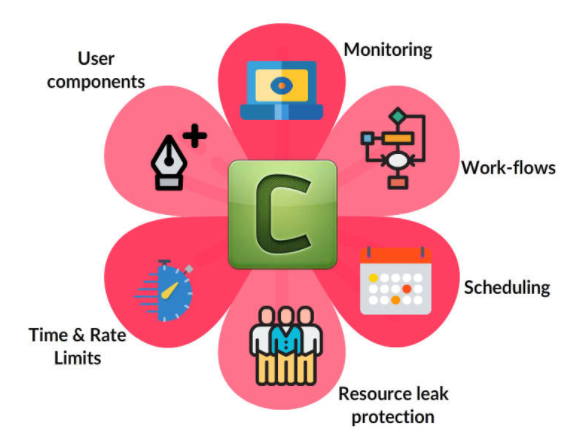
Celery

**Definition**:

**Celery** – helps in managing distributed task queue.

**Features**:



* **Monitoring** – built-in and external tools monitor events
* **Scheduling** – run a task immediately, at a specified time or set periodic runs
* **Work-flows** – Create simple or complex workflows through grouping, chaining, chunking
* **Resource** **leak** **protection** – Set max tasks for user tasks leaking resources and prevent leaking.
* **Time** **&** **Rate** **Limits** – control how many tasks to be executed or how long task can be executed.
* **User** **components** – Worker component can be customized, additional components can be defined by user
* **Web Framework Integration** - Celery integrates with different web frameworks easily through integration packages. These packages help in easy message subscription and consumption
* Pyramid - pyramid\_celery
* Pylons - celery-pylons
* web2py - web2py-celery
* tornado - tornado-celery
* Flask - in-built package
* Django - in-built package(from 3.1)

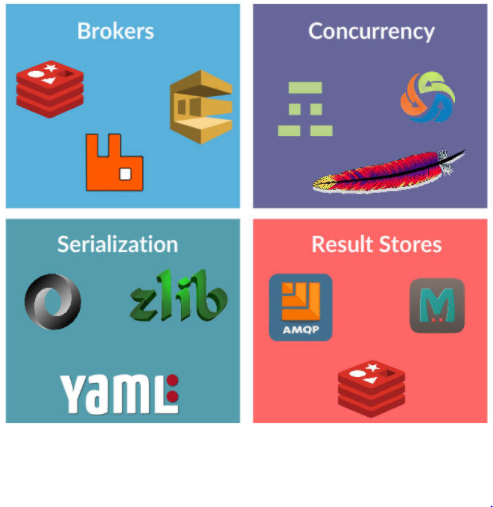
**Why** **celery**:

Celery is

* **Simple** – easy to use and maintain, doesn’t need configuration files
* **Highly** **Available** – Working nodes will automatically retry in the event of connection loss or failure
* **Fast** – single celery worker can process millions of tasks per minute
* **Flexible** – Can be extended or used as is for custom implementations

**Advantage:**

Celery supports various technologies / tools



* **Brokers** – RabbitMQ, Redis, Amazon SQS and more(experimental)
* **Concurrency** – prefork (multiprocessing), Eventlet, gevent, solo (single thread)
* **Result** **Stores** – AMQP, Redis, Memcached, SQLAlchemy, Django ORM, Cassandra, Elastic Search
* **Serialization** - pickle, json, yaml, msgpack, zlib, bzip2 compression, Cryptographic message signing

**How Celery work?**

Celery work is based on Master-Worker Architecture where master is client which generates tasks and takes processed output and worker is celery workers which performs the tasks and puts results in backend.(Your words)

* Celery’s **distributed task queue** architecture is a form of **master-worker architecture** using queues for work requests, queue or database storage to hold results. A **message queue broker** like RabbitMQ or a database like Redis is required.
* The master process (client or producer) **puts work requests** (tasks) into one of the task queues and **fetches results** from a result backend - queue or database
* Worker processes (celery workers), **subscribe to some or all** of the task queues to know what **work to perform** and **put their results** into the result backend.

**Benefits of Master and Workers**

* Need not know **which machines** the processes are triggered from
* Need to track **where the queues are** and either post or fetch work from.
* **Easy to scale** by adding more workers
* Can be written in **different coding languages** e.g. Python as a master generates work, and C as a worker performs work.