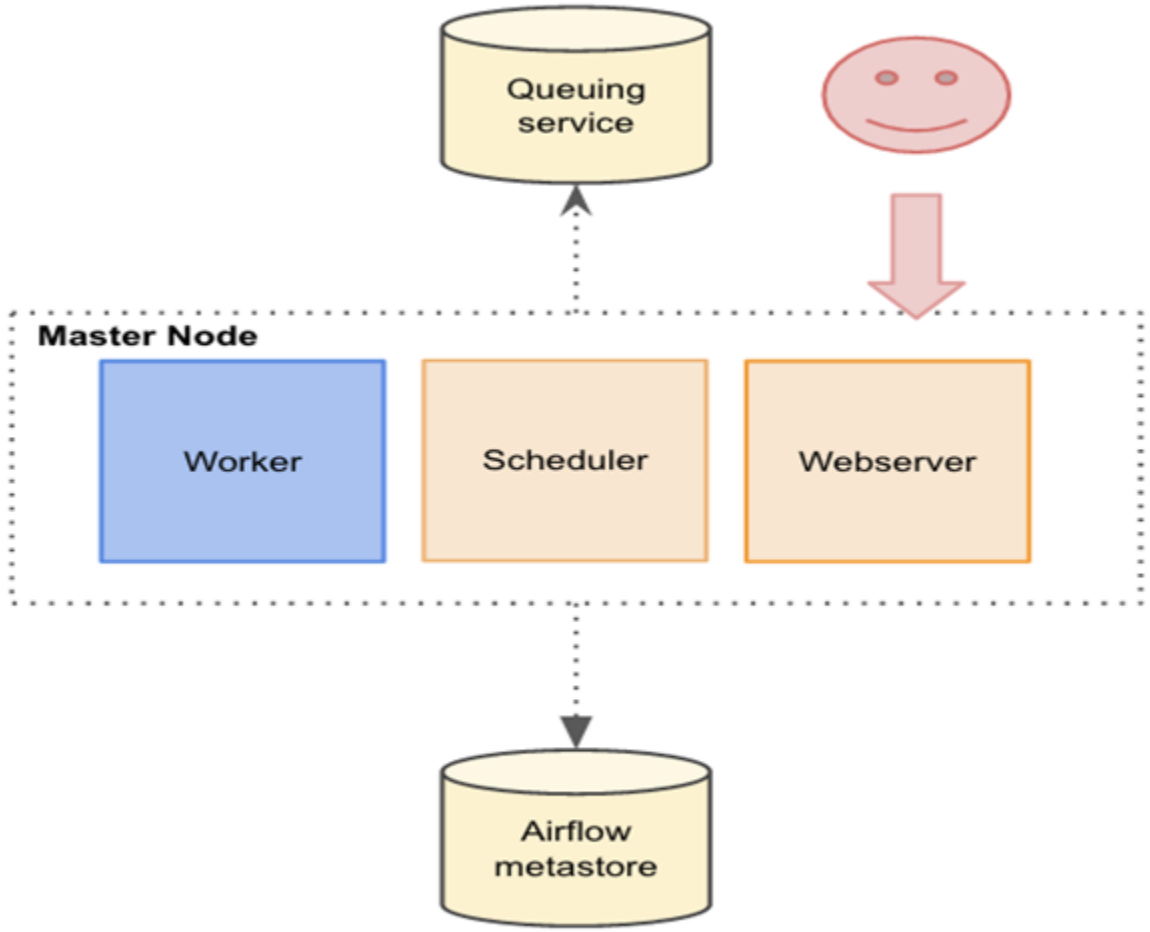
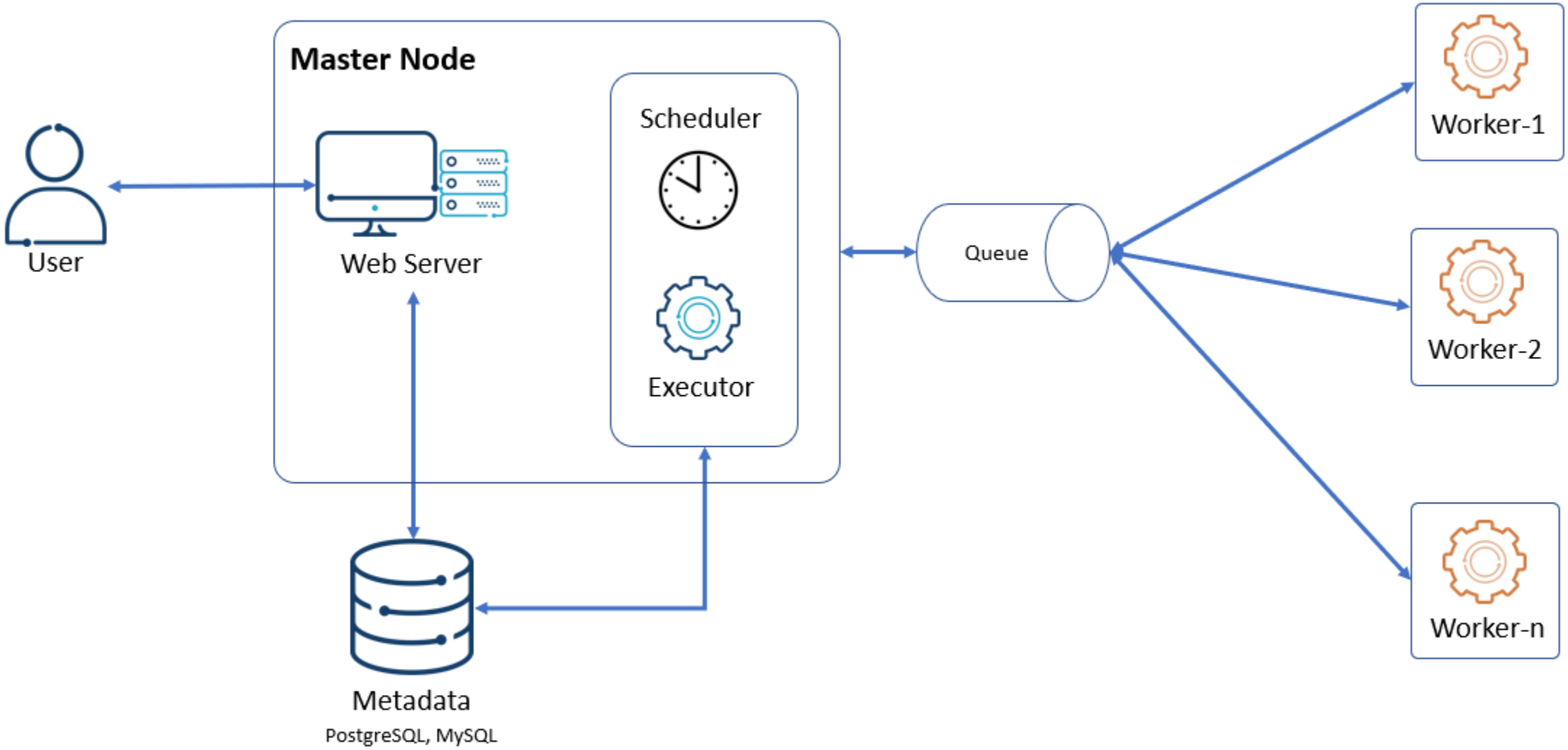
CELERY

WWW.ideo-lab.co - Guillaume Oneill
ideo-lab - January 2025

Celery Diagram Flow



Celery – Global Architecture



Installation of Redis Server

Redis steps to Install

- Update Ubuntu Package
- sudo apt-get Install redis-server
- sudo apt-get install python3-celery
- sudo apt-get install libmysqlclient-dev python3-dev
- pip install redis
- pip install –U « celery[redis] »
- Check Redis Server Running
  - Redis-cli
  - Ping
  - Expected result : pong

Linux system settings

```
sudo useradd celery -d /home/celery -b /bin/bash
```

steps to Django base settings

```
# CELERY SETTINGS

CELERY_BACKEND                = 'redis://localhost:6379/3’
CELERY_BROKER_URL             = 'redis://localhost:6379/4’
CELERY_RESULT_BACKEND = 'redis://localhost:6379/5’

 CELERY_TASK_SERIALIZER     = 'json’
CELERY_RESULT_SERIALIZER = 'json’
CELERY_ACCEPT_CONTENT    = ['json’]
CELERY_ENABLE_UTC              = True
CELERY_IMPORTS          = ['mink_kocliko_api.tasks', ]
```

steps to Django custom settings

```
CELERY_ENABLE                    =  False
CELERY_TASK_TRACK_STARTED        =  True
CELERY_TASK_TIME_LIMIT           =  30 * 60
CELERY_ENABLE_SIGNAL             =  False
SIGNAL_CALLER_INSPECT            =  10
COMMON_CELERY_CALLER             =  7
ALLOWED_CELERY_CALLER            =  ['ajax.py',  'main.py']
RESIDENCE_CELERY_CALLER          =  11
ACCOMODATION_CELERY_CALLER       =  10
```

steps to Local Linux settings

```
CELERY_ENABLE_SIGNAL  =  True
CELERY_ENABLE         =  True
```

Celery Django Modules

- Residence.py
  - createCeleryTask
- Celery.py
- Tasks.py
  - celeryTask
- Utils.py
  - storeCeleryTask
- Export_airtable.py
- __init__.py
- Settings.py
- Local_settings.py

Customisation of model Modules

```python
@receiver(signals.post_save, sender=Residence)
def createCeleryTask(sender, instance, created, **kwargs):
    """
    send  assynchrnous Task to Celery
    """
    from mink_kocliko_api.tasks import  celeryTask
    from lib.utils import  getCallingModule

    (process,  module) =  getCallingModule(depth = settings.COMMON_CELERY_CALLER)

    #  ----------------------------------------
    #  ----- DJANGO  SAVE OR CREATE          ---
    mode = "U"
    if created:
        mode =  "C"

    # ------------------------------------------------
    #  --- SIGNAL + CELERY TASK IF ENABLED      ---
    if settings.CELERY_ENABLE_SIGNAL:
        if settings.CELERY_ENABLE:
            residenceId =  instance.id
            celeryTask.delay(residenceId,  mode = mode, process = process,  module = module)
        else:
            celeryTask(residenceId,  mode = mode,  depth = settings.RESIDENCE_CELERY_CALLER)
```

Customisation of Celery.py Modules

- import os
- from celery import Celery

- os.environ.setdefault("DJANGO_SETTINGS_MODULE", « **project_api**.settings")

- app = Celery(« **project_api**")

- app.config_from_object("django.conf:settings", namespace="CELERY")

- app.autodiscover_tasks()

- @app.task(bind=True)
- def debug_task(self):
- print(f'Request: {self.request!r}')

Customisation of __init__ Module

- # django_celery/__init__.py
- from .celery import app as celery_app
- __all__ = ("celery_app",)

Customisation of tasks.py Module

```python
@shared_task
def celeryTask(record,  mode = "U",  model = "Residence", process = "",  module = "",  depth = 0):
    """
    store celery task request
    """
    from lib.utils import  storeCeleryTask
    from mink_kocliko_api.homepage.utils import caller_info
    from mink_kocliko_api.api.models.residence import  Residence
    from mink_kocliko_api.api.models.accomodation import  Accomodation,  AccomodationRoom


    dicModel =  {
        'Residence'         :  Residence,
        'Accomodation'      :  Accomodation,
        'AccomodationRoom' :  AccomodationRoom,
        }

    #  ------------------------------------------------------
    #  --- ACCEPTED MODEL FOR CELERY TASK AND AIRTABLE   ---
    #  ------------------------------------------------------
    if model in  dicModel:
        currentModel =  dicModel[model]
        rootRecord   =  currentModel.objects.filter(pk = record)
        if rootRecord:
            rootRecord =  rootRecord[0]
            storeCeleryTask(rootRecord,  mode = mode, model = model,  process = process,  module = module)
```

Flow Installation

- 1) Add Django signal receiver (Data Model)

```python
@receiver(signals.post_save, sender=Residence)
def createCeleryTask(sender, instance, created, **kwargs):
    """
    send  assynchrnous Task to Celery
    """
    from mink_kocliko_api.tasks import  celeryTask
    from lib.utils import  commonCeleryprocess

    commonCeleryprocess(instance, created)
```

- 2) Update celery settings.py
  - *Allowed Data Model*
  - *Allowed Python Callers*

```python
#  ------------------------------------------------
#  ---- INSPECT DEPTH FOR DJANGO SIGNAL        ---
#  ------------------------------------------------
CELERY_ALLOWED_ACCOMODATION =  {
        'ajaxApplyExcelImport'    :  ['Accomodation', True],
        'updateLocataireData'     :  ['Accomodation', True],
        'commonSignupProcess'     :  ['Accomodation', True],
        'ajaxApplyPleiadesImport' :  ['Accomodation', True],
        'updateSensorStatistics'  :  ['Accomodation', True],
        'getUserFromSerial'       :  ['Accomodation', True],
        'ajaxUpdateLogementData'  :  ['Accomodation', True]
        }
```

```python
CELERY_ALLOWED_COMMON  =  {
        'Accomodation'           :  [CELERY_ALLOWED_ACCOMODATION, 2],
        'Residence'              :  [CELERY_ALLOWED_RESIDENCE,    2],
        'ImportUser'             :  [CELERY_ALLOWED_KIZEO,        2],
        'CounterCurrentStatus'   :  [CELERY_ALLOWED_COUNTER,      2],
        'SensorCurrentStatus'    :  [CELERY_ALLOWED_SENSOR,       2],

        }
```

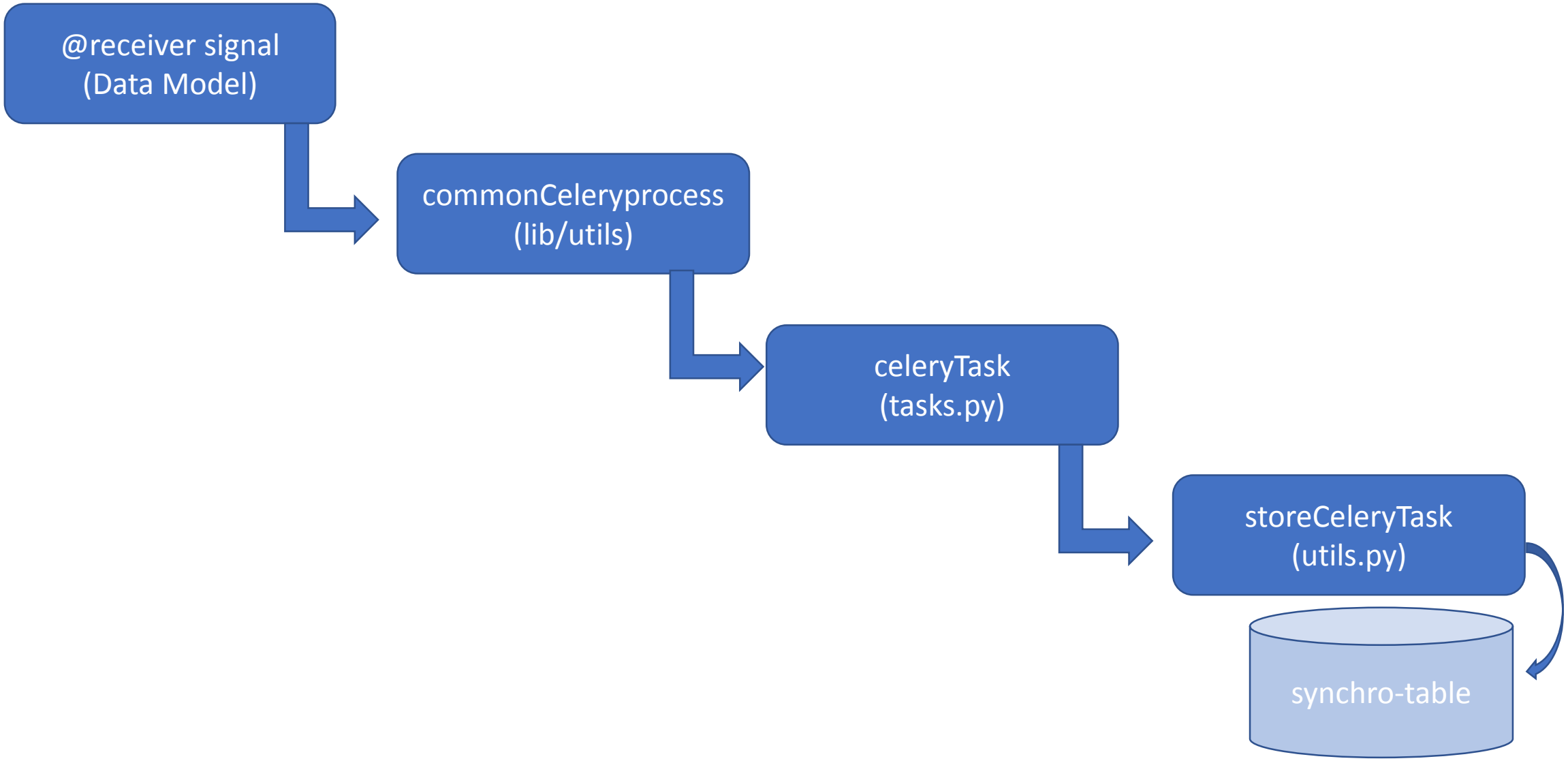
- *3) Update airtableSynchro Model :*
  - *Add Allowed Model to be processed by Django signal/Export Cron*

```python
class airtableSynchro(TimestampedModel):
    """
    Log updates to AirTable
    https://docs.djangoproject.com/fr/4.1/ref/contrib/contenttypes/#methods-on-contenttype-instances
    https://micropyramid.medium.com/understanding-genericforeignkey-in-django-c25ec5471d5e
    """

    from django.contrib.contenttypes.models import ContentType
    from django.contrib.contenttypes.fields import GenericForeignKey
    from django.utils import timezone

    MODE = (
            ('RESIDENCE',      'Residence')      ,
            ('ACCOMODATION',   'Accomodation')   ,
            ('ROOM',           'AccomodationRoom') ,
            ('KIZEO',          'User')           ,
        )
```

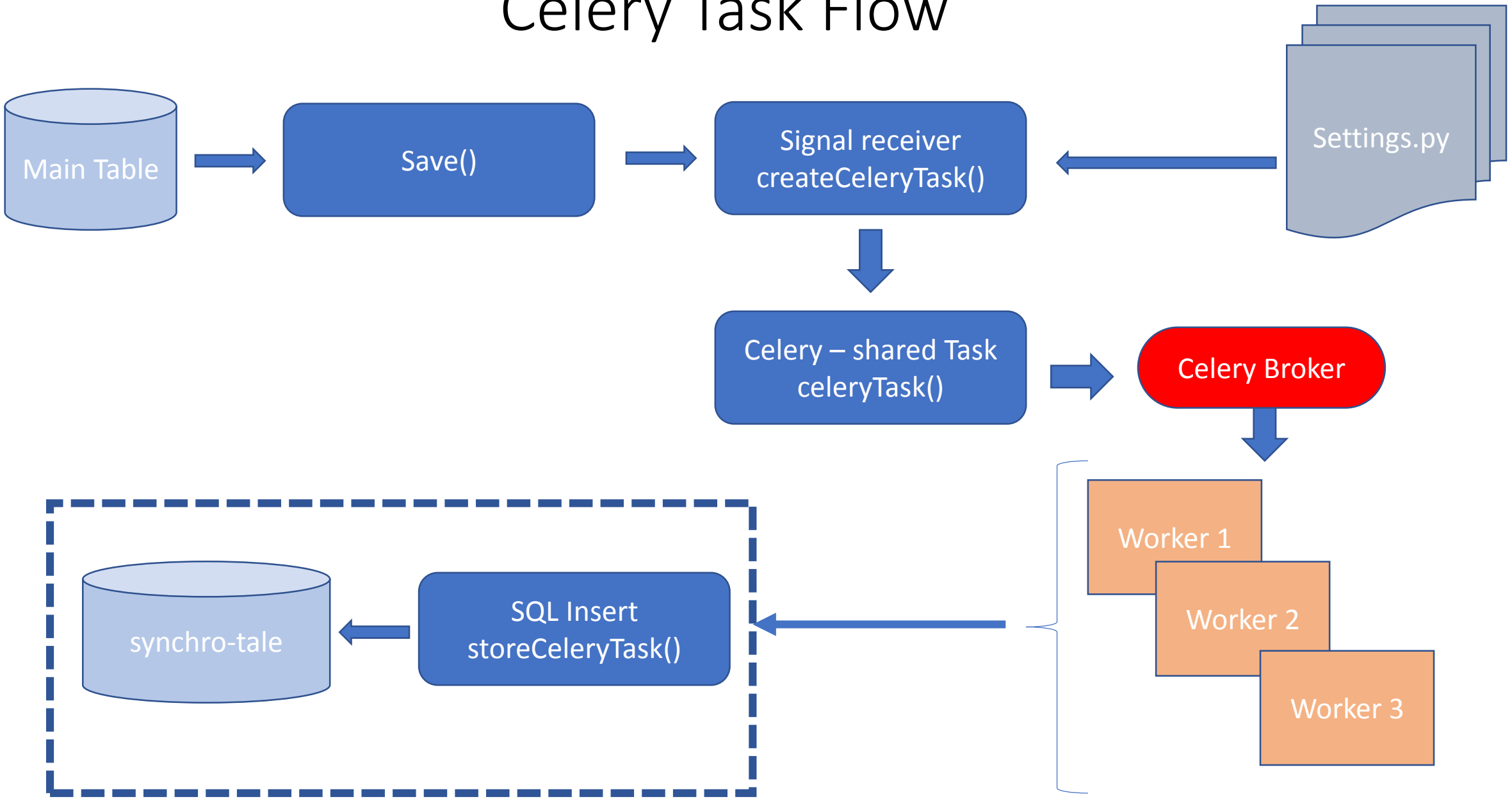
Celery Settings

```
CELERY_ENABLE_MODEL        = {
        'Residence'            :  True,
        'Accomodation'         :  True,
        'AccomodationRoom'     :  False,
        'ImportUser'           :  True,
        'CounterCurrentStatus' :  True,
        'SensorCurrentStatus'  :  True,
        }


SIGNAL_CALLER_INSPECT      =  10       #  ---- INSPECT DEPTH FOR SIGNAL/CELERY CALL ORIGIN  ---
COMMON_CELERY_CALLER       =  8        #  ---- caller depth to catch caller of save()       ---
ALLOWED_CELERY_CALLER      =  ['ajax.py',  'main.py']  #  ---- to be checked --
RESIDENCE_CELERY_CALLER    =  12        #  ---- caller depth to catch caller of save()       ---
ACCOMODATION_CELERY_CALLER =  7         #  ---- caller depth to catch caller of save()       ---

#  ------------------------------------------------
#  ---- INSPECT DEPTH FOR DJANGO SIGNAL        ---
#  ------------------------------------------------
CELERY_ALLOWED_ACCOMODATION =  {
        'ajaxApplyExcelImport'    :  ['Accomodation', True],
        'updateLocataireData'     :  ['Accomodation', True],
        'commonSignupProcess'     :  ['Accomodation', True],
        'ajaxApplyPleiadesImport' :  ['Accomodation', True],
        'updateSensorStatistics'  :  ['Accomodation', True],
        'getUserFromSerial'       :  ['Accomodation', True],
        'ajaxUpdateLogementData'  :  ['Accomodation', True]
        }
CELERY_ALLOWED_RESIDENCE =  {
        'process_residence'              :  ['Residence', False],
        'ajaxUpdateResidenceAutreDonnee' :  ['Residence', True],
        'ajaxUpdateResidenceComptage'    :  ['Residence', True],
        'ajaxUpdateResidenceData'        :  ['Residence', True],
        'ajaxUpdateResidenceGeneralInfo' :  ['Residence', True],
        }
CELERY_ALLOWED_KIZEO =  {
        'ajaxApplyairTableKizeoExport'   :  ['ImportUser', True],
        }
CELERY_ALLOWED_SENSOR =  {
        'ajaxApplyairTableKizeoExport'   :  ['SensorCurrentStatus', True],   #  --- check allowed Modules --
        }
CELERY_ALLOWED_COUNTER =  {
        'ajaxApplyairTableKizeoExport'   :  ['CounterCurrentStatus', True],   #  --- check allowed Modules --
        }
CELERY_ALLOWED_COMMON  =  {
        'Accomodation'           :  [CELERY_ALLOWED_ACCOMODATION, 2],
        'Residence'              :  [CELERY_ALLOWED_RESIDENCE,    2],
        'ImportUser'             :  [CELERY_ALLOWED_KIZEO,        2],
        'CounterCurrentStatus'   :  [CELERY_ALLOWED_COUNTER,      2],
        'SensorCurrentStatus'    :  [CELERY_ALLOWED_SENSOR,       2],


        }
```

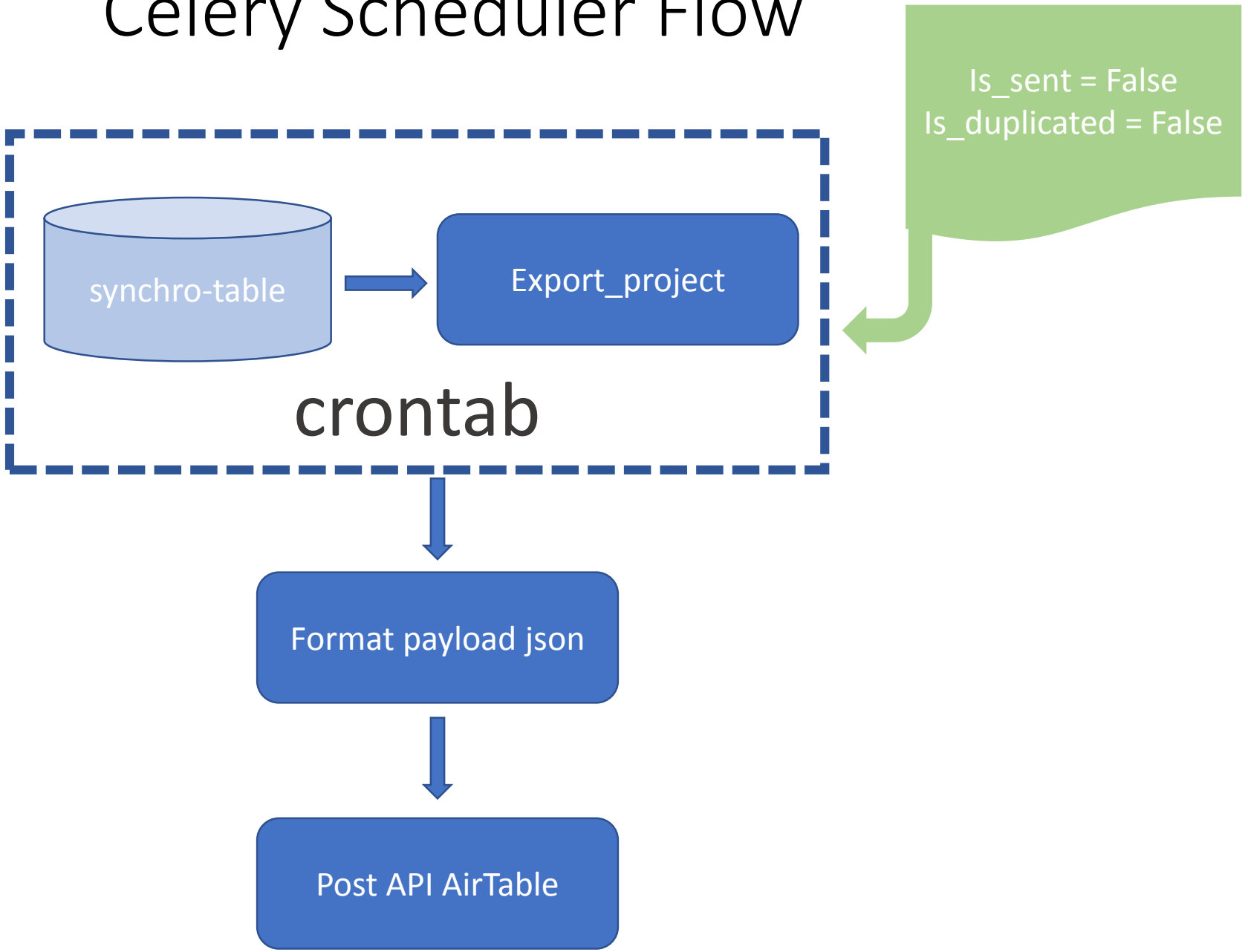
General Signal Flow



Celery Task Flow



Celery Scheduler Flow



Django Celery Ideo-Lab Architecture

- Data Model customized Save()
- Django Signal receiver
- Celery Tasks :
  - Store in SQl Celery project queing Table
- Cron process : export_task
  - API Post request to project

Celery – Redis architecture

- 3 services Should run at the same time :

  - **Producer** : Ideo-lab application (project_api)
  - **Message Broker** : The Redis server
  - **Consumer** : Our Celery app

  - Python manage.py runserver (or Gunicorn/Nginx proxy)
  - Redis-server (started by Linux Kernel at Load)
  - Python –m celery –A project_api worker –l INFO

Celer.py

```python
from __future__ import absolute_import, unicode_literals

import os
import dotenv

from celery import Celery

# Load .env variables
dotenv.read_dotenv()

# set the default Django settings module for the 'celery' program.
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'projectile.settings')

app = Celery('projectile')

# Using a string here means the worker doesn't have to serialize
# the configuration object to child processes.
# - namespace='CELERY' means all celery-related configuration keys
#   should have a `CELERY_` prefix.
app.config_from_object('django.conf:settings', namespace='CELERY')

# Load task modules from all registered Django app configs.
app.autodiscover_tasks()

@app.task(bind=True)
def debug_task(self):
    print('Request: {0!r}'.format(self.request))
```

__init__.py

```python
from __future__ import absolute_import, unicode_literals

# This will make sure the app is always imported when
# Django starts so that shared_task will use this app.
from .celery import app as celery_app

__all__ = ('celery_app',)
```

Start Celery Server

Python –m celery –A ideolab_api worker –l INFO &

or

supervisorctl restart api_celery

Celery Documentations