

**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>

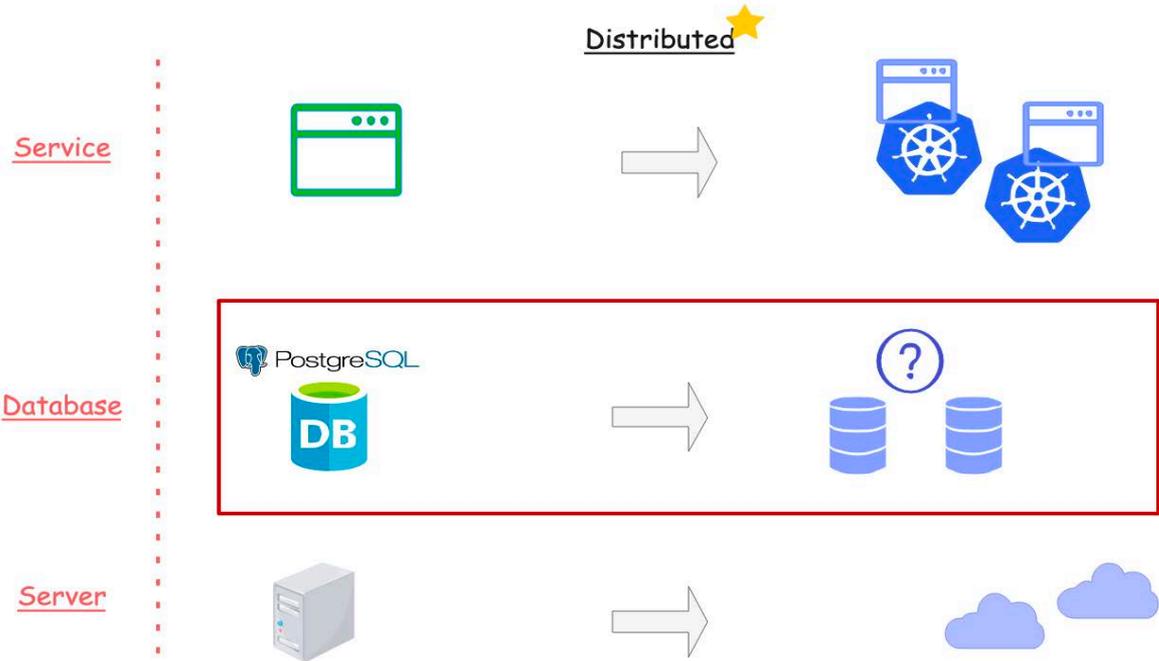
Twitter: @tristaZero

Project Twitter: @ShardingSphere

Content

- ✓ Issues
- ✓ Kubernetes & database
- ✓ Distributed database architecture
- ✓ New Idea & solution
- ✓ Demo show

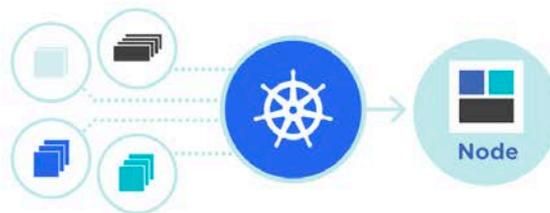
Issues



Kubernetes

Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.

It groups containers that make up an application into logical units for easy management and discovery. Kubernetes builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community.



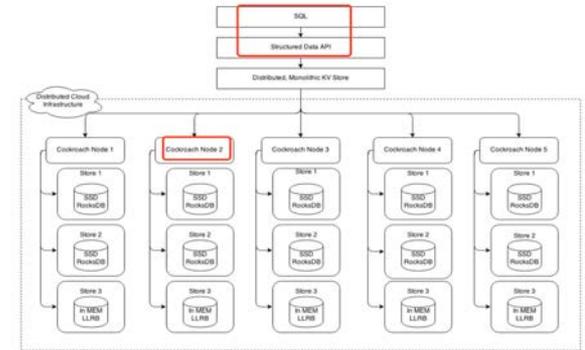
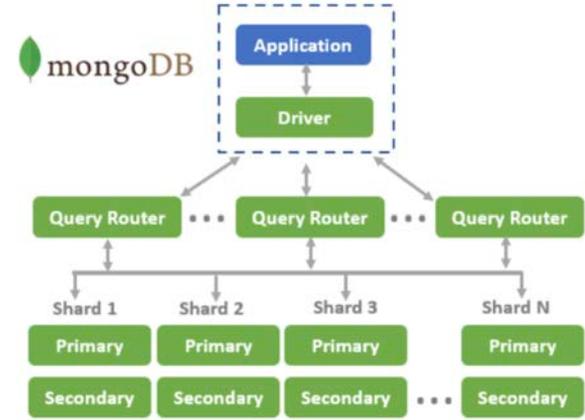
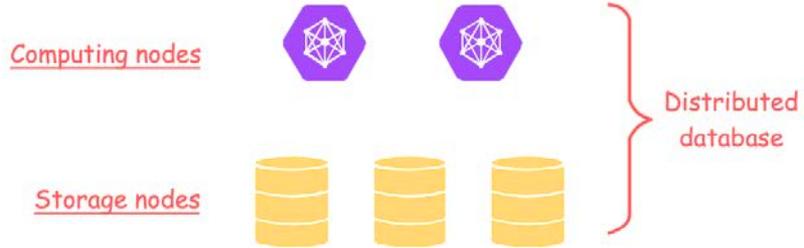
Stateless service VS stateful service

- ✓ Data persistence
- ✓ State management
- ✓ Backup & restore
- ✓ Monitor & HA & deployment & Scaling & Security & QoS

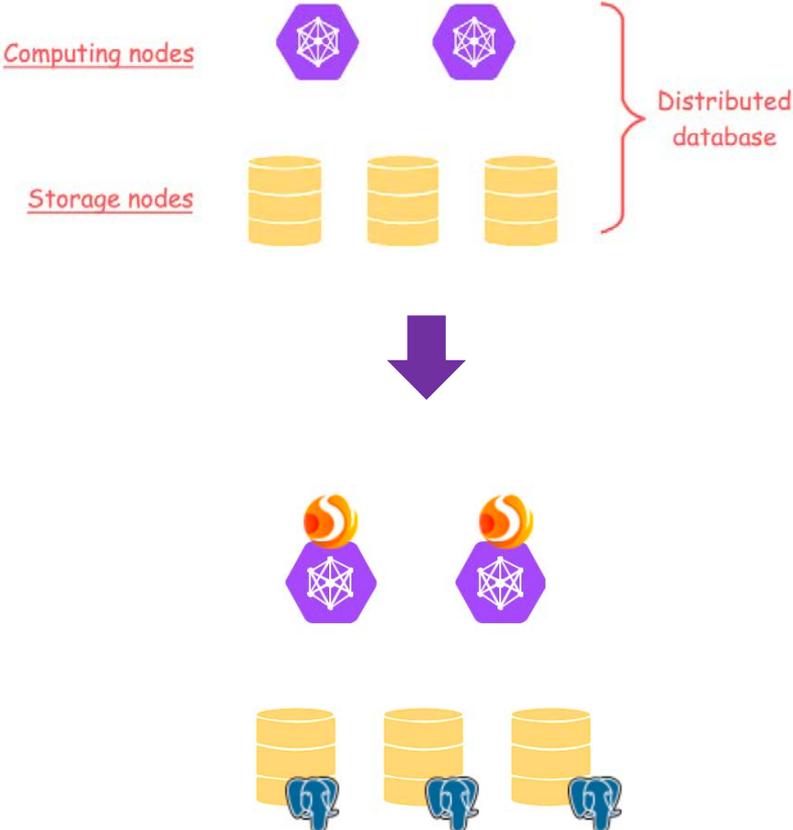


- ✓ PV & PVC & Storageclass
- ✓ StatefulSet & Pod Identity

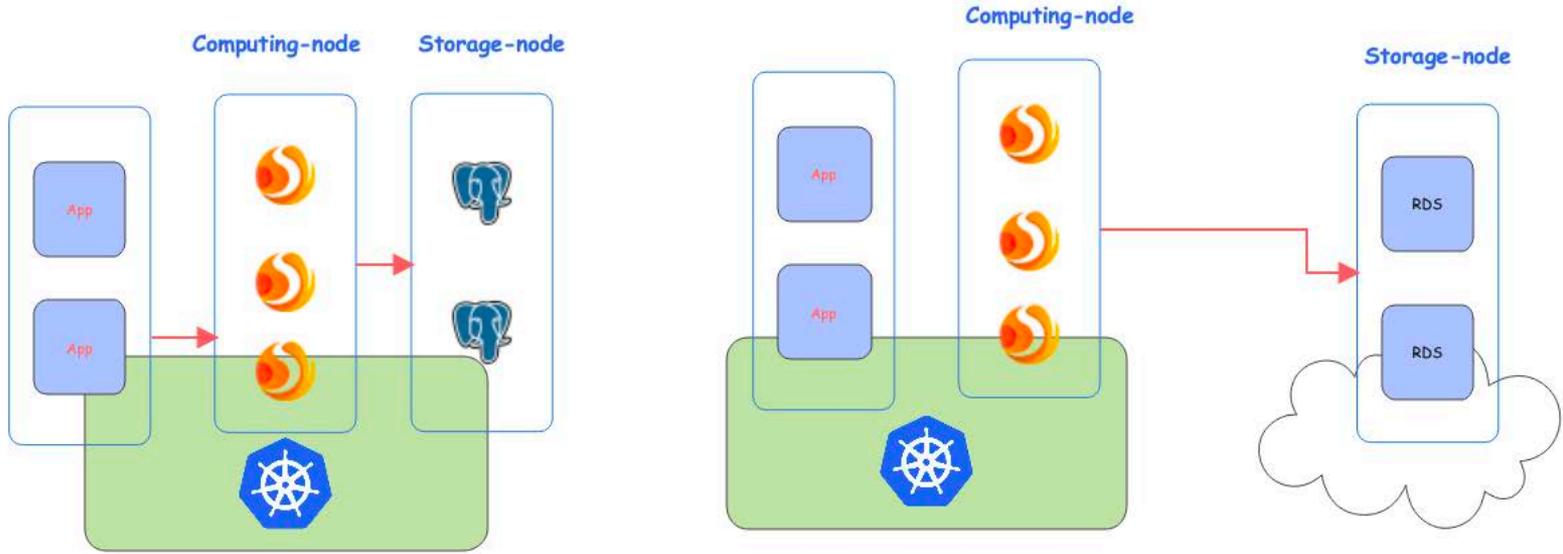
Distributed database



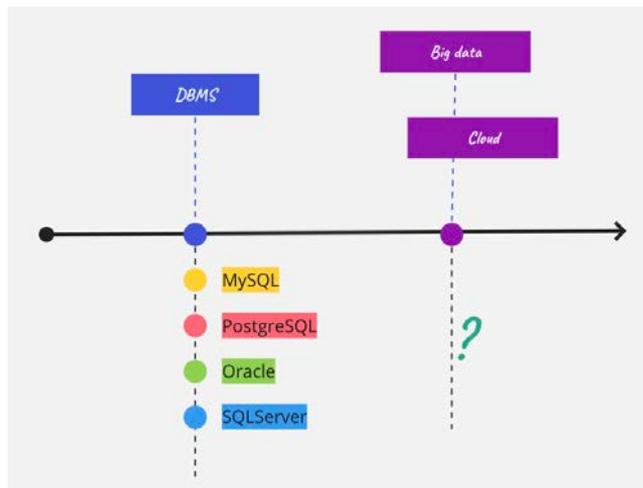
Distributed database



Solution



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

Edit Pins ▾ Watch 1k ▾ Fork 6.2k ▾ Starred 18.1k

About

Ecosystem to transform any database into a distributed database system, and enhance it with sharding, elastic scaling, encryption features & more

mysql sql database bigdata
postgresql shard rdms
distributed-transactions
distributed-database dba encrypt
hacktoberfest database-cluster otp
distributed-sql-database database-plus

Readme

Apache-2.0 license

Code of conduct

Security policy

18.1k stars

1k watching

6.2k forks

Releases 52

5.3.1 Latest
on Jan 10

+ 51 releases

What is Apache ShardingSphere?

The ecosystem to transform any database into a distributed database system, and enhance it with sharding, elastic scaling, encryption features & more.

Download

Learn More

Academic Publications



ShardingSphere

ShardingSphere English ▾

ShardingSphere > Overview Edit this page

- Overview
- Quick Start
- Features
- User Manual
- Dev Manual
- Test Manual
- Reference
- FAQ
- Downloads

This chapter mainly introduces what Apache ShardingSphere is, as well as its design philosophy and deployment architecture.

For frequently asked questions, please refer to FAQ.

What is ShardingSphere

Introduction

Apache ShardingSphere is an open source ecosystem that allows you to transform any database into a distributed database system. The project includes a JDBC and a Proxy, and its core adopts a micro-kernel and pluggable architecture. Thanks to its plugin-oriented architecture, features can be flexibly expanded at will.

The project is committed to providing a multi-source heterogeneous, enhanced database platform and further building an ecosystem around the upper layer of the platform. Database Plus, the design philosophy of Apache ShardingSphere, aims at building the standard and ecosystem on the upper layer of the heterogeneous database. It focuses on how to make full and reasonable use of the computing and storage capabilities of existing databases rather than creating a brand new database. It attaches greater importance to the collaboration between multiple databases instead of the database itself.

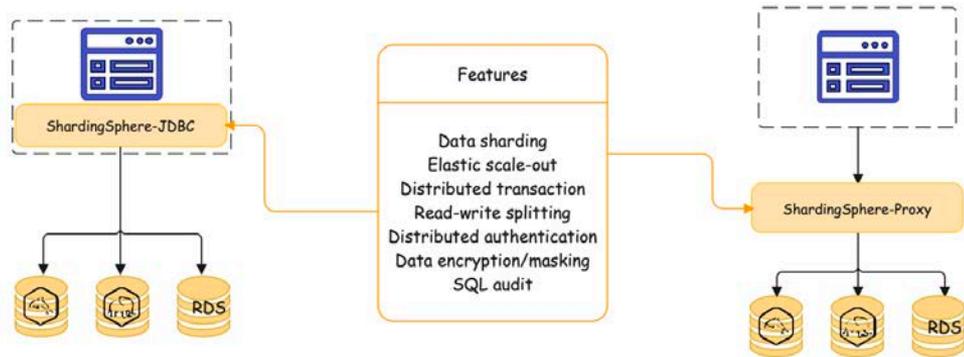
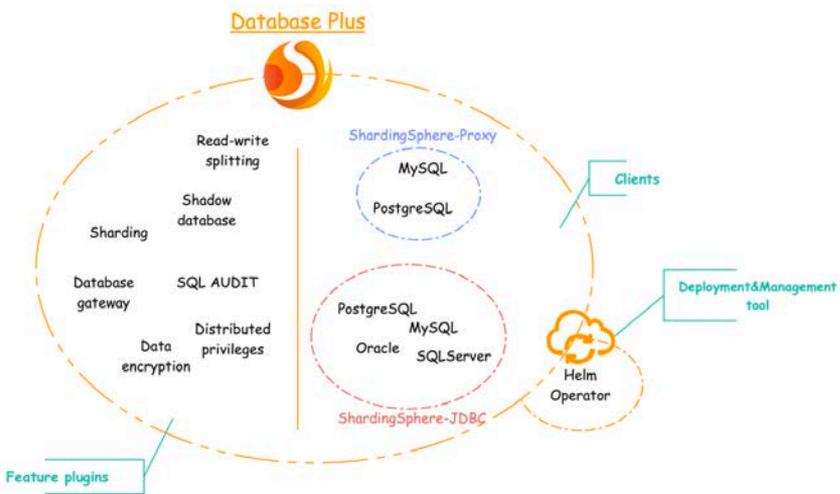
ShardingSphere-JDBC

ShardingSphere-JDBC is a lightweight Java framework that provides additional services at Java's JDBC layer.

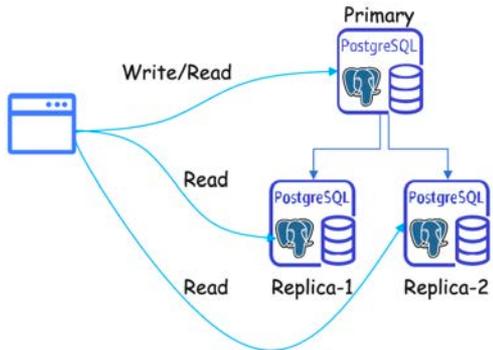
Download PDF

- What is ShardingSphere
 - Introduction
 - Product Features
 - Advantages
 - Roadmap
 - How to Contribute
- Design Philosophy
 - Context: Create database upper level standard
 - Enhance: Database computing enhancement engine
 - Pluggable: Building database function ecology
- Deployment
 - Deployment
 - Running Modes

ShardingSphere features

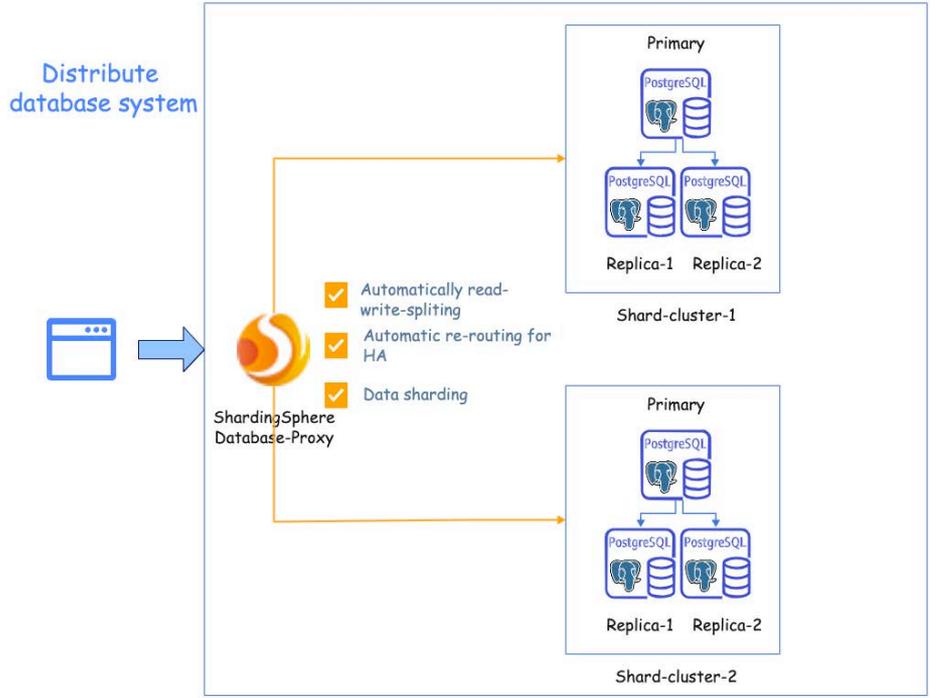


Application -> Database



Before

Distribute database system



After

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.



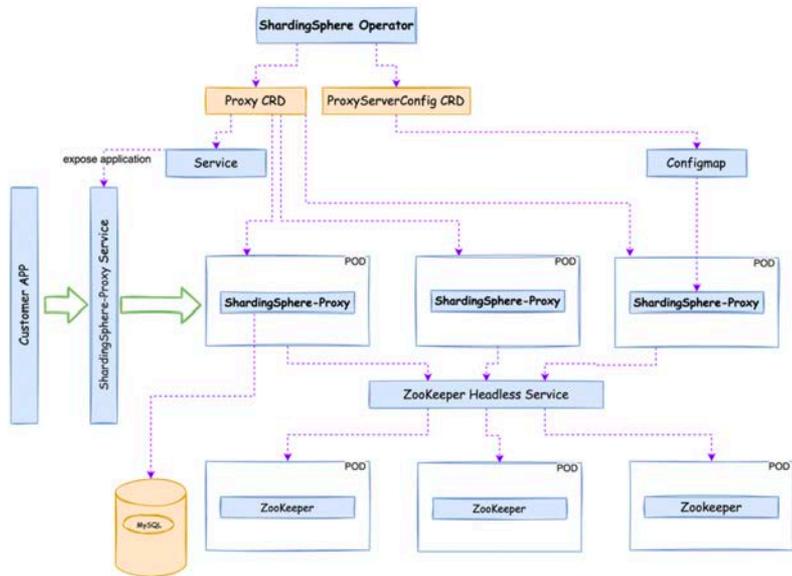
Automatic Horizontal Scaling

Custom metrics autoscaling on Kubernetes and AWS.



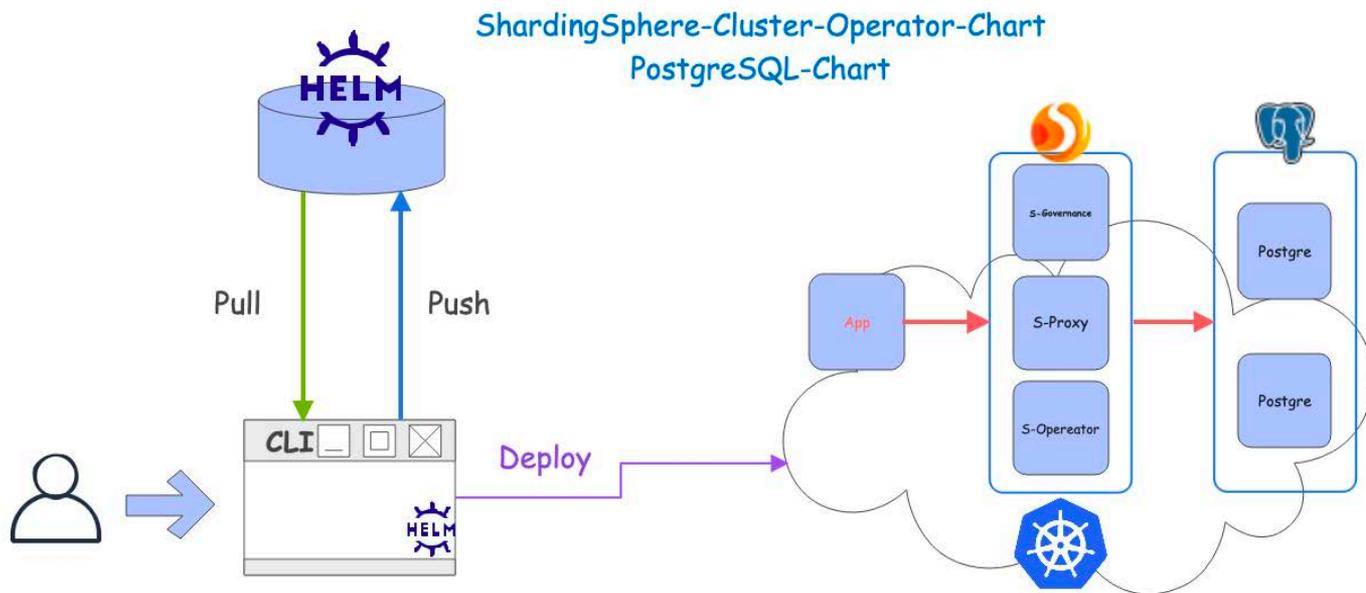
Load Balancing & Readiness

Ensure proxy connection readiness behind the load balancer.

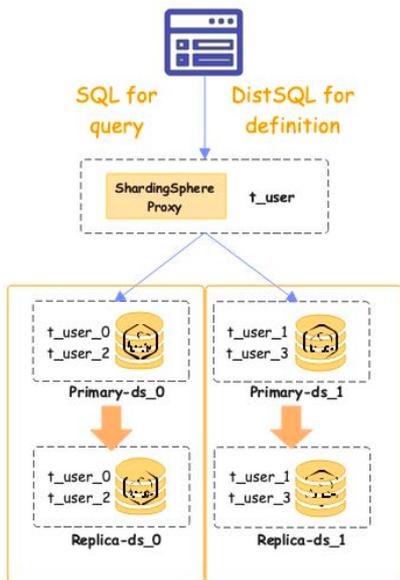


<https://github.com/apache/shardingsphere-on-cloud>

Solution



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.

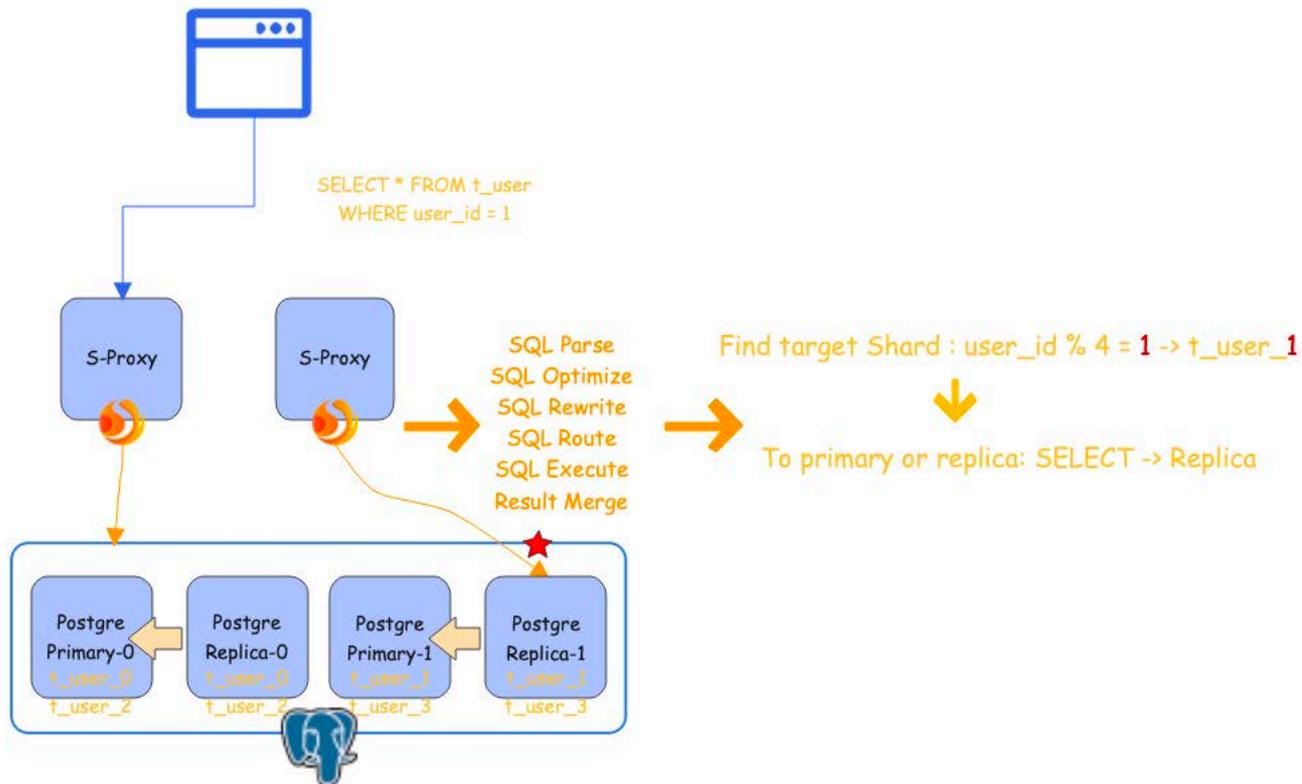
- Create sharding rule

```
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



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,

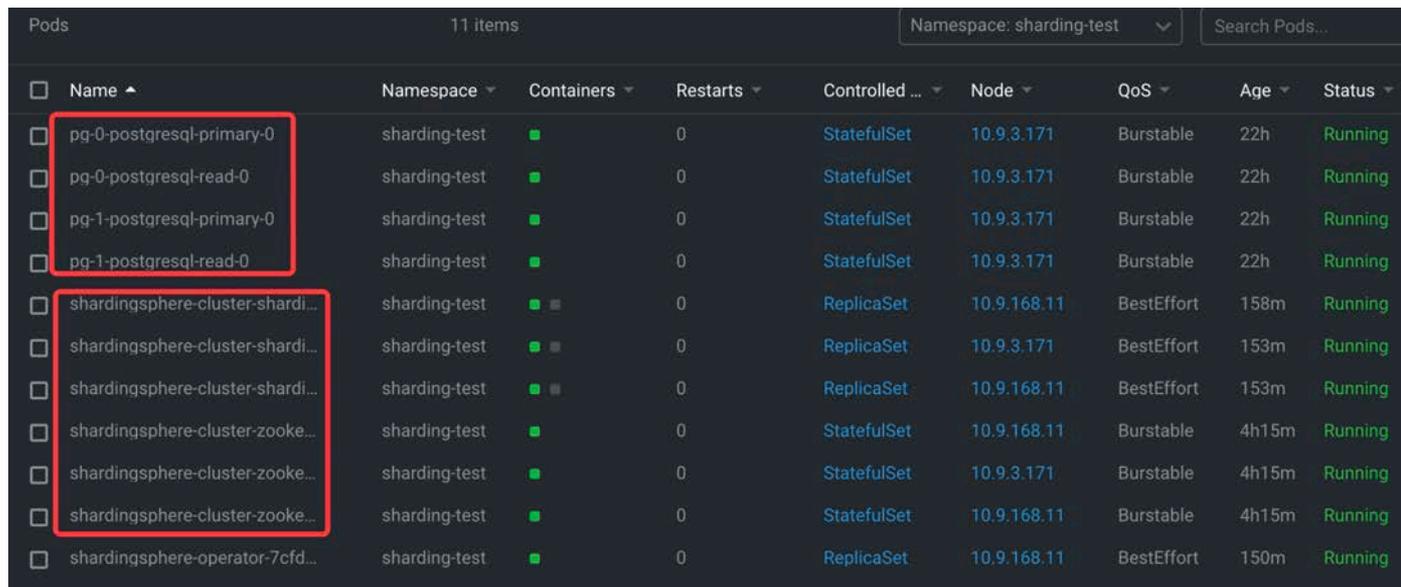
```
git clone https://github.com/apache/shardingsphere-on-cloud
```

```
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
```



Name	Namespace	Containers	Restarts	Controlled ...	Node	QoS	Age	Status
pg-0-postgresql-primary-0	sharding-test	■	0	StatefulSet	10.9.3.171	Burstable	22h	Running
pg-0-postgresql-read-0	sharding-test	■	0	StatefulSet	10.9.3.171	Burstable	22h	Running
pg-1-postgresql-primary-0	sharding-test	■	0	StatefulSet	10.9.3.171	Burstable	22h	Running
pg-1-postgresql-read-0	sharding-test	■	0	StatefulSet	10.9.3.171	Burstable	22h	Running
shardingsphere-cluster-shardi...	sharding-test	■ ■	0	ReplicaSet	10.9.168.11	BestEffort	158m	Running
shardingsphere-cluster-shardi...	sharding-test	■ ■	0	ReplicaSet	10.9.3.171	BestEffort	153m	Running
shardingsphere-cluster-shardi...	sharding-test	■ ■	0	ReplicaSet	10.9.168.11	BestEffort	153m	Running
shardingsphere-cluster-zooke...	sharding-test	■	0	StatefulSet	10.9.168.11	Burstable	4h15m	Running
shardingsphere-cluster-zooke...	sharding-test	■	0	StatefulSet	10.9.3.171	Burstable	4h15m	Running
shardingsphere-cluster-zooke...	sharding-test	■	0	StatefulSet	10.9.168.11	Burstable	4h15m	Running
shardingsphere-operator-7cfd...	sharding-test	■	0	ReplicaSet	10.9.168.11	BestEffort	150m	Running

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
```

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=>
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=>
```

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

```
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=> 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=>
```

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