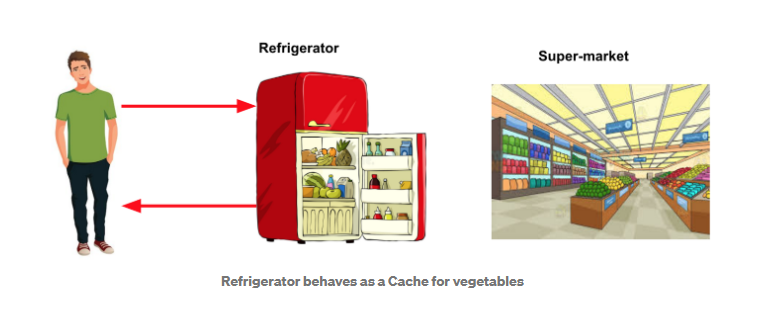
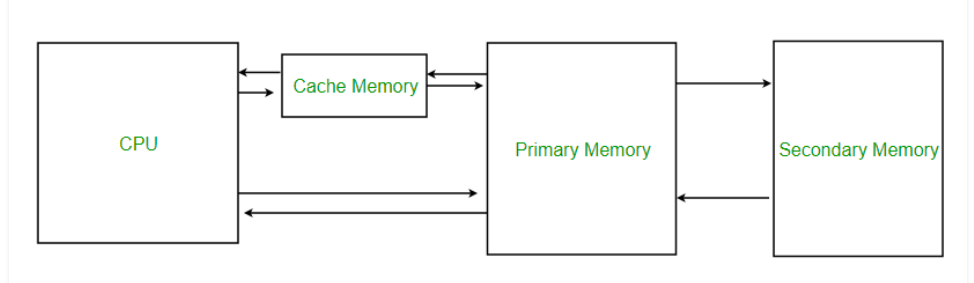
**Cache (Microservices Distributed Caching)**



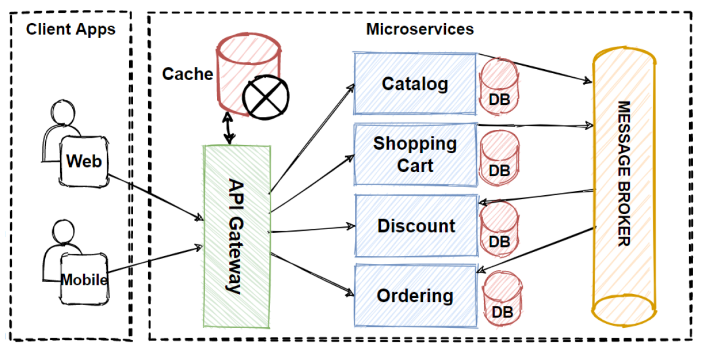
cook dinner every day. need different ingredients, vegetables, spices, etc for food preparation. do you visit the super-market to purchase this every day? That would be way too cumbersome and time-consuming. So, you check your kitchen or refrigerator first in case you have piled up your grocery. That would avoid a tiresome visit to the supermarket.



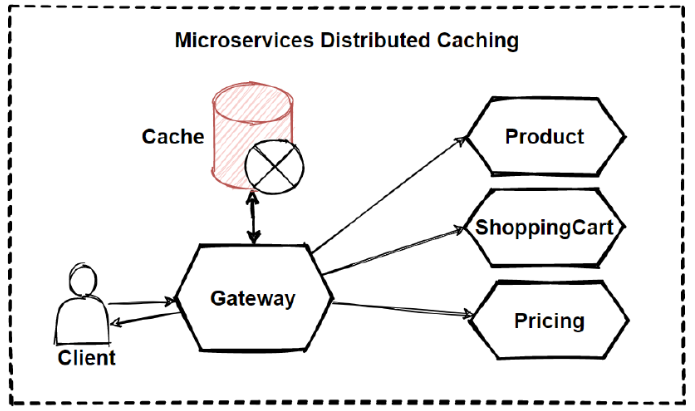
Some of the data retrieval or processing tasks performed by your application could be CPU intensive or take several seconds to complete. When this is the case, it is common to cache the retrieved data for a time so it can be retrieved quickly on subsequent requests for the same data.

Caching can increase performance, scalability, and availability for microservices. The main idea is reducing latency with cache and makes application faster. When the number of requests is increased, caching will be really important for accommodating whole requests with high availability.

If application request mostly comes for reading data that is not changes so frequently, then Caching will be so efficient. For example, reading product catalog from e-commerce application can be cached in the architecture.



Caching also provide to avoid re-calculation processes. If one operation calculates by one server, then other application can consume calculated data from cache. In Microservices architectures are typically implement a distributed caching architecture.



So how we can increase the speed of this use case?

Of Course, we should use the distributed cache. The distributed cache increases system responsiveness by returning cached data.

Additionally, separating the cache server from the microservices, gives ability to independently scale up cache services. This can be very useful when increased traffic of requests.

You can also use in-memory cache in our microservices, but it is not efficiency like distributed cache due to scalability issues. So now we can use distributed cache in our e-commerce design.

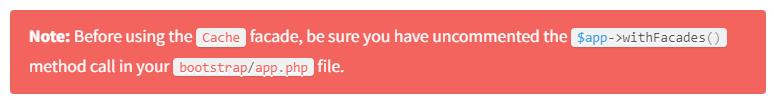
Redis, which stands for Remote Dictionary Server, is a fast, open-source, in-memory key-value data store for use as a database, cache, message broker, and queue. Redis is Fast, open source in-memory data store for use mostly as a database and cache. So, it’s good to use Redis in our architecture.

So, we should evolve our architecture with applying other Microservices Data Patterns in order to accommodate business adaptations faster time-to-market and handle larger requests.

**Cache in Lumen Microservices**

Laravel provides a unified API for various caching systems. Laravel supports popular caching backends like Memcached and Redis out of the box (file or database or DynamoDB).

The Lumen cache drivers utilize the exact same code as the full Laravel cache drivers. Beyond configuration, there are no differences between using the cache in Lumen and using the cache in Laravel.



Redis Support

Before using a Redis cache with Lumen, you will need to install the illuminate/redis package via Composer. Then, you should register the Illuminate\Redis\RedisServiceProvider in your bootstrap/app.php file:



If you have not called $app->withEloquent () in your bootstrap/app.php file, then you should call $app->configure('database'); in the bootstrap/app.php file to ensure the Redis database configuration is properly loaded.

**Storing Item in Cache**

Cache Facade:

Put () - > method => Cache::put(‘key’, ‘value’, $second=10)

Global Cache Helper:

Cache() -> retrieve and store data via cache

cache(‘key’), cache([‘key’ =>’value’], $second(or now()->addMinutes(10)))

example :

cache()->remember(‘users’, $second, function(){

return DB::table()

})

Cache::add(‘key’, ‘value’, $sec) return true/false

Cache::forever(‘key’, ‘value’) store for forever and delete by manual

**Retrieve Items from cache**

Get() method => cache::get(‘key’), cache::get(‘key’, ‘default\_value’), cache::get(‘key’, callback)

Item Exists in Cache

Has() method => if(Cache::has(‘key’))

**Removing Items from Cache**

Forget() method => cache::forget(‘key’), cache::forget(‘key’, ‘value’, 0)

Clear Cache

Flush() method => cache::flush();

Remember() method => retrieve item and store item from cache but also store default value if request item does not exists.

$value = cache::remember(‘users’, $second, function(){

Return DB::table

})

Cache::rememberForever(‘users’, function(){

Return DB::table

});