Komlogd Documentation

Publicación 0.1.0

Komlog

13 de February de 2017

Índice general

Komlogd es el agente del servicio web Komlog, siendo éste una plataforma pensada para visualizar y compartir series temporales.

Komlogd está escrito en python y puede utilizarse como aplicación independiente o como API, para así integrar fácilmente la funcionalidad ofrecida por Komlog en cualquier aplicación python.

Instalación automática:

```
pip install komlogd
```

Komlogd está en el listado de PyPI se puede instalar con `pip` o `easy_install`.

Requisitos

Komlogd necesita Python 3.5 o superior. Además, necesita los siguientes paquetes adicionales (que se instalarán con `pip` automáticamente):

- aiohttp
- cryptography
- pandas
- pyyaml

Nota: Para instalar las dependencias, pip necesitará varios paquetes de tu distribución, entre ellos `gcc`, `libffi-dev`, `numpy`, etc. En dispositivos lentos la instalación de pandas y cryptography con `pip` puede tardar bastante, por lo que tal vez prefieras instalar directamente los paquetes que proporciona tu distribución.

Documentación

CAPÍTULO 1

Instalación y primeros pasos

1.1 Instalación automática

Komlogd está en el listado de PyPI y se puede instalar con `pip` o `easy_install`:

```
pip install komlogd
```

1.2 Requisitos

Komlogd necesita Python 3.5 o superior. Además, necesita los siguientes paquetes adicionales (que se instalarán con `pip` automáticamente):

- aiohttp
- cryptography
- pandas
- pyyaml

Nota: Para instalar las dependencias, pip necesitará varios paquetes de tu distribución, entre ellos `gcc`, `libffi-dev`, `numpy`, etc. En dispositivos lentos la instalación de pandas y cryptography con `pip` puede tardar bastante, por lo que tal vez prefieras instalar directamente los paquetes que proporciona tu distribución.

1.3 Primera ejecución

Una vez instalado el agente, lo arrancaremos ejecutando el siguiente comando:

```
komlogd &
```

Durante esta primera ejecución, komlogd inicializará los archivos de configuración y creará el par de claves RSA utilizado para la comunicación con Komlog. Debido a que el agente crea el archivo de configuración sin establecer variables necesarias para su funcionamiento, como por ejemplo el usuario de acceso a Komlog, el agente terminará su ejecución.

En el directorio `$HOME` del usuario que lanzó la ejecución de komlogd se debería haber creado el directorio `.komlogd` con la siguiente estructura:

```
.komlogd
-- key.priv
-- key.pub
-- komlogd.yaml
-- log
    -- komlogd.log
```

Los archivos creados son los siguientes:

- **key.priv**: Es la clave privada del agente.
- **key.pub**: Es la clave pública del agente. Esta clave la añadiremos a través de la web de Komlog para autorizar al agente.
- **komlogd.yaml**: Archivo principal de Komlogd, con toda la configuración necesaria del agente.
- **log/komlogd.log**: Archivo de log del agente.

> **Nota:** La clave privada del agente nunca se transmitirá a Komlog

Una vez llegados a este punto, podemos pasar a la *Configuración* de komlogd.

CAPÍTULO 2

Configuración

2.1 Configuración de acceso a Komlog

Una vez que tenemos inicializada la estructura básica de ficheros y directorios de komlogd, como se explica en el punto *Primera ejecución*, es el momento de configurar komlogd para que pueda comunicarse con Komlog.

2.1.1 Configuración de usuario y clave RSA

Para que komlogd tenga acceso a Komlog, debemos indicar el usuario de conexión y la clave del agente en el archivo de configuración (**komlogd.yaml**).

El usuario lo especificaremos estableciendo la siguiente variable:

```
- username: <username>
```

Sustituyendo <*username*> por nuestro usuario de acceso a Komlog.

La clave del agente por defecto será la generada durante la primera ejecución, que debe estar en la ruta `$HOME/.komlogd/key.priv`. En cualquier caso, si quisieramos utilizar una clave diferente, podríamos establecerla mediante la siguiente variable en el archivo de configuración:

```
- keyfile: <path_to_key.priv>
```

Sustituyendo <*path_to_key.priv*> por el archivo con nuestra clave RSA privada.

Si no queremos utilizar una clave distinta a la generada automáticamente, no es necesario establecer esa variable.

Nota: La clave privada del agente nunca se transmitirá a Komlog

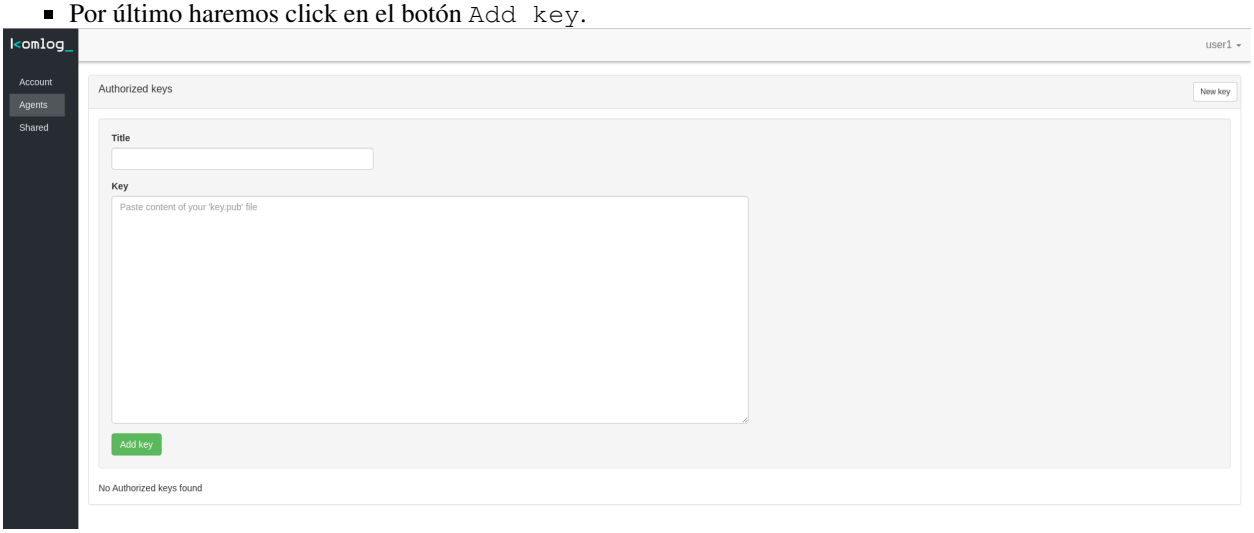
---

2.1.2 Autorización del agente en Komlog

Para que el agente pueda enviar información a Komlog, es necesario añadir la clave pública al listado de claves autorizadas. Para ello realizaremos lo siguiente:

- Accederemos a la página de configuración de nuestra cuenta de Komlog.
- Vamos al submenú `Agents` y hacemos click sobre el botón `New Agent`.
- Se mostrará un formulario con dos campos, en el primero de ellos introduciremos el nombre que queremos dar al agente. En el segundo campo pegaremos el contenido de nuestra clave pública, por defecto almacenada en el archivo `key.pub` de nuestro directorio de komlogd.

- Por último haremos click en el botón `Add key`.



2.2 Configuración de tareas programadas

komlogd permite la ejecución de tareas programadas y el envío del resultado a Komlog. Esta funcionalidad te permite visualizar estos resultados via web, identificar variables directamente en los textos, o compartir el contenido con otros usuarios.

Supongamos que queremos enviar cada hora la salida del comando "*df -k*" a Komlog.

Para ello editaríamos el archivo de configuración de komlogd (**komlogd.yaml**) y añadiríamos un bloque **job** como el siguiente:

```
- job:
    uri: system.info.disk
    command: df -k
    enabled: yes
    schedule:
        - '0 * * * *'
```

Los parámetros que definen el *bloque job* son los siguientes:

- **uri**: La uri es un identificador único que asignamos a nuestros datos dentro de Komlog.

En Komlog, todos los datos que el usuario sube se organizan en un árbol como si de un sistema de ficheros se tratara. Cada identificador dentro de este árbol se conoce como **uri**.

La *uri* identifica de forma unívoca el dato que estamos subiendo, permitiéndonos identificarlos y localizarlos rápidamente. Al igual que con un sistema de ficheros, existe la posibilidad de anidar los datos en diferentes niveles. Si en sistemas *UNIX* se utiliza el carácter **/** para ello, en Komlog se utiliza el punto **.**

Por ejemplo, si tenemos las siguientes uris: *system.info.disk* y *system.info.disk.sda1*, éstas se anidarían de la siguiente manera:

```
system.info.disk
-- sda1
```

Importante: Una uri puede contener **exclusivamente** los siguientes caracteres:

- Letras [A-Z] mayúsculas o minúsculas: Caracteres ASCII de la *A* a la *Z* mayúsculas o minúsculas. No son válidos caracteres no ASCII como la *ñ*, o acentuados (*á,é*), etc.

- Números [0-9].
- Caracteres especiales:
  - Guión (-), underscore (_)
  - Punto (.)

La uri **no puede empezar** por el carácter especial punto (.).

- **command**: Es el comando a ejecutar.

Se puede indicar un comando del sistema operativo o cualquier script. La salida por pantalla será lo que se envíe a Komlog (La salida por *stdout*, la salida *stderr* no se envía).

Importante: Hay que tener en cuenta que en el comando a ejecutar no se pueden añadir caracteres especiales como son las **tuberías (**|**)**, o **redirecciones (<,>)**, por lo que si se desean ejecutar comandos enlazados mediantes tuberías o redirecciones habría que hacerlo en un script.

- **enabled**: Puede tomar los valores *yes* o *no*. Indica si el *job* está habilitado.
- **schedule**: El schedule determina cuándo se ejecutará el job. Se utiliza el siguiente formato:

```
------------- minutos (0 - 59)
| ------------- horas (0 - 23)
| | ------------- día del mes (1 - 31)
| | | ------------- mes (1 - 12)
| | | | ------------- día de la semana (0 - 6) (Domingo a Sábado)
| | | | |
| | | | |
| | | | |
* * * * *
```

Además acepta los siguientes caracteres especiales:

- El asterisco (*) para indicar todos los posibles valores de un grupo.
- La coma (,) para indicar varios valores en un grupo.
- El carácter / para indicar los valores de una división cuyo resto sea 0. Por ejemplo, en lugar de indicar los minutos *0,10,20,30,40,50* podemos indicar **/10*.

El parámetro schedule permite indicar un listado de ellos, para así poderlo ejecutar en base a diferentes planificaciones.

Se pueden añadir tantos *bloques job* como se desee. Cada uno se lanza en un proceso independiente, por lo que su ejecución no interfiere con la ejecución de komlogd, tan solo hay que tener en cuenta que para proteger al sistema, **komlogd no planificará la ejecución un job hasta que la ejecución anterior de ese mismo job haya terminado**. Por ejemplo, si tengo un job cuya ejecución se demora 10 minutos y lo planifico para que se ejecute cada 5 minutos, komlogd no lo lanzará con la frecuencia configurada.

2.2.1 Carga de jobs desde archivo externo

En algunas ocasiones nos puede interesar que komlogd cargue los jobs a ejecutar desde un archivo externo, en lugar de añadirlos directamente en el archivo *komlogd.yaml*

Para ello, editamos el archivo *komlogd.yaml* y realizamos lo siguiente:

1. Habilitamos la opción que permite cargar jobs desde un archivo externo:

```
    - allow_external_jobs: yes
```

2. Por cada fichero de jobs, añadimos lo siguiente:

```
    - external_job_file: <path_to_file>
```

sustituyendo <*path_to_file*> por la ruta del archivo que contiene el listado de *bloques job* que queremos ejecutar.

Podemos añadir tantos bloques *external_job_file* al archivo *komlogd.yaml* como queramos.

2.3 Configuración de funciones impulso

Una *función impulso* es una función que se ejecuta como respuesta a una actualización de los datos de una *uri*.

komlogd permite a un usuario suscribirse a cualquiera de sus *uris* y recibir notificaciones cuando se reciban datos en ellas.

Esta funcionalidad la podemos utilizar para realizar automatización de tareas, generación de alarmas, comunicación con servicios externos, análisis de datos en tiempo real, y en definitiva cualquier tarea que se nos ocurra que pueda estar asociada a eventos.

En el apartado *Funciones impulso* se explica cómo crear este tipo de funciones correctamente.

Una vez que tenemos el archivo con las funciones impulso, para añadirlas a la configuración de komlogd editaríamos el archivo de configuración (**komlogd.yaml**) y añadiríamos un nuevo bloque **impulses** como el siguiente:

```
- impulses:
    enabled: yes
    filename: <path_to_file>
```

Los parametros del *bloque impulses* son los siguientes:

- **enabled**: Puede tomar los valores *yes* o *no*. Indica si el bloque *impulses* está habilitado.
- **filename**: Ruta del archivo que contiene las funciones *impulso*. La ruta puede ser absoluta o relativa al directorio de configuración de komlogd.

Al igual que en el caso de los jobs, se pueden añadir tantos *bloques impulses* como se desee.

2.4 Configuración del nivel de log

komlogd permite adaptar algunos de los parámetros de logging en función de nuestras preferencias.

La configuración de logs viene establecida en el bloque *logging* dentro del archivo de configuración de komlogd (*komlogd.yaml*). Por defecto tiene estos valores:

```
- logging:
    level: DEBUG
    rotation: yes
    max_bytes: 10000000
    backup_count: 3
    dirname: log
    filename: komlogd.log
```

Los parámetros del *bloque logging* son los siguientes:

- **level**: Indica el nivel de log. Los valores posibles son *CRITICAL, ERROR, WARNING, INFO, DEBUG, NOTSET*.

- **rotation**: Indica si se rotará el archivo de logs. Los valores posibles son *yes* o *no*.
- **max_bytes**: En caso de rotar el fichero, indica el tamaño en bytes que tiene que alcanzar para que se rote.
- **backup_count**: Indica el número de rotaciones a almacenar del fichero de logs.
- **dirname**: Directorio en el que se almacenará el fichero de log. La ruta puede ser absoluta o relativa al directorio de configuración de komlogd.
- **filename**: Nombre del fichero de logs.

Una vez que hayamos configurado komlogd, podemos proceder a su ejecución como ya vimos en el apartado *Instalación y primeros pasos*:

```
komlogd &
```

CAPÍTULO 3

API

Komlogd puede utilizarse como librería en otras aplicaciones, para así integrar en ellas la funcionalidad ofrecida por Komlog.

Nota: komlogd utiliza asyncio internamente, por lo que es necesario que la aplicación donde se integre corra un loop de asyncio para su funcionamiento.

3.1 Inicio de sesión en Komlog

Para establecer una sesión con komlog necesitamos:

- Un usuario de acceso.
- Una clave RSA.

Nota: Recuerda que komlogd durante su *Primera ejecución* crea la clave RSA para la comunicación con Komlog. En caso de querer utilizar otra clave RSA, ésta debe ser mayor o igual a 4096 bytes.

Para inicializar una sesión en Komlog bastaría con realizar lo siguiente:

```
import asyncio
from komlogd.api import session, crypto

loop = asyncio.get_event_loop()

async def example():
    username='username'
    privkey=crypto.load_private_key('/path/to/key.priv')
    komlog_session=session.KomlogSession(username=username, privkey=privkey)
    await komlog_session.login()
    # En este punto la sesión estaría establecida.
    # Si necesitamos que nuestra tarea espere mientras
    # la sesión está establecida:
    # await komlog_session.t_loop

    # Para cerrar la sesión:
    await komlog_session.close()

try:
```

```
    loop.run_until_complete(example())
except:
    loop.stop()
loop.close()
```

Los pasos son:

- Establecemos el usuario de la sesión.
- Cargamos la clave privada del agente, llamando a la función *crypto.load_private_key()* pasándole como parámetro la ruta del archivo que la contiene.
- Inicializamos un objeto *KomlogSession*, que será el encargado de contener nuestra sesión con Komlog, pasándole como parámetros el usuario y la clave privada.
- Llamamos al método *login()*. Este método crea una tarea en el loop de asyncio que será la encargada de realizar login en el servidor y mantener nuestra sesión con Komlog. Hasta que no se ejecute el loop de asyncio no se realizará login en Komlog.
- Por último arrancamos el loop de asyncio. En este momento se ejecutará la tarea de login en Komlog.

A partir de ese momento, siempre y cuando el login haya sido correcto, tendríamos una sesión establecida con Komlog, y estaríamos preparados para enviar y recibir información.

Internamente, komlogd establece con el servidor una conexión mediante *web sockets*, por lo que la sesión se mantiene establecida en todo momento. En caso de recibir una desconexión, la propia librería se encarga de restablecerla.

Importante: Para que el login sea correcto, hay que autorizar previamente la clave pública de komlogd, como se explica en *Autorización del agente en Komlog*.

La clave pública se puede obtener a partir de la clave privada, ese es el motivo por el cual nunca tienes que introducir la clave pública como parámetro, sino la privada, pero komlogd **nunca** enviará la clave privada a Komlog.

Ante un inicio de sesión fallido porque la clave pública no ha sido autorizada, komlogd la muestra en los logs, para que el usuario vea cual debe autorizar.

3.2 Envío de datos a Komlog

En Komlog el usuario organiza sus métricas en una estructura en árbol en la que cada métrica está identificada unívocamente por lo que llamamos *uri*. A esta estructura la llamamos *el modelo de datos del usuario*. A los diferentes valores que puede tomar una métrica a lo largo del tiempo se les conoce como *samples*.

En el modelo de datos del usuario se pueden crear métricas de los siguientes tipos:

- **Datasource**
- **Datapoint**

Komlog crea las métricas automáticamente la primera vez que subimos datos de ellas. El tipo de datos de una métrica se establece cuando ésta se crea, y no hay posibilidad de cambiarlo mientras exista. Para modificar el tipo de dato habría que borrar la métrica (lo que implicaría borrar todas las muestras de datos recibidas de ella) y volverla a crear con un tipo diferente.

Métricas de tipo Datasource

Una métrica de tipo Datasource se utiliza para almacenar textos. Podemos subir cualquier texto de una longitud máxima de 130KB.

Para enviar a Komlog un sample de un Datasource se podría hacer de la siguiente manera:

```
import asyncio
import pandas as pd
from komlogd.api import session, crypto
from komlogd.api.model.orm import Datasource, Sample

loop = asyncio.get_event_loop()

async def send_datasource_sample():
    # establecemos la sesión
    username='username'
    privkey=crypto.load_private_key('/path/to/key.priv')
    komlog_session=session.KomlogSession(username=username, privkey=privkey)
    await komlog_session.login()

    # preparamos el sample
    uri='my_datasource'
    data='Datasource content'
    ts = pd.Timestamp('now',tz='Europe/Madrid')
    metric = Datasource(uri=uri)
    sample = Sample(metric=metric, ts=ts, data=data)

    # enviamos sample y cerramos sesión
    komlog_session.send_samples(samples=[sample])
    await komlog_session.close()

try:
    loop.run_until_complete(send_datasource_sample())
except:
    loop.stop()
finally:
    loop.close()
```

Métricas de tipo Datapoint

Una métrica de tipo Datapoint se utiliza para almacenar valores numéricos. Se aceptar variables de tipo int, float o Decimal (de estas últimas sólamente las que tienen representación numérica, es decir, no se aceptan valores como *infinity*, *-infinity*, *NaN*, etc).

En el siguiente ejemplo se muestra como enviar un par de samples asociados a dos métricas de tipo Datapoint:

```
import asyncio
import pandas as pd
from komlogd.api import session, crypto
from komlogd.api.model.orm import Datapoint, Sample

loop = asyncio.get_event_loop()

async def send_datapoint_samples():
    # establecemos la sesión
    username='username'
    privkey=crypto.load_private_key('/path/to/key.priv')
    komlog_session=session.KomlogSession(username=username, privkey=privkey)
    await komlog_session.login()

    # preparamos samples
    samples = []
    ts = pd.Timestamp('now',tz='Europe/Berlin')
    metric1 = Datapoint(uri='cpu.system')
    metric2 = Datapoint(uri='cpu.user')
```

```
    samples.append(Sample(metric=metric1, ts=ts, data=14.63))
    samples.append(Sample(metric=metric2, ts=ts, data=28.5))

    # enviamos samples y cerramos sesión
    komlog_session.send_samples(samples=samples)
    await komlog_session.close()

try:
    loop.run_until_complete(send_datapoint_samples())
except:
    loop.stop()
finally:
    loop.close()
```

Komlog considera las diferentes métricas del modelo de datos del usuario como series temporales independientes, por lo que cuando subimos datos a alguna de nuestras métricas siempre hay que asociarle un *timestamp*. **El usuario es el encargado de establecer el timestamp, por lo que el valor del timestamp no tiene por qué coincidir con el del momento en el que se envían los datos.**

El timestamp que asociamos al contenido de una métrica puede ser de los siguientes tipos:

- tipo *pandas.Timestamp*
- tipo *datetime.datetime*
- tipo string en formato ISO8601

Hay que tener en cuenta que **es necesario incluir la zona horaria y que la precisión máxima aceptada es de milisegundos**.

3.3 Funciones impulso

En komlogd existe la posibilidad de ejecutar una función cada vez que llegue una muestra de una métrica. A este tipo de funciones les denominamos *funciones impulso*.

Para crear una función impulso simplemente hay que aplicarle el decorador *@impulsemethod*. Este decorador admite los siguientes parámetros:

- **uris**: lista con las uris de las métricas a las que queremos suscribirnos.
- **data_reqs**: objeto de tipo DataRequirements, donde le indicamos los requisitos a nivel de datos que tiene la función para su correcta ejecución.
- **min_exec_delta**: objecto tipo pandas.Timedelta. Este parámetro indica el periodo mínimo entre ejecuciones de la función. Por defecto, komlogd ejecutará la función impulso cada vez que se reciban muestras en los métricas suscritas, sin embago, este comportamiento puede no siempre ser el deseado, por lo que este parámetro indica a komlogd que entre ejecución y ejecución al menos debe haber pasado el tiempo especificado.

El siguiente código muestra como se crearía una función impulso:

```
from komlogd.api.impulse import impulsemethod

@impulsemethod(uris=['cpu.system','cpu.user'])
async def example():
    print('hello komlog.')
```

En el ejemplo anterior, cada vez que se actualicen las métricas *cpu.system* y *cpu.user* komlogd ejecutaría la función *example*. Como se puede ver, example es una corrutina. **El decorador @impulsemethod puede aplicarse tanto a funciones normales o como a corrutinas**.

Una función impulso puede provocar la actualización de métricas en nuestro modelo de datos. Para ello debe devolver una lista con los samples que se deben enviar a Komlog:

```
from komlogd.api.impulse import impulsemethod
from komlogd.api.model.orm import Datasource, Sample

@impulsemethod(uris=['cpu.system','cpu.user'], data_reqs=DataRequirements(past_delta=pd.Timedelta('1h
def summary(ts, updated, data, others):
    result={'samples':[]}
    for metric in updated:
        int_data=data[metric][ts-pd.Timedelta('60 min'):ts]
        info=str(int_data.describe())
        stats = Datasource(uri='.'.join((metric.uri,'last_60min_stats')))
        sample = Sample(metric=stats, data=info, ts=ts)
        result['samples'].append(sample)
    return result
```

En el ejemplo anterior nos suscribimos a las métricas *cpu.system* y *cpu.user* y realizamos una serie de cálculos estadísticos sobre sus datos de la última hora. Posteriormente se escriben los resultados en las métricas *cpu.system.last_60min_stats* y *cpu.user.last_60min_stats*. La función se ejecutará cada vez que se reciban datos.

A continuación la comentamos en detalle.

- **En primer lugar aplicamos a la función *summary* el decorador *@impulsemethod* con los siguientes parámetros:**
    - **uris=['cpu.system','cpu.user']**. Con este parámetro indicamos que la función se suscribe a las métricas *cpu.system* y *cpu.user*.
    - **data_reqs=DataRequirements(past_delta=pd.Timedelta('1h'))**. Aquí le indicamos que la función necesita 1 hora de datos para su correcta ejecución.
- **La función *summary* recibe una serie de parámetros (Es opcional que nuestra función reciba estos parámetros. Si no los va**
    - **ts**: objecto pandas.Timestamp. Es el timestamp de los samples que provocaron la ejecución de la función.
    - **updated**: Lista de métricas actualizadas en esta ejecución.
    - **data**: Diccionario que contiene una key por cada una de las métricas suscritas. El valor es un objeto tipo pandas.Series, con los datos de la métrica.
    - **others**: Lista con métricas a los que está suscrita la función pero que no han provocado la ejecución actual.
- En la primera línea de la función *summary* declaramos el diccionario result. Éste contiene la clave *samples* que será la que almacene los samples a enviar a Komlog.
- **A continuación, por cada una de las métricas que se han actualizado hacemos lo siguiente:**
    - Obtenemos los datos de la última hora.
    - Obtenemos el resultado al aplicarles la función *describe()* (Esta es una función del módulo pandas que obtiene una serie de valores estadísticos sobre una serie).
    - Creamos un Datasource cuya *uri* será la de la métrica + *.last_60min_stats*, es decir, crearíamos **cpu.system.last_60_min_stats** y **cpu.user.last_60_min_stats**.
    - Creamos un Sample del datasource y establecemos los datos y el timestamp.
    - Añadimos el sample al listado de samples del diccionario result.
- Por último la función devuelve el diccionario *result*, con los samples a enviar a Komlog.

Se puede aplicar el decorador *impulsemethod* a una función tantas veces como se necesite, simplemente apilando las llamadas al mismo. Por ejemplo, en la función anterior, si quisiésemos aplicar la función a dos grupos de métricas diferentes, bastaría con aplicar el decorador a la función una vez por cada grupo de métricas en lugar de crear dos funciones para la misma operación.

```
@impulsemethod(uris=['host1.cpu.system','host1.cpu.user'], data_reqs=DataRequirements(past_delta=pd.T
@impulsemethod(uris=['host2.cpu.system','host2.cpu.user'], data_reqs=DataRequirements(past_delta=pd.T
def summary(ts, updated, data, others):
    ...
```

3.3.1 Trabajando con métricas remotas

Komlog permite compartir partes del modelo de datos con otros usuarios.

Nota: Esta funcionalidad está accesible desde el menú de configuración web de Komlog. Hay que tener en cuenta que los datos **siempre se comparten en modo de sólo lectura y de forma recursiva**, es decir, si comparto la métrica *cpu.system* estaría compartiendo dicha métrica y todas sus métricas anidadas en el modelo de datos del usuario, sin importar si ya existían o no en el momento de compartirla.

Al compartir las métricas en modo solo lectura, si una *función impulso* trata de actualizar una métrica remota, dicha actualización fallará. El usuario **sólo puede modificar su modelo de datos**.

Esta funcionalidad permite la creación de aplicaciones que utilicen modelos de datos distribuidos. La forma para indicar una métrica remota es anteponer el usuario al nombre de la uri:

```
uri_remota = 'user:uri'
```

Por ejemplo, si el usuario *my_friend* nos compartiese la uri *host1.cpu*, podríamos crear una función impulso que se suscribiese a *host1.cpu.system* y *host1.cpu.user* de la siguiente forma:

```
@impulsemethod(uris=['my_friend:host1.cpu.system','my_friend:host1.cpu.user'], data_reqs=DataRequirer
def summary(ts, updated, data, others):
    ...
```

Un impulse method se puede suscribir a métricas propias y remotas a la vez:

```
uris = [
    'my_friend:host1.cpu.system',
    'my_friend:host1.cpu.user',
    'host1.cpu.system',
    'host1.cpu.user'
]

@impulsemethod(uris=uris, data_reqs=DataRequirements(past_delta=pd.Timedelta('1h')))
def summary(ts, updated, data, others):
    ...
```