INTERNET DRAFT
Document: draft-dejong-remotestorage-01

Intended Status: Proposed Standard
Expires: 10 December 2013

Michiel B. de Jong
(independent)
F. Kooman
SURFnet

8 June 2013

remoteStorage

Abstract

This draft describes a protocol by which client-side applications, running inside a web browser, can communicate with a data storage server that is hosted on a different domain name. This way, the provider of a web application need not also play the role of data storage provider. The protocol supports storing, retrieving, and removing individual documents, as well as listing the contents of an individual directory, and access control is based on bearer tokens.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 10 December 2013.

Copyright Notice

Table of Contents

1. Introduction

Many services for data storage are available over the internet. This specification describes a vendor-independent interface for such services. It is based on https, CORS and bearer tokens. The metaphor for addressing data on the storage is that of folders containing documents and subfolders. The actions the interface exposes are:

* GET a folder: retrieve the names and current versions of the documents and subfolders currently contained by the folder

* GET a document: retrieve its content type, current version, and contents

* PUT a document: store a new version, its content type, and contents, conditional on the current version

* DELETE a document: remove it from the storage, conditional on the current version

The exact details of these four actions are described in this specification.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [WORDS].

"SHOULD" and "SHOULD NOT" are appropriate when valid exceptions to a general requirement are known to exist or appear to exist, and it is infeasible or impractical to enumerate all of them. However, they should not be interpreted as permitting implementors to fail to implement the general requirement when such failure would result in interoperability failure.

3. Storage model

The server stores data in nodes that form a tree structure. Internal nodes are called 'folders' and leaf nodes are called 'documents'. For a folder, the server stores references to nodes contained in the folder, and it should be able to produce a list of them, with for each contained item:

* item name
* item type (folder or document)
* current version

For a document, the server stores, and should be able to produce:

* content type
* content
* current version

4. Requests

Client-to-server requests SHOULD be made over https [HTTPS]. The root folder of the storage tree is represented by the URL <storage_root> '/'. Subsequently, if <parent_folder> is the URL of a folder, then the URL of an item contained in it is <parent_folder> <document_name> for a document, or <parent_folder> <folder_name> '/' for a folder. Item names MAY contain a-z, A-Z, 0-9, %, ., -, _, and MUST NOT have zero length.

A successful GET request to a folder SHOULD be responded to with a JSON document (content type 'application/json'), representing a map in which contained documents appear as entries <item_name> to <current_version>, and contained folders appear as entries <item_name> '/' to <current_version>, for instance:

```
{
  "abc": "DEADBEEFDEADBEEFDEADBEEF",
  "def/": "1337ABCD1337ABCD1337ABCD"
}
```

Empty folders are treated as non-existing, and therefore GET requests to them SHOULD be responded to with a 404 response, and an empty folder MUST NOT be listed as an item in its parent folder. Also, folders SHOULD be created silently, as necessary to contain newly added items. This way, PUT and DELETE requests only need to be made to documents, and folder management becomes an implicit result.

A successful GET request to a document SHOULD be responded to with the full document contents in the body, the document's content type in a 'Content-Type' header, and the document's current version in an 'ETag' header.

A successful PUT request to a document MUST result in:

* the request body being stored as the document's new content,
* parent and further ancestor folders being silently created as necessary, with the document (name and version) being added to its parent folder, and each folder added to its subsequent parent,
* the value of its Content-Type header being stored as the document's new content type,
* its version being updated, as well as that of its parent folder and further ancestor folders.

The response MUST contain an ETag header, with the document's new version (for instance a hash of its contents) as its value.

A successful DELETE request to a document MUST result in:

* the deletion of that document from the storage, and from its parent folder,
* silent deletion of the parent folder if it is left empty by this, and so on for further ancestor folders,
* the version of its parent folder being updated, as well as that of further ancestor folders.

The response MUST contain an ETag header, with the document's version that was deleted as its value.

A successful OPTIONS request SHOULD be responded to as described in the CORS section below.

5. Response codes

The following responses SHOULD be given in the indicated cases, in order of preference, and SHOULD be recognized by the client:

* 500 if an internal server error occurs,
* 420 if the client makes too frequent requests or is suspected of malicious activity,
* 400 for all malformed requests (e.g. foreign characters in the path or unrecognized http verb, etcetera), as well as for all PUT and DELETE requests to folders,
* 401 for all requests that don't have a bearer token with sufficient permissions,
* 404 for all DELETE and GET requests to nodes that do not exist on the storage,
* 412 for a conditional request whose pre-condition fails (see "Versioning" below),
* 507 in case the user's account is over its storage quota,
* 200 for all successful requests, including PUT and DELETE.

Clients SHOULD also handle the case where a response takes too long to arrive, or where no response is received at all.

6. Versioning

All successful requests MUST return an 'ETag' header [HTTP] with, in the case of GET, the current version, in the case of PUT, the new version, and in case of DELETE, the version that was deleted. PUT and DELETE requests MAY have an 'If-Match' request header [HTTP], and MUST fail with a 412 response code if that doesn't match the document's current version. GET requests MAY have an 'If-None-Match' header [HTTP], and SHOULD be responded to with a 412 response if that includes the document or folder's current version.

A PUT request MAY have an 'If-None-Match:*' header [HTTP], in which case it MUST fail with a 412 response code if the document already exists.

A provider MAY offer its users a way to roll back to a previous version of the storage contents, but this specification does not define any such rollback functionality.

7. CORS headers

All responses MUST carry CORS headers [CORS]. The server MUST also reply to OPTIONS requests as per CORS. For GET requests, a wildcard origin MAY be returned, but for PUT and DELETE requests, the response MUST echo back the Origin header sent by the client.

8. Session description

The information that a client needs to receive in order to be able to connect to a server SHOULD reach the client as described in the 'bearer token issuance' sections below. It consists of:

* <storage_root>, consisting of 'https://' followed by a server host, and optionally a server port and a path prefix as per [IRI]. Examples:
  * 'https://example.com' (host only)
  * 'https://example.com:8080' (host and port)
  * 'https://example.com/path/to/storage' (host, port and path prefix; note there is no trailing slash)
* <access_token> as per [OAUTH]. The token SHOULD be hard to guess and SHOULD NOT be reused from one client to another. It can however be reused in subsequent interactions with the same client, as long as that client is still trusted. Example:
  * 'ofb24f1ac3973e70j6vts19qr9v2eei'
* <storage_api>, always 'draft-dejong-remotestorage-01' for this version of the specification.

The client can make its requests using https with CORS and bearer tokens, to the URL that is the concatenation of <storage_root> with '/' plus one or more <folder> '/' strings indicating a path in the folder tree, followed by zero or one <document> strings, indicating a document. For example, if <storage_root> is "https://storage.example.com/bob", then to retrieve the folder contents of the /public/documents/ folder, or to retrieve a 'draft.txt' document from that folder, the client would make requests to, respectively:

* https://storage.example.com/bob/public/documents/
* https://storage.example.com/bob/public/documents/draft.txt

9. Bearer tokens and access control

A bearer token represents one or more access scopes. These access scopes are represented as strings of the form <module> <level>, where the <module> string SHOULD be lower-case alphanumerical, other than the reserved word 'public', and <level> can be ':r' or ':rw'. The access the bearer token gives is the sum of its access scopes, with each access scope representing the following permissions:

'root:rw') any request,

'root:r') any GET request,

<module> ':rw') any requests to paths that start with '/' <module> '/' or '/public/' <module> '/',

<module> ':r') any GET requests to paths that start with '/' <module> '/' or '/public/' <module> '/',

As a special exceptions, GET requests to a document (but not a folder) whose path starts with '/public/' are always allowed. They, as well as OPTIONS requests, can be made without a bearer token. All other requests should present a bearer token with sufficient access scope, using a header of the following form:

```
Authorization: Bearer <access_token>
```

10. Application-first bearer token issuance

To make a remoteStorage server available as 'the remoteStorage of <user> at <host>', exactly one link of the following format SHOULD be added to the webfinger record [WEBFINGER] of <user> at <host>:

```
{
  href: <storage_root>,
  rel: "remotestorage",
  type: <storage_api>,
  properties: {
    "http://tools.ietf.org/html/rfc6749#section-4.2": <auth-dialog>
  }
}
```

Here <storage_root> and <storage_api> are as per "Session description" above, and <auth-dialog> SHOULD be a URL where an OAuth2 implicit-grant flow dialog [OAUTH] is be presented, so the user can supply her credentials (how, is out of scope), and allow or reject a request by the connecting application to obtain a bearer token for a certain list of access scopes.

The server SHOULD NOT expire bearer tokens unless they are revoked, and MAY require the user to register applications as OAuth clients before first use; if no client registration is required, then the server MAY ignore the client_id parameter in favour of relying on the redirect_uri parameter for client identification.

11. Storage-first bearer token issuance

The provider MAY also present a dashboard to the user, where she

has some way to add open web app manifests [MANIFEST]. Adding a manifest to the dashboard is considered equivalent to clicking 'accept' in the dialog of the application-first flow. Removing one is considered equivalent to revoking its access token.

As an equivalent to OAuth's 'scope' parameter, a 'remotestorage' field SHOULD be present in the root of such an application manifest document, as a JSON array of strings, each string being one access scope of the form <module> <level>.

When the user gestures she wants to use a certain application whose manifest is present on the dashboard, the dashboard SHOULD redirect to the application or open it in a new window. To mimic coming back from the OAuth dialog, it MAY add 'access_token' and 'scope' parameters to the URL fragment.

Regardless of whether 'access_token' and 'scope' are specified, it SHOULD add a 'remotestorage' parameter to the URL fragment, with a value of the form <user> '@' <host>. When the application detects this parameter, it SHOULD resolve the webfinger record for <user> at <host> and extract the <storage_root> and <storage_api> information.

If no access_token was given, then the application SHOULD also extract the <auth_endpoint> information from webfinger, and continue as per application-first bearer token issuance.

Note that whereas a remoteStorage server SHOULD offer support of the application-first flow with webfinger and OAuth, it MAY choose not to support the storage-first flow, provided that users will easily remember their <user> '@' <host> webfinger address at that provider. Applications SHOULD, however, support both flows, which means checking the URL for a 'remoteStorage' parameter, but giving the user a way to specify her webfinger address if there is none.

If a server provides an application manifest dashboard, then it SHOULD merge the list of applications there with the list of issued access tokens as specified by OAuth into one list. Also, the interface for revoking an access token as specified by OAuth SHOULD coincide with removing an application from the dashboard.

12. Security Considerations

To prevent man-in-the-middle attacks, the use of https instead of http is important for both the interface itself and all end-points involved in webfinger, OAuth, and (if present) the storage-first application launch dashboard.

A malicious party could link to an application, but specifying a remoteStorage user address that it controls, thus tricking the user into using a trusted application to send sensitive data to the wrong remoteStorage server. To mitigate this, applications SHOULD clearly display to which remoteStorage server they are sending the user's data.

Applications could request scopes that the user did not intend to give access to. The user SHOULD always be prompted to carefully review which scopes an application is requesting.

An application may upload malicious html pages and then trick the user into visiting them, or upload malicious client-side scripts, that take advantage of being hosted on the user's domain name. The origin on which the remoteStorage server has its interface SHOULD therefore NOT be used for anything else, and the user SHOULD be warned not to visit any web pages on that origin. In particular, the OAuth dialog and launch dashboard or token revokation interface SHOULD be on a different origin than the remoteStorage interface.

Where the use of bearer tokens is impractical, a user may choose to store documents on hard-to-guess URLs whose path after <storage_root> starts with '/public/', while sharing this URL only with the intended audience. That way, only parties who know the

document's hard-to-guess URL, can access it. The server SHOULD therefore make an effort to detect and stop brute-force attacks that attempt to guess the location of such documents.
The server SHOULD also detect and stop denial-of-service attacks that aim to overwhelm its interface with too much traffic.

13. IANA Considerations

This document registers the 'remotestorage' link relation.

14. Acknowledgements

The authors would like to thank everybody who contributed to the development of this protocol, including Kenny Bentley, Javier Diaz, Daniel Groeber, Bjarni Runar, Jan Wildeboer, Charles Schultz, Peter Svensson, Valer Mischenko, Michiel Leenaars, Jan-Christoph Borchardt, Garret Alfert, Sebastian Kippe, Max Wiehle, Melvin Carvalho, Martin Stadler, Geoffroy Couprie, Niklas Cathor, Marco Stahl, James Coglan, Ken Eucker, Daniel Brolund, elf Pavlik, Nick Jennings, Markus Sabadello, and Steven te Brinke, among many others.

15. References

15.1. Normative References

[WORDS]
Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[IRI]
Duerst, M., "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005.

[WEBFINGER]
Jones, Paul E., Salguerio, Gonzalo, and Smarr, Joseph, "WebFinger", draft-ietf-appsawg-webfinger-14, Work in Progress

[OAUTH]
"Section 4.2: Implicit Grant", in: Hardt, D. (ed), "The OAuth 2.0 Authorization Framework", RFC6749, October 2012.

15.2. Informative References

[HTTPS]
Rescorla, E., "HTTP Over TLS", RFC2818, May 2000.

[HTTP]
Fielding et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC2616, June 1999.

[CORS]
van Kesteren, Anne (ed), "Cross-Origin Resource Sharing -- W3C Working Draft 3 April 2012", http://www.w3.org/TR/2012/WD-cors-20120403/CORS, April 2012.

[MANIFEST]
Mozilla Developer Network (ed), "App manifest -- Revision 330541", https://developer.mozilla.org/en-US/docs/Apps/Manifest$revision/330541, November 2012.

16. Authors' addresses

Michiel B. de Jong
(independent)

Email: michiel@michielbdejong.com

F. Kooman
SURFnet bv
Postbus 19035
3501 DA Utrecht
The Netherlands

Email: Francois.Kooman@surfnet.nl