# Integrating Redis into Your Recommendation System

This guide explains how to integrate Redis into your Flask-based recommendation system to improve performance by caching recommendations per user.

## Why Use Redis?

- Avoid recomputing recommendations on every request  
- Improve response time for frequently accessed recommendations  
- Store personalized recommendations per user with expiry  
- Reduce load on your ML models and database

## 1. Setup Redis

Use Docker to run Redis locally:

docker run -d --name redis-server -p 6379:6379 redis

## 2. Install Python Redis Client

Install redis-py in your Flask project:

pip install redis

## 3. Connect to Redis in Flask

Add this to your Flask app:

import redis  
  
redis\_client = redis.Redis(host='localhost', port=6379, db=0)  
CACHE\_EXPIRY = 10 \* 24 \* 60 \* 60 # 10 days

## 4. Caching Logic in Endpoints

Check Redis cache before computing recommendations:

cache\_key = f"recommendation:product:{user\_id}"  
cached = redis\_client.get(cache\_key)  
if cached:  
 return jsonify(json.loads(cached)), 200

If not cached, compute and store:

result = orchestrator.get\_recommendations(user\_id, top\_n=5)  
redis\_client.setex(cache\_key, CACHE\_EXPIRY, json.dumps(result))

## 5. Invalidate Cache After Order or Review

In Flask webhook or Kafka listener:

def invalidate\_user\_cache(user\_id):  
 for t in ['product', 'restaurant', 'full']:  
 key = f"recommendation:{t}:{user\_id}"  
 redis\_client.delete(key)

## Summary

Integrating Redis lets you cache and reuse recommendation results efficiently.  
Use Redis when user behavior changes (orders, reviews, feedback), and combine it with smart invalidation strategies.