Apache Sedona - Introduction and Setup with Docker

Pia Bereuter, Robert Wüest

25.11.2025

Inhaltsverzeichnis

The following practical provides an introduction to Apache Sedona and shows how to set it up with a local docker container with Apache Spark and a postgreSQL database with PostGIS extension.

Apache Sedona (previously GeoSpark) is a geospatial engine for distributed computing that integrates with Apache Spark. It allow to process large geospatial datasets across clusters, integrates Spark SQL layer, provides similar queries like PostGIS, implements spatial operations and geometry based on the Java Topology Suite (JTS), provides spatial indexing (quadtree, R-tree) and batch and streaming spatial analytics.

– URL: https://sedona.apache.org/latest/
– Dokumentation: https://sedona.apache.org/latest/api/sql/Overview/

Objectives of this tutorial:
In this tutorial you learn
– how to setup and run Apache Sedona with a local docker container
– how to perform basic geospatial operations with Apache Sedona
– how to import and export from PostGIS to Apache Sedona
– how to read and visualize geospatial data with Apache Sedona

Docker File Structure:

```
docker_sedona
│ .env
│ docker-compose(local postgres).yml
│ docker-compose.yml
│ test-data.sql
├──jars
│     postgresql-42.7.4.jar
└──notebooks
    │   Sedona PostGIS.ipynb (notebooks visible to sedona container)
    └──data
          dataset.parquet (data used in notebooks relative path ”data/” )
```

1 Prerequisites

Download the .zip file containing the necessary files for this tutorial: `E11_Sedona.zip`

- `docker-compose.yml` files to setup up the docker container. For those with a local postreSQL PostGIS installation, there is also a `docker-compose(local postgres).yml` file that does not include PostgreSQL.
- `.env` file with environment variables for the docker container.
- `jars/postgresql-42.7.4.jar` with the PostgreSQL JDBC driver to connect Apache Sedona with PostgreSQL.
- `notebooks` folder with the Jupyter notebooks and a `data` folder containing sample data for the exercises.
- `test-data.sql` sql file with sample data for PostGIS.

2 Setup Apache Sedona with Docker

Install Docker Desktop on your device. Docker provides an environment to run applications in isolated virtual containers, simplifying software deployment and dependences, simpliifying the setup of Apache Sedona and PostgreSQL with PostGIS. For this tutorial we use a pre-configured docker image with Apache Sedona and PostGIS.

Unzip `E11_Sedona.zip` and move the content to your working directory. Open a terminal and navigate your workding directory (root). Check if docker is working with `docker --version`.

```
# Go to your working directory
cd path/to/E11_Sedona
# Check if docker is installed
# you can also run ”docker” or ”docker compose” to see if the commands are available.
docker --version
```

Set the environment variables

Open the `.env` file and check if the environment variables `POSTGRES_USER`, `POSTGRES_PASSWORD` and `POSTGRES_DB` are set. The variables are set in the `.env` files to avoid hardcoding sensitive information in the `docker-compose.yml` file.

Listing 1 `.env`

```
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres
POSTGRES_DB=gis
```

The environment variables of the `.env` are referenced in the `docker-compose.yml` file as `${VARIABLE_NAME}` or with a default variable `${VARIABLE_NAME:-default_value}`. The variables for PostGIS are defined under `environment` of the `postgis` image in the `docker-compose.yml` file. Note, not to use default values for database credentials in general.

Load the JDBC driver

The Sedona requires PostgreSQL JDBC driver to connect Apache Sedona with PostgreSQL. The docker compose file defines in the volumes section, how the driver is mapped (mounted) into the sedona container `./jars:/opt/jars`. Hence, it looks for the JDBC diver in the `./jars` folder of the workind directory.

```
# Download the PostgreSQL JDBC driver
# Create the jars folder in your working directory if it doesn't exist
mkdir -p jars
# Download the driver into the jars folder
curl -L -o jars/postgresql-42.7.4.jar
↪  https://repo1.maven.org/maven2/org/postgresql/postgresql/42.7.4/postgresql-42.7.4.jar
```

Start the docker container

Run the following command from your project folder (root) to start the docker container.

```
# start docker compose from the project root and run:
docker compose up -d
# shutdown docker and remove volumes
# (in case you need to run the whole container again)
# docker-compose down -v
```

The docker compose command pulls (downloads) the docker images and sets up the containers in the background (`-d` detaiched mode), this may takes a while.

Hinweis

If you have a local PostGIS instance on your computer running, use the `docker-compose(local postgres).yml` as your `docker-compose.yml` file to start only the Sedona container without PostgreSQL or stop your local PostGIS Service before you start the docker container. In the Windows menu search for “Service” or “Dienste” open it as an administrator and stop the postgreSQL service.

```
# Check log files for both services (exit with ctrl+c)
docker compose logs -f db    # PostGIS
docker compose logs -f sedona  # Sedona / Jupyter
# Check if PostGIS is ready:
# messsage: /var/run/postgresql:5432 - accepting connections
docker compose exec db pg_isready -U ”$POSTGRES_USER”
# otherwise wait for PostGIS to start and check with:
docker compose logs db
```

You can also open Docker Desktop and verify if the containers are running.

3 Import sample data into PostGIS

Make sure that you’re not already running another PostGIS instance on your computer (see above). The following command imports the `test-data.sql` file into the PostGIS database using the varibles defined in the `.env` file.

```
# Command using .env variable
docker compose exec -T db psql -U $POSTGRES_USER -d $POSTGRES_DB < test-data.sql
# Command using username db name
# username: postgres , database: gis
docker compose exec -T db psql -U postgres -d gis < test-data.sql
```

Verify database name “gis” and generated tables.

```
# Connect to PostgreSQL container
docker exec -it sedona-postgis psql -U postgres
# in postgres list the databases
\l
# connect to the gis database
\c gis
# among other tables you should see the following tables
# demo_points, demo_lines, demo_polygons`, demo_multipolygons
# list the tables
\dt
# query with SQL a list of databases
```

```
SELECT datname FROM pg_database WHERE datistemplate = false;
# quit with
\q
```

Connect directly to the created *gis* database and list the tables with sql queries.

```
docker exec -it sedona-postgis psql -U postgres -d gis
# list tables
SELECT table_name FROM information_schema.tables WHERE table_schema = 'public' ORDER BY
↪  table_name;
# list the number of entries for each table
SELECT COUNT(*) FROM demo_points;
SELECT COUNT(*) FROM demo_lines;
SELECT COUNT(*) FROM demo_polygons;
SELECT COUNT(*) FROM demo_multipolygons;
```

4 Jupyter Lab on Apache Sedona

Open Jupyter Lab on http://localhost:8888. The docker compose file exposes this by default. If a credential token is required you find it in the `sedona` service logs. The notebook runs in Jupyter inside the sedona container. You can copy the notebooks from / to the notebooks folder in your working directory.

```
# Service logs for the sedona container
docker compose logs -f sedona
```

Access Jupyter Lab from VS Code

Open VS Code and set your project directory as the working directory. Open the Jupyter notebook with the *introduction to Apache Sedona* in VS Code (you may need to install the Jupyter extension in VS Code). On the top right press “select kernel” or try to run the first code cell. You’ll be prompted to select a kernel, select *Existing Jupyter Server…*, enter the URL to the Jupyter server at `http://localhost:8888`, press yes to confirm the http server, set a name “localhost” and select the Python 3 kernel.

Now head over and continue in the juypter notebook.