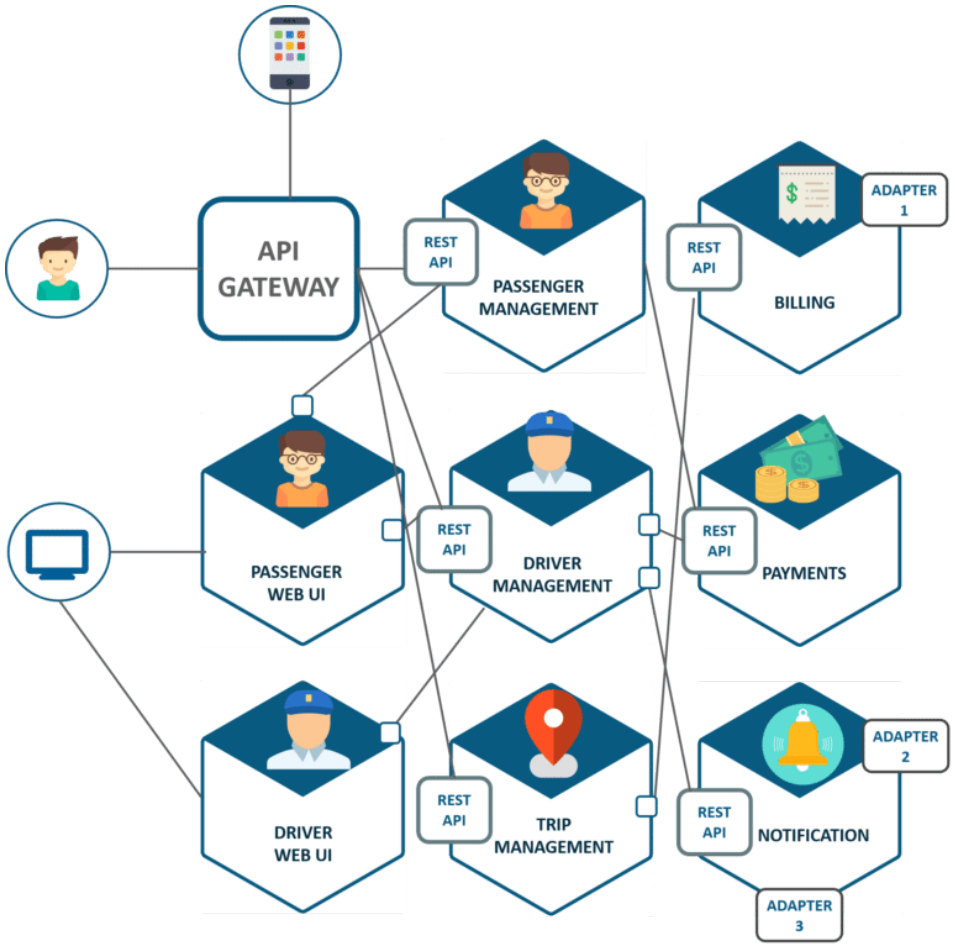
ALOM

HIGH AVAILABILITY & MICRO-SERVICES PATTERNS

UBER


API
GATEWAY
REST
API
PASSENGER
MANAGEMENT
REST
API
BILLING
ADAPTER
1
PASSENGER
WEB UI
REST
API
DRIVER
MANAGEMENT
REST
API
PAYMENTS
DRIVER
WEB UI
REST
API
TRIP
MANAGEMENT
REST
API
NOTIFICATION
ADAPTER
2
ADAPTER
3

PROBLÉMATIQUES :

Comment absorber la charge ?

Que faire si un micro-service ne répond pas ?

Comment limiter la charge ?

TROUVER LE POINT DE RUPTURE D'UN MICRO-SERVICE

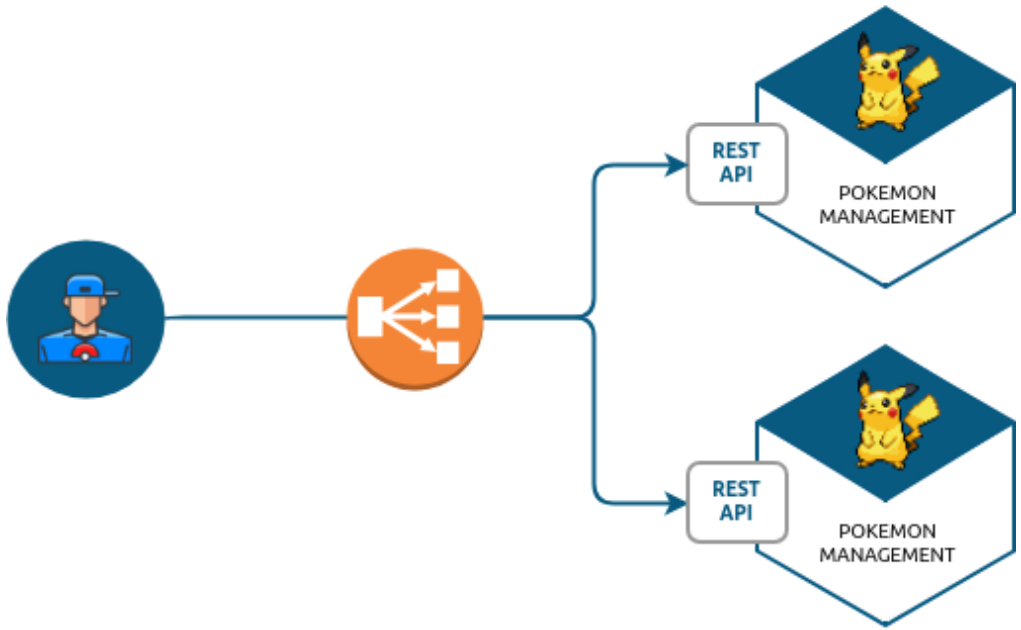
Faire un test de charge

- JMeter
- Gatling

PATTERNS D'ARCHITECTURE

LOAD-BALANCING

Répartition de charge sur plusieurs serveurs



LOAD-BALANCING

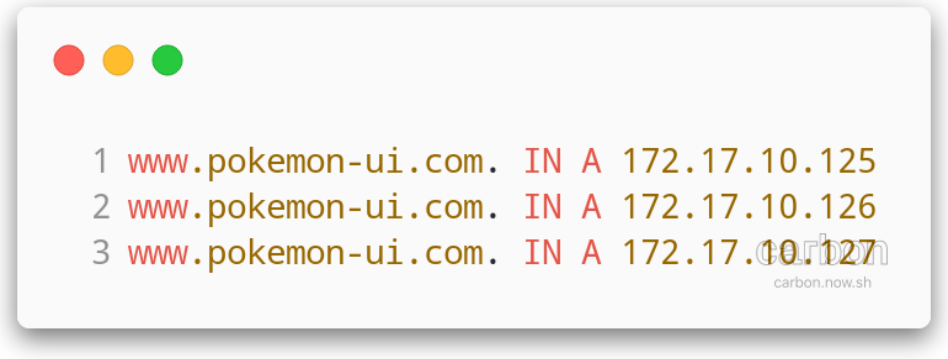
Plusieurs techniques :

- DNS load-balancing
- Reverse-proxy load-balancing
- Client-side load-balancing

Plusieurs stratégies

- round-robin
- least-connection
- ip-hash
- cookie

DNS LOAD-BALANCING (ROUND-ROBIN)



Le serveur DNS répondra séquentiellement chaque adresse IP

⚠️ Les services doivent être "stateless" !

REVERSE-PROXY LOAD-BALANCING

un serveur intermédiaire qui fait le travail (Apache httpd, HAProxy, Nginx...)

```
http{
    upstream pokemonUIBackend {
        # least_conn;
        # ip-hash;
        server 172.17.10.125:8080;
        server 172.17.10.126:8080;
        server 172.17.10.127:8080;
    }

    server {
        server_name www.pokemon-ui.com
        listen 80;

        location / {
            proxy_pass http://pokemonUIBackend;
        }
    }
}
```

REVERSE-PROXY LOAD-BALANCING

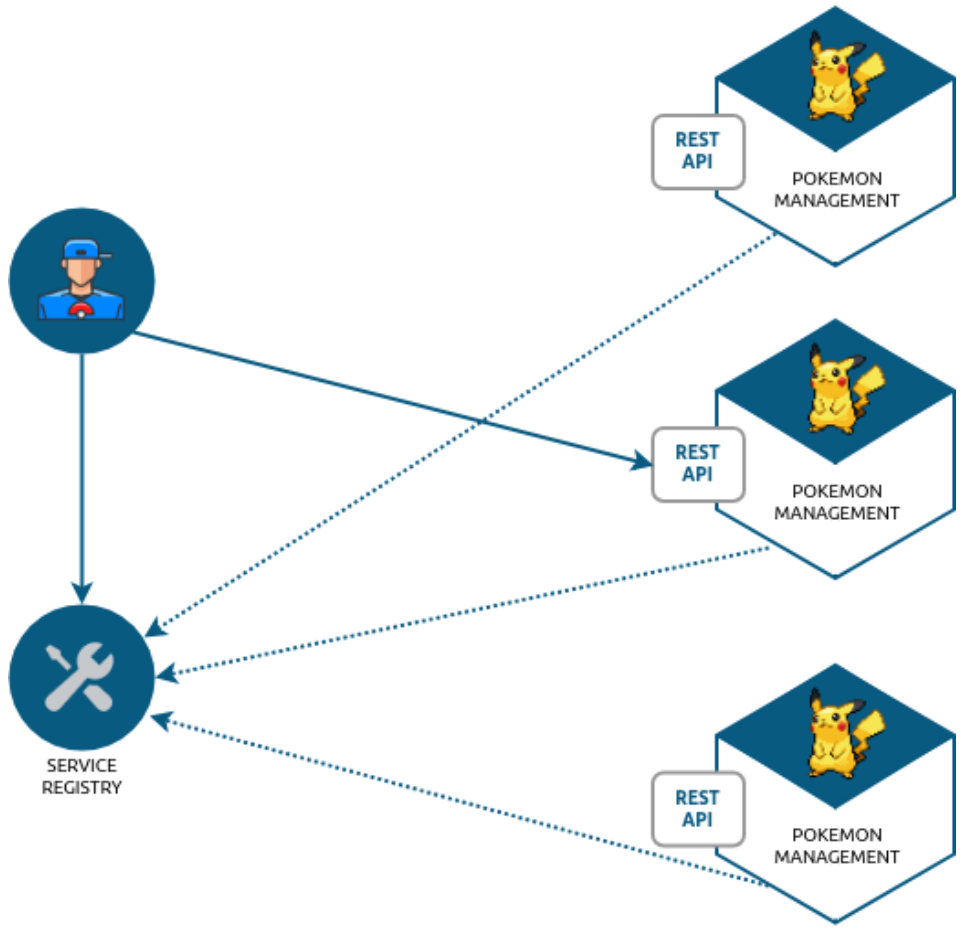
ℹ️ Les services peuvent être "statefull" en `ip-hash`!

```
http{
    upstream pokemonUIBackend {
        # least_conn;
        # ip-hash;
        server 172.17.10.125:8080;
        server 172.17.10.126:8080;
        server 172.17.10.127:8080;
    }

    server {
        server_name www.pokemon-ui.com
        listen 80;

        location / {
            proxy_pass http://pokemonUIBackend;
        }
    }
}
```

CLIENT-SIDE LOAD-BALANCING



CLIENT-SIDE LOAD-BALANCING

Le client (micro-service java) récupère la liste de toutes les adresses d'un service

Il se charge de répartir les appels

Nécessite un `service-registry`

CACHE

Mettre en cache des objets :

- Améliore les temps de réponse
- Limite les appels aux systèmes sous-jacents

⚠️ La mise en cache est facile. Supprimer des objets en cache est à réfléchir!

CACHE

Types de cache

- Local : en mémoire ou sur le disque (une Map !, EhCache, ...)
- Distribué : un service dédié (Redis, Hazelcast, ...)

Le cache local est toujours plus performant, le cache distribué est partagé entre toutes les instances d'un service, mais nécessite un appel réseau !

ℹ️ CACHE DISTRIBUÉ

Un cache distribué peut permettre de gérer une session utilisateur dans un système stateless !

La session est stocké en cache, et non plus en mémoire, elle peut être disponible sur l'ensemble des instances de notre application !

SUPPRIMER DES OBJETS DU CACHE

Le cache est pratique pour les données "référentielles"

Pour les données "vivantes", prévoir une éviction du cache quand l'objet est modifié

Notion d'expiration ou `TTL`

CACHE VS THE COST OF I/O :

Action	Latency	# of cycles	Human Time
1 Cycle CPU (3GHz Clock)	0.3 ns	1	1 s
RAM access	70 - 100 ns	233 - 333	3.5 - 5.5 m
NVMe SSD	7 - 150 µs	23k - 500k	6.5 h - 5.5 d
Internet: SF to NYC	40 ms	130 M	4.2 years

RÉSILIENCE

Capacité d'un système à surmonter une altération de son environnement

EN MICRO-SERVICES

Que faire si le micro-service auquel j'ai envoyé une requête n'est pas disponible ?

- 500 Internal Server Error
- java.io.IOException: Connection reset by peer
- java.net.ConnectException: Connection refused

🔃 Retry !

🔃 Retry !

Avec un peu de chance, le micro-service sera dispo quelques milli-secondes après cette erreur.

Si le micro-service est derrière un load-balancer, le load-balancer va désactiver le routage vers l'instance en erreur.

Permet de compenser des erreurs temporaires

JAVA.IO.IOEXCEPTION: CONNECTION TIMED OUT

Notre service a passé du temps à attendre une réponse qu'il n'a pas eu.

Il est probable que les appels suivants auront le même problème

CIRCUIT-BREAKER

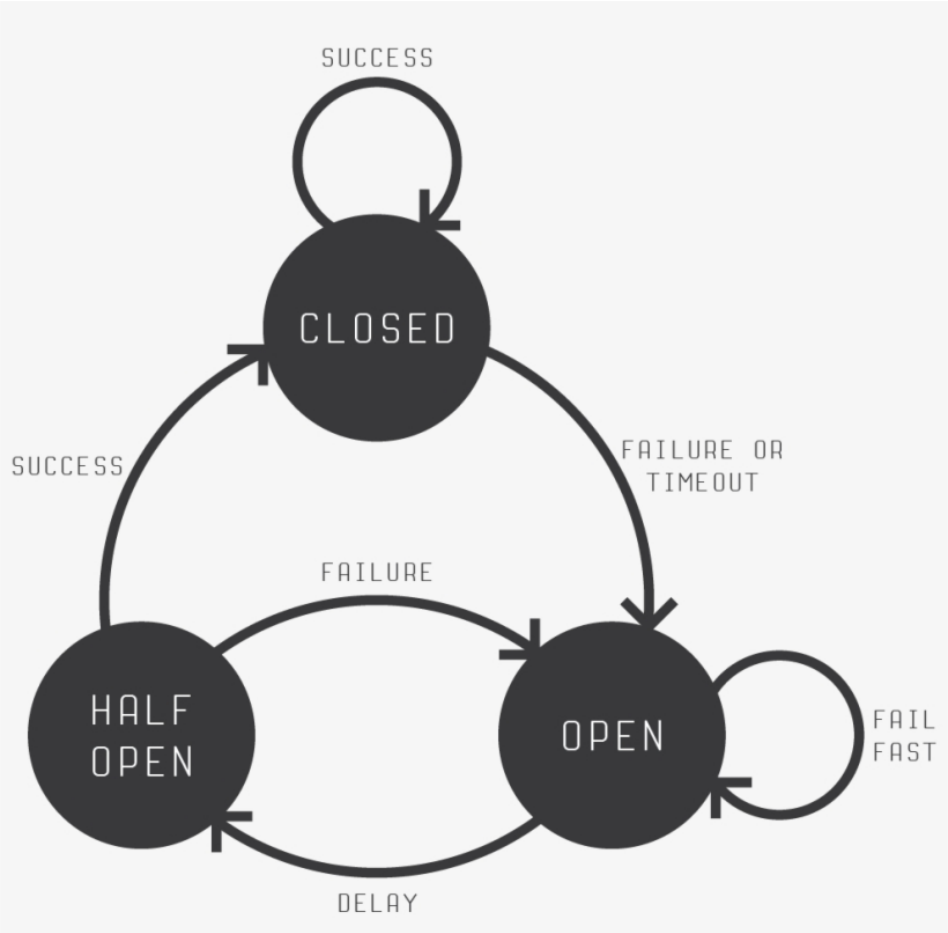
On "ouvre" le circuit si on a rencontré plusieurs erreurs/timeout

`fail-fast`

On peut fournir un mode-dégradé

On refermera le circuit petit à petit si tout se passe bien

CIRCUIT-BREAKER



IMPLÉMENTATIONS AVEC SPRING / SPRING-BOOT

- Cache
- Retry

CACHE (SPRING-BOOT-STARTER-CACHE)

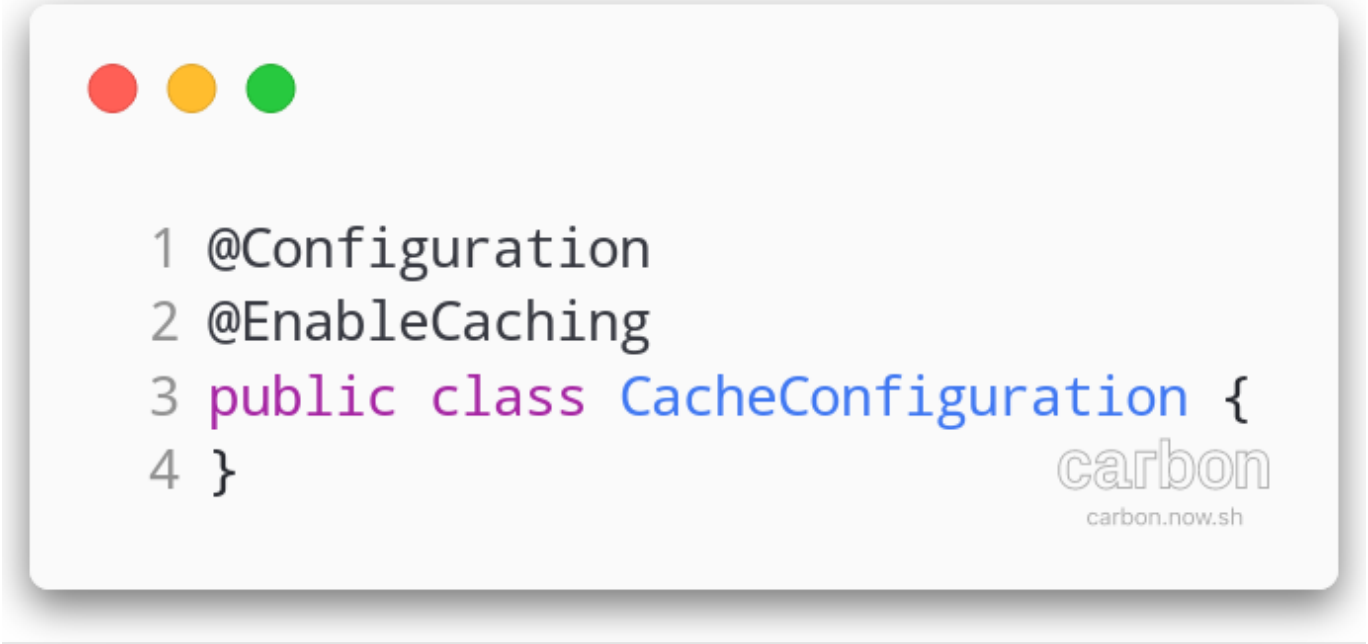
- `@EnableCaching` active la gestion du cache (création d'un `CacheManager`)
- `@Cacheable("<cache-name>")` active le cache sur une méthode

Si pas de librairie de cache dans le classpath, Spring utilisera une `ConcurrentHashMap`

Librairies (Cache providers): EhCache, Redis

Il est possible de recevoir le `CacheManager` en injection de dépendances pour manipuler directement le cache

@ENABLECACHING



@CACHEABLE

```java
@Cacheable("pokemon-types")
public PokemonType getPokemonType(int id) {
    return restTemplate
        .getForObject(pokemonServiceUrl+"/pokemon-types/{id}", PokemonType.class, id);
}
```

RETRY SPRING-RETRY

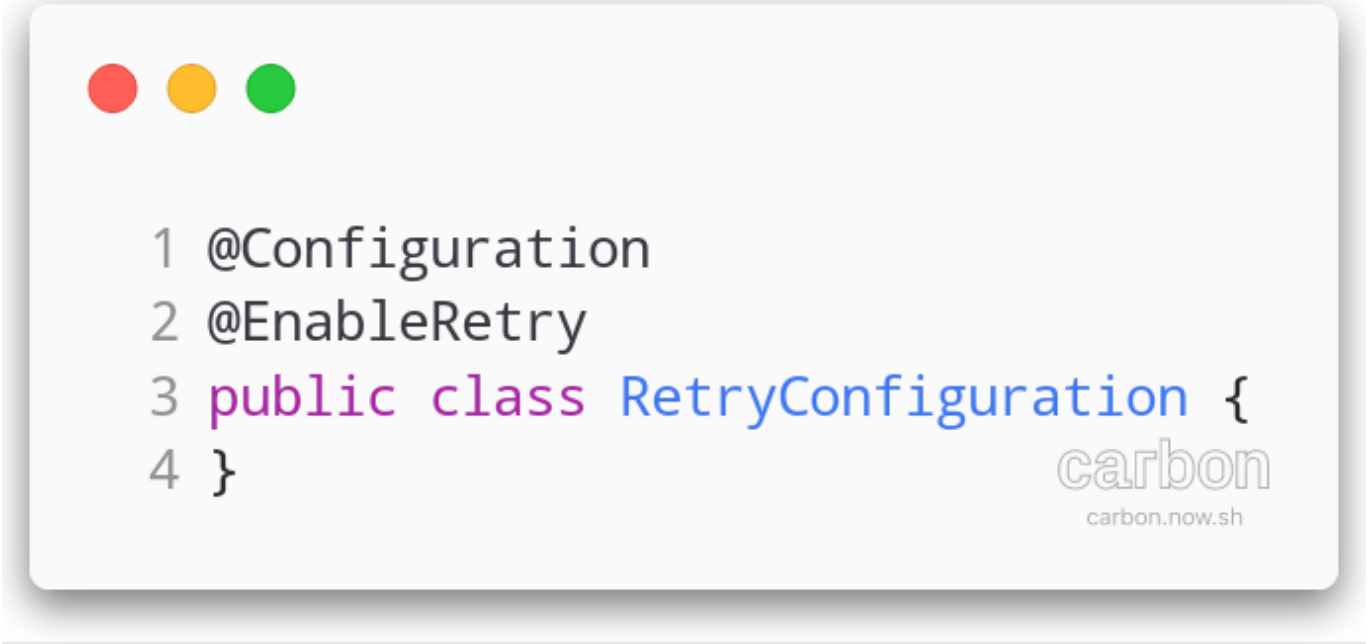
- `@EnableRetry` active la gestion du retry
- `@Retryable` active le retry sur une méthode à chaque exception ! (3 tentatives par défaut)

SPRING-RETRY

```xml
<!--retry uses AOP-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.retry</groupId>
    <artifactId>spring-retry</artifactId>
</dependency>
```

@ENABLERETRY



@RETRYABLE

```java
@Retryable
public PokemonType getPokemonType(int id) {
    return restTemplate
        .getForObject(pokemonServiceUrl+"/pokemon-types/{id}", PokemonType.class, id);
}
```

TP

High-Availability 🦄