OCAPI Performance Best Practices External FAQ (FINAL)

The purpose of this document is to capture best practices related to developing OCAPI applications that scale and perform at a high-level. Customers are encouraged to use the guidance in this document to inform their design and delivery approach for “Commerce Everywhere” applications build on the Commerce Cloud Platform that leverage OCAPI.

OCAPI Requests vs. Storefront Requests

This section provides guidance on the similarities and differences between OCAPI and Storefront requests on the Salesforce B2C Commerce platform. It contrasts the resource costs and platform overhead differences between OCAPI and storefront requests.

WHAT ARE THE KEY DIFFERENCES BETWEEN OCAPI REQUESTS AND STOREFRONT-BASED PIPELINE OR CONTROLLER REQUESTS?

There are several fundamental differences between OCAPI and pipeline / controller-based requests that should be considered by developers when designing an OCAPI strategy for API-first applications.

- OCAPI requests are not cached by the web-tier; they are cached via a separate cache that is owned by the application-tier – which follows the same cache-clear behavior as the web-tier.
- OCAPI requests always incur application-tier overhead – regardless of their cache status. Since OCAPI requests are not cached by the web-tier, they always incur the overhead associated with uncached storefront requests.
- OCAPI documents follow a different caching behavior than storefront requests. Where storefront requests will respect the individual cache TTLs of remote-include content, *OCAPI requests that comprise multiple documents will respect the minimum cache TTL of all documents retrieved via the request.*

> *As each OCAPI request (cached or not) is processed by the application-tier – OCAPI requests are more expensive to process from a resource standpoint than cached web-requests. The Salesforce B2C Commerce platform can scale the performance of these requests, but we advise all customers to follow performance best-practices to ensure that instance-group resources are maximized during storefront operation.*

DO OCAPI REQUESTS FOLLOW THE SAME THREAD BEHAVIOR AS PIPELINE / CONTROLLER REQUESTS?

Yes. When an OCAPI request is made, a new thread is generated to handle the retrieval of the requested storefront data and the generation of the OCAPI response. As each OCAPI request generates a new thread, it is critical to remember that:

- Long running requests will keep threads occupied
- Request performance and payload contents should be monitored and performance tuned to locate the sweet-spot between payload size and request performance
    - Ex. Instead of pulling fifty (50) products in a request, pull ten (10) across five separate requests
    - Ex. Leverage but don't over-use or poorly implement batch requests

> *This example is provided as guidance on how to strike a balance between request payload size and performance. Performance monitoring and tuning can be used to locate the sweet spot between these two criteria. The sweet spot will always be application and customer specific – which is why OCAPI requests must be tuned to identify a predictable level of performance.*

WHAT IMPACT CAN LONG-RUNNING OCAPI REQUESTS HAVE ON CUSTOMER EXPERIENCES?

The simple answer here is that customer experiences dependent on slow-performing OCAPI requests will be forced to wait for those requests to complete before continuing to interact with the storefront. Additionally, long-running OCAPI requests will tie-up platform resources that could be leveraged by other storefront or OCAPI requests. If enough of these long-running requests are being processed concurrently, they could negatively impact customer experiences provided by the platform.

> *OCAPI, storefront, and scheduled jobs all pull from the same platform resource pool. As such, it is critical to remember that poorly performing routines in one area (ex. OCAPI or jobs) can negatively impact the performance of customer experiences in another area (ex. the storefront). OCAPI requests can influence storefront behavior – just as poorly performing storefront logic can negatively impact OCAPI performance.*

OCAPI Request Caching

This section provides guidance on the the caching options available for OCAPI requests by the Salesforce B2C Commerce platform. It explains which OCAPI api-type requests can be cached as well as cache-key structure best practices..

CAN OCAPI DATA API REQUESTS BE CACHED BY SALESFORCE B2C COMMERCE?

No. Due to the nature of OCAPI Data API requests, they cannot be cached. All Data API requests will incur real-time processing via the application-tier to retrieve the data requested, format it as part of the REST API response, and deliver the response.

> *Since Data API requests are not cached and generate application-tier overhead with each request, please be mindful of the frequency by which the Data API is used – and the impact OCAPI Data API requests can have on customer experiences.*

CAN META API REQUESTS BE CACHED BY SALESFORCE B2C COMMERCE?

Yes. The Meta API by default has caching enabled. The default cache time for Meta API resources is one (1) day – and cannot be configured. To clear the instance cache for the Meta API documents you must invalidate the Business Manager page cache.

> *The page cache can be manually invalidated via Business Manager or programmatically by leveraging the InvalidateWebCache pipelet. Please note that the entire page cache must be invalidated to clear the cache of Meta API documents (clearing a cache partition will not invalidate the Meta API cache).*

CAN OCAPI SHOP API REQUESTS BE CACHED BY SALESFORCE B2C COMMERCE?

Yes. a sub-set of OCAPI Shop API requests can be cached by Salesforce B2C Commerce. Each Shop resource can have its individual cache times configured following the structure of the OCAPI settings document. There are two properties that can be used to configure cache behavior for Shop API requests.

Cache_Time Property (integer)
The cache_time property represents the period in seconds before the cached version of an OCAPI document becomes stale and is re-requested by the Salesforce B2C Commerce platform.

- The default cache time for shop resources is sixty (60) seconds
- The minimum cache time is zero (0) seconds
- The maximum cache time is twenty-four hours (86,400 seconds)

> *It is important to remember that OCAPI Shop API requests are not cached by the web-tier – but instead via an application-tier cache dedicated to OCAPI documents. This means that cached OCAPI requests are inherently more expensive to process from a resource standpoint than cached web-tier requests – as they need to go to the application-tier to retrieve cached OCAPI content.*

Personalized_Caching_Enabled (boolean)
The personalized_caching_enabled flag is used to determine the personalized caching behavior of a cacheable Shop API resource. By default, the Salesforce B2C Commerce platform will cache personalized resources as soon as a customer context (JWT Token) is provided.

> *This property can be used to explicitly disable personalized caching to improve the performance of your storefront or OCAPI dependent customer experiences. Before disabling personalized caching, please confirm that the JWT Token being provided is necessary – as this will trigger personalized caching.*

WHAT URL PARAMETERS ARE USED TO GENERATE A CACHE-KEY FOR OCAPI SHOP API REQUEST CONTENT?

The following elements of an OCAPI request are used to generate the internal cache-key for data used in Shop API responses. These elements include:

- hostname
- siteId
- clientId
- Any expansion-parameters provided
- Any custom url / request parameters

> *OCAPI-cache keys behave in a similar manner to storefront page-cache keys. Maximizing the performance of OCAPI requests requires that customers minimize cache misses by configuring appropriate cache_time TTLs and eliminate cache-stores (the duplication of request content in-cache due to a lax url strategy). A coherent and enforced cache key strategy should be followed by customers to ensure that similar OCAPI requests generate consistently structured cache keys. This will minimize cache-misses – and improve application performance.*

CAN URL QUERY PARAMETERS INFLUENCE CACHE HIT / MISS PROBABILITY FOR OCAPI REQUESTS?

Yes. The internal OCAPI request cache-key includes the query parameters included with an OCAPI request. To maximize cache-hits for OCAPI requests, we recommend building urls in client applications with the query parameters in alphabetic sort order. Urls that represent the same OCAPI request but do so with url uniqueness will generate multiple cache entries in the OCAPI cache for this data. This will increase cache-stores – making inefficient use of cache-resources.

In this example, the following three urls request the same data, but generate duplicate cache entries:

- dw/shop/v18_8/products/123?client_id=aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa&expand=availability,variations
- dw/shop/v18_8/products/123?expand=availability,variations&client_id=aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
- dw/shop/v18_8/products/123?expand=variations,availability&client_id=aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

To maximize cache performance, always evaluate the url query parameters attached to an OCAPI request – and sort parameters and parameter values alphabetically.

> *Improving the cache-hit rate of OCAPI Shop API requests is a key to improving storefront performance. The best way to ensure that cache-hit rates are high for OCAPI Shop API requests is to always build OCAPI urls with query parameters in alphabetic order.*

DOES OCAPI CACHING OCCUR AT THE REQUEST LEVEL?

No. OCAPI caching occurs at the resource level in a manner similar to remote-include caching via pipelines. The OCAPI request model supports retrieving the data for multiple resources as part of a single request (ex. via OCAPI batch requests or OCAPI expansions). When this occurs, the individual resource documents are retrieved from cache and used to construct the appropriate response for a given request.

- An OCAPI request is made to request the variations and availability for a product.
    - dw/shop/v18_8/products/123?expand=availability,variations
- OCAPI checks its cache to see if the response containing the expansion properties and response document exist in cache. If not, the response is assembled and placed in cache – referencing the cache key that includes the request url and expansion properties.
- OCAPI calculates the TTL for the cache entry by examining all child documents that comprise the response – and finding the lowest cache TTL configured. This TTL is then applied to the response document via the cache-control header.

> *What this means practically is that if you retrieve expansion properties with divergent cache time configurations – the shortest configured cache duration is applied to the request. This will force documents that may have a long cache time configuration (ex. variations at 10 hours) to be retrieved from the database tier if they are requested with documents configured with a short cache time (ex. availability at 5 minutes).*

- The response is returned to the caller – and includes the collection of resource documents describing the variations and their availability; the OCAPI cache is used to retrieve the resource documents without having to make any ORM calls to access the database tier data for this specified product.

> *The important item to take away from this question is that OCAPI responses are not cached. Instead, the request response can be pulled from cache. Every OCAPI request, however, still incurs the overhead associated with an application-tier request that is parsing the OCAPI request – retrieving the resource documents from cache, and assembling the documents into the expected response.*

WHEN AN OCAPI REQUEST RETURNS AN INSTANCE OF A RESOURCE DOCUMENT, IS THAT INSTANCE AUTOMATICALLY CACHED AND AVAILABLE TO BE USED BY SUBSEQUENT REQUESTS?

No. OCAPI cache entries are tied directly to their parent requests. When OCAPI checks its cache for cached documents, it looks specifically for the “master” document representing the request response. It does not retrieve child documents from cache – and does not assemble responses from multiple cached documents.

> *The key takeway from this is is that calling a resource document via one request does not mean that document has been cached and can now be leveraged by subsequent OCAPI requests. Cache entries are always oriented on their parent request – and are not shareable by other requests.*

WHAT ARE THE DIFFERENT TYPES OF OCAPI DOCUMENTS THAT LIVE IN ITS CACHE?

The OCAPI cache typically contains two different types of documents:

- Object (ex. product, category, content) entry – which represents an instance of a specific API object.
- Composite (ex. product with variations and availability) entry – which represents an instance of an API object and includes expansion documents defined at request time (ex. variations and availability).

The primary difference between these two cache entry types is that object entries can be re-used by OCAPI requests that retrieve multiple instances of the same API object and do not include expansion parameters. Composite entries, on the other hand, cache the response results as a single entry – and access the database tier to request all api objects regardless of whether an object entry may exist in cache.

> *Composite entries are all request driven, have their own cache key, and are created by accessing the database-tier to retrieve the objects representing the documents requested. Composite documents do not leverage any cache object type documents during the construction of their response.*

THIS BEHAVIOR SEEMS FUNDAMENTALLY DIFFERENT FROM FROM STOREFRONT PIPELINE AND PAGE CACHING BEHAVIOR. ARE THERE ANY OTHER DIFFERENCES TO BE AWARE OF REGARDING THE CACHE BEHAVIOR OF OCAPI REQUESTS?

The OCAPI cache behavior is different from storefront pipeline and page caching behavior – and these differences must be taken into consideration when designing an OCAPI strategy for customer experiences. The complete set of cache behavior rules for OCAPI Shop API requests are as follows:

- OCAPI Shop API resources support configurable internal caching that prevents database-tier level transactions from being used to retrieve the data used to build resource documents.
- A sub-set of OCAPI Shop API resources support cache-control headers embedded in their responses. The value embedded in these headers is driven by the cache_time OCAPI setting configured for each resource and describes the remaining time available that the specific resource will continue to be pulled from the internal OCAPI cache before being requested via the database-tier.
- When an OCAPI request includes multiple resources that can be pulled from cache, the Salesforce B2C Commerce platform will identify the lowest cache_time configured among the resources used to assemble the response – and apply that cache_time to calculate the retrieval method of all resources within a request.
- The lowest cache_time is also applied to the response's cache-control header.

HOW DOES THIS BEHAVIOR DIFFERENCE IMPACT THE PERFORMANCE OF OCAPI REQUESTS CONTAINING REFERENCES TO MULTIPLE RESOURCES?

Simply put, the cache_times configured for individual resources are not applied to requests comprising multiple resources. Instead, the lowest cache_time found among the resources requested is used to drive OCAPI cache behavior. What this means is that customers should not expect the specific cache_time for Shop API resources to be applied globally across all OCAPI requests. Making this assumption will likely lead to poorly performing OCAPI Shop API requests.

> *This is a **significant difference** in OCAPI cache behavior compared to storefront pipeline and page-cache behavior. What this means is that cache_times for OCAPI Shop API requests that retrieve multiple resources will be overridden by the lowest cache_time configuration found within the resources that comprise a request.*

As a reminder – a request that includes resources which have varying cache times (ex. variations vs. availability) will result in the lowest configured cache_time being used to determine when resource data should be retrieved from the dedicated OCAPI cache vs. the database-tier. The following example demonstrates the impact this can have on an OCAPI request:

- An OCAPI request is made to request the variations and availability for a product
    - dw/shop/v18_8/products/123?expand=availability,variations
        - Variations are configured with a cache_time of twenty-four hours (86,400 seconds)
        - Availability is configured with a cache_time of five minutes (300 seconds)
- Salesforce B2C Commerce reviews the configured cache_time values for each resource included in the request, and uses that cache_time to evaluate which resources should be retrieved from the OCAPI cache vs. via the database-tier.
    - In this request, the variations resources are given a cache_time of five minutes instead of their configured twenty-four hours. This means that the OCAPI cache documents representing each variation will be

requested from the database-tier once the cache_time for their parent request expires – even though their configured cache_time threshold has not been met.

- Salesforce B2C Commerce checks the OCAPI cache for an entry representing the request response. If no entry is found, OCAPI will then access the database-tier and request each of the individual platform objects that meet the expansion criteria defined in the request. These documents are then used to assemble the response -- and the response is cached prior to returning it to the caller.
- The response is returned to the caller – and is then given a cache-control header value representing the lowest cache_time value found among all resources included in the original OCAPI request.

These rules must be taken into account when using OCAPI – as the cache behavior of OCAPI Shop API requests that support multiple resources is opposite to storefront pipeline and page caching behavior. Customers that apply storefront pipeline and caching behavior principles to their OCAPI strategy are likely to see poor OCAPI request performance. Adjusting the request strategy to respect these cache-rules will improve the runtime performance of OCAPI requests.

> *When designing an OCAPI request that retrieves multiple resources having various unique cache_time considerations, customers should evaluate the configured cache_time setting of each resource and evaluate the performance of the request as well as the calculated cache-control header TTL value. Depending on request performance, it may be worthwhile to group resources that have similar cache_time configurations – to ensure that requests with longer configured cache_time settings have their TTLs respected.*

Improving OCAPI Request Performance

This section provides guidance on how to improve OCAPI Shop API request performance by leveraging the cache-control headers included in certain Shop API responses. OCAPI responses can be cached via http by respecting the cache-control headers – effectively enabling the caching of OCAPI Shop API responses outside of the platform (ex. by a browser or proxy-server) and dramatically improving OCAPI request performance.

WHAT OPTIONS ARE AVAILABLE WITHIN SALESFORCE B2C COMMERCE TO IMPROVE OCAPI REQUEST PERFORMANCE?

Specific OCAPI Shop API responses include cache headers that can be used to enable client-side or proxy caching of these requests via http. The cache-control headers included represent configured cache_time property for the requested resource – minus the time elapsed since the response document was placed in cache. As an example:

- The max age of 900 seconds is configured for the categories resource.
- After an initial GET request, the value added to the HTTP header would be 900.
- If a second GET request for the same document arrived 120 seconds later, the HTTP header value would be 780.

These headers can be used to enforce cache-behavior of responses at a layer that exists outside of the Salesforce B2C Commerce platform. By leveraging these headers, the platform can effectively be bypassed as requests are now informed of how long response-content should be cached.

> *Please note that response cache headers are only delivered for specific GET and HEAD Shop API requests. The resources that support these headers include categories, content, content_search, custom_objects, folders, products, product_search, promotions, search_suggestions, site, and stores.*

WHICH OCAPI SHOP API RESOURCES DO NOT SUPPORT CACHE-CONTROL HEADERS?

The following OCAPI Shop API resources do not include cache-control headers in their GET and HEAD responses: AI, baskets, customers, gift_certificate, order_search, orders, price_adjustment_limits, product_lists, and sessions.

> *These OCAPI Shop API resources represent data objects tied to customer data, personalization data, or order processing. Due to the personal and time / interaction sensitive nature of these data objects, caching has not been enabled for these resources.*

HOW CAN THE CACHE-CONTROL HEADERS BE USED TO IMPROVE OCAPI REQUEST PERFORMANCE?

An external CDN can be configured to reside in-front of the Salesforce B2C Commerce platform to broker all OCAPI requests and use the cache-control headers embedded in the supported OCAPI Shop API responses to enable the caching of these responses off-platform as static content. As cache-behavior can now be managed via the external CDN, customers have the ability to tune cache behavior for their specific OCAPI needs.

- Customers can treat some OCAPI requests as static content and provide default minimum cache times
- Depending on the CDN, customers can create cache partitions for OCAPI content – customizing the cache behavior for each partition based on the sub-set of OCAPI requests that comprise the partition
- Customers can manage the external CDN cache separately from the storefront page cache – effectively de-coupling the OCAPI cache from the Salesforce B2C Commerce storefront's page cache

> *What the external CDN does is implement a caching layer for the sub-set of supported OCAPI Shop API requests outside of the platform. The caching layer can be managed separately from the Salesforce B2C Commerce platform's existing page cache – and its caching behavior can be customized to support the customer experience use-cases that are leveraging OCAPI.*

IS AN EXTERNAL CDN INCLUDED IN THE SERVICES THAT SALESFORCE PROVIDES COMMERCE CLOUD CUSTOMERS?

No. Customers have the option to implement and configure an external CDN – and many do to leverage the performance and security benefits that are provided by these offerings. While Salesforce B2C Commerce does provide customers with an embedded CDN, this offering does not include a reverse-proxy that customers can use to set up a caching layer for OCAPI requests. To achieve this goal, customers are required to set up an external CDN.

> *An external CDN is something that customers will be required to pay for outside of their existing Salesforce contract. If customers are seeking advice on external CDN vendors, Cloudflare does offer reverse-proxy capabilities that can be used to provide an OCAPI caching layer. For additional guidance on using an external CDN, please review the xChange article titled Using a Third Party CDN with Demandware. The article is dated, but the content is directionally relevant.*

IS THERE A PREFERRED WAY TO MAKE OCAPI REQUESTS FROM DIFFERENT CLIENTS (EX. BROWSERS, MOBILE APPLICATIONS, ETC)?

We recommend that customers always retrieve the JWT token and use that token to make OCAPI requests. Leveraging the JWT token will conserve platform resources incurred during OCAPI requests and including it in an OCAPI request introduces multiple benefits:

- session stickiness – which allows a session to be re-used and prevents new sessions from being created
- caching of platform objects via OCAPI requests is enabled with the presence of a JWT token
- personalization can be enabled or disabled when JWT tokens are leveraged via the personalized_caching_enabled OCAPI configuration setting.

Extending OCAPI Best Practices

This section provides guidance of how to extend OCAPI through custom hooks or custom end-points. Please follow this guidance to ensure that customizations are made in a manner that does not negatively impact storefront performance.

HOW DOES CACHE TIME CONFIGURATIONS ON A RESOURCE IMPACT THE EXECUTION OF CUSTOM HOOKS DESIGNED TO MODIFY AN OCAPI RESPONSE?

When an OCAPI resource has a cache time configured, the custom hook logic attached to the OCAPI resource is only executed once – before the response content is cached. Once the response content is available from the OCAPI cache – the hook logic designed to modify the response is no longer executed.

WHAT ARE SOME DEVELOPMENT PRACTICES TO AVOID WHEN WORKING WITH HOOK CUSTOMIZATIONS FOR A GIVEN OCAPI RESOURCE?

The following collection of OCAPI customization principles should be considered when evaluating the use of hooks to customize an existing OCAPI response document.

- Only use hooks when necessary – as hooks introduce dependencies that can impact overall request performance. Hooks introduce additional request overhead with their implementation so their use should be driven by a valid business case.
- When modifying an OCAPI response, use the web object as much as possible. Only use scriptAPI objects on an as-needed basis and minimize the business logic complexity within a hook.
- Avoid requesting persisted objects from within a hook (ex. ProductMgr.getProduct() or product.getVariations()).

> *The pre and post-processing logic embedded in a custom hook should be minimized – as this overhead is executed in addition to the overhead of the parent OCAPI request. Depending on the complexity of the custom-hook logic, it may be better to implement the processing as part of a separate request. This should be evaluated on a case-by-case basis, and driven by performance metrics collected from the parent OCAPI request.*

WHAT GUIDANCE CAN WE PROVIDE DEVELOPERS CREATING CUSTOM API ENDPOINTS LEVERAGING CUSTOM CONTROLLERS?

It is important to remember that controllers are not stateless; accessing every controller endpoint spawns a new request object and creates a new session. To prevent the creation of orphaned sessions (sessions instantiated, but never leveraged beyond the first request) – client applications calling controller-driven API endpoints should accept the cookies embedded in the endpoint response – and include these cookies in subsequent request.

> *Controller endpoints that are requested by clients that do not accept the cookies included in the response will create a new session for every endpoint request made. If the endpoint experiences high-traffic, the impact on the platform will be a flood of orphaned sessions that can negatively impact overall performance.*

OCAPI Cache Management

This section provides guidance on considerations to be mindful of when managing your instance group's cache footprint to support both storefront and OCAPI requests at scale.

HOW DOES REPLICATION IMPACT THE BEHAVIOR OF THE INTERNAL OCAPI CACHE?

Replication has the same impact on the internal OCAPI cache as the site / web-tier cache. Whenever a replication event is performed, the site and OCAPI caches are both cleared and will be rebuilt through request traffic according to cache behavior configuration.

HOW CAN I CLEAR THE INTERNAL OCAPI CACHE MANUALLY?

The OCAPI cache can be manually cleared by clearing the site / web-tier cache for a given storefront. Please note that doing so will invalidate both the internal OCAPI cache as well as the page-cache used by the storefront to serve cached content. These two caches cannot be managed separately.

DOES SALESFORCE B2C COMMERCE PROVIDE ANY VISIBILITY INTO THE CACHE BEHAVIOR OF OCAPI REQUESTS? SOMETHING SIMILAR TO THE PIPELINE PERFORMANCE REPORT – THAT HIGHLIGHTS THE CACHE-HIT, CACHE-STORE,

AND CACHE-MISS PERCENTAGES FOR A GIVEN OCAPI REQUEST?

No. At this time, the Salesforce B2C Commerce platform does not provide any visibility into the cache performance of OCAPI requests. The existing cache reporting and visualizations available in the Technical Reports and Reports and Dashboards views only describe the caching behavior of storefront pipeline request behavior.