

cache

Motivation

Der Cache soll vorrangig die Microservices entlasten, indem er die Response zwischenspeichert und der Registry für eine gewisse Zeit zur Verfügung stellt. Insbesondere der Service Weather ist davon betroffen, da dieser eine API von [OpenWeatherMap](#) nutzt, welches pro Tag 4000 Wettervorhersagen treffen kann, sonst wird dieser Kostenpflichtig bzw. kann dann keine Requests mehr entgegen nehmen.

Free	Startup 40 USD / month	Developer 180 USD / month	Professional 470 USD / month	Enterprise 2.000 USD / month
60 calls/minute 1,000,000 calls/month	600 calls/minute 10,000,000 calls/month	3,000 calls/minute 100,000,000 calls/month	30,000 calls/minute 1,000,000,000 calls/month	200,000 calls/minute 5,000,000,000 calls/month
Current Weather Minute Forecast 1 hour* Hourly Forecast 2 days* Daily Forecast 7 days* Historical weather 5 days* Climatic Forecast 30 days Bulk Download	Current Weather Minute Forecast 1 hour** Hourly Forecast 2 days** Daily Forecast 16 days Historical weather 5 days** Climatic Forecast 30 days Bulk Download	Current Weather Minute Forecast 1 hour Hourly Forecast 4 days Daily Forecast 16 days Historical weather 5 days Climatic Forecast 30 days Bulk Download	Current Weather Minute Forecast 1 hour Hourly Forecast 4 days Daily Forecast 16 days Historical weather 5 days Climatic Forecast 30 days Bulk Download	Current Weather Minute forecast 1 hour Hourly Forecast 4 days Daily Forecast 16 days Historical weather 5 days Climatic Forecast 30 days Bulk Download
Basic weather maps Historical maps	Basic weather maps Historical maps	Advanced weather maps Historical maps	Advanced weather maps Historical maps	Advanced weather maps Historical maps
Weather triggers	Weather triggers	Weather triggers	Weather triggers	Weather triggers
Weather widgets	Weather widgets	Weather widgets	Weather widgets	Weather widgets
Uptime 95%	Uptime 95%	Uptime 99.5%	Uptime 99.5%	Uptime 99.9%

* - 1,000 API calls per day by using One Call API

** - 2,000 API calls per day by using One Call API

Requirements

Die Registry soll Requests von dem Deconcentrator entgegennehmen und überprüfen, ob diese Request innerhalb einer fest definierten Zeit bereits eine Response erhalten hat. Ist dies der Fall guckt die Registry in den Cache, um sich die dort Zwischengespeicherte Response zu holen und diese an

den Sender der Request zu leiten. Dabei "ersetzt" der Cache den angesprochenen Microservice. Ist dies allerdings nicht der Fall wendet sich die Registry weiter an den angesprochenen Microservice und speichert dessen Response in den Cache.

Functional

- /CAF100/ The system must check if the requested resource is available in the cache before relaying the request to a microservice.
- /CAF100/ The system must place the response of a microservice in the cache.
- /CAF200/ The cache must offer an option to save a response of a microservice.
- /CAF201/ The cache must offer an option to retrieve a saved response.
- /CAF202/ The cache must automatically delete a saved response if the given timeout has been exceeded.

Non Functional

- /CANF100/ The system must answer faster with a cached response than if a request is relayed to a microservice.
- /CANF200/ The cache must save at least 1000 Responses.
- /CANF201/ The cache must answer in at least 5ms.

User Stories

- /CAUS100/ Als Betreiber möchte ich Anfragen die das selbe Ergebnis erzeugen abfangen und damit die Microservices entlasten.
- /CAUS101/ Als Betreiber möchte ich die Anfragen an die verschiedenen APIs reduzieren um nicht in ein teureres Preispaket zu fallen.

Use Cases

Technologies

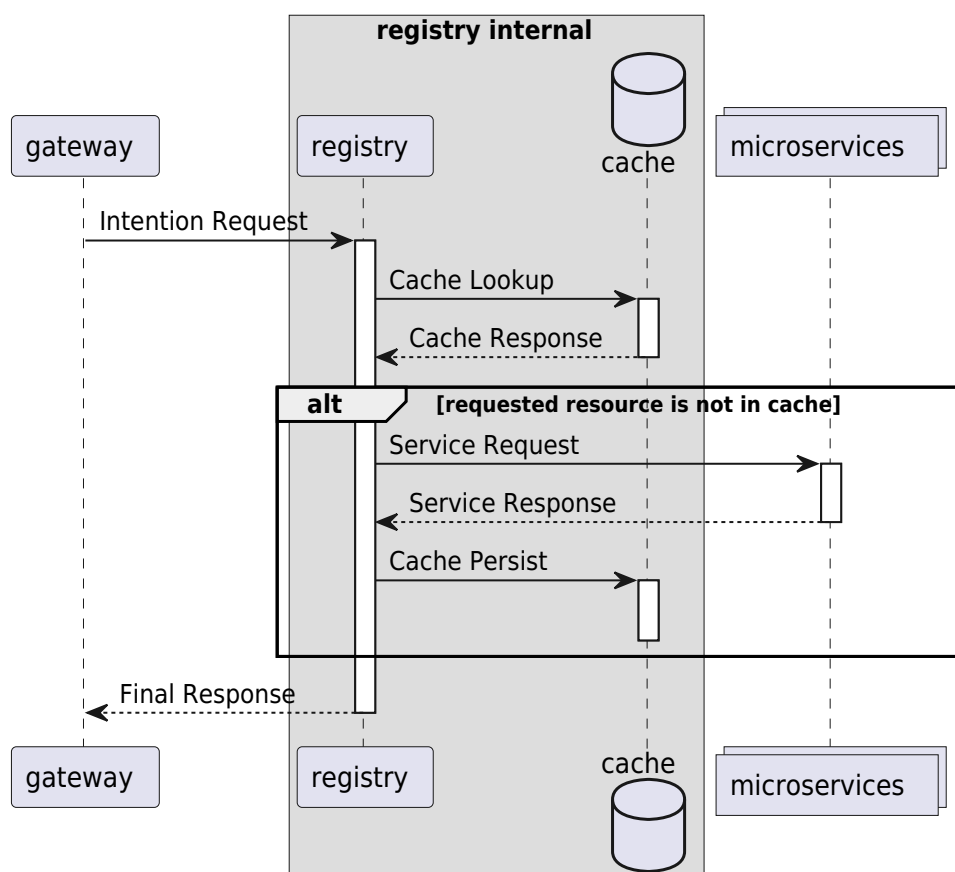
Für Node.js existieren mehrere Caching Lösungen. Bei den ersten Recherchen fielen die npm packages "memory-cache" und "node-cache" auf. Da "memory-cache" seit drei Jahren kein Update bekommen hat, haben wir uns letzten Endes für "node-cache" entschieden.

"node-cache" ist eine simple Caching Lösung, die nach dem Key-Value Prinzip funktioniert. Der Funktionsumfang besteht dabei aus den Methoden "set", "get" und "delete", wobei die Methode "set" einem zusätzlich erlaubt noch einen Timeout ("ttl" bzw. "time to live" genannt) zu übergeben. Ist der Timeout überschritten, wird der Eintrag automatisch aus dem Cache gelöscht. Der Nachteil dieser Lösung ist, dass nur eine Millionen Einträge pro Cache Instanz eingetragen werden können. Da aber gleich viel in den Cache eingetragen wird, wie die Anzahl angebotener Funktionen aller Microservices, wird dieser Nachteil nicht eintreffen.

Integration

Der Cache wird wie schon in Technologies beschrieben in Node.js verwendet. Spezifisch wird der Cache dem „registry“ Server hinzugefügt.

Das Resultat (veranschaulicht im folgenden UML diagram) besteht darin, dass registry versucht die angefragte Ressource aus dem Cache zu holen und gegebenenfalls eine Anfrage an den entsprechenden Microservice zu stellen, falls die Ressource nicht im Cache vorhanden ist.



Resultate

Die momentane Implementierung ist wie zuvor beschrieben umgesetzt. Der einzige Unterschied besteht darin, dass die Microservices die Möglichkeit besitzen, einen ttl mitzuschicken. Wird kein ttl vom Microservice mitgeschickt, so wird ein Standard ttl (momentan 30 Minuten) verwendet.

Wenn ein Microservice einen ttl mitschicken möchte, so muss dem „answer“ Object lediglich ein integer namens „ttl“ hinzugefügt werden. Dieser repräsentiert die Anzahl an Sekunden, wie lang zwischengespeichert werden soll.

Nutzungshinweis: Auf dieses vorliegende Schulungs- oder Beratungsdokument (ggf.) erlangt der Mandant vertragsgemäß ein nicht ausschließliches, dauerhaftes, unbeschränktes, unwiderrufliches und nicht übertragbares Nutzungsrecht. Eine hierüber hinausgehende, nicht zuvor durch *datenschutz-maximum* bewilligte Nutzung ist verboten und wird urheberrechtlich verfolgt.