Google Cloud

Edge Advanced Mediation

Caching

Why Use Caching

Performance

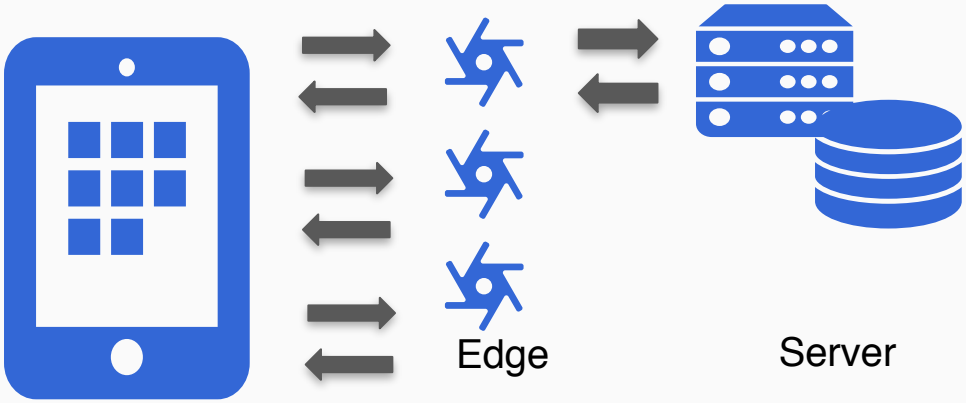
- reducing network latency
- eliminate redundant request/response processing

Stability

- reduce amount of load to backend services

Scalability

- support higher TPS without adding additional hardware

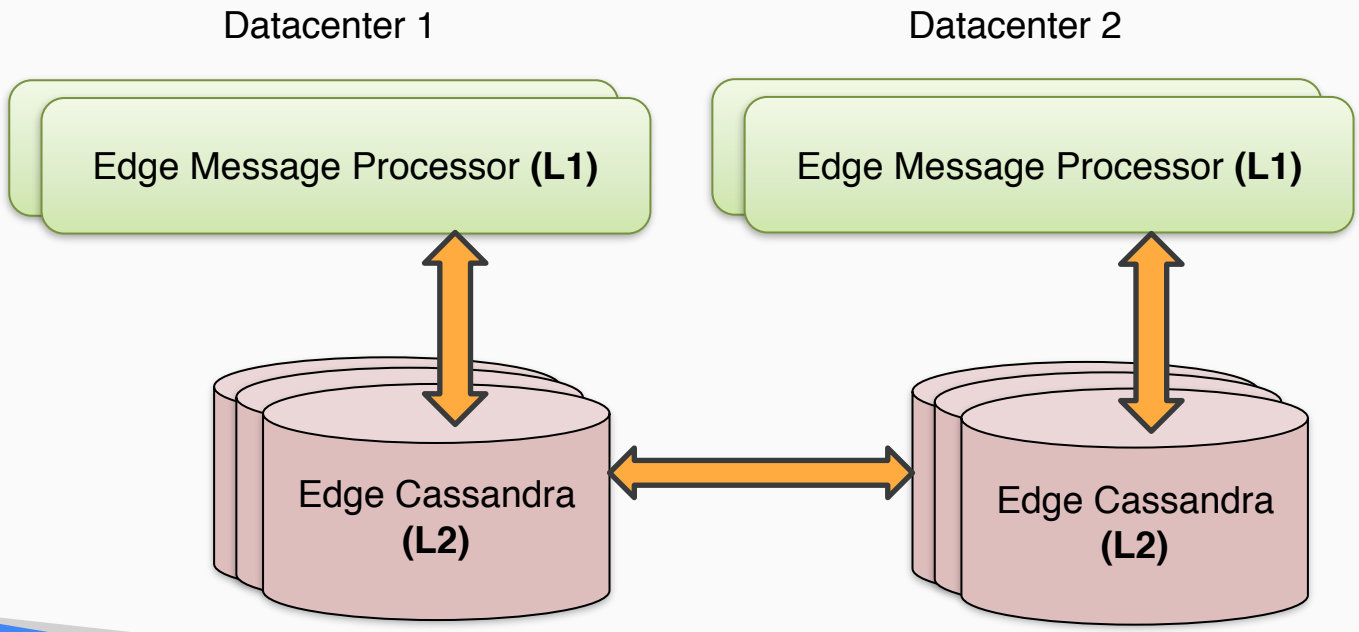


Caching Policies

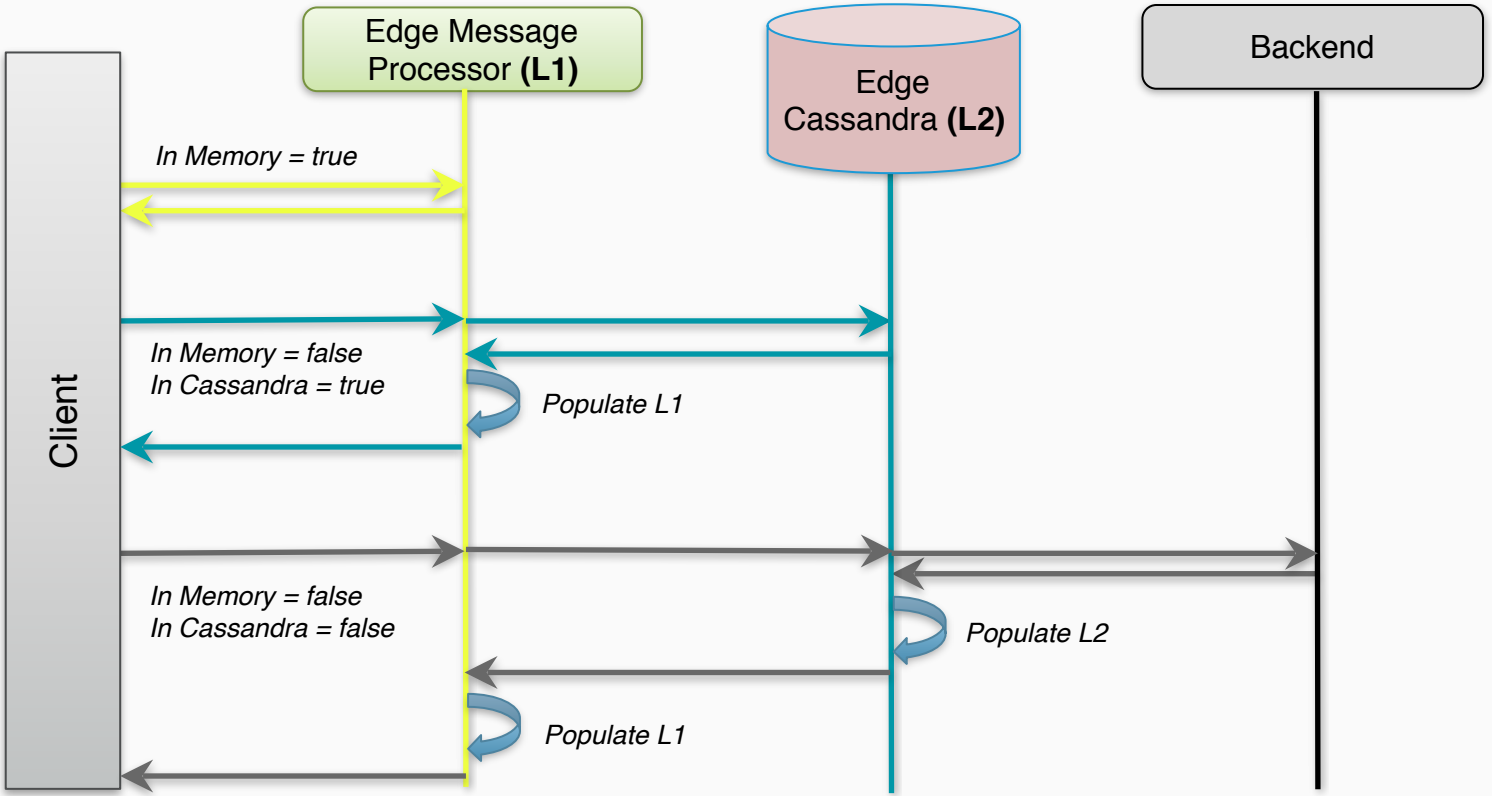
- Response Cache
  - caches the entire HTTP response (headers, payload, etc.)
  - can set a different time-to-live for each entry
  - option for honoring HTTP cache headers for dynamic TTL (Expires, Cache-Control)
  - option to support caching of multiple formats based on request ‘Accept’ header
- Populate Cache/Lookup Cache
  - full control over caching, store any objects
  - add/update and read entries using separate policies

Distributed Caching

Distributed caching allows cache entries to be distributed across multiple Message Processors within a region and across regions.



Cache Lookup/Populate Sequence



Response Cache

Reduces latency and network traffic by avoiding calls to backend and extra processing.

Cache Key

- can use multiple key fragments
- include a unique user identifier if user data is cached
- scoping affects the cache key

Expiration

- can specify a time of day or date the cache entry expires
- can set TTL based on backend response HTTP cache headers (UseResponseCacheHeaders configuration)

Response Cache (cont'd)

optional conditions for skipping cache lookup or cache population

same policy attached to request and response segments

- when request segment policy is reached, cache lookup happens
  - if cache hit, response is retrieved from cache and processing bypasses backend and other policies until ResponseCache policy in Response segment
  - if cache miss, request processing continues
- when response segment policy is reached, cache population happens and response message is stored

Response Cache (cont'd)

```
<ResponseCache name="responseCacheForStores">
 <CacheResource>stores_cache</CacheResource>
    <Scope>Application</Scope>
    <CacheKey>
        <KeyFragment ref="request.queryparam.nearby" />
        <KeyFragment ref="request.queryparam.range" />
        <KeyFragment ref="request.queryparam.limit" />
        <KeyFragment ref="request.queryparam.offset" />
        <KeyFragment ref="request.queryparam.storetype" />
        <KeyFragment ref="request.queryparam.capabilities" />
        <KeyFragment ref="request.queryparam.locale" />
        <KeyFragment ref="storeId" />
    </CacheKey>
    <UseAcceptHeader>true</UseAcceptHeader>
    <UseResponseCacheHeaders>true</UseResponseCacheHeaders>
    <SkipCacheLookup>(request.header.Skip-Cache Equals "true") or (request.verb NotEquals "GET")</SkipCacheLookup>
    <SkipCachePopulation>(request.header.Skip-Cache Equals "true") or (request.verb NotEquals "GET")</
    SkipCachePopulation>
    <ExpirySettings>
        <TimeOfDay>10:00:00</TimeOfDay>
    </ExpirySettings>
</ResponseCache>
```

Populate Cache / Lookup Cache

Optimizes API performance by reducing request or response processing.

- can cache any object. Typically used after building custom data
- separate policies for cache population vs. lookup
- specify a time-to-live
- full control of population and lookup of cache via policies

Choosing which caching policy to use…

- Use Response Cache:
  - when identical requests and their corresponding identical responses are common (cache hits will be frequent)
  - to reduce unnecessary traffic to backend
  - to reduce latency for common requests
- Use Populate Cache / Lookup Cache:
  - when storing custom data objects (not always backend http response) to store data sets that have a specific maximum number of entries (vs. key value maps)
  - to persist data across multiple API transactions

Cache Utilization and Optimization

The use of caching is only beneficial if requests are being served from cache. There are key things to ensure your cache solution is obtaining optimal performance.

- Ensure your cache key is built in such a way that the only request parameters used for the key are ones that dictate the client's response. (e.g. don't just use the URI as things like timestamps and other unique items can end up as part of the key resulting in 0% cache hits.
- Take advantage of the cache scope. Multiple APIs might utilize the same cache so set the scope to be 'organization' and re-use.
- Cache performance dashboard is available in Apigee Analytics to report on cache hit % and performance gains.

Clearing the cache

Using the UI
- allows you to clear the entries for a complete cache resource
- typically used to invalidate all cache entries

Management API
- same as UI with the addition of being able to specify a prefix used for cache keys
- typically used to invalidate all cache entries

Invalidate Cache Policy
- allows you build a fully customized API resource to invalidate specific entries
- used in implementations to invalidate specific cache entries

Lab

THANK YOU