] *Using OAuth with Flickr*. URL: `http://www.flickr.com/services/api/auth.oauth.html`.

[22] *YAML Ain't Markup Language*. URL: `http://yaml.org/`.