**Проектная работа на тему: Сбор и обработка данных по валютам с помощью ClickHouse.**

Система ежедневно получает данные с сайта банка России по api в формате xml:

<https://www.cbr.ru/development/SXML/>

1. **Сборка компонентов для получения и хранения данных.**

Переходим в папку проекта - OtusClickHouse-Project\.

Собираем и запускаем контейнер PostgreSQL с помощью скрипта docker/build\_docker\_postgres.sh.

* build\_docker\_postgres.sh:

#!/bin/bash

docker stop postgresql

docker rm postgresql

docker run --name postgresql \

-e POSTGRES\_PASSWORD=passwordpg1234 \

-e POSTGRES\_USER=userpg \

-e POSTGRES\_DB=pgdb \

-p 5433:5432 \

-v "./postgresql/data":"/var/lib/postgresql/data" \

-d postgres:latest

Подключаемся к базе и создаем схему с таблицей:

**CREATE** **SCHEMA** pgclick;

**CREATE** **TABLE** **IF** **NOT** **EXISTS** pgclick.valute\_data(

id **bigint** **NOT** **NULL** **PRIMARY** **KEY**,

date **VARCHAR**(36) **NOT** **NULL**,

name **VARCHAR**(200) **NOT** **NULL**,

str\_id **VARCHAR**(20) **NOT** **NULL**,

num\_code **VARCHAR**(20) **NOT** **NULL**,

char\_code **VARCHAR**(20) **NOT** **NULL**,

nominal **VARCHAR**(20) **NOT** **NULL**,

value **VARCHAR**(50) **NOT** **null**

);

**CREATE** **UNIQUE** **INDEX** idx\_valute\_pk

**ON** pgclick.valute\_data(id);

**CREATE** **UNIQUE** **INDEX** idx\_valute\_data

**ON** pgclick.valute\_data(**date**, str\_id);

**CREATE** **SEQUENCE** **IF** **NOT** **EXISTS** pgclick.seq\_valute\_id;

Собираем приложение xml-parser с помощью скрипта xml-parser/build\_app.sh. Для сборки потребуется установленный maven. Docker-контейнер собираем и запускаем с помощью скрипта xml-parser/build\_docker.sh.

* build\_app.sh:

#!/bin/bash

git pull

mvn clean install

* build\_docker.sh:

#!/bin/bash

docker build -f app.Dockerfile -t uoles/xml-parser:1.0.1 .

docker stop xml-parser

docker rm xml-parser

docker run -d --name xml-parser \

--network=host \

--publish 8090:8090 \

uoles/xml-parser:1.0.1

* app.Dockerfile:

FROM docker.io/library/openjdk:17

MAINTAINER Maksim Kulikov <max.uoles@rambler.ru>

COPY target/xml-parser.jar xml-parser.jar

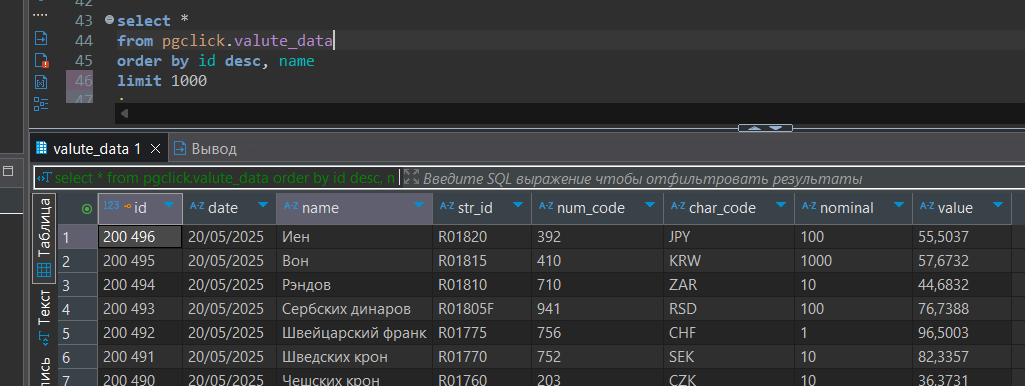
ENV TZ=Europe/Moscow

RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone

EXPOSE 8090

CMD ["java","-jar","xml-parser.jar"]

После запуска базы и приложения начинается сбор данных с 1 декабря 2010 года по текущее время. Так же, ежедневно в 12:30 подгружаются данные на текущие сутки.



1. **Сборка компонентов для обработки данных.**

Переходим в каталог OtusClickHouse-Project\docker и запускаем сборку docker-compose.yml, в котором запускаются clickhouse, kafka и superset.

Для этого выполняем команду: docker-compose up –d

Superset – <http://localhost:8088>

Kafka-ui – <http://localhost:8089>

docker-compose.yml:

x-superset-image: &superset-image apache/superset:${TAG:-latest-dev}

x-superset-depends-on: &superset-depends-on

- db

- redis

x-superset-only-depends-on: &superset-only-depends-on

- db

- redis

- clickhouse-server

x-superset-volumes: &superset-volumes

- ./docker:/app/docker

- superset\_home:/app/superset\_home

version: "3.7"

services:

clickhouse-server:

container\_name: clickhouse-server

image: uoles/clickhouse:25.2.1

build:

context: .

dockerfile: clickhouse-25.2.1.Dockerfile

environment:

CLICKHOUSE\_DB: my\_database

CLICKHOUSE\_USER: username

CLICKHOUSE\_DEFAULT\_ACCESS\_MANAGEMENT: 1

CLICKHOUSE\_PASSWORD: password

ports:

- "18123:8123"

- "19000:9000"

volumes:

- ./clickhouse/config.xml:/etc/clickhouse-server/config.xml

- ./clickhouse/zookeeper-servers.xml:/etc/clickhouse-server/conf.d/zookeeper-servers.xml

ulimits:

nofile:

soft: 262144

hard: 262144

depends\_on:

- kafka

links:

- kafka

zookeeper:

container\_name: zookeeper

image: confluentinc/cp-zookeeper:6.2.4

healthcheck:

test: [ "CMD", "nc", "-vz", "localhost", "2181" ]

interval: 10s

timeout: 3s

retries: 3

environment:

ZOOKEEPER\_CLIENT\_PORT: 2181

ZOOKEEPER\_TICK\_TIME: 2000

ports:

- 22181:2181

kafka:

container\_name: kafka

image: confluentinc/cp-kafka:6.2.4

depends\_on:

zookeeper:

condition: service\_healthy

ports:

- "29092:29092"

- "29093:29093"

healthcheck:

test: [ "CMD", "nc", "-vz", "localhost", "9092" ]

interval: 10s

timeout: 3s

retries: 3

environment:

KAFKA\_BROKER\_ID: 1

KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181

KAFKA\_LISTENERS: OUTSIDE://:29092,INTERNAL://:9092,EXTERNAL\_DIFFERENT\_HOST://:29093

KAFKA\_ADVERTISED\_LISTENERS: OUTSIDE://localhost:29092,INTERNAL://kafka:9092,EXTERNAL\_DIFFERENT\_HOST://89.169.3.137:29093

KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: INTERNAL:PLAINTEXT,OUTSIDE:PLAINTEXT,EXTERNAL\_DIFFERENT\_HOST:PLAINTEXT

KAFKA\_INTER\_BROKER\_LISTENER\_NAME: INTERNAL

KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

kafka-ui:

image: provectuslabs/kafka-ui

container\_name: kafka-ui

ports:

- "8089:8080"

restart: always

depends\_on:

kafka:

condition: service\_healthy

environment:

KAFKA\_CLUSTERS\_0\_NAME: local

KAFKA\_CLUSTERS\_0\_BOOTSTRAPSERVERS: kafka:9092

redis:

image: redis:7

container\_name: superset\_cache

restart: unless-stopped

volumes:

- redis:/data

db:

env\_file: docker/.env-non-dev

image: postgres:14

container\_name: superset\_db

restart: unless-stopped

volumes:

- db\_home:/var/lib/postgresql/data

superset:

env\_file: docker/.env-non-dev

image: \*superset-image

container\_name: superset\_app

command: [ sh, -c, "pip install clickhouse-connect && /app/docker/docker-bootstrap.sh app-gunicorn" ]

user: "root"

restart: unless-stopped

ports:

- 8088:8088

depends\_on: \*superset-only-depends-on

volumes: \*superset-volumes

links:

- clickhouse-server

superset-init:

image: \*superset-image

container\_name: superset\_init

command: [ "/app/docker/docker-init.sh" ]

env\_file: docker/.env-non-dev

depends\_on: \*superset-depends-on

user: "root"

volumes: \*superset-volumes

healthcheck:

disable: true

superset-worker:

image: \*superset-image

container\_name: superset\_worker

command: [ "/app/docker/docker-bootstrap.sh", "worker" ]

env\_file: docker/.env-non-dev

restart: unless-stopped

depends\_on: \*superset-depends-on

user: "root"

volumes: \*superset-volumes

healthcheck:

test: [ "CMD-SHELL", "celery inspect ping -A superset.tasks.celery\_app:app -d celery@$$HOSTNAME" ]

superset-worker-beat:

image: \*superset-image

container\_name: superset\_worker\_beat

command: [ "/app/docker/docker-bootstrap.sh", "beat" ]

env\_file: docker/.env-non-dev

restart: unless-stopped

depends\_on: \*superset-depends-on

user: "root"

volumes: \*superset-volumes

healthcheck:

disable: true

volumes:

superset\_home:

external: false

db\_home:

external: false

redis:

external: false

- Подключаемся к clickhouse и проливаем таблицы и вьюхи:

**CREATE** **TABLE** valute\_data

(

id UInt64,

date **Date**,

name String,

str\_id String,

num\_code String,

char\_code String,

nominal UInt32,

value Float32

)

ENGINE = MergeTree

**ORDER** **BY** (id, **date**, str\_id);

**CREATE** **TABLE** valute\_data\_queue

(

id UInt64,

date String,

name String,

str\_id String,

num\_code String,

char\_code String,

nominal String,

value String

)

ENGINE = Kafka

SETTINGS

kafka\_broker\_list = 'kafka:9092',

kafka\_topic\_list = 'valute\_topic',

kafka\_group\_name = 'valute\_group',

kafka\_format = 'JSONEachRow',

kafka\_row\_delimiter = '\n',

kafka\_num\_consumers = 1,

kafka\_skip\_broken\_messages = 10,

kafka\_thread\_per\_consumer = 0;

**CREATE** MATERIALIZED **VIEW** valute\_data\_queue\_mv **TO** valute\_data

(

id UInt64,

**date** **Date**,

name String,

str\_id String,

num\_code String,

char\_code String,

nominal UInt32,

value Float32

) **AS**

**SELECT**

id,

parseDateTimeBestEffortOrNull(**date**,'Europe/Moscow') **AS** **date**,

name,

str\_id,

num\_code,

char\_code,

toInt32OrNull(nominal) **as** nominal,

toFloat32OrNull(replaceOne(value, ',', '.')) / nominal **as** value

**FROM** valute\_data\_queue

**WHERE** name **not** **in** ('Евро', 'Дирхам ОАЭ', 'Тенге');

**CREATE** MATERIALIZED **VIEW** valute\_data\_evro\_mv **TO** valute\_data\_evro

(

id UInt64,

**date** **Date**,

name String,

str\_id String,

num\_code String,

char\_code String,

nominal UInt32,

value Float32

) **AS**

**SELECT**

id,

parseDateTimeBestEffortOrNull(**date**,'Europe/Moscow') **AS** **date**,

name,

str\_id,

num\_code,

char\_code,

toInt32OrNull(nominal) **as** nominal,

toFloat32OrNull(replaceOne(value, ',', '.')) / nominal **as** value

**FROM** valute\_data\_queue

**WHERE** name = 'Евро';

**CREATE** **TABLE** valute\_data\_evro

(

id UInt64,

date **Date**,

name String,

str\_id String,

num\_code String,

char\_code String,

nominal UInt32,

value Float32

)

ENGINE = MergeTree

**ORDER** **BY** (id, **date**, str\_id);

**CREATE** MATERIALIZED **VIEW** valute\_data\_dirham\_mv **TO** valute\_data\_dirham

(

id UInt64,

**date** **Date**,

name String,

str\_id String,

num\_code String,

char\_code String,

nominal UInt32,

value Float32

) **AS**

**SELECT**

id,

parseDateTimeBestEffortOrNull(**date**,'Europe/Moscow') **AS** **date**,

name,

str\_id,

num\_code,

char\_code,

toInt32OrNull(nominal) **as** nominal,

toFloat32OrNull(replaceOne(value, ',', '.')) /nominal **as** value

**FROM** valute\_data\_queue

**WHERE** name = 'Дирхам ОАЭ';

**CREATE** **TABLE** valute\_data\_dirham

(

id UInt64,

**date** **Date**,

name String,

str\_id String,

num\_code String,

char\_code String,

nominal UInt32,

value Float32

)

ENGINE = MergeTree

**ORDER** **BY** (id, **date**, str\_id);

**CREATE** MATERIALIZED **VIEW** valute\_data\_tenge\_mv **TO** valute\_data\_tenge

(

id UInt64,

**date** **Date**,

name String,

str\_id String,

num\_code String,

char\_code String,

nominal UInt32,

value Float32

) **AS**

**SELECT**

id,

parseDateTimeBestEffortOrNull(**date**,'Europe/Moscow') **AS** **date**,

name,

str\_id,

num\_code,

char\_code,

toInt32OrNull(nominal) **as** nominal,

toFloat32OrNull(replaceOne(value, ',', '.')) /nominal **as** value

**FROM** valute\_data\_queue

**WHERE** name = 'Тенге';

**CREATE** **TABLE** valute\_data\_tenge

(

id UInt64,

date **Date**,

name String,

str\_id String,

num\_code String,

char\_code String,

nominal UInt32,

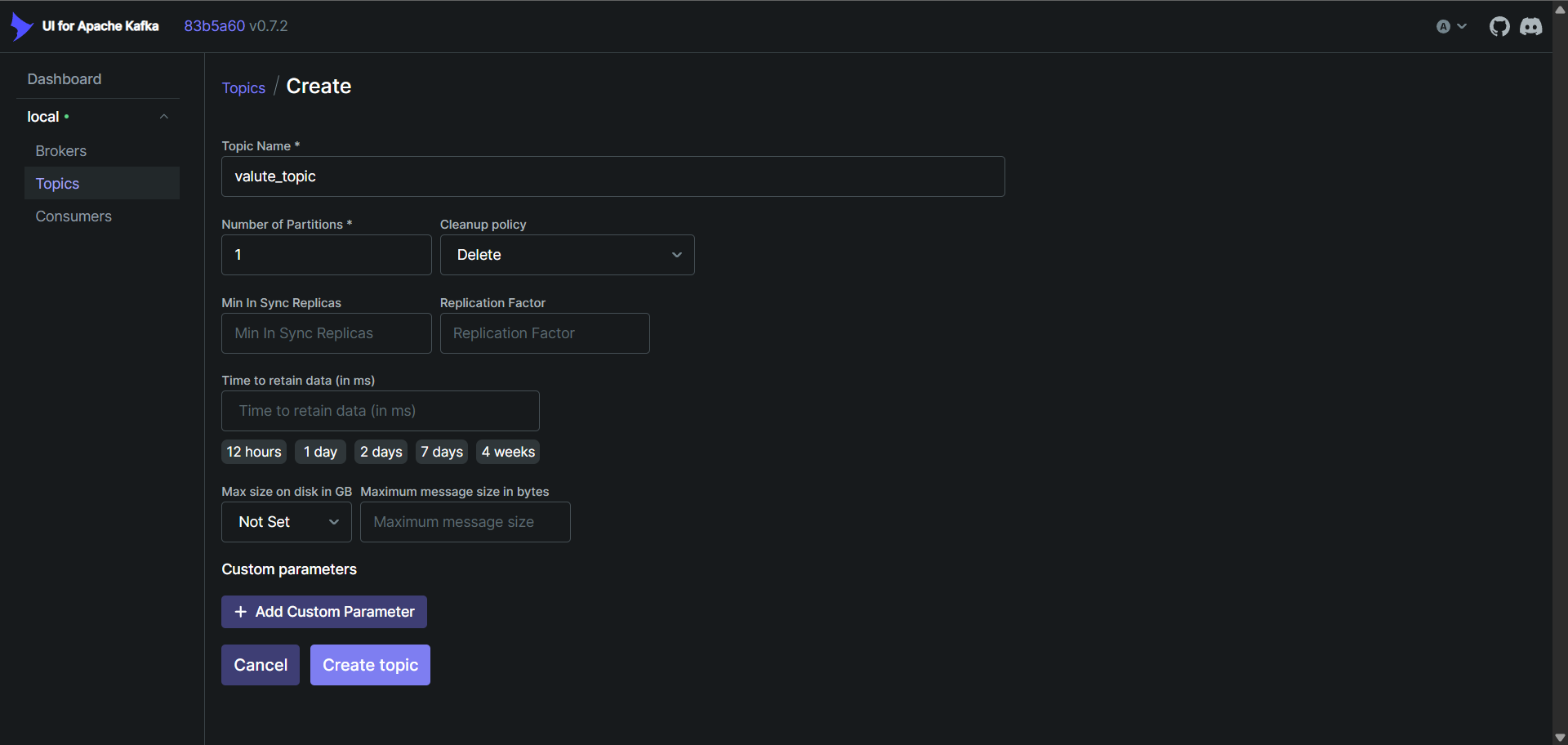
value Float32

)

ENGINE = MergeTree

**ORDER** **BY** (id, **date**, str\_id);

- Подключаемся к kafka-ui и создаем топик “valute\_topic” куда будут приходить обновления от debezium:



Теперь kafka и clickhouse могут принимать данные.

1. **Настраиваем PostgreSQL и разворачиваем приложение с компонентом debezium.**

Подключаемся к ранее развернутому контейнеру PostgreSQL и подготавливаем ее к подключению приложения. Для этого нужно выставить:

- уровень репликации “logical” и перезапустить базу;

- создать публикацию, с описанием таблиц и операций, которые будем обрабатывать;

- нужен пользователь со свойством REPLICATION (есть по умолчанию у пользователя при создании контейнера).

Скрипты:

**ALTER** **SYSTEM** **SET** wal\_level = logical;

SHOW wal\_level; -- проверяем уровень репликации после рестарта

**CREATE** PUBLICATION pgclick\_publication

**FOR** TABLES **IN** **SCHEMA** pgclick

**with**(publish = 'insert');

Для запуска приложения переходим в каталог OtusClickHouse-Project\debezium\. Так же, как и xml-parser собираем и разворачиваем debezium скриптами build\_app.sh и build\_docker.sh.

* build\_app.sh:

#!/bin/bash

git pull

mvn clean install

* build\_docker.sh:

#!/bin/bash

docker build -f app.Dockerfile -t uoles/debezium:1.0.1 .

docker stop debezium

docker rm debezium

docker run -d --name debezium \

--network=host \

--publish 8091:8091 \

uoles/debezium:1.0.1

* app.Dockerfile:

FROM docker.io/library/openjdk:17

MAINTAINER Maksim Kulikov <max.uoles@rambler.ru>

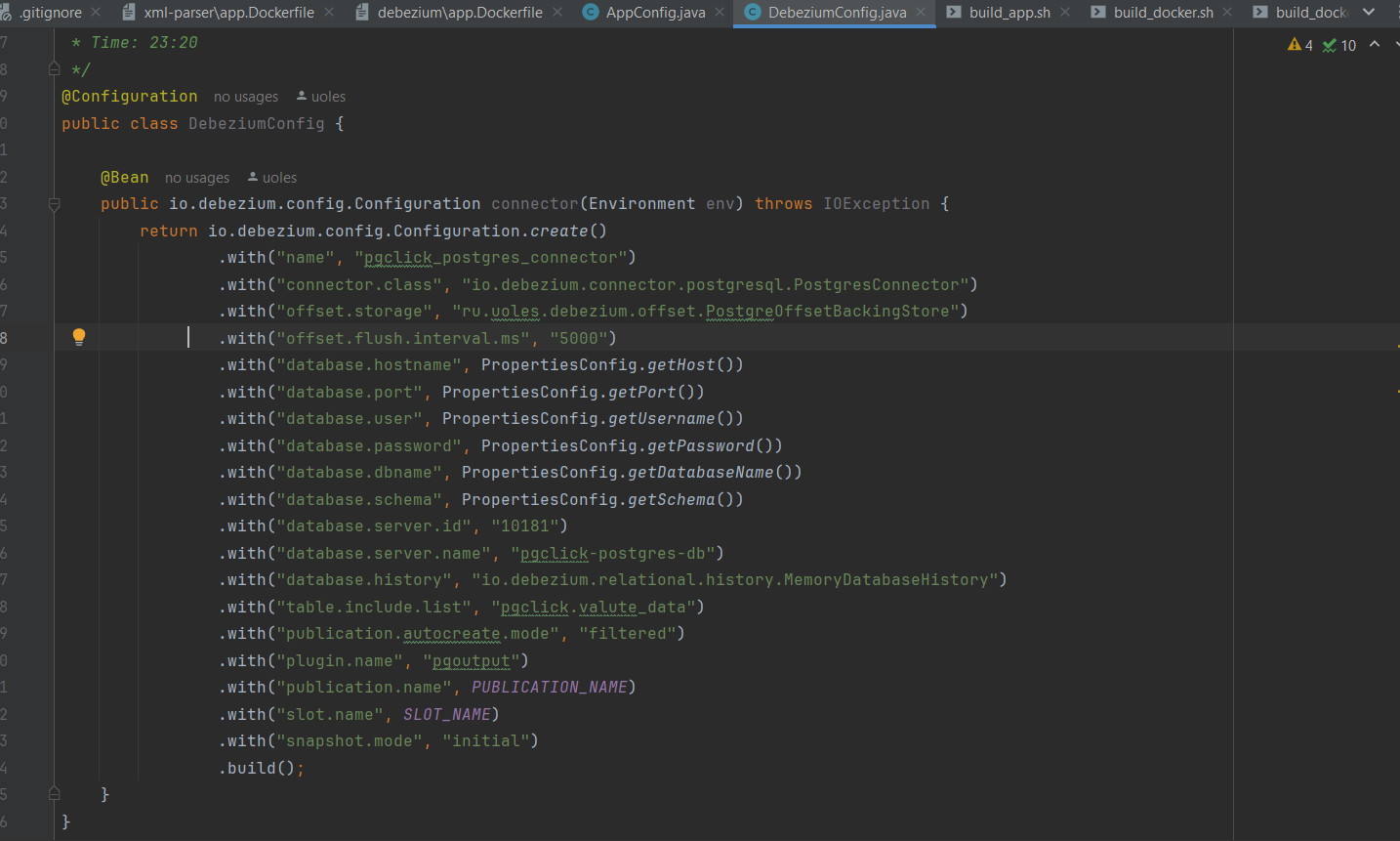
COPY target/debezium.jar debezium.jar

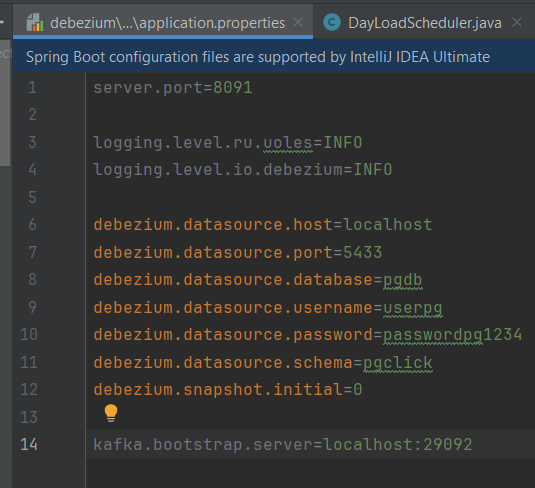
EXPOSE 8091

CMD ["java","-jar","debezium.jar"]

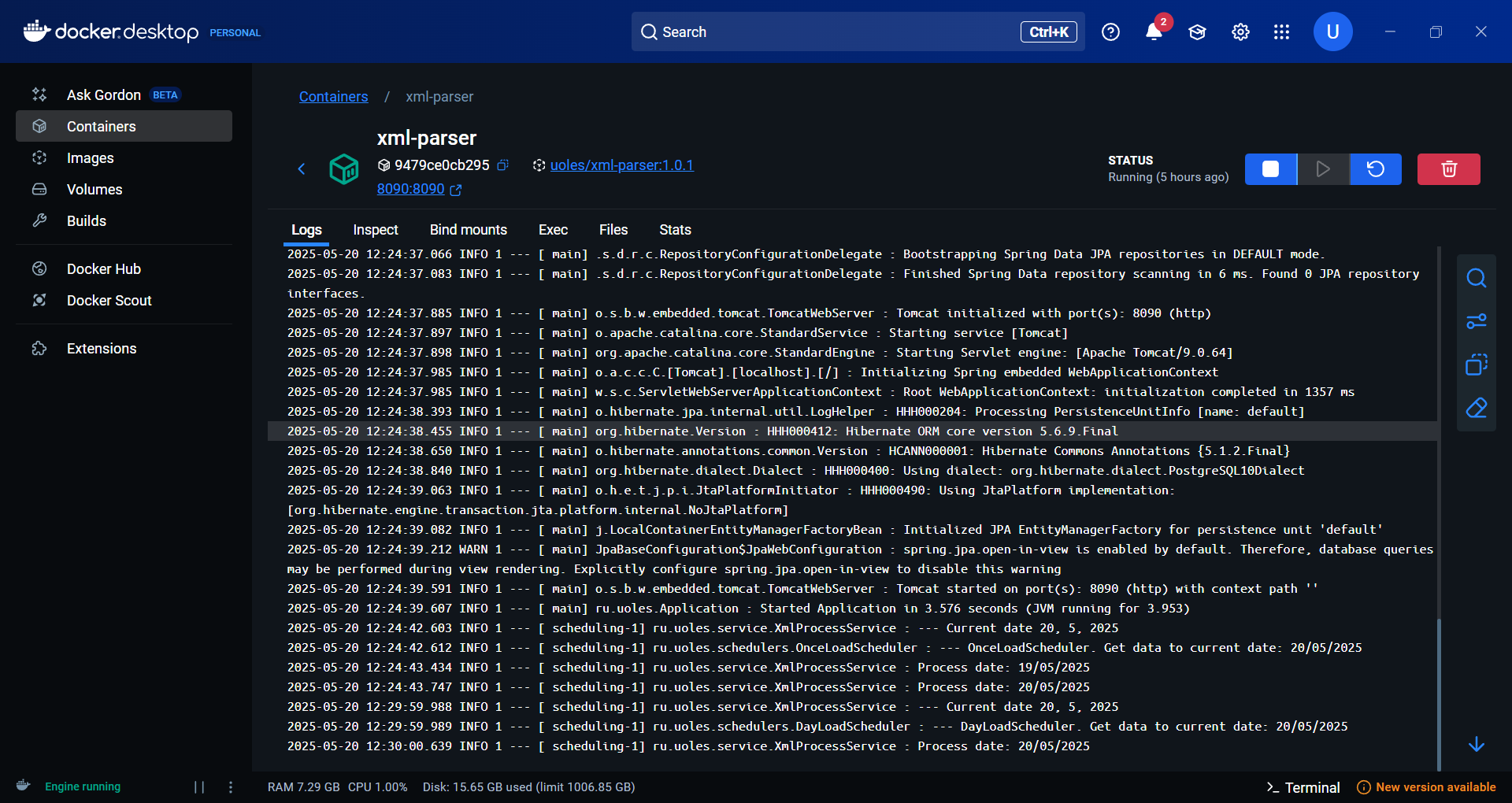
Приложение подключается к бд PostgreSQL, получает все изменения в базе из wal-журнала через библиотеку debezium и отправляет данные в kafka в формате json.

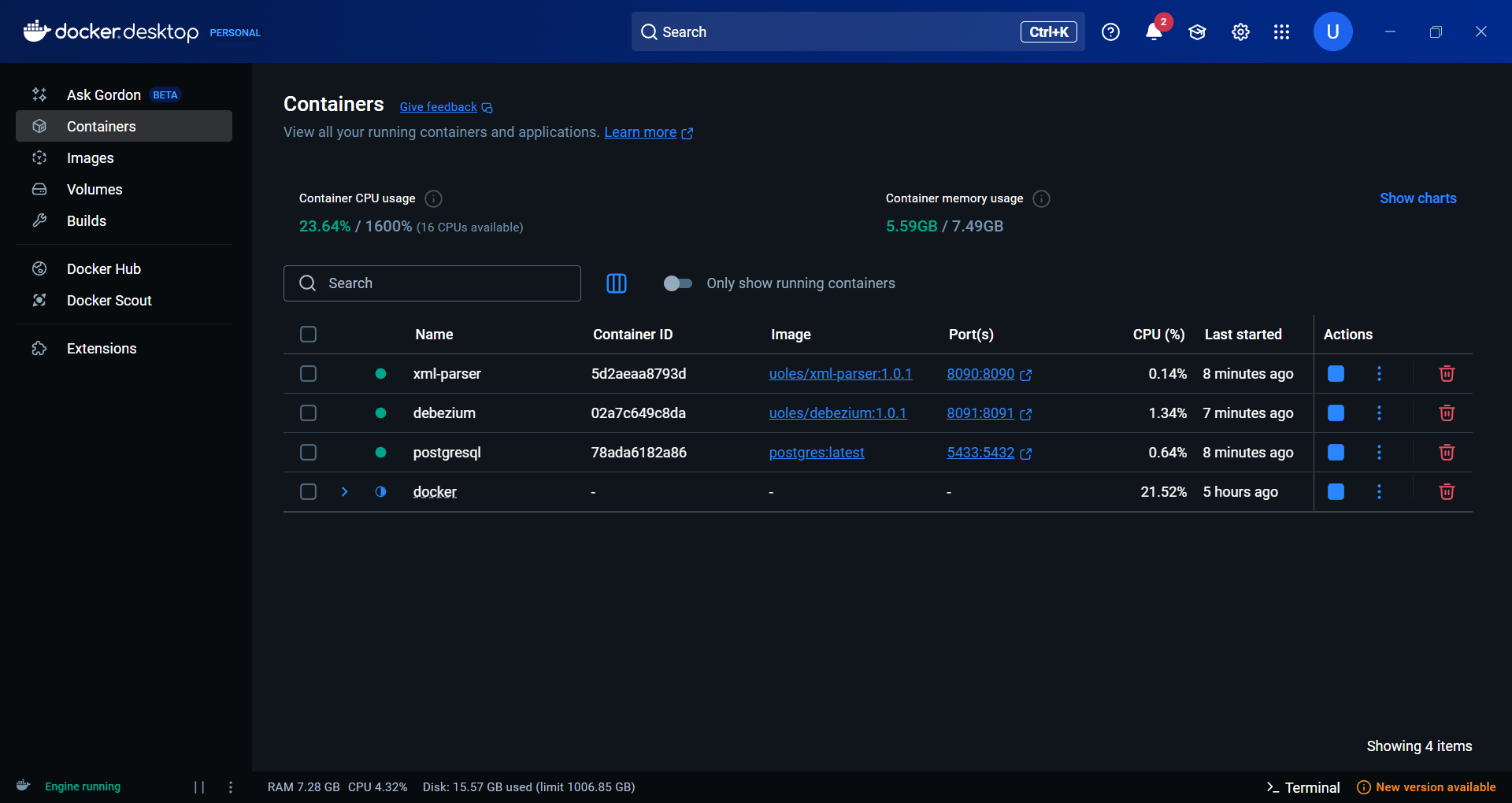
Конфигурация debezium:

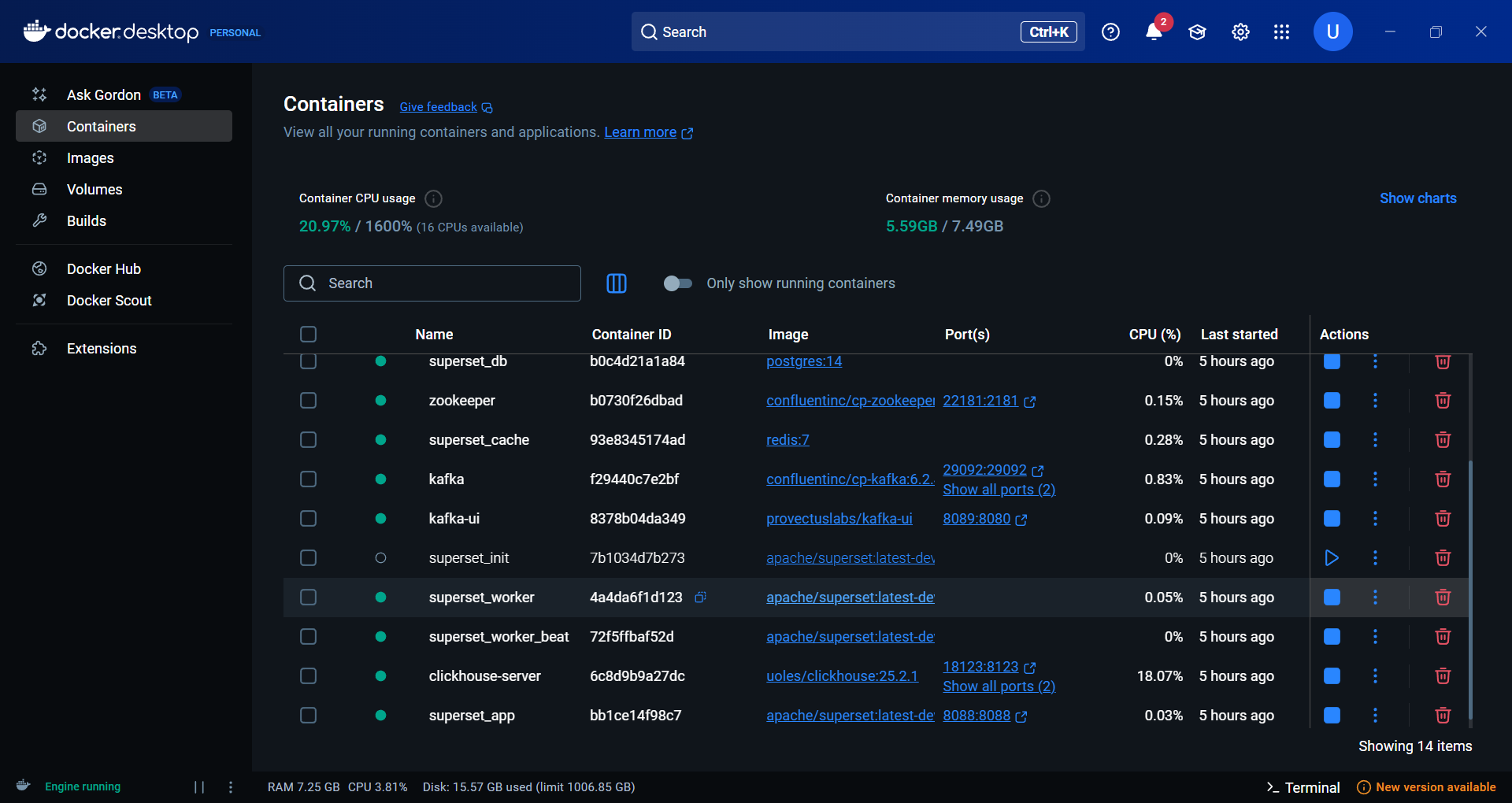


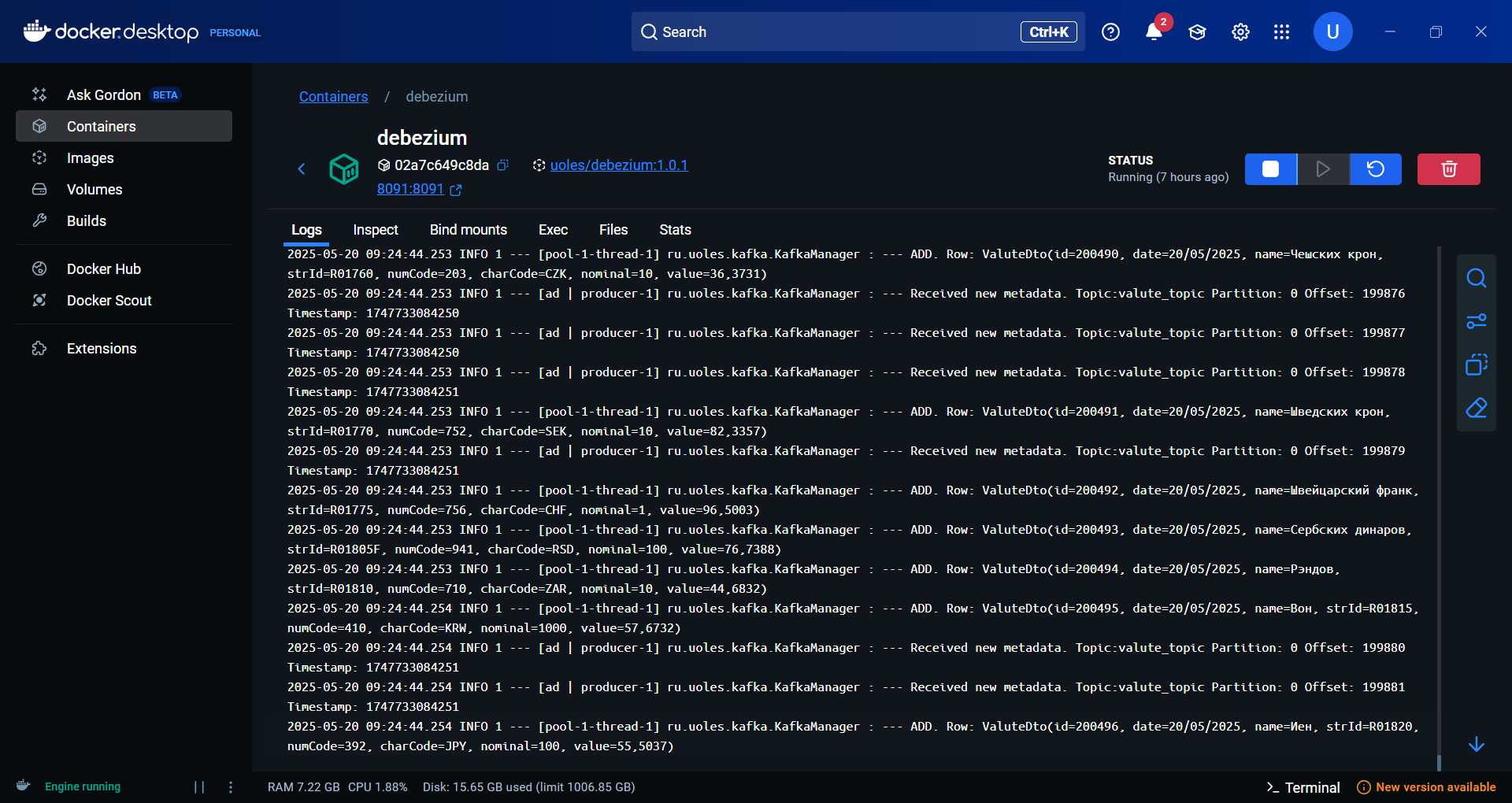


Скрины логами и данными:

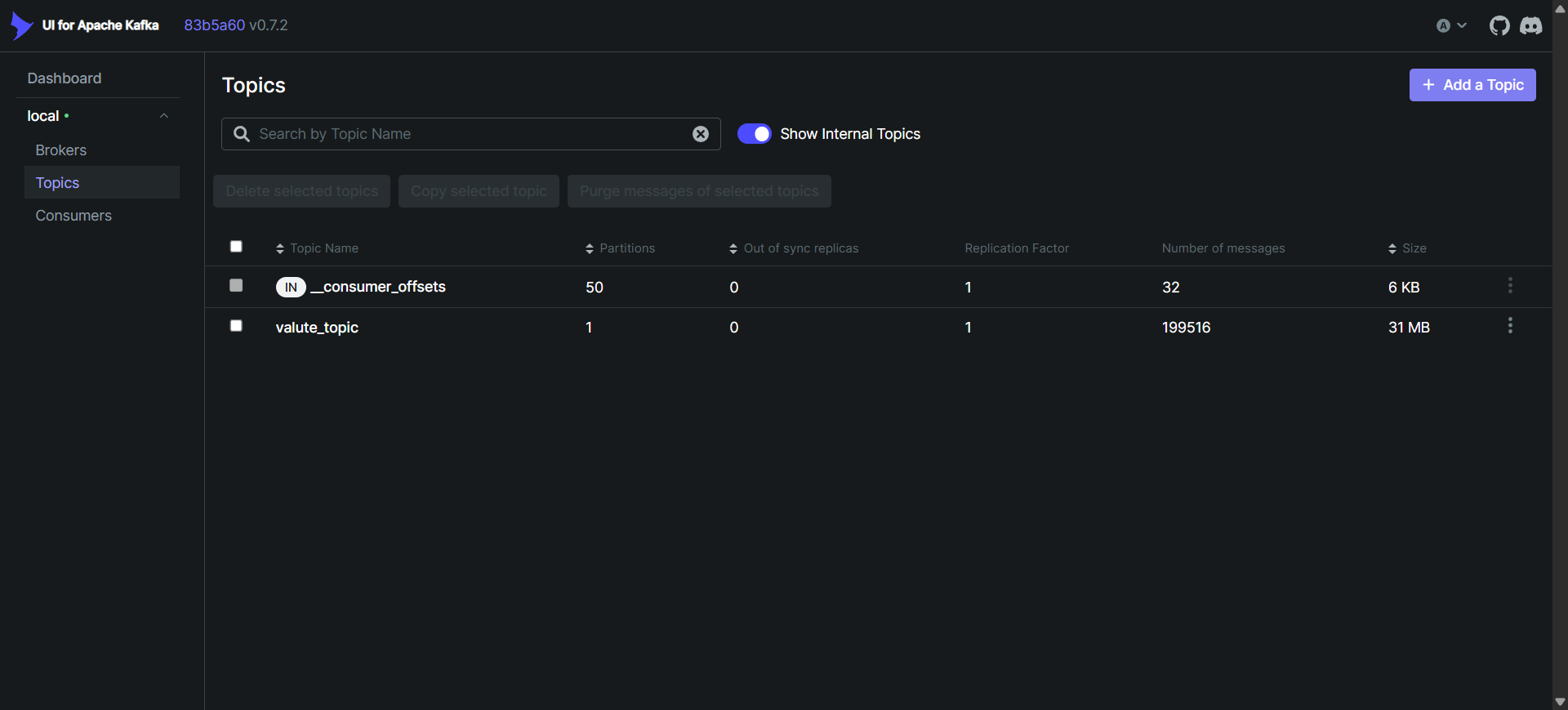


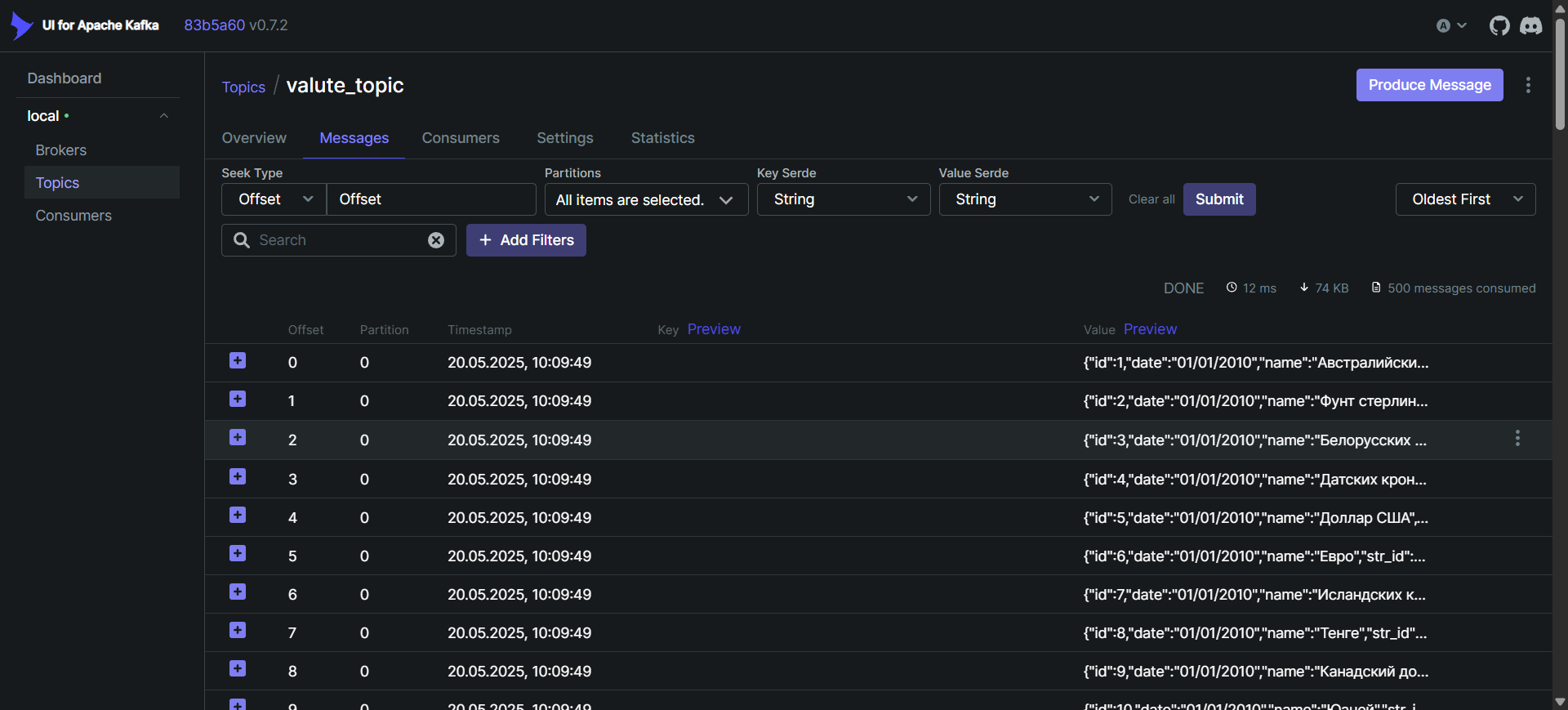




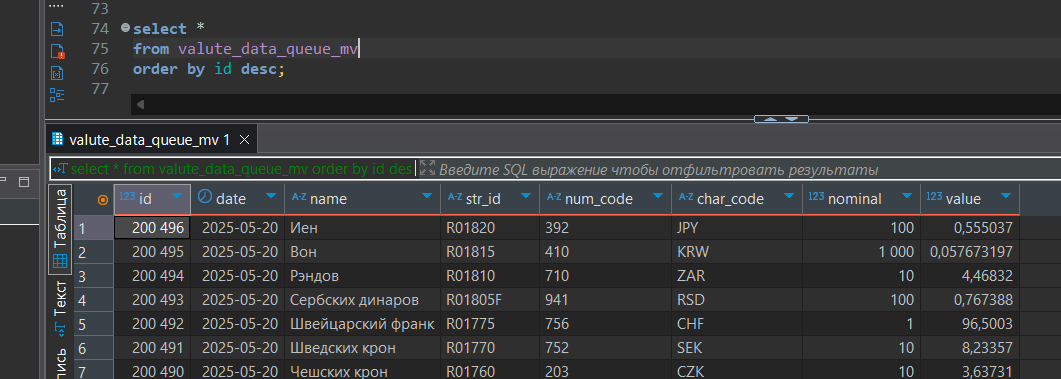


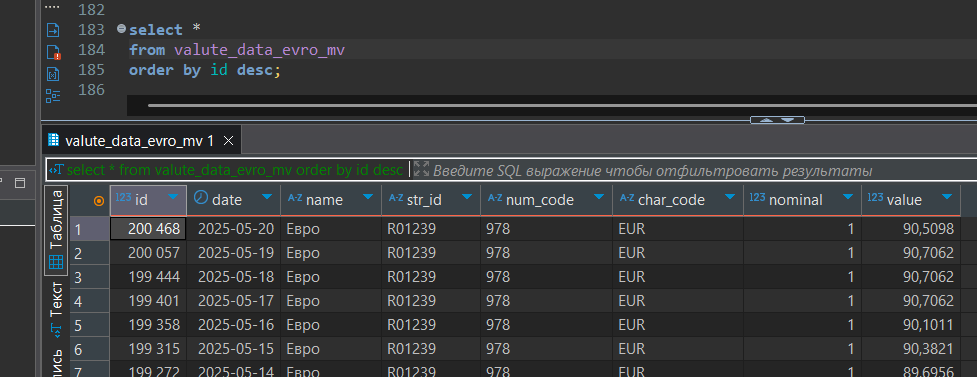
Данные в топике kafka-ui:

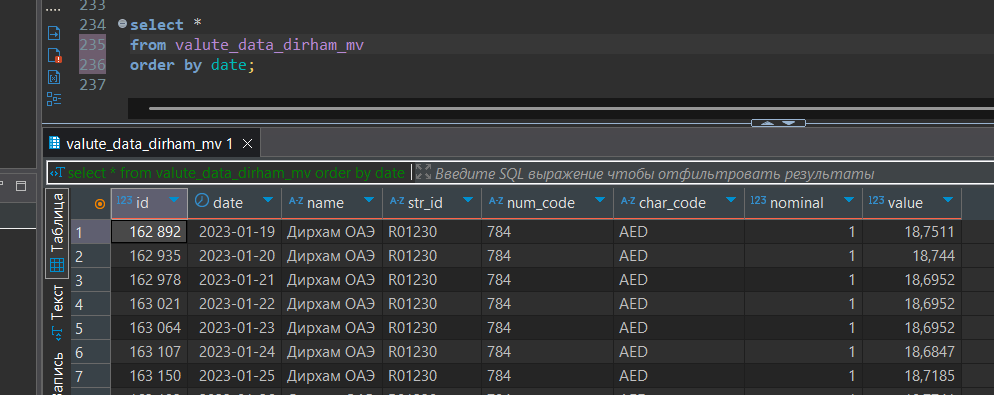


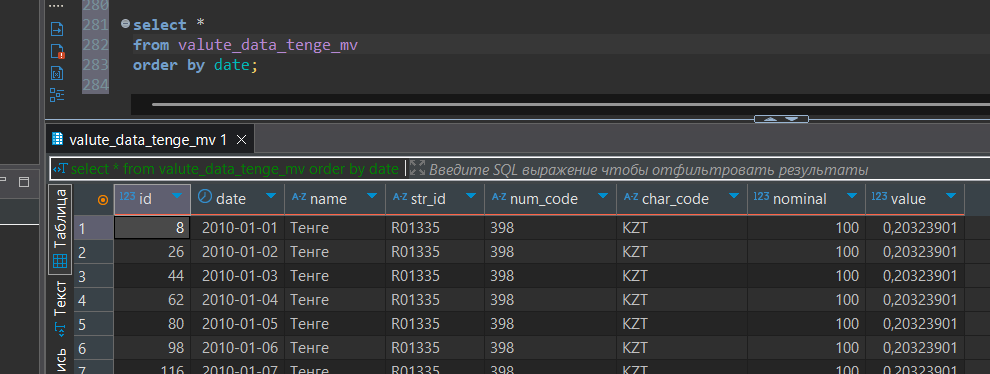


Данные в clickhouse:









1. **Создание графиков.**

После разворачивания инфраструктуры заходим в Superset и создаем графики.

Графики с отдельными вьюхами:

- курс Евро с 2010 года;

- курс Тенге с 2010 года;

- курс Дирхам ОАЭ с 2010 года;

Графики на общей вьюхе:

- курсы Доллара США и Австралийского доллара;

- курс Вон, минимальный и максимальные значения за месяц;

- курсы Норвежской кроны, Датской кроны и Белорусских рублей.

