Workgroup:
Content Delivery Networks Interconnection
Published: 24 April 2024
Intended Status: Standards Track
Expires: 26 October 2024
Authors: A. Siloniz G. Goldstein
Telefonica Lumen Technologies

CDNI Edge Control Metadata

Abstract

This specification defines configuration metadata objects related to controlling edge access to resources via content delivery networks (CDNs) and Open Caching systems. Configuring Cross-Origin Resource Sharing (CORS) access rules and the dynamic generation of CORS headers is a key feature of typical configurations, as are the ability to define response body compression rules and client connection timeouts.

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 https://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 26 October 2024.

Table of Contents

1. Introduction

CDNs typically require a set of configuration metadata to provide directives for the processing of responses downstream (at the edge and in the user agent). This document specifies GenericMetadata objects to meet those requirements, defining edge processing rules such as Cross-Origin Resource Sharing (CORS) handling, response compressions, and client connection failures.

2. Requirements

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 [RFC2119].

3. MI.CrossoriginPolicy

Delegation of traffic between CDNs over an Open Caching node (OCN) based on Hypertext Transfer Protocol (HTTP) redirection changes the domain name in the client requests. This represents a cross-origin request that must be managed appropriately using CORS headers in the responses.

The dynamic generation of CORS headers is typical in modern HTTP request processing and avoids CORS validation forwarded to the CDN origin servers, particularly with the preflight OPTIONS requests. The CDN Interconnection (CDNI) metadata model requires extensions to specify how a CDN or Open Caching node should generate and evaluate these headers.

Required capabilities:

- Set a default value for CORS response headers independent of the origin request header value.
- Match the origin request header with a list of valid values to return or not return the CORS response headers.
- Set a list of custom headers that can be exposed to the client (expose headers).
- Support preflight requests using the OPTIONS method, including custom header validation, expose headers, and methods.
- Support credentials validation within CORS.

Simple CORS requests are those where both HTTP method and headers in the request don't require a preflight request. The user agent (UA) request can include an Origin header set to a URL domain of the webpage from which the User Agent made it. Depending on the metadata configuration, the logic to apply by the Open Caching node is:

- Validation of the origin header - Metadata can include a list of valid domains to validate the request origin header. If it does not match, the CORS header MUST NOT be included in the response.
- WIldcard usage - Depending on the configuration, the resultant CORS header to include in the response will be the same as the request origin header, or a wildcard.
- If no validation of request is included in the origin header, set a default value for CORS response headers independent of the origin request header value.

When a UA makes a request that includes a method or headers that require a CORS preflight request, the UA will first use the OPTIONS method to the resource, including the origin header. If CORS is enabled and the request passes the origin validation, the dCDN SHOULD respond with the set of headers that indicate what is permitted for that resource, including one or more of the following:

- Allowed methods
- Allowed credentials
- Allowed request headers
- Maximum age that the OPTIONS request is valid
- Headers that can be exposed to the client

If the returned headers allow the UA browser to provide access to the resource, the UA will then request the resource with the proper HTTP method (GET, PUT, POST, etc)

When an upstream CDN (uCDN) configures any of those advanced parameters, it is requesting the dCDN to generate synthetic responses to OPTIONS requests. Therefore, no conditional request is performed to the uCDN origin, and typically to inject the proper CORS headers in the response to the resource access. The uCDN SHOULD configure these values taking that into account. If some of the advanced parameters are empty, the dCDN would not send the corresponding header into its response to the UA request.

In cases where the uCDN only configures the MI.AccessControlAllowOrigin subobject, the dCDN will not generate synthetic responses to OPTIONS requests. Instead, the dCDN will forward tothe uCDN every OPTIONS request to obtain the response.

MI.CrossoriginPolicy is a GenericMetadata object that allows configuring dynamically generated CORS headers.

Property: allow-origin

*Description: Validation of simple CORS requests.

*Type: Object MI.AccessControlAllowOrigin (Section 3.1)

*Mandatory-to-Specify: Yes

Property: expose-headers

*Description: A list of values the dCDN will include in the Access-Control-Expose-Headers response header to a preflight request.

*Type: Array of strings

*Mandatory-to-Specify: No. If not specified, the default behavior is to not add the header in the response

Property: allow-methods

*Description: A list of values the OCN will include in the Access-Control-Allow-Methods response header to a preflight request.

*Type: Array of strings

*Mandatory-to-Specify: No. If not specified, the default behavior is to not add the header in the response

Property: allow-headers

  *Description: A list of values the OCN will include in the Access-Control-Allow-Headers response header to a preflight request.

  *Type: Array of strings

  *Mandatory-to-Specify: No. If not specified, the default behavior is to not add the header in the response

Property: allow-credentials

  *Description: The value the OCN will include in the Access-Control-Allow-Credentials response header to a preflight request.

  *Type: Boolean

  *Mandatory-to-Specify: No. If not specified, the default behavior is to not add the header in the response

Property: max-age

  *Description: The value the OCN will include in the Access-Control-Max-Age response header to a preflight request.

  *Type: Integer

  *Mandatory-to-Specify: No. If not specified, the default behavior is to not add the header in the response

Property: no-origin-response-headers

  *Description: In the case of a request that has no Origin field, return this set of headers with the response.

  *Type: Array of MI.HTTPHeader objects, defined in Processing Stages Metadata [SVTA2032].

  *Mandatory-to-Specify: No. If not specified, the default behavior is to not add any CORS response headers

Property: preflight-only

  *Description: Setting this flag to "True" the dCDN will only generate synthetic responses to OPTIONS requests with proper CORS response headers

  *Type: Boolean

*Mandatory-to-Specify: No. The default is "False", so CORS response headers logic will be injected for all HTTP methods

3.1. MI.AccessControlAllowOrigin

The MI.AccessControlAllowOrigin object has the following properties:

Property: allow-list

*Description: List of valid expressions that will be used to match the request Origin header. The Origin header is an HTTP extension. Its value is a version of the Referer header that does not reveal a path in some specific requests and used for cross-origin requests. Permitted values for the Origin header are of the form schema://host[:port] as defined in Sections 3.1, 3.2.2 and 3.2.3 of [RFC3986]

Type: Array of Strings. Each string represents a pattern for matching against the Origin header. The pattern can contain the wildcards "" and "?", where "*" matches any sequence of pchar [RFC3986] or "/" characters (including the empty string) and "?" matches exactly one pchar character. The three literals "$", "*", and "?" MUST be escaped as "$$", "$*", and "$?" (where "$" is the designated escape character). All other characters are treated as literals

*Mandatory-to-Specify: Yes

Property: wildcard-return

*Description: If "True", the OCN will include a wildcard (*) in the Access-Control-Allow-Origin response header. If "False", the dcdn MUST reflect the request origin header in the Access-Control-Allow-Origin response header.

*Type: Boolean

*Mandatory-to-Specify: Yes

3.2. Examples

The examples below demonstrate how to configure response headers dynamically for CORS validation.

The following is an example of a simple CORS validation configuration:

```
{
  "generic-metadata-type": "MI.CrossoriginPolicy",
  "generic-metadata-value": {
    "allow-origin": {
      "allow-list": [
        {
          "pattern": "*"
        }
      ],
      "wildcard-return": true
    }
  }
}
```

Figure 1

The following is an example of validation of a preflight request when some of the headers included in the subsequent object request are not included in the CORS specification safelist:

```
{
  "generic-metadata-type": "MI.CrossoriginPolicy",
  "generic-metadata-value": {
    "allow-origin": {
      "allow-list": [
        {
          "pattern": "*://sourcepage.example.com"
        },
        "wildcard-return": false
      },
      "allow-methods": [ "GET", "POST" ],
      "allow-credentials": true,
      "allow-headers": [ "X-PINGOTHER", "Content-Type" ],
      "expose-headers": [ "X-User", "Authorization" ],
      "max-age": 3600,
      "no-origin-response-headers": false,
      "preflight-only": false
    }
  }
}
```

Figure 2

4. MI.AllowCompress

Downstream CDNs often have the ability to compress HTTP response bodies in cases where the client has declared that it can accept compressed responses (via an Accept-Encoding header), but the source/origin has returned an uncompressed response.

The specific compression algorithm used by the dCDN is negotiated by the client's Accept-Encoding header according to [RFC9110] (including "q=" preferences) and the compression capabilities available on the dCDN.

In addition, HeaderTransform allows the uCDN to normalize, or modify, the Accept-Encoding header to allow for fine-grain control over the selection of the compression algorithm (e.g., gzip, compress, deflate, br, etc.).

MI.AllowCompress is a new GenericMetadata object that allows the dCDN to compress content before sending it to the client.

Property: allow-compress

*Description: If set to "True", the dCDN SHOULD try to compress the response to the client based on the Accept-Encoding request header.

*Type: Boolean.

*Mandatory-to-Specify: No. The default is "false".

The following examples illustrate the use of MI.AllowCompress in the context of the Processing Stages Model that allowed for metadata to be applied conditionally based on evaluation of HTTP request headers. See the Processing Stages Metadata Specification [SVTA2032] and Metadata Model Expression Language (MEL) Specification [SVTA2031]

The following is an example of the usage of MI.AllowCompress:

```
{
    "generic-metadata-type": "MI.AllowCompress",
    "generic-metadata-value": {
        "allow-compress": true
    }
}
```

Figure 3

More examples can be found in the Informative Examples (Section 6) section.

5. MI.ClientConnectionControl

Configuration metadata is required to define how connections against a client are maintained by a dCDN. In some use cases, like video streaming or other critical object delivery, UA applications connection to the cache server must be in control to have the best

user experience possible. This metadata allows a uCDN to accommodate device-specific constraints and performance optimization. A dCDN can also benefit from this configuration metadata to meet its security and resource consumption requirements.

MI.ClientConnectionControl is a new GenericMetadata object that specifies how a dCDN SHOULD manage its connections to UAs.

Property: connection-keep-alive-time-ms

*Description: Specifies the time, in milliseconds, to keep an idle connection open.

*Type: Integer

*Mandatory-to-Specify: No. When not specified, a default value selected by the dCDN will be used.

The following example shows how a connection setup and a keep alive timeout can be set for client connections to a dCDN:

```
{
  "generic-metadata-type": "MI.ClientConnectionControl",
  "generic-metadata-value": {
    "connection-keep-alive-time-ms": 3
  }
}
```

Figure 4

6. Informative Examples

The following is an example of an MI.AllowCompress in the context of the Processing Stages Model (see the Processing Stages Metadata Specification [SVTA2032]) that allows compression for requests to objects to the dCDN with extension *.m3u8:

```
{
  "generic-metadata-type": "MI.StageRules",
  "generic-metadata-value": {
    "match": {
      "expression": "req.h.uri *= '*.m3u8'"
    },
    "stage-metadata": {
      "generic-metadata": [
        {
          "generic-metadata-type": "MI.AllowCompress",
          "generic-metadata-value": {
            "allow-compress": true
          }
        }
      ]
    }
  }
}
```

Figure 5

The following is an example of an MI.AllowCompress that allows compression in the edge by the dCDN based on the client's Accept-Encoding header:

```
{
  "generic-metadata-type": "MI.StageRules",
  "generic-metadata-value": {
    "match": {
      "expression": "req.h.accept-encoding *= '*gzip*'"
    },
    "stage-metadata": {
      "generic-metadata": [
        {
          "generic-metadata-type": "MI.AllowCompress",
          "generic-metadata-value": {
            "allow-compress": true
          }
        }
      ]
    }
  }
}
```

Figure 6

7. Security Considerations

The FCI and MI objects defined in the present document are transferred via the interfaces defined in CDNI [RFC8006] which describes how to secure these interfaces protecting integrity and confidentiality while ensuring the authenticity of the dCDN and uCDN.

8. IANA Considerations

8.1. CDNI Payload Types

This document requests the registration of the following entries under the "CDNI Payload Types" registry hosted by IANA:

Payload Type	Specification
MI.CrossoriginPolicy	RFCthis
MI.AccessControlAllowOrigin	RFCthis
MI.AllowCompress	RFCthis
MI.ClientConnectionControl	RFCthis

Table 1: CDNI Payload Types

[RFC Editor: Please replace RFCthis with the published RFC number for this document.]

9. Acknowledgements

The authors would like to express their gratitude to the members of the Streaming Video Technology Alliance [SVTA] Open Caching Working Group for their guidance / contribution / reviews ...)

Particulary the following people contribute in one or other way to the content of this draft:

*Guillaume Bichot (Broadpeak)

*Christoph Neumann (Broadpeak)

*Chris Lemmons (Comcast)

*Pankaj Chaudhari (Disney Streaming)

*Robert Colantuoni (Disney Streaming)

*Will Power (Lumen)

*Rajeev RK (picoNETS)

*Shmuel Asafi (Qwilt)

*Yoav Gressel (Qwilt)

*Nir Sopher (Qwilt)

*Arnon Warshavsky (Qwilt)

*Eric Klein (Sirius XM)

*Francisco Cano Hila (Telefonica)

*Ben Rosenblum (Vecima)

10. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC8006] Niven-Jenkins, B., Murray, R., Caulfield, M., and K. Ma, "Content Delivery Network Interconnection (CDNI) Metadata", RFC 8006, DOI 10.17487/RFC8006, December 2016, <https://www.rfc-editor.org/info/rfc8006>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

11. Informative References

[SVTA] SVTA, "Streaming Video Technology Alliance Home Page", <https://www.svta.org>.

[SVTA2031] SVTA, "Metadata Expression Language (MEL)", <https://svta.org/documents/SVTA2031>.

[SVTA2032] SVTA, "Processing Stages Metadata Specification", <https://svta.org/documents/SVTA2032>.

Authors' Addresses

Alfonso Siloniz
Telefonica
Spain

Email: alfonsosiloniz@gmail.com

Glenn Goldstein
Lumen Technologies
United States of America

Email: glenng1215@gmail.com