SVTA Configuration Interface

IETF/CDNi Metadata Model Extension: Cache Control Metadata
March 2023 (IETF 116)
Cache Control Metadata

- RFC 8006 defines an initial set of CDNI GenericMetadata objects that allow a dCDN to serve content requests on behalf of a uCDN
- draft-power-cdni-cache-control-metadata-00 proposes extensions to RFC 8006 to give Content Providers and uCDNs more fine-grained control over dCDN caching
- Additional CDNI GenericMetadata objects are defined that allow control over:
  - Internal and external cache policy
  - Caching of negative responses (e.g., 503)
  - Serving of stale cached resources if an uCDN is unavailable
  - Customization of cache key for more advanced production use cases
  - Selective bypass of cached resources
Cache Control Metadata

**MI.CachePolicy**
- Allows the internal and external cache policy of resources to be configured
- Supports cache TTL values in seconds and directives: as-is, no-cache, no-store
- Properties ‘force-internal’ and ‘force-external’ allow the cache policy to override Cache-Control provided by the Origin, if required
- Example:

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

**MI.NegativeCachePolicy**
- Allows cache policy for ‘negative’ responses to be configured
- Supports setting cache policy for an array of ‘negative’ response codes
- Example:

```json
{
    "generic-metadata-type": "MI.NegativeCachePolicy",
    "generic-metadata-value": {
        "error-codes": [ "404", "503", "504" ],
        "cache-policy": {
            "internal": "5",
            "force-internal": "true"
        }
    }
}
```
Cache Control Metadata

**MI.StaleContentCachePolicy**
- Allows the cache policy of ‘stale’ resources to be configured
- Supports stale-while-revalidating, stale-if-error, and failed-refresh-ttl
- Example:

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

**MI.ComputedCacheKey**

- Allows advanced control over setting the cache key for resources
- Supports constructing a cache key using attributes from a request or from an origin response.
- Used the proposed CDNI Metadata Expression Language (MEL) to dynamically construct cache keys
- Example:

  ```json
  {
    "generic-metadata-type": "MI.ComputedCacheKey",
    "generic-metadata-value": {
      "expression": "req.h.X-Cache-Key"
    }
  }
  
  Note: CDNI Metadata Expression Language (MEL) will be presented in a draft planned for IETF-117. Specification currently under review within the SVTA.
Cache Control Metadata

**MI.CacheBypassPolicy**
- Allows a client request to bypass cache
- For example, allow a request to bypass cache while testing
- Existing cached resources are not evicted by this directive
- Example:

```json
# Match on path or request header value
{
    "generic-metadata-type": "MI.CacheBypassPolicy",
    "generic-metadata-value": {
        "bypass-cache": "true"
    }
}
```
Questions from Original Submission

CachePolicy: why should a dCDN allow a uCDN to dictate caching policies? could this be achieved with cache-control headers?

This capability extends Cache-control headers by allowing specification of both client and CDN caching parameters. dCDNs add a caching layer that require distinct control parameters.

ComputedCacheKey: is there something specific that is missing from the existing Cache metadata?

Yes, there can be cases, for example, when the cache key is computed based on the value of an HTTP response header.

NegativeCacheKey: what kind of caching policy for what kind of error?

As an example, it may be desirable to cache error responses at the CDN for a short period of time to prevent an overwhelmed origin service from being flooded with requests.

CacheBypassPolicy: is this specifying something that helps the dCDN identify requests for which to bypass caching?

This capability’s purpose is to facilitate QA and testing scenarios where caching rules on the CDN need to be overridden on a per-request basis.

StaleContentCachingPolicy: similar to other questions about dictating dCDN policy.

Typical use would allow the content provider to specify that stale content be served from cache for a specified time period while refreshes from the origin occur asynchronously.
Conclusion

Based on the contents of this presentation, Can the CDNI working group accept this document as a Working Group Draft?