

Rapport du laboratoire 2

PAR

David NGUYEN, NGUD24049607

RAPPORT PRÉSENTÉ À FABIO PETRILLO DANS LE CADRE DU COURS
ARCHITECTURE LOGICIELLE (LOG430-02)

MONTRÉAL, LE 26 SEPTEMBRE 2025

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

Tables des matières

- [Mise en place](#)
 - [Question 1](#)
 - [Question 2](#)
 - [Question 3](#)
 - [Question 4](#)
 - [Question 5](#)
-

Mise en place

Le projet a été fork, puis ensuite clôné. Le fichier `.env` a été créé à partir du `.env.example` et `docker-compose.yml`. En ce qui concerne le port 5000, `store_manager` écoute à l'intérieur du conteneur sur le port 5000 (par défaut, les ports d'un conteneur ne sont pas accessibles depuis la machine, car ils sont isolés). Donc, selon ma compréhension, si je veux ouvrir mon navigateur et aller sur `http://localhost:5000`, il faut dire à Docker: "Le port 5000 de ma machine hôte doit être relié au port 5000 du conteneur". Ainsi, dans mon `docker-compose.yml`, j'y ai ajouté

```
store_manager:
  build: .
  volumes:
    - ./app
  environment:
    DB_HOST: mysql
    DB_PORT: 3306
    DB_USER: labo02
    DB_PASSWORD: labo02
    DB_NAME: labo02_db
    REDIS_HOST: redis
    REDIS_PORT: 6379
    REDIS_DB: 0
  ports:
    - "5000:5000" # Nouvellement ajouté!
  depends_on:
    - mysql
    - redis
```

Par la suite, j'ai `docker compose build` et `docker compose up -d` pour orchestrer les conteneurs ensemble.

Ainsi, on peut lancer l'application dans son conteneur, soit en allant sur `http://localhost:5000/` pour voir la vue de l'application.

 localhost:5000

Le Magasin du Coin

Application de gestion de magasins

Formulaires d'enregistrement

[Utilisateurs](#)[Articles](#)[Commandes](#)

Rapports

[Les plus gros acheteurs](#)[Les articles les plus vendus](#)

Finalement, pour la préparation de l'environnement de déploiement et le pipeline, j'ai commencé par exécuter les tests en local:

```
python -m venv .venv
.venv/Scripts/Activate.ps1
pip install -r requirements.txt
pytest -q
```

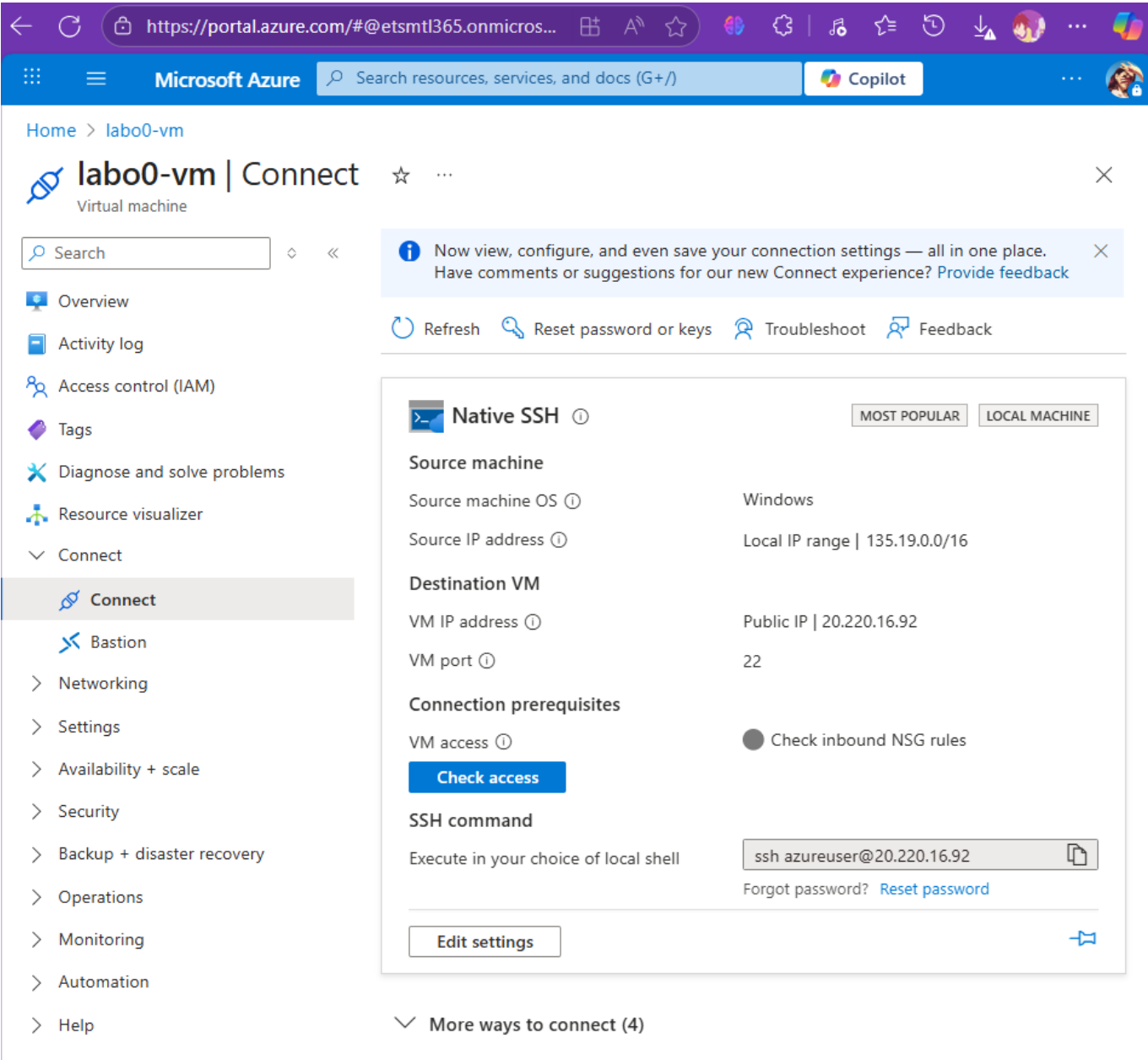
Le fichier `ci.yml` a été modifié en fonction du CI. J'ai procédé par la suite au versionnage du code avec:

```
git add .
git commit -m "MESSAGE DE COMMIT"
git push
```

À noter qu'il y a eu un changement au niveau du script `init_sql`. De ce fait, j'ai appliqué les changements, puis j'ai rebuild les images comme suit:

```
docker compose down
docker volume ls
docker volume rm log430-a25-labo2_mysql_data
docker build .
docker compose up -d
```








Maintenant, au niveau du CD, ma stratégie est d'y aller avec la VM de Azure (étant donné qu'actuellement nous travaillons encore à distance et donc les locaux de l'école ne sont pas accessibles).



Ensuite, les secrets et les variables ont été configurés dans le repo GitHub:








Repository secrets

New repository secret

Name 	Last updated	
 SSH_PASS	1 minute ago	 
 SSH_USER	now	 

Repository variables

New repository variable

Name 	Value	Last updated	
 SSH_PORT	22	1 minute ago	 
 VM_HOST_IP	20.220.16.92	1 minute ago	 

Par la suite, j'ai vérifié la connectivité depuis ma machine locale en m'y connectant:

```
azureuser@labo0-vm: ~  
C:\Users\nguye>ssh azureuser@20.220.16.92  
azureuser@20.220.16.92's password:  
Permission denied, please try again.  
azureuser@20.220.16.92's password:  
Permission denied, please try again.  
azureuser@20.220.16.92's password:  
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.11.0-1018-azure x86_64)  
  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:       https://ubuntu.com/pro  
  
System information as of Wed Sep 24 16:16:04 UTC 2025  
  
System load:  0.05          Processes:            114  
Usage of /:   9.6% of 28.02GB Users logged in:          0  
Memory usage: 40%          IPv4 address for eth0: 172.16.0.4  
Swap usage:   0%  
  
* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s  
  just raised the bar for easy, resilient and secure K8s cluster deployment.  
  
  https://ubuntu.com/engage/secure-kubernetes-at-the-edge  
  
Expanded Security Maintenance for Applications is not enabled.  
  
14 updates can be applied immediately.  
2 of these updates are standard security updates.  
To see these additional updates run: apt list --upgradable  
  
1 additional security update can be applied with ESM Apps.  
Learn more about enabling ESM Apps service at https://ubuntu.com/esm  
  
*** System restart required ***  
Last login: Fri Sep 12 18:02:22 2025 from 135.19.242.22  
azureuser@labo0-vm:~$ |
```

En ce qui concerne le déploiement, j'ai ajouté le fichier `cd.yml` qui dépend du fichier `ci.yml` et après un `git push` on peut voir sur GitHub Actions que le workflow fonctionne bien, car la commande `docker-compose ps` montre que les services roulent:

5 workflow runs

Event ▾

Status ▾

Branch ▾

Actor ▾

✓

CD

CD #2: completed by [Orchydae](#)

📅

6 minutes ago

⌚

1m 3s

✓

CD typo fix

CI #4: Commit [be87101](#) pushed by [Orchydae](#)

📅

7 minutes ago

⌚

1m 50s

main

715	out:	NAME	IMAGE	COMMAND	SERVICE	CREATED	STATUS	PORTS
716	out:	log430-a25-labo2-mysql-1	mysql:8	"docker-entrypoint.s..."	mysql	3 seconds ago	Up 1 second	0.0.0.0:3306->3306/tcp,
		[::]:3306->3306/tcp, 33060/tcp						
717	out:	log430-a25-labo2-redis-1	redis:7	"docker-entrypoint.s..."	redis	3 seconds ago	Up 1 second	0.0.0.0:6379->6379/tcp,
		[::]:6379->6379/tcp						
718	out:	log430-a25-labo2-store_manager-1	log430-a25-labo2-store_manager	"python store_manage..."	store_manager	2 seconds ago	Up Less than a second	0.0.0.0:5000->5000/tcp,
		[::]:5000->5000/tcp						
719		=====						
720	✓	Successfully executed commands to all host.						
721								

azureuser@labo0-vm: ~/log4

Last login: Wed Sep 24 16:16:05 2025 from 135.19.242.22

azureuser@labo0-vm:~\$ ls

log430-a25-labo0 log430-a25-labo2

azureuser@labo0-vm:~\$ cd log430-a25-labo2

azureuser@labo0-vm:~/log430-a25-labo2\$ ls

Dockerfile LICENSE README.md db-init docker-compose.yml docs requirements.txt

src

azureuser@labo0-vm:~/log430-a25-labo2\$ docker-compose ps

Finalement, j'ai ajouté mon fichier .env dans la VM et puis j'ai lancé la commande `docker-compose up -d -build` faisant en sorte que l'application est officiellement déployée sur `http://20.220.16.92:5000`:



Question 1

Lorsque l'application démarre, la synchronisation entre Redis et MySQL est initialement déclenchée par quelle méthode ? Veuillez inclure le code pour illustrer votre réponse.

La méthode `sync_all_orders_to_redis()` est appelée au démarrage de l'application. C'est elle qui copie toutes les commandes de MySQL vers Redis (mais seulement si Redis est vide).

```
# === store_manager.py ===
if __name__ == "__main__":
    # Sync MySQL -> Redis
    print("Initial sync MySQL -> Redis")
    sync_all_orders_to_redis()

    # Démarrage du serveur
    server = HTTPServer(("0.0.0.0", 5000), StoreManager)
    print("Server running on http://0.0.0.0:5000")
    server.serve_forever()
```

Question 2

Quelles méthodes avez-vous utilisées pour lire des données à partir de Redis ? Veuillez inclure le code pour illustrer votre réponse.

- `keys("order:*")`: pour lister les clés d'orders (on exclut celles finissant par `:items`).
- `hgetall(f"order:{oid}")`: pour lire les champs du hash d'une commande (`user_id`, `total_amount`).

```
def get_orders_from_redis(limit=9999):
    """Get last X orders"""
    r = get_redis_conn()

    # 1) Lister les clés de type order:* et extraire les IDs
    keys = r.keys("order:*")
    order_ids = []
    for k in keys:
        if k.endswith(":items"):
            continue
        try:
            order_ids.append(int(k.split(":")[1]))
        except:
            continue

    # 2) Trier les IDs par ordre décroissant et limiter le nombre de résultats
    order_ids.sort(reverse=True)
    order_ids = order_ids[:limit]

    # 3) Lire les hashes correspondants en pipeline
    pipe = r.pipeline()
    for oid in order_ids:
        pipe.hgetall(f"order:{oid}")
    orders = pipe.execute()

    # 4) Retourner des objets (id, total_amount) pour la vue
    OrderRow = type("OrderRow", (), {})
    result = []
    for oid, h in zip(order_ids, orders):
        if not h:
            continue
        row = OrderRow()
        row.id = oid
        row.total_amount = float(h.get('total_amount', 0))
        result.append(row)
    return result
```

Question 3

Quelles méthodes avez-vous utilisées pour ajouter des données dans Redis ? Veuillez inclure le code pour illustrer votre réponse.

Pour ajouter des données dans Redis, j'ai utilisé:

1. `hset(key, mapping=...)` pour écrire le hash de la commande;
2. `delete(key)` pour réinitialiser la liste d'items avant d'écrire;
3. `rpush(key, value)` pour empiler chaque item (JSON) dans la liste

```
def add_order_to_redis(order_id, user_id, total_amount, items):  
    """Insert order to Redis  
    - order:{id} -> hash (user_id, total_amount)  
    - order:{id}:items -> list of JSON items (product_id, quantity, unit_price)  
    """  
    r = get_redis_conn()  
    # Hash + list of JSON items  
    order_key = f"order:{order_id}"  
    items_key = f"{order_key}:items"  
  
    # 1) Hash des metadonnees (cree ou remplace)  
    r.hset(order_key, mapping={  
        "user_id": int(user_id),  
        "total_amount": float(total_amount)  
    })  
  
    # 2) Liste des items (reinitialise)  
    r.delete(items_key) # Supprime les items existants, s'il y en a  
    for item in items:  
        """ Tolerons deux formes:  
        - dict: {'product_id': 1, 'quantity': 2}  
        - objet SQLAlchemy: OrderItem(product_id=1, quantity=2, unit_price=9.99)  
        """  
        if isinstance(item, dict):  
            product_id = int(item["product_id"])  
            quantity = float(item["quantity"])  
            unit_price = float(item["unit_price"])  
        else:  
            product_id = int(item.product_id)  
            quantity = float(item.quantity)  
            unit_price = float(item.unit_price)  
  
        r.rpush(items_key, json.dumps({  
            "product_id": product_id,  
            "quantity": quantity,  
            "unit_price": unit_price  
        }))  
    print(r)
```

Question 4

Quelles méthodes avez-vous utilisées pour supprimer des données dans Redis ? Veuillez inclure le code pour illustrer votre réponse.

La méthode `delete(key)` a été utilisée pour retirer les clés liées à une commande.

```
def delete_order_from_redis(order_id):  
    """Delete order from Redis"""  
    r = get_redis_conn()  
    r.delete(f"order:{order_id}")  
    r.delete(f"order:{order_id}:items")
```

Question 5

Si nous souhaitons créer un rapport similaire, mais présentant les produits les plus vendus, les informations dont nous disposons actuellement dans Redis sont-elles suffisantes, ou devrions-nous chercher dans les tables sur MySQL ? Si nécessaire, quelles informations devrions-nous ajouter à Redis ? Veuillez inclure le code pour illustrer votre réponse.

Les informations que nous disposons dans Redis sont effectivement suffisantes, car pour chaque commande, on peut agréger par `product_id` directement depuis Redis (sans lire MySQL).

```
def get_highest_spending_users(limit=10):  
    """Get report of best selling products for Redis only"""  
    r = get_redis_conn()  
  
    # 1) Recuprer les cles d'ordres  
    order_keys = [k for k in r.keys("order:*") if not k.endswith(":items")]  
  
    # 2) Lire tous les hashes en pipeline  
    pipe = r.pipeline()  
    for k in order_keys:  
        pipe.hgetall(k)  
    hashes = pipe.execute()  
  
    # 3) Agréger par user_id  
    expenses_by_user = defaultdict(float)  
    for h in hashes:  
        if not h: continue  
        try:  
            uid = int(h.get('user_id', 0))  
            total = float(h.get('total_amount', 0))  
        except Exception:  
            continue  
        expenses_by_user[uid] += total  
  
    # 4) Trier et limiter  
    get_highest_spending_users = sorted(  
        expenses_by_user.items(),  
        key=lambda x: x[1],  
        reverse=True  
   )[:limit]
```

```
# 5) Retourner une liste de dicts
Row = type("UserSpendingRow", (), {})
result = []
for uid, total in get_highest_spending_users:
    row = Row()
    row.user_id = uid
    row.total_spent = total
    result.append(row)

return result
```