

```
{
  "start": "",
  "stop": "",
  "bucket":
  data:[{"timestamp":"xxx","measurement":"mmm", "source":"xxx", フィールド:値},...]
}
```

start : 検索条件で指定した start 日時

stop : 検索条件で指定した stop 日時

bucket : 検索条件で指定した bucket

data : 検索結果のデータの配列

timestamp : point の記録日時

measurement : point の保存されている measurement

source : point の source Tag に保持されている文字列

{フィールド:値} : point に保持されている field とその値

注 : 書き込み時には point に対して複数 field を一括書き込みできるが、取り出し時には 1field ごとに timestamp, measurement, source がセットで出力される  
リスト |

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4
5  import sys
6  import json
7  import datetime
8  import influxdb_client
9  from influxdb_client.client.write_api import SYNCHRONOUS
10
11  DEBUG = False # True
12
13  # InfluxDB にアクセスするライブラリ
14
15  class InfluxDB:
16      # コンストラクタ
17      def __init__(self, url, org, token):
18          print("Constructor of InfluxDB")
19          self.url = url
20          self.organization = org
21          self.api_token = token
22          # InfluxDB へ接続するクライアント作成
23          self.client = influxdb_client.InfluxDBClient(
24              url=url,
25              token=token,
26              org=org
27          )
28          # Bucket API 呼び出し準備
29          self.bucket_api = self.client.buckets_api()
30          # 検索 API 呼び出し準備
31          self.query_api = self.client.query_api()
32          # 書き込み API 呼び出し準備
33          self.write_api = self.client.write_api(write_options=SYNCHRONOUS)
34
35      # デストラクタ
36      def __del__(self):
37          print("Destructor of InfluxDB")
38
39
40      # Bucket 作成
```

```

41 def create_bucket(self, bucket_name):
42     print("create_bucket")
43     # bucket 存在確認
44     bkt = self.bucket_api.find_bucket_by_name(bucket_name=bucket_name)
45     if DEBUG : print(bkt)
46
47     # 存在しなければ bucket 作成
48     if bkt == None:
49         print("bucket " + bucket_name + " is Not exist.")
50         self.bucket_api.create_bucket(bucket_name=bucket_name, org=self.organization)
51
52
53 # Bucket 削除
54 def delete_bucket(self, bucket_name):
55     print("delete_bucket")
56     # bucket 存在確認
57     bkt = self.bucket_api.find_bucket_by_name(bucket_name=bucket_name)
58     if DEBUG : print(bkt)
59
60     # 存在すれば bucket 削除
61     if bkt != None:
62         print("bucket " + bucket_name + " exist.")
63         self.bucket_api.delete_bucket(bkt)
64
65 def json2point(self, measurement, data_json):
66     data = json.loads(data_json)
67     if DEBUG : print(data)
68
69     # tag
70     p = influxdb_client.Point(measurement).tag("source", data['source'])
71
72     # timestamp
73     if 'timestamp' in data:
74         if data['timestamp']:
75             p.time(datetime.datetime.strptime(data['timestamp'], "%Y%m%d%H%M%S"))
76
77     # field
78     r = []
79     it = None
80     for d in data['data']:
81         if DEBUG : print(d)
82         for k,v in d.items():
83             if DEBUG : print(k,v)
84             p.field(k, v)
85
86     return p
87
88
89 # データ保存
90 """
91 データは以下の JSON 形式でもらう。timestamp は UTC で指定。
92 {
93     "timestamp":,
94     "source":,
95     "data":[]
96 }
97 """
98 def put_data(self, bucket_name, measurement, data):
99     print("put_data:", measurement, data)

```

```

100     r = self.json2point(measurement, data)
101     self.write_api.write(bucket=bucket_name, org=self.organization, record=r)
102
103
104     # データ取得
105     def get_data(self, bucket_name, measurement, start=None, stop=None, source=None):
106         print("get_data:", bucket_name, measurement, start, stop)
107         # 検索
108         """
109         start, stop は UTC で指定。
110         """
111         range = ''
112         if start != None:
113             range = '|> range(start: ' + str(int(datetime.datetime.strptime(start, "%Y%m%d%H%M%S").replace(tzinfo=datetime.timezone.utc).timestamp()))
114         else:
115             range = '|> range(start: 0'
116
117         if stop != None:
118             range = range + ', stop: ' + str(int(datetime.datetime.strptime(stop, "%Y%m%d%H%M%S").replace(tzinfo=datetime.timezone.utc).timestamp())) + ')'
119         else:
120             range = range + ', stop: now()''
121
122         query = ' from(bucket:"' + bucket_name + '" ) ' + range
123
124         if measurement:
125             query = query + '|> filter(fn:(r) => r._measurement == "' + measurement + '" )'
126
127         if source:
128             query = query + '|> filter(fn:(r) => r.source == "' + source + '" )'
129
130
131         if DEBUG : print(query)
132
133         result = self.query_api.query(org=self.organization, query=query)
134
135         if DEBUG:
136             for table in result:
137                 #print(table)
138                 for record in table.records:
139                     print(record.get_measurement())
140                     print(record.get_time())
141                     print(record.get_field())
142                     print(record.get_value())
143                     for v in record.values:
144                         print(v)
145                         print(record[v])
146
147
148         """
149         結果は、以下の形式で返す
150         {
151             "start": "",
152             "stop": "",
153             "bucket":
154             data:[{"timestamp": "xxx", "measurement": "mmm", "source": "xxx", フィールド: 値}, ...]
155         }
156
157         """
158         res = {"start": start, "stop": stop, "bucket": bucket_name}

```

```

159         dp = []
160         for table in result:
161             for record in table.records:
162                 if DEBUG: print("RECORD: ",record)
163                 d = {"timestamp":record.get_time().strftime('%Y%m%d%H%M%S'), "measurement":record.get_measurement(), "source":record['source'],
record.get_field():record.get_value()}
164                 if DEBUG : print(d)
165                 dp.append(d)
166
167             res['data'] = dp
168             if DEBUG : print(res)
169
170         return res

```

リスト 2

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4
5  import sys
6  import json
7
8  from fastapi import APIRouter
9  from pydantic import BaseModel
10 from typing import Union, List, Optional
11
12 from LIB import db
13 from LIB import influxdb
14
15 DEBUG = False # True
16
17 router = APIRouter(
18     prefix="/data",
19     tags=["data"],
20 )
21
22 # InfluxDB へのアクセス情報取得
23 s = db.DB()
24 idb_rec = s.select(db.DsInfluxDB, "InfluxDB")
25 if DEBUG : print("idb_rec=",idb_rec)
26 if idb_rec != None:
27     url = "http://" + idb_rec.address + ":" + str(idb_rec.port)
28     organization = idb_rec.organization
29     token = idb_rec.token
30 else:
31     url = "http://localhost:8086"
32     organization = "MyDataServer"
33     token = "jG99-jU4YqMZixUN3IGoBCdQ7iAbekCXPkaM2SEvHX+0IXEQKMDkJppaiXYn66bnsOuaEDZ323k l p3EYIHw-zg=="
34
35 if DEBUG:
36     print("url = ", url)
37     print("organization = ", organization)
38     print("token = ", token)
39

```

```

40 idb = influxdb.InfluxDB(url, organization, token)
41
42
43 """
44 InfluxDB
45 """
46 class DList(BaseModel):
47     timestamp: Optional[str]
48     source: str
49     data: List[dict]
50
51 @router.get("/")
52 async def get_data(channel_id: str, measurement: Union[str, None] = None, start: Union[str, None] = None, stop: Union[str, None] = None):
53     print("get_data", channel_id, measurement, start, stop)
54     """
55     データを取得する
56     """
57     res = idb.get_data(channel_id, measurement, start, stop, None)
58     if DEBUG : print(res)
59
60     return {"Status": "OK", "data":res}
61
62
63
64 @router.post("/{channel_id}/{measurement}")
65 async def put_data(channel_id: str, measurement: str, data : DList):
66     print("put_data:", channel_id, measurement, data)
67     """
68     データを登録する
69     """
70     idb.put_data(channel_id, measurement, data.json())
71     return {"Status": "OK"}
72
73

```

リスト 3

```

{
  "Status": "OK",
  "data": {
    "start": null,
    "stop": null,
    "bucket": "mych",
    "data": [
      {
        "timestamp": "20220910113015",
        "measurement": "me",
        "source": "test",
        "humidity": 40.2
      },
      {
        "timestamp": "20220910113115",
        "measurement": "me",
        "source": "test",
        "humidity": 39.2
      },
      {
        "timestamp": "20220910113015",

```

```
    "measurement": "me",
    "source": "test",
    "pressure": 1000.1
  },
  {
    "timestamp": "20220910113115",
    "measurement": "me",
    "source": "test",
    "pressure": 1000.1
  },
  {
    "timestamp": "20220910113015",
    "measurement": "me",
    "source": "test",
    "temperature": 30
  },
  {
    "timestamp": "20220910113115",
    "measurement": "me",
    "source": "test",
    "temperature": 31.5
  }
]
}
```

リスト 4

```
{
  "Status": "OK",
  "data": {
    "start": "20220910113115",
    "stop": "20220910113116",
    "bucket": "mych",
    "data": [
      {
        "timestamp": "20220910113115",
        "measurement": "me",
        "source": "test",
        "humidity": 39.2
      },
      {
        "timestamp": "20220910113115",
        "measurement": "me",
        "source": "test",
        "pressure": 1000.1
      },
      {
        "timestamp": "20220910113115",
        "measurement": "me",
        "source": "test",
        "temperature": 31.5
      }
    ]
  }
}
```

リスト 5