Uso de protocolos de Colas de Mensajes

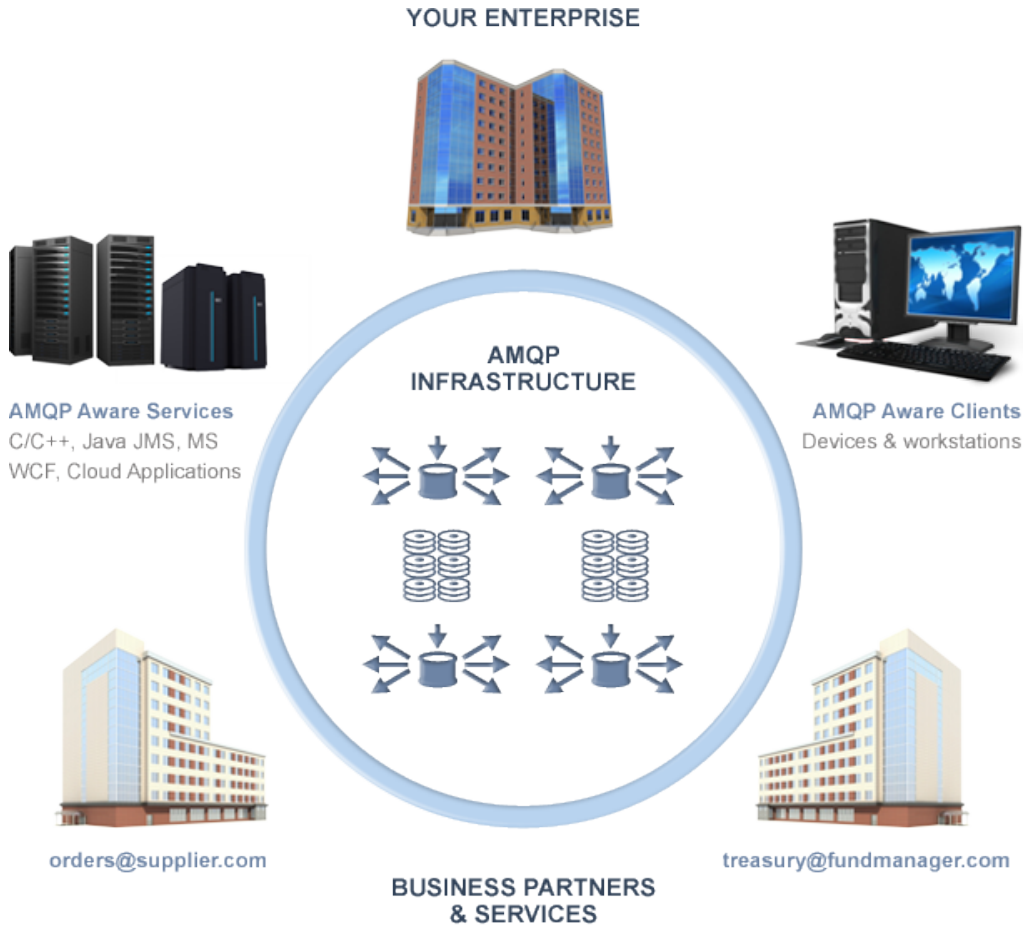
Sistemas Distribuidos

Grado en Ingeniería Informática

Índice

Índice

Advanced Message Queuing Protocolo (AMQP)



AMQP

Estándar abierto para sistemas de pasos de mensajes entre aplicaciones y organizaciones.

AMQ

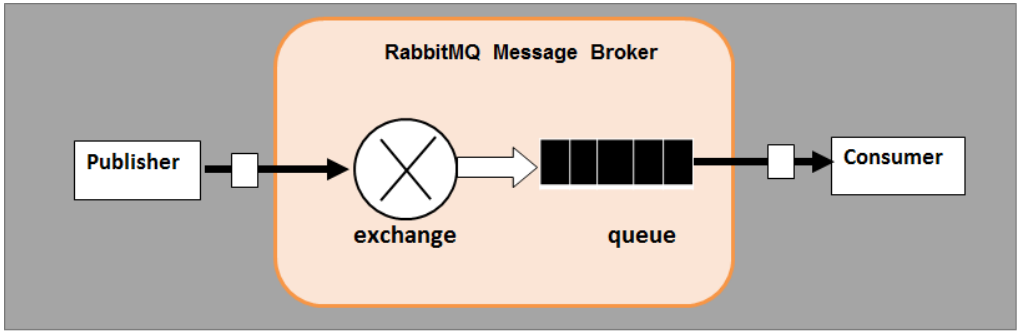


Fig: RabbitMQ

Descripción

- AMQP es un protocolo de mensajes que comunica procesos productores con consumidores.
- Los productores producen los mensajes, que se guardan esperando ser atendidos.
- Los consumidores procesan los mensajes (se eliminan).

Ventajas del AMPQ

Separación conceptual

- Asíncrono: El productor indica la tarea a realizar.
- Extensible: El número de consumidores es adaptable.
- Multi-plataforma.
- Robusto.
  - Recupera si se cae, mensajes persistentes.
  - Comprueba si se procesó bien antes de eliminar el mensaje.
- Múltiples implementaciones *open-source*.

Conceptos

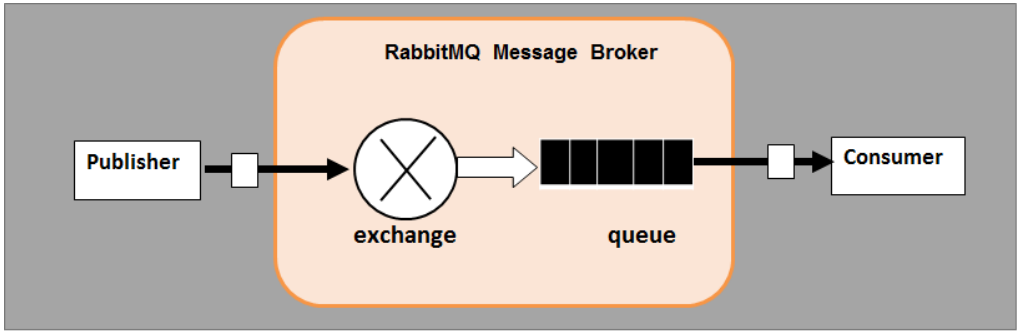
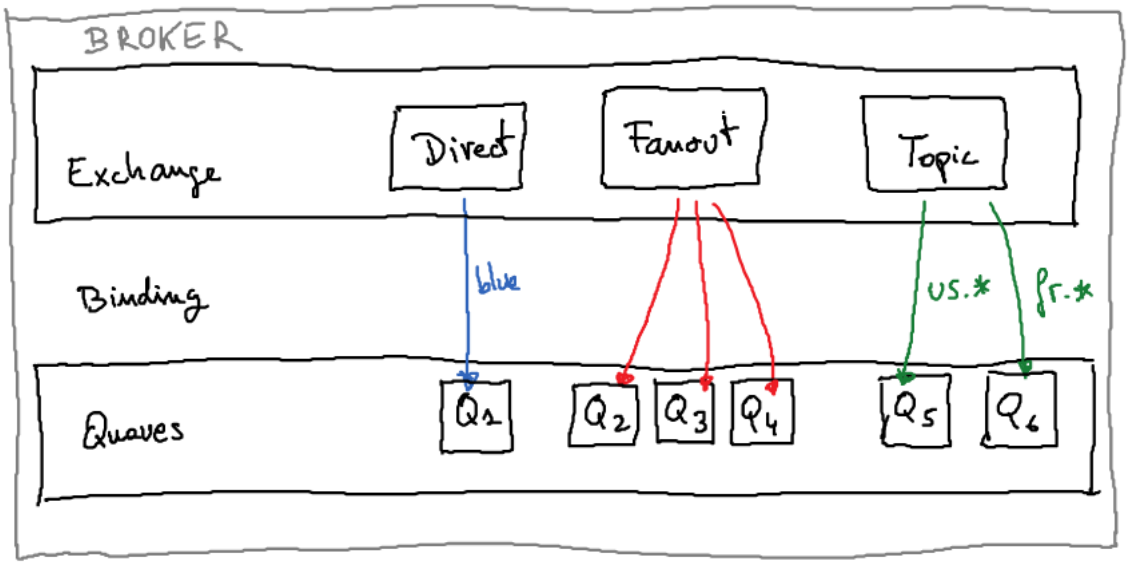


Fig: RabbitMQ

Conceptos

Broker	Sistema que se encarga de recibir y enviar los mensajes usando AMQP.
Cliente	Productor/consumidor que envía o recibe mensajes al Broker.
Canal	Permite varios clientes sobre una conexión física.
Cola	Buffer de mensajes.
Intercambiador	Recibe los mensajes, los filtra y/o enruta a una cola.

Intercambiadores



Tipos

direct Envía directamente a la cola indicada por el productor.

fanout Replica el mensaje a todas las colas.

topic Envía a la cola según el tema/topic del mensaje.

Intercambiadores

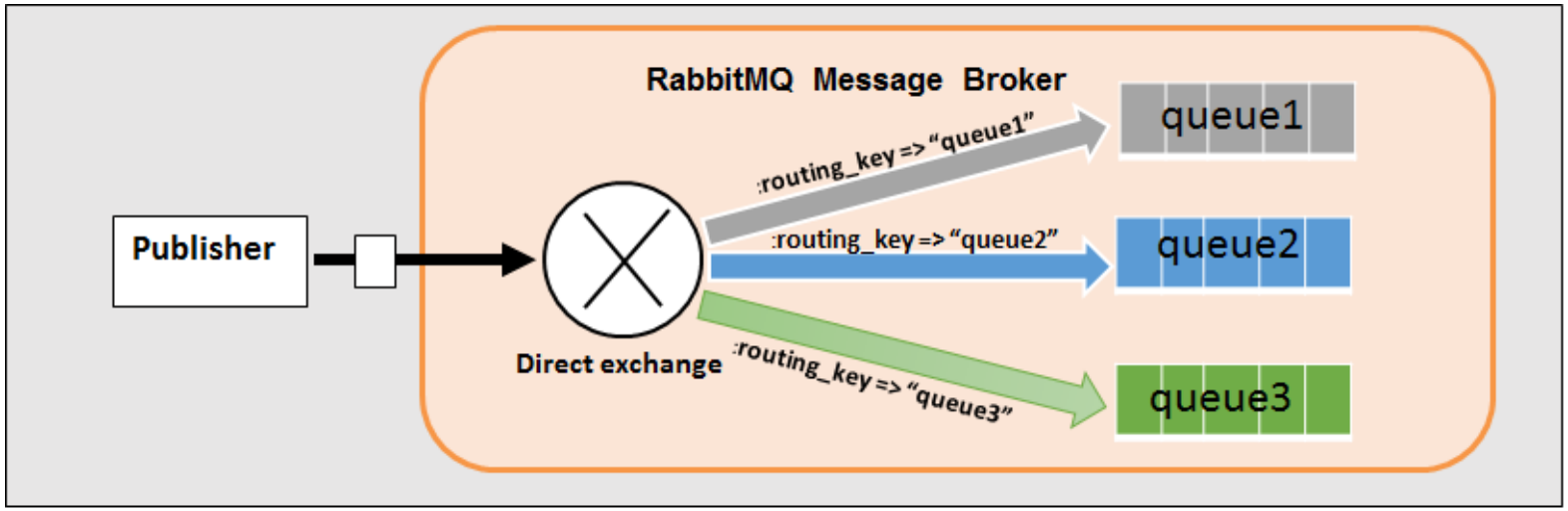


Fig: Direct Exchange Routing

Tipos

direct	Envía directamente a la cola indicada por el productor.
fanout	Replica el mensaje a todas las colas.
topic	Envía a la cola según el tema/topic del mensaje.

Intercambiadores

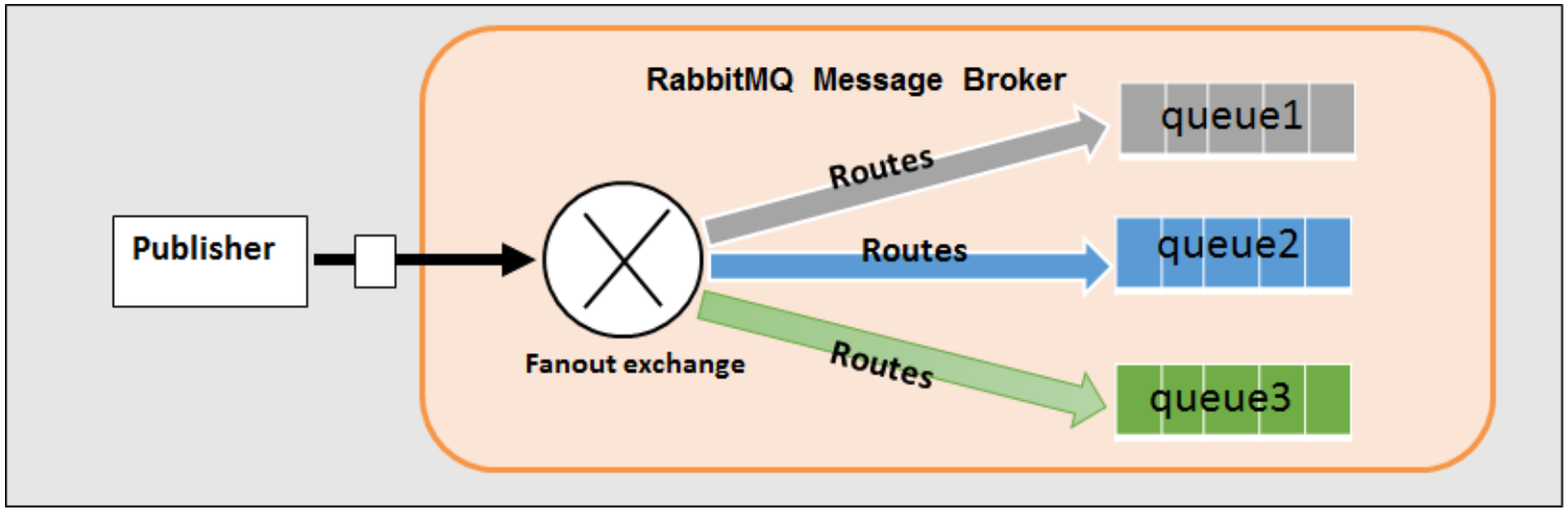


Fig: Fanout Exchange Routing

Tipos

direct	Envía directamente a la cola indicada por el productor.
fanout	Replica el mensaje a todas las colas.
topic	Envía a la cola según el tema/topic del mensaje.

Intercambiadores

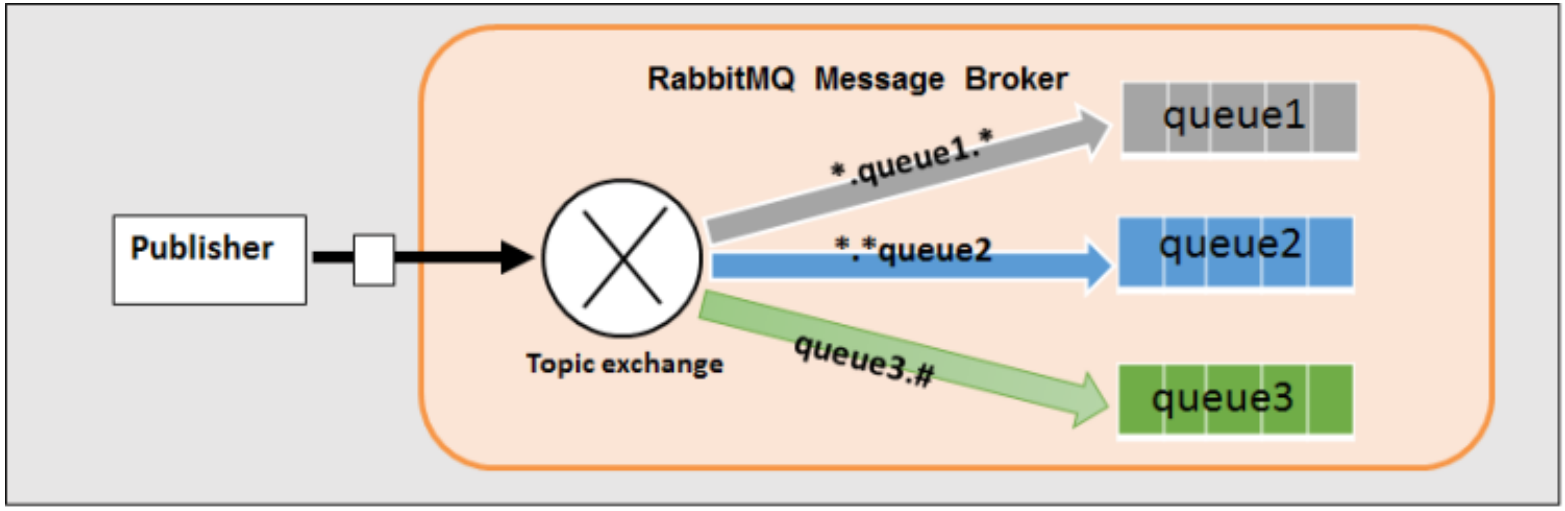


Fig: Topic Exchange Routing

Tipos

direct	Envía directamente a la cola indicada por el productor.
fanout	Replica el mensaje a todas las colas.
topic	Envía a la cola según el tema/topic del mensaje.

Índice

RabbitMQ

RabbitMQ

- Implementación open-source.
- Muy aceptada en las empresas.
- Gran rendimiento (implementado en Erlang).
- Muy fácil de usar: *broker* centralizado.
- Consola web de administración (módulo).
- Muy robusto.

Instalación de RabbitMQ

1. Instalar servidor

   ```
   $ sudo aptitude install rabbitmq-server
   ```

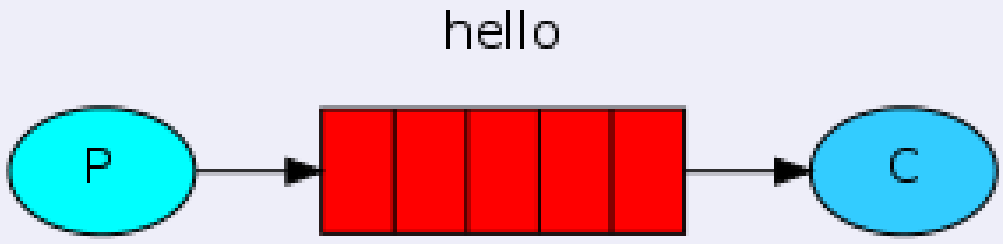
2. Instalar consola de administración

   ```
   $ sudo rabbitmq-plugins enable rabbitmq_management
   ```

3. Comprobar que funciona, con la url `http://localhost:15672/`
4. Entrar con guest/guest.

Ejemplo usando Pika (librería Python)

Objetivo: Hello-world



Hello-world

Creación de colas

- Ambos (productor y consumidor) definen las colas.
  - Sólo se crea si no existiese.

Código común

```
#!/usr/bin/env python
import pika

# Crea conexión
con_params = pika.ConnectionParameters(localhost)
connection = pika.BlockingConnection(con_params)
# Crea Canal
channel = connection.channel()
# Crea cola
channel.queue_declare(queue=hello)
```

Productor y consumidor

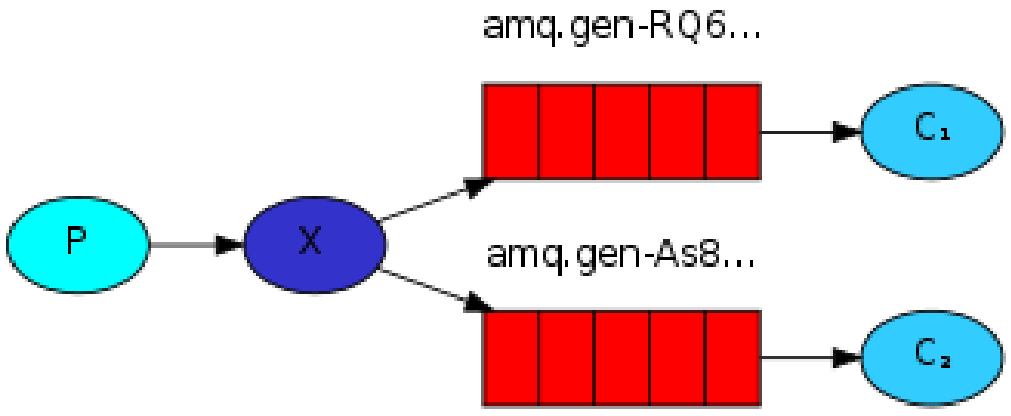
Productor

```
# Envía mensaje
channel.basic_publish(exchange="",
                      routing_key=hello,
                      body=Hello World!)
print("␣[x]␣Sent␣Hello World!")
connection.close()
```

Consumidor

```
# Mensaje de recepción
def callback(ch, method, properties, body):
    print("␣[x]␣Received␣%r" % body)
# Consume mensaje
channel.basic_consume(callback,
                      queue=hello,
                      no_ack=True)
connection.close()
```

Otro ejemplo, repartiendo carga



Comentarios

- El productor no genera las colas, sólo el "exchange".
- El consumidor va a generar una cola nueva para él (temporal).

Productor

Común

```
#!/usr/bin/env python
import pika

# Crea conexión
con_params = pika.ConnectionParameters(localhost)
connection = pika.BlockingConnection(con_params)
# Crea Canal
channel = connection.channel()
# Crea intercambiador
channel.exchange_declare(exchange=logs,
                         type=fanout) # or direct or topic
```

Productor

```
# Crea mensaje a enviar
message = "info: Hello World!"
# Envía mensaje
channel.basic_publish(exchange=logs,
                      routing_key="",
                      body=message)
```

Consumidor

Crea canal y lo asocia

```
# Declara un canal nuevo (al ser exclusivo se borrará)
result = channel.queue_declare(exclusive=True)
queue_name = result.method.queue
# Asocia el exchange con la cola
channel.queue_bind(exchange=logs,
                   queue=queue_name)
```

Escucha y consume

```
print( [*] Waiting for logs.  To exit press CTRL+C)

def callback(ch, method, properties, body):
    print(" [x] %r" % body)

# Recibe de la cola
channel.basic_consume(callback, queue=queue_name,
                      no_ack=True)

channel.start_consuming()
```

Definiendo exchanges

Exchange direct

```
channel.exchange_declare(exchange="direct_logs",
                         type="direct")
severity=["error", "warning"]

for severity in severities:
    channel.queue_bind(exchange="direct_logs",
                       queue=queue_name,
                       routing_key=severity)
```

Exchange topic

```
channel.exchange_declare(exchange="topic_logs",
                         type="topic")

for binding_key in binding_keys:
    channel.queue_bind(exchange="topic_logs",
                       queue=queue_name,
                       routing_key=binding_key)
```

Uso manual de RabbitMQ

Portable

- Librerías de todos los lenguajes.
- Hay que aprender cada lenguaje.

Tedioso

- No se usa mucho RabbitMQ directamente.
- Uso por medio de librerías.

Celery

- Permite usar RabbitMQ/Redis para tareas asíncronas.

Índice

Motivación de Celery

Necesidad de tareas asíncronas en web

Facebook Likes, añadir amistad.

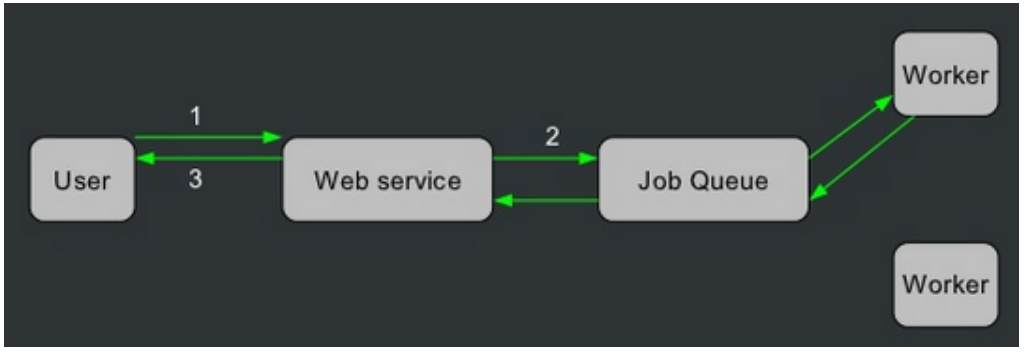
Flickr/Instagram Crear imágenes con distinta resolución.

No es bueno hacer esperar al usuario

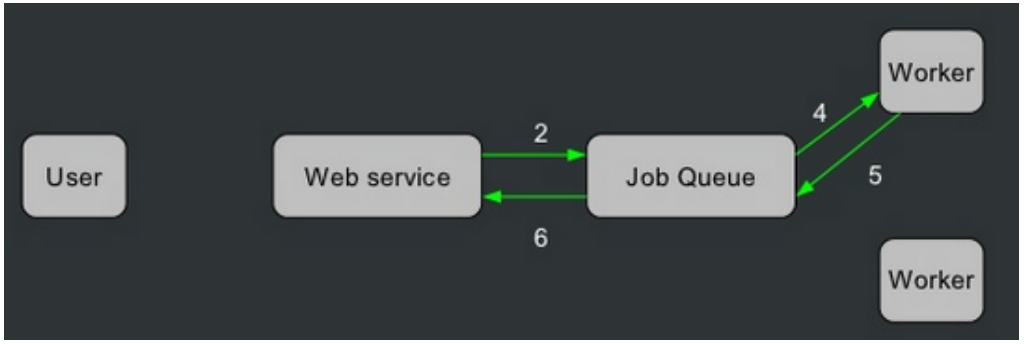
- Las tareas en segundo plano no le incumben.
- No necesita esperar.
- Se le puede notificar luego (email, . . .).

Flujo de trabajo

Paso 1: Se apunta la acción y se responde al usuario



Paso 2: Se termina de realizar la operación



Celery

Uso de AMQP

- Escalable: varios *worker* procesando peticiones.
- Robusto.
- Uso opcional de ack (repite si no se procesa bien).

Funcionalidades

- Tareas asíncronas.
- Tareas con cierta periodicidad.
- Control de tiempos, y reinicios.

Celery

Define el evento asíncrono con decorador task

- Muy sencillo, uso de decorador task.

Ejemplo: fichero tasks

```
from celery import Celery, task
# Crea usando RPC para devolver los datos, y el broker AMQP
app = Celery("tasks", backend="rpc://",
             broker="pyamqp://guest@localhost//")

# Con no_ack no espera un ack al terminar
@app.task(no_ack=True)
def add(a, b):
    return a+b
```

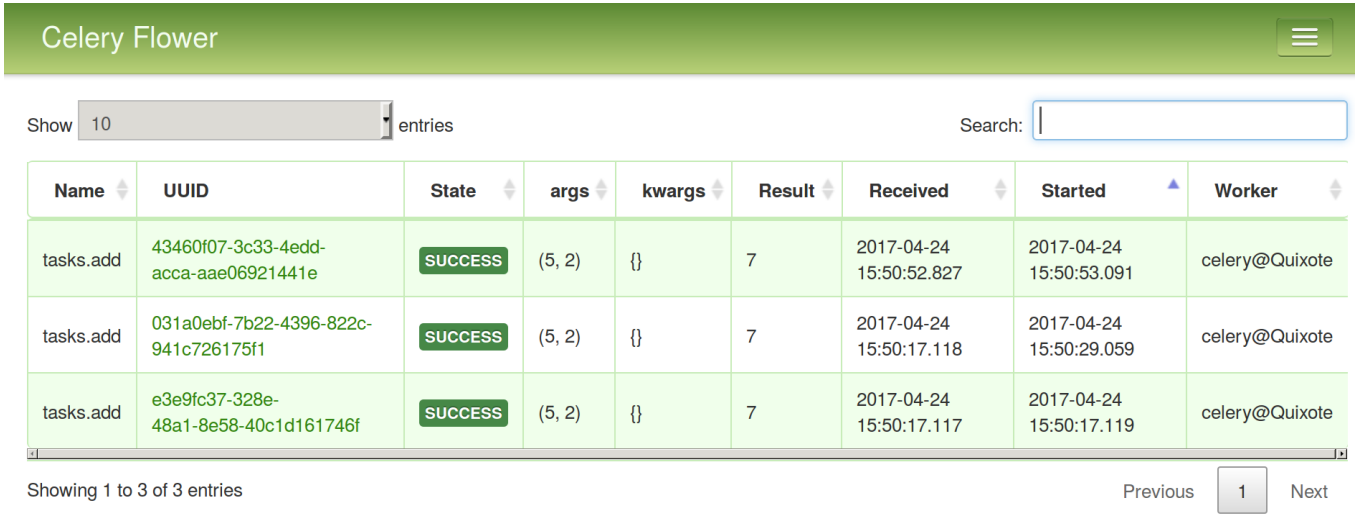
Probar el programa

```
$ celery -A tasks worker -l=info
```

Consola de administración de celery

Instalar y arrancar consola de administración (flower)

```
$ pip install flower --user
$ flower
$ (en puerto localhost:5555)
```



Lanzando proceso asíncrono

Puede ser muy sencillo

```
import tasks

# result = tasks.async_call((5,2))
result = tasks.add.delay(5, 2)
# Espera como mucho un segundo (opcional)
print(result.get(timeout=1))
```

Para lanzar la operación

delay Con los mismos parámetros que la función.

apply_async Versión más completa de llamada:

- Tupla de parámetros.
- parámetros opcionales a Celery.

Ejemplos de llamada

Algunos ejemplos

```
# Ejecuta dentro de 10 segundos
T.apply_async(countdown=10)
# Ejecuta en 10 segundos desde ahora (usanto eta)
T.apply_async(eta=now + timedelta(seconds=10))
# Ejecuta dentro de un minuto, pero lo revoca si no se atiende
# en 2 minutos
executes in one minute from now, but expires after 2 minutes.
# Expira en 2 dias
T.apply_async(expires=now + timedelta(days=2))
```

Obtener valor

Modo síncrono

get([timeout]) Recupera el valor, bloqueante.

Modo asíncrono

sucessful() Comprueba si terminó la tarea de forma correcta.

failed() Indica si falló.

waiting() Indica si tiene que esperar.

revoke() Revoca la tarea lanzada.

Ejemplos obteniendo el valor

Espera inmediata (no asíncrona)

```
# Espera la ejecución
result = tasks.add(2, 3).delay()
value1 = result.get()
```

Espera al cabo de un rato

```
# No espera, hace cosas mientras
result = tasks.add(2, 3).delay()
...
# Ahora obtiene el valor
value1 = result.get()
```

Ejemplos obteniendo el valor

Comprobando el estado

```
result = tasks.add(2, 3).delay()
...
# Si están los resultados espera
if result.successful():
   value = result.get()
else result.failed():
   raise ...
elif not result.failed():
   ...
   # Ahora exige sus resultados
   value = result.get()
```

Signatures

Signatures

Convierten la llamada con parámetros en una llamada sin parámetros.

Ejemplo

```
result = tasks.add(2, 3)
value1 = result.get()
# add2 = tasks.signature(2, 3)
add1_23 = tasks.s(2, 3)
# Empieza cuando se llamada
value2 = add2().get()
assert value1 == value2
# Uso de parciales, se reemplaza
add_partial = task.s(2)
value3 = add_partial(2).get()
assert value2 == value3
# El .si() no admite parciales, mas eficiente que .s()
add2_23 = tasks.si(2,3)
value4 = add2_23().get()
assert value4 == value3
```

Uso de Signaturas

Uso de signaturas

- Encadenar tareas fácilmente (secuenciales).
- Agrupar tareas asíncronas.

Encadenar tareas

```
newfun = chain(add.s(2, 2), add.s(4))
res = newfun()
res.get() # Da 8
```

Nueva sintaxis de encadenar tareas

```
newfun2 = chain(add.s(2, 2) | add.s(5))
res = newfun2()
res.get() # Da 9
```

Uso de Signaturas

Uso de signaturas

- Encadenar tareas fácilmente (secuenciales).
- Agrupar tareas asíncronas.

Agrupando tareas

```
lazy_group = group([add.s(2, 2), add.s(4, 4)])
promise = lazy_group() # <-- Empieza a evaluar
promise.get() # <-- Espera al resultado, da [4, 8]
```

Independientes

Cada tarea debe ser independiente del resto.

Algunas opciones de group

Opciones de group

successful()	Todas exitosas.
failed()	Alguna falló.
waiting()	Si alguna todavía no ha terminado.
complete_count()	Número de tareas terminadas.
revoke()	Revoca todas las tareas.
get()	Espera a todas.
join()	Espera a todas (más ineficiente).

Ejemplos de uso de grupos

Ejemplo

```
from celery import group
from tasks import add

job = group([
    add.s(2, 2),
    add.s(4, 4),
    add.s(8, 8),
    add.s(16, 16),
    add.s(32, 32),
])

result = job.apply_async()

result.ready()
result.successful()
result.get() # da [4, 8, 16, 32, 64]
```

Lanzando tareas

Línea de comandos

1. Incluir en cada nodo *worker*

```
$ celery worker -A /project/ [-l=...]
```

2. Lanzar las tareas asíncronas (puede ser con espera).
3. Las tareas se dividen entre los workers pendientes.

Planificador

1. Asignarle una hora a las tareas.
    1. Indicado en el decorador de la propia tarea.
    2. Mediante una opción de planificación (tarea y horaÇ).
2. Ejecutar el planificador del celery:

```
$ celery -A /proyecto/ worker -B
```

Planificando tareas en el decorador

Uso de timedelta

timedelta(<unidad>=) Espera tanto como unidad.

time seconds, minutes, hours.

days days.

Timedelta es relativo

Usarlo con now para sumar la hora actual en campos como eta.

```
@task(countdown=timedelta(seconds=30))
@task(eta=now+timedelta(seconds=30))
```

Planificando tareas

Uso de crontab

crontab(hour=xxx,minutes=...) Permite especificar hora concreta, misma sintaxis que crontab en la shell.

Ejemplo

```
# Se ejecuta esa tarea todos los días a las 7:30
@periodic_task(run_every=crontab(hour=7, minute=30))
# Se ejecuta cada media hora
@periodic_task(run_every=crontab(minute="*/30"))
```

Cuidado con las horas

- Por defecto la hora es UTC.
- Se debe cambiar al uso horario que interesa.

```
app.config.timezone = "Europe/Madrid"
```

Planificando fuera de los decoradores

Ejemplo de uso

```
from celery.schedules import crontab

app.conf.beat_schedule = {
    # Executes every Monday morning at 7:30 a.m.
    "add-every-monday-morning": {
        "task": "tasks.add",
        "schedule": crontab(hour=7, minute=30, day_of_week=1),
        "args": (16, 16),
    },
}
```

Consulta el *Manual* para obtener mayor información.

Ejemplo completo

Ejemplo completo: task_period.py

```
from celery.schedules import crontab, timedelta
from celery import Celery

app = Celery("tasks", backend="rpc://", broker="pyamqp://guest@localhost//")

app.conf.beat_schedule = {
        "every-day": {
                "task": "hazalgo",
# "schedule": crontab(hour=13, minute=21),
                "schedule": timedelta(seconds=5),
                "args": (4, 5),
        }
}
app.conf.timezone = "Europe/Madrid"

@app.task(name="hazalgo")
def suma(n1, n2):
        print("Running")
        with open("/tmp/algo.txt", "w") as fout:
                print(n1+n2, file=fout)
```

Lanzando el ejemplo concreto

Instrucción

```
$ celery -A task_period -B -l=info
```

Salida

```
[tasks]
  . hazalgo
  . suma

[...: INFO/Beat] beat: Starting...
[...: INFO/MainProcess] Connected to amqp://guest:**@127.0.0.1:5672//
[...: INFO/Beat] Scheduler: Sending due task every-day (hazalgo)
[...: INFO/MainProcess] celery@Quixote ready.
[...: INFO/MainProcess] Received task: hazalgo[...]
[...: INFO/MainProcess] Received task: hazalgo[...]
[...: WARNING/PoolWorker-2] Running
[...: WARNING/PoolWorker-5] Running
[...: INFO/PoolWorker-2] Task hazalgo[...] succeeded in 0.0167s: None
[...: INFO/PoolWorker-5] Task hazalgo[...] succeeded in 0.0174s: None
```

Índice

Referencias

AMQP y RabbitMQ

- Sobre AMQP : `http://amqp.org/`
- Sobre RabbitMQ : `http://rabbitmq.org/`
- Introducción de Javier Arias Lozada

Sobre Celery

- `http://celeryproject.org`
- Manual: `http://celery.org/docs/getting%C2%ADstarted/`