

Workgroup: Network Working Group
Internet-Draft:
draft-power-cdni-cache-control-metadata-01
Published: 10 July 2023
Intended Status: Standards Track
Expires: 11 January 2024
Authors: W. Power G. Goldstein
 Lumen Technologies Lumen Technologies
 CDNI Cache Control Metadata

Abstract

This specification adds to the basic cache control metadata defined in RFC8006, providing content providers and upstream CDNs (uCDNs) more fine-grained control over downstream CDN (dCDN) caching. Use cases include overriding or adjusting cache control headers from the origin, bypassing caching altogether, or altering cache keys with dynamically generated values.

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 11 January 2024.

Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in

Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

- [1. Introduction](#)
- [2. Requirements](#)
- [3. MI.CachePolicy](#)
- [4. MI.NegativeCachePolicy](#)
- [5. MI.StaleContentCachePolicy](#)
- [6. MI.CacheBypassPolicy](#)
- [7. MI.ComputedCacheKey](#)
- [8. Conclusion](#)
- [9. Security Considerations](#)
- [10. IANA Considerations](#)
 - [10.1. CDNI Payload Types](#)
- [11. Acknowledgements](#)
- [12. References](#)
 - [12.1. Normative References](#)
 - [12.2. Informative References](#)
- [Authors' Addresses](#)

1. Introduction

In addition to the cache control parameters currently specified by the Cache object in [RFC8006], content providers and uCDNs often need more fine-grained control over dCDN caching, including scenarios where it is desirable to override or adjust cache control headers from the origin.

The following capabilities are required for commercial CDN and Open Caching use cases:

1. Positive Cache Control - Allows the uCDN to specify internal caching policies for the dCDN and external caching policies advertised to clients of the dCDN, overriding any cache control policy set in the response from the uCDN.
2. Negative Cache Control - Allows the specification of caching policies based on error response codes received from the origin, allowing for fine-grained control of the downstream caching of error responses. For example, it may be desirable to cache error responses at the dCDN for a short period of time to prevent an overwhelmed origin service or uCDN from being flooded with requests.
3. Cache Bypass Control - Allows content providers to bypass CDN caching when needed (typically for testing or performance benchmarking purposes).

4. Stale Content Policies - Allows control over how the dCDN should process requests for stale content. For example, this policy allows the content provider to specify that stale content be served from cache for a specified time period while refreshes from the origin occur asynchronously.
5. Dynamically Constructed Cache Keys - It is typical in advanced CDN configurations to generate cache keys that are dynamically constructed via lightweight processing of various properties of the Hypertext Transfer Protocol (HTTP) request and/or response. As an example, an origin may specify a cache key as a value returned in a specific HTTP response header. The Metadata Expression Language (MEL) is used to allow for such advanced cache key construction.

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.CachePolicy

MI.CachePolicy is a new GenericMetadata object that allows the uCDN to specify internal caching policies for the dCDN, as well as external caching policies advertised to clients of the dCDN (overriding any cache control policy set in the response from the uCDN).

Property: internal

-Description: Specifies the internal cache control policy to be used by the dCDN.

-Type: String. Either an integer in seconds (e.g., "5" is a five-second cache) or one of these enumerated options: "as-is", or "no-cache" or "no-store" as documented in [STD98].

-Mandatory-to-Specify: No. The default is to use the cache control policy specified in the response from the uCDN.

Property: external

-Description: Specifies the external cache control policy to be used by clients of the dCDN.

-Type: String. Either an integer in seconds (e.g., "5" is a five-second cache) or one of these enumerated options: "as-is", or "no-cache" or "no-store" as documented in [STD98].

-Mandatory-to-Specify: No. The default is to use the cache control policy specified in the response from the uCDN.

Property: force-internal

-Description: If set to "True", the metadata interface cache policy defined in the MI.CachePolicy internal property value will override any cache control policy set in the response from the uCDN. If set to "False", the MI.CachePolicy internal property value is only used if there is no cache control policy provided in the response from the uCDN.

-Type: Boolean

-Mandatory-to-Specify: No. The default is "False", which will apply the MI.CachePolicy internal property value only if no policy is provided in the response from the uCDN.

Property: force-external

-Description: If set to "True", the metadata interface cache policy defined in the MI.CachePolicy external property value will override any cache control policy set in the response from the uCDN. If set to "False", the MI.CachePolicy external property value is only used if there is no cache control policy provided in the response from the uCDN.

-Type: Boolean

-Mandatory-to-Specify: No. The default is "False", which will apply the MI.CachePolicy external property value only if no policy is provided in the response from the uCDN.

In example 1, an MI.CachePolicy sets the internal cache control policy to five seconds. The external cache policy is set to 'no-cache' and both policies are forced:

```
{
  "generic-metadata-type": "MI.CachePolicy",
  "generic-metadata-value": {
    "internal": "5",
    "external": "no-cache",
    "force-internal": true,
    "force-external": true
  }
}
```

In example 2, an MI.CachePolicy sets the internal cache control policy to "as-is" (keep the policy set in the response from the uCDN). The external cache policy is set to 'no-cache' and forced:

```
{
  "generic-metadata-type": "MI.CachePolicy",
  "generic-metadata-value": {
    "internal": "as-is",
    "external": "no-cache",
    "force-external": true
  }
}
```

In example 3, cache policies are set in the context of the Processing Stages Model (see the Processing Stages Metadata Specification [[SVTA2032](#)]). If the HTTP status code received from the origin is a 200, cache expiration is set to 300 seconds if no caching directives were set from the uCDN (unforced). If the HTTP status code received from the origin is a 503 or 504, the internal CDN caching policy is forced to 5 seconds and the external downstream policy is forced to “no-cache”.

```

{
  "generic-metadata-type": "MI.OriginResponseStage",
  "generic-metadata-value": {
    "match-groups": [
      {
        "if-rule": {
          "match": {
            "expression": "resp.status == 200"
          },
          "stage-metadata": {
            "generic-metadata": [
              {
                "generic-metadata-type": "MI.CachePolicy",
                "generic-metadata-value": {
                  "internal": "300",
                  "external": "300"
                }
              }
            ]
          }
        },
        "else-if-rules": [
          {
            "match": {
              "expression": "resp.status == 503 or resp.status == 504"
            },
            "stage-metadata": {
              "generic-metadata": [
                {
                  "generic-metadata-type": "MI.CachePolicy",
                  "generic-metadata-value": {
                    "internal": "5",
                    "external": "no-cache",
                    "force-internal": true,
                    "force-external": true
                  }
                }
              ]
            }
          }
        ]
      }
    ]
  }
}

```

4. MI.NegativeCachePolicy

MI.NegativeCachePolicy is a new GenericMetadata object that allows the specification of caching policies based on response codes received from the origin. MI.NegativeCachePolicy is a simple alternative to using the origin response processing stage [[SVTA2032](#)] with a match criteria on specific HTTP response codes, useful when there a single caching policy needs to be specified for a list of one or more response codes from the origin.

Property: error-codes

-Description: An array of HTTP response status codes (see Sections 15.5 and 15.6 of [[RFC9110](#)]) , that, if returned from the uCDN, will be cached using the cache policy defined by the cache-policy property.

-Type: Array of HTTP response status codes encoded as strings. Any HTTP status code from 100 to 599, or one of the special values, "2xx", "3xx", "4xx" or "5xx", where "xx" implies everything from 00 to 99. Note that the use of 4xx would specify that 416 responses are cached.

-Mandatory-to-Specify: No. The default is to revert to [[RFC8006](#)] behavior. An empty or unspecified list MAY function as a means to revoke a list inherited from an upper-level configuration.

Property: cache-policy

-Description: The MI.CachePolicy to apply to the HTTP response status codes returned by the uCDN.

-Mandatory-to-Specify: Yes

In the following example, a MI.NegativeCachePolicy object applies a no-cache policy whenever error codes 503 or 504 are seen from the origin..

```

{
  "generic-metadata-type": "MI.NegativeCachePolicy",
  "generic-metadata-value": {
    "error-codes": [ "503", "504" ],
    "cache-policy": {
      "internal": "5",
      "external": "no-cache",
      "force-internal": true,
      "force-external": true
    }
  }
}

```

5. MI.StaleContentCachePolicy

MI.StaleContentCachePolicy is a new GenericMetadata object that allows the uCDN to specify the policy to use by a dCDN when responding with stale content. For example, this policy allows the content provider to specify that stale content be served from cache for a specified time period while refreshes from the origin occur asynchronously.

Property: stale-while-revalidating

- Description: Instructs the dCDN to serve a stale version of a resource while refreshing the resource with the uCDN. When set to "True", the dCDN will return a previously cached version of a resource while the resource is refreshed with the uCDN in the background.

- Type: Boolean

- Mandatory-to-Specify: No. The default is "False", which waits for the uCDN to refresh a resource before responding to the client.

Property: stale-if-error

- Description: Instructs the dCDN to serve a stale version of a resource if any one of a specified set of HTTP status codes was received when trying to refresh the resource with the uCDN. In this case, the dCDN will return a previously cached version of a resource instead of caching the error response. While this capability is typically used for well-understood HTTP error status codes, a list of any HTTP codes can be provided for maximum flexibility.

- Type: Array of HTTP response status codes encoded as strings.. Any HTTP status code from 100 to 599, or one of the special

values "2xx", "3xx", "4xx" or "5xx", where "xx" implies everything from 00 to 99.

-Mandatory-to-Specify: No. The default is to not serve stale content. An empty or unspecified list may function as a means to revoke a list inherited from an upper-level configuration.

Property: failed-refresh-ttl

-Description: Instructs the dCDN to serve a stale version of a resource for the number of seconds specified in failed-refresh-ttl before trying to revalidate the resource with the uCDN. Use of failed-refresh-ttl allows the load to be reduced on the uCDN during times of system stress.

-Type: Integer

-Mandatory-to-Specify: No. The default is zero, in which no stale content is served.

In example 1, an MI.StaleContentCachePolicy where stale-while-revalidating is true instructs the dCDN to respond with a stale cached version of the resource while it refreshes the resource with the uCDN in the background:

```
{
  "generic-metadata-type": "MI.StaleContentCachePolicy",
  "generic-metadata-value": {
    "stale-while-revalidating": true
  }
}
```

In example 2, an MI.StaleContentCachePolicy where stale-if-error instructs the dCDN to use the stale cached resource if it receives an error of type 503 or 504 when trying to refresh the resource with the uCDN.

failed-refresh-ttl instructs the dCDN to use a five-second cache time-to-live (TTL) on the resource that receives an error when refreshing from the uCDN. That is, after five seconds, the dCDN will attempt to refresh the resource with the uCDN.

```
{
  "generic-metadata-type": "MI.StaleContentCachePolicy",
  "generic-metadata-value": {
    "stale-if-error": [ "503", "504" ],
    "failed-refresh-ttl": 5
  }
}
```

In example 3, an `MI.StaleContentCachePolicy` where `stale-while-revalidating` is true instructs the dCDN to respond with a stale cached version of the resource while it refreshes the resource with the uCDN in the background.

`stale-if-error` instructs the dCDN to use the stale cached resource if it receives an error of type 404 or any 5xx status when trying to refresh the resource with the uCDN.

1. `stale-if-error` instructs the dCDN to use the stale cached resource if it receives an error of type 404 or any 5xx status when trying to refresh the resource with the uCDN.
2. `failed-refresh-ttl` instructs the dCDN to use a five-second cache TTL on the resource that receives an error when refreshing from the uCDN. That is, after five seconds, the dCDN will attempt to refresh the resource with the uCDN.

```
{
  "generic-metadata-type": "MI.StaleContentCachePolicy",
  "generic-metadata-value": {
    "stale-while-revalidating": true,
    "stale-if-error": [ "404", "5xx" ],
    "failed-refresh-ttl": 5
  }
}
```

6. `MI.CacheBypassPolicy`

`MI.CacheBypassPolicy` is a new `GenericMetadata` object that allows a client request to be set as non-cacheable. It is expected that this feature will be used to allow clients to bypass caching when testing the uCDN fill path. Note: `MI.CacheBypassPolicy` is typically used in conjunction with a path match or match expression on a header value or query parameter. Any content previously cached (by client requests that do not set `MI.CacheBypassPolicy`) MUST NOT be evicted.

Property: `bypass-cache`

-Description: A Boolean value that can activate the feature for a given client request. It is expected that this feature will be used within ProcessingStages [[SVTA2032](#)] to allow a client request to be marked to bypass caching.

-Type: Boolean

-Mandatory-to-Specify: No. The default is "False".

In the following example, an `MI.CacheBypassPolicy` is invoked when the `clientrequest` HTTP header of "cdn-bypass" is true:

```

{
  "generic-metadata-type": "MI.ProcessingStages",
  "generic-metadata-value": {
    "client-request": [
      {
        "match": {
          "expression": "req.h.cdn-bypass == 'true'"
        },
        "stage-metadata": {
          "generic-metadata": [
            {
              "generic-metadata-type": "MI.CacheBypassPolicy",
              "generic-metadata-value": {
                "bypass-cache": true
              }
            }
          ]
        }
      }
    ]
  }
}

```

7. MI.ComputedCacheKey

It is typical in advanced CDN configurations to generate cache keys that are dynamically constructed via lightweight processing of various properties of the HTTP request and/or response. As an example, an origin might specify a cache key as a value returned in a specific HTTP response header.

MI.ComputedCacheKey is a new GenericMetadata object that allows the specification of a cache key using the Metadata Expression Language (MEL) defined in [[SVTA2031](#)]. Typical use cases would involve constructing a cache key from one or more elements of the HTTP request. In cases where both the MI.ComputedCacheKey and the Cache object are applied, the MI.ComputedCacheKey MUST take precedence.

MI.ComputedCacheKey is, by default, allowed within any of the processing stages defined in [[SVTA2032](#)]. It should be noted, however, that a dCDN MAY only allow cache keys to be altered at certain processing stages (such as the clientRequestStage) but not at other stages (such as the originResponse or clientResponseStage).

A dCDN MAY use FCI.MetadataExtended [SVTA2041] to advertise such restricted usage contexts.

Property: expression

-Description: The expression that specifies how the cache key is to be constructed.

-Type: String. An expression using the Metadata Expression Language (MEL) [SVTA2031] to dynamically construct the cache key from elements of the HTTP request and/or response.

-Mandatory-to-Specify: Yes

In the following example, a custom request header is used as the cache key instead of the Uniform Resource Identifier (URI) path:

```
{
  "generic-metadata-type": "MI.ComputedCacheKey",
  "generic-metadata-value": {
    "expression": "req.h.X-Cache-Key"
  }
}
```

8. Conclusion

This specification defines a new set of cache control metadata objects that meet the needs of content providers, CDNs, and Open Caching Systems. As the standard matures and gains wider adoption, it is expected that additions to this set of cache control policies will be required.

9. Security Considerations

The FCI and MI objects defined in the present document are transferred via the interfaces defined in CDNI [RFC8006]. [RFC8006] describes how to secure these interfaces, protecting the integrity, confidentiality and ensuring the authenticity of the dCDN and uCDN. The security provide by [RFC8006] should therefore address the above security concerns.

10. IANA Considerations

10.1. CDNI Payload Types

TBD.

11. 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 ...)

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

Guillaume Bichot - Broadpeak

Pankaj Chaudhari - Disney Streaming Services

Yoav Gressel - Qwilt

Alfonso Siloniz - Telefonica

Ben Rosenblum - Vecima

12. References

12.1. Normative References

- [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>>.
- [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>>.
- [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>>.
- [STD98] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, June 2022.

12.2. Informative References

- [SVTA] "Streaming Video Technology Alliance Home Page", <<https://www.svta.org>>.
- [SVTA2032] Goldstein, G., Ed., Chaudhari, P., Power, W., Gressel, Y., and A. Warshavsky, "Processing Stages Metadata Specification", 2 July 2021, <<https://svta.org/documents/SVTA2032>>.

[SVTA2031]

Goldstein, G., Ed., Chaudhari, P., Power, W., Gressel, Y., and A. Warshavsky, "Metadata Model Expression Language (MEL) Specification", 2 July 2021, <<https://svta.org/documents/SVTA2031>>.

[SVTA2041]

Goldstein, G., Ed., Chaudhari, P., Power, W., Gressel, Y., and A. Warshavsky, "Metadata Capabilities", <<https://svta.org/documents/SVTA2041>>.

Authors' Addresses

Will Power
Lumen Technologies
United States of America

Email: wrcpower@gmail.com

Glenn Goldstein
Lumen Technologies
United States of America

Email: glennng1215@gmail.com