

Proyecto Reportes

Silva Serna José del Refugio

5 de junio de 2018

Índice

1. Resumen	2
2. Palabras clave	2
3. Marco teórico	2
4. Introducción	2
5. Planteamiento del problema	2
6. Desarrollo	2
7. Implementación	3
8. Conclusión	3
9. Referencias	3

1. Resumen

Se describe como se llevó a cabo un proyecto con ayuda de celery, apoyados por la estructura de django y con redis el cual se corrió en otra máquina de la misma red, con la finalidad de poder enviar y recibir información de un programa a redis estos en máquinas distintas.

2. Palabras clave

Celery, Redis, Django.

3. Marco teórico

Celery: Es una aplicación que nos permite crear tareas de trabajo asíncronas gestionadas por un gestor de colas que está basada en el envío de mensajes de manera distribuida. Se focaliza en operaciones en tiempo real pero también soporta la calendarización de tareas, es decir, puede lanzar tareas que se tengan que ejecutar en un momento determinado o de manera periódica. Las unidades de ejecución, llamadas tareas, se ejecutan de manera concurrente en uno o más nodos de trabajo. Estas tareas pueden ejecutarse de manera asíncrona bien de manera síncrona (esperando hasta que la tarea está lista).

Redis: es un motor de base de datos en memoria, basado en el almacenamiento en tablas de hashes (clave/valor) pero que opcionalmente puede ser usada como una base de datos durable o persistente. Está escrito en ANSI C por Salvatore Sanfilippo quien fue patrocinado por VMware

Django: Django es un framework web de alto nivel que permite el desarrollo de sitios web rápido, seguro y mantenible. Desarrollado por programadores experimentados, Django se encarga de gran parte de las complicaciones del desarrollo web, por lo que puedes concentrarte en escribir tu aplicación sin necesidad de reinventar la rueda.

4. Introducción

Se desarrolló un proyecto con la ayuda de celery, django y redis, y con atom como editor, el proyecto maneja reportes los cuales cada 5 y 10 segundos se irán generando, en este documento se verá el cómo se realizó la conexión del cliente con el servidor mostrando la configuración y los pasos a seguir para que redis se pueda correr en otra máquina conectada a la misma red como servidor.

5. Planteamiento del problema

En la actualidad se generan y se imprimen una gran cantidad de reportes con el cual se puede transmitir mucha información en el día a día, los reportes son informes que organizan y exhiben la información contenida en una base de datos. Su función es aplicar un formato determinado a los datos para mostrarlos por medio de un diseño atractivo y que sea fácil de interpretar por los usuarios. Por lo cual es importante que se tenga un buen manejo de expedición para

que la gente que lo necesita lo tenga al alcance de su mano y haga un buen uso con la información que estos contienen.

6. Desarrollo

Utilizando un ambiente virtual se creó el proyecto llamado Reporte y una aplicación llamada reportes, en el cual cada 5 segundos realizara una tarea, la cual la tarea es en una lista de reportes con los campos report id, created at, user, colored status, async task status los cuales se declaran en el modelo.

```
1 # -*- coding: utf-8 -*-
2 from __future__ import unicode_literals
3
4 from django.db import models
5
6 # Create your models here.
7 class ReporteItem(models.Model):
8     report_id = models.CharField(max_length=120, null=True, blank = True)
9     created_at = models.CharField(max_length=120, null=True, blank = True)
10    user = models.CharField(max_length=120, null=True, blank = True)
11    colored_status = models.CharField(max_length=120, null=True, blank = True)
12    async_task_status = models.CharField(max_length=120, null=True, blank = True)
13    mensaje = models.CharField(max_length=120, null=True, blank = True)
14
15    def __str__(self):
16        return str(self.mensaje)
```

Model

En el modelo se crea como ya se explicó los valores en una clase llamada ReporteItem

```
from __future__ import absolute_import, unicode_literals
import random
from celery.decorators import task
from models import ReporteItem
from datetime import datetime
import os
import sys

def send(report_id, created_at, user, colored_status, async_task_status):
    list_mails = ['sent']
    ([report_id: 'Solicitado por Oficina', 'created_at: datetime.now().strftime("%d %b %Y %H:%M:%S %Z")', 'user: "Administrador"',
    [report_id: 'Solicitado por Almacén', 'created_at: datetime.now().strftime("%d %b %Y %H:%M:%S %Z")', 'user: "Administrador"',
    ])
    [random.randint(0,1)
    report_id=list_mails[0], sent=[report_id]
    created_at=list_mails[1], sent=[report_id]
    user=list_mails[2], sent=[report_id]
    colored_status=list_mails[3], sent=[report_id]
    async_task_status=list_mails[4], sent=[report_id]

    sent_report_id = report_id
    sent_created_at = created_at
    sent_user = user
    sent_colored_status = colored_status
    sent_async_task_status = async_task_status
    mensaje = report_id, created_at, user, colored_status, async_task_status
    new_obj = ReporteItem.objects.create(report_id=sent_report_id, created_at=sent_created_at, user=sent_user, colored_status=sent_colored_status, async_task_status=sent_async_task_status, mensaje=mensaje)
```

Task

Como tarea en el task se crea una lista y con un número random que tenga el mismo tamaño que la lista, para guardar los valores de la lista en variables con ayuda del random que nos estará dando los elementos de la lista aleatoriamente para y con esto poder retornar las como un mensaje.

```
1 from __future__ import absolute_import, unicode_literals
2 import os
3 from celery import Celery
4 from celery.schedules import crontab
5
6 os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'Reportes.settings')
7 app = Celery('proj')
8 app.config_from_object('django.conf:settings', namespace='CELERY')
9 app.autodiscover_tasks()
10
11
12 app.conf.beat_schedule = {
13     'add-every-5-seconds': {
14         'task': 'mandar_reporte',
15         'schedule': 5.0,
16         'args': (16, 16, 16, 16, 16),
17     },
18     'add-every-10-seconds': {
19         'task': 'mandar_reporte',
20         'schedule': 10.0,
21         'args': (16, 16, 16, 16, 16),
22     },
23 }
24
25 @app.task(bind=True)
26 def debug_task(self):
27     print('Request: {0!r}'.format(self.request))
```

Celery

Para poder correr la tarea que se creó en task en un determinado tiempo tenemos que llamarlas en celery gracias a que importamos el módulo de celery podemos usar app.conf.beat_schedule que este nos permite asignar el tiempo

po a las tareas con parámetros.

```
CONTRIBUTING INSTALL README.md runtest-cluster src
jose@jose-ThinkPad-X230: ~/Escritorio/redis-stable/src
jose@jose-ThinkPad-X230:~/Escritorio/redis-stable/src$ cd src
jose@jose-ThinkPad-X230:~/Escritorio/redis-stable/src$ redis-server
10571:C 04 Jun 19:57:43.587 # Warning: no config file specified, using the default
10571:M 04 Jun 19:57:43.588 * Increased maximum number of open files to 10032 (it was originally set to 1024).

Redis 3.0.6 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 10571
http://redis.io

10571:M 04 Jun 19:57:43.589 # WARNING: The TCP backlog setting of 511 cannot be
10571:M 04 Jun 19:57:43.589 # Server started, Redis version 3.0.6
10571:M 04 Jun 19:57:43.589 # WARNING overcommit_memory is set to 0! Background
10571:M 04 Jun 19:57:43.589 # save may fail under low memory condition. To fix this issue add 'vm.overcommit_m
10571:M 04 Jun 19:57:43.589 # memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.ove
10571:M 04 Jun 19:57:43.589 # rcommit_memory=1' for this to take effect.
10571:M 04 Jun 19:57:43.589 # WARNING you have Transparent Huge Pages (THP) support
10571:M 04 Jun 19:57:43.589 # enabled in your kernel. This will create latency and memory usage issues with
10571:M 04 Jun 19:57:43.589 # Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent
10571:M 04 Jun 19:57:43.589 # hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the
10571:M 04 Jun 19:57:43.589 # setting after a reboot. Redis must be restarted after THP is disabled.
10571:M 04 Jun 19:57:43.599 * DB loaded from disk: 0.010 seconds
10571:M 04 Jun 19:57:43.599 * The server is now ready to accept connections on port 6379
```

Redis-server

Ahora para poder realizar la conexión e ingresar a redis como servidor mediante la terminal entrando a la src en la carpeta redis e ingresar el comando redis-server con el cual se correrá redis como servidor.

Después el usuario que se conectará en una máquina externa teniendo en cuenta que se tiene que conectar a la misma red que en la que se encuentra el servidor, el usuario que ingresa como cliente necesariamente tiene que modificar el archivo local en la parte donde se tiene la dirección de redis, localhost se cambia a la ip del servidor.

```
124 STATIC_URL = '/static/'
125 CELERY_BROKER_URL='redis://172.16.1.130:6379'
126 CELERY_RESULT_CONTENT='redis://172.16.1.130:6379'
127 CELERY_ACCEPT_CONTENT=['application/json']
128 CELERY_RESULT_SERIALIZER='json'
129 CELERY_TIMEZONE=TIME_ZONE
130
```

Local

Después el cliente necesita correr el worker para que este le mande permisos al servidor, y después el servidor ingresa como cliente y da los permisos al cliente con el comando CONFIG SET protected-mode no, lo que hará que no esté protegido y el cliente pueda ingresar, teniendo en cuenta que solo 1 cliente se debe de conectar ya que tendrán problemas por conectarse al mismo tiempo.

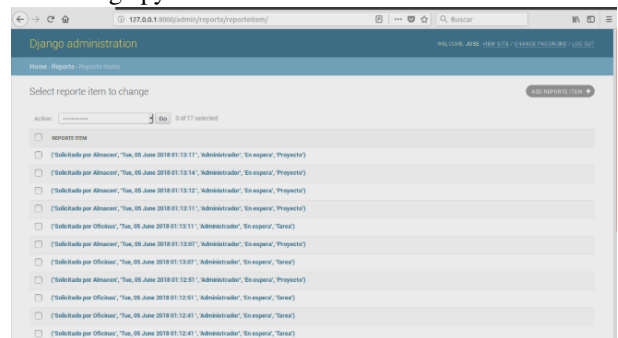
```
jose@jose-ThinkPad-X230: ~/Documentos/Proyecto/src
- ** -----> transport: redis://localhost:6379//
- ** -----&; results: disabled//
- ** -----> concurrency: 4 (prefork)
- ** -----> task events: OFF (enable -E to monitor tasks in this worker)
- ** -----
[queues]
-> celery exchange=celery(direct) key=celery

[tasks]
- Reportes.celery.debug_task
- mandar_reporte

[2018-06-05 01:46:02.733: INFO/MainProcess] Connected to redis://localhost:6379/
[2018-06-05 01:46:02.743: INFO/MainProcess] mingle: searching for neighbors
[2018-06-05 01:46:03.761: INFO/MainProcess] mingle: all alone
[2018-06-05 01:46:03.811: WARNING/MainProcess] /home/jose/Documentos/Proyecto/ll
b/python3.5/site-packages/celery/fixups/django.py:200: UserWarning: Using settin
gs-DEBUG leads to a memory leak, never use this setting in production environmen
ts!
warnings.warn('Using settings.DEBUG leads to a memory leak, never '
[2018-06-05 01:46:03.812: INFO/MainProcess] celery@jose-ThinkPad-X230 ready.
```

Worker

Y a continuación el cliente ejecuta el beat para que se empiecen a realizar las tareas y estas se puedan ver reflejadas en el servidor y también en django con el comando python manage.py runserver.



Django administration

7. Implementación

Este proyecto se de mucha utilidad ya que se puede implementar en la solicitud de reportes y en la generacion de estos, con esto se podria facilitar estos procesos.

8. Conclusión

Como se pudo ver, la herramienta redis y nos funcionan muy bien en este tipo de proyectos en los cuales se maneja celery, ya que el motor de base de datos que al conectarlo en red puede servirnos para cualquier proyecto a futuro y con este ver las llaves que le envia el cliente al servidor.

9. Referencias

- <https://es.wikipedia.org/wiki/Redis>
- <http://blog.enriqueoriol.com/2014/06/integrando-celery-con-django-programar.html>
- <https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introducción—>