Atelier Redis – Documentation Complète

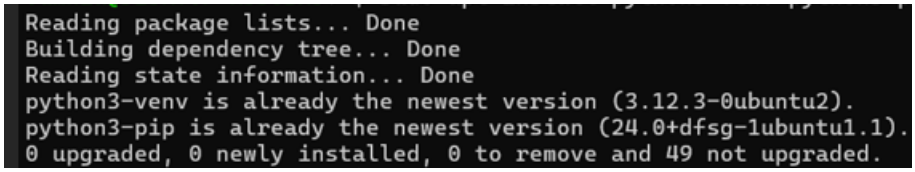
Plan de déroulement

1. Installation de Redis
2. Mise en place d'une réplication Redis
3. Développement d'une application web avec cache Redis
4. Vérifications et démonstrations

Partie 1 – Installation de Redis

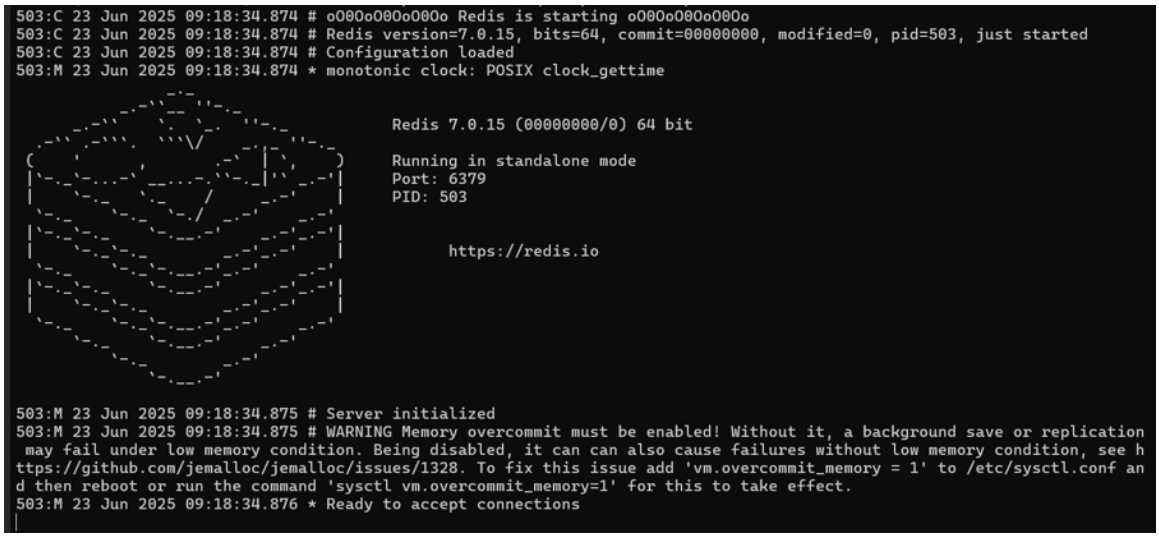
Installation de Redis sur Ubuntu avec la commande suivante :

sudo apt install redis-server



Lancement du serveur Redis maître :

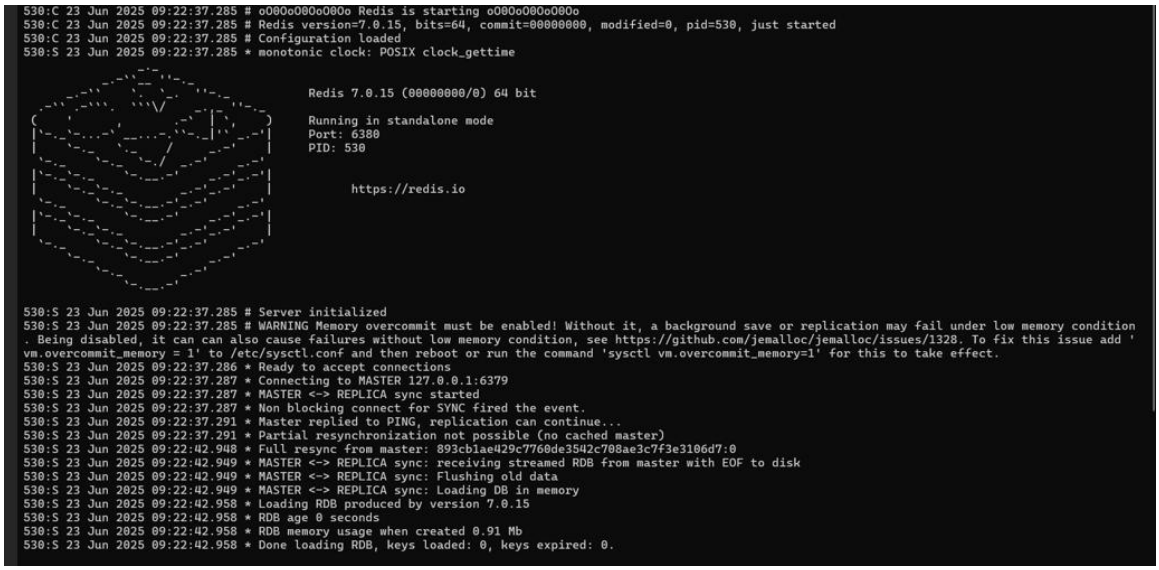
redis-server --port 6379 --requirepass "motdepasse"



Partie 2 – Architecture distribuée (réplication)

Lancement d’un serveur esclave Redis :

redis-server --port 6380 --slaveof 127.0.0.1 6379 --masterauth "motdepasse"



Vérification de la réplication :

redis-cli -p 6380 info replication

```
# Replication
role:slave
master_host:127.0.0.1
master_port:6379
master_link_status:up
master_last_io_seconds_ago:8
master_sync_in_progress:0
slave_read_repl_offset:126
slave_repl_offset:126
slave_priority:100
slave_read_only:1
replica_announced:1
connected_slaves:0
master_failover_state:no-failover
master_replid:893cb1ae429c7760de3542c708ae3c7f3e3106d7
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:126
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:126
```

Partie 3 – Application Web avec Cache Redis

Technologies utilisées : Python, Flask, redis-py.

Création d'une route /data/<key> avec une logique de cache :

```
 GNU nano 7.2                                                app.py *
from flask import Flask
import redis
import time

app = Flask(__name__)
cache = redis.Redis(host='localhost', port=6379, password='ton_mot_de_passe', decode_responses=True)

@app.route('/data/<key>')
def get_data(key):
    value = cache.get(key)
    if value:
        return f"[CACHE] {value}"
    else:
        time.sleep(2)
        value = f"data_pour_{key}"
        cache.setex(key, 60, value)
        return f"[SLOW] {value}"

if __name__ == '__main__':
    app.run(debug=True)
from flask import Flask, jsonify
import redis
import time
import logging


logging.basicConfig(level=logging.INFO)


app = Flask(__name__)


try:
    cache = redis.Redis(host='localhost', port=6379, password='ton_mot_de_passe', decode_responses=True)
    cache.ping()
    logging.info("Connexion à Redis réussie.")
except redis.RedisError as e:
```

```python
except redis.RedisError as e:
    logging.error("Erreur de connexion à Redis.", exc_info=e)
    cache = None

def get_from_cache(key):
    if not cache:
        return None
    try:
        return cache.get(key)
    except redis.RedisError as e:
        logging.error("Erreur lors de la lecture du cache.", exc_info=e)
        return None

def set_to_cache(key, value, ttl=60):
    if not cache:
        return
    try:
        cache.setex(key, ttl, value)
    except redis.RedisError as e:
        logging.error("Erreur lors de l'écriture dans le cache.", exc_info=e)

def simulate_slow_database(key):
    time.sleep(2)
    fake_data = {
        "foo": "valeur_de_foo",
        "bar": "valeur_de_bar"
    }
    return fake_data.get(key, f"data_pour_{key}")

@app.route('/data/<key>')
def get_data(key):
    start_time = time.time()

    value = get_from_cache(key)
    if value:
        elapsed = time.time() - start_time
        logging.info(f"Cache hit pour la clé: {key}")
```

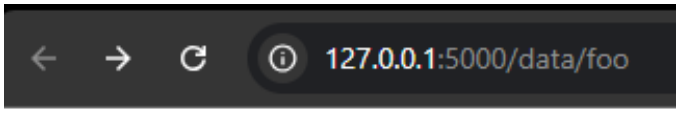
Lancement de l'application Flask.

Python app.py

```
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 282-458-232
```

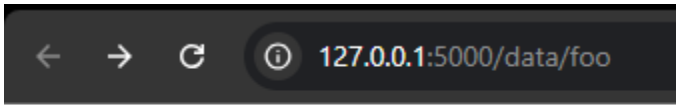
Partie 4 – Vérifications

Première requête (donnée absente du cache, réponse lente) :



[SLOW] data_pour_foo

Deuxième requête (donnée présente dans le cache, réponse rapide) :



[CACHE] data_pour_foo