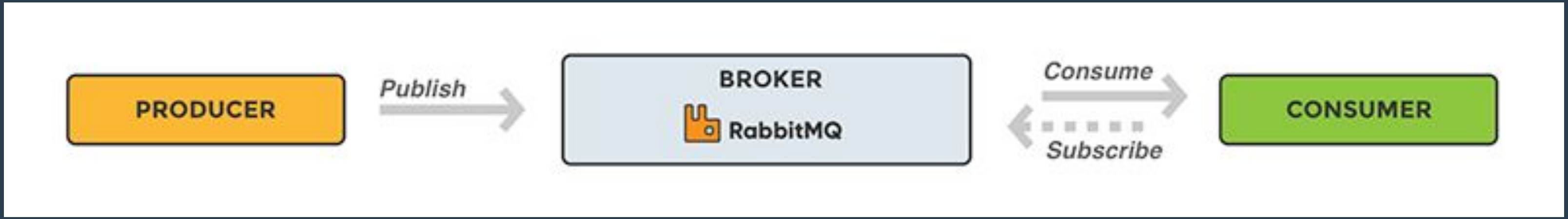
Celery + RMQ + MongoDB

●

Highlights from an ETL Story

Base Architecture



Goal: Build a long running processing server suitable for extracting data out of audio files or a worker testing websites periodically.

Building Blocks

- MongoDB; Performant, NoSQL, map-reduce ready.
- Celery; De-facto standard for batch & async processing within Python and Django.
- RabbitMQ; The ‘broker’ part, used by Celery.
- Flower; Well featured celery web monitor.
- Ansible; deployment automation.

MongoDB

- Used Ubuntu 18.10 provided package (3.6).
  - But, better to use latest stable.
- Use GridFS to store BLOBs onto MongoDB;
  - Streaming ready, OS & Cloud agnostic.
  - No Cluster, No Journaling (-performance).
- No NFS.
- Mongo-Express for UI:
  - https://github.com/mongo-express/mongo-express

Celery

- Default pip install version; 4.2.0
  - Use master to test bugs and fixes.
- `setup.py` to deploy celery project as an egg.

- For AMQP performance; librabbitmq (also avail @ pkg mngr).

- amqp-tools for cmd line test & debug.

- Systemd is better than supervisord;
  - https://github.com/celery/celery/issues/102
- Use Flower.

Celery;tasks

- No retry strategy (reprocessing ftw).
- Tasks: Idempotency & Atomicity are your new best friends.
- Use `result_backend`; for monitoring, tombstone inspection in MongoDB (`JSON` ftw).
- Use `task_time_limit;` fixes RAM issues, guarantee resource availability, hanging tasks.
- `worker_concurrency;`
    - `IO bound =(#CPU * 2), Compute bound = (#cores - 1)`

Celery; tasks

- -=Log=- like you mean it (-and love it, `get_task_logger` ftw).

- Use separate Queues for long running tasks; especially if your workload is strictly split between very short and long tasks;
  - http://docs.celeryproject.org/en/latest/userguide/routing.html#automatic-routing

- Celery trades off RAM for performance; to keep it sane:
  - `worker_max_tasks_per_child`
  - `worker_max_memory_per_child`
  - Set value to reflect requirements, else CPU will spike as workers will be busy killing child processes.

Celery;tasks::debugging

- Multiprocessing distorts TTY reality, more so if system is remote. Keeping your sanity:
  - `task_always_eager`
  - `task_eager_propogates` (unswallow exceptions).

- `from celery.contrib import rdb;`
  - Exceptions tend to plague certain code paths more than others; use cross-project debug flag. Life saver debugging production.

RMQ

- Use HAProxy instead of AWS ELB in front of RMQ.
    - https://groups.google.com/d/msg/rabbitmq-users/bl--2ba1F_0/E9_Is1KiAgAJ
- Keep queues as short as possible:
    - Trade performance with lazy_queues (disk bound).
- Use `rabbitmq_management`.
- Keep payloads small.
- Prefer VPC pairing to TLS.

RMQ

- Use latest stables; Both for Erland and RabbitMQ;
    - https://www.erlang-solutions.com/resources/download.html
    - https://www.rabbitmq.com/install-debian.html#apt

- Drop `guest` user for production;
    - Favor user per app;
    - https://www.rabbitmq.com/access-control.html

- Don't modify `vm_memory_high_watermark;`
    - Try to make messages and queuing time minimized.
    - Failing to leave enough memory can have adverse effects on OS and file system operations.

- Limit queue size to anticipate the unexpected:
    - https://www.rabbitmq.com/maxlength.html

Laten

- https://github.com/sivang/laten-fw
- Use as a basis to create your out-of-band processing server.
- Production ready audio file example coming soon.
- sivan@vitakka.co