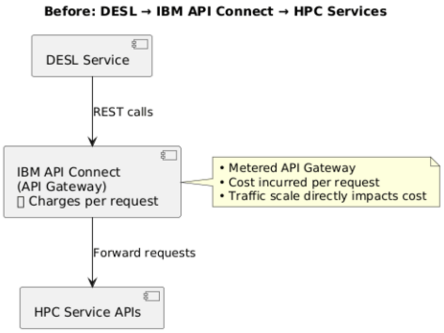
HPC Proxy Service Over REDIS Cache

This document provides a detailed technical overview of the HPC Proxy Service operating with Redis as a distributed caching layer (See the functional requirement design doc: Technical Design Specification - DESL HPC proxy service over Redis). It outlines the end-to-end request flow, including cache key design, read/write patterns, TTL configuration. The document also describes how the service interacts with downstream systems on cache misses, concurrency considerations, failure and fallback handling, and Redis-related configurations across environments. Performance implications, consistency trade-offs, and operational considerations such as monitoring and deployment are also covered to support development and troubleshooting.

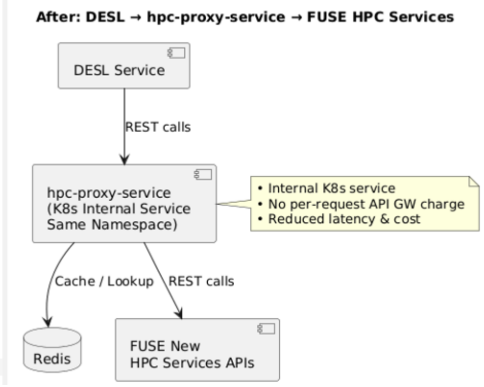
WHY HPC Service Integration



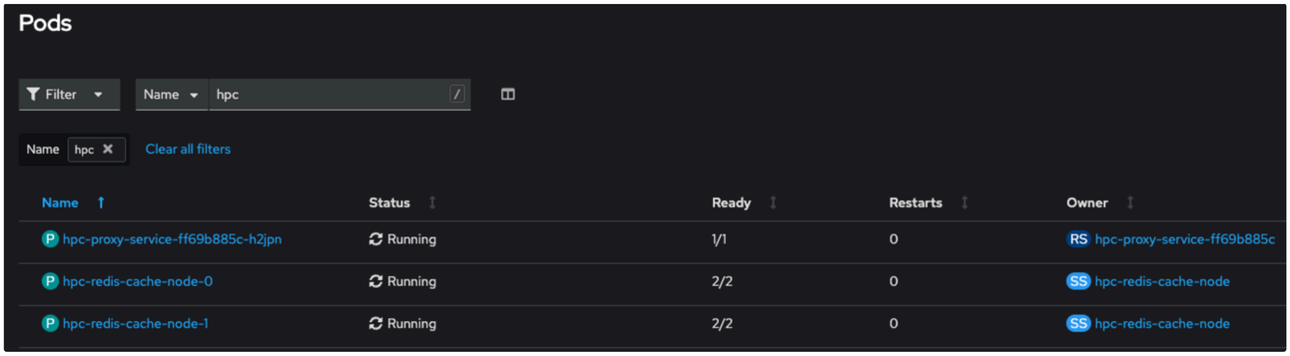
Problems:

DESL → IBM API Connect → HPC APIs

- Internal east-west traffic is treated the **same as external traffic**
- High-frequency internal calls (sometimes duplicated internal HPC API calls from the same/different DESL services) = **unexpected cost growth**
- Caching *behind* the gateway doesn't reduce gateway cost



How hpc-proxy-service Talks to Redis



Pods

Name	Status	Ready	Restarts	Owner
P hpc-proxy-service-ff69b885c-h2jpn	Running	1/1	0	RS hpc-proxy-service-ff69b885c
P hpc-redis-cache-node-0	Running	2/2	0	SS hpc-redis-cache-node
P hpc-redis-cache-node-1	Running	2/2	0	SS hpc-redis-cache-node

No Direct Pod-to-Pod Coupling

Although Redis pods are visible as:

- `hpc-redis-cache-node-0`
- `hpc-redis-cache-node-1`
- `hpc-proxy-service` **does NOT connect to these pods directly.**

Instead, it connects via Kubernetes Services backed by Redis Sentinel.

Kubernetes Services Used

Service	Type	Purpose
`hpc-redis-cache`	ClusterIP	Stable access to Redis & Sentinel
`hpc-redis-cache-headless`	Headless	Pod-level DNS for StatefulSet

- **Port 6379** → Redis data
- **Port 26379** → Redis Sentinel

```
kubectl get services | grep hpc-redis
hpc-redis-cache                             ClusterIP   10.221.136.60    <none>        6379/TCP,26379/TCP   92d
hpc-redis-cache-headless                    ClusterIP   None             <none>        6379/TCP,26379/TCP   92d
```

Runtime Connection Flow

Step 1: Sentinel Discovery

- `hpc-proxy-service` connects to:

```
hpc-redis-cache:26379

```

- This reaches one of the Sentinel containers running inside:
  - `hpc-redis-cache-node-0`
  - `hpc-redis-cache-node-1`

Sentinel responds with:

- Current **Redis master**
- Available **replicas**

Step 2: Redis Operations

Based on Sentinel metadata:

- **Writes**
  - Routed to the current Redis **master**

- **Reads**
  - Served from master (or replicas if configured)

The application does not need to know whether:

- `node-0` or `node-1` is the master
- A failover has occurred

Role of the Headless Service

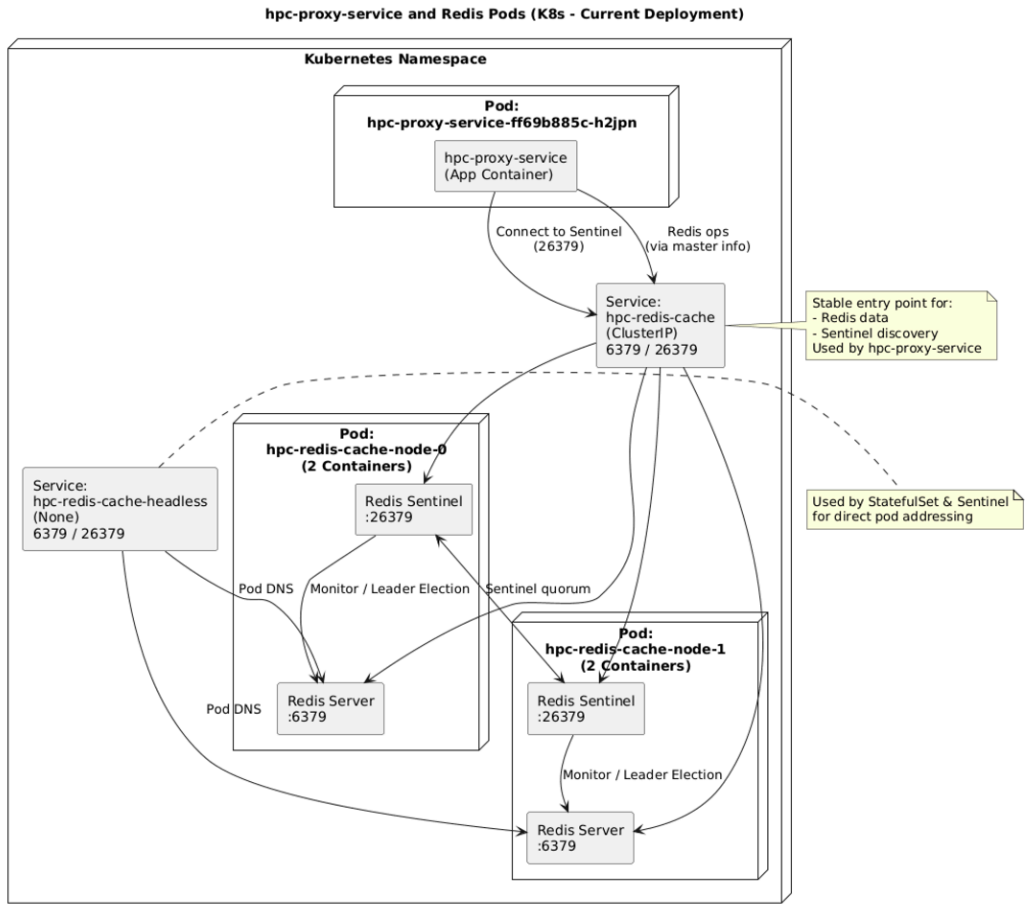
The headless service enables DNS names like:

```
hpc-redis-cache-node-0.hpc-redis-cache-headless
hpc-redis-cache-node-1.hpc-redis-cache-headless
```

These are used by **Sentinel**, not by the application, to:

- Monitor Redis health
- Perform leader election
- Coordinate failover

This is a standard Kubernetes pattern for **stateful HA systems**.



Redis Customer Profile Cache Key Structure (HPC Standard)

The **CustomerProfileCacheKeyBuilder** generates canonical **Redis cache keys** for the **HPC Proxy – Customer Profile** flow. These keys are used by **DESL-facing services** when interacting with the HPC Proxy to ensure consistent cache hits across identical request scopes.

Within the HPC architecture, the cache key represents the *request contract*, not the caller. This guarantees that multiple DESL services requesting the same logical customer profile data will resolve to the same Redis entry.

The builder enforces a **stable, deterministic key format** to prevent cache fragmentation, over-fetching from HPC downstream systems, and unintended cache misses.

The generated cache key is a single string composed of fixed segments, separated by the pipe character (`|`). Each segment is expressed as a `key:value` pair.

```
customerNumbers:<value>|accountNumbers:<value>|lineNumbers:<value>|includePaymentMethods:<value>
```

Segment order is fixed and must not be changed. Redis key lookup is string-exact, and any deviation will result in a cache miss, as cache lookup depends on exact string matching.

Normalisation Rules (Redis Safety)

To guarantee deterministic output, **all blank or missing identifiers are normalised to the literal string** `"null"`.

This applies to:

- `customerNumbers`
- `accountNumbers`
- `lineNumbers`

Normalisation rules:

- `null`, empty string, or whitespace → `"null"`
- Explicit values are preserved verbatim

This ensures that logically identical requests always generate identical cache keys. The following are examples:

```
customerNumbers:123|accountNumbers:null|lineNumbers:null|includePaymentMethods:Y
```

```
customerNumbers:null|accountNumbers:456|lineNumbers:null|includePaymentMethods:N
```

```
customerNumbers:null|accountNumbers:null|lineNumbers:789|includePaymentMethods:Y
```

```
customerNumbers:123|accountNumbers:456|lineNumbers:789|includePaymentMethods:Y
```

HPC Cache Design Considerations

- **Immutability of format**: Any change to key structure requires cache invalidation.
- **Human readability**: Keys are intentionally verbose to aid debugging and Redis inspection.
- **Extensibility**: New segments should only be added after careful assessment of cache impact.
- **Logging safety**: Keys contain identifiers but no sensitive payment data.

Change & Extension Guidelines (Redis Impact)

If additional dimensions are required in the future:

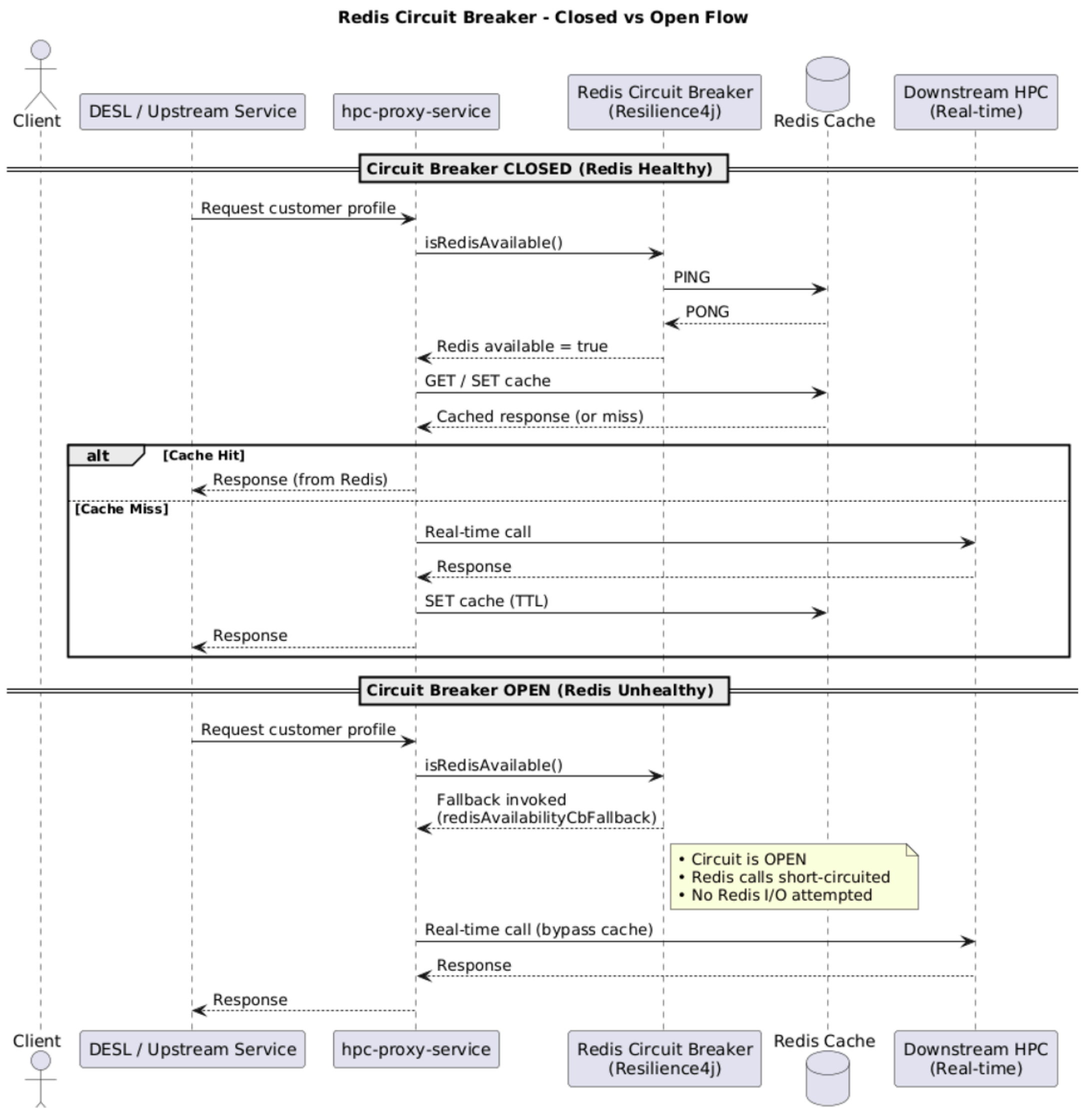
1. Add a new fixed segment at the end of the key
2. Default missing values to `"null"`
3. Update all consumers simultaneously
4. Flush or version existing caches

Failure to follow these rules may result in cache pollution or stale reads.

How Circuit Breaker of REDIS works

The **Redis Circuit Breaker** is implemented using **Resilience4j** to protect `hpc-proxy-service` from Redis-related failures such as timeouts, connection errors, or high contention. Its primary goal is to prevent Redis issues from cascading upstream and impacting real-time HPC requests. Redis is treated as an **optional performance optimisation**, not a critical dependency. When Redis is unhealthy, the service **fails fast and bypasses cache usage**, falling back to real-time data retrieval.

The following sequence diagram shows how `hpc-proxy-service` behaves when the Redis Circuit Breaker is **closed (healthy)** versus **open (unhealthy)**.
The key difference is whether Redis is consulted or bypassed entirely.



Where the Circuit Breaker Is Applied

The Circuit Breaker is applied to the Redis health check method, in `HpcProxyServiceHelper`

```
@Override
@CircuitBreaker(name = "REDIS", fallbackMethod = "redisAvailabilityCbFallback")
public boolean isRedisAvailable()
{
   try (var connection = redisConnectionFactory.getConnection()) {
      return connection.ping() != null;
   }
}
```

This method performs a lightweight `PING` against Redis using a pooled connection. Any exception (e.g. connection failure, timeout) is recorded by the Circuit Breaker.

Behaviour by Circuit Breaker State

Closed (Normal Operation)

- Redis `PING` is executed normally.
- Successful calls return `true`.
- Failures are tracked within the sliding window.

Redis cache reads and writes are allowed.

Open (Redis Unhealthy)

When the failure rate exceeds the configured threshold:

- The circuit transitions to **OPEN**
- Redis calls are **short-circuited**
- The fallback method is invoked immediately

```
    private boolean redisAvailabilityCbFallback(final Throwable throwable)
    {
        if (throwable != null) {
            LOGGER.warn("REDIS CircuitBreaker triggered — Redis unavailable.", throwable);
        } else {
            LOGGER.warn("REDIS CircuitBreaker open — Redis unavailable.");
        }
        return false;
    }
```

Behaviour:

- Returns `false` deliberately
- Redis is treated as unavailable
- Cache reads and writes are skipped
- Requests continue via real-time HPC paths

This ensures:

- No thread blocking
- No retry storms
- No increased latency due to Redis slowness

Half-Open (Recovery Probe)

After the open-state wait period:

- A small number of Redis calls are allowed
- If they succeed → circuit closes
- If they fail → circuit reopens

This enables **automatic recovery without redeployment**.

Fallback Design Rationale

The fallback always returns:

```
return false;
```

This is intentional.

Why?

- Redis availability checks are used as a **gate** for cache usage
- Returning `false` ensures callers:
  - Fail fast
  - Avoid blocking on unhealthy Redis
  - Maintain predictable latency

Logging behaviour:

- Exceptions are logged when present
- Open-circuit events are logged separately
- This aids operational visibility without excessive noise

Circuit Breaker Configuration (REDIS)

```
# ------------------------------------------------------------------------
# Resilience4j Configuration for Redis
#
# Purpose:
#  - Protect hpc-proxy-service from cascading failures when Redis is slow,
#    unavailable, or under high contention.
#  - Ensure that cache issues never block or degrade real-time HPC calls.
#
```

```
# --- Circuit Breaker ---
# This CB monitors Redis operations (cache get/put) and opens temporarily
# when repeated failures occur, allowing the service to bypass Redis safely.
#
# - slidingWindowSize=20:
#   Checks the last 20 Redis calls to measure failure rate.
#   Smaller window => quicker detection for transient Redis issues.
#
# - failureRateThreshold=50:
#   If more than 50% of Redis calls in the last window failed, open the CB.
#
# - waitDurationInOpenState=3s:
#   Keep the circuit open (blocking Redis calls) for 3 seconds before retrying.
#
# - permittedNumberOfCallsInHalfOpenState=5:
#   When half-open, allow 5 test calls to Redis to see if it has recovered.
#
# ------------------------------------------------------------------------
resilience4j.circuitbreaker.instances.REDIS.slidingWindowSize=20
resilience4j.circuitbreaker.instances.REDIS.failureRateThreshold=50
resilience4j.circuitbreaker.instances.REDIS.waitDurationInOpenState=3s
resilience4j.circuitbreaker.instances.REDIS.permittedNumberOfCallsInHalfOpenState=5
```

Functional Tests:

Step 1: Redis cache is empty, issue `curl -Sv "http://localhost:8080/api/hpc/v1/customerProfiles?customerNumbers=751016026"` in hpc-proxy-service pod in INT03, the following result is returned:

```
curl -Sv "http://localhost:8080/api/hpc/v1/customerProfiles?customerNumbers=751016026"
*   Trying ::1:8080...
* Connected to localhost (::1) port 8080 (#0)
> GET /api/hpc/v1/customerProfiles?customerNumbers=751016026 HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.76.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Wed, 21 Jan 2026 02:39:40 GMT
<
{"messages":[{"message":"Success","code":2000}]...}
```

Step 2: Check server log, you will see the request was sent to FUSE HPC customer profile endpoint: `cache-customer-profile-v1-intg-int03.apps.npe04.ocp.internal.spark.co.nz`. The log message also states there is a “Cache miss”.

```
2026-01-21 15:39:40,147 INFO [tomcat-handler-285] [HpcProxyService] [] [] [DASL7848873130346945] [nz.co.spark.dasl.base.rest.RequestIdAdvice] -
Request for GET http://localhost:8080/api/hpc/v1/customerProfiles
2026-01-21 15:39:40,226 INFO [tomcat-handler-285] [HpcProxyService] [] [] [DASL7848873130346945] [nz.co.spark.dasl.utility.hpcproxy.services.HpcProxyServiceImpl] -
Cache miss — no cached HPC customer profile data for combined customerNumbers: 751016026, accountNumbers: null, lineNumbers: null and includePaymentMethod: N
2026-01-21 15:39:40,230 INFO [tomcat-handler-285] [HpcProxyService] [] [] [DASL7848873130346945] [nz.co.spark.dasl.webservice.rest.AbstractRestFactory] -
Connection pool for http://cache-customer-profile-v1-intg-int03.apps.npe04.ocp.internal.spark.co.nz/api/hpc/v1/customerProfiles?customerNumbers=751016026&includePaymentMethods=N, 0 of 200 (idle 1) leased; 0 waiting
2026-01-21 15:39:40,230 INFO [tomcat-handler-285] [HpcProxyService] [] [] [DASL7848873130346945] [nz.co.spark.dasl.webservice.rest.RestErrorHandler] -
Performing request to GET http://cache-customer-profile-v1-intg-int03.apps.npe04.ocp.internal.spark.co.nz/api/hpc/v1/customerProfiles?customerNumbers=751016026&includePaymentMethods=N
2026-01-21 15:39:40,231 INFO [tomcat-handler-285] [HpcProxyService] [] [] [DASL7848873130346945] [nz.co.spark.dasl.webservice.interceptors.FuseHeaderInterceptor] -
Request will use transaction id ONL574165531250541017347625051601444620 0
...
2026-01-21 15:39:40,331 INFO [tomcat-handler-285] [HpcProxyService] [] [] [DASL7848873130346945] [nz.co.spark.dasl.webservice.rest.RestErrorHandler] -
Received 200 OK response from GET http://cache-customer-profile-v1-intg-int03.apps.npe04.ocp.internal.spark.co.nz/api/hpc/v1/customerProfiles?customerNumbers=751016026&includePaymentMethods=N (101ms)
```

Step 3: Now there is an entry in Redis cache for customerNumber 751016026. Issue the `http://localhost:8080/api/hpc/v1/customerProfiles?customerNumbers=751016026` command again, you’ll see a “Cache Hit” in server log, and get the same response.

```
curl -Sv "http://localhost:8080/api/hpc/v1/customerProfiles?customerNumbers=751016026"
```

```
*   Trying ::1:8080...
* Connected to localhost (::1) port 8080 (#0)
> GET /api/hpc/v1/customerProfiles?customerNumbers=751016026 HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.76.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Wed, 21 Jan 2026 02:39:40 GMT
<
{"messages":[{"message":"Success","code":2000}]...}
```

Server log:

```
2026-01-21 15:47:53,045 INFO [tomcat-handler-387] [HpcProxyService] [] [] [DASL3022550460723082]
[nz.co.spark.dasl.base.rest.RequestIdAdvice] -
Request for GET http://localhost:8080/api/hpc/v1/customerProfiles
2026-01-21 15:47:53,122 INFO [tomcat-handler-387] [HpcProxyService] [] [] [DASL3022550460723082]
[nz.co.spark.dasl.utility.hpcproxy.services.HpcProxyServiceImpl] -
Cache hit – found cached HPC customer profile data for customerNumbers: 751016026, accountNumbers: null, lineNumbers: null and
includePaymentMethod: N
2026-01-21 15:47:53,125 INFO [tomcat-handler-387] [HpcProxyService] [] [] [DASL3022550460723082]
[nz.co.spark.dasl.base.rest.LoggingJsonMapper] -
Outbound entity is {"messages":[{"message":"Success","code":2000}]...}
```

Step 4: Now test “realTime” query parameter: `curl -Sv "http://localhost:8080/api/hpc/v1/customerProfiles?realTime=true&customerNumbers=751016026"`

When `realTime = true`, the real-time data will be returned from FUSE HPC:

```
curl -Sv "http://localhost:8080/api/hpc/v1/customerProfiles?realTime=true&customerNumbers=751016026"
*   Trying ::1:8080...
* Connected to localhost (::1) port 8080 (#0)
> GET /api/hpc/v1/customerProfiles?realTime=true&customerNumbers=751016026 HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.76.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Wed, 21 Jan 2026 02:49:56 GMT
<
{"messages":[{"message":"Success","code":2000}]...}
```

Server log:

```
2026-01-21 15:49:56,444 INFO [tomcat-handler-413] [HpcProxyService] [] [] [DASL1933901551247731]
[nz.co.spark.dasl.base.rest.RequestIdAdvice] -
Request for GET http://localhost:8080/api/hpc/v1/customerProfiles
2026-01-21 15:49:56,445 INFO [tomcat-handler-413] [HpcProxyService] [] [] [DASL1933901551247731]
[nz.co.spark.dasl.utility.hpcproxy.services.HpcProxyServiceImpl] -
Fetching real-time hpc customer profile from HPC
2026-01-21 15:49:56,446 INFO [tomcat-handler-413] [HpcProxyService] [] [] [DASL1933901551247731]
[nz.co.spark.dasl.webservice.rest.AbstractRestFactory] -
Connection pool for http://cache-customer-profile-v1-intg-int03.apps.npe04.ocp.internal.spark.co.nz/api/hpc/v1/customerProfiles?
customerNumbers=751016026&includePaymentMethods=N, 0 of 200 (idle 1) leased; 0 waiting
2026-01-21 15:49:56,447 INFO [tomcat-handler-413] [HpcProxyService] [] [] [DASL1933901551247731]
[nz.co.spark.dasl.webservice.rest.RestErrorHandler] -
Performing request to GET http://cache-customer-profile-v1-intg-int03.apps.npe04.ocp.internal.spark.co.nz/api/hpc/v1/customerProfiles?
customerNumbers=751016026&includePaymentMethods=N
2026-01-21 15:49:56,447 INFO [tomcat-handler-413] [HpcProxyService] [] [] [DASL1933901551247731]
[nz.co.spark.dasl.webservice.interceptors.FuseHeaderInterceptor] -
Request will use transaction id ONL5249318234689689930687903948000530102
...
2026-01-21 15:49:56,587 INFO [tomcat-handler-413] [HpcProxyService] [] [] [DASL1933901551247731]
[nz.co.spark.dasl.webservice.rest.RestErrorHandler] -
Received 200 OK response from GET http://cache-customer-profile-v1-intg-
int03.apps.npe04.ocp.internal.spark.co.nz/api/hpc/v1/customerProfiles?customerNumbers=751016026&includePaymentMethods=N (140ms)
```

When `realTime=false` (which is the default value when realTime parameter is not present), the cached value will be returned if there’s a matched entry in the cache.

```
curl -Sv "http://localhost:8080/api/hpc/v1/customerProfiles?realTime=false&customerNumbers=751016026"
*   Trying ::1:8080...
* Connected to localhost (::1) port 8080 (#0)
> GET /api/hpc/v1/customerProfiles?realTime=false&customerNumbers=751016026 HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.76.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Wed, 21 Jan 2026 02:53:36 GMT
<
```

Server log:

```
2026-01-21 15:53:35,994 INFO [tomcat-handler-459] [HpcProxyService] [] [] [DASL2992127148012373] [nz.co.spark.dasl.base.rest.RequestIdAdvice] -
Request for GET http://localhost:8080/api/hpc/v1/customerProfiles
2026-01-21 15:53:36,732 INFO [tomcat-handler-459] [HpcProxyService] [] [] [DASL2992127148012373] [nz.co.spark.dasl.utility.hpcproxy.services.HpcProxyServiceImpl] -
Cache hit — found cached HPC customer profile data for customerNumbers: 751016026, accountNumbers: null, lineNumbers: null and includePaymentMethod: N
2026-01-21 15:53:36,826 INFO [tomcat-handler-459] [HpcProxyService] [] [] [DASL2992127148012373] [nz.co.spark.dasl.base.rest.LoggingJsonMapper] -
Outbound entity is {"messages":[{"message":"Success","code":2000}]...}
```

Test cache eviction endpoint:

Issue `curl -X DELETE -Sv "localhost:8080/api/hpc/v1/evict?hpcCacheType=customerprofile"` or `curl -X DELETE -Sv "localhost:8080/api/hpc/v1/evict?hpcCacheType=products"`

```
curl -X DELETE -Sv "localhost:8080/api/hpc/v1/evict?hpcCacheType=customerprofile"
*   Trying ::1:8080...
* Connected to localhost (::1) port 8080 (#0)
> DELETE /api/hpc/v1/evict?hpcCacheType=customerprofile HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.76.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 204
< Date: Wed, 21 Jan 2026 03:38:06 GMT
<
* Connection #0 to host localhost left intact
# Server log:
2026-01-21 16:38:06,518 INFO [tomcat-handler-96] [HpcProxyService] [] [] [DASL3532170268031317] [nz.co.spark.dasl.base.rest.RequestIdAdvice] -
Request for DELETE http://localhost:8080/api/hpc/v1/evict
2026-01-21 16:38:06,519 INFO [tomcat-handler-96] [HpcProxyService] [] [] [DASL3532170268031317] [nz.co.spark.dasl.utility.hpcproxy.services.HpcProxyServiceImpl] -
Evicting customerprofile cache...
2026-01-21 16:38:06,528 INFO [tomcat-handler-96] [HpcProxyService] [] [] [DASL3532170268031317] [nz.co.spark.dasl.base.rest.ThreadContextInterceptor] -
Service response: 204 DELETE http://localhost:8080/api/hpc/v1/evict (12281us)

curl -X DELETE -Sv "localhost:8080/api/hpc/v1/evict?hpcCacheType=products"
*   Trying ::1:8080...
* Connected to localhost (::1) port 8080 (#0)
> DELETE /api/hpc/v1/evict?hpcCacheType=products HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.76.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 204
< Date: Wed, 21 Jan 2026 03:36:56 GMT
<
* Connection #0 to host localhost left intact
# Server log
2026-01-21 16:36:52,921 INFO [tomcat-handler-80] [HpcProxyService] [] [] [DASL5205024490221442] [nz.co.spark.dasl.base.rest.RequestIdAdvice] -
Request for DELETE http://localhost:8080/api/hpc/v1/evict
2026-01-21 16:36:52,922 INFO [tomcat-handler-80] [HpcProxyService] [] [] [DASL5205024490221442] [nz.co.spark.dasl.utility.hpcproxy.services.HpcProxyServiceImpl] -
Evicting products cache...
2026-01-21 16:36:56,712 INFO [tomcat-handler-80] [HpcProxyService] [] [] [DASL5205024490221442] [nz.co.spark.dasl.base.rest.ThreadContextInterceptor] -
Service response: 204 DELETE http://localhost:8080/api/hpc/v1/evict (3794560us)

```

```
# Negative case 1: when "hpcCacheType" is not specified:
curl -X DELETE -Sv "localhost:8080/api/hpc/v1/evict"
*   Trying ::1:8080...
* Connected to localhost (::1) port 8080 (#0)
> DELETE /api/hpc/v1/evict HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.76.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 400
< Content-Type: application/json
< Transfer-Encoding: chunked
< Date: Wed, 21 Jan 2026 02:59:12 GMT
< Connection: close
<
* Closing connection 0
{"messages":[{"message":"evictAllCachedEntries.hpcCacheType (null): must not be null","code":4002}]}

# Negative case 2: when "hpcCacheType" is "unknown" (Neither "customerprofile" nor "products").
curl -X DELETE -Sv "localhost:8080/api/hpc/v1/evict?hpcCacheType=unknown"
*   Trying ::1:8080...
* Connected to localhost (::1) port 8080 (#0)
> DELETE /api/hpc/v1/evict?hpcCacheType=unknown HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.76.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 204
< Date: Wed, 21 Jan 2026 03:10:39 GMT
<
* Connection #0 to host localhost left intact
2026-01-21 15:59:06,920 INFO [tomcat-handler-543] [HpcProxyService] [] [] [DASL9765721224211934] [nz.co.spark.dasl.base.rest.RequestIdAdvice] -
Request for DELETE http://localhost:8080/api/hpc/v1/evict
2026-01-21 15:59:06,921 WARN [tomcat-handler-543] [HpcProxyService] [] [] [DASL9765721224211934] [nz.co.spark.dasl.utility.hpcproxy.services.HpcProxyServiceImpl] -
Unknown cache type: unknown, do nothing.
2026-01-21 15:59:06,922 INFO [tomcat-handler-543] [HpcProxyService] [] [] [DASL9765721224211934] [nz.co.spark.dasl.base.rest.ThreadContextInterceptor] -
Service response: 204 DELETE http://localhost:8080/api/hpc/v1/evict (3192us)
```

Performance sanity comparison: (Note: **this is *not* a reliable way to compare performance**, but it *is* a reasonable **first sanity check**.)

```
time curl -Sv "http://localhost:8080/api/hpc/v1/customerProfiles?realTime=true&customerNumbers=751016026"
real    0m0.224s
user    0m0.003s
sys 0m0.004s

time curl -Sv "http://localhost:8080/api/hpc/v1/customerProfiles?realTime=false&customerNumbers=751016026"
real    0m0.021s
user    0m0.002s
sys 0m0.005s
```