```
$ mkdir -p $HOME/CQ/MyDataServer/MinIO/DATA
$ MINIO_ROOT_USER=admin MINIO_ROOT_PASSWORD=admin001 minio server $HOME/CQ/MyDataServer/MinIO/DATA --console-address ":9001"
API: http://169.254.43.114:9000  http://127.0.0.1:9000
RootUser: admin
RootPass: admin001

Console: http://169.254.43.114:9001 http://127.0.0.1:9001
RootUser: admin
RootPass: admin001

Command-line: https://docs.min.io/docs/minio-client-quickstart-guide
   $ mc alias set myminio http://169.254.43.114:9000 admin admin001

Documentation: https://docs.min.io
```

リスト1

```
(base) tsuchiya@macbook MinIO % ./mc ls myminio
(base) tsuchiya@macbook MinIO %
```

リスト2

```
$ ./mc mb myminio/mybucket
Bucket created successfully `myminio/mybucket`.
$ MinIO % ./mc ls myminio
[2022-09-12 11:45:23 JST]     0B mybucket/
$
```

リスト3

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


import sys
import json
import datetime

import boto3

DEBUG = False # True

# Storage(MinIO)にアクセスするライブラリ

class Storage:
    # コンストラクタ
    def __init__(self, url, key_id, access_key):
        print("Constructor of Storage")
        self.url = url
        self.access_key_id = key_id
        self.secret_access_key = access_key
        # S3への接続
        self.c = boto3.resource(
            service_name = 's3',
            use_ssl=False,
            endpoint_url=url,
            aws_access_key_id=key_id,
            aws_secret_access_key=access_key)

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


    # バケット名補正
    def bucket_name(self, bucket_name):
        # 小文字で3文字以上
        if len(bucket_name) < 3:
            print("at least 3 charactors.")
            return None

        return bucket_name.lower()

```

```
44
45      # バケット作成
46      def create_bucket(self, bucket_name):
47          print("create_bucket: " + bucket_name)
48          name = self.bucket_name(bucket_name)
49          if name == None:
50              print("Invalid name")
51              return None
52          bkt = self.c.Bucket(name)
53          res = bkt.create()
54          return res
55
56
57      # バケット削除
58      def delete_bucket(self, bucket_name):
59          print("delete_bucket")
60          name = self.bucket_name(bucket_name)
61          if name == None:
62              print("Invalid name")
63              return None
64          bkt = self.c.Bucket(name)
65          res = bkt.delete()
66          return res
67
68
69      # ファイル保存
70      def put_file(self, bucket_name, file_name, file):
71          print("put_file", bucket_name, file_name)
72
73          obj = self.c.Object(bucket_name, file_name)
74          res = obj.put(Body=file)
75          return res
76
77
78      # ファイル取り出し
79      def get_file(self, bucket_name, file_name, path=None):
80          print("get_file", bucket_name, file_name)
81          try:
82              obj = self.c.Object(bucket_name, file_name)
83              if path:
84                  res = obj.download_file(path)
85                  return res
86              else:
87                  res = obj.get()
```

```
                f = res['Body'].read()
                return f
            except:
                return None

        # ファイル削除
        def delete_file(self, bucket_name, file_name):
            print("delete_file", bucket_name, file_name)
            obj = self.c.Object(bucket_name, file_name)
            res = obj.delete()
            print(res)
            return res


        # ファイル一覧
        def list_file(self, bucket_name):
            print("list_file")
            bkt = self.c.Bucket(bucket_name)
            res = []
            for obj in bkt.objects.all():
                res.append(obj.key)

            return res
```

リスト4　LIB/storage.py

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


import sys
import json
import datetime

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:
80          url = "http://" + st_rec.address + ":" + str(st_rec.port)
81          print("url = " + url)
82          st = storage.Storage(url, st_rec.access_key_id, st_rec.secret_access_key)
83          print(st)
84          st.delete_bucket(id)
85
86
87  ## 一覧
88  def list_channel():
89      print("list_channel")
90      s = db.DB()
91      channel = s.list(db.DsChannel)
92      for c in channel:
93          print(c.name,c.id)
94
95      return channel
96
97
```

```
 98  ## 情報取得
 99  def get_channel(id):
100      print("get_channel")
101      s = db.DB()
102      c = s.select(db.DsChannel, id)
103      print(c)
104      return c
105
106
107  """
108  create_channel("TTT", "TT name2", "tsuyo2")
109  create_channel("TTT2", "TT name3", "tsuyo3")
110
111  get_channel("TTT22")
112
113  channel = list_channel()
114  for c in channel:
115      print("L", c.id,c.name,c.owner)
116
117  change_channel_name("TTT", "UPDATED NAE")
118  channel = list_channel()
119  for c in channel:
120      print("L2", c.id,c.name,c.owner)
121  delete_channel("TTT")
122  channel = list_channel()
123  for c in channel:
124      print("L3", c.id,c.name,c.owner)
125
126  delete_channel("TTT2")
127  """
```

第2部第5章リスト3　チャネル管理機能(APP/chnnel.py)

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

```python

import sys
import json

from fastapi import APIRouter, File
from fastapi.responses import Response
from pydantic import BaseModel
from typing import Union, List, Optional

from LIB import db
from LIB import storage

DEBUG = False # True

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

# MinIOへのアクセス情報取得
s = db.DB()
st_rec = s.select(db.DsStorage, "MinIO")
if DEBUG : print("st_rec=",st_rec)
if st_rec != None:
    url = "http://" + st_rec.address + ":" + str(st_rec.port)
    access_key_id = st_rec.access_key_id
    secret_access_key = st_rec.secret_access_key
else:
    url = "http://localhost:9000"
    access_key_id = "admin"
    secret_access_key = "admin001"

if DEBUG:
    print("url = ", url)
    print("access_key_id = ", access_key_id)
    print("secret_access_key = ", secret_access_key)

st = storage.Storage(url, access_key_id, secret_access_key)

"""
Storage
"""
@router.get("/{channel_id}")
```

```
47  async def list_file(channel_id: str):
48      print("read_file", channel_id)
49      """
50      データを検索
51      """
52      res = st.list_file(channel_id)
53
54      return {"Status": "OK", "OBJECTS":res}
55
56  @router.get("/{channel_id}/{filename}")
57  async def get_file(channel_id: str, filename: str, need_token: bool = False, account: str = None):
58      print("get_file", channel_id, filename, need_token, account)
59      """
60      データを検索
61      """
62      d = st.get_file(channel_id, filename)
63
64      res = Response(content=d, status_code=200)
65
66      return res
67
68  @router.post("/{channel_id}/{filename}")
69  async def put_file(channel_id: str, filename: str, file: bytes = File(...)):
70      print("put_file", channel_id, filename)
71
72      res = st.put_file(channel_id, filename, file)
73
74      return {"Status": "OK", "RESULT": res}
75
76
77  @router.delete("/{channel_id}/{filename}")
78  async def delete_file(channel_id: str, filename: str):
79      print("delete_file", channel_id, filename)
80
81      res = st.delete_file(channel_id, filename)
82
83      return {"Status": "OK", "RESULT": res}
```

リスト5　routers/file_rt.py

(登録実行例)

```
(mydata_server)  % cd LIB
(mydata_server)  % ./setminio.py --id MinIO --name "Storage(MinIO)" --access_key_id admin  --secret_access_key admin001
```

(登録内容の確認)

```
mysql> select * from ds_storage;
+-------+----------------+-----------+------+---------------+-------------------+
| id    | name           | address   | port | access_key_id | secret_access_key |
+-------+----------------+-----------+------+---------------+-------------------+
| MinIO | Storage(MinIO) | 127.0.0.1 | 9000 | admin         | admin001          |
+-------+----------------+-----------+------+---------------+-------------------+
1 row in set (0.00 sec)

mysql>
```

リスト6

```
(mydata_server) % pip install boto3
Collecting boto3
  Downloading boto3-1.24.18-py3-none-any.whl (132 kB)
     |████████████████████████████████| 132 kB 1.9 MB/s
Collecting s3transfer<0.7.0,>=0.6.0
  Downloading s3transfer-0.6.0-py3-none-any.whl (79 kB)
     |████████████████████████████████| 79 kB 9.4 MB/s
Collecting botocore<1.28.0,>=1.27.18
  Downloading botocore-1.27.18-py3-none-any.whl (8.9 MB)
     |████████████████████████████████| 8.9 MB 18.7 MB/s
Collecting jmespath<2.0.0,>=0.7.1
  Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in
/Users/tsuchiya/opt/anaconda3/envs/mydata_server/lib/python3.9/site-packages (from botocore<1.28.0,>=1.27.18->boto3) (2.8.2)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in
```

```
/Users/tsuchiya/opt/anaconda3/envs/mydata_server/lib/python3.9/site-packages (from botocore<1.28.0,>=1.27.18->boto3) (1.26.9)
Requirement already satisfied: six>=1.5 in /Users/tsuchiya/opt/anaconda3/envs/mydata_server/lib/python3.9/site-packages (from
python-dateutil<3.0.0,>=2.1->botocore<1.28.0,>=1.27.18->boto3) (1.16.0)
Installing collected packages: jmespath, botocore, s3transfer, boto3
Successfully installed boto3-1.24.18 botocore-1.27.18 jmespath-1.0.1 s3transfer-0.6.0
(mydata_server) %
```

リスト7

```
$ ./mc ls myminio/
[2022-09-12 14:56:18 JST]     0B ch2/
$ ./mc ls myminio/ch2
[2022-09-12 15:02:09 JST] 1.0MiB ama.jpg
$
```

リスト8