

# Complete Working Example: Celery + Redis + MongoDB Join Pipeline

This document presents a fully self-contained example demonstrating how a Celery client enqueues a request into Redis, how a Celery worker performs a MongoDB join operation, and how the final result is stored back into Redis as a key-value entry.

## 1. Architecture Overview

The system follows an asynchronous request-processing model. Redis acts as the message broker, Celery workers execute background jobs, MongoDB performs native joins, and Redis also serves as an application-level cache for results.

- Client submits request using Celery `delay()`
- Redis stores the task in a queue
- Celery worker retrieves and executes the task
- Result stored in Redis key-value store

## 2. MongoDB Collections

Users collection document:

```
{ "_id": "u1", "name": "Alice", "email": "alice@example.com" }
```

Orders collection document:

```
{ "_id": "o101", "userId": "u1", "amount": 1200, "status": "PAID" }
```

## 3. Celery Configuration (`celery_app.py`)

Celery is configured with Redis as both broker and result backend. JSON serialization and late acknowledgements are enabled.

```
from celery import Celery

celery_app = Celery(
    "join_tasks",
    broker="redis://localhost:6379/0",
    backend="redis://localhost:6379/1",
)

celery_app.conf.update(
    task_serializer="json",
    result_serializer="json",
    accept_content=["json"],
    task_acks_late=True,
    worker_prefetch_multiplier=1,
)
```

## 4. Celery Worker Task (`tasks.py`)

The worker executes a MongoDB aggregation pipeline with \$lookup and stores the result in Redis as a key-value entry.

```
from celery_app import celery_app
from pymongo import MongoClient
import redis, json

mongo = MongoClient("mongodb://localhost:27017")
db = mongo.shop

redis_client = redis.Redis(
    host="localhost",
    port=6379,
    db=2,
    decode_responses=True
)

@celery_app.task(bind=True)
def join_users_orders(self, user_id):

    pipeline = [
        {"$match": {"_id": user_id}},
        {
            "$lookup": {
                "from": "orders",
                "localField": "_id",
                "foreignField": "userId",
                "as": "orders",
            }
        },
    ]
    result = list(db.users.aggregate(pipeline))

    redis_key = f"user_order_result:{user_id}"
    redis_client.set(redis_key, json.dumps(result), ex=300)

    return {
        "redis_key": redis_key,
        "count": len(result),
    }
```

## 5. Celery Client (client.py)

The client submits the asynchronous request without blocking.

```
from tasks import join_users_orders

task = join_users_orders.delay("u1")
print("Submitted task ID:", task.id)
```

## 6. Worker Startup

The worker is started using the Celery command-line interface.

```
celery -A celery_app worker -l info
```

## 7. Redis Key–Value Result

The joined data is stored in Redis as JSON. This storage is independent of the Celery result backend.

```
Key: user_order_result:u1
```

```
Value:  
[  
  {  
    "_id": "u1",  
    "name": "Alice",  
    "email": "alice@example.com",  
    "orders": [  
      {  
        "_id": "o101",  
        "amount": 1200,  
        "status": "PAID"  
      }  
    ]  
  }  
]
```

## 8. Client Result Retrieval (fetch\_result.py)

The client retrieves the data directly from Redis.

```
import redis, json  
  
r = redis.Redis(  
    host="localhost",  
    port=6379,  
    db=2,  
    decode_responses=True  
)  
  
data = r.get("user_order_result:u1")  
print(json.loads(data))
```

## 9. Design Notes

- Redis acts as both broker and cache
- MongoDB performs the join operation
- Tasks must be idempotent
- TTL prevents stale cache buildup