


## 1단계(3차년도) 주요 결과물

(과제명) 대규모 분산 에너지 저장장치 인프라의 안전한 자율운영  
및 성능 평가를 위한 지능형 SW 프레임워크 개발  
(과제번호) 2021-0-00077

- 결과물명 : 하이브리드 클라우드 기반 데이터 수집·저장 시스템 통합 관리 모듈(SW)
- 작성일자 : 2023년 11월 20일

과학기술정보통신부 SW컴퓨팅산업원천기술개발사업  
“1단계(3차년도) 주요 결과물” 로 제출합니다.

수행기관	성명/직위	확인
한국전자기술연구원	최효섭/책임연구원	

정보통신기획평가원장 귀하

## 사 용 권 한

본 문서에 대한 서명은 한국전자기술연구원 내부에서 본 문서에 대하여  
수행 및 유지관리의 책임이 있음을 인정하는 것임.

본 문서는 작성, 검토, 승인하여 승인된 원본을 보관한다.

---

작성자 :	윤태일	일자 :	2023. 11. 20
-------	-----	------	--------------

---

검토자 :	김창우	일자 :	2023. 11. 21
-------	-----	------	--------------

---

승인자 :	최효섭	일자 :	2023. 11. 22
-------	-----	------	--------------

## 제 · 개정 이력

버전	변경일자	제·개정 내용	작성자
1.0	2023-11-20	최초 등록	윤태일

## 목 차

1. 개요 -----	1
2. 표준 데이터 모델 기반 대용량 에너지 데이터 수집/저장 처리 성능 최적화 -	3
3. 하이브리드 클라우드 지향 데이터 수집SW 및 저장 시스템 연계 확장 --	7
4. 에너지 저장 장치 데이터 수집·저장 운영관리 모니터링 시스템 개발 --	8

## 1. 개요

### □ 목적

- 본 명세서의 목적은 대규모 분산에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위한 지능형 안전SW 프레임워크를 개발하기 위해 하이브리드 클라우드 기반 데이터 수집·저장 시스템을 고도화하고 유지보수 및 운영관리를 위한 모니터링 시스템을 개발한다.

### □ 범위

- 본 설계서는 하이브리드 클라우드 기반 에너지 데이터 수집·저장 시스템에서 표준 데이터 모델기반 에너지 데이터 수집/저장 처리 성능을 최적화 하고 하이브리드 클라우드 지향 데이터 수집 및 저장시스템 연계확장과 에너지 저장장치 데이터 수집·저장 운영관리 모니터링 시스템 개발을 중심으로 설명한다.

### □ 시스템 개요

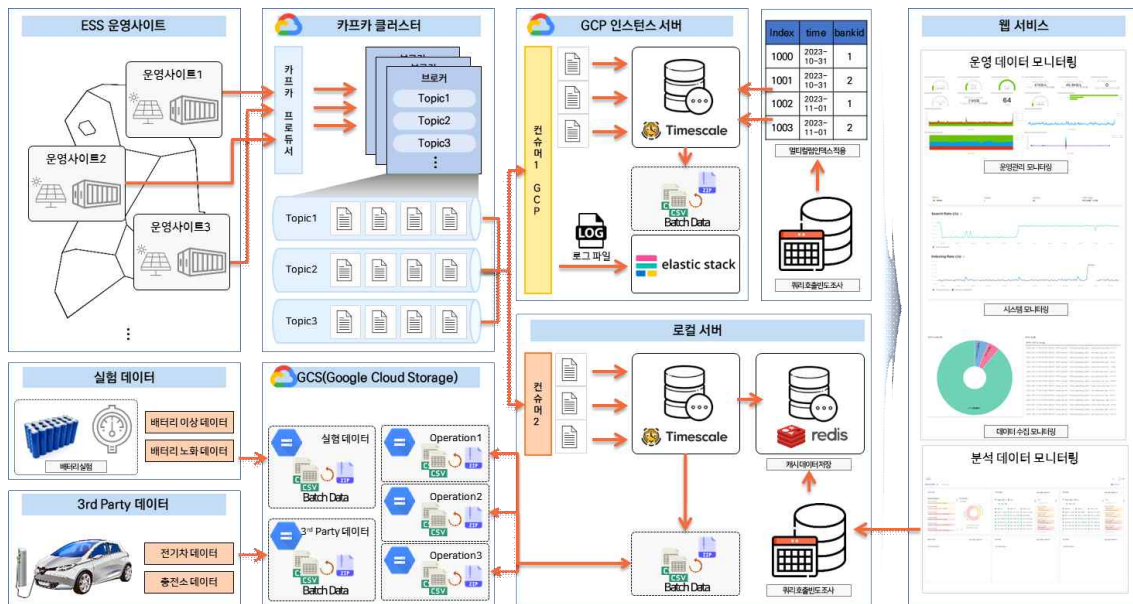


그림 1 ESS 데이터 수집 구조

- 에너지 저장장치 인프라 기반의 에너지 데이터 수집·저장 시스템은 실제 운영사이트로부터 수집되는 ESS 데이터는 Modbus TCP 프로토콜을 통해 클라우드에 구축된 데이터를 분산 처리하고 처리한 데이터를 하이브리드 클라우드(퍼블릭 클라우드와 온프레미스 클라우드)에 구축된 데이터베이스와 스토리지에 저장하며, 테스트베드에서 수집되는 데이터 또한 클라우드 스토리지에 저장하는 구조를 가진다. 데이터 처리과정에서 생성되는 로그를 통해 웹 서비스에서 데이터 상태 모니터링을 제공한다.

□ 관련 계획 및 표준

o 본 설계서는 아래 계획 및 표준을 참고한다.

구분	식별자	세부 내용	설명
설계서	ISO/IEC 9126	9126-1 (품질 모델) 9126-2 (외부 품질) 9126-3 (내부 품질) 9126-4 (사용 품질)	품질 특성 및 측정기준을 제시 소프트웨어의 기능성, 신뢰성, 사용성, 효율성, 유지보수 용이성, 이식성
	ISO/IEC 14598	14598-1 (개요) 14598-2 (계획과 관리) 14598-3 (개발자용 프로세스) 14598-4 (구매자용 프로세스) 14598-5 (평가자용 프로세스) 14598-6 (평가 모듈)	ISO 9126에 따른 제품 평가 표준: 반복성, 공정성, 객관성, 재생산성
	ISO/IEC 12119	소프트웨어 패키지 -제품설명서 -사용자문서 -프로그램과 데이터	패키지 SW 품질 요구사항 및 테스트

## 2. 표준 데이터 모델 기반 대용량 에너지 데이터 수집/저장 처리 성능 최적화

### □ 멀티컬럼 인덱스적용을 통한 데이터 수집/저장 처리 성능 최적화

- 시계열 데이터베이스에 멀티컬럼 인덱스 적용
  - 인덱스 적용 가능한 컬럼 정리
  - 쿼리 호출 빈도에 따른 멀티컬럼 인덱스 적용

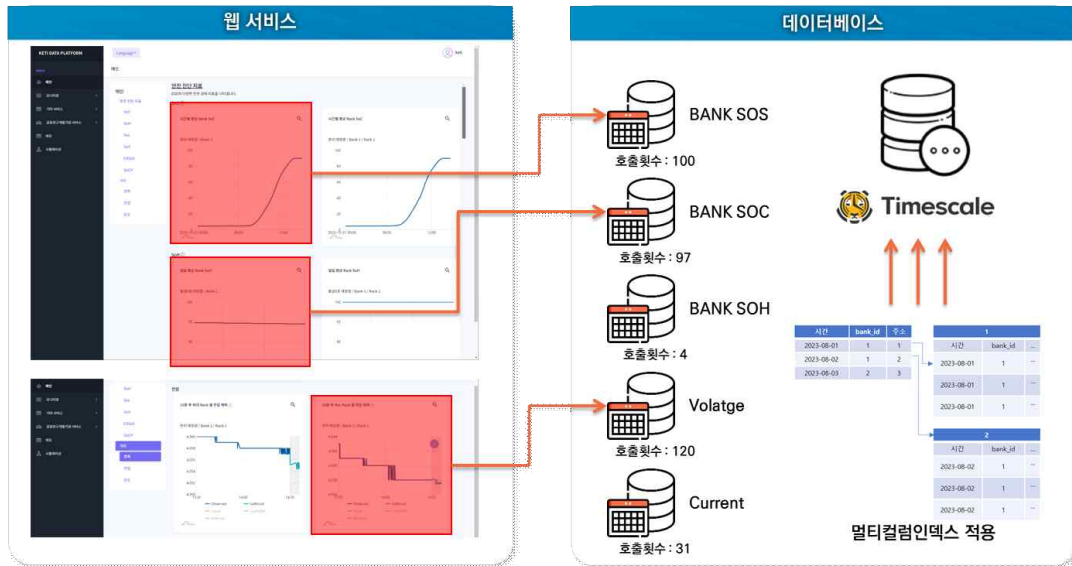


그림 2 멀티컬럼 인덱스 적용

- 멀티컬럼인덱스 적용 SW 모듈 개발
  - 멀티컬럼적용 컬럼 선택기능 개발
  - 인덱스 삭제기능 개발

### 멀티컬럼 인덱스 생성 및 삭제 모듈

```
# -*- coding:utf-8 -*-
```

```
import psycopg2
from datetime import date, datetime
from psycopg2.extensions import AsIs
from dateutil.relativedelta import relativedelta
from pytz import timezone
import pandas as pd
import argparse
import pprint
```

```
class timescale:
```

```
    # 기본 클라이언트 설정
```

```
    def __init__(self, ip, port, username, password, dbname):
```

```

try:
    self.CONNECTION = (
        """postgres://{_username}:{_password}@{_ip}:{_port}/{_dbname}""".format(
            _username=username,
            _password=password,
            _ip=ip,
            _port=port,
            _dbname=dbname,
        )
    )

    with psycopg2.connect(self.CONNECTION) as self.conn:
        self.cursor = self.conn.cursor()

        print("-----timescaledb connected-----")

except Exception as error:
    print(error)

def get_table_list(self):
    return self.query("SELECT tablename FROM pg_tables WHERE schemaname = 'public';")

def get_column_list(self, table_name):
    query_text = f"""SELECT column_name
FROM information_schema.columns
WHERE table_name = '{table_name}';"""

    return self.query(query_text)

def add_index(self, table_name, column_name):
    index_name = f"{table_name}_timestamp_{column_name.lower()}_idx"

    query_text = f"""
CREATE INDEX {index_name} ON public.{table_name} ("TIMESTAMP",{column_name});
"""

    cursor = self.conn.cursor()
    cursor.execute(query_text)

    # 커밋
    self.conn.commit()

    cursor.execute(f"SELECT * FROM pg_indexes WHERE indexname = '{index_name}'")

    result = cursor.fetchone()

    if result is not None:
        print("인덱스가 성공적으로 생성되었습니다.")
    else:
        print("CREATE INDEX 쿼리 실행에 실패했습니다.")
    cursor.close()

def drop_index(self, index_name):
    query_text = f"""

```

```

DROP INDEX IF EXISTS public.{index_name};
"""

cursor = self.conn.cursor()
cursor.execute(query_text)
# 커밋
self.conn.commit()

for notice in self.conn.notices:
    print(notice)

cursor.close()

def query(self, query_text):
    cursor = self.conn.cursor()

    cursor.execute(query_text)

    # 쿼리결과
    result = cursor.fetchall()

    cursor.close()

    df = pd.DataFrame(result, columns=[desc[0] for desc in cursor.description])

    pd.set_option("display.max_columns", None)
    pd.set_option("display.width", None)

    return df

if __name__ == "__main__":
    # 입력받을 값
    # 인덱스를 추가할 테이블
    # -> 테이블 리스트
    # 추가할 컬럼명
    # -> 컬럼명 조회
    # python 코드로 입력받고 사용

    # ArgumentParser 객체 생성
    parser = argparse.ArgumentParser()

    # 각 argument 추가

    parser.add_argument("--host", type=str, default="", help="PostgreSQL host")
    parser.add_argument("--port", type=int, default=0, help="PostgreSQL port")
    parser.add_argument("--user", type=str, default="", help="PostgreSQL user")
    parser.add_argument("--password", type=str, default="", help="PostgreSQL password")
    parser.add_argument("--db", type=str, default="", help="PostgreSQL database")

    # 입력받은 argument를 저장할 Namespace 객체 반환
    args = parser.parse_args()

    # 입력받은 argument 출력

```



```

print(args.host)
print(args.port)
print(args.user)
print(args.password)
print(args.db)

ip = args.host
port = args.port
username = args.user
password = args.password
dbname = args.db

DB = timescale(ip, port, username, password, dbname)
table_list = DB.get_table_list()
print(table_list)

print("-----")
print("")
table_name = input("table 명을 입력해주세요 : ")
print("")

# 테이블 리스트에서 입력한 테이블이름이 존재하는지 확인
if table_list.isin([table_name]).any().any():
    print("해당 table이 존재합니다.")
else:
    print("해당 table이 존재하지 않습니다.")
    exit()

print("-----")

column_list = DB.get_column_list(table_name)

print(column_list)


print("-----")
print("")
column_name = input("column 명을 입력해주세요 : ")
print("")

# 컬럼 리스트에서 입력한 컬럼이 존재하는지 확인
if column_list.isin([column_name]).any().any():
    print("해당 column이 존재합니다.")
else:
    print("해당 column이 존재하지 않습니다.")
    exit()

print("-----")

DB.add_index(table_name, column_name)

```

	하이브리드 클라우드 기반 데이터 수집·저장 시스템 통합 관리 모듈(SW)	
	프로젝트	대규모 분산 에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위한 지능형 SW 프레임워크 개발

### 3. 하이브리드 클라우드지향 데이터 수집 SW 및 저장 시스템 연계 확장

#### □ 인메모리 데이터베이스를 통한 저장 시스템 연계 확장

- 웹서비스 노출 빈도수에 따른 분석데이터 종류 정리
  - 쿼리 빈도수가 높은 분석데이터의 캐싱을 통한 성능향상을 위해 인메모리 데이터베이스에 저장할 데이터 기간별 정리 및 저장
- 분석데이터 종류에 따른 인메모리 데이터베이스 설계 및 구축
  - 로컬 클라우드에 레디스 인메모리 데이터베이스 구축
  - 기존 시계열 데이터베이스에 저장된 분석 데이터 및 웹서비스 관련 특이치 데이터를 저장할 수 있도록 레디스 설정

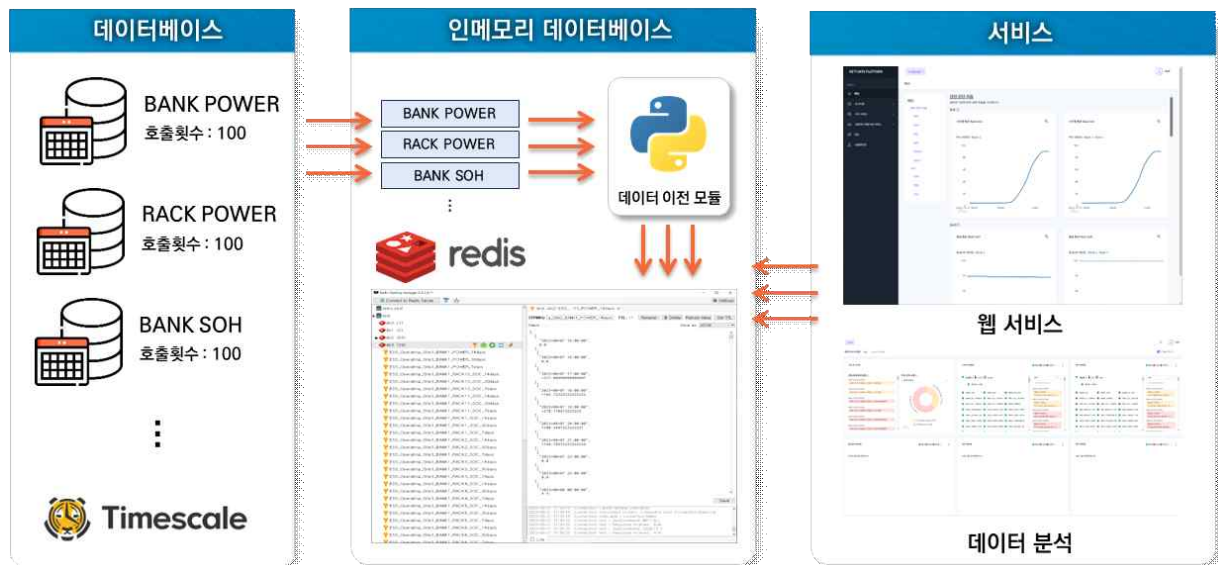


그림 3 인메모리 데이터베이스를 통한 저장 시스템 확장

#### 4. 에너지 저장 장치 데이터 수집 저장 운영관리 모니터링 시스템 개발

##### □ 데이터 수집 모니터링 시스템 개발 및 고도화

##### ○ 데이터 수집 저장 운영관리 프로그램 개발

- 클라우드 간 데이터 마이그레이션을 통해 데이터 손실률을 최소화 시키는 자동화 모듈 개발
- 로그데이터를 통한 수집 데이터 모니터링 및 트래킹 기능 개발

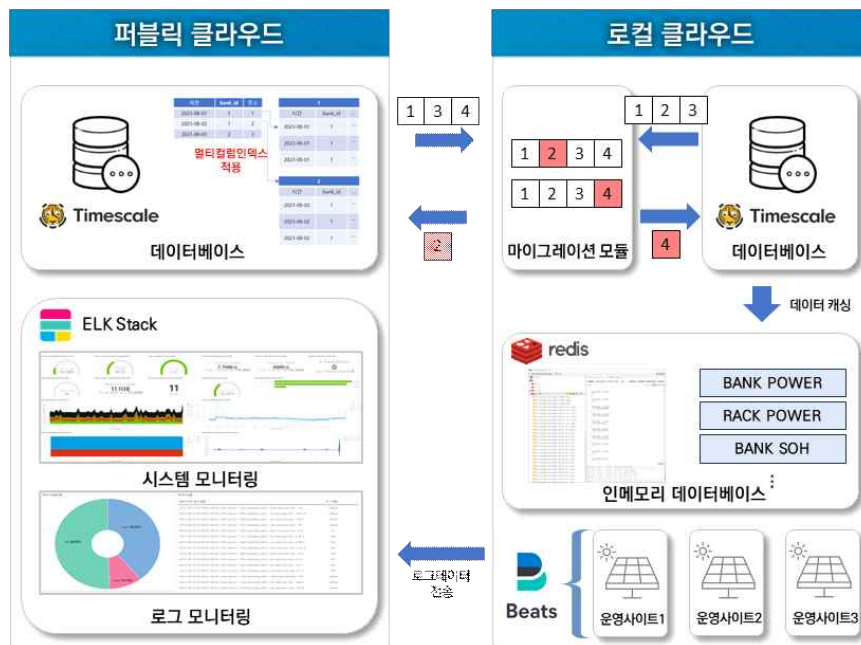


그림 4 데이터 마이그레이션 모듈 및 모니터링 시스템

클라우드 간 데이터 마이그레이션 자동화 소프트웨어
<pre># DB데이터 보관하는 코드  import psycopg2 import pandas as pd from psycopg2 import sql from psycopg2.extras import Json from datetime import datetime, timedelta import time from pprint import pprint # from psycopg2.extras import RealDictCursor from sqlalchemy import create_engine from sqlalchemy.sql.expression import false import traceback  def get_data_in_timescale(oper, mode, bank_id, rack_id=None):     """타임스케일디비에서 데이터 가져오는 코드"""      # TimescaleDB 연결 정보</pre>

```

db_host = ""
db_port = ""
db_name = oper
db_user = ""
db_password = ""

db2_host = ""
db2_port = ""
db2_name = oper
db2_user = ""
db2_password = ""

gcp_timescaledb_connection = psycopg2.connect(
    host=db_host,
    port=db_port,
    dbname=db_name,
    user=db_user,
    password=db_password,
)

local_timescaledb_connection = psycopg2.connect(
    host=db2_host,
    port=db2_port,
    dbname=db2_name,
    user=db2_user,
    password=db2_password,
)

current_time = (datetime.now().replace(hour=0,minute=0,second=0,microsecond=0)).strftime("%Y-%m-%d
%H:%M:%S")
# 1일 전의 시간 계산
one_day_ago = (datetime.now().replace(hour=0,minute=0,second=0,microsecond=0) -
timedelta(days=1)).strftime("%Y-%m-%d %H:%M:%S")

print(one_day_ago)
print(current_time)

query = ""
add_query = ""

if mode == "rack":
    add_query = f""AND "RACK_ID" = {rack_id} ""

query = f""
SET TIME ZONE 'Asia/Seoul';
SELECT * FROM {mode} WHERE "BANK_ID" = {bank_id} {add_query}AND "TIMESTAMP" BETWEEN
'{one_day_ago} Asia/Seoul' AND '{current_time} Asia/Seoul' ORDER BY "TIMESTAMP" DESC;""

# 1 뱅크기준 쿼리문
# 2 rack 기준 쿼리문

gcp_df = pd.read_sql(query, gcp_timescaledb_connection)

```

```

local_df = pd.read_sql(query, local_timescaledb_connection)

# 타임존 변환
gcp_df['TIMESTAMP'] = gcp_df['TIMESTAMP'].dt.tz_convert('Asia/Seoul')
local_df['TIMESTAMP'] = local_df['TIMESTAMP'].dt.tz_convert('Asia/Seoul')

save_to_local_df = gcp_df[~gcp_df['TIMESTAMP'].isin(local_df['TIMESTAMP'])] # << local_df에 없는 gcp_df데이터들, local_df에 저장할 데이터들
save_to_gcp_df = local_df[~local_df['TIMESTAMP'].isin(gcp_df['TIMESTAMP'])] # << gcp_df에 없는 local_df데이터들, gcp_df에 저장할 데이터들

# A 데이터프레임에 있는 TIMESTAMP가 B 데이터프레임에 없을 경우 해당 행들 추출

print(save_to_local_df)

print(save_to_gcp_df)

# with 문을 사용하여 커서 생성
with local_timescaledb_connection.cursor() as cur:
    # 데이터프레임의 각 행을 순회
    for i, row in save_to_local_df.iterrows():
        # 행을 딕셔너리로 변환
        row_dict = row.to_dict()

        # JSON 형식의 컬럼이 있다면, psycopg2.extras.Json을 사용하여 변환
        for key, value in row_dict.items():
            if isinstance(value, dict):
                row_dict[key] = Json(value)

        # 쿼리 생성
        insert = sql.SQL('INSERT INTO {} ({} ) VALUES ({}').format(
            mode,
            sql.SQL(',').join(map(sql.Identifier, row_dict.keys())),
            sql.SQL(',').join(map(sql.Placeholder, row_dict.keys()))
        )

        # 쿼리 실행
        cur.execute(insert, row_dict)

# 변경사항 커밋
local_timescaledb_connection.commit()

with gcp_timescaledb_connection.cursor() as cur:
    # 데이터프레임의 각 행을 순회
    for i, row in save_to_gcp_df.iterrows():
        # 행을 딕셔너리로 변환
        row_dict = row.to_dict()

        # JSON 형식의 컬럼이 있다면, psycopg2.extras.Json을 사용하여 변환
        for key, value in row_dict.items():
            if isinstance(value, dict):

```

```

row_dict[key] = Json(value)

# 쿼리 생성
insert = sql.SQL('INSERT INTO bank ({} VALUES ({})).format(
    sql.SQL(',').join(map(sql.Identifier, row_dict.keys())),
    sql.SQL(',').join(map(sql.Placeholder, row_dict.keys()))
)

# 쿼리 실행
cur.execute(insert, row_dict)

# 변경사항 커밋
gcp_timescaledb_connection.commit()

def main():

    OPERATION_SITE_DICT = {
        # "ESS_Operating_Site1": {1: 8},
        "ESS_Operating_Site2": {1: 9, 2: 8},
        "ESS_Operating_Site3": {1: 11},
        "ESS_Operating_Site4": {1: 9},
    }

    for site, info in OPERATION_SITE_DICT.items():
        print("site : ", site)
        num_bank = info.keys() # بانک개수 리스트로 반환

        for bank_id in num_bank:
            # 여기선 बैं크를 처리
            try:
                print("bank_id : ", bank_id)
                get_data_in_timescale(site,"bank",bank_id)
            except Exception as e:
                traceback.print_exc()
                continue # 에러나면 다음걸로
            num_rack = info[bank_id] # बैं크에 따른 랙 개수
            for rack_id in range(1, num_rack + 1):
                try:
                    print("rack_id : ", rack_id)
                    get_data_in_timescale(site,"rack",bank_id, rack_id)
                except Exception as e:
                    traceback.print_exc()
                    continue # 에러나면 다음걸로

if __name__ == "__main__":
    # 크론탭으로 동작하게하기

    main()

```

## o 경량 데이터 수집기 Beats를 통한 데이터 수집

- MetricBeats를 통한 서버 로그데이터 수집으로 서버 모니터링

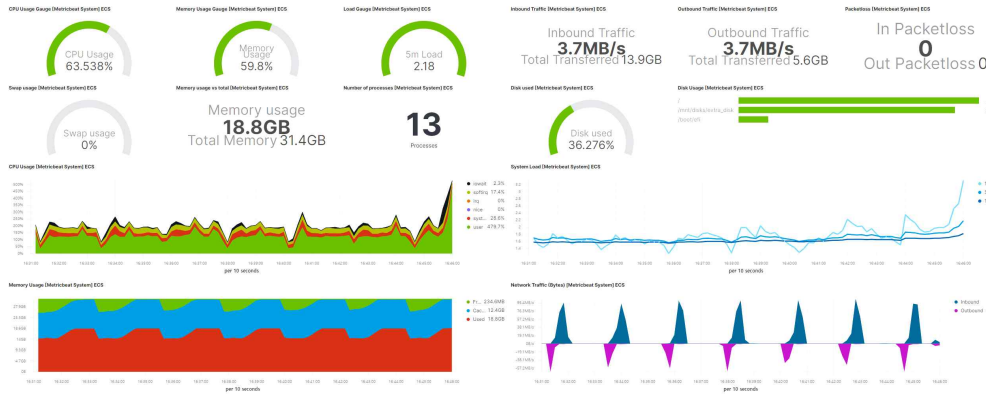


그림 5 수집한 로그 데이터를 통한 서버 모니터링

## □ 하이브리드 클라우드 기반 데이터 수집·저장 시스템 통합 관리모듈 개발

### o 클라우드별 데이터 손실률 체크를 통한 수집·저장 관리모듈개발

- 특정 기간별 수집된 데이터 개수를 기준으로 네트워크 오류 등으로 발생한 데이터 손실률 파악
- 서버 및 운영사이트 기준으로 데이터를 구분하여 각각의 손실률 파악 후 각 클라우드 서버 및 운영사이트별 손실 데이터 계산
- 손실률 로그데이터 저장 및 모니터링을 통해 수집·저장 시스템 통합관리

### 데이터 손실률 계산 모듈

```
import timescale_input_test
from pytz import timezone
import pytz
from datetime import date, datetime, timedelta
from datetime import datetime
import logs

def convert_bytes(bytes_number):
    """바이트숫자를 입력하면 단위에 맞게 변환해준다.
    Args:
        bytes_number (int): 변환할 바이트 값
    Returns:
        str: 변환된 값에 단위를 붙여서 반환
    """
    tags = ["Byte", "Kilobyte", "Megabyte", "Gigabyte", "Terabyte"]

    i = 0
    double_bytes = bytes_number

    while i < len(tags) and bytes_number >= 1024:
```

```

double_bytes = bytes_number / 1024.0
i = i + 1
bytes_number = bytes_number / 1024

return str(round(double_bytes, 2)) + " " + tags[i]

def calc_loss_rate(database, operating_site, start_time, end_time, table_name):
    if database == timescale_local:
        DB_region = "local_server1"
    elif database == timescale_GCP:
        DB_region = "GCP_server1"

    bankdivide_var = 1
    if operating_site == "ESS_Operating_Site1":
        if table_name == "rack":
            bankdivide_var = 8
    elif operating_site == "ESS_Operating_Site2":
        if table_name == "rack":
            bankdivide_var = 17
        elif table_name == "bank":
            bankdivide_var = 2
    elif operating_site == "ESS_Operating_Site3":
        if table_name == "rack":
            bankdivide_var = 11
    elif operating_site == "ESS_Operating_Site4":
        if table_name == "rack":
            bankdivide_var = 9

    query_result = database.query_data(
        """SET TIME ZONE 'Asia/Seoul';
        select count(*) from {table_name} where "TIMESTAMP" between '{start_time}' and '{end_time}';""").format(
            start_time=start_time,
            end_time=end_time,
            table_name=table_name,
        )
    )

    hypertable_size = database.query_data(
        """SET TIME ZONE 'Asia/Seoul';
        SELECT hypertable_size('{table_name}');""").format(
            table_name=table_name
        )
    )[0][0]

    hypertable_size = convert_bytes(hypertable_size)

    table_size_log_message = (
        """{DB_region} / {oper} / {table} data size : {hypertable_size}""").format(
            DB_region=DB_region,
            oper=operating_site,
            table=table_name,
            hypertable_size=hypertable_size,
        )
    )

    dataloss_logger.info(table_size_log_message)

    data_count = query_result[0][0] / bankdivide_var

```



```

if data_count > 86400:
    data_count = 86400

date_diff = end_time - start_time

total_daily_data_count = 86400 * date_diff.days

loss_rate = round(((total_daily_data_count - data_count) / total_daily_data_count * 100), 2)

loss_rate_log_message = (
    """"{start_time} / {DB_region} / {oper} / {table} data loss rate : {loss_rate} %""".format(
        start_time=start_time,
        DB_region=DB_region,
        oper=operating_site,
        table=table_name,
        loss_rate=loss_rate,
    )
)

if loss_rate > 5:
    dataloss_logger.critical(loss_rate_log_message)
    pass
elif loss_rate > 2:
    dataloss_logger.warning(loss_rate_log_message)
    pass
else:
    loss_rate_log_message = """"{start_time} / {DB_region} / {oper} / {table} data loss rate : {loss_rate}
%""".format(
        start_time=start_time,
        DB_region=DB_region,
        oper=operating_site,
        table=table_name,
        loss_rate=loss_rate,
    )
    dataloss_logger.info(loss_rate_log_message)

print(start_time, "~", end_time)
print(date_diff.days, "일간", table_name, "수집률 및 손실률")
print("collection rate :", 100 - loss_rate, "%")
print("loss rate :", loss_rate, "%")

if __name__ == "__main__":
    operating_site1 = "ESS_Operating_Site1"
    operating_site2 = "ESS_Operating_Site2"
    operating_site3 = "ESS_Operating_Site3"
    operating_site4 = "ESS_Operating_Site4"

    operating_site_list = []
    operating_site_list.append(operating_site1)
    operating_site_list.append(operating_site2)
    operating_site_list.append(operating_site3)
    operating_site_list.append(operating_site4)
    # operating_site = operating_site2

    for operating_site in operating_site_list:
        if operating_site == operating_site1:
            log_path = "/home/keti_iisrc/test/log/"
            logname = "operation1"
            logfile = logname + ".json"

```

```

elif operating_site == operating_site2:
    log_path = "/home/keti_iisrc/operation2/log/"
    logname = "operation2"
    logfile = logname + ".json"

elif operating_site == operating_site3:
    log_path = "/home/keti_iisrc/operation3/log/"
    logname = "operation3"
    logfile = logname + ".json"

elif operating_site == operating_site4:
    log_path = "/home/keti_iisrc/operation4/log/"
    logname = "operation4"
    logfile = logname + ".json"

dataloss_logger = logs.get_logger(logname, log_path, logfile)

timescale_local = timescale_input_test.timescale(
    ip="",
    port=,
    username="",
    password="",
    dbname=operating_site,
)

timescale_GCP = timescale_input_test.timescale(
    ip="",
    port=,
    username="",
    password="",
    dbname=operating_site,
)

"""1. 특정 날짜 지정"""
seoul = pytz.timezone("Asia/Seoul")
start_time = datetime(2023, 7, 5, 0, 0, 0)
start_time = seoul.localize(start_time)
end_time = datetime(2023, 7, 6, 0, 0, 0)
end_time = seoul.localize(end_time)

table_list = ["bank", "rack", "pcs", "etc"]

for table_name in table_list:
    print("local DB")
    print(operating_site)
    calc_loss_rate(timescale_local, operating_site, start_time, end_time, table_name)
    print("-----")
    print(operating_site)
    print("GCP DB")
    calc_loss_rate(timescale_GCP, operating_site, start_time, end_time, table_name)
    print("-----")

```