Theoretical Project: Apache Sedona

Advanced Data Management, May 2024

Ruben Rodrigues

Department of Electrical and Computer Engineering

University of Thessaly, Volos

Apache Sedona

Abstract: This report presents a tutorial on Apache Sedona, a spatial data processing framework. We begin by outlining the objectives of Apache Sedona, followed by an exploration of its advantages and specific use cases. We then delve into the primary features of Apache Sedona and describe the suitable data types for storage and management within the framework. Hardware requirements, licensing options, and installation costs for Apache Sedona are analyzed.

Additionally, detailed instructions for installing Apache Sedona are provided, accompanied by screenshots illustrating its functionality, each accompanied by a brief explanation. Finally, we enumerate the sources from which the information was obtained.

1. Apache Sedona Aim

Apache Sedona™ is a cluster computing system for processing large-scale spatial data[1]. Sedona extends existing cluster computing systems, such as Apache Spark, Apache Flink, and Snowflake, with a set of out-of-the-box distributed Spatial Datasets and Spatial SQL that efficiently load, process, and analyze large-scale spatial data across machines.

The primary objective of Apache Sedona is to provide a scalable and efficient framework for processing huge amounts of spatial data within the Apache Spark ecosystem.

By incorporating spatial data processing capabilities into Apache Spark, Apache Sedona promises to enable users to execute complex geospatial analytics, spatial queries, and visualization activities on large datasets. Its purpose is to make it easier to construct geospatial applications in a variety of fields, including urban planning, transportation, environmental monitoring, and location services.

Sedona offers Scala, Java, Spatial SQL, Python, and R APIs and integrates them into underlying system kernels with care. You can simply create spatial analytics and data mining applications and run them in any cloud environment.

Furthermore, Apache Sedona seeks to provide ease of use, easy integration with existing Spark workflows, and community-driven development to meet the growing demand for geographic data processing and analysis.

1- Spatial data can be referred to as geographic data or geospatial data. Spatial data provides information that identifies the location of features and boundaries on Earth. Spatial data can be processed and analyzed using Geographical Information Systems (GIS) or Image Processing packages.

2. Advantages and Use Cases

High Speed:

Apache Sedona is optimized for computation-intensive query workloads, delivering significantly faster performance compared to other Spark-based geospatial data systems. With speed improvements ranging from 2X to 10X, users can execute spatial queries and analytics tasks more efficiently, leading to reduced processing times and faster insights.

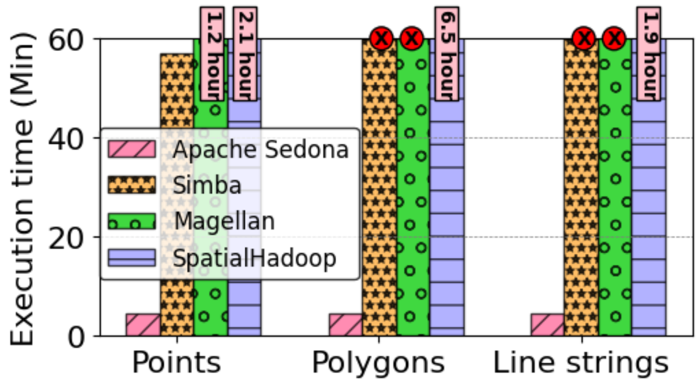


Figure 1. Apache Sedona speed benchmark

Low Memory Consumption:

Apache Sedona boasts 50% less peak memory consumption compared to other Spark-based geospatial data systems for large-scale in-memory query processing. This translates to more efficient resource utilization, enabling users to process and analyze large spatial datasets with lower hardware requirements and reduced costs.

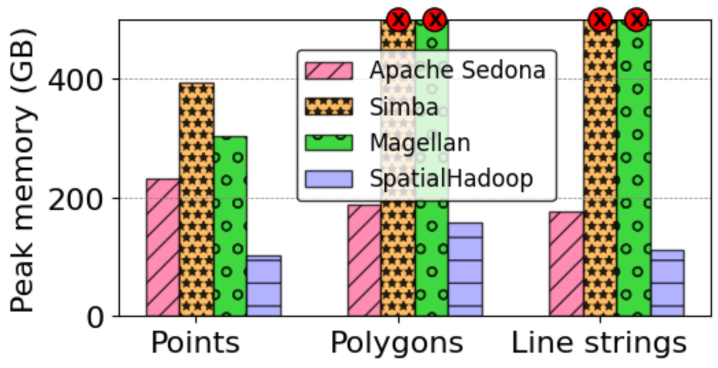


Figure 2. Peak memory consumption benchmark

Ease of Use:

Apache Sedona provides a user-friendly experience with support for multiple programming languages, including Scala, Java, Spatial SQL, Python, and R APIs.

By integrating these APIs seamlessly into the underlying system kernels, Sedona enables developers and data scientists to create spatial analytics and data mining applications effortlessly. Moreover, its compatibility with various cloud environments ensures flexibility and scalability for deploying spatial applications across different infrastructures.

Integration with Existing Spark Ecosystem:

Apache Sedona seamlessly integrates with the broader Apache Spark ecosystem, enabling users to leverage existing Spark components and libraries for their spatial data processing tasks.

This integration simplifies development and deployment workflows, as users can reuse their knowledge, skills, and infrastructure already built around Spark, thereby reducing the learning curve and development overhead associated with adopting a new spatial data processing framework.

Scalability:

Apache Sedona is built on top of Apache Spark, a widely adopted distributed computing framework known for its scalability. As a result, Sedona inherits Spark's scalability capabilities, allowing users to scale their spatial data processing workflows horizontally across large clusters of machines.

This scalability is crucial for handling the growing volumes of spatial data generated by modern applications and accommodating increasing computational demands.

Apache Sedona, with its capabilities for processing large-scale spatial data efficiently, finds applications across various domains. Some common use cases of Apache Sedona include:

Urban Planning and Development:

Urban planners use Apache Sedona to analyze spatial data related to demographics, land use, transportation networks, and infrastructure to optimize urban development plans. Sedona can identify optimal locations for new infrastructure projects, assess the impact of proposed developments on traffic flow and environmental factors, and facilitate informed decision-making for sustainable urban growth.

Transportation and Logistics Optimization:

Sedona is used in transportation and logistics to analyze and optimize routing, fleet management, and supply chain operations. It can perform spatial analysis to identify efficient delivery routes, optimize vehicle dispatching, and minimize transportation costs. Sedona's capabilities are also valuable for real-time tracking of vehicles and assets, enabling better fleet management and route optimization.

Environmental Monitoring and Conservation:

Environmental researchers and conservationists leverage Sedona to analyze spatial data related to ecosystems, biodiversity, and natural resources. Sedona can help monitor changes in land cover, analyze habitat suitability for wildlife, and identify areas at risk of environmental degradation. By analyzing spatial data, researchers can make informed decisions to support conservation efforts and sustainable land management practices.

Location-Based Services and Geospatial Intelligence:

Sedona powers location-based services and geospatial intelligence applications that provide users with contextual information based on their geographic location. These applications include location-based advertising, geotagging, geofencing, and personalized recommendations based on spatial data analysis. Sedona's fast and scalable processing capabilities enable real-time analysis of location data, enhancing user experiences and engagement.

Agricultural Monitoring and Precision Farming:

In agriculture, Sedona is used for precision farming applications, where spatial data analysis helps optimize crop yield, water usage, and resource allocation. Sedona can analyze soil characteristics, weather patterns, and satellite imagery to generate insights for precision irrigation, crop planning, and disease detection. By applying spatial analysis techniques, farmers can improve productivity, reduce input costs, and minimize environmental impact.

Emergency Response and Disaster Management:

Sedona plays a crucial role in emergency response and disaster management by analyzing spatial data to assess risks, plan evacuation routes, and coordinate response efforts. Sedona can integrate data from various sources, such as satellite imagery, weather forecasts, and demographic data, to create spatial models for predicting disaster scenarios and mitigating their impact. During emergencies, Sedona enables real-time monitoring and analysis of spatial data to support decision-making by emergency responders and authorities.

3. Types of Data

Apache Sedona's support for various data formats such as shapefile, **GeoJSON**, Well-Known Text (**WKT**), Well-Known Binary (**WKB**), **PostGIS**, and **Spatial Parquet** enables users to load spatial data in these formats seamlessly into their processing pipelines.

This flexibility allows users to work with diverse spatial datasets sourced from different data providers, databases, or file formats, facilitating interoperability and integration across spatial data workflows.

Some of the common types of spatial data that can be used in Apache Sedona include:

Point Data:

Point data represents individual geographic points defined by their coordinates (latitude and longitude). Examples include locations of landmarks, addresses, or GPS coordinates of vehicles.

Line Data:

Line data consists of linear features defined by sequences of connected points. Examples include roads, rivers, pipelines, or flight paths.

Polygon Data:

Polygon data represents closed geometric shapes with multiple interconnected points, enclosing an area. Examples include administrative boundaries, land parcels, or building footprints.

Multi-Point Data:

Multi-point data comprises collections of individual points grouped together. It represents sets of spatially related points. Examples include clusters of GPS coordinates or sensor measurements.

Multi-Line Data:

Multi-line data consists of collections of linear features grouped together. It represents sets of spatially related linear features. Examples include transportation networks or utility lines.

Multi-Polygon Data:

Multi-polygon data comprises collections of polygons grouped together. It represents sets of spatially related areas. Examples include complex administrative boundaries or geological formations.

Geometry Collections:

Geometry collections can include a combination of points, lines, and polygons grouped together. They allow for representing heterogeneous spatial data sets. Examples include mixed-use development areas or complex geographic features.

4. Requirements for installation and hardware

The hardware requirements for Apache Sedona depend on the scale of your spatial data processing tasks and the size of your datasets. Generally, Apache Sedona can run on standard hardware configurations commonly used for Apache Spark deployments.

CPU:

Apache Sedona benefits from having multiple CPU cores for parallel processing, like Apache Spark. It's recommended to have multi-core CPUs to take advantage of parallelism, especially for processing large datasets.

RAM Memory:

The amount of RAM required depends on the size of the datasets you're processing and the complexity of the spatial operations you're performing. In general, having more RAM allows for better performance, especially when working with large datasets that need to be processed in memory. It's advisable to allocate sufficient memory to accommodate both the data and the processing tasks.

Storage:

Apache Sedona doesn't have specific storage requirements beyond those of Apache Spark. You'll need storage space to store your spatial datasets and any intermediate data generated during processing. The amount of storage required depends on the size and number of datasets you're working with.

Operating System:

Apache Sedona is compatible with various operating systems, including Linux, macOS, and Windows.

Dependencies:

Java Runtime Environment (JRE) or Java Development Kit (JDK), Apache Spark. Any other additional libraries or packages may be required depending on the specific use case.

5. Potential Costs Attached

Apache Sedona is an open-source project released under the Apache License 2.0, which means it is free to use, modify, and distribute without any licensing costs.

As an open-source project, Apache Sedona is freely available for both commercial and non-commercial use, and there are no direct costs associated with acquiring or using the software.

6. Guidelines for Installation

To get started with Apache Sedona, follow these steps:

Install Jupyter Notebook or Jupyter Lab:

Before proceeding, ensure you have Jupyter Notebook or Jupyter Lab installed on your system. These are popular interactive computing environments for Python that allow you to create and share documents containing live code, equations, visualizations, and narrative text.

Preferably Install Docker:

Docker is a containerization platform that simplifies the deployment and management of applications by packaging them into standardized units called containers. Installing Docker is preferred for running Apache Sedona in a containerized environment, providing ease of setup and portability across different systems. Why Docker? Docker containers ensure consistency in the runtime environment, making it easier to reproduce the Apache Sedona environment across different systems without worrying about software dependencies and configuration issues.

Search for Apache Sedona Image on Docker Hub:

After installing Docker, navigate to Docker Hub and search for the Apache Sedona image. You can find the image by searching for ***"apache/sedona:latest"*** or ***"apache/sedona:1.5.1"***.

Pull and Run the Apache Sedona Container:

Once you've found the Apache Sedona image, pull it using the following command:

“docker pull apache/sedona:latest”

After pulling the image, run the container using the following command:

“docker run -p 8888:8888 -p 8080:8080 -p 8081:8081 -p 4040:4040 apache/sedona:latest”

This command will start the Apache Sedona container and expose the necessary ports for accessing the Jupyter Lab interface.

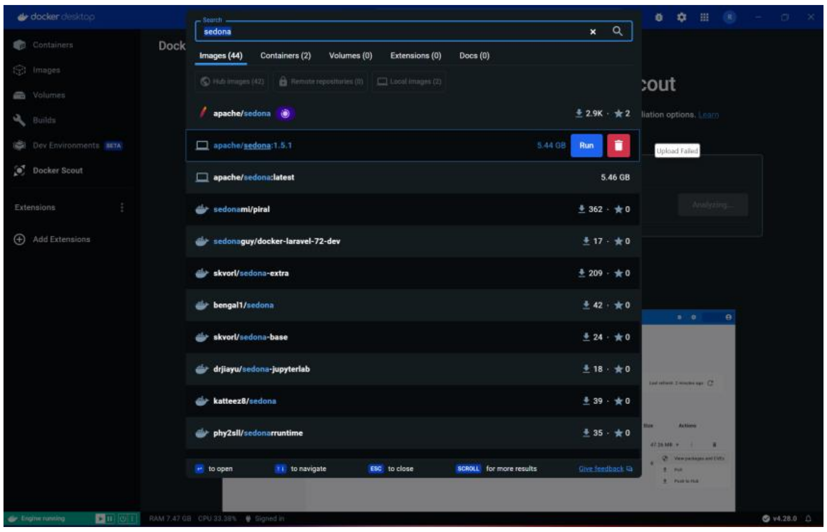
Access Jupyter Lab:

After running the container, you'll receive a link to access Jupyter Lab in your terminal. Open the provided link in your web browser to access the Jupyter Lab interface.

Start Coding or Explore Examples:

You can now start coding with Apache Sedona in Jupyter Lab. Alternatively, you can explore the provided examples and documentation available within the Jupyter Lab environment to get familiar with Apache Sedona's functionalities and capabilities.

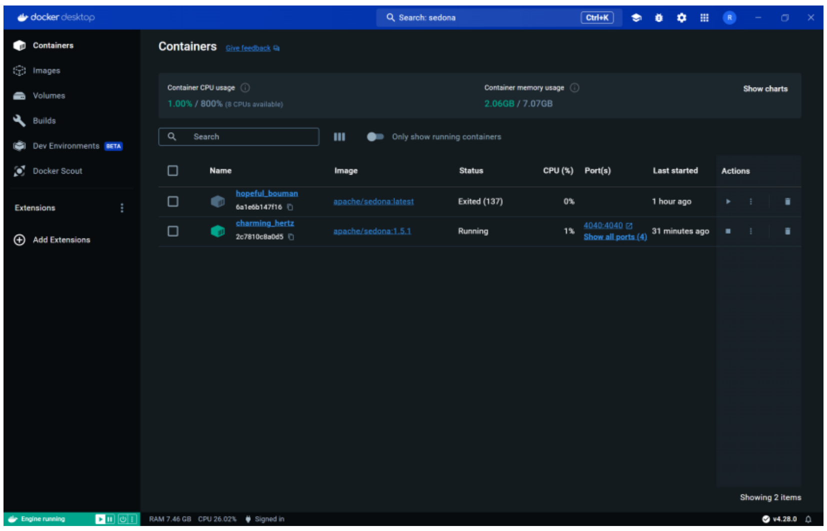
7. Examples:



sedona

Images (44) Containers (2) Volumes (0) Extensions (0) Docs (0)

apache/sedona	2.9K · ★ 2
apache/sedona:1.5.1	5.44 GB
apache/sedona:latest	5.46 GB
sedonaml/piral	362 · ★ 0
sedonaguy/docker-laravel-72-dev	17 · ★ 0
skvorl/sedona-extra	209 · ★ 0
bengal1/sedona	42 · ★ 0
skvorl/sedona-base	24 · ★ 0
drjiayu/sedona-jupyterlab	18 · ★ 0
katteez8/sedona	39 · ★ 0
phy2sll/sedonarruntime	35 · ★ 0



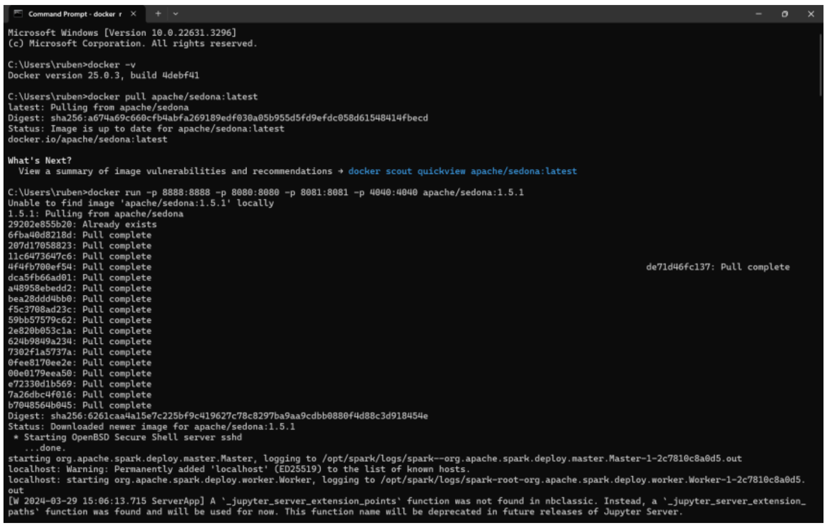
Containers

Container CPU usage 1.00% / 800% (8 CPUs available)

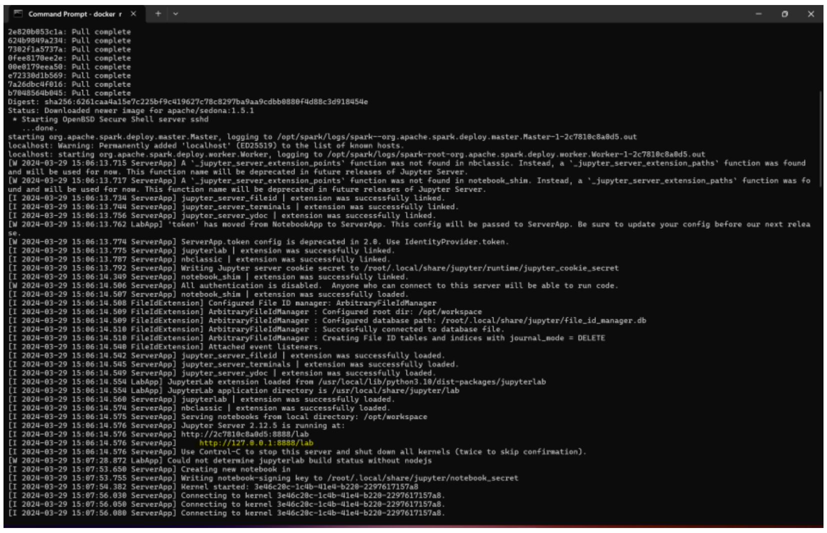
Container memory usage 2.06GB / 7.07GB

Name	Image	Status	CPU (%)	Port(s)	Last started
hopeful_bouman 6a1e6b147f16	apache/sedona:latest	Exited (137)	0%		1 hour ago
charming_hertz 2c7810c8a0d5	apache/sedona:1.5.1	Running	1%	4040:4040 Show all ports (4)	31 minutes ago

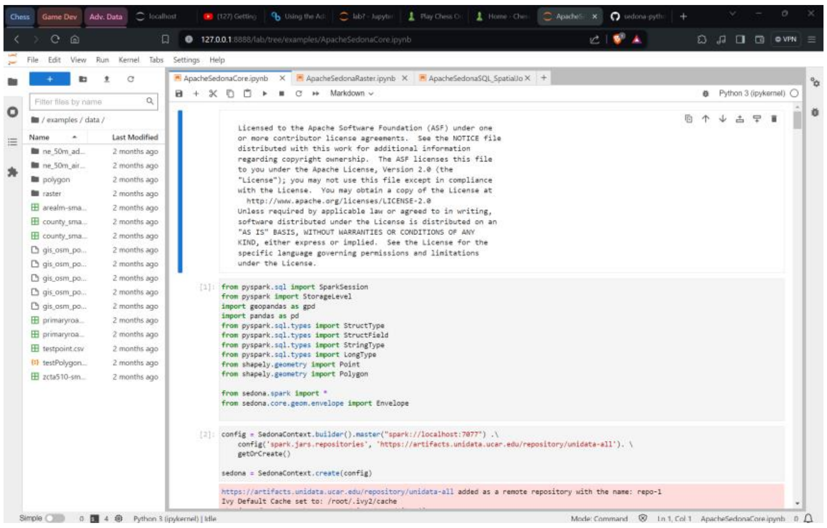
Showing 2 items



```
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ruben>docker -v
Docker version 25.0.3, build 4debf41

C:\Users\ruben>docker pull apache/sedona:latest
latest: Pulling from apache/sedona
Digest: sha256:a674a69c660cfb4abfa269189edf030a05b955d5fd9efdc058d61548414fbecd
Status: Image is up to date for apache/sedona:latest
docker.io/apache/sedona:latest

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview apache/sedona:latest

C:\Users\ruben>docker run -p 8888:8888 -p 8080:8080 -p 8081:8081 -p 4040:4040 apache/sedona:1.5.1
Unable to find image 'apache/sedona:1.5.1' locally
1.5.1: Pulling from apache/sedona
29202e855b20: Already exists
6fba40d8218d: Pull complete
207d17058823: Pull complete
11c6473647c6: Pull complete
4f4fb700ef54: Pull complete                                                            de71d46fc137: Pull complete
dca5fb66ad01: Pull complete
a48958ebedd2: Pull complete
bea28ddd4bb0: Pull complete
f5c3708ad23c: Pull complete
59bb57579c62: Pull complete
2e820b053c1a: Pull complete
624b9849a234: Pull complete
7302f1a5737a: Pull complete
0fee8170ee2e: Pull complete
00e0179eea50: Pull complete
e72330d1b569: Pull complete
7a26dbc4f016: Pull complete
b7048564b045: Pull complete
Digest: sha256:6261caa4a15e7c225bf9c419627c78c8297ba9aa9cdbb0880f4d88c3d918454e
Status: Downloaded newer image for apache/sedona:1.5.1
 * Starting OpenBSD Secure Shell server sshd
   ...done.
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/logs/spark--org.apache.spark.deploy.master.Master-1-2c7810c8a0d5.out
localhost: Warning: Permanently added 'localhost' (ED25519) to the list of known hosts.
localhost: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-2c7810c8a0d5.
out
[W 2024-03-29 15:06:13.715 ServerApp] A `_jupyter_server_extension_points` function was not found in nbclassic. Instead, a `_jupyter_server_extension_
paths` function was found and will be used for now. This function name will be deprecated in future releases of Jupyter Server.
```

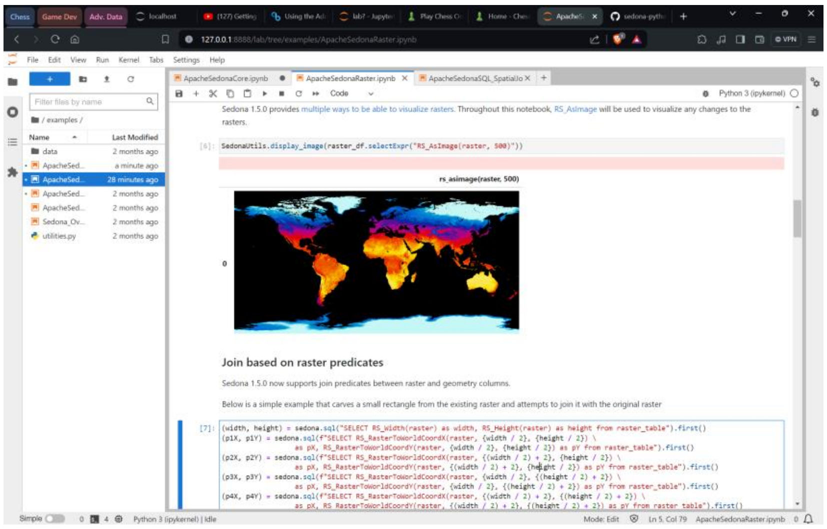


```
2e820b853c1a: Pull complete
624b9849a234: Pull complete
7302f1a5737a: Pull complete
0fee8170ee2e: Pull complete
00e0179eea50: Pull complete
e72330d1b569: Pull complete
7a26dbc4f016: Pull complete
b7048564b045: Pull complete
Digest: sha256:6261caa4a15e7c225bf9c419627c78c8297ba9aa9cdbb0880f4d88c3d918454e
Status: Downloaded newer image for apache/sedona:1.5.1
 * Starting OpenBSD Secure Shell server sshd
   ...done.
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/logs/spark--org.apache.spark.deploy.master.Master-1-2c7810c8a0d5.out
localhost: Warning: Permanently added 'localhost' (ED25519) to the list of known hosts.
localhost: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-2c7810c8a0d5.out
[W 2024-03-29 15:06:13.715 ServerApp] A `_jupyter_server_extension_points` function was not found in nbclassic. Instead, a `_jupyter_server_extension_paths` function was found
and will be used for now. This function name will be deprecated in future releases of Jupyter Server.
[W 2024-03-29 15:06:13.717 ServerApp] A `_jupyter_server_extension_points` function was not found in notebook_shim. Instead, a `_jupyter_server_extension_paths` function was fo
und and will be used for now. This function name will be deprecated in future releases of Jupyter Server.
[I 2024-03-29 15:06:13.734 ServerApp] jupyter_server_fileid | extension was successfully linked.
[I 2024-03-29 15:06:13.744 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2024-03-29 15:06:13.756 ServerApp] jupyter_server_ydoc | extension was successfully linked.
[W 2024-03-29 15:06:13.762 LabApp] 'token' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before our next relea
se.
[W 2024-03-29 15:06:13.774 ServerApp] ServerApp.token config is deprecated in 2.0. Use IdentityProvider.token.
[I 2024-03-29 15:06:13.775 ServerApp] jupyterlab | extension was successfully linked.
[I 2024-03-29 15:06:13.787 ServerApp] nbclassic | extension was successfully linked.
[I 2024-03-29 15:06:13.792 ServerApp] Writing Jupyter server cookie secret to /root/.local/share/jupyter/runtime/jupyter_cookie_secret
[I 2024-03-29 15:06:14.349 ServerApp] notebook_shim | extension was successfully linked.
[W 2024-03-29 15:06:14.506 ServerApp] All authentication is disabled.  Anyone who can connect to this server will be able to run code.
[I 2024-03-29 15:06:14.507 ServerApp] notebook_shim | extension was successfully loaded.
[I 2024-03-29 15:06:14.508 FileIdExtension] Configured File ID manager: ArbitraryFileIdManager
[I 2024-03-29 15:06:14.509 FileIdExtension] ArbitraryFileIdManager : Configured root dir: /opt/workspace
[I 2024-03-29 15:06:14.509 FileIdExtension] ArbitraryFileIdManager : Configured database path: /root/.local/share/jupyter/file_id_manager.db
[I 2024-03-29 15:06:14.510 FileIdExtension] ArbitraryFileIdManager : Successfully connected to database file.
[I 2024-03-29 15:06:14.510 FileIdExtension] ArbitraryFileIdManager : Creating File ID tables and indices with journal_mode = DELETE
[I 2024-03-29 15:06:14.540 FileIdExtension] Attached event listeners.
[I 2024-03-29 15:06:14.542 ServerApp] jupyter_server_fileid | extension was successfully loaded.
[I 2024-03-29 15:06:14.545 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2024-03-29 15:06:14.549 ServerApp] jupyter_server_ydoc | extension was successfully loaded.
[I 2024-03-29 15:06:14.554 LabApp] JupyterLab extension loaded from /usr/local/lib/python3.10/dist-packages/jupyterlab
[I 2024-03-29 15:06:14.554 LabApp] JupyterLab application directory is /usr/local/share/jupyter/lab
[I 2024-03-29 15:06:14.560 ServerApp] jupyterlab | extension was successfully loaded.
[I 2024-03-29 15:06:14.574 ServerApp] nbclassic | extension was successfully loaded.
[I 2024-03-29 15:06:14.575 ServerApp] Serving notebooks from local directory: /opt/workspace
[I 2024-03-29 15:06:14.576 ServerApp] Jupyter Server 2.12.5 is running at:
[I 2024-03-29 15:06:14.576 ServerApp] http://2c7810c8a0d5:8888/lab
[I 2024-03-29 15:06:14.576 ServerApp]     http://127.0.0.1:8888/lab
[I 2024-03-29 15:06:14.576 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 2024-03-29 15:07:28.872 LabApp] Could not determine jupyterlab build status without nodejs
[I 2024-03-29 15:07:53.650 ServerApp] Creating new notebook in
[I 2024-03-29 15:07:53.755 ServerApp] Writing notebook-signing key to /root/.local/share/jupyter/notebook_secret
[I 2024-03-29 15:07:54.382 ServerApp] Kernel started: 3e46c20c-1c4b-41e4-b220-2297617157a8
[I 2024-03-29 15:07:56.030 ServerApp] Connecting to kernel 3e46c20c-1c4b-41e4-b220-2297617157a8.
[I 2024-03-29 15:07:56.050 ServerApp] Connecting to kernel 3e46c20c-1c4b-41e4-b220-2297617157a8.
[I 2024-03-29 15:07:56.080 ServerApp] Connecting to kernel 3e46c20c-1c4b-41e4-b220-2297617157a8.
```


127.0.0.1:8888/lab/tree/examples/ApacheSedonaCore.ipynb
File Edit View Run Kernel Tabs Settings Help
ApacheSedonaCore.ipynb
ApacheSedonaRaster.ipynb
ApacheSedonaSQL_SpatialJo
Markdown
Python 3 (ipykernel)
/ examples / data /
Name
Last Modified
Simple
Python 3 (ipykernel) | Idle
Mode: Command
Ln 1, Col 1

```
Licensed to the Apache Software Foundation (ASF) under one
or more contributor license agreements.  See the NOTICE file
distributed with this work for additional information
regarding copyright ownership.  The ASF licenses this file
to you under the Apache License, Version 2.0 (the
"License"); you may not use this file except in compliance
with the License.  You may obtain a copy of the License at
  http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing,
software distributed under the License is distributed on an
"AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
KIND, either express or implied.  See the License for the
specific language governing permissions and limitations
under the License.
```

```
[1]: from pyspark.sql import SparkSession
     from pyspark import StorageLevel
     import geopandas as gpd
     import pandas as pd
     from pyspark.sql.types import StructType
     from pyspark.sql.types import StructField
     from pyspark.sql.types import StringType
     from pyspark.sql.types import LongType
     from shapely.geometry import Point
     from shapely.geometry import Polygon

     from sedona.spark import *
     from sedona.core.geom.envelope import Envelope

[2]: config = SedonaContext.builder().master("spark://localhost:7077") .\
         config('spark.jars.repositories', 'https://artifacts.unidata.ucar.edu/repository/unidata-all'). \
         getOrCreate()

     sedona = SedonaContext.create(config)

     https://artifacts.unidata.ucar.edu/repository/unidata-all added as a remote repository with the name: repo-1
     Ivy Default Cache set to: /root/.ivy2/cache
```


127.0.0.1:8888/lab/tree/examples/ApacheSedonaRaster.ipynb
File Edit View Run Kernel Tabs Settings Help
ApacheSedonaCore.ipynb
ApacheSedonaRaster.ipynb
ApacheSedonaSQL_SpatialJo
Code
Python 3 (ipykernel)
/ examples /
Name
Last Modified
rs_asimage(raster, 500)
0
Simple
Python 3 (ipykernel) | Idle
Mode: Edit
Ln 5, Col 79

Sedona 1.5.0 provides multiple ways to be able to visualize rasters. Throughout this notebook, RS_AsImage will be used to visualize any changes to the rasters.

```
[6]: SedonaUtils.display_image(raster_df.selectExpr("RS_AsImage(raster, 500)"))
```

Join based on raster predicates

Sedona 1.5.0 now supports join predicates between raster and geometry columns.

Below is a simple example that carves a small rectangle from the existing raster and attempts to join it with the original raster

```
[7]: (width, height) = sedona.sql("SELECT RS_Width(raster) as width, RS_Height(raster) as height from raster_table").first()
     (p1X, p1Y) = sedona.sql(f"SELECT RS_RasterToWorldCoordX(raster, {width / 2}, {height / 2}) \
                 as pX, RS_RasterToWorldCoordY(raster, {width / 2}, {height / 2}) as pY from raster_table").first()
     (p2X, p2Y) = sedona.sql(f"SELECT RS_RasterToWorldCoordX(raster, {(width / 2) + 2}, {height / 2}) \
                 as pX, RS_RasterToWorldCoordY(raster, {(width / 2) + 2}, {height / 2}) as pY from raster_table").first()
     (p3X, p3Y) = sedona.sql(f"SELECT RS_RasterToWorldCoordX(raster, {width / 2}, {(height / 2) + 2}) \
                 as pX, RS_RasterToWorldCoordY(raster, {width / 2}, {(height / 2) + 2}) as pY from raster_table").first()
     (p4X, p4Y) = sedona.sql(f"SELECT RS_RasterToWorldCoordX(raster, {(width / 2) + 2}, {(height / 2) + 2}) \
                 as pX, RS_RasterToWorldCoordY(raster, {(width / 2) + 2}, {(height / 2) + 2}) as pY from raster_table").first()
```

8. Sources

https://www.youtube.com/watch?v=7MVK90s9Dn8

https://github.com/apache/sedona

https://medium.com/getindata-blog/introduction-to-apache-sedona-incubating-d819cbc0886

https://sedona.apache.org/1.5.1/

https://seattle2023.pydata.org/cfp/talk/87ABHY/

https://blog.disy.net/apache-sedona-environment/

https://www.quora.com/How-is-Apache-Sedona-compared-to-ArcGIS

JupyterLab

Docker

Chat GPT

https://www.youtube.com/watch?v=WZ-vv7c6YSg&t=92s