```
create database if not exists mydataserver;
use mydataserver;
drop table if exists ds_channel;
drop table if exists ds_influxdb;
drop table if exists ds_storage;
drop table if exists ds_ethereum;
drop table if exists ds_function;
drop table if exists ds_event;

create table ds_channel(
        id VARCHAR(100) not null,
        name VARCHAR(100),
        owner VARCHAR(100),
        primary key(id)
);
create table ds_influxdb(
        id VARCHAR(100) not null,
        name VARCHAR(100),
        address VARCHAR(100),
        port INT,
        token VARCHAR(100),
        organization VARCHAR(100),
        primary key(id)
);
create table ds_storage(
        id VARCHAR(100) not null,
        name VARCHAR(100),
        address VARCHAR(100),
        port INT,
        access_key_id VARCHAR(100),
        secret_access_key VARCHAR(100),
        primary key(id)
);
create table ds_ethereum(
        id VARCHAR(100) not null,
        name VARCHAR(100),
        address VARCHAR(100),
        port INT,
        account VARCHAR(100),
        password VARCHAR(100),
        private_key VARCHAR(100),
        primary key(id)
);
create table ds_function(
        id VARCHAR(100) not null,
        name VARCHAR(100),
        address VARCHAR(100),
        abi VARCHAR(8192),
        primary key(id)
);
create table ds_event(
        id VARCHAR(100) not null,
        name VARCHAR(100),
        url VARCHAR(1024),
        method VARCHAR(100),
        act_pre BOOLEAN,
        act_post BOOLEAN,
        func_id VARCHAR(100),
        primary key(id)
);
```

リスト1

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or ¥g.
Your MySQL connection id is 1337
Server version: 8.0.29 Homebrew

...

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mydataserver       |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.04 sec)

pmysql> use mydataserver;

mysql> desc ds_channel;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | varchar(100) | NO   | PRI | NULL    |       |
| name  | varchar(100) | YES  |     | NULL    |       |
| owner | varchar(100) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)

mysql> desc ds_influxdb;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| id          | varchar(100) | NO   | PRI | NULL    |       |
| name        | varchar(100) | YES  |     | NULL    |       |
| address     | varchar(100) | YES  |     | NULL    |       |
| port        | int         | YES  |     | NULL    |       |
| token       | varchar(100) | YES  |     | NULL    |       |
| organization | varchar(100) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
6 rows in set (0.01 sec)

mysql> desc ds_storage;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| id              | varchar(100) | NO   | PRI | NULL    |       |
| name            | varchar(100) | YES  |     | NULL    |       |
| address         | varchar(100) | YES  |     | NULL    |       |
| port            | int         | YES  |     | NULL    |       |
| access_key_id   | varchar(100) | YES  |     | NULL    |       |
| secret_access_key | varchar(100) | YES  |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)

mysql> desc ds_ethereum;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | varchar(100) | NO   | PRI | NULL    |       |
| name       | varchar(100) | YES  |     | NULL    |       |
```

```
| address     | varchar(100) | YES  |     | NULL    |       |
| port        | int          | YES  |     | NULL    |       |
| account     | varchar(100) | YES  |     | NULL    |       |
| password    | varchar(100) | YES  |     | NULL    |       |
| private_key | varchar(100) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
7 rows in set (0.01 sec)

mysql> desc ds_function;
+---------+---------------+------+-----+---------+-------+
| Field   | Type          | Null | Key | Default | Extra |
+---------+---------------+------+-----+---------+-------+
| id      | varchar(100)  | NO   | PRI | NULL    |       |
| name    | varchar(100)  | YES  |     | NULL    |       |
| address | varchar(100)  | YES  |     | NULL    |       |
| abi     | varchar(8192) | YES  |     | NULL    |       |
+---------+---------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> desc ds_event;
+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| id       | varchar(100)  | NO   | PRI | NULL    |       |
| name     | varchar(100)  | YES  |     | NULL    |       |
| url      | varchar(1024) | YES  |     | NULL    |       |
| method   | varchar(100)  | YES  |     | NULL    |       |
| act_pre  | tinyint(1)    | YES  |     | NULL    |       |
| act_post | tinyint(1)    | YES  |     | NULL    |       |
| func_id  | varchar(100)  | YES  |     | NULL    |       |
+----------+---------------+------+-----+---------+-------+
7 rows in set (0.01 sec)

mysql>
```

リスト２

```
 1  #!/usr/bin/env python
 2  # -*- coding: utf-8 -*-
 3
 4
 5  import sys
 6  import os
 7  import json
 8  import datetime
 9  import argparse
10
11  from sqlalchemy import Column
12  from sqlalchemy import ForeignKey
13  from sqlalchemy import Integer
14  from sqlalchemy import String
15  from sqlalchemy.orm import declarative_base
16  from sqlalchemy.orm import relationship
17  from sqlalchemy import create_engine
18
19  DEBUG = False # True
20
21  # 設定ファイルを読み込む
22  config_path = os.getenv('MYDATASERVER_LIB_CONFIG_PATH')
23  if config_path == None:
24      config_path = '.'
25  fp = open(config_path + '/config.json')
26  config = json.load(fp)
```

```
fp.close()
if DEBUG : print(config)

db_user = config['metadb']['USER']
db_pass = config['metadb']['PASS']
db_name = config['metadb']['DB']
db_host = config['metadb']['HOST']
db_port = config['metadb']['PORT']

# MySQL に接続
from sqlalchemy.engine.url import URL
db_url = URL.create(
    drivername="mysql",
    username=db_user,
    password=db_pass,
    host=db_host,
    port=db_port,
    database=db_name,
    query = {"charset": 'utf8'},
)

engine = create_engine(db_url)


# データモデル
Base = declarative_base()
Base.metadata.bind = engine

## channel
class DsChannel(Base):
    __tablename__ = "ds_channel"
    __table_args__ = {"autoload":True}

if DEBUG : print(DsChannel.__dict__)

## InfluxDB
class DsInfluxDB(Base):
    __tablename__ = "ds_influxdb"
    __table_args__ = {"autoload":True}

if DEBUG : print(DsInfluxDB.__dict__)


## Storage
class DsStorage(Base):
    __tablename__ = "ds_storage"
    __table_args__ = {"autoload":True}

if DEBUG : print(DsStorage.__dict__)

## Ethereum
class DsEthereum(Base):
    __tablename__ = "ds_ethereum"
    __table_args__ = {"autoload":True}

if DEBUG : print(DsEthereum.__dict__)


## function
class DsFunction(Base):
    __tablename__ = "ds_function"
    __table_args__ = {"autoload":True}

if DEBUG : print(DsFunction.__dict__)
```

```
91
92  ## event
93  class DsEvent(Base):
94      __tablename__ = "ds_event"
95      __table_args__ = {"autoload":True}
96
97  if DEBUG : print(DsEvent.__dict__)
98
99
100 class DB:
101     def __init__(self):
102         print("Constructor of DB")
103         # DB 接続
104         from sqlalchemy.orm import sessionmaker
105         SessionClass = sessionmaker(engine)
106         self.session = SessionClass()
107
108     def __del__(self):
109         print("Destructor of DB")
110         self.db_disconnect()
111
112     # DB 切断
113     def db_disconnect(self):
114         print("db_disconnect")
115         self.session.close()
116
117     # 全レコード取得
118     def list(self, tbl):
119         print("list", tbl)
120         r = self.session.query(tbl).all()
121         if DEBUG : print(r)
122         return r
123
124     # レコード追加
125     def insert(self, rec):
126         print("insert", rec)
127         try:
128             self.session.add(rec)
129             self.session.commit()
130         except Exception as e:
131             print(e)
132         finally:
133             pass
134
135     # レコード取得
136     def select(self, tbl, id):
137         print("select")
138         r   = self.session.query(tbl).filter(tbl.id == id).first()
139         if DEBUG : print(r)
140         return r
141
142     # レコード削除
143     def delete(self, rec):
144         print("delete", rec)
145         self.session.delete(rec)
146         self.session.commit()
147
148     # レコード更新
149     def update(self, rec):
150         print("update", rec)
151         self.session.commit()
152
153
```

リスト 3

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-


import sys
import json
import datetime
import influxdb_client
from influxdb_client.client.write_api import SYNCHRONOUS

DEBUG = False # True

# InfluxDB にアクセスするライブラリ

class InfluxDB:
    # コンストラクタ
    def __init__(self, url, org, token):
        print("Constructor of InfluxDB")
        self.url = url
        self.organization = org
        self.api_token = token
        # InfluxDB へ接続するクライアント作成
        self.client = influxdb_client.InfluxDBClient(
            url=url,
            token=token,
            org=org
        )
        # Bucket API 呼び出し準備
        self.bucket_api = self.client.buckets_api()
        # 検索 API 呼び出し準備
        self.query_api = self.client.query_api()
        # 書き込み API 呼び出し準備
        self.write_api = self.client.write_api(write_options=SYNCHRONOUS)

    # デストラクタ
    def __del__(self):
        print("Destructor of InfluxDB")


    # Bucket 作成
    def create_bucket(self, bucket_name):
        print("create_bucket")
        # bucket 存在確認
        bkt = self.bucket_api.find_bucket_by_name(bucket_name=bucket_name)
        if DEBUG : print(bkt)

        # 存在しなければ bucket 作成
        if bkt == None:
            print("bucket " + bucket_name + " is Not exist.")
            self.bucket_api.create_bucket(bucket_name=bucket_name, org=self.organization)


    # Bucket 削除
    def delete_bucket(self, bucket_name):
        print("delete_bucket")
        # bucket 存在確認
        bkt = self.bucket_api.find_bucket_by_name(bucket_name=bucket_name)
        if DEBUG : print(bkt)

        # 存在すれば bucket 削除
        if bkt != None:
            print("bucket " + bucket_name + " exist.")
            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４

```
1   #!/usr/bin/env python
2   # -*- coding: utf-8 -*-
3
4
5   import sys
6   import json
7   import datetime
8
9   from LIB import db
10  from LIB import influxdb
11  from LIB import storage
12
13  DEBUG = False # True
14
15  ## 作成
```

```
16  def create_channel(id, name, owner):
17      print("create_channel:" + name)
18
19      s = db.DB()
20
21      # チャネル存在確認
22      c = s.select(db.DsChannel, id)
23      print(c)
24      if c != None:
25          print("Channel " + id + " is exists.")
26          return None
27
28      # チャネル登録
29      c = db.DsChannel(id=id, name=name, owner=owner)
30      s.insert(c)
31      print(s)
32
33      # InfluxDB に Bucket を作成する
34      idb_rec = s.select(db.DsInfluxDB, "InfluxDB")
35      print("idb_rec=",idb_rec)
36      if idb_rec != None:
37          url = "http://" + idb_rec.address + ":" + str(idb_rec.port)
38          print("url = " + url)
39          idb = influxdb.InfluxDB(url, idb_rec.organization, idb_rec.token)
40          print(idb)
41          idb.create_bucket(id)
42
43      # Storage(MinIO)に Bucket を作成する
44      st_rec = s.select(db.DsStorage, "MinIO")
45      print("st_rec=",st_rec)
46      if st_rec != None:
47          url = "http://" + st_rec.address + ":" + str(st_rec.port)
48          print("url = " + url)
49          st = storage.Storage(url, st_rec.access_key_id, st_rec.secret_access_key)
50          print(st)
51          st.create_bucket(id)
52
53      return c
54
55
56  ## 削除
57  def delete_channel(id):
58      print("delete_channel:" + id)
59      s = db.DB()
60      c = s.select(db.DsChannel, id)
61      print(c)
62      if c == None:
63          print("Channel " + id + " is NOT exists.")
64          return None
65      s.delete(c)
66
67      # InfluxDB の Bucket を削除する
68      idb_rec = s.select(db.DsInfluxDB, "InfluxDB")
69      print("idb_rec=",idb_rec)
70      if idb_rec != None:
71          url = "http://" + idb_rec.address + ":" + str(idb_rec.port)
72          print("url = " + url)
73          idb = influxdb.InfluxDB(url, idb_rec.organization, idb_rec.token)
74          idb.delete_bucket(id)
75
76      # Storage(MinIO)の Bucket を削除する
77      st_rec = s.select(db.DsStorage, "MinIO")
78      print("st_rec=",st_rec)
79      if st_rec != None:
```

```
            url = "http://" + st_rec.address + ":" + str(st_rec.port)
            print("url = " + url)
            st = storage.Storage(url, st_rec.access_key_id, st_rec.secret_access_key)
            print(st)
            st.delete_bucket(id)


## 一覧
def list_channel():
    print("list_channel")
    s = db.DB()
    channel = s.list(db.DsChannel)
    for c in channel:
        print(c.name,c.id)

    return channel


## 情報取得
def get_channel(id):
    print("get_channel")
    s = db.DB()
    c = s.select(db.DsChannel, id)
    print(c)
    return c


"""
create_channel("TTT", "TT name2", "tsuyo2")
create_channel("TTT2", "TT name3", "tsuyo3")

get_channel("TTT22")

channel = list_channel()
for c in channel:
    print("L", c.id,c.name,c.owner)

change_channel_name("TTT", "UPDATED NAE")
channel = list_channel()
for c in channel:
    print("L2", c.id,c.name,c.owner)
delete_channel("TTT")
channel = list_channel()
for c in channel:
    print("L3", c.id,c.name,c.owner)

delete_channel("TTT2")
"""
```

リスト 5

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-


import sys
import json

from fastapi import APIRouter, Depends
from pydantic import BaseModel
from typing import Optional

from APP import channel

DEBUG = False # True


class Channel(BaseModel):
    id: str
    name: Optional[str] = None
    owner: Optional[str] = None

router = APIRouter(
    prefix="/channel",
    tags=["channel"],
)


@router.get("/")
async def channel_list():
    print("channel_list")

    """
    DB を検索してチャネル一覧を取得して返す、結果は id,name のリスト
    """

    c_list = channel.list_channel()

    return {"Status": "OK", "channel":c_list}

@router.post("/")
async def create_channel(ch: Channel):
    print("create_channel", ch)
    """
    ユーザを登録する
    リクエストボディで id,name,owner をもらう
    """

    res = channel.create_channel(ch.id, ch.name, ch.owner)

    return {"Status": "OK"}

@router.get("/{channel_id}", response_model=Channel)
async def get_channel(channel_id: str):
    print("get_channelt", channel_id)
    """
    channel_id でチャネルを検索して情報を返す
    """
    res = channel.get_channel(channel_id)
    c = None
    if res:
        c = Channel(id=res.id, name=res.name, owner=res.owner)

    return c
```

```
65
66
67    @router.delete("/{channel_id}")
68    async def delete_channel(channel_id: str):
69        print("delete_channel", channel_id)
70
71        res = channel.delete_channel(channel_id)
72
73        return {"Status": "OK"}
```

リスト 6

```
$ ./setinfluxdb.py --id InfluxDB --name "My InfluxDB" --token "jG99-jU4YqMZixUN3IGoBCdQ7iAbekCXPkaM2SEvHXt0lXEQKMDkJppaiXYn66bnsOuaEDZ323kIp3EYIHw-zg=="
```

リスト 7

```
$ mysql -p -u root
Enter password:

mysql> use mydataserver;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from ds_influxdb;
+----------+-------------+-----------+------+----------------------------------------------+--------------+
| id       | name        | address   | port | token                                        | organization |
+----------+-------------+-----------+------+----------------------------------------------+--------------+
| InfluxDB | My InfluxDB | l27.0.0.l | 8086 | jG99-jU4YqMZixUN3IGoBCdQ7iAbekCXPkaM2SEvHXt0lXEQKMDkJppaiXYn66bnsOuaEDZ323kIp3EYIHw-zg== | MyDataServer |
+----------+-------------+-----------+------+----------------------------------------------+--------------+
l row in set (0.08 sec)

mysql>
```

リスト 8

```
$ mysql -p -u root
Enter password:

mysql> use mydataserver;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from ds_channel;
Empty set (0.00 sec)

mysql> select * from ds_channel;
+-------+-------------------------+----------+
| id    | name                    | owner    |
+-------+-------------------------+----------+
| chanl | テストチャネル l          | tsuchiya |
+-------+-------------------------+----------+
l row in set (0.00 sec)

mysql>
```

リスト 9

```
mysql> select * from ds_channel;
+-------+-------------------------+----------+
| id    | name                    | owner    |
+-------+-------------------------+----------+
| chan1 | テストチャネル1         | tsuchiya |
| chan2 | テストチャネル2         | tsuchiya |
+-------+-------------------------+----------+
2 rows in set (0.00 sec)

mysql>
```

リスト 10

```
{
  "Status": "OK",
  "channel": [
    {
      "name": "テストチャネル1",
      "owner": "tsuchiya",
      "id": "chan1"
    },
    {
      "name": "テストチャネル2",
      "owner": "tsuchiya",
      "id": "chan2"
    }
  ]
}
```

リスト 11

```
mysql> select * from ds_channel;
+-------+-------------------------+----------+
| id    | name                    | owner    |
+-------+-------------------------+----------+
| chan2 | テストチャネル2         | tsuchiya |
+-------+-------------------------+----------+
1 row in set (0.00 sec)

mysql>
```

リスト 12