Automatically shard and scale-out your traditional databases on Kubernetes for true digital transformation

Trista Pan
panjuan@apache.org

Trista Pan

SphereEx Co-Founder & CTO

Apache Member

AWS Data Hero

Tencent Cloud TVP

Apache ShardingSphere PMC

Apache brpc & Apache AGE

& Apache HugeGraph (Incubating) mentor

China Mulan Community Mentor

Bio: https://tristazero.github.io

LinkedIn: https://www.linkedin.com/in/panjuan

GitHub: https://github.com/tristaZero

Twitter: @tristaZero

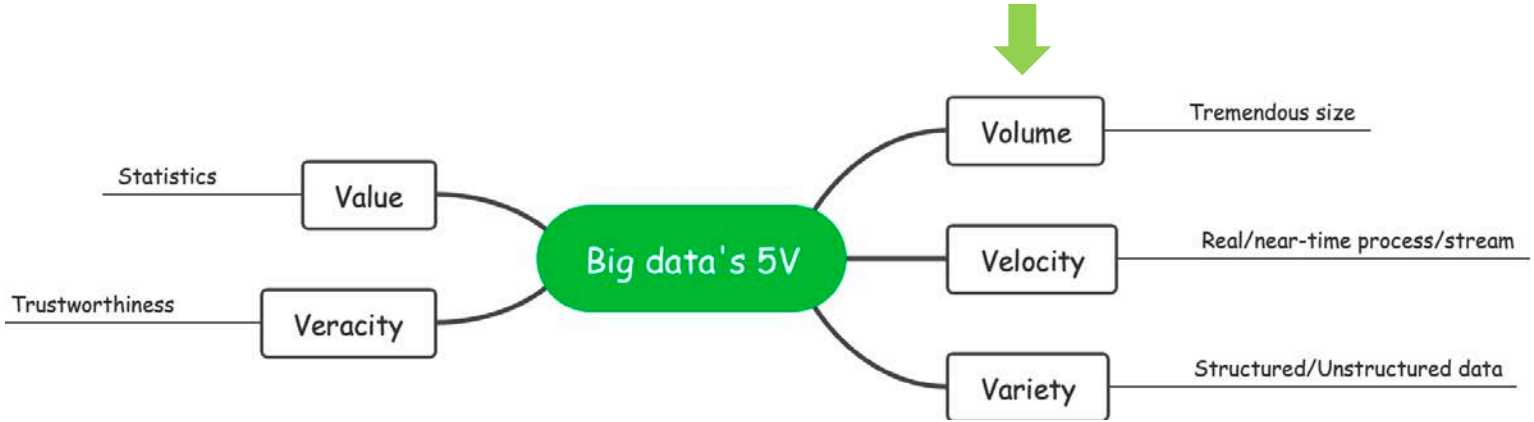
Project Twitter: @ShardingSphere

SphereEx

Content

- ✓ Big data 5’v
- ✓ Distributed database architecture
- ✓ New Idea & solution
- ✓ Demo show

Digital transformation



SphereEx

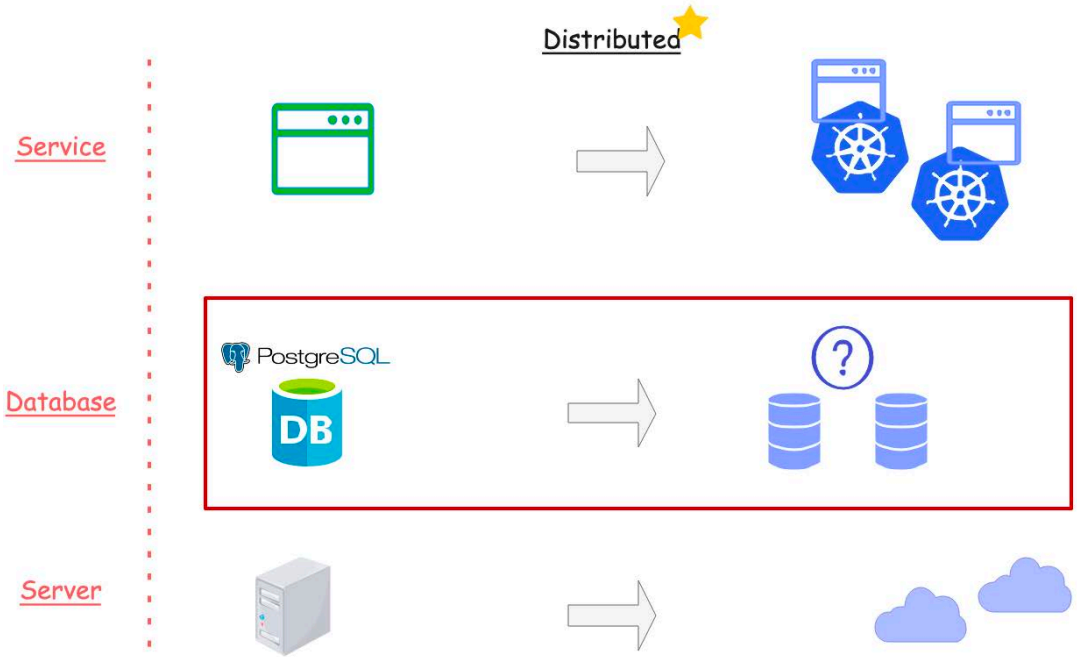
OLTP & OLAP

The main distinction between OLAP vs. OLTP is the core purpose of each system. An OLAP system is designed to process large amounts of data quickly, allowing users to analyze multiple data dimensions in tandem. Teams can use this data for decision-making and problem-solving.

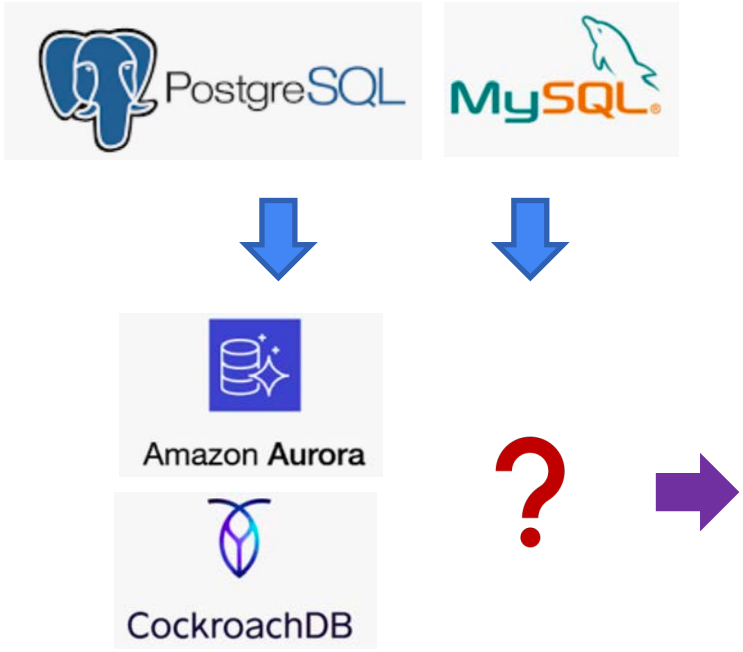
In contrast, OLTP systems are designed to handle large volumes of transactional data involving multiple users. Relational databases rapidly update, insert, or delete small amounts of data in real time. Most OLTP systems are used for executing transactions such as online hotel bookings, mobile banking transactions, ecommerce purchases, and in-store checkout. Many OLAP systems pull their data from OLTP databases via an ETL pipeline and can provide insights such as analyzing ATM activity and performance over time.

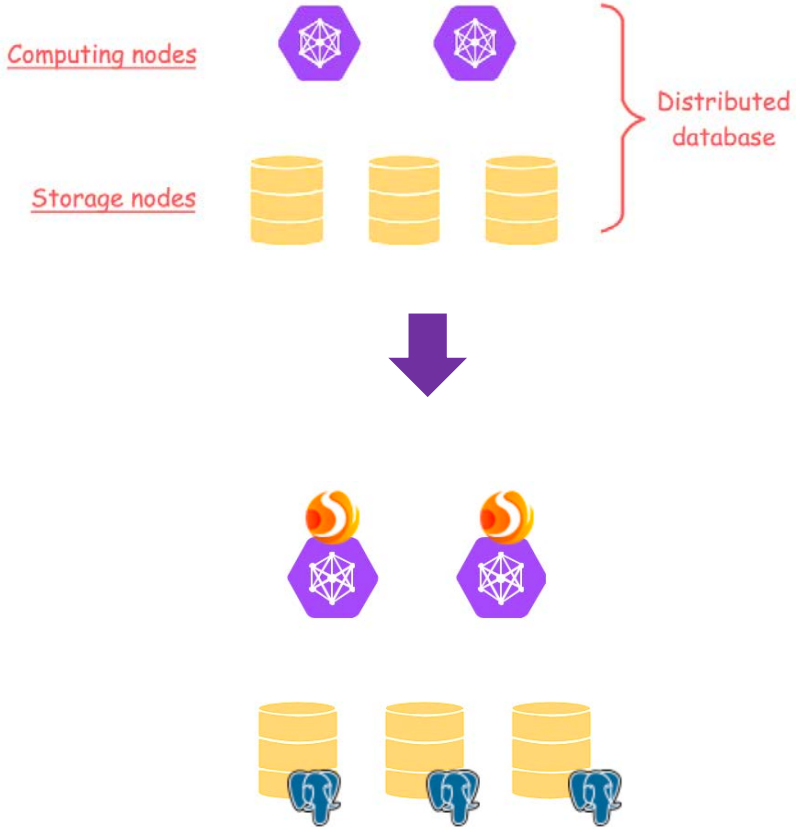
Simply put, organizations use OLTP systems to run their business while OLAP systems help them understand their business.

Distributed database

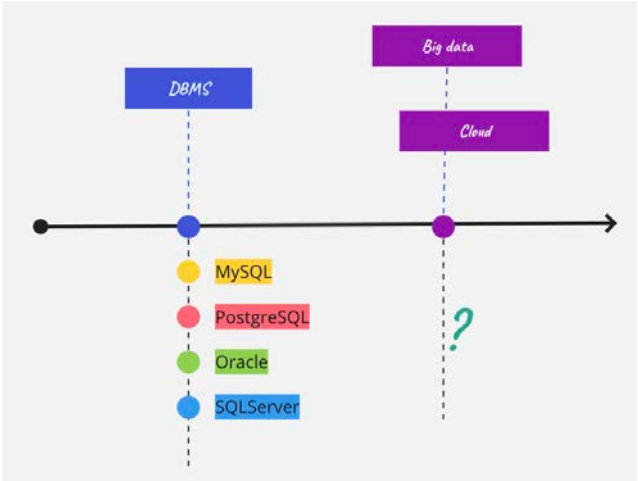


Distributed database



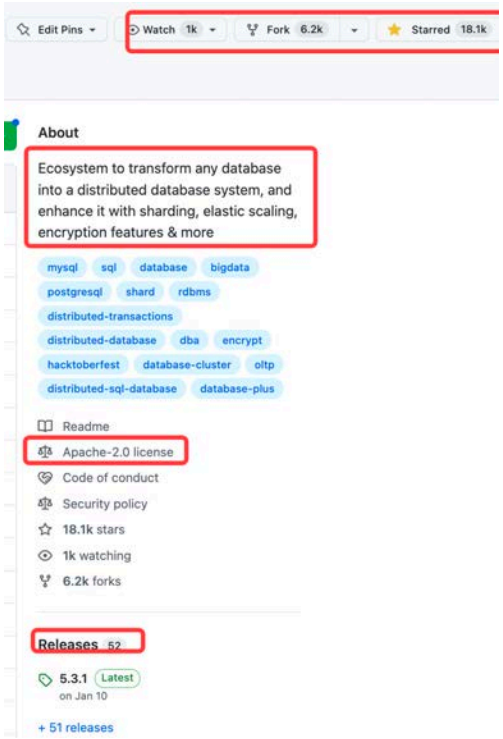


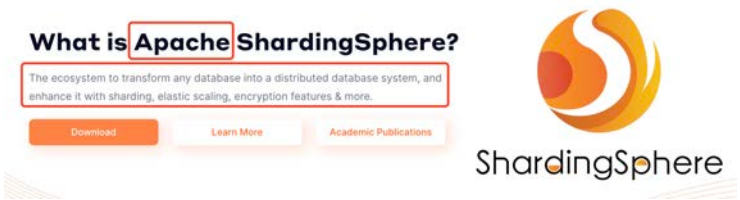
Benefits

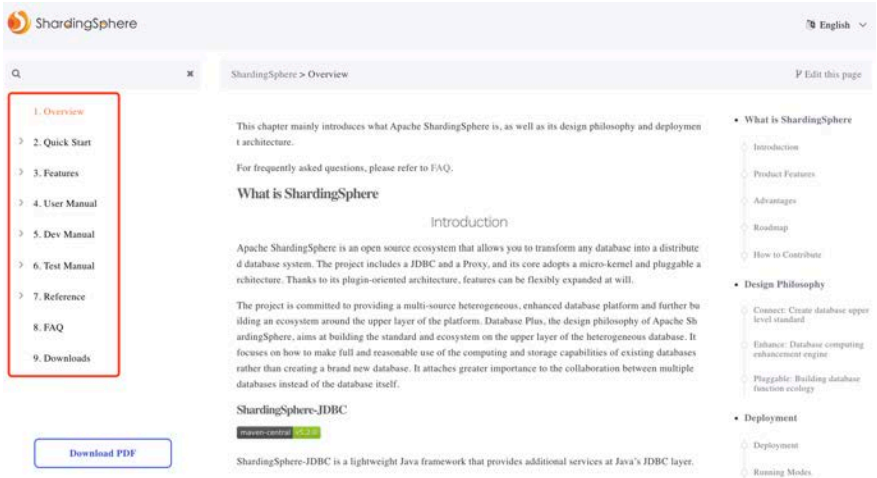


- ✓ Leverage the existing databases
- ✓ Upgrade it into a distributed database at low cost
- ✓ SQL audit & Traffic governance & Elastic scaling
- ✓ Solve the headache of moving database into Kubernetes
- ✓ Out-of-the-box deployment
- ✓ No lock-in

Apache ShardingSphere

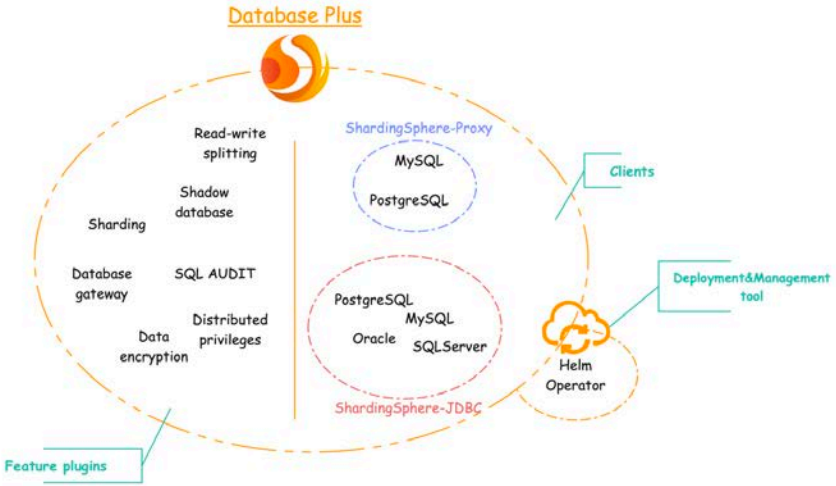


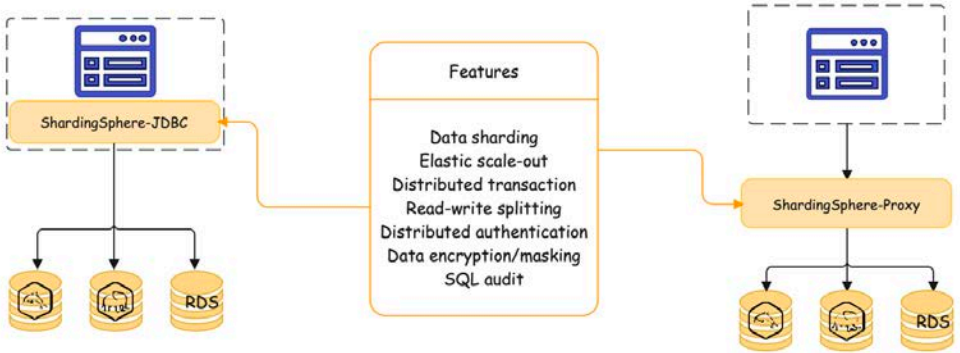




SphereEx

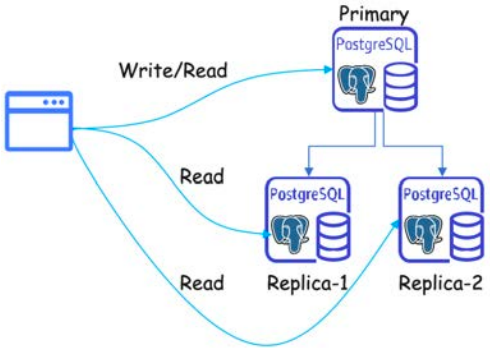
ShardingSphere features



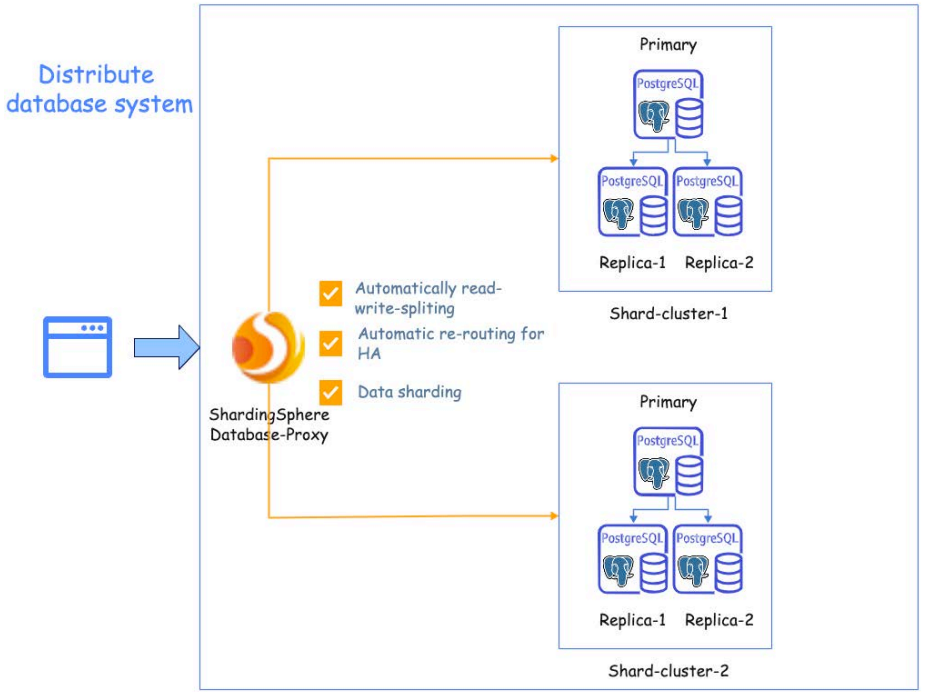


SphereEx

Application -> Database



Before



After

SphereEx

One command to deploy the cluster on Kubernetes

ShardingSphere-on-Cloud

Take Apache ShardingSphere to the cloud

A collection of tools & best practices including automated deployment scripts to virtual machines in AWS, Google Cloud Platform, Alibaba Cloud, CloudFormation Stack templates, and Terraform one-click deployment scripts.

Helm Charts, Operators, automatic horizontal scaling, and other tools for the Kubernetes cloud-native environment are also included.

One-click Kubernetes Deployment
One-click deployment in Kubernetes for ShardingSphere Proxy based on Helm Charts.

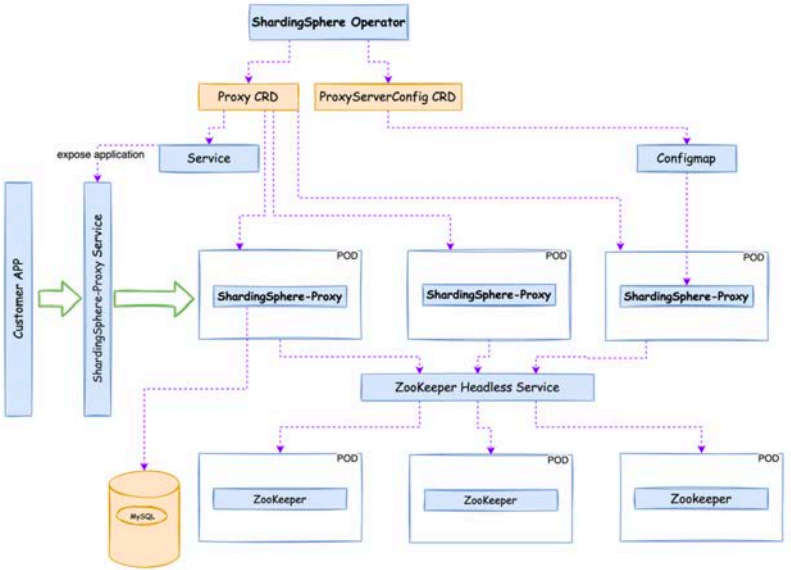
Automatic Kubernetes DevOps
One-click deployment and automatic DevOps in Kubernetes for ShardingSphere Proxy based on Operator.

Rapid CloudFormation Deployment
Rapid deployment of ShardingSphere Proxy based on AWS CloudFormation.

Programmable Terraform Deployment
Deploy Terraform-based ShardingSphere-Proxy in an AWS environment.

Automatic Horizontal Scaling
Custom metrics autoscaling on Kubernetes and AWS.

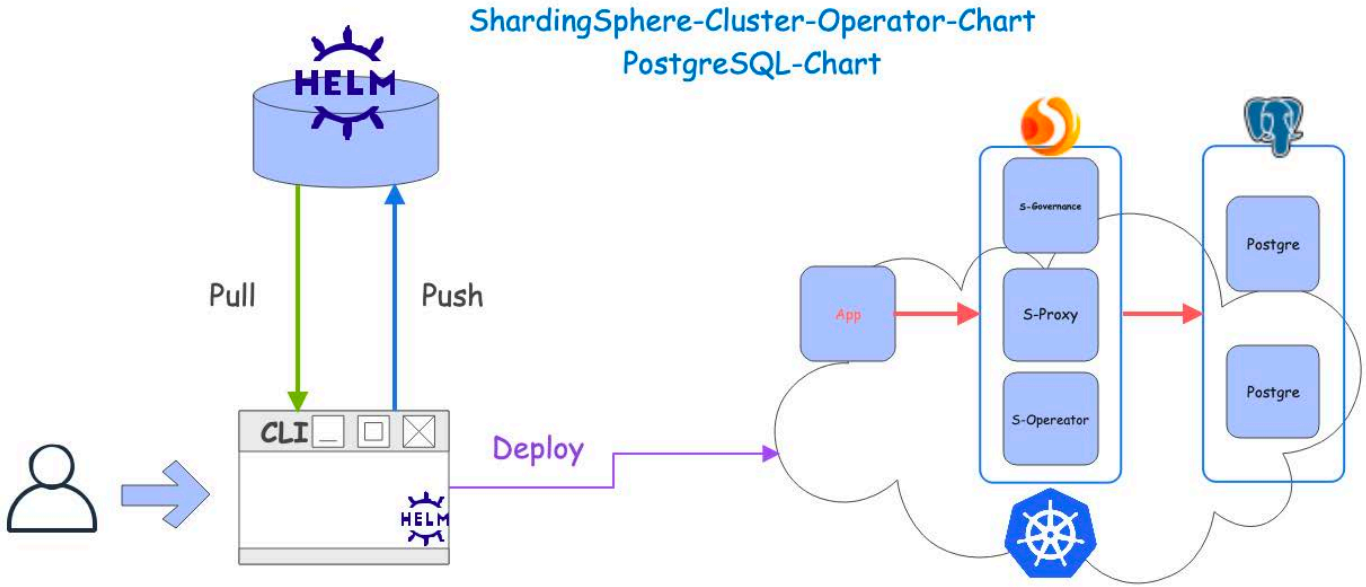
Load Balancing & Readiness
Ensure proxy connection readiness behind the load balancer.



https://github.com/apache/shardingsphere-on-cloud

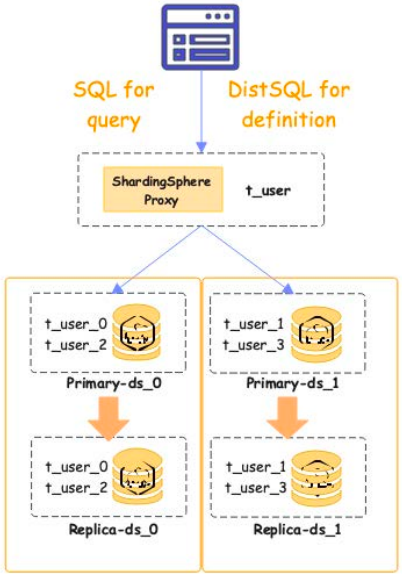
SphereEx

Solution



https://github.com/apache/shardingsphere-on-cloud

Solution



Definition

DistSQL (Distributed SQL) is Apache ShardingSphere's specific SQL, providing additional operation capabilities compared to standard SQL.

Flexible rule configuration and resource management & control capabilities are one of the characteristics of Apache ShardingSphere.

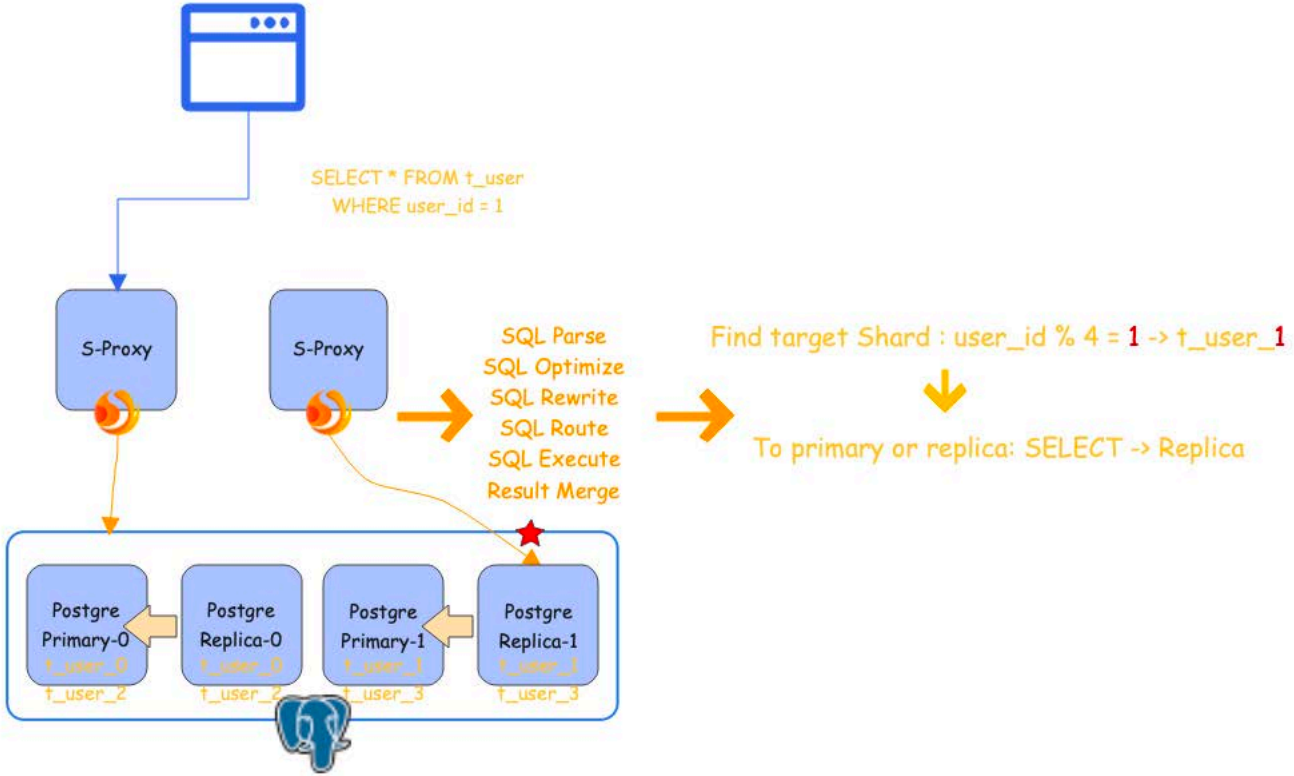
- Create sharding rule

```
CREATE SHARDING TABLE RULE t_order(
STORAGE_UNITS(ds_0,ds_1),
SHARDING_COLUMN=order_id,
TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="4")),
KEY_GENERATE_STRATEGY(COLUMN=order_id,TYPE(NAME="snowflake"))
);
```

- Create sharding table

```
CREATE TABLE `t_order` (
  `order_id` int NOT NULL,
  `user_id` int NOT NULL,
  `status` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`order_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

Solution

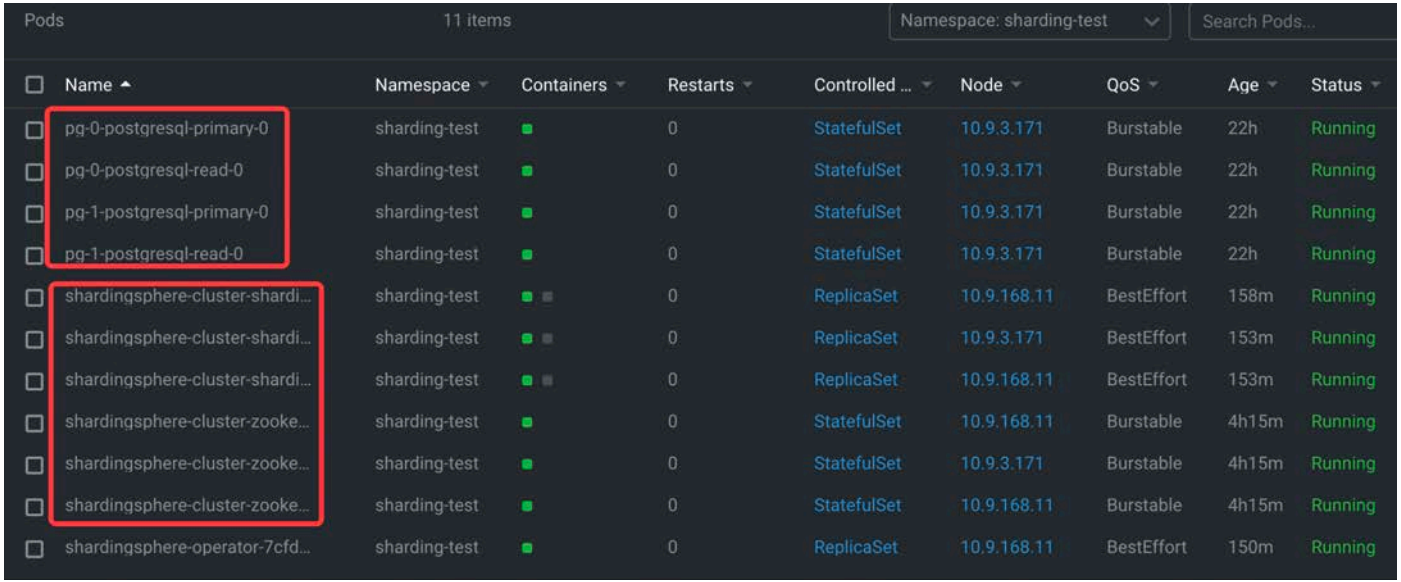

SELECT * FROM t_user
WHERE user_id = 1
S-Proxy
S-Proxy
SQL Parse
SQL Optimize
SQL Rewrite
SQL Route
SQL Execute
Result Merge
Find target Shard : user_id % 4 = 1 -> t_user_1
To primary or replica: SELECT -> Replica
Postgre Primary-0
Postgre Replica-0
Postgre Primary-1
Postgre Replica-1
t_user_2
t_user_3
t_user_3

The demo show

1. Deploy two PostgreSQL (Storage node) clusters made of a primary node and a replica
2. Deploy two ShardingSphere-Proxy (Computing node) and ShardingSphere-governance
3. Register PostgreSQL resources and their relationship into ShardingSphere-Proxy
4. Create sharding table t_user on ShardingSphere-Proxy
5. Show the metadata of this distributed database system
6. INSERT data for test on ShardingSphere-Proxy
7. Preview SELECT routing result
8. Execute SELECT query

Step 1, 2,

git clone https://github.com/apache/shardingsphere-on-cloud
cd charts/shardingsphere-operator-cluster
helm dependency build
helm install shardingsphere-cluster shardingsphere-operator-cluster -n sharding-test
helm install pg-0 bitnami/postgresql -n sharding-test --set global.storageClass=csi-udisk-rssd --set architecture=replication



SphereEx

Step 3, 4, 5

```
postgres=> create database sharding_rw_splitting_db;
CREATE DATABASE
postgres=> \c sharding_rw_splitting_db
psql (14.6 (Homebrew), server 12.3-ShardingSphere-Proxy 5.3.1)
You are now connected to database "sharding_rw_splitting_db" as user "root".
sharding_rw_splitting_db=>
```

```
sharding_rw_splitting_db=> REGISTER STORAGE UNIT write_ds_0 (
    URL="jdbc:postgresql://pg-0-postgresql-primary.sharding-test:5432/sharding_rw_splitting_db",
    USER="postgres",
    PASSWORD="0Yr2fMKXP4",
    PROPERTIES("maximumPoolSize"="50","idleTimeout"="60000")
),read_ds_0 (
    URL="jdbc:postgresql://pg-0-postgresql-read.sharding-test:5432/sharding_rw_splitting_db",
    USER="postgres",
    PASSWORD="0Yr2fMKXP4",
    PROPERTIES("maximumPoolSize"="50","idleTimeout"="60000")
),write_ds_1 (
    URL="jdbc:postgresql://pg-1-postgresql-primary.sharding-test:5432/sharding_rw_splitting_db",
    USER="postgres",
    PASSWORD="By5x6xHC7v",
    PROPERTIES("maximumPoolSize"="50","idleTimeout"="60000")
),read_ds_1 (
    URL="jdbc:postgresql://pg-1-postgresql-read.sharding-test:5432/sharding_rw_splitting_db",
    USER="postgres",
    PASSWORD="By5x6xHC7v",
    PROPERTIES("maximumPoolSize"="50","idleTimeout"="60000")
);
SUCCESS
```

SphereEx

Step 3, 4, 5

```
sharding_rw_splitting_db=> CREATE READWRITE_SPLITTING RULE group_0 (
WRITE_STORAGE_UNIT=write_ds_0,
READ_STORAGE_UNITS(read_ds_0),
TYPE(NAME="random")
);
SUCCESS
sharding_rw_splitting_db=> CREATE READWRITE_SPLITTING RULE group_1 (
WRITE_STORAGE_UNIT=write_ds_1,
READ_STORAGE_UNITS(read_ds_1),
TYPE(NAME="random")
);
SUCCESS
```

```
test=> CREATE SHARDING TABLE RULE t_user(
STORAGE_UNITS(group_0,group_1),
SHARDING_COLUMN=user_id,
TYPE(NAME="hash_mod",PROPERTIES("sharding-count"="4"))
);
```

```
sharding_rw_splitting_db=> CREATE TABLE t_user (
  user_id int4,
  user_name varchar(32),
  tel varchar(32)
);
CREATE TABLE
```

```
sharding_rw_splitting_db-> SHOW SHARDING TABLE NODES t_user;
  name   |                                        nodes
---------+-------------------------------------------------------------------------------
 t_order | group_0.t_order_0, group_1.t_order_1, group_0.t_order_2, group_1.t_order_3
(1 row)

sharding_rw_splitting_db=>
```

SphereEx

Step 6, 7, 8

```
postgres=>
postgres=> INSERT INTO t_user values (1,'name1','tel11111');
INSERT INTO t_user values (2,'name2','tel22222');
INSERT INTO t_user values (3,'name3','tel33333');
INSERT INTO t_user values (4,'name4','tel44444');
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
```

```
sharding_rw_splitting_db=>
sharding_rw_splitting_db=> PREVIEW SELECT * FROM t_user;
 data_source_name |                        actual_sql
------------------+-----------------------------------------------------------
 read_ds_0        | SELECT * FROM t_user_0 UNION ALL SELECT * FROM t_user_2
 read_ds_1        | SELECT * FROM t_user_1 UNION ALL SELECT * FROM t_user_3
(2 rows)
```

```
sharding_rw_splitting_db=> PREVIEW SELECT * FROM t_user WHERE user_id=1;
 data_source_name |              actual_sql
------------------+---------------------------------------
 read_ds_1        | SELECT * FROM t_user_1 WHERE user_id=1
(1 row)
```

```
sharding_rw_splitting_db=> SELECT * FROM t_user ORDER BY user_id;
 user_id | user_name |   tel
---------+-----------+----------
       1 | name1     | tel11111
       2 | name2     | tel22222
       3 | name3     | tel33333
       4 | name4     | tel44444
(4 rows)

sharding_rw_splitting_db=>
```

```
sharding_rw_splitting_db=>
sharding_rw_splitting_db=> SELECT * FROM t_user WHERE user_id=1;
 user_id | user_name |   tel
---------+-----------+----------
       1 | name1     | tel11111
(1 row)
```

SphereEx

Thanks!
Any questions?

Bio: https://tristazero.github.io

LinkedIn: https://www.linkedin.com/in/panjuan

GitHub: https://github.com/tristaZero

Twitter: @tristaZero

Project Twitter: @ShardingSphere

SphereEx