

Mise en place d'un web service sous FastAPI
SPGoO – Août 2026 – YS
ANF NoSQL

Objectif : mettre en place un service web de démonstrateur pour accès
source : <https://github.com/ystroppa/anfNoSQL2026>

Cache d'objets — démonstrateur FastAPI + Redis

Web service exposant un cache Redis générique : pousser, récupérer, modifier et retirer des objets de nature quelconque (JSON libre), avec expiration optionnelle.

Même architecture que le démonstrateur MongoDB (routes FastAPI / logique métier isolée dans `crud.py` / modèles Pydantic), adaptée à un usage de cache plutôt que de base de données persistante.

Structure

```
objets-cache-api/
├── app/
│   ├── main.py          # routes FastAPI
│   ├── crud.py          # opérations Redis (set/get/scan/delete)
│   ├── models.py        # modèles Pydantic
│   └── database.py       # connexion redis-py (client asyncio)
├── requirements.txt
├── docker-compose.yml    # Redis local en un clic
└── .env.example
```

Principe

Chaque objet est stocké sous la clé `{type}:{id}` — par exemple `chercheur:marie-curie` ou `session:3f2a1c`. Cette convention permet de :

- lister ou vider tout un type d'un coup (SCAN sur `{type}:*`) sans tenir d'index séparé ;
- mélanger librement des objets de natures différentes dans le même cache, chacun avec son propre contenu JSON (`data`) et sa propre durée de vie (`ttl_secondes`).

Lancer le démonstrateur

Chaque objet est stocké sous la clé `{type}:{id}` — par exemple `chercheur:marie-curie` ou `session:3f2a1c`. Cette convention permet de :

- lister ou vider tout un type d'un coup (SCAN sur `{type}:*`) sans tenir d'index séparé ;
- mélanger librement des objets de natures différentes dans le même cache, chacun avec son propre contenu JSON (`data`) et sa propre durée de vie (`ttl_secondes`).

Lancer le démonstrateur

1. Démarrer Redis (si vous n'avez pas déjà une instance) :

```
docker compose up -d
```

2. Installer les dépendances et lancer l'API :

```
python -m venv venv && source venv/bin/activate # Windows :  
venv\Scripts\activate  
pip install -r requirements.txt  
cp .env.example .env  
uvicorn app.main:app --reload
```

3. Ouvrir la documentation interactive :

<http://localhost:8000/docs>

Modèle de données

```
{  
  "type": "chercheur",  
  "id": "marie-curie",  
  "data": { "nom": "Curie", "discipline": "Physique" },  
  "ttl_secondes": 3600  
}
```

- `id` est optionnel à la création : un identifiant est généré automatiquement si absent.
- `ttl_secondes` est optionnel : sans lui, l'objet reste en cache indéfiniment (jusqu'à suppression explicite ou éviction Redis selon la politique du serveur).
- `data` accepte n'importe quelle structure JSON — le cache ne fait aucune hypothèse sur la nature des objets qu'il contient.

Routes disponibles

Méthode	Route	Description
POST	/objets	Pousser un objet dans le cache
GET	/objets	Lister les objets (filtre optionnel type)
GET	/objets/{type}/{id}	Récupérer un objet
PATCH	/objets/{type}/{id}	Modifier un objet (fusion partielle)
DELETE	/objets/{type}/{id}	Retirer un objet
DELETE	/objets/{type}	Vider tous les objets d'une nature donnée
GET	/types	Lister les natures d'objets actuellement en cache

Exemples avec curl

```
# Pousser un objet (id explicite)  
curl -X POST http://localhost:8000/objets \  
  -H "Content-Type: application/json" \  
  -d '{"type": "chercheur", "id": "marie-curie", "data": {"nom": "Curie", "discipline": "Physique"}, "ttl_secondes": 3600}'
```

```

-d '{"type": "chercheur", "id": "marie-curie",
  "data": {"nom": "Curie", "discipline": "Physique"}}'

# Pousser un objet d'une autre nature, sans id (généré automatiquement),
# avec expiration à 5 minutes
curl -X POST http://localhost:8000/objets \
  -H "Content-Type: application/json" \
  -d '{"type": "session", "data": {"utilisateur": "yvan"}, "ttl_secondes": 300}'

# Récupérer un objet
curl http://localhost:8000/objets/chercheur/marie-curie

# Lister tous les objets d'un type
curl "http://localhost:8000/objets?type=chercheur"

# Modifier (fusion partielle)
curl -X PATCH http://localhost:8000/objets/chercheur/marie-curie \
  -H "Content-Type: application/json" -d '{"data": {"discipline": "Chimie"}}'

# Retirer un objet
curl -X DELETE http://localhost:8000/objets/chercheur/marie-curie

# Vider tout un type
curl -X DELETE http://localhost:8000/objets/chercheur

```

Notes de conception

- **redis-py** (client asyncio natif depuis la version 4.2) est utilisé pour rester cohérent avec le modèle async de FastAPI, comme Motor pour le démonstrateur MongoDB.
- Les objets sont sérialisés en JSON côté application (`json.dumps`) ; Redis les stocke comme simples chaînes — c'est le service qui porte la structure, pas le cache lui-même.

Ecran de visualisation des entrées de ce web service à partir de l'url suivante :

<http://localhost:8000/docs>

Lancement du service à partir d'un terminal

```
(formmongodb) (base) ystroppa@isto.cnrs-orleans.fr@5CG6092ZML:~/formation_NOSQL/objets-cache-api-redis$ uvicorn app.main:app --reload
INFO: Will watch for changes in these directories:
[/home/ystroppa@isto.cnrs-orleans.fr/formation_NOSQL/objets-cache-api-redis']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [1078807] using WatchFiles
INFO: Started server process [1078809]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

Cache d'objets (Redis) 1.0.0 OAS 3.1

/openapi.json

Démonstrateur FastAPI exposant un cache Redis générique : pousser, récupérer, modifier et retirer des objets de nature quelconque.

default

POST	/objets	Pousser un objet dans le cache	⌵
GET	/objets	Lister les objets en cache	⌵
GET	/types	Lister les natures d'objets présentes en cache	⌵
GET	/objets/{type}/{id}	Récupérer un objet	⌵
PATCH	/objets/{type}/{id}	Modifier un objet (fusion partielle)	⌵
DELETE	/objets/{type}/{id}	Retirer un objet du cache	⌵
DELETE	/objets/{type}	Vider tous les objets d'une nature donnée	⌵

Schemas

HTTPValidationError > Expand all object

Objet > Expand all object

ObjetCreate > Expand all object

ObjetUpdate > Expand all object

ValidationError > Expand all object