Passerelle - Development #12469

infra générale pour l'exécution de jobs asynchrones

07 juillet 2016 08:18 - Frédéric Péters

Statut:	Fermé	**Début:**	07 juillet 2016
Priorité:	Normal	**Echéance:**	
Assigné à:		**% réalisé:**	0%
Catégorie:		**Temps estimé:**	0:00 heure
Version cible:			
Patch proposed:	Oui	**Planning:**	

Description

À se poser un peu hier sur le taf de mise à jour des rues depuis la base adresse, je me dis ce matin que Passerelle gagnerait à avoir en standard de quoi permettre des jobs asynchrones.

J'imagine un système qui ressemblerait à ça :

```
# passerelle/base/models.py

class Job(models.Model):
    start_timestamp = models.DateTimeField()
    end_timestamp = models.DateTimeField()
    app_name = models.CharField()
    kind = models.CharField()
    options = JSONField()

    @classmethod
    def add_job(self, ...): # ...

[...]

# passerelle/apps/myconnector/models.py

class MyConnector(...):
    [...]
    def whatever(self):
        Job.add_job("update-streets", ...)

    def runjob(self, kind, options):
        [...]

# passerelle/base/management/commands/runjobs.py

for job in Job.objects.select(start_time__isnull=True):
    connector = get_app(job.app_name).get_connector_model()
    connector.runjob(...)
```

Commentaires sur l'idée ?

Révisions associées

Révision 52574718 - 20 février 2019 11:31 - Frédéric Péters

general: add basic asynchronous job infrastructure (#12469)

Historique

#1 - 13 février 2019 12:29 - Frédéric Péters

- Fichier 0001-general-add-basic-asynchronous-job-infrastructure-12.patch ajouté

- Statut changé de Nouveau à Solution proposée

- Patch proposed changé de Non à Oui

J'avais besoin d'un truc de ce type pour cart@ds (pour ne pas bloquer un appel sur le transfert par FTP d'un zip volumineux) et du coup j'ai fait ça puis j'ai retrouvé ce ticket et ça correspond plutôt bien.

Rien n'empêche à un moment de remplacer l'exécution via cron par quelque chose de plus sophistiqué mais là ça fait le job (pun intended). Plus tard aussi des possibilités pour visualiser dans l'UI les jobs, etc.

#2 - 13 février 2019 13:50 - Frédéric Péters

- Fichier 0001-general-add-basic-asynchronous-job-infrastructure-12.patch ajouté

Avec la migration.

#3 - 13 février 2019 13:51 - Frédéric Péters

- Fichier 0001-general-add-basic-asynchronous-job-infrastructure-12.patch ajouté

Et les tests...

#4 - 13 février 2019 14:35 - Benjamin Dauvergne

Pour la même chose coté zoo je prends la peine de locker les jobs, si on peut se permettre de n'être compatible que Django 1.11 désormais on peut faire :

```
while True:
    with atomic():
        # lock a job
        job = Job.objects.filter(..., status='registered').select_for_update(skip_locked=True)[:1].first()
        if not job:
            break
        job.status = 'running'
        job.save()
        # release lock
    try:
        getattr(self, job.method_name)(**job.parameters)
        job.status = 'completed'
    except SkipJob:
        job.status = 'registered'
    except Exception as e:
        (exc_type, exc_value, tb) = sys.exc_info()
        job.status = 'failed'
        job.done_timestamp = timezone.now()
        job.status_details = {
            'error_summary': '\n'.join(traceback.format_exception_only(exc_type, exc_value)).strip(),
        }
    job.save()
```

Si on veut être compatible Django 1.8, on ne peut utiliser que nowait=True et pas skip_locked=Trye et donc il faut boucler et récupérer DatabaseError.

Aussi ça peut être intéressant d'avoir un identifiant sur un job pour l'associer à un truc qu'on voudrait retrouver plus tard, genre pour remonter au workflow qu'un job à foirer, le relancer ou que sais-je (un simple job_id = models.CharField(max_length=256, blank=True, null=True)).

À part ça c'est ack pour moi.

#5 - 18 février 2019 13:23 - Frédéric Péters

- Fichier 0001-general-add-basic-asynchronous-job-infrastructure-12.patch ajouté

La version dans la branche, qui implémente ce lock en 1.11, et un lock de toutes les lignes en 1.8.

```
+        skip_locked = {'skip_locked': True}
+        if django.VERSION < (1, 11, 0):
+            skip_locked = {}
```

et **skip_locked passé à select_for_update().

C'est pas super joli mais ça permet de ne pas s'imposer la construction plus lourde suggérée pour 1.8.

Par rapport au commentaire, il y a aussi la prise en compte du fait qu'un job peut être sauté et qu'il doit alors être ignoré pour le reste de la boucle.

#6 - 20 février 2019 10:44 - Emmanuel Cazenave

- Statut changé de Solution proposée à Solution validée

#7 - 20 février 2019 11:34 - Frédéric Péters

- Statut changé de Solution validée à Résolu (à déployer)

Rebasé avec renumérotation migration suite à #29965.

```
commit 5257471818eedc214dd16dd9bc56f444dfeecae9
Author: Frédéric Péters <fpeters@entrouvert.com>
Date:   Wed Feb 13 12:24:06 2019 +0100

    general: add basic asynchronous job infrastructure (#12469)
```

#8 - 20 février 2019 15:16 - Frédéric Péters

- Statut changé de Résolu (à déployer) à Solution déployée

#9 - 12 avril 2019 10:27 - Benjamin Dauvergne

- Statut changé de Solution déployée à Fermé

Fichiers

0001-general-add-basic-asynchronous-job-infrastructure-12.patch	4,51 ko	13 février 2019	Frédéric Péters
0001-general-add-basic-asynchronous-job-infrastructure-12.patch	6,31 ko	13 février 2019	Frédéric Péters
0001-general-add-basic-asynchronous-job-infrastructure-12.patch	7,75 ko	13 février 2019	Frédéric Péters
0001-general-add-basic-asynchronous-job-infrastructure-12.patch	8,52 ko	18 février 2019	Frédéric Péters