```
SERVER_REC_DB_SERVICE_NAME=server_rec_db
SERVER_REC_DB_CONTAINER_NAME=dimis-server-db-container
DB_NAME=server-db
POSTGRES_USER=dimis_dbuser
POSTGRES_PASSWORD=dimis_dbpassword
SERVER_REC_DB_VOLUME=server_rec_db-volume
SERVER_REC_PORT_DOCKER=5432
SERVER_REC_PORT_EXT=4321
CONSUMER_CONTAINER_NAME=server_consumer
RABBIT_HOST=servermq-container
RABBIT_USER=rabbitmq
RABBIT_PASSWORD=rabbitmq
RABBIT_PORT=5672
ADMINER_WEB_PORT_DOCKER=8181
ADMINER_WEB_PORT_EXT=8182
NETWORK_NAME=server-mttq-net
```

```yaml
version: '3.8'
services:
  server_rec_db:
    image: "postgres:11"
    container_name: "${SERVER_REC_DB_CONTAINER_NAME}"
    restart: always
    environment:
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
      - POSTGRES_DB=${DB_NAME}
    ports:
      - "${SERVER_REC_PORT_EXT}:${SERVER_REC_PORT_DOCKER}"

  server_consumer:
    build: .
    container_name: "${CONSUMER_CONTAINER_NAME}"
    depends_on:
      - ${SERVER_REC_DB_SERVICE_NAME}
    restart: always
    command: ["./wait-for-it.sh", "${SERVER_REC_DB_CONTAINER_NAME}:${SERVER_REC_PORT_DOCKER}", "--", "python3", "-u", "./pika_consumer.py"]
    environment:
      - DB_SERVICE_NAME=${SERVER_REC_DB_SERVICE_NAME}
      - DB_CONTAINER_NAME=${SERVER_REC_DB_CONTAINER_NAME}
      - DB_NAME=${DB_NAME}
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
      - RABBIT_HOST=${RABBIT_HOST}
      - RABBIT_USER=${RABBIT_USER}
      - RABBIT_PASSWORD=${RABBIT_PASSWORD}
      - RABBIT_PORT=${RABBIT_PORT}

  server_adminer:
    image: adminer
    restart: always
    environment:
      - ADMINER_DEFAULT_SERVER=${
```

```yaml
SERVER_REC_DB_CONTAINER_NAME}:${
SERVER_REC_PORT_DOCKER}
      - ADMINER_DEFAULT_USER=${POSTGRES_USER}
      - ADMINER_DEFAULT_PASSWORD=${POSTGRES_PASSWORD}
    ports:
    - "${ADMINER_WEB_PORT_EXT}:${
ADMINER_WEB_PORT_DOCKER}"
    command:
      - 'php'
      - '-S'
      - '[::]:${ADMINER_WEB_PORT_EXT}'
      - '-t'
      - '/var/www/html'
    entrypoint:
      - 'entrypoint.sh'
      - 'docker-php-entrypoint'

volumes:
  server_rec_db-volume:
    driver: local
    driver_opts:
      type: volume
      device: /docker/projects/dimis/db
      o: bind

networks:
  default:
    external: true
    name: ${NETWORK_NAME}

```

```
FROM python:3.7-stretch

COPY requirements.txt /tmp/

RUN pip install --no-cache-dir -r /tmp/requirements.
txt

RUN useradd --create-home appuser
WORKDIR /home/appuser
USER appuser

COPY wait-for-it.sh .
COPY pika_consumer.py .
COPY .env .

#RUN ./wait-for-it.sh dmis_db:5432 -- python3 -u ./
pika_consumer.py

CMD ["python3", "-u", "./pika_consumer.py"]

```

```python
import json
import time
import zlib
from os import environ

import pika
import sqlalchemy
from sqlalchemy import Table, Column, MetaData
from sqlalchemy import create_engine
from sqlalchemy.dialects import postgresql
from sqlalchemy.dialects.postgresql import insert
from sqlalchemy.exc import IntegrityError
from sqlalchemy.types import BIGINT

global DIMIS_RECORDINGS_DB_PATH

with open('.env', 'rb') as env_file:
    environment_variables = env_file.read().split(sep=b"\n")
    postgres_user = postgres_password = db_name = service_name = None
    for variable_line in environment_variables:
        break


def dict_key_filter(d_in: dict) -> dict:
    valid_keys = ('DeviceName', 'Unixtime Request', 'Unixtime Reply',
                  'Wechselspannung', 'Wechselspannung', 'Wechselstrom', 'Leistung')
    d_out = {key: d_in[key] for key in valid_keys}
    return d_out


if __name__ == '__main__':
    # We use an environment variable to configure the consumer-container via docker-compose
    missing_environ = []
    expected_environ = ['RABBIT_HOST', 'POSTGRES_USER', 'POSTGRES_PASSWORD', 'DB_NAME',
                        'DB_CONTAINER_NAME', '
RABBIT_USER', 'RABBIT_PASSWORD', 'RABBIT_PORT']
    for element in expected_environ:
        if element not in environ:
            missing_environ.append(element)

    if missing_environ.__len__() == 0:
        rabbit_host = str(environ['RABBIT_HOST'])
        postgres_user = str(environ['POSTGRES_USER'])
        postgres_password = str(environ['POSTGRES_PASSWORD'])
        db_name = str(environ['DB_NAME'])
        db_host_name = str(environ['DB_CONTAINER_NAME'])
        rabbit_user = str(environ['RABBIT_USER'])
        rabbit_password = str(environ['RABBIT_PASSWORD'])
        rabbit_port = str(environ['RABBIT_PORT'])

    else:
        print('Missing .env configuration:', missing_environ)
        exit(10)

    DIMIS_RECORDINGS_DB_PATH = f'postgresql://{postgres_user}:{postgres_password}@{db_host_name}/{db_name}'

    time.sleep(3)  # sleep for SQL start

engine = create_engine(
    DIMIS_RECORDINGS_DB_PATH,
    echo=False)

# noinspection PyUnboundLocalVariable
credentials = pika.PlainCredentials(rabbit_user, rabbit_password)
# noinspection PyUnboundLocalVariable
connection = pika.BlockingConnection(
    pika.ConnectionParameters(host=rabbit_host, port=rabbit_port, credentials=credentials))
channel = connection.channel()
```

```python
channel.queue_declare(queue='task_queue', durable=True)
channel.basic_qos(prefetch_count=1)

for method_frame, properties, body in channel.consume('task_queue'):
    body_decompressed = zlib.decompress(body)
    list_of_dicts = json.loads(body_decompressed, encoding='utf-8')
    print("Tag ", method_frame.delivery_tag,
          " Received %0.2f kB %03d Messages" % ((len(body) / 1024), len(list_of_dicts)), flush=True)

    values_of_device_dict = {}  # dict of devicename -> list of recorded values of that device
    list_of_dicts = [dict_key_filter(d) for d in list_of_dicts]
    # Convert Unixtime from seconds[float] to milliseconds[int]
    for d in list_of_dicts:
        device_name = d['DeviceName']
        del d['DeviceName']
        if type(d['Unixtime Request']) == float:
            d['Unixtime Request'] = int(d['Unixtime Request'] * 1000)
        if type(d['Unixtime Reply']) == float:
            d['Unixtime Reply'] = int(d['Unixtime Reply'] * 1000)
        try:
            float(d['Wechselspannung'])  # check if value is float
            float(d['Leistung'])
            float(d['Wechselstrom'])
            if device_name in values_of_device_dict:
                previous_list = values_of_device_dict.get(device_name)
                list(previous_list).append(d)
                values_of_device_dict[device_name] = previous_list
            else:
                values_of_device_dict[device_name
```

```python
] = [d]
        except ValueError:
            print("Not a float, ignored")
            continue

    metadata = MetaData()
    recordings_table_dict = {}  # dict devicename -> sql table objects
    for recording_device_name in values_of_device_dict.keys():
        recordings_table = Table(recording_device_name, metadata,
                                 Column('Unixtime Request', BIGINT, primary_key=True, autoincrement=False),
                                 Column('Unixtime Reply', BIGINT, primary_key=True, autoincrement=False),
                                 Column('Wechselspannung', postgresql.DOUBLE_PRECISION),
                                 Column('Wechselstrom', postgresql.DOUBLE_PRECISION),
                                 Column('Leistung', postgresql.DOUBLE_PRECISION))
        recordings_table_dict[recording_device_name] = recordings_table

    metadata.create_all(engine)

    print(f'received recordings for the following devices: {values_of_device_dict.keys()}')

    for recording_device_name in recordings_table_dict.keys():
        try:
            # https://docs.sqlalchemy.org/en/13/core/tutorial.html#executing-multiple-statements
            # runs as SQL-transaction
            with engine.begin() as connection:
                print(
                    f'Begin insert to table of
```

```python
device {recording_device_name} with {values_of_device_dict[recording_device_name].__len__()} elements')
                result = connection.execute(recordings_table_dict.get(recording_device_name).insert(),
                                            values_of_device_dict[recording_device_name])
                assert result
        except sqlalchemy.exc.IntegrityError as e:
            print("sqlalchemy.exc.IntegrityError -> Postgres UPSERT DO_NOTHING")
            print(e)
            with engine.begin() as connection:
                insert_stmt = insert(table=recordings_table_dict.get(recording_device_name),
                                     values=values_of_device_dict[recording_device_name])
                do_nothing_stmt = insert_stmt.on_conflict_do_nothing(
                    index_elements=['Unixtime Request', 'Unixtime Reply'])
                result = connection.execute(do_nothing_stmt)
                assert result
        except Exception as e:
            channel.basic_nack(method_frame.delivery_tag)
            raise e

    # Acknowledge the message
    channel.basic_ack(method_frame.delivery_tag)

# Cancel the consumer and return any pending messages
requeued_messages = channel.cancel()
print('Requeued %i messages' % requeued_messages)

# Close the channel and the connection
channel.close()
connection.close()
```