robopager

Release 0.1.0

Equinox Fitness

May 12, 2021

CONTENTS

Robopager is a job monitor and notification tool designed for daily email check and job latency check. The daily email check function can help you to monitor your inbox and whether you receive the expected emails at a certain time on a daily basis. The job latency check function helps to monitor every few seconds or minutes whether your scheduled workflows finish on time. If emails are not received in an expected timeframe or the scheduled jobs don't complete on time, you will receive the alerts from PagerDuty in email, text message or phone call depending on your choice.

CHAPTER

ONE

INSTALLATION

Robopager requires Python 3.6+

```
python3 -m venv <virtual environment name>
source <virtual environment name>/bin/activate
pip install robopager
```

CHAPTER

TWO

SETUP INSTRUCTIONS

Before start using this tool, you need to configure the settings in the **etl.cfg** and create your **checklist.yaml** file. You can find example files above and change to your own values. Detailed explanations as below.

2.1 1. etl.cfg

This file contains the loggin or server information of the applications or services that will be used in this tool, such as your monitored email, PagerDuty service and other servers. It also has the settings for you to choose the function mode.

- google_apps: enter the email address you want to monitor and use b64encode to encrypt your password for security concerns
- pager_duty: enter the information of the subdomain you will use and the API access key for authentication
- enable_redis: enter "True" if you want to use Redis, which is highly recommended because of API rate limit in PagerDuty; otherwise "False". Please refer to Note section for more information about Redis and please refer to this link for more details about API rate limit.
- redis: enter the server and database information to get access to redis database, if you opt to use redis to store the job cache
- batchy: enter the server and port information to connect to your scheduled batch jobs
- function_type: you can enter either "email" for daily email check or "batchy" for intraday job latency check or both, separated by comma
- heartbeat: enter the "Integration Key" of the startup service created on PagerDuty. It usually consists of 32 characters. Since we recommend users to restart the robopager application at regular intervals to avoid the unnecessary crash-down of the checks, heartbeat as a service can notify user that robopager has restarted successfully
- timezone: your local timezone to make sure the correct delivery time. You can use the code below to search the correct timezone:

```
import pytz
for tz in pytz.all_timezones:
    print tz
```

2.2 2. checklist.yaml

This file lists the information of all the jobs that you want to monitor. Each job must have a unique name. You could find examples below for both types of job.

Daily Email Checks

- type: email
- pd_description: a brief description of the check job
- pd_service: the "Integration Key" of a PagerDuty service only for this specific job, usually consists of 32 characters
- senders: list of senders that robopager will be monitoring for particular emails
- delivery_time: the expected delivery time of emails you are monitoring; emails received before this time will not be scanned. Please pay attention to your local timezone
- check_time: the time at which robopager actually starts checking for delivery. Even if an email is expected to arrive at 7:00AM we may wait untill 7:15 to account for minor latency and avoid false alarms. Please pay attention to pass the time that is in the same timezone where you run the jobs, such as your local computer or cloud server, etc.
- subjects: a list of email subjects requires to satisify the check

```
unique_email_check_job_name:
    type: email
    pd_description: "Check whether you are OK"
    pd_service: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
    senders: ['email_address@gmail.com']
    delivery_time: 03:00
    check_time: 03:15
    subjects:
        - "Great - You look awesome!"
        - "Good - You look OK!"
```

Intraday Latency Check

- type: batchy
- pd_description: a brief description of the check job
- pd_service: the "Integration Key" of a PagerDuty service only for this specific job, usually consists of 32 characters
- wf_name: name of the scheduled workflow you want to monitor, please enter the correct and accurate name
- check_time: the time at which robopager actually starts checking
- poll_sec: the number of seconds in between checks
- latency_min: when it has been more than x minutes from last end time a failure will be generated, warnings at 80% of threshold

```
unique_latency_check_name:
    type: batchy
    pd_description: "intraday latency check for xxxx job"
    pd_service: "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
    wf_name: "intraday_latency_check"
    check_time: "09:00"
    poll_sec: 180
    latency_min: 60
```

CHAPTER

THREE

QUICK START

After finish all the setups above, you can start to use the tool to help monitor you scheduled jobs. Robopager has two modes:

- Single Job Mode: This will run the specific check and exit when complete. Execute with a -c parameter and a specific job name you want to run. For example:

```
python3 -m robopager.robopager -c unique_email_check_job_name -y checklist_file_
↪path
-cf core -p config_file_path
```

- Service Mode: This will execute all the check jobs in the checklist.yaml file.

```
python3 -m robopager.robopager -y checklist_file_path -cf core -p config_file_path
```

Notice: This module uses datacoco_core's config() to parse the configuration file as default. We also integrate AWS Secret Manager as alternative configuration method to retriece your credentials but this function is still in progress. Please check back for update. The parameter "-cf" in command line is to choose whether using datacoco or secret manager

CHAPTER

FOUR

NOTES

- **Redis**: The main purpose of using Redis is to prevent Robopager from being annoying. Using Redis allows PagerDuty to store the incident history data in it. PagerDuty will check the state of the last run in Redis and only create a new incident if the same key of last run is not found (key is deterministic based on date + hour). This will avoid PagerDuty from sending the same alerts in every run during the check period. For each check, two key patterns of a new incident will be created:
  - *jobname*: stores latest state for a check (success or failure)
  - *alert key*: stores pagerduty submission informaton

  Below are a few helpful redis commands:
  - connect to redis cli, assuming database 1 will be used for robopager: `redis-cli -n 1`
  - list keys based on pattern: `keys *` or `keys key_name*`
  - get all fields within a hash (Robopager stores all keys as hashes): `hgetall key_full_name`
- Robopager submits to PagerDuty using a deterministic key, therefore you can have multiple Robopager instance running (for redundancy without producing duplicate tickets). There is a special offset_sec parameter in the PDInteraction class, setting this will have a specific server wait the specified number of seconds before checking state, and subsequently triggering incidents in Pagerduty. This will prevent unnecessary API calls.
- Robopager will not yet reload the checklist.yaml if changed, it will need to be restarted
- Finally, this is not a very sophisticated application (this simplicity is deliberate), we are using features like threading and some 0.x modules. So, we suggest rebooting or restarting the service often (weekly or daily) to avoid the unnecessary crash-down of the checks

CHAPTER

FIVE

DEVELOPMENT

5.1 Getting Started

It is recommended to use the steps below to set up a virtual environment for development:

```
python3 -m venv <virtual env name>
source <virtual env name>/bin/activate
pip install -r requirements.txt
```

5.2 Testing

```
pip install -r requirements-dev.txt
```

To run the testing suite, please modify the credentials in test_data folder, then simply run the command:

```
python3 -m unittest discover tests
```

CHAPTER

SIX

CONTRIBUTING

Contributions to Robopager are welcome! Please reference guidelines to help with setting up your development environment here