

# 非同期タスクキューを使って業務を自動化しまくった話

はんなりプログラミング:

1年の締めくくり！2023年にチャレンジしたことのLT祭

2023-12-15

佐野 浩士 @hrsano645

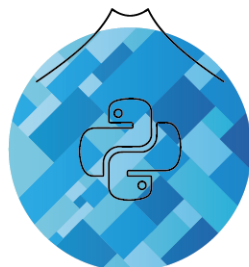
# お前誰よ / Self Introduction

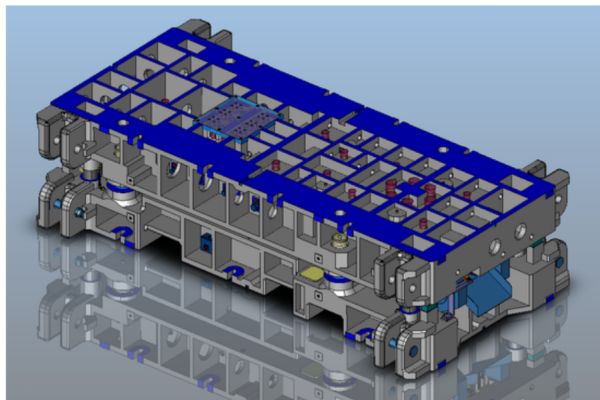
佐野 浩士 (Hiroshi Sano) [@hrs\\_sano645](#)

- 🌐: 静岡県富士市 🏔️
- 🏢: 株式会社佐野設計事務所 CEO
- 👤🤝
  - 🐍: PyCon mini Shizuoka Stuff / Shizuoka.py / Unagi.py / Python駿河
  - CivicTech, Startup Weekend Organizer
- Hobby: Camp 🏕️, DIY 🛠️, IoT 💡



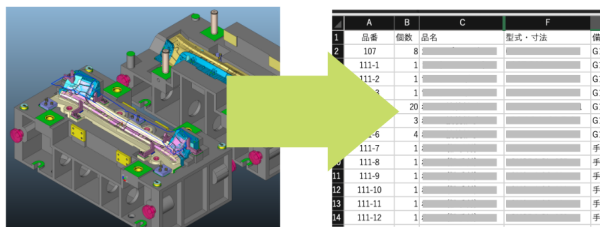
**PyCon**  
mini  
**SHIZUOKA**





## 3D機械設計 / 3Dモデリング

自動車向けプレス金型の3D設計  
金型設計関連の3Dモデリング  
製品データの3Dモデリング



	A	B	C	F	
1	品番	個数	品名	型式・寸法	備
2	107	8			G1
3	111-1	1			G1
4	111-2	1			G1
5		1			G1
6		20			G1
7		3			G1
8		4			G1
9	111-7	1			手
10	111-8	1			手
11	111-9	1			手
12	111-10	1			手
13	111-11	1			手
14	111-12	1			手



## 製造業DX支援

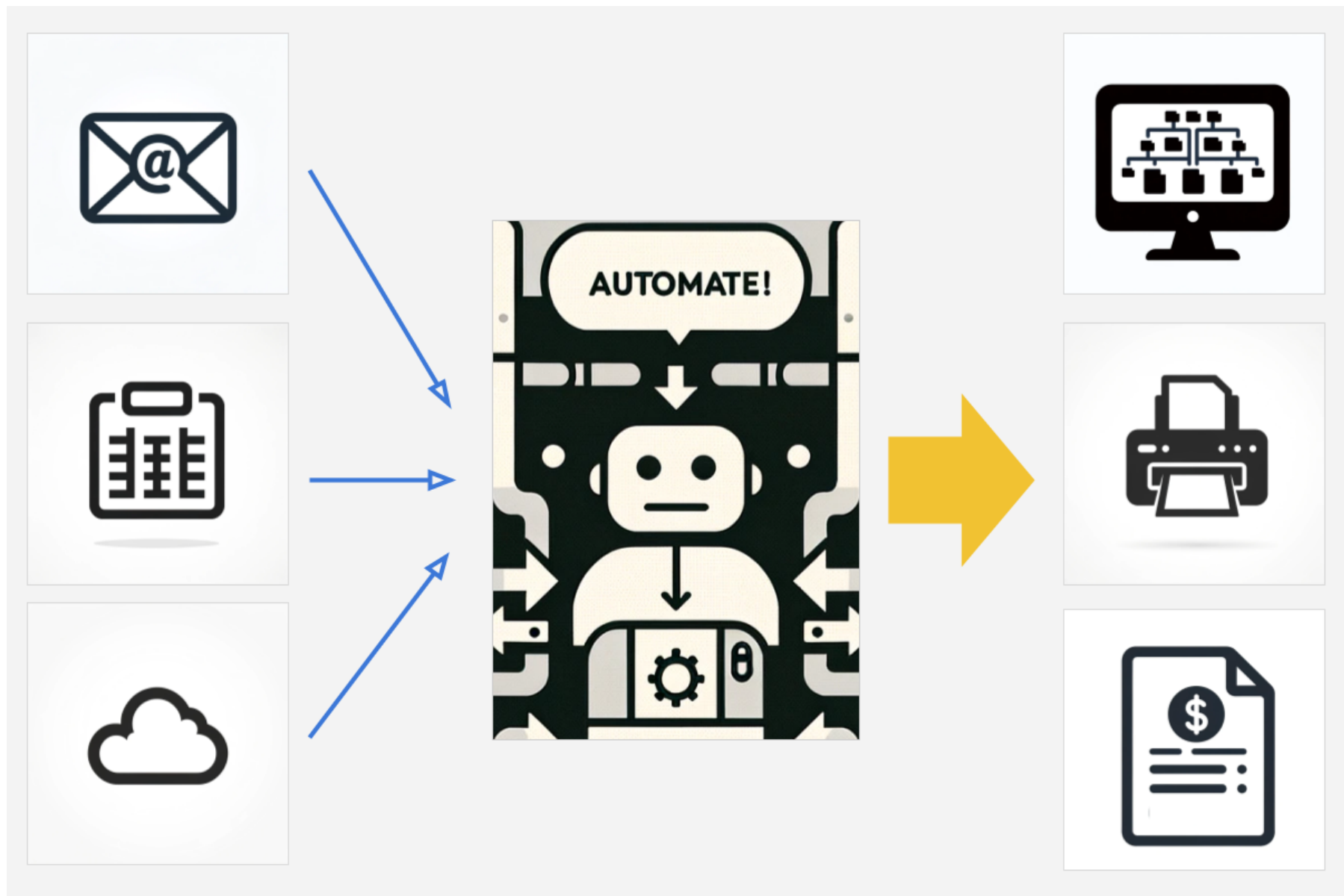
設計データをデジタル化  
製造業DX取り組みサポート

## 2023年の本業の話

**目標: 業務の効率化を限界まで進める**

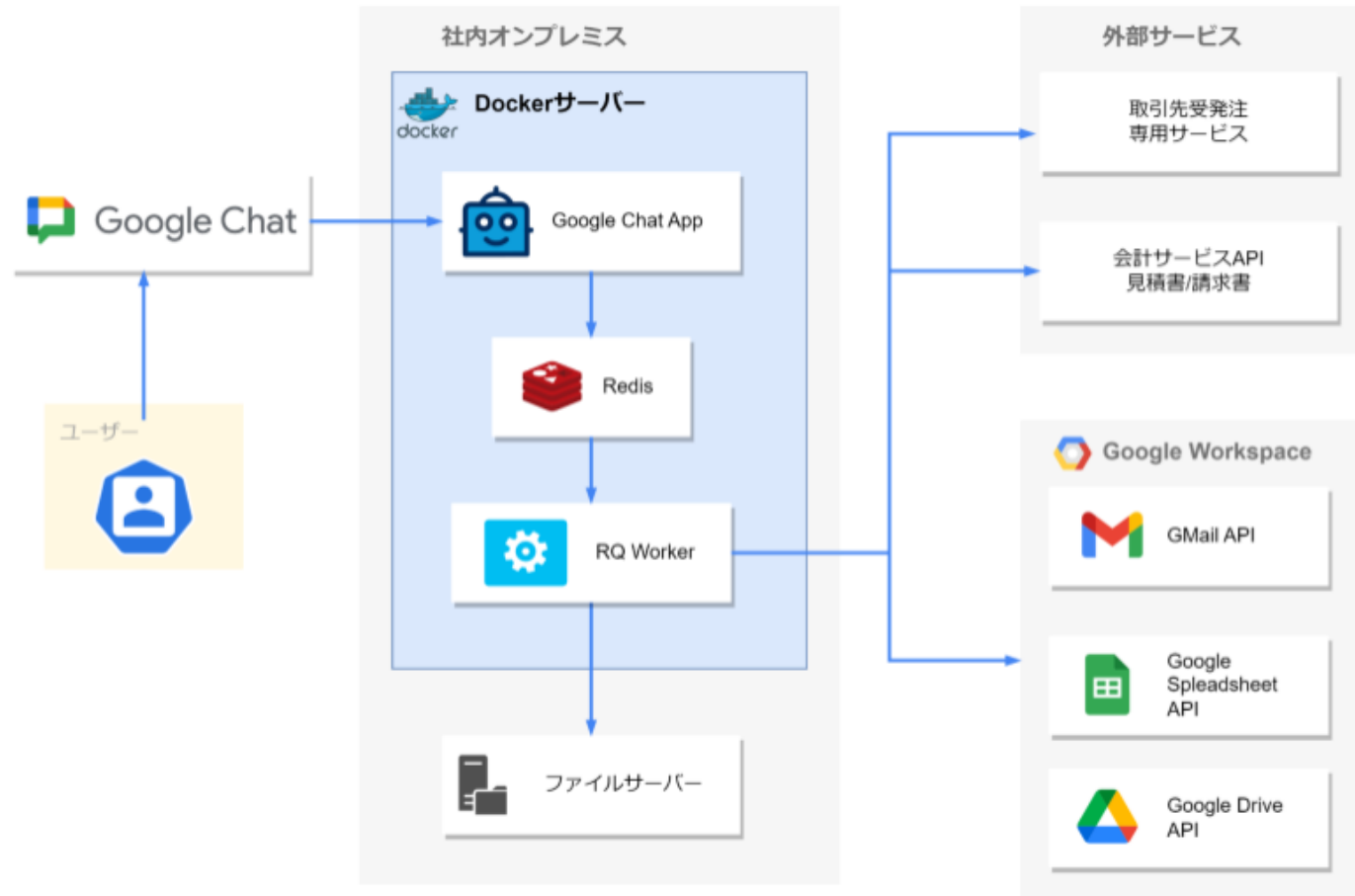
# 業務自動化ガッツリやりました

- 依頼ベースの案件業務
- 今まではそれほど多くなかったが今年になって急激に増える
  - 人力でやっていたのは追いつかなそうでやばい
- 人が必要な部分以外人力でやるのを止める！  
-> **止めることに成功した！！** 🙌😊



こんな絵を想像して

# 業務タスク自動化サービスの構成





# どんなことを効率化？

- 自動生成
  - 依頼受注（メール）→ボイラープレートツールで作業プロジェクトフォルダーを生成
  - スケジュール管理→Googleスプレッドシート連携
  - 会計サービスと連携して見積書/請求書生成（書類作成）
  - 依頼企業側のシステム連携: WEBスクレイピング
- タスク操作をChatOps
  - Google Chatでチャットボット作成
- 過去の依頼からサマリー情報のデータベース化: (現在取り組み中)

**自動生成の部分を非同期タスクキューを使って作業させています**

## なんで非同期にしたの？

- これらは重い処理: ファイル操作、APIアクセス -> I/Oバウンド処理
  - 組み合わせると**数秒ではなく数十秒～分単位**の処理
  - 結果が返ってくるタイミングはその時次第
- 同期処理でやると、処理が終わるまで待たされる  
-> ブロッキング処理
- **チャットボット側がロックされてしまう->応答が返せない**  
基本チャットボットは非同期前提

## Google Chatの場合

「同期的に応答するには、Chat アプリが 30 秒以内に応答し、その応答をインタラククションが発生したスペースに投稿する必要があります。それ以外の場合は、Chat アプリは非同期で応答できます。」

<https://developers.google.com/chat/api/guides/message-formats?hl=ja#sync-response>

(Slackの3秒よりも全然緩いけど、非同期前提な様子)

## 非同期とは

- 同期処理と非同期処理の違い:  
処理のオフロードと並列処理が可能。処理の待ち時間を有効活用できる
- チャットボットのために非同期処理を使うことになる: これが結局制約あるため
- ノンブロッキング処理: 処理が終わるまで待たされない（チャットの場合、応答が素早く返せる）

## Pythonでの非同期処理の選択肢

- 標準ライブラリ: (並列) threading, (並列) multiprocessing, (非同期) asyncio, (並列?) sub-interpreter (3.13以降)
- メッセージキュー活用: celery, rq, pyzmq(ZeroMQ)
- クラウドのメッセージング: Cloud Pub/Sub (イベントベースで)
- etc...

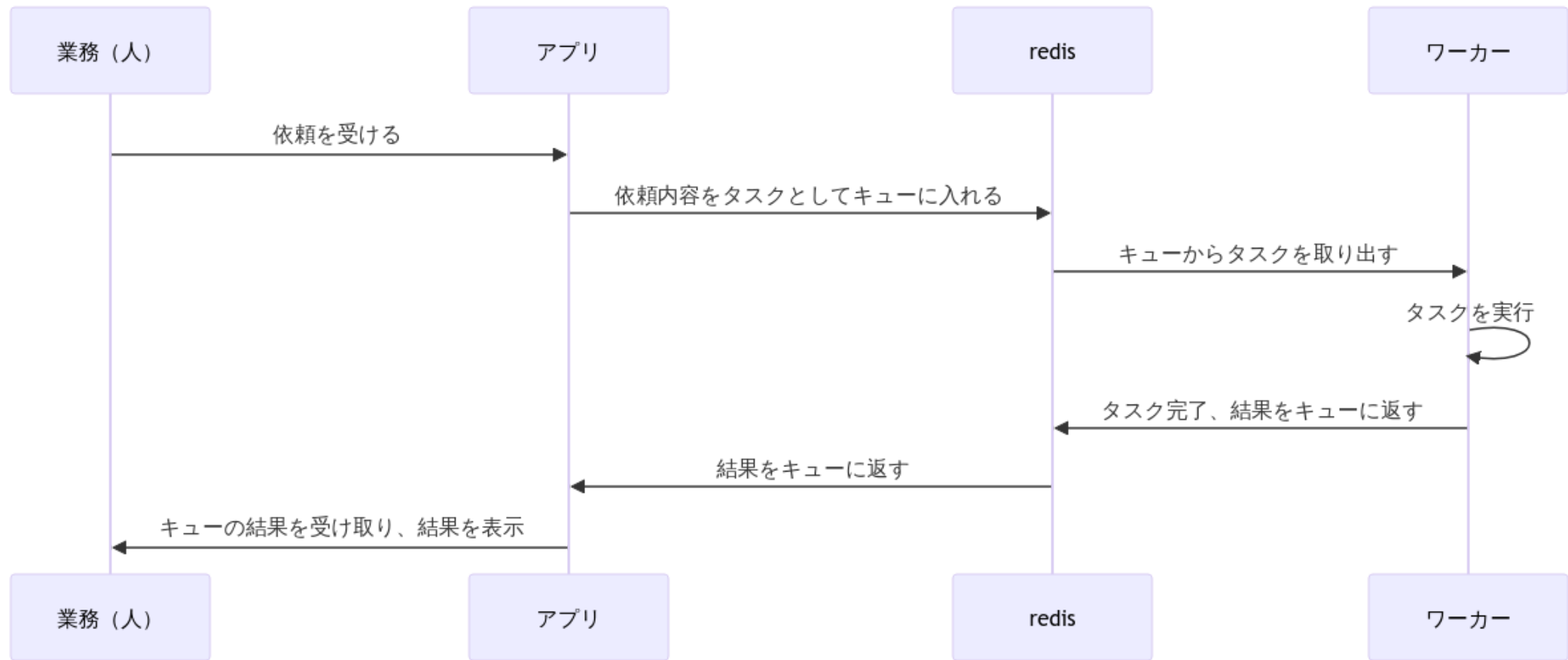
## 今回はRQ(python-rq)を使いました

python-rq: <https://python-rq.org/>

以下の3つの要素で構成される

- アプリ: タスク発行→キューへ入れる→ワーカーから処理結果を受け取る
- ワーカー: タスクの処理を行う
- redis: アプリとワーカーの間に入りキューとして利用

# 非同期タスクのざっくりイメージ





## なぜRQを選んだのか

ドキュメント見ていたら利用しやすいシンプルさが良かった

- asyncioと悩んだ -> RQがシンプルにできそうだった
- celeryと悩んだ -> celeryを使うほどの規模ではなかったと思う

※I/Oバウンス処理はasyncio, multiprocessingは制限にならないので、この選択肢がベストとは限らない（速度とか）

※redisの扱いに慣れたくて使いたかったという意味も 😊

## 注意

- RQはWindowsは非対応
  - WSLなら動かせる
    - > <https://python-rq.org/docs/#limitations>
- 今回はdockerで動かす例です
  - チャットボットも動かすのでどうせならLinux系がお手軽

ということで、ちょっぴやでDocckerで用意する場合の例

参考: [Python で分散タスクキュー \(RQ 編\) #Python - Qiita @hoto17296](#)

## Dockerfile

```
FROM python:3.11  
RUN pip install rq
```

# compose.yml

```
version: '3'
services:
  redis:
    image: redis
  worker:
    build: .
    depends_on:
      - redis
    environment:
      RQ_REDIS_URL: redis://redis
    command: rq worker
    volumes:
      - ./app
    working_dir: /app
  app:
    build: .
    depends_on:
      - redis
      - worker
    environment:
      RQ_REDIS_URL: redis://redis
    command: python app.py
    volumes:
      - ./app
    working_dir: /app
```

## 簡単なサンプル: printしてみる

tasks.py

```
import logging

logger = logging.getLogger(__name__)

def add(a, b):
    logger.debug("{} + {} = {}".format(a, b, a + b))
    return a + b
```

app.py

```
import os
from time import sleep
import redis
from rq import Queue
from tasks import add

q = Queue(connection=redis.from_url(os.environ.get("RQ_REDIS_URL")))

# 10個のタスクの実行をキューに投げる
tasks = [q.enqueue(add, args=(i, 1)) for i in range(10)]

# タスク実行が完了するまで少し待つ
sleep(1)

# 結果を出力する
print([task.result for task in tasks])
```

# 実行

```
# シングルワーカー  
$ docker-compose up  
  
# マルチワーカー: 4つのワーカーを起動  
$ docker-compose up --scale worker=4  
## ログは別途ファイルで見せます
```



## dockerで動かす時

- RQはredis（キュー）へタスクを渡すときはpickleを使ってる
  - ワーカー側でもpickleで渡されたオブジェクトが理解できないといけない  
-> ワーカー側にも同じライブラリをインストールする必要がある
- 手っ取り早い方法として
  - タスク側もワーカー側も同じ環境=Dockerfileを使う
  - コード参照や利用するボリュームも同じ箇所を参照すると楽
- タスクとワーカーを同時に動かすならcomposeが便利

## まとめ

- 膨大な退屈なこと手作業は間違えるので自動化しよう
- 自動化は重い処理をよく扱う->非同期前提で考える
- 非同期タスクキューを使うことで、重い処理を任せられ  
自動化の幅や連携方法が広がる（はず

Google Chatアプリの話はまたどこかで～

## 参考

- メッセージキュー - Wikipedia
- python-rq
- 【Pythonで高速化】I / Oバウンドとか並列処理とかマルチプロセスとかってなんぞや #Python - Qiita
- docker利用時の参考: Python で分散タスクキュー (RQ 編) #Python - Qiita

## サンプルコード

<https://github.com/hrsano645/exam-python-rq-by-docker>

## Google Chatと合わせる時

- チャットボット側で操作をする -> タスクをキューに入れる
- チャットボット側に応答をする
- ワーカー側でタスクを実行する
- ワーカー側でチャット側に非同期で応答を返す
  - Google ChatならGoogle Chat REST APIで非同期応答できる

