Workflow of a Python Celery Project Using Redis and MySQL for Scheduled LinkedIn Posts

This document outlines the workflow of a Python Celery project that utilizes Redis as a message broker and MySQL for storing scheduled LinkedIn post jobs. The system is designed to check for upcoming posts every 5 minutes, and when a post is due, a Celery worker process is initiated to handle several pre and post tasks surrounding the LinkedIn posting process.

Overview

The project leverages Celery for asynchronous task management, Redis for message brokering, and MySQL for persistent storage of scheduled jobs. The architecture allows for efficient handling of scheduled LinkedIn posts, ensuring that tasks are executed at the right time with necessary preparations and follow-ups.

Components

1. **Celery**: A distributed task queue that allows for the execution of tasks asynchronously.
2. **Redis**: A fast in-memory data structure store used as a message broker for Celery.
3. **MySQL**: A relational database management system used to store scheduled LinkedIn post jobs.

Workflow

1. Job Scheduling

- Users schedule LinkedIn posts by entering the content and the desired posting time into a web interface.
- The scheduled jobs are stored in a MySQL database with fields such as **post_content**, **scheduled_time**, and **status**.

2. Celery Beat

- A Celery Beat scheduler runs every 5 minutes to check the MySQL database for any posts that are due to be published.
- It queries the database for jobs where the **scheduled_time** is less than or equal to the current time and where the **status** is not marked as completed.

3. Task Execution

- For each job that is due, Celery Beat sends a message to the Redis message broker, triggering a Celery worker to start processing the job.
- The worker executes a series of pre-tasks, which may include:
    - Validating the post content.
    - Checking for any scheduled conflicts.
    - Preparing the LinkedIn API credentials.

4. Posting to LinkedIn

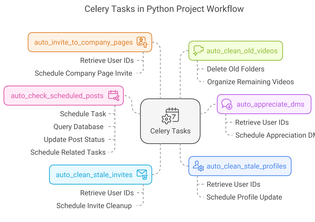
- Once the pre-tasks are successfully completed, the worker proceeds to post the content to LinkedIn using the LinkedIn API.
- The post is made with the content retrieved from the MySQL database.

5. Post-Task Handling

- After the post is made, the worker executes several post-tasks, which may include:
    - Updating the job status in the MySQL database to indicate that the post has been successfully published.
    - Logging the post details for future reference.
    - Sending notifications to users about the successful posting.

6. Error Handling

- In case of any errors during the pre-tasks, posting, or post-tasks, the worker logs the error details and updates the job status to reflect the failure.
- Notifications may also be sent to users in case of failures, allowing them to take corrective actions.



Workflow Description

1. **auto_check_scheduled_posts**
    1. **Description**: Checks if there are any posts to publish.
    2. **Docker Service/Container**: **celery_beat**
    3. **Workflow**:
        1. **celery_beat** schedules the task.
        2. The task queries the database for posts scheduled between yesterday and the next 20 minutes.
        3. For each post found, it updates the post status to "scheduled".
        4. Schedules the **post_to_linkedin** task to publish the post at the scheduled time.
        5. Schedules the **automate_commenting** and **automate_profile_viewer_engagement** tasks to run before the post is published.
2. **auto_appreciate_dms**
    1. **Description**: Sends appreciation DMs for each active user.
    2. **Docker Service/Container**: **celery_beat**
    3. **Workflow**:
        1. **celery_beat** schedules the task.
        2. The task retrieves all active user IDs.
        3. For each user, it schedules the **automate_appreciation_dms_for_user** task to run for 5 minutes.
3. **auto_clean_stale_invites**
    1. **Description**: Cleans up stale invites for each active user.
    2. **Docker Service/Container**: **celery_beat**
    3. **Workflow**:
        1. **celery_beat** schedules the task.
        2. The task retrieves all active user IDs.
        3. For each user, it schedules the **clean_stale_invites** task.
4. **auto_clean_stale_profiles**
    1. **Description**: Cleans up stale profiles for each active user.
    2. **Docker Service/Container**: **celery_beat**
    3. **Workflow**:
        1. **celery_beat** schedules the task.
        2. The task retrieves all active user IDs.
        3. For each user, it schedules the **update_stale_profile** task.
5. **auto_invite_to_company_pages**
    1. **Description**: Sends invites to the company page for each active user who has a LinkedIn company page.
    2. **Docker Service/Container**: **celery_beat**
    3. **Workflow**:
        1. **celery_beat** schedules the task.
        2. The task retrieves all active user IDs.
        3. For each user, it schedules the **automate_invites_to_company_page_for_user** task.
6. **auto_clean_old_videos**
    1. **Description**: Cleans up old videos in the selenium folder.
    2. **Docker Service/Container**: **celery_beat**
    3. **Workflow**:
        1. **celery_beat** schedules the task.
        2. The task deletes folders in the selenium directory that are older than a specified number of days.
        3. Organizes remaining videos by name and timestamp.

Celery Tasks and Docker Services/Containers

1. **post_to_linkedin**
    1. **Description**: Posts to LinkedIn using the LinkedIn API.
    2. **Docker Service/Container**: **celery_worker**
    3. **Workflow**:
        1. Logs in to LinkedIn.
        2. Publishes the post.
        3. Updates the database with the post status.
        4. Schedules the **automate_reply_commenting** task to reply to comments on the post.
2. **automate_commenting**
    1. **Description**: Automates commenting on posts.
    2. **Docker Service/Container**: **celery_worker**
    3. **Workflow**:
        1. Logs in to LinkedIn.
        2. Navigates to the feed.
        3. Comments on posts in the feed.
3. **automate_profile_viewer_engagement**
    1. **Description**: Engages with users who viewed the profile.
    2. **Docker Service/Container**: **celery_worker**
    3. **Workflow**:
        1. Logs in to LinkedIn Navigates to the profile views page.
        2. Engages with users who viewed the profile within the last day.
4. **automate_appreciation_dms_for_user**
    1. **Description**: Sends appreciation DMs for a specific user.
    2. **Docker Service/Container**: **celery_worker**
    3. **Workflow**:
        1. Logs in to LinkedIn.
        2. Accepts connection requests.
        3. Sends appreciation DMs to new connections.
5. **clean_stale_invites**
    1. **Description**: Cleans up stale invites for a specific user.
    2. **Docker Service/Container**: **celery_worker**
    3. **Workflow**:
        1. Logs in to LinkedIn.
        2. Cleans up stale invites.
6. **update_stale_profile**
    1. **Description**: Updates the profile of a specific user.
    2. **Docker Service/Container**: **celery_worker**
    3. **Workflow**:
        1. Logs in to LinkedIn.
        2. Updates the profile.
7. **automate_invites_to_company_page_for_user**
    1. **Description**: Sends invites to the company page for a specific user.
    2. **Docker Service/Container**: **celery_worker**
    3. **Workflow**:
        1. Logs in to LinkedIn.
        2. Sends invites to the company page.
8. **automate_reply_commenting**
    1. **Description**: Replies to comments on a post.
    2. **Docker Service/Container**: **celery_worker**
    3. **Workflow**:
        1. Logs in to LinkedIn.
        2. Navigates to the post.
        3. Replies to comments on the post.
9. **send_private_dm**
    1. **Description**: Sends a private DM to a profile.
    2. **Docker Service/Container**: **celery_worker**
    3. **Workflow**:
        1. Logs in to LinkedIn.
        2. Sends a private DM to the specified profile.
10. **invite_to_connect**
    1. **Description**: Sends a connection request to a profile.
    2. **Docker Service/Container**: **celery_worker**
    3. **Workflow**:
        1. Logs in to LinkedIn.
        2. Sends a connection request to the specified profile.
11. **engage_with_profile_viewer**
    1. **Description**: Engages with a profile viewer.
    2. **Docker Service/Container**: **celery_worker**
    3. **Workflow**:
        1. Logs in to LinkedIn.
        2. Engages with the specified profile viewer.

Conclusion

This workflow provides a robust solution for scheduling and posting content to LinkedIn using Python Celery, Redis, and MySQL. By automating the process and ensuring that tasks are executed efficiently, users can manage their LinkedIn presence effectively without manual intervention. The combination of Celery's task management capabilities and Redis's fast message brokering ensures that posts are made on time, while MySQL provides reliable storage for job scheduling.