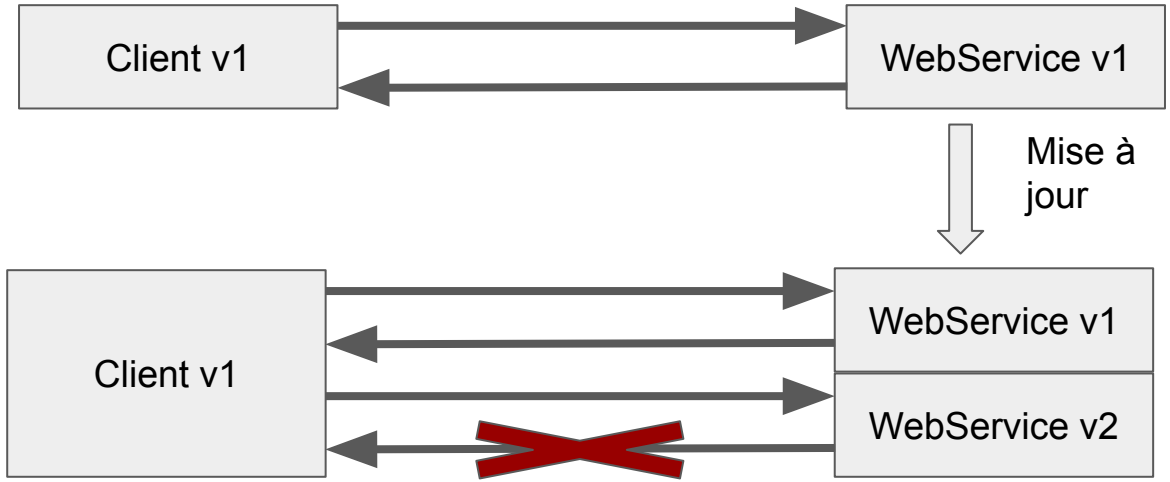
Versionning

Versionner une API

- Ne pas briser le contrat entre le client et notre serveur lors d'un changement.
- On fait coexister l’ancienne et la nouvelle version



Quels changements nécessitent un changement de version ?

Changement sans impacts (si le client les gèrent) :

- Modification de l'ordre des éléments de la représentation d'une ressource
- Ajout d'un nouvel élément dans de la représentation d'une ressource
- Nouveau code retour

Changement nécessitant un changement :

- Modification majeure de la représentation d'une ressource (suppression d'un élément, changement de type, …)
- Dépréciation

Configurer le destinataire pour qu'il gère des changements

Exemple :

Service de suivi vétérinaire, permet de consulter la fiche d'un animal de compagnie.

V1

```
[
    {
        "id": 123,
        "name": "Mochi",
        "species": "CAT",
        "sex": "MALE",
        "birthday": "2020-09-09T22:00:00.000+00:00",
        "ownerFirstName": "Théo",
        "ownerLastName": "HENRY"
    }
]
```

V2

```
[
    {
        "id": 123,
        "name": "Mochi",
        "species": "CAT",
        "sex": "MALE",
        "birthday": "2020-09-09T22:00:00.000+00:00",
        "owner": {
            "lastName": "HENRY",
            "firstName": "Théo"
        }
    }
]
```

En Java avec Jackson Json (spring)

On peut gérer le problème d'un élément inconnu à plusieur niveau :

- En ajoutant l'annotion `@JsonIgnoreProperties (ignoreUnknown = true)` à votre DTO indique au désérialisateur de ne pas lever l'exception si il est confronté à un élément inconnu.
- Au moment de l'instanciation du désérialisateur :

```
ObjectMapper mapper = new
ObjectMapper().configure(DeserializationFeature. FAIL_ON_UNKNOWN_PROPERTIES, false);
```

- À la définition de notre end point, en précisant l'objet souhaité dans la signature de la méthode :

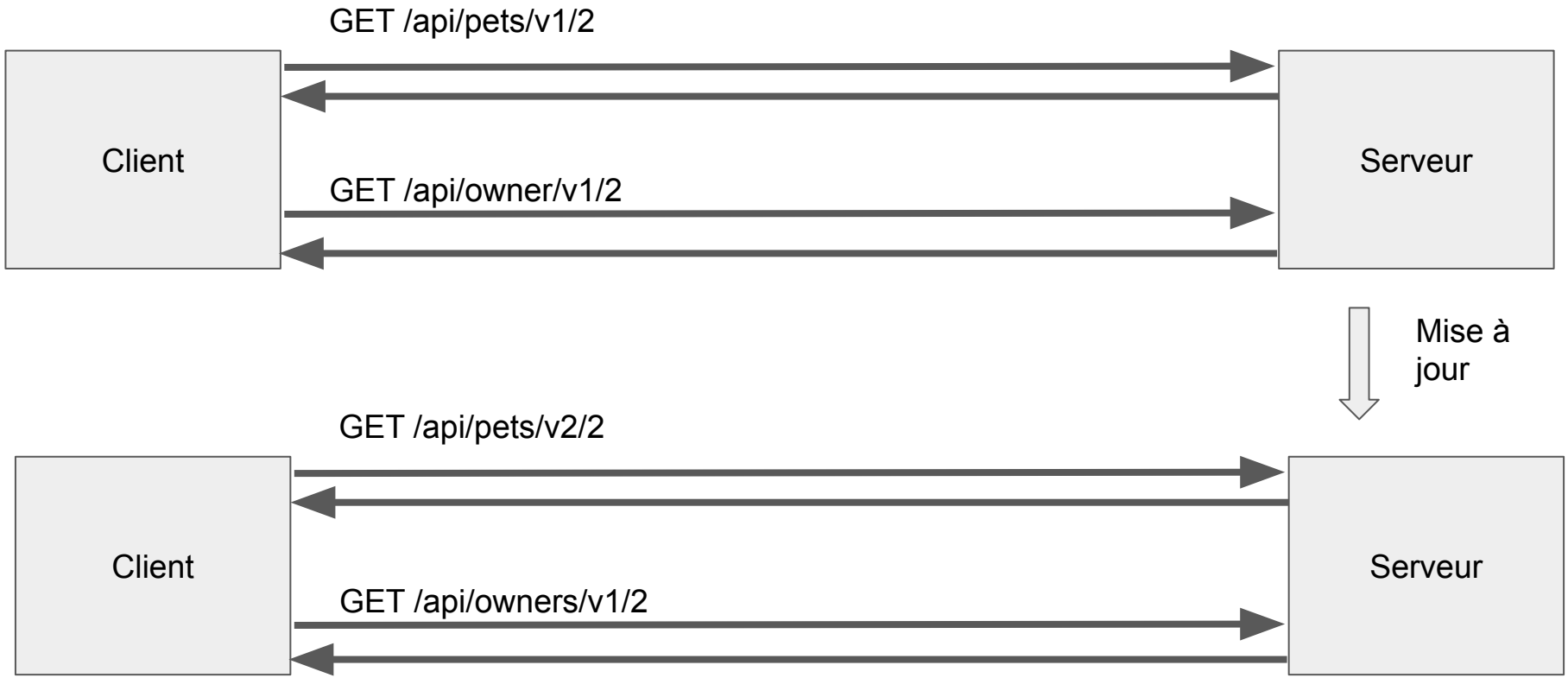
```
public String addPet(@RequestBody PetDeserialisation pet) {
```

Comment versionner

Plusieurs solutions :

- Versionning via l’URI
- Versionning via un paramètre
- Versionning via le header Accept

Versionning via l’URI



Versionning via l'URI

- On ajoute dans l'url le numéro de la version de l'api
  - api/pets/v1/{id}
  - api/pets/v2/{id}
- On versione par ressources et pas toute l'api
  - api/pets/v2/{id} != api/v1/pets/{id}
  - api/owners/v1/{id} != api/v1/owners/{id}

- Versionner une sous-ressources ?
  - la fiche de notre animal de compagnie contient la liste de ses consultations vétérinaire
  - Comment verssionner ces fiches ?
    - toute la ressource ? → api/pets/v2/{id}/checkups/{id}
    - la sous ressources ? → api/pets/v1/{id}/checkups/v2/{id}
- On s'éloigne de la définition de ressource de Roy Thomas Fielding
  - *"Finally, it allows an author to reference the concept rather than some singular representation of that concept, thus removing the need to change all existing links whenever the representation changes (assuming the author used the right identifier)."* https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#sec_5_2_1_1

Versionning via un paramètre

- On ajoute la version dans un paramètre au moment de notre appel :
  - api/pets/{id}?version=1
  - api/pets/{id}?version=2
- Simple à mettre en place

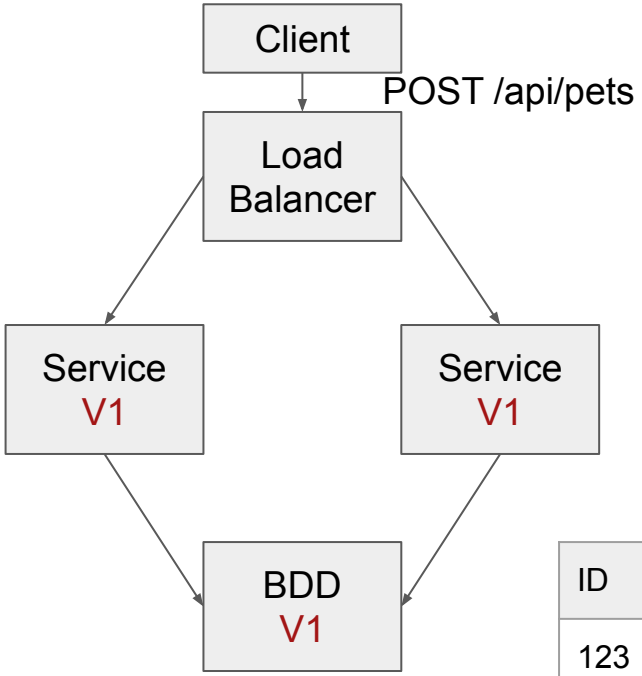
- Mélange des paramètres techniques et fonctionnels
  - une recherche filtrer par espèce → api/pets/?version=2&species?CAT

Versionning via le header Accept

- On indique la version via le header de négociation de contenu (ACCEPT)
  -
- Permet une bonne granularité

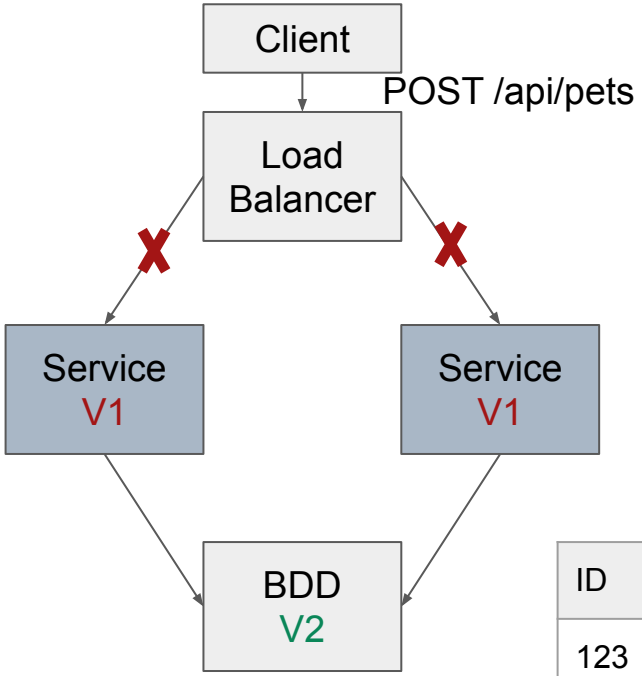
- Difficilement testable depuis un navigateur web

Versionning de la BDD



ID	name	[...]	ownerFirstName	ownerLastName
123	Mochi		Théo	HENRY
999	Padmé		Gérard	ONFROY

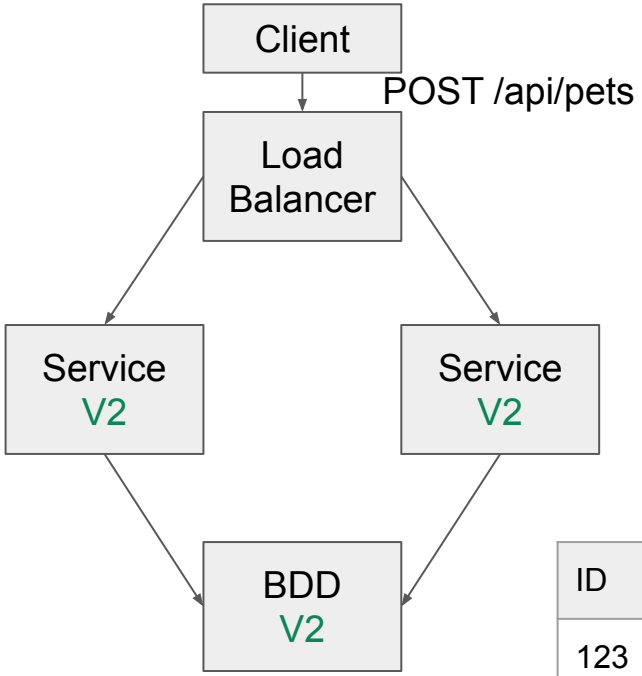
Migration de la BDD avec interruption



ALTER TABLE pets DROP COLUMN ownerFirstName, ownerLastName

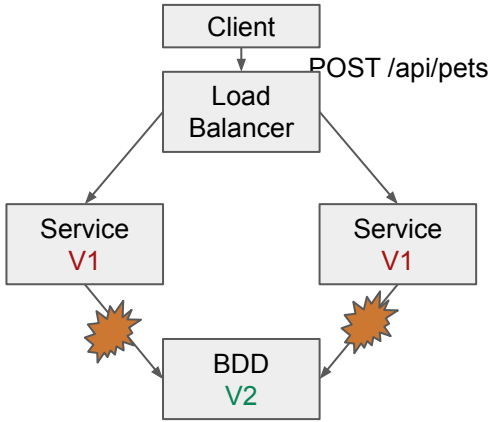
ID	name	[...]
123	Mochi	
999	Padmé	

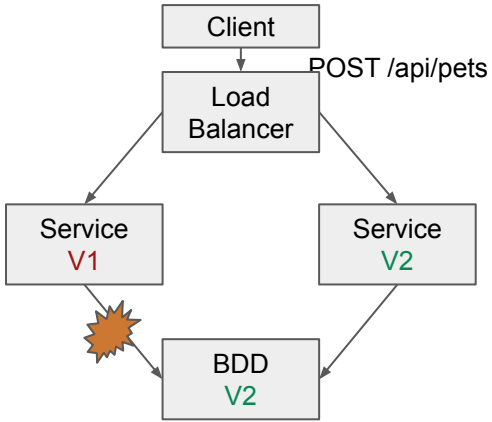
C’est fini

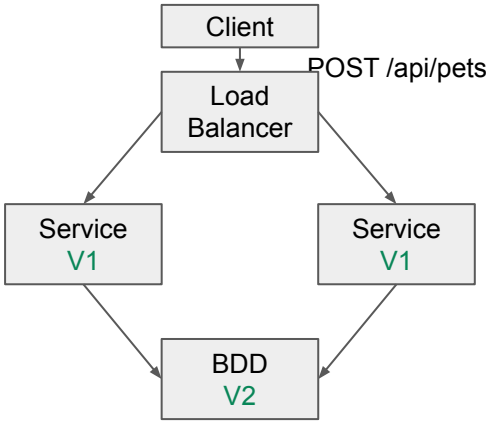


ID	name	[...]
123	Mochi	
999	Padmé	

Sans interruption, les étapes d’une migration







Rétro-compatibilité de la BDD

La version **n+1** de la base de donnée doit être rétrocompatible avec la version **n** de l'application

Cela permet également de revenir sur la version applicative précédente

Version	BDD	Application
V1	Colonne non NULL	Lecture et écriture
V2	Colonne nullable	écriture mais plus de lecture
V3	-	Ni écriture ni lecture
V4	DROP de la colonne	

Exemples et points d’attentions

Ajouter une valeur à une enum → 2 versions applicatif

- V2 - Ajouter la nouvelle valeur mais ne pas la persister en base
- V3 - Autoriser l’écriture de la nouvelle valeur

Eviter les select * car cela va casser les Prepared Statement

Les locks en base de données (ex: ALTER, lock toutes les requêtes sur les lignes) peuvent générer des timeouts → interruption de service

Webographie

https://apisyouwonthate.com/blog/api-versioning-has-no-right-way

https://restfulapi.net/versioning/

https://datatracker.ietf.org/doc/html/rfc7231

https://www.youtube.com/watch?v=pIkA-aPtkNs

https://dzone.com/articles/versioning-rest-api-with-spring-boot-and-swagger

https://cloud.google.com/architecture/rate-limiting-strategies-techniques

https://en.wikipedia.org/wiki/Token_bucket

https://github.com/bucket4j/bucket4j/blob/master/README.md

https://systemsdesign.cloud/SystemDesign/RateLimiter

https://www.javadevjournal.com/spring/etags-for-rest-with-spring/

https://www.youtube.com/watch?v=pIkA-aPtkNs