Celery - Practice Sheet

10 Q&A on brokers, task routing, retries, Beat, and testing

Q1. What is Celery and what role does a message broker play?

A: Celery is a distributed task queue. When you call task.delay(), Celery serializes the call and sends it to a broker (Redis or RabbitMQ). Worker processes consume messages from the broker, execute tasks, and optionally store results in a backend.

Q2. What is the difference between delay() and apply_async()?

A: task.delay(arg1, arg2) is shorthand for apply_async((arg1, arg2)). apply_async gives you full control: countdown (delay in seconds), eta (absolute time), queue name, priority, retry policy, and task id override.

Q3. How does Celery handle task retries and what is autoretry_for?

A: Call self.retry(exc=e, countdown=5, max_retries=3) inside a task on failure. @app.task(autoretry_for=(RequestException,), retry_backoff=True, max_retries=5) automates this: Celery retries on the listed exceptions with exponential backoff.

Q4. What is Celery Beat and how do you schedule a periodic task?

A: Celery Beat is a scheduler that enqueues tasks on a schedule. Define entries in beat_schedule: {'run-daily': {'task': 'app.tasks.cleanup', 'schedule': crontab(hour=0, minute=0)}}. Beat requires a persistent store (db or Redis) to track last-run times.

Q5. What are Celery chains, groups, and chords?

A: A chain(t1.s(), t2.s()) pipes output of t1 as input to t2. A group([t1.s(), t2.s()]) runs tasks in parallel. A chord(group, callback) runs the group in parallel then passes all results to callback useful for map-reduce-style workflows.

Q6. How do you route tasks to specific queues in Celery?

A: Define queues in the task_queues config. Set task_routes = {'myapp.tasks.heavy': {'queue': 'heavy_queue'}}. Start workers with --queues heavy_queue to consume from that queue. This segregates slow tasks from fast ones for independent scaling.

Q7. How does Celery handle task result storage?

A: Set result_backend = 'redis://localhost' (or 'db+postgresql://...'). After completion, AsyncResult(task_id).get() blocks until the result is available. Results are stored for result_expires seconds (default 24 h) then purged.

Q8. What is the Celery Flower monitoring dashboard?

A: Flower is a real-time web UI for Celery. Run celery -A app flower. It displays active workers, task rates, queue lengths, failed task details, and lets you revoke pending tasks or rate-limit workers from the browser.

Q9. How do you test Celery tasks without a running broker?

A: Set task_always_eager = True in the test config. This causes .delay() and .apply_async() to execute the task synchronously in the calling process without a broker. For unit tests of task logic only, call the function directly: mytask(args) skipping Celery entirely.

Q10. What are Celery soft and hard time limits?

A: soft_time_limit raises SoftTimeLimitExceeded inside the task, giving it a chance to clean up. hard time_limit (task_time_limit) sends SIGKILL to the worker process if the task exceeds the limit, with no cleanup possible. Prefer soft limits.