Клиент для отправки метрик

Хранение метрик

Если вы разрабатываете настоящий проект, у которого есть большое количество пользователей, то необходимо наблюдать за всеми процессами, происходящими в нем. Для этого нужно смотреть за численными показателями в проекте. Показатели могут быть самыми разными - количество запросов к вашему приложению, время ответа вашего сервиса на каждый запрос, количество пользователей в сутки, и т.д. Эти всевозможные численные показатели мы будем называть метриками.

Для сбора, хранения и отображения подобных метрик существуют готовые решения, например `Graphite`, `InfluxDB`. Мы в рамках курса разработаем свою систему для сбора метрик - сервер и клиент.

В этом блоке мы начнем с разработки клиента для отправки подобных метрик на сервер, где они хранятся, и могут быть запрошены в любой момент времени. Затем в качестве финального задания в шестом блоке вам будет предложено реализовать и сам сервер.

Протокол взаимодействия

Итак, в этом блоке вам необходимо разработать сетевую программу-клиент, при помощи которой можно отправлять различные метрики на сервер. Клиент и сервер должны взаимодействовать между собой по простому текстовому протоколу через TCP сокеты. Текстовый протокол имеет главное преимущество – он наглядный – можно просмотреть диалог взаимодействия клиентской и серверной стороны без использования дополнительных инструментов.

Прежде чем реализовывать клиентское приложение давайте рассмотрим взаимодействие между клиентом и сервером на конкретных примерах.

Предположим, необходимо собирать метрики о работе операционной системы: `cpu` (загрузка процессора), `memory usage` (потребление памяти), `disk usage` (потребление места на жестком диске), `network usage` (статистика сетевых интерфейсов) и т.д. Это понадобится для контроля загрузки серверов и прогноза по расширению парка железа компании - проще говоря для мониторинга.

Пусть у нас имеется в наличии два сервера `huginn` и `muninn`. Мы будем получать загрузку центрального процессора на сервере и отправлять метрику с названием `имя_сервера.cpu`

```
client -> server: put huginn.cpu 10.6 1642667947\n
server -> client: ok\n\n
```

```
client -> server: put muninn.cpu 15.3 1642667959\n
server -> client: ok\n\n
```

Чтобы отправить метрику на сервер, вы отправляете в TCP-соединение строку вида:

```
put huginn.cpu 10.6 1642667947\n
```

Ключевое слово `put` означает команду отправки метрики. За ней через пробел следует название (имя) самой метрики, например `huginn.cpu`, далее опять через пробел значение метрики, и через еще один пробел временная метка `unix timestamp`. Таким образом, во время `1642667947` значение метрики `huginn.cpu` было равно `10.6`. Наконец, команда заканчивается символом переноса строки `\n`.

В ответ на эту команду `put` сервер присылает уведомление об успешном сохранении метрики в виде строки:

```
ok\n\n
```

Два переноса строки в данном случае означают маркер конца сообщения от сервера клиенту.

Команды

Необходимо реализовать две команды:

`put` - для сохранения метрик на сервере.

`get` - для получения метрик.

Формат команды `put` для отправки метрик — это строка вида:

```
put <key> <value> <timestamp>\n
```

Успешный ответ от сервера:

```
ok\n\n
```

Ошибка сервера:

```
error\nwrong command\n\n
```

Обратите внимание на то, что за каждым ответом сервера указано два символа `\n`. В качестве значения метрики `value` используется вещественное число.

Данные нужно не только отправлять на сервер, но и запрашивать их. Это может потребоваться для визуализации и анализа нужных метрик в определенные промежутки времени.

Формат команды `get` для получения метрик — это строка вида:

```
get <key>\n
```

В качестве ключа можно указывать символ `*`, для этого символа будут возвращены все доступные метрики. В данном задании мы никак не ограничиваем количество метрик, которые должен вернуть сервер – сервер должен возвращать все метрики, удовлетворяющие ключу.

Успешный ответ от сервера:

```
ok\nhuginn.cpu 10.5 1642667947\nmuninn.cpu 15.3 1642667959\n\n
```

Если ни одна метрика не удовлетворяет условиям поиска, то вернется ответ:

```
ok\n\n
```

Обратите внимание, что каждая успешная операция начинается с `"ok"`, а за ответом сервера всегда указано два символа `\n`.

Реализация клиента.

Необходимо реализовать класс `Client`, в котором будет инкапсулировано соединение с сервером, клиентский сокет и методы для получения и отправки метрик на сервер. В конструктор класса `Client` должна передаваться адресная пара хост и порт, а также необязательный аргумент `timeout` (`timeout=None` по умолчанию). У класса `Client` должно быть 2 метода: `put` и `get`, соответствующих протоколу выше.

Пример вызова клиента для отправки метрик и затем их получения:

In [1]:
```python
from client import Client

client = Client("127.0.0.1", 8888, timeout=15)

client.put("huginn.cpu", 0.5, timestamp=1642667947)
client.put("huginn.cpu", 2.0, timestamp=1642667948)
client.put("huginn.cpu", 0.5, timestamp=1642667948)

client.put("muninn.cpu", 3, timestamp=1642667950)
client.put("muninn.cpu", 4, timestamp=1642667951)
client.put("muninn.memory", 4200000)

print(client.get("*"))
```

```
{'huginn.cpu': [(1642667947, 0.5), (1642667948, 2.0), (1642667948, 0.5)], 'muni
nn.cpu': [(1642667950, 3.0), (1642667951, 4.0)], 'muninn.memory': [(1642748845,
4200000.0)]}
```

Клиент получает данные в текстовом виде, метод `get` должен возвращать словарь с полученными ключами с сервера. Значением ключа в словаре является список кортежей `[(timestamp, metric_value), ...]`, отсортированный по `timestamp`

от меньшего к большему. Значение timestamp должно быть преобразовано к целому числу `int`. Значение метрики `metric_value` нужно преобразовать к числу с плавающей точкой `float`.

Метод `put` принимает первым аргументом название метрики, вторым численное значение, третьим - необязательный именованный аргумент `timestamp`. Если пользователь вызвал метод `put` без аргумента `timestamp`, то клиент автоматически должен подставить текущее время в команду `put - int(time.time())`

Метод `put` не возвращает ничего в случае успешной отправки и выбрасывает исключение `ClientError` в случае неуспешной.

Метод `get` принимает первым аргументом имя метрики, значения которой мы хотим выгрузить. Также вместо имени метрики можно использовать символ `*`, о котором говорилось в описании протокола.

Метод `get` возвращает словарь с метриками (смотрите ниже пример) в случае успешного получения ответа от сервера и выбрасывает исключение `ClientError` в случае неуспешного.

Пример возвращаемого значения при успешном вызове `client.get("huginn.cpu")`:

```
{
  'huginn.cpu': [
    (1642667947, 0.5),
    (1642667948, 0.5)
  ]
}
```

Пример возвращаемого значения при успешном вызове `client.get("*")`:

```
{
  'huginn.cpu': [
    (1642667947, 0.5),
    (1642667948, 0.5)
  ],
  'muninn.cpu': [
    (1642667950, 3.0),
    (1642667951, 4.0)
  ],
  'muninn.memory': [
    (1642668557, 4200000.0)
  ]
}
```

Если в ответ на get-запрос сервер вернул положительный ответ `ok\n\n`, но без данных (то есть данных по запрашиваемому ключу нет), то метод `get` клиентадолжен вернуть пустой словарь:

In [2]:
```
print(client.get("non_existing_key"))
```

```
{}
```

Обратите внимание, что сервер хранит данные с максимальным разрешением в одну секунду. Это означает, что если в одну и ту же секунду отправить две одинаковые метрики, то будет сохранено только одно значение, которое было обработано последним. Все остальные значения будут перезаписаны.

Итак, вам необходимо предоставить модуль с классом `Client`, исключением `ClientError`. В этом классе `Client` должны быть доступны методы `get` и `put` с описанной выше сигнатурой. При вызове методов `get` и `put` клиент должен посылать сообщения в TCP-соединение с сервером в соответствии с описанным текстовым протоколом, получать ответ от сервера, преобразовывать его в удобный для использования формат, описанный выше.

Код клиента неудобно разрабатывать и отлаживать без сервера. Для удобства тестирования во время разработки кода клиента мы разработали `unittest`-ты.

test_client.py

Используйте данный `unittest` для проверки работы Вашего клиента для отправки метрик. Это ускорит процесс разработки клиента и упростит отладку.

In [16]:
```
! python -m unittest test_client.py
```

```
.....
----------------------------------------------------------------------
Ran 5 tests in 0.001s

OK
```

Успехов при выполнении задания!