


1차년도 주요 결과물

(과제명) 대규모 분산 에너지 저장장치 인프라의 안전한 자율운영
및 성능 평가를 위한 지능형 SW 프레임워크 개발

(과제번호) 2021-0-00077

- 결과물명 : 퍼블릭 클라우드 기반 에너지 데이터 수집·저장 시스템(SW)
- 작성일자 : 2021년 12월 1일

과학기술정보통신부 SW컴퓨팅산업원천기술개발사업
“1차년도 주요 결과물” 로 제출합니다.

수행기관	성명/직위	확인
한국전자기술연구원	최효섭/책임연구원	

정보통신기획평가원장 귀하



퍼블릭 클라우드 기반 에너지 데이터 수집·저장 시스템(SW)

프로젝트

대규모 분산 에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위한
지능형 SW 프레임워크 개발

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 : 2021. 09. 30

검토자 : 김창우 일자 : 2021. 11. 01

승인자 : 최효섭 일자 : 2021. 12. 01

제 · 개정 이력

버전	변경일자	제·개정 내용	작성자
1.0	2021-09-30	최초 등록	윤태일

목 차

1. 개요	1
2. 에너지 저장 관리 장치 기반 표준 데이터 모델링 설계	3
3. 에너지 저장 관리 장치 기반 데이터 수집 SW 설계	4
4. 퍼블릭 클라우드 플랫폼 기반 수집 데이터 저장 시스템 설계 및 개발	5

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

1. 개요

□ 목적

- 본 명세서의 목적은 대규모 분산에너지 저장장치 인프라의 안전한 자율운영 및 성능 평가를 위한 지능형 안전SW 프레임워크를 개발하기 위해 퍼블릭 클라우드 기반 에너지 데이터 수집·저장 시스템 구축에 필요한 설계를 기록한다.

□ 범위

- 본 설계서는 퍼블릭 클라우드 기반 에너지 데이터 수집·저장 시스템에서 수집되는 데이터 모델링 설계와 수집·저장 시스템 아키텍처 설계 및 구축을 중심으로 설명한다.

□ 시스템 개요

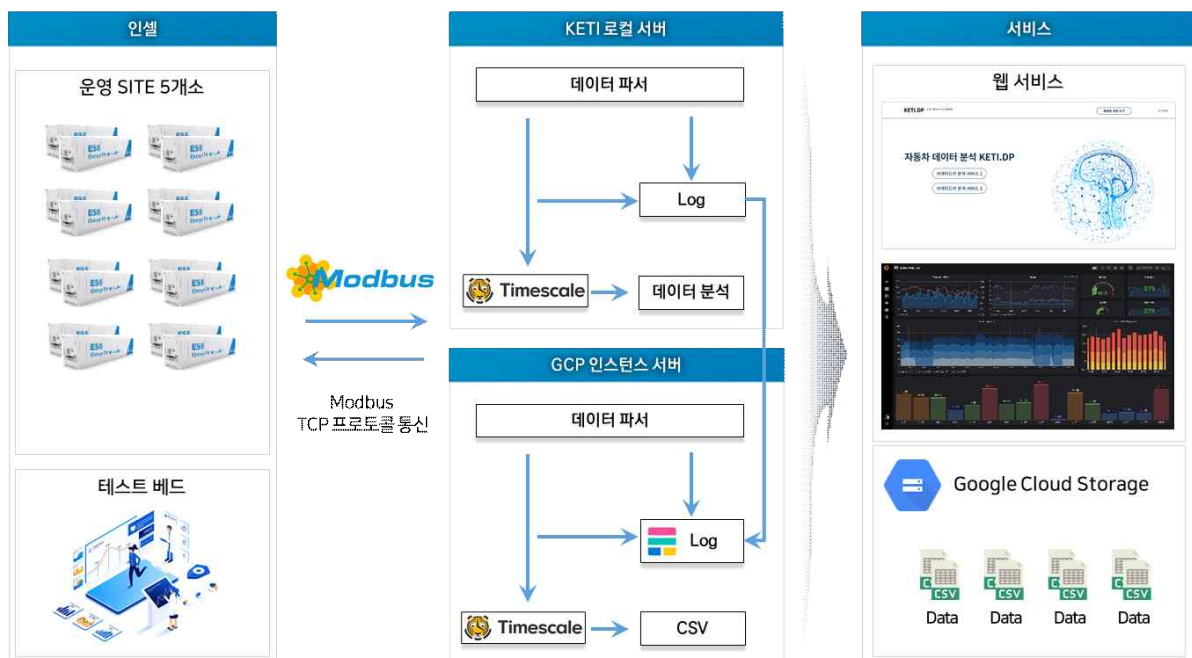


그림 1 에너지 데이터 수집 및 저장 시스템 구조도

- 퍼블릭 클라우드 기반 에너지 데이터 수집·저장 시스템은 실제 운영사이트로부터 수집되는 ESS 데이터를 Modbus TCP 프로토콜 통신을 통해 퍼블릭 클라우드에 구축된 데이터베이스와 스토리지에 저장하는 시스템이다. 수집과정에서 데이터 파싱, 전처리, 저장, 축적 등의 로그를 수집하고 수집된 로그는 웹 서비스에서 데이터 상태 모니터링에 사용된다.

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

□ 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. 에너지 저장 관리 장치 기반 표준 데이터 모델링 설계

□ 데이터 수집 구조에 따른 표준 데이터 모델링 설계 및 공통 데이터 규격 문서 작성

o 배터리 크기 및 수집 데이터별 raw 데이터 추출

- Bank, Rack, PCS, ETC 데이터 종류에 따른 데이터 구분
- 수집되는 BMS 1, 2 데이터를 가공하여 데이터 사용을 위한 DB 스키마 구조 설계

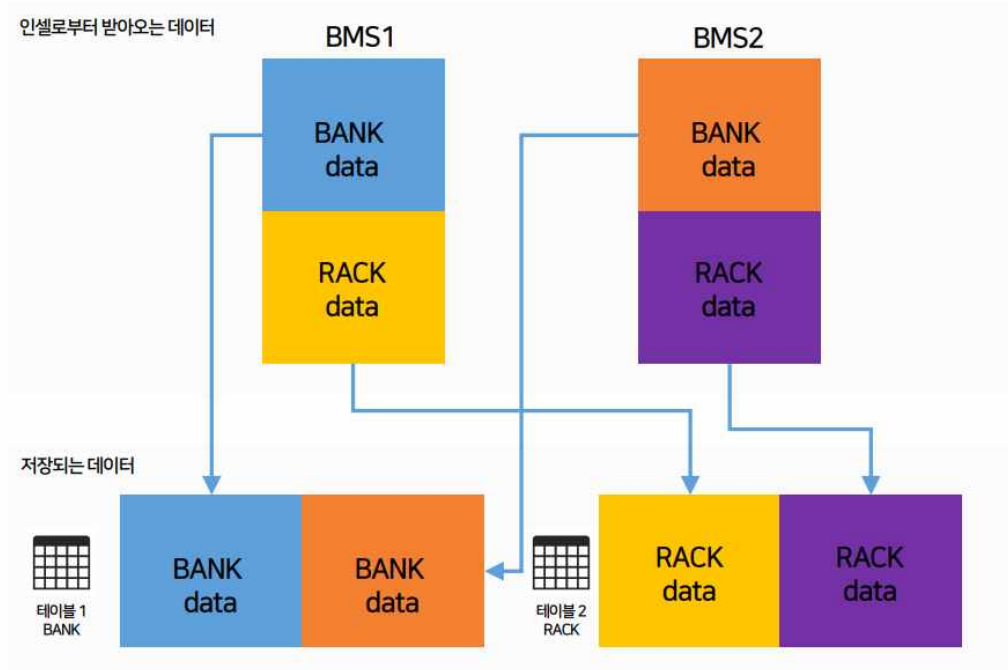



그림 2 수집되는 데이터를 가공하여 DB 스키마 설계

o 공통 데이터 규격 문서 작성 및 전처리 모듈 구조 설계

- DB 스키마 구조를 반영한 공통 데이터 규격 문서 작성
- Modbus TCP 프로토콜에 의한 비트 단위 데이터 수신 시 값 변환을 위한 Scale Factor 처리 및 음수 변환을 적용한 전처리 모듈 구조 설계
- 수집 데이터 범위에 따른 오류 판별 및 처리

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

3. 에너지 저장 관리 장치 기반 데이터수집 SW 설계

□ 표준 데이터 모델링 기반의 수집 SW 설계

- 공통 데이터 규격 문서에 따른 데이터 재가공
 - Bank, Rack, PCS, ETC 데이터 종류에 따른 데이터 구분
 - Modbus TCP 프로토콜 데이터 수신 시 비트 데이터 처리
- Timescale DB 선정 및 데이터 수집
 - 수집 데이터 별 하이퍼테이블을 통한 분산형 데이터베이스 구조 설계
 - 시계열 RDBMS 구조로 간편한 쿼리 및 빠른 속도
- Log 데이터 수집
 - Elastic Stack 및 File beat를 통한 수집, 주장, 축적 Log 데이터 수집
 - kibana 모니터링 대시보드를 통한 Log 모니터링

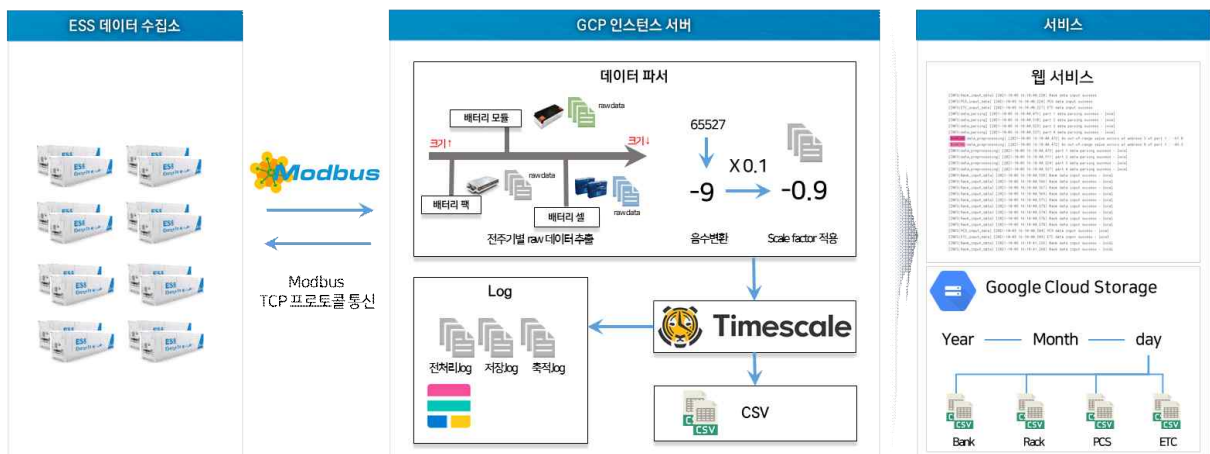


그림 3 데이터 수집 및 전처리 SW 설계

4. 퍼블릭 클라우드 플랫폼 기반 수집 데이터 저장 시스템 설계 및 개발

□ 데이터 저장 및 축적 구조 설계 및 개발

o 퍼블릭 클라우드에 메인DB 설계 및 구축

- GCP(Google Cloud Platform) 인스턴스에 PostgreSQL 기반 TimescaleDB 구축

PostgreSQL 기반 TimescaleDB 구축
<pre># Docker 설치 \$ sudo apt-get update \$ sudo apt-get install docker-ce docker-ce-cli containerd.io # TimescaleDB 이미지 다운로드 \$ docker pull # TimescaleDB 이미지를 실행하여 컨테이너 작동 \$ docker run -d --shm-size=8G --name timescaledb -p 5432:5432 -e POSTGRES_PASSWORD=keti1234! -v timescale_volume:/var/lib/postgresql/data timescale/timescaledb-postgis:latest-pg12</pre>

- Docker Container를 활용하여 높은 이식성 및 볼륨 위치 변경을 통한 데이터 안전성 확보

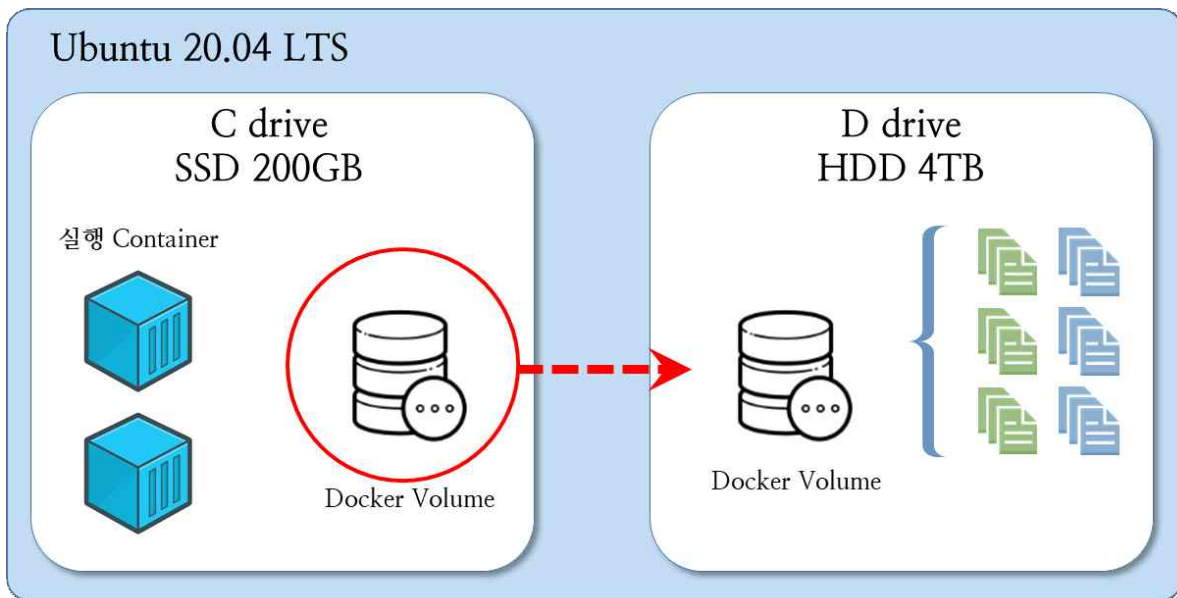


그림 4 도커 데이터 저장 위치변경을 통한 저장공간 확보

우분투 추가 디스크 마운트 및 Docker 볼륨 위치 변경
<pre># 디스크 정보 확인 후 포맷하고 마운트하려는 디스크 찾기 \$ sudo lsblk # mkfs로 디스크 포맷 \$ sudo mkfs.ext4 -m 0 -E lazy_itable_init=0,lazy_journal_init=0,discard /dev/<디스크이름></pre>


```
# 디스크 마운트
$ sudo mkdir -p /mnt/disks/<마운트지점디렉토리이름>

$ sudo mount -o discard,defaults /dev/<디스크이름> /mnt/disks/<마운트지점디렉토리이름>

$ sudo chmod a+w /mnt/disks/<마운트지점디렉토리이름>

# Docker 서비스 중지
$ service docker stop

# 이동할 디렉토리 생성
$ mkdir -p /data/docker

# 도커 옵션 변경
# /etc/docker/daemon.json
...
"data-root": "/data/docker",
...

# 다음 명령어로 루트 디렉토리 변경 확인
$ docker info | grep "Docker Root Dir"
```

- 수집 데이터 별 하이퍼 테이블 생성을 통한 분산형 데이터베이스 구조 구축

하이퍼 테이블 생성 코드

```
class timescale:

    # 기본 클라이언트 설정
    def __init__(self):
        # timescale DB 연결
        self.CONNECTION = (
            """postgres://postgres:keti1234!@34.64.147.133:5432/ESS_Operating_Site1"""
            # """postgres://postgres:keti1234!@1.214.41.250:5434/ESS_Operating_Site1"""
        )
        with psycopg2.connect(self.CONNECTION) as self.conn:
            self.cursor = self.conn.cursor()

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

    # 테이블 생성
    def create_hypertable(self, table_name):

        # 하이퍼 테이블 생성 SQL 문
        # Not Null 은 문자 그대로 비어있으면 안됨

        if "bank" in table_name:

            # Bank 생성 SQL문
            query_create_sensordata_table = """CREATE TABLE "{tablename}" (
```

	"TIMESTAMP" timestampz NOT NULL,
	"BANK_ID" int4 NOT NULL,
	"BANK_SOC" float8 NULL,
	"BANK_SOH" float8 NULL,
	"BANK_DC_VOLT" float8 NULL,
	"BANK_DC_CURRENT" float8 NULL,
	"MAX_CELL_VOLTAGE_OF_BANK" float8 NULL,
	"MIN_CELL_VOLTAGE_OF_BANK" float8 NULL,
	"MAX_CELL_TEMPERATURE_OF_BANK" float8 NULL,
	"MIN_CELL_TEMPERATURE_OF_BANK" float8 NULL,
	"BANK_POWER" float8 NULL,
	"RACK_TEMPERATURE_IMBALANCE_WARNING" int4 NULL,
	"RACK_UNDER_TEMPERATURE_WARNING" int4 NULL,
	"RACK_OVER_TEMPERATURE_WARNING" int4 NULL,
	"RACK_VOLTAGE_IMBALANCE_WARNING" int4 NULL,
	"RACK_UNDER_VOLTAGE_PROTECTION_WARNING" int4
NULL,	
	"RACK_OVER_VOLTAGE_PROTECTION_WARNING" int4 NULL,
	"RACK_OVER_CURRENT_CHARGE_WARNING" int4 NULL,
	"RACK_OVER_CURRENT_DISCHARGE_WARNING" int4 NULL,
	"RACK_TEMPERATURE_IMBALANCE_FAULT" int4 NULL,
	"RACK_UNDER_TEMPERATURE_FAULT" int4 NULL,
	"RACK_OVER_TEMPERATURE_FAULT" int4 NULL,
	"RACK_VOLTAGE_IMBALANCE_FAULT" int4 NULL,
	"RACK_UNDER_VOLTAGE_PROTECTION_FAULT" int4 NULL,
	"RACK_OVER_VOLTAGE_PROTECTION_FAULT" int4 NULL,
	"RACK_OVER_CURRENT_CHARGE_FAULT" int4 NULL,
	"RACK_OVER_CURRENT_DISCHARGE_FAULT" int4 NULL,
	"RACK_CHARGE_RELAY_PLUS_FAULT_STATUS" int4 NULL,
	"RACK_DISCHARGE_RELAY_MINUS_FAULT_STATUS" int4
NULL,	
	"RACK_FUSE_MINUS_FAULT_STATUS" int4 NULL,
	"RACK_FUSE_PLUS_FAULT_STATUS" int4 NULL,
	"RACK_TRAY_RACK_COMMUNICATION_FAULT" int4 NULL,
	"RACK_N001_MASTER_RACK_COMMUNICATION_FAULT" int4
NULL,	
	"RACK_N002_MASTER_RACK_COMMUNICATION_FAULT" int4
NULL,	
	"RACK_N003_MASTER_RACK_COMMUNICATION_FAULT" int4
NULL,	
	"RACK_N004_MASTER_RACK_COMMUNICATION_FAULT" int4
NULL,	
	"RACK_N005_MASTER_RACK_COMMUNICATION_FAULT" int4
NULL,	
	"RACK_N006_MASTER_RACK_COMMUNICATION_FAULT" int4
NULL,	
	"RACK_N007_MASTER_RACK_COMMUNICATION_FAULT" int4
NULL,	
	"RACK_N008_MASTER_RACK_COMMUNICATION_FAULT" int4
NULL,	
	"BATTERY_STATUS_FOR_RUN" int4 NULL,
	"BATTERY_STATUS_FOR_CHARGE" int4 NULL,
	"BATTERY_STATUS_FOR_FAULT" int4 NULL,
	"BATTERY_STATUS_FOR_WARNING" int4 NULL,

```

"MASTER_RACK_COMMUNICATION_FAULT" int4 NULL
);""".format(
    _tablename=table_name
)

# FOREIGN KEY (Bank_ID) REFERENCES RACK (Bank_ID)

query_create_sensordata_hypertable = (
    """SELECT create_hypertable('{_tablename}', 'TIMESTAMP');""".format(
        _tablename=table_name
    )
)

# self.cursor.execute(query_create_sensordata_table)

self.cursor.execute(query_create_sensordata_hypertable)
# commit changes to the database to make changes persistent
self.conn.commit()
self.cursor.close()

if "rack" in table_name:
    # Rack 생성 SQL문
    query_create_sensordata_table = """CREATE TABLE IF NOT EXISTS "{_tablename}" (
        "TIMESTAMP" timestamptz NOT NULL,
        "BANK_ID" int4 NOT NULL,
        "RACK_ID" int4 NOT NULL,
        "RACK_SOC" float8 NULL,
        "RACK_SOH" float8 NULL,
        "RACK_VOLTAGE" float8 NULL,
        "RACK_CURRENT" float8 NULL,
        "RACK_MAX_CELL_VOLTAGE" float8 NULL,
        "RACK_MAX_CELL_VOLTAGE_POSITION" float8 NULL,
        "RACK_MIN_CELL_VOLTAGE" float8 NULL,
        "RACK_MIN_CELL_VOLTAGE_POSITION" float8 NULL,
        "RACK_CELL_VOLTAGE_GAP" float8 NULL,
        "RACK_CELL_VOLTAGE_AVERAGE" float8 NULL,
        "RACK_MAX_CELL_TEMPERATURE" float8 NULL,
        "RACK_MAX_CELL_TEMPERATURE_POSITION" float8 NULL,
        "RACK_MIN_CELL_TEMPERATURE" float8 NULL,
        "RACK_MIN_CELL_TEMPERATURE_POSITION" float8 NULL,
        "RACK_CELL_TEMPERATURE_GAP" float8 NULL,
        "RACK_CELL_TEMPERATURE_AVERAGE" float8 NULL,
        "RACK_DISCHARGE_RELAY_MINUS_STATUS" int4 NULL,
        "RACK_CHARGE_RELAY_PLUS_STATUS" int4 NULL,
        "RACK_CELL_BALANCE_STATUS" int4 NULL,
        "RACK_CURRENT_SENSOR_DISCHARGE" int4 NULL,
        "RACK_CURRENT_SENSOR_CHARGE" int4 NULL,
        "RACK_STATUS_FOR_RUN" int4 NULL,
        "RACK_STATUS_FOR_FAULT" int4 NULL,
        "RACK_STATUS_FOR_WARNING" int4 NULL,
        "RACK_RACK_TEMPERATURE_IMBALANCE_WARNING" int4 NULL,
        "RACK_RACK_UNDER_TEMPERATURE_WARNING" int4 NULL,
        "RACK_RACK_OVER_TEMPERATURE_WARNING" int4 NULL,
        "RACK_RACK_VOLTAGE_IMBALANCE_WARNING" int4 NULL,
    )

```

```

NULL,
                                "RACK_RACK_UNDER_VOLTAGE_PROTECTION_WARNING" int4
NULL,
                                "RACK_RACK_OVER_VOLTAGE_PROTECTION_WARNING" int4
NULL,
                                "RACK_OVER_CURRENT_CHARGE_WARNING" int4 NULL,
                                "RACK_OVER_CURRENT_DISCHARGE_WARNING" int4 NULL,
                                "RACK_RACK_TEMPERATURE_IMBALANCE_FAULT" int4 NULL,
                                "RACK_RACK_UNDER_TEMPERATURE_FAULT" int4 NULL,
                                "RACK_RACK_OVER_TEMPERATURE_FAULT" int4 NULL,
                                "RACK_RACK_VOLTAGE_IMBALANCE_FAULT" int4 NULL,
                                "RACK_RACK_UNDER_VOLTAGE_PROTECTION_FAULT" int4 NULL,
                                "RACK_RACK_OVER_VOLTAGE_PROTECTION_FAULT" int4 NULL,
                                "RACK_OVER_CURRENT_CHARGE_FAULT" int4 NULL,
                                "RACK_OVER_CURRENT_DISCHARGE_FAULT" int4 NULL,
                                "RACK_CHARGE_RELAY_PLUS_FAULT_STATUS" int4 NULL,
                                "RACK_DISCHARGE_RELAY_MINUS_FAULT_STATUS" int4 NULL,
                                "RACK_FUSE_MINUS_FAULT_STATUS" int4 NULL,
                                "RACK_FUSE_PLUS_FAULT_STATUS" int4 NULL,
                                "RACK_TRAY_RACK_COMMUNICATION_FAULT" int4 NULL
                                );""".format(
                                _tablename=table_name
                                )
                                # FOREIGN KEY (Bank_ID) REFERENCES BANK (Bank_ID)

                                query_create_sensordata_hypertable = """SELECT create_hypertable('{_tablename}', 'TIMESTAMP',
if_not_exists => TRUE);""".format(
                                _tablename=table_name
                                )

                                # self.cursor.execute(query_create_sensordata_table)
                                self.cursor.execute(query_create_sensordata_hypertable)
                                # commit changes to the database to make changes persistent
                                self.conn.commit()
                                self.cursor.close()

                                if "pcs" in table_name:

                                # PCS 생성 SQL문
                                query_create_sensordata_table = """CREATE TABLE IF NOT EXISTS "{_tablename}" (
                                "TIMESTAMP" timestamptz NOT NULL,
                                "BANK_ID" int4 NOT NULL,
                                "PCS_AC_L1_THD" float8 NULL,
                                "PCS_AC_L2_THD" float8 NULL,
                                "PCS_AC_L3_THD" float8 NULL,
                                "SYSTEM_TEMPERATURE_MAX" float8 NULL,
                                "DC_VOLTAGE_1" float8 NULL,
                                "DC_VOLTAGE_2" float8 NULL,
                                "DC_CURRENT" float8 NULL,
                                "DC_POWER" float8 NULL,
                                "AC_FREQUENCY" float8 NULL,
                                "AC_VOLTAGE" float8 NULL,
                                "AC_CURRENT_LOW" float8 NULL,
                                "AC_CURRENT_HIGH" float8 NULL,
                                "AC_POWER" float8 NULL,

```

```
"AC_ACTIVE_POWER" float8 NULL,
"AC_REACTIVE_POWER" float8 NULL,
"AC_POWER_FACTOR" float8 NULL,
"AC_L1_VOLTAGE" float8 NULL,
"AC_L1_CURRENT" float8 NULL,
"AC_L2_VOLTAGE" float8 NULL,
"AC_L2_CURRENT" float8 NULL,
"AC_L3_VOLTAGE" float8 NULL,
"AC_L3_CURRENT" float8 NULL,
"AC_ACCUMULATED_DISCHARGE_ENERGY_LOW" float8 NULL,
"AC_ACCUMULATED_DISCHARGE_ENERGY_HIGH" float8 NULL,
"AC_ACCUMULATED_CHARGE_ENERGY_LOW" float8 NULL,
"AC_ACCUMULATED_CHARGE_ENERGY_HIGH" float8 NULL,
"PCS_STATUS_0" float8 NULL,
"PCS_STATUS_1" float8 NULL,
"PCS_STATUS_2" float8 NULL,
"PCS_STATUS_3" float8 NULL,
"PCS_STATUS_4" float8 NULL,
"PCS_STATUS_5" float8 NULL,
"PCS_STATUS_6" float8 NULL,
"PCS_STATUS_7" float8 NULL,
"PCS_STATUS_8" float8 NULL,
"PCS_STATUS_9" float8 NULL,
"PCS_STATUS_10" float8 NULL,
"PCS_STATUS_11" float8 NULL,
"PCS_STATUS_14" float8 NULL,
"PCS_STATUS_15" float8 NULL,
"PCS_PROTECTION_1_0" float8 NULL,
"PCS_PROTECTION_1_1" float8 NULL,
"PCS_PROTECTION_1_2" float8 NULL,
"PCS_PROTECTION_1_3" float8 NULL,
"PCS_PROTECTION_1_4" float8 NULL,
"PCS_PROTECTION_1_5" float8 NULL,
"PCS_PROTECTION_1_6" float8 NULL,
"PCS_PROTECTION_1_7" float8 NULL,
"PCS_PROTECTION_1_8" float8 NULL,
"PCS_PROTECTION_1_9" float8 NULL,
"PCS_PROTECTION_1_10" float8 NULL,
"PCS_PROTECTION_1_11" float8 NULL,
"PCS_PROTECTION_1_12" float8 NULL,
"PCS_PROTECTION_1_13" float8 NULL,
"PCS_PROTECTION_1_14" float8 NULL,
"PCS_PROTECTION_1_15" float8 NULL,
"PCS_PROTECTION_2_0" float8 NULL,
"PCS_PROTECTION_2_1" float8 NULL,
"PCS_PROTECTION_2_2" float8 NULL,
"PCS_PROTECTION_2_3" float8 NULL,
"PCS_PROTECTION_2_4" float8 NULL,
"PCS_PROTECTION_2_5" float8 NULL,
"PCS_PROTECTION_2_6" float8 NULL,
"PCS_PROTECTION_2_7" float8 NULL,
"PCS_PROTECTION_2_8" float8 NULL,
"PCS_PROTECTION_2_9" float8 NULL,
"PCS_PROTECTION_2_10" float8 NULL,
```

```

"PCS_PROTECTION_2_11" float8 NULL,
"PCS_PROTECTION_2_12" float8 NULL,
"PCS_PROTECTION_2_13" float8 NULL,
"PCS_PROTECTION_2_14" float8 NULL,
"PCS_PROTECTION_2_15" float8 NULL,
"PCS_PROTECTION_3_0" float8 NULL,
"PCS_PROTECTION_3_1" float8 NULL,
"PCS_PROTECTION_3_2" float8 NULL,
"PCS_PROTECTION_3_3" float8 NULL,
"PCS_PROTECTION_3_4" float8 NULL,
"PCS_PROTECTION_3_5" float8 NULL,
"PCS_PROTECTION_3_6" float8 NULL,
"PCS_PROTECTION_3_7" float8 NULL,
"PCS_PROTECTION_3_8" float8 NULL,
"PCS_PROTECTION_3_9" float8 NULL,
"PCS_PROTECTION_3_10" float8 NULL,
"PCS_CONTROL_0" float8 NULL,
"SET_POWER" float8 NULL,
"SET_SOC_LOW_TRIP_LEVEL" float8 NULL,
"SET_SOC_HIGH_TRIP_LEVEL" float8 NULL,
"SET_VBAT_LOW_TRIP_LEVEL" float8 NULL,
"SET_VBAT_HIGH_TRIP_LEVEL" float8 NULL
);"""
format(
    _tablename=table_name
)

# FOREIGN KEY (Bank_ID) REFERENCES RACK (Bank_ID)

query_create_sensordata_hypertable = """SELECT create_hypertable('{_tablename}', 'TIMESTAMP',
if_not_exists => TRUE);"""
format(
    _tablename=table_name
)

# self.cursor.execute(query_create_sensordata_table)
self.cursor.execute(query_create_sensordata_hypertable)
# commit changes to the database to make changes persistent
self.conn.commit()
self.cursor.close()

if "etc" in table_name:

    # 생성 ETC문
    query_create_sensordata_table = """CREATE TABLE IF NOT EXISTS "{_tablename}" (
        "TIMESTAMP" timestampz NOT NULL,
        "BANK_ID" int4 NOT NULL,
        "SENSOR1_TEMPERATURE" float8 NULL,
        "SENSOR1_HUMIDITY" float8 NULL,
        "INV1_ACTIVE_POWER" float8 NULL,
        "INV1_CUMLUATIVE_POWER_GENERATION" float8 NULL
    );"""
    format(
        _tablename=table_name
    )

# FOREIGN KEY (Bank_ID) REFERENCES RACK (Bank_ID)

```

```

        query_create_sensordata_hypertable = """SELECT create_hypertable('{tablename}', 'TIMESTAMP',
if_not_exists => TRUE);"""
        .format(
            _tablename=table_name
        )

        # self.cursor.execute(query_create_sensordata_table)
        self.cursor.execute(query_create_sensordata_hypertable)
        # commit changes to the database to make changes persistent
        self.conn.commit()
        self.cursor.close()

    print("---- Table Created ----")

```

o 실시간 데이터 수집

- 멀티 스레드(multi thread) 구조를 통해 매 초 데이터 수집 및 저장

멀티 스레드 구조를 통한 데이터 수집 코드

```

class ESS_Modbus:

    # 기본 클라이언트 설정
    def __init__(self):
        self.client = ModbusClient("121.178.23.187", 40001, unit_id=1)
        self.client.open()

    # 클라이언트 세팅
    def client_set(self, IP, PORT, ID):
        self.client = ModbusClient(IP, PORT, unit_id=ID)

    # 저장 데이터 만들기
    def storaging_data_make(self):

        pass

    # 단순 데이터 파싱
    def data_parsing(self, partID):

        try:
            # 파싱로그
            parsing_logger = logs.get_logger(
                "log1", "./Data_Ingestion/log/", "parsing.logs"
            )

            # 인풋레지스터의 경우 최대 125개밖에 못가져오기때문에 BMS데이터의 경우 수정이 필요함
            if partID == 1: # BMS1
                start_address = 0
                parsingdata_num = 137

```

```

elif partID == 2: # BMS2
    start_address = 200
    parsingdata_num = 265
elif partID == 3: # BMS2
    start_address = 500
    parsingdata_num = 89
elif partID == 4: # BMS2
    start_address = 600
    parsingdata_num = 5

# 파싱데이터 리스트
data = []

# 데이터 파싱

data = self.client.read_input_registers(start_address, parsingdata_num)
# ETC의 경우 604번의 값은 쓰레기 데이터
if partID == 4:
    data.pop(3)

# BMS1, 2의 경우 데이터 파싱 개수를 초과하기때문에 나눠서 처리
elif partID == 1:
    data1 = self.client.read_input_registers(start_address, 125)
    data2 = self.client.read_input_registers(125, 12)
    data = data1 + data2

elif partID == 2:
    data1 = self.client.read_input_registers(start_address, 125)
    data2 = self.client.read_input_registers(325, 125)
    data3 = self.client.read_input_registers(450, 16)
    data = data1 + data2 + data3

parsing_logger.info("part %s data parsing success", partID)

except Exception as e:
    parsing_logger.error("part %s data parsing error : ", partID, e)

print("잘못된값", data[12])

# 데이터 전처리
result = self.data_preprocessing(partID, data)

# बैं크 아이디 추가
if partID == 3 or partID == 4:
    BANK_ID = 1
    data.insert(0, BANK_ID)
# dtype("uint16")
# for i in range(len(result)):
#     print("registernum :", start_address + i, "value :", result[i])
# print("type :", type(result[i]))

# for i in range(len(result)):
#     print(i+200, result[i])

```



```

self.client.close()

return list(result)

# 데이터 전처리 (스케일팩터, 범위 확인)
def data_preprocessing(self, partID, data):

    preprocessing_logger = logs.get_logger(
        "log2", "./Data_Ingestion/log/", "preprocessing.logs"
    )

    try:
        if partID == 1: # BMS1
            scale_factor_file_path =
"C:/Users/taeil/Documents/GitHub/ESS/Data_Ingestion/preprocessing_filter/scalefactor_BMS1.txt"
            value_range_file_path =
"C:/Users/taeil/Documents/GitHub/ESS/Data_Ingestion/preprocessing_filter/range_BMS1.txt"
            elif partID == 2: # BMS2 -> boolean
                scale_factor_file_path =
"C:/Users/taeil/Documents/GitHub/ESS/Data_Ingestion/preprocessing_filter/scalefactor_BMS2.txt"
                # BMS2는 0 1값밖에 없기 때문에 인트형으로 변경
                for i in range(len(data)):
                    data[i] = int(data[i])

            elif partID == 3: # PCS
                scale_factor_file_path =
"C:/Users/taeil/Documents/GitHub/ESS/Data_Ingestion/preprocessing_filter/scalefactor_PCS.txt"
            elif partID == 4: # ETC
                scale_factor_file_path =
"C:/Users/taeil/Documents/GitHub/ESS/Data_Ingestion/preprocessing_filter/scalefactor_ETC.txt"
                value_range_file_path = "D:/vscode/ESS/range_ETC.txt"

        # 스케일 팩터파일
        scale_factor_file = open(scale_factor_file_path, "r")
        scale_factor = scale_factor_file.readlines()

        # 음수 변환
        for i in range(len(data)):
            if data[i] <= 32767:
                pass
            elif 32767 < data[i] < 65535:
                data[i] = -(65535 - data[i] + 1)

        # 파트 2, 3
        if partID == 2:
            for i in range(len(data)):
                if self.in_range(data[i], 0, 3):
                    pass
                else:
                    print("partID : ", partID)
                    print("번호 : ", i)
                    print("범위를 벗어난 값 입니다.", data[i])
                    preprocessing_logger.warning(
                        "An out-of-range value occurs at address %s of part %s : %s",

```

```

        i,
        partID,
        data[i],
    )
    scale_factor_file.close()
    return data

elif partID == 3:
    pass
else:
    value_range_file = open(value_range_file_path, "r")
    value_range = value_range_file.readlines()

# 스케일 팩터 적용
for i in range(len(data)):

    scale_factor[i] = scale_factor[i].strip("\n") # 스케일팩터 줄바꿈 문자 제거

    data[i] = int(data[i]) * float(scale_factor[i]) # 스케일팩터 적용
    data[i] = float("{:.3f}".format(data[i])) # 소수점 한자리 적용

# 값 범위(레인지) 판별
# PCS의 경우 값 범위가 없기때문에 그냥 리턴
if partID == 3:
    scale_factor_file.close()
    preprocessing_logger.info("part %s data parsing success", partID)
    return data

for i in range(len(data)):
    value_range[i] = value_range[i].strip("\n") # 레인지 줄바꿈 문자 제거
    min_value = float(value_range[i].split()[0])
    max_value = float(value_range[i].split()[2])

    if self.in_range(data[i], min_value, max_value):
        pass
    else:
        print("partID : ", partID)
        print("번호 : ", i)
        print("범위를 벗어난 값 입니다.", data[i])
        preprocessing_logger.warning(
            "An out-of-range value occurs at address %s of part %s : %s",
            i,
            partID,
            data[i],
        )

scale_factor_file.close()
value_range_file.close()

preprocessing_logger.info("part %s data parsing success", partID)

return data
except Exception as e:
    preprocessing_logger.error("part %s data preprocessing error : ", partID, e)

```

```
# 최대값 최소값 사이에 존재하는지 판별
def in_range(self, value, min, max):
    return min <= value <= max if max >= min else max <= value <= min

# BMS1,2 데이터를 조작하기 위한 메서드
def data_manipulation(BMS1, BMS2):

    # BANK 데이터 만들기

    BANK_ID = 1

    BMS1_data1 = BMS1[:9]
    BMS1_data2 = BMS1[9:]
    BMS2_data1 = BMS2[:34]
    BMS2_data2 = BMS2[34:]

    BMS1_data1.insert(0, BANK_ID)

    # 0:정상 1:이상으로 치환 221~229, 233번

    for i in range(9):
        if BMS2_data1[21 + i] == 0:
            BMS2_data1[21 + i] = 1
        elif BMS2_data1[21 + i] == 1:
            BMS2_data1[21 + i] = 0
    if BMS2_data1[-1] == 0:
        BMS2_data1[-1] = 1
    elif BMS2_data1[-1] == 1:
        BMS2_data1[-1] = 0

    BANK = BMS1_data1 + BMS2_data1

    # print("BANK : ", BANK)

    # RACK 데이터 1~8까지 만들기
    Rack_list = []
    for i in range(1, 9):
        temp1 = BMS1_data2[0 + 16 * (i - 1) : 16 * i]
        temp2 = BMS2_data2[0 + 29 * (i - 1) : 29 * i]

        temp3 = temp1 + temp2
        temp3.insert(0, i) # BANK, RACK ID 추가
        temp3.insert(0, BANK_ID) # BANK, RACK ID 추가
        Rack_list.append(temp3)
    # print("RACK : ", Rack_list)

    return BANK, Rack_list

def main():

    seoultime = datetime.now(timezone("asia/seoul"))
```

```

print(seoultime)

start = time.time()

test1 = ESS_Modbus()
test2 = ESS_Modbus()
test3 = ESS_Modbus()
test4 = ESS_Modbus()

list1 = test1.data_parsing(1)

exit(1)

list2 = test2.data_parsing(2)
list3 = test3.data_parsing(3)
list4 = test4.data_parsing(4)

Bank_data, Rack_data = data_manipulation(list1, list2)
PCS_data, ETC_data = list3, list4

timescale = timescale_input_test.timescale()
timescale.Bank_input_data(seoultime, Bank_data)
# timescale.Rack_input_data(seoultime, Rack_data)
# timescale.PCS_input_data(seoultime, PCS_data)
# timescale.ETC_input_data(seoultime, ETC_data)

end = time.time()
print(end - start)

class thread(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)

    def run(self):
        main()


def job():
    make_thread = thread()
    make_thread.run()

if __name__ == "__main__":

    start = int(time.time())
    count = 0
    while True:
        end = int(time.time())
        count += 1
        if start == end - 1:

            print("check1 : ", count)

```

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

```

job()
# main()
count = 0

# time.sleep(1)
start = end

```

o 로컬 DB 설계 및 데이터 축적 구조 설계

- 동일한 데이터를 로컬 DB에 저장하여 데이터 수집 안정성 확보
- GCS(Google Cloud Storage)를 통한 csv 파일 축적을 통해 다양한 방식의 데이터 제공
- GCS 버킷에 날짜별 데이터 종류에 따라 4종류의 데이터 축적

ess-bucket-1

위치: asia (아시아 지역의 다중 리전) | 스토리지 클래스: Standard | 공개 액세스: 공개 아님 | 보호: 없음

객체 구성 권한 보호 수명 주기

버킷 > ess-bucket-1 > 2021 > 09

파일 업로드 볼더 업로드 볼더 만들기 보존 조치 관리 다운로드 삭제

이름 프리픽스로만 필터링 | 볼더 객체 및 볼더 필터링

<input type="checkbox"/>	이름	크기	유형	생성 시간
<input type="checkbox"/>	22/	-	볼더	-
<input type="checkbox"/>	23/	-	볼더	-
<input type="checkbox"/>	24/	-	볼더	-
<input type="checkbox"/>	25/	-	볼더	-
<input type="checkbox"/>	26/	-	볼더	-
<input type="checkbox"/>	27/	-	볼더	-
<input type="checkbox"/>	28/	-	볼더	-

ess-bucket-1

위치: asia (아시아 지역의 다중 리전) | 스토리지 클래스: Standard | 공개 액세스: 공개 아님 | 보호: 없음

객체 구성 권한 보호 수명 주기

버킷 > ess-bucket-1 > 2021 > 09 > 28

파일 업로드 볼더 업로드 볼더 만들기 보존 조치 관리 다운로드 삭제

이름 프리픽스로만 필터링 | 볼더 객체 및 볼더 필터링

<input type="checkbox"/>	이름	크기	유형	생성 시간
<input type="checkbox"/>	20210928_bank.csv	12.6MB	text/csv	2021. 9. 29. AM 12:00:30
<input type="checkbox"/>	20210928_etc.csv	4.8MB	text/csv	2021. 9. 29. AM 12:00:36
<input type="checkbox"/>	20210928_pcs.csv	34.2MB	text/csv	2021. 9. 29. AM 12:00:35
<input type="checkbox"/>	20210928_rack.csv	117.4MB	text/csv	2021. 9. 29. AM 12:00:33

그림 5 날짜별 데이터 종류에 따라 GCS에 csv파일 저장