**Redis Cluster Configuration for Task Durability**

1. **Enable Append-Only File (AOF) Persistence**

This ensures Redis writes every operation to disk and can replay it on restart.

In your Redis cluster nodes' redis.conf:

appendonly yes

appendfsync everysec

* + appendfsync always → slow but safest.
  + appendfsync everysec → good trade-off for durability vs performance.

1. **Cluster Configuration**

Make sure:

* + You’re **not sending tasks to replicas** (which might lag).
  + All **writes go to the primary** node of the shard.
  + Redis Sentinel or Cluster has **automatic failover** set up.

**✅ Celery Configuration for Redis Broker Fault Tolerance**

celery\_app.conf.update(

task\_acks\_late=True,

task\_reject\_on\_worker\_lost=True,

worker\_max\_tasks\_per\_child=100,

timezone=Config.TIMEZONE,

broker\_connection\_retry\_on\_startup=True,

task\_default\_delivery\_mode='persistent',

broker\_transport\_options={

'visibility\_timeout': 3600,

'socket\_timeout': 30,

'retry\_on\_timeout': True

}

)

**🧠 What Each Setting Does**

| **Setting** | **Purpose** |
| --- | --- |
| task\_acks\_late=True | Acknowledge tasks only **after successful execution** (avoids task loss if worker crashes). |
| task\_reject\_on\_worker\_lost=True | If worker dies mid-task, requeue the task. |
| task\_default\_delivery\_mode='persistent' | Ensures Redis queues store tasks **persistently**. |
| broker\_transport\_options['visibility\_timeout'] | Requeue tasks if worker doesn’t ack in this time (used in crashes/timeouts). |
| retry\_on\_timeout | Retry if Redis times out when publishing/consuming tasks. |

Redis **does not guarantee exactly-once delivery** — tasks may be **retried or duplicated** under crash/failure scenarios.

**But wait — isn’t Redis an in-memory store?**

Yes — but it **can persist data to disk** using:

* **AOF (Append Only File)** or
* **RDB (Redis Database snapshots)**

When Celery sets delivery\_mode='persistent', it adds a message header flag, and the **Redis broker** *should* treat the message as one to persist. However:

**🚨 Important Caveat**

Unlike **RabbitMQ**, **Redis does not actually support persistent delivery the same way.**

Redis **does not differentiate between transient and persistent messages**, because Redis queues (lists) persist based on Redis’ own persistence settings — not message flags.

So this setting:

celery\_app.conf.task\_default\_delivery\_mode = 'persistent'

✅ **Does work** for **AMQP brokers** like RabbitMQ  
⚠️ **Has no effect** when using Redis broker — it's silently ignored.

**So how to persist tasks with Redis then?**

You must:

1. **Enable Redis persistence** yourself:
   * appendonly yes in your redis.conf
   * appendfsync everysec for balance of safety/perf
2. **Ensure Redis cluster replicas + failover** are configured
3. Use Celery’s retry/durability settings:
   * acks\_late=True
   * task\_reject\_on\_worker\_lost=True
   * visibility\_timeout via broker\_transport\_options

**✅ What should you do?**

Even though task\_default\_delivery\_mode='persistent' has no effect on Redis, you **should still keep it** in your code if you may migrate to RabbitMQ later or want portability.

Redis Cluster shards data across multiple nodes and supports:

* **Partitioning** (keyspace divided into 16384 hash slots)
* **Replication** (each master has replicas)
* **Automatic failover** (if a master fails, one of its replicas is promoted)

✅ **Purpose:** Both **scalability** and **high availability**

**If using Redis Cluster:**

Use redis:// connection string to point to any **startup node**, and ensure:

CELERY\_BROKER\_URL = "redis://<one\_cluster\_node>:6379/0"

Install **redis-py-cluster** if needed (for client support). However, Celery’s native Redis support may not yet fully support Redis Cluster out of the box — use [kombu-redis-cluster](https://github.com/celery/kombu/issues/1417)

**What does celery-redis-cluster do?**

celery-redis-cluster is a custom transport plugin for Celery that adds support for Redis Cluster as a message broker. The default Celery Redis transport does **not** natively support Redis Cluster due to the complexity of routing messages to specific nodes. This plugin bridges that gap.

**Key Features:**

* Enables Celery to use **Redis Cluster** as a broker (not just as a single-node or sentinel-based setup).
* Properly routes commands to appropriate cluster nodes.
* Handles Redis Cluster topology discovery.

**When to use it:**

* You have a production Redis Cluster setup.
* You cannot switch to RabbitMQ or another broker.
* You want to keep the benefits of Celery (task queues, retries, etc.) with Redis as broker.

**✅ How to enable replication and automatic failover in Redis Cluster**

Redis Cluster provides built-in support for both:

**🔁 Replication:**

Each master node in the cluster should have one or more replica nodes. You define these when creating the cluster or add them later.

Example (when using redis-cli):

redis-cli --cluster add-node 127.0.0.1:7004 127.0.0.1:7000 --cluster-slave --cluster-master-id <master\_node\_id>

This makes node 7004 a replica of the given master node.

**⚠️ Automatic Failover:**

Redis Cluster will automatically promote a replica to master if its master node fails (this is different from Sentinel).

You need:

* At least one replica per master.
* cluster-enabled yes in your Redis config.
* cluster-node-timeout set properly (default is 15000ms).

**🚀 How to Make the Most of Redis Cluster with Celery**

Since you can’t use Sentinel or switch brokers:

1. **Install celery-redis-cluster:**

pip install celery-redis-cluster

1. **Update Celery config to use the plugin:**

broker\_transport\_options = {

"cluster": True,

"startup\_nodes": [

{"host": "redis-node1", "port": "7000"},

{"host": "redis-node2", "port": "7001"},

{"host": "redis-node3", "port": "7002"}

]

}

CELERY\_BROKER\_URL = "redis-cluster://"

1. **Use task\_acks\_late=True and task\_reject\_on\_worker\_lost=True** for delivery guarantees.
2. **Monitor your Redis Cluster using redis-cli, RedisInsight, or Prometheus.**

In **Redis Cluster**, replication and automatic failover are **built-in by design**. Unlike standalone Redis or Sentinel-based setups where you configure replication and failover manually, Redis Cluster:

* Automatically replicates data to **replica nodes**.
* Handles **automatic failover** if a master node fails.

**✅ How Redis Cluster Handles Replication and Failover**

1. **Each master** has one or more **replicas**.
2. If a master fails, the **cluster automatically promotes a replica** to master.
3. Clients are updated with the new slot mappings automatically.
4. No external tooling like Redis Sentinel is needed.

**🔧 Steps to Enable Replication & Automatic Failover in Redis Cluster**

**1. Create Nodes Configuration**

Redis Cluster typically consists of:

* **3 master nodes**
* **3 replica nodes** (1 replica per master)

You need at least **6 nodes** (for minimal redundancy):

Node1 → Master A

Node2 → Master B

Node3 → Master C

Node4 → Replica of A

Node5 → Replica of B

Node6 → Replica of C

**2. redis.conf Configuration for Each Node**

Set in each node’s redis.conf:

cluster-enabled yes

cluster-config-file nodes.conf

cluster-node-timeout 5000

appendonly yes

Optional tuning:

replica-priority 100 # lower value means less likely to be promoted

**3. Create the Cluster**

Use the redis-cli tool with the --cluster flag:

redis-cli --cluster create \

127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 \

127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005 \

--cluster-replicas 1

This:

* Creates a 3-master, 3-replica cluster.
* Automatically assigns replicas to each master.
* Enables failover.

**4. Verify Cluster Status**

redis-cli -c -p 7000 cluster info

redis-cli -c -p 7000 cluster nodes

Look for:

* master and slave roles
* Replica relationships
* Node health

**5. Simulate a Master Failure**

You can kill a master Redis process and observe:

* A replica gets promoted to master
* Cluster remains operational

**🧠 Best Practices**

* Use **at least 3 master and 3 replica nodes**.
* Set appendonly yes to persist data to disk.
* Use **redis-trib.rb** or redis-cli for safe cluster creation and resharding.
* Monitor cluster using tools like:
  + redis-cli --cluster info
  + Prometheus + Grafana

**⚠️ Note for Celery**

Celery does **not natively support Redis Cluster**. You need a **Redis proxy like** [**Twemproxy**](https://github.com/twitter/twemproxy) or **Redis Cluster client library** (e.g., [celery-redis-cluster](https://github.com/Grokzen/redis-py-cluster#celery-redis-backend-support)).

**What is AOF (Append Only File) in Redis?**

**AOF (Append Only File)** is a Redis persistence mechanism that logs every write operation the server receives, and replays this log at startup to rebuild the dataset.

**Key Properties:**

* Writes are logged as Redis commands (like SET, LPUSH, etc.).
* Safer than RDB in terms of durability (can be configured to flush every write).
* Can be used **with** or **instead of** RDB snapshots.

**💡 How does AOF help in Redis Cluster?**

**In Redis Cluster:**

* **Each node (master or replica)** maintains its own AOF file if AOF is enabled.
* If a node **crashes and restarts**, AOF can **recover** its last consistent state (if not replaced by a replica during failover).
* If a **replica gets promoted** (failover), having AOF lets it start with the most recent persisted data.

**❓ Do you need AOF if you already have replicas?**

**🔎 No, replication alone does not guarantee durability.**

| **Feature** | **Replication** | **AOF** |
| --- | --- | --- |
| Prevents downtime | ✅ Yes | ❌ No |
| Prevents data loss (e.g., crash) | ❌ No | ✅ Yes (to some extent) |
| Helps with failover | ✅ Yes | ✅ Yes |
| Helps node restore state on restart | ❌ Partial (may resync from master) | ✅ Yes |

**Why?**

* Redis replication is in-memory and asynchronous:
  + If a master crashes before a replica syncs → **data is lost**.
* If AOF is enabled → node can **replay recent commands** to recover.

**🧠 When should you enable AOF in Redis Cluster?**

✅ **Enable AOF if:**

* You want to minimize data loss after a crash.
* Your use case requires **stronger durability guarantees**.
* You don’t want to rely solely on replicas (which may lag behind).

❌ **You can skip AOF if:**

* Your data is not critical and can be rebuilt.
* You're using Redis purely as a cache.
* You're okay with minimal data loss (e.g., seconds).

**⚙️ Best Practice**

In **production Redis Cluster**, many teams:

* Enable **replication** for high availability.
* Enable **AOF with appendfsync everysec** for durability with performance balance.
* Periodically **rewrite AOF** to keep it compact (auto-aof-rewrite-percentage).

**TL;DR:**

* **AOF improves durability** by persisting commands to disk.
* **Replication improves availability**, but not persistence.
* **Together**, AOF + replicas give better **failover + crash recovery**.
* In production, **enable both** unless you’re using Redis strictly as a cache.

**Why Do You Need celery-redis-cluster?**

Celery’s default Redis broker (redis://) **does not natively support Redis Cluster**. Redis Cluster uses **key-based partitioning** across multiple nodes (shards), and Celery’s default Redis transport assumes a **single-node or Sentinel-managed** Redis deployment.

Without special support, using Redis Cluster as a broker may result in:

**❌ Errors like:**

* MOVED or ASK errors
* Inability to handle sharded keys correctly
* Lost tasks if a node doesn't own the key slot
* Celery startup failures

This is where celery-redis-cluster comes in:

**✅ What Is celery-redis-cluster?**

celery-redis-cluster is a **custom Celery broker transport** that adds **native support** for Redis Cluster.

**It enables:**

* Seamless communication with **multiple Redis nodes**
* Slot-aware routing of tasks
* Retry handling for MOVED/ASK redirects
* Usage of Redis Cluster in production for **high availability and scalability**

It’s built on top of redis-py-cluster.

**📦 Installation**

bash

CopyEdit

pip install celery-redis-cluster redis-py-cluster

Make sure you're using:

* celery >= 5.1
* redis-py-cluster >= 2.1.0

**⚙️ Configuration**

Let’s say your Redis Cluster has 6 nodes: 3 masters and 3 replicas.

**🔑 Example cluster node addresses:**

python

CopyEdit

# config.py

REDIS\_CLUSTER\_NODES = [

"redis://redis-node-1:6379/0",

"redis://redis-node-2:6379/0",

"redis://redis-node-3:6379/0",

]

**🧠 Celery App Configuration:**

python

CopyEdit

from celery import Celery

from rediscluster import RedisCluster

from config import REDIS\_CLUSTER\_NODES

# NOTE: Must set broker\_transport to use custom transport

celery\_app = Celery("myapp")

celery\_app.conf.update(

broker\_transport="rediscluster", # << This is key!

broker\_url=",".join(REDIS\_CLUSTER\_NODES), # comma-separated list

result\_backend="redis://redis-node-1:6379/0", # or use Redis Sentinel if needed

task\_default\_delivery\_mode='persistent',

broker\_transport\_options={

"cluster": {

"startup\_nodes": REDIS\_CLUSTER\_NODES,

"max\_connections": 100,

"readonly": False, # Only master writes

}

},

)

**🔐 Notes:**

* All broker\_url addresses must point to **Redis Cluster masters**.
* Avoid using replicas for publishing tasks.
* Celery uses **hash slots**, so internally it knows which node to hit.

**💥 What Happens Without celery-redis-cluster?**

Without it:

* Celery may connect to a **single node** blindly.
* Tasks routed to incorrect shards may **fail silently or hang**.
* Failover may not work properly, resulting in **downtime**.

**🛠 Features of celery-redis-cluster**

| **Feature** | **Supported** |
| --- | --- |
| Redis Cluster topology | ✅ Yes |
| Key slot-aware routing | ✅ Yes |
| Handling MOVED/ASK responses | ✅ Yes |
| Task persistence | ✅ With task\_default\_delivery\_mode='persistent' |
| TLS/SSL | ✅ With broker\_use\_ssl=True |
| Retry on connection failure | ✅ Supported |

**🔒 Optional: Enable Persistence & Reliability**

Add these in your celery\_app.conf.update():

python

CopyEdit

task\_acks\_late=True, # Retry tasks on worker crash

task\_reject\_on\_worker\_lost=True, # Don't lose task if worker dies

broker\_connection\_retry\_on\_startup=True,

task\_default\_delivery\_mode='persistent',

**🧪 Test the Setup**

You can verify that the Redis cluster is working:

bash

CopyEdit

celery -A your\_app worker --loglevel=info

And ensure Redis logs or redis-cli -c shows traffic distributed among master nodes.

**📦 Summary**

| **Item** | **Info** |
| --- | --- |
| Package name | celery-redis-cluster |
| Why needed? | Celery doesn’t support native Redis Cluster |
| Broker transport | rediscluster |
| Key config | broker\_transport\_options['cluster']['startup\_nodes'] |
| Production ready? | ✅ Yes (widely used in Redis Cluster environments) |