Aug, 2015

Foundational Training

Caching with Edge

Why use Caching?

Performance

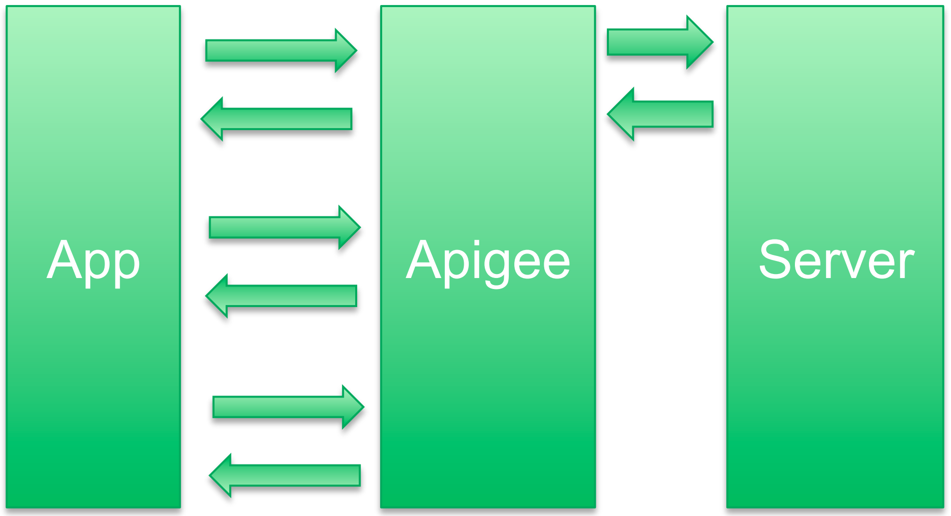
- Reducing Network latency
- Eliminate redundant request/response processing

Stability

- Reduce amount of load to backend services

Scalability

- Support higher TPS without adding additional hardware



Caching Policies

2 different ways to populate/lookup cache data in Apigee

Response Cache

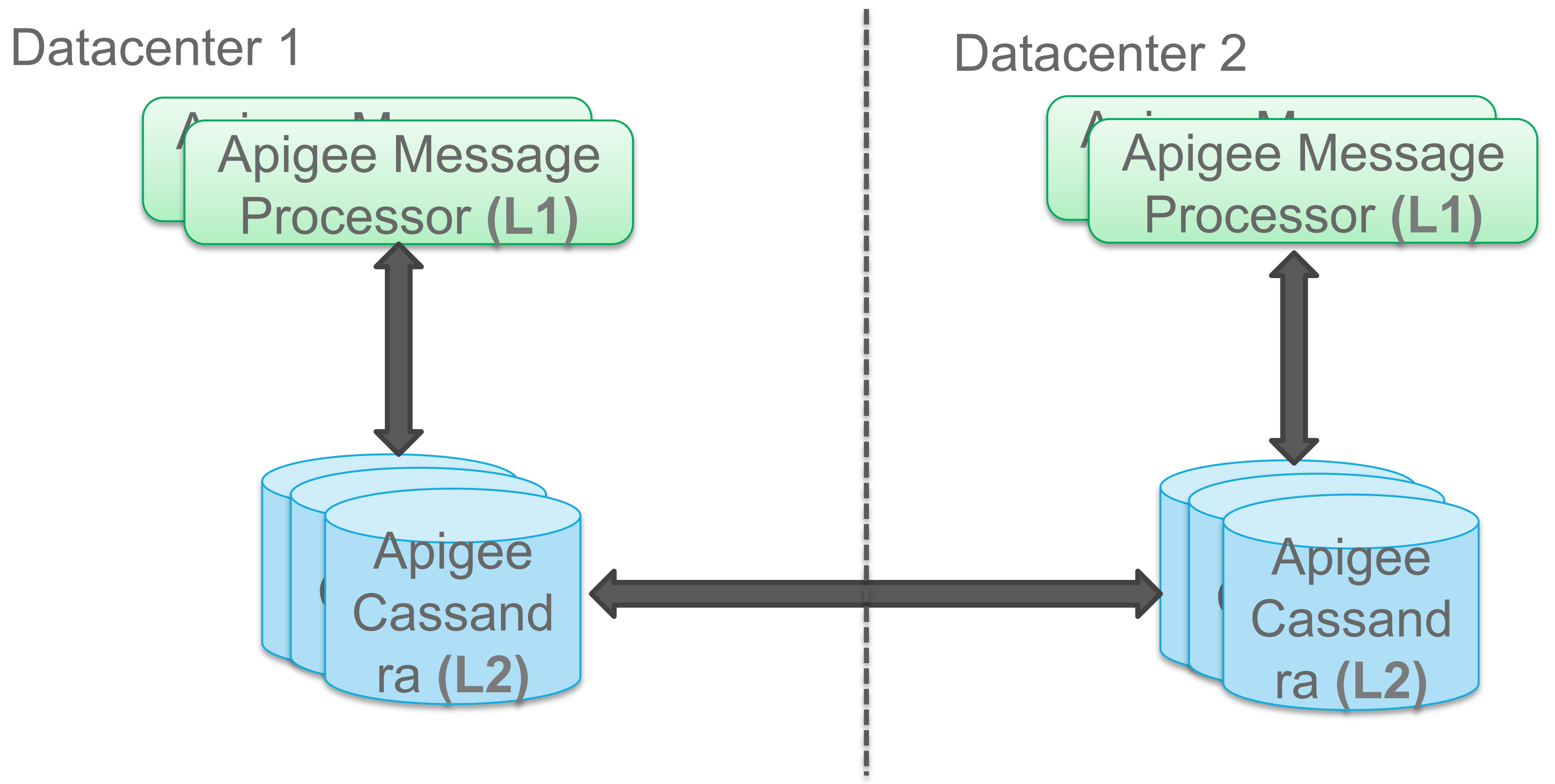
- caches the entire HTTP response (headers, payload, etc.)
- can set a different time-to-live for each entry
- option for honoring HTTP cache headers for dynamic TTL (Expires, Cache-Control)
- option to support caching of multiple formats based on request ‘Accept’ header

Populate Cache/Lookup Cache

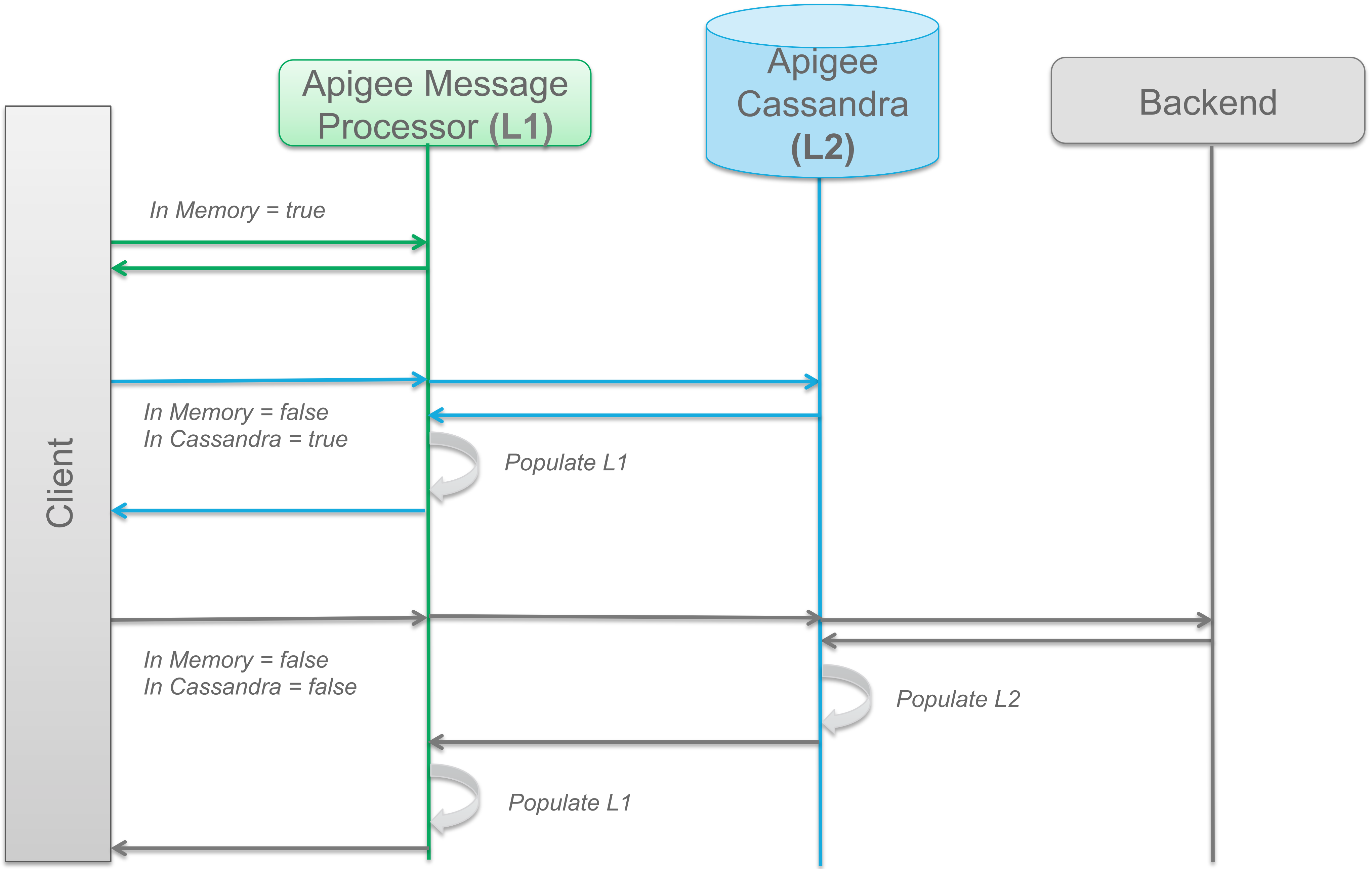
- full control over caching, store any objects
- Add/update and read entries using separate policies

Distributed Caching

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



Cache Lookup/Populate Sequence



ResponseCache

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

Cache Key

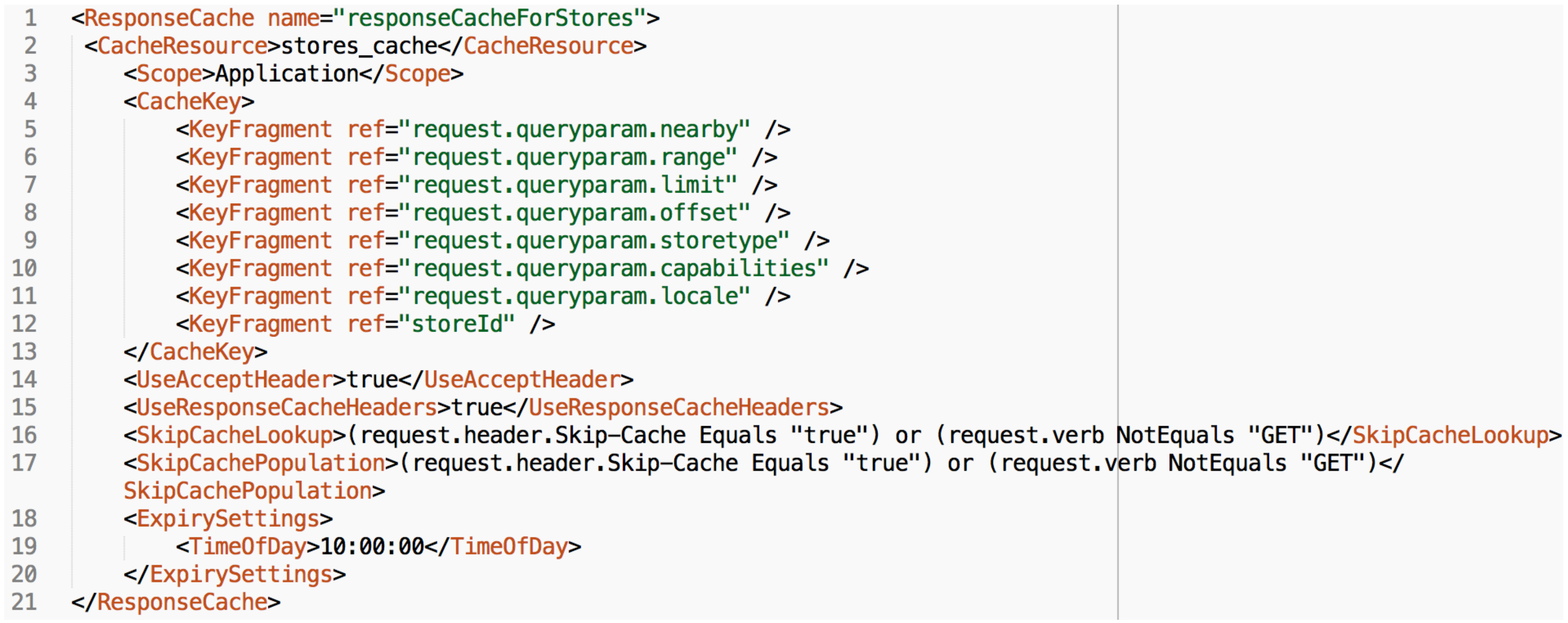
- can use multiple key fragments
- include a unique user identifier if user data is cached
- scoping affects the cache key

Expiration

- can specify a time of day or date the cache entry expires
- Leverage <UseResponseCacheHeaders> configuration so the policy dynamically sets TTL based on backend response HTTP cache headers.

ResponseCache (cont'd)

- Optional conditions for skipping cache lookup or 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



```xml
<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>
```

PopulateCache/LookupCache

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 ResponseCache:

- 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 PopulateCache/LookupCache:

- 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

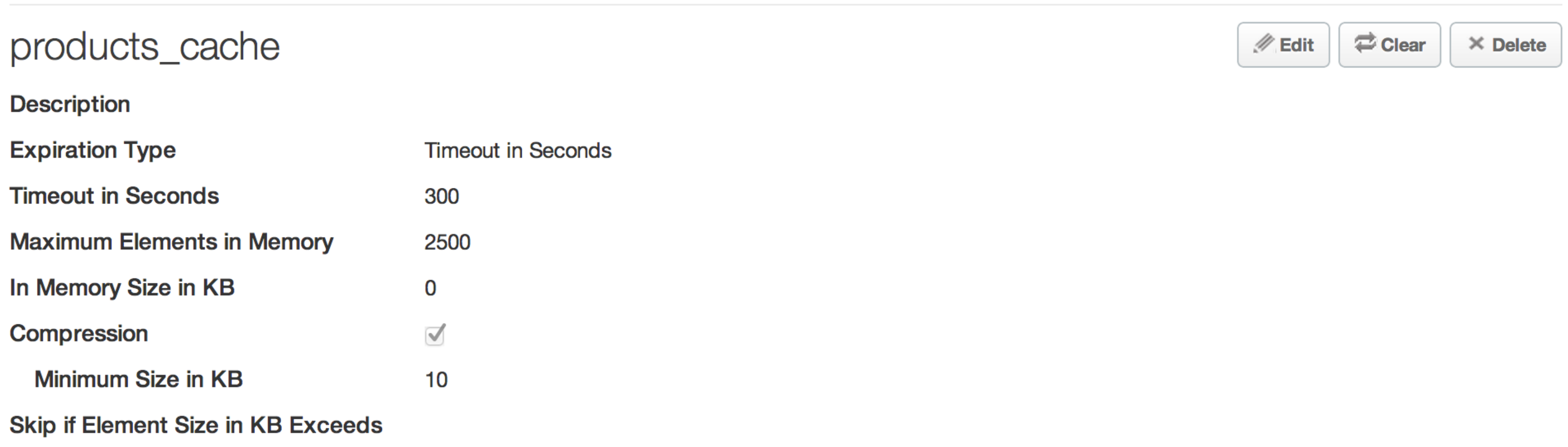
Cache Utilization and Optimization

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.

Cache Resources

Resource configured via UI or management API to define default cache settings.



products_cache

Description	
Expiration Type	Timeout in Seconds
Timeout in Seconds	300
Maximum Elements in Memory	2500
In Memory Size in KB	0
Compression	✓
Minimum Size in KB	10
Skip if Element Size in KB Exceeds	

- **Maximum Elements in Memory** – defines the number of objects held in Apigee servers' memory(L1). This can be used for tuning as there is a finite limit of server memory vs. unlimited storage in (L2). L2 is defined later.
- **Compression** – this setting enables compression on the object to be cached and allows you to define at which point to start compressing the object based on its size. Useful as small objects or already compressed data may not benefit from compression, thus poorly utilizing resources.
- **Skip If Element in KB exceeds** – option to skip caching an object if it is larger than the size specified. Useful if objects are large in size and could take up more L1 memory, reducing the # of objects that can be cached overall.

Clearing the cache

Using the UI

- Allows you to clear the entries for a complete cache resource
- Typically used to invalidate all cache entries

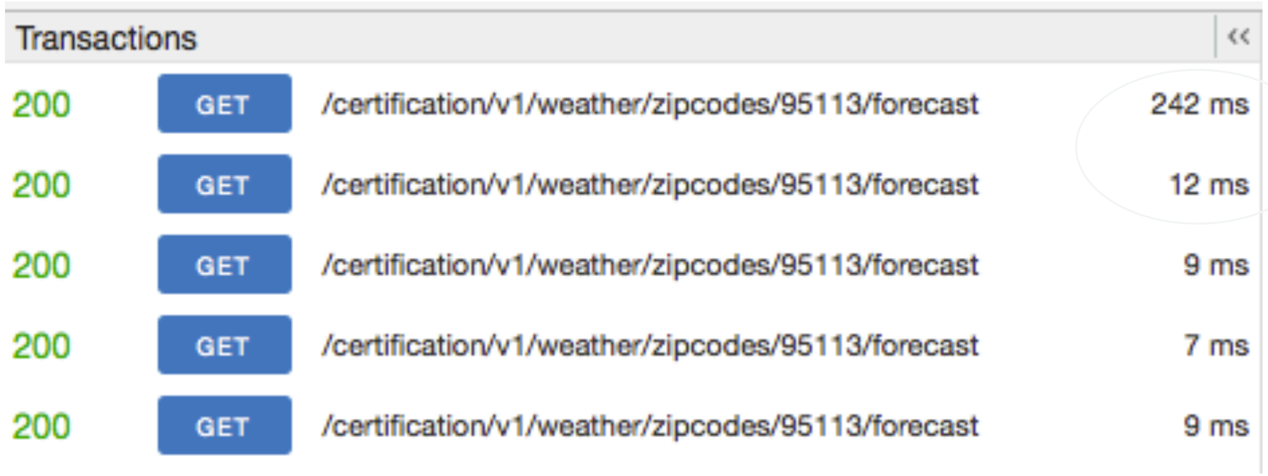
Mgmt API

- Same as UI with the addition of being able to specify a prefix used for cache keys
- Typically used to invalidate all cache entries

InvalidateCache Policy

- Allows you build a fully customized API resource to invalidate specific entries
- Used in implementations to invalidate specific cache entries

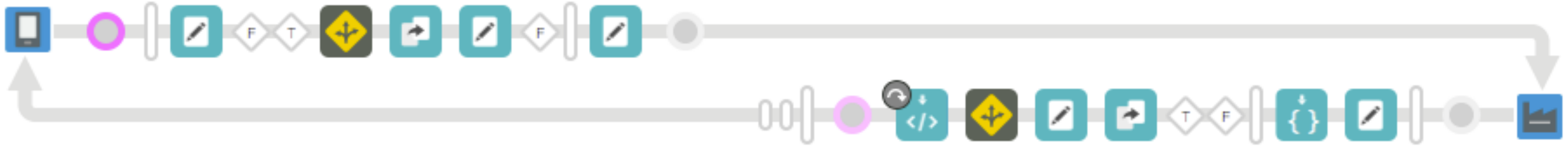
Identify Caching within Trace



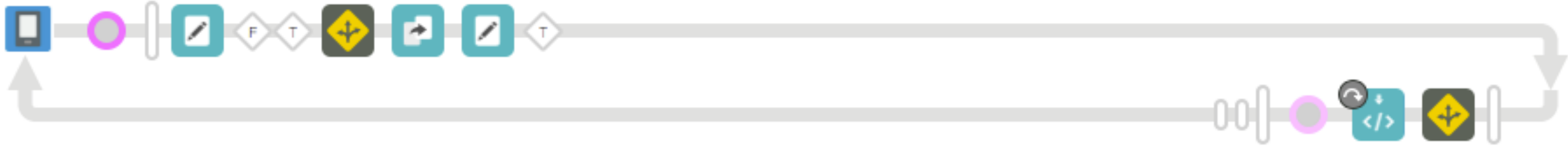
Transactions			
200	GET	/certification/v1/weather/zipcodes/95113/forecast	242 ms
200	GET	/certification/v1/weather/zipcodes/95113/forecast	12 ms
200	GET	/certification/v1/weather/zipcodes/95113/forecast	9 ms
200	GET	/certification/v1/weather/zipcodes/95113/forecast	7 ms
200	GET	/certification/v1/weather/zipcodes/95113/forecast	9 ms

Major drop in response times

Not in Cache – all policies and trip to backend



Cache hit – fewer policies, no backend



Day 1 – Lab 6 Response Cache

- Create a Response Cache Policy on ratings resource
  - The policy will attach itself to target postflow as well
  - Update the policy with cachekey as follows:
    - <KeyFragment ref="proxy.pathsuffix" />

```
<ExpirySettings>
        <TimeoutInSec>30</TimeoutInSec>
 </ExpirySettings>
```

- Test using postman to see if cache hits happen. Look at response time

Day 1 – Lab 7 Cache Demo

- DEMO APIEatery Lookup Cache
- DEMO APIEatery Populate Cache

Thank you

January 2014