ShardingSphere

Apache ShardingSphere 架构解析与应用实践

孟浩然

2021/12/11

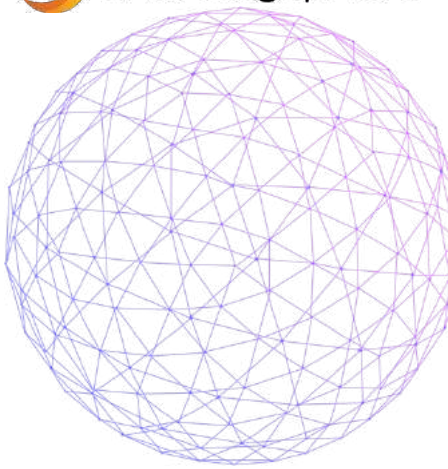
孟浩然

SphereEx 高级研发工程师

Apache ShardingSphere PMC

曾就职京东科技，负责数据库产品研发，热爱开源，关注数据库生态，目前就职 SphereEx，专注于 Apache ShardingSphere 分布式数据库中间件研发以及开源社区建设

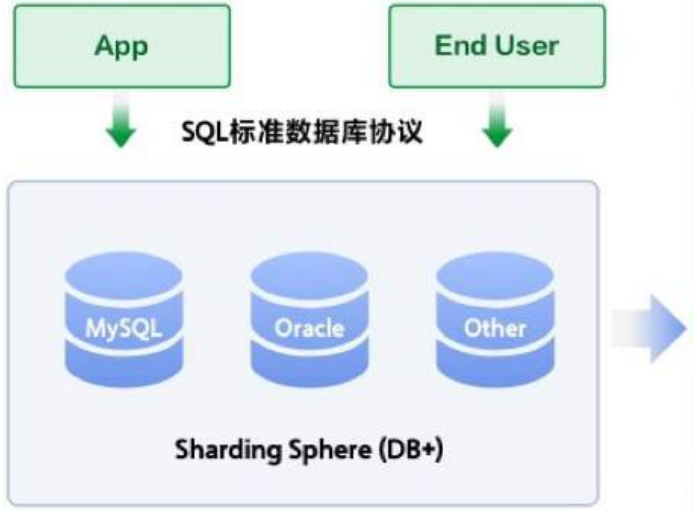
ShardingSphere



目录

产品定位

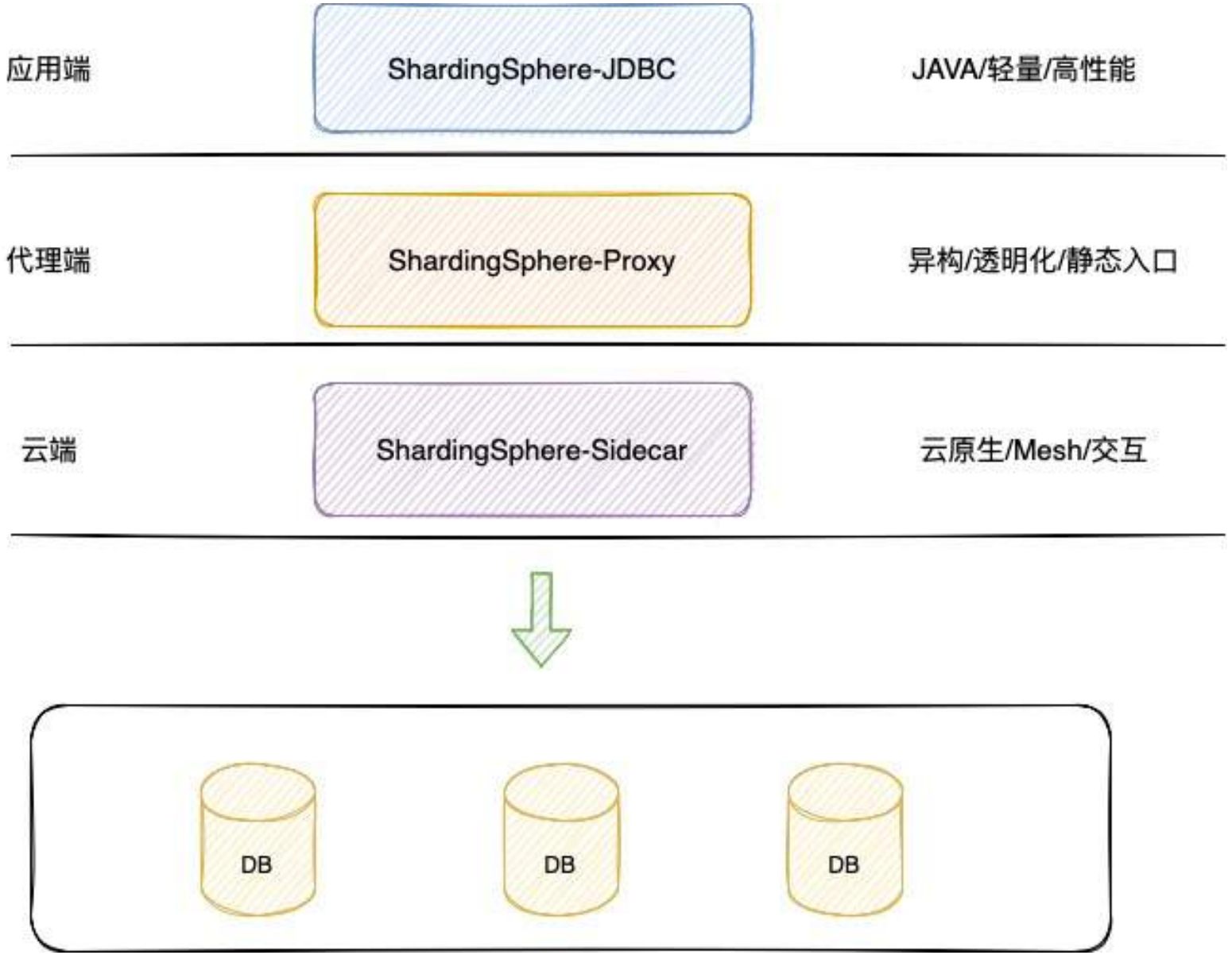
- 构建异构数据库的上层标准和生态
- 提供精准化和差异化的能力



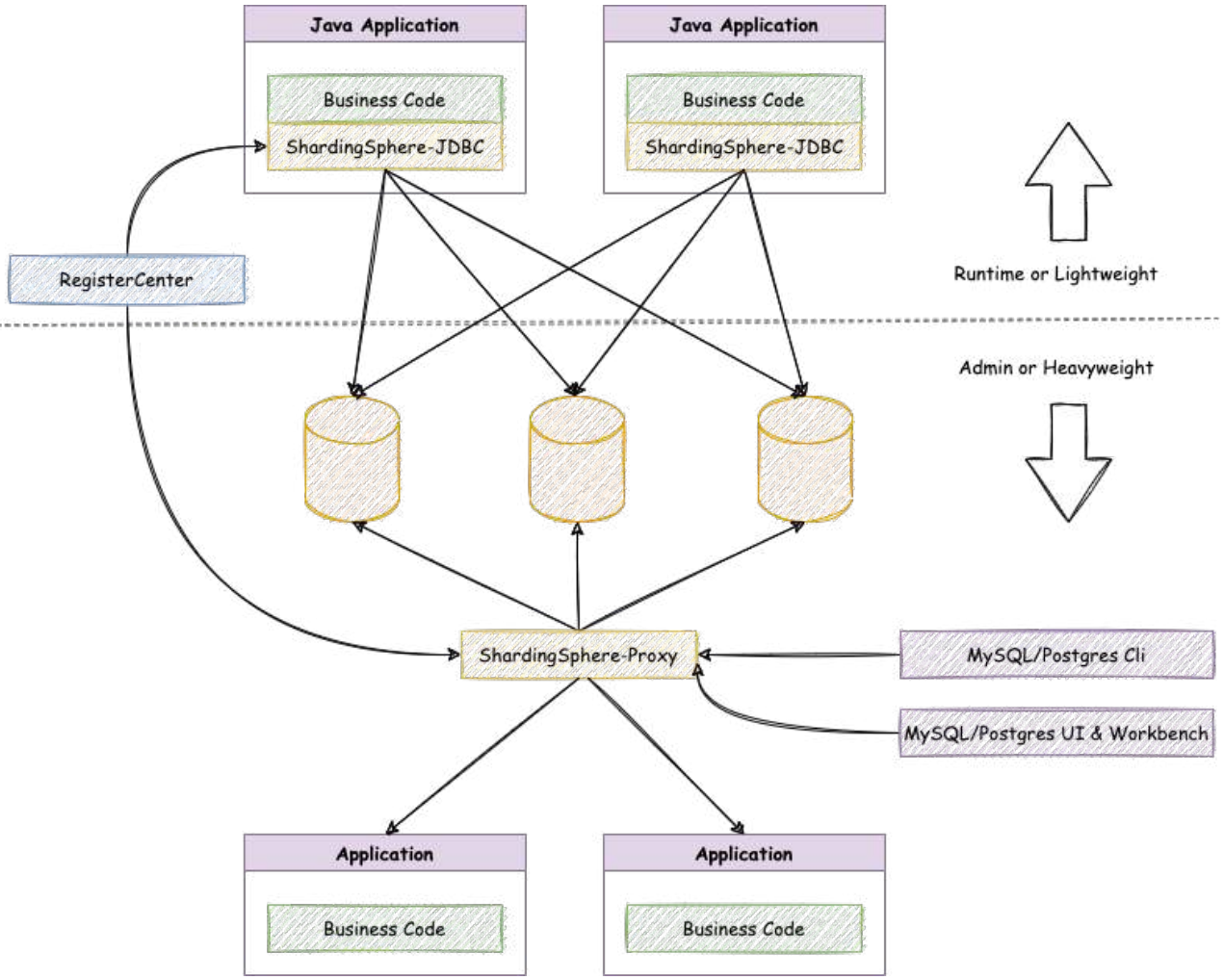
产品定位

	4.X	5.X
定位	分库分表中间件	分布式数据库生态系统
功能	提供基础功能	提供基础设施和最佳实践
驱动方式	配置文件	标准 DistSQL
耦合	耦合较大，存在功能依赖	相互隔离，互无感知
组合方式	固定的组合方式： 以数据分片为基础，叠加读写分离和数据加密等功能	自由的组合方式： 数据分片、读写分离和数据加密等功能自由组合使用

产品架构

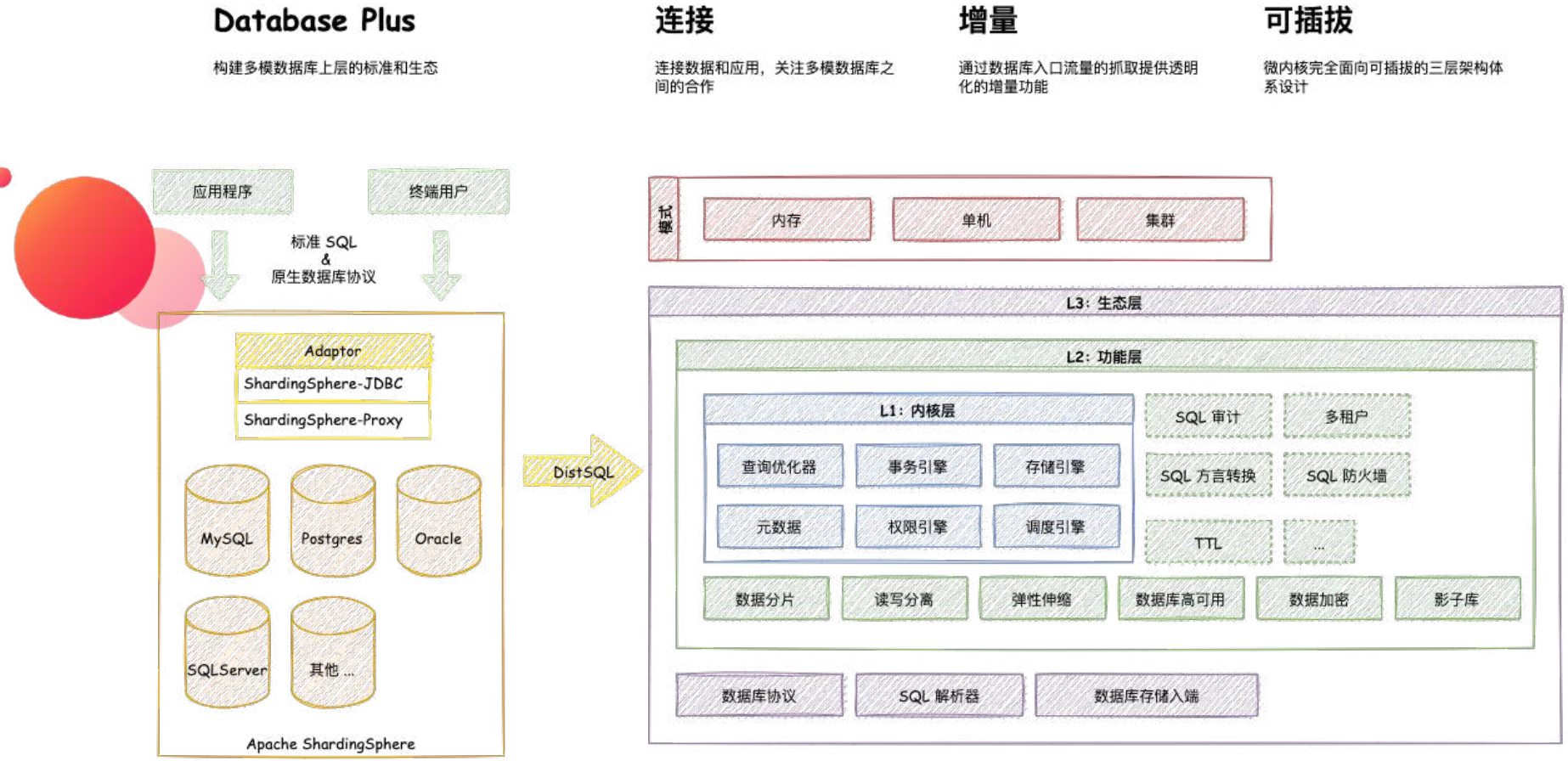

应用端
ShardingSphere-JDBC
JAVA/轻量/高性能
代理端
ShardingSphere-Proxy
异构/透明化/静态入口
云端
ShardingSphere-Sidecar
云原生/Mesh/交互
DB
DB
DB

部署架构



- ShardingSphere-JDBC 采用无中心化架构，与应用程序共享资源，适用于 Java 开发的高性能的轻量级 OLTP 应用；
- ShardingSphere-Proxy 提供静态入口以及异构语言的支持，独立于应用程序部署，适用于 OLAP 应用以及对分片数据库进行管理和运维的场景。

整体架构



- L1 内核层：面向数据库内核，包括数据库事务引擎，查询优化器等；
- L2 功能层：可定制化开发平台。具有高定制化、高度内聚、灵活扩展等特点；
- L3 生态层：通过三个接口分别实现数据库协议、 SQL 方言和数据库存储对接，用于打造异构数据网关；

整体架构

连接

连接是 ShardingSphere 的基础能力，可以有效简化数据和应用之间的连接。连接的设计要点在于强大的数据库的兼容性，在应用和数据之间搭建了一层与具体数据库实现无关的桥梁，为增量能力提供了基础。

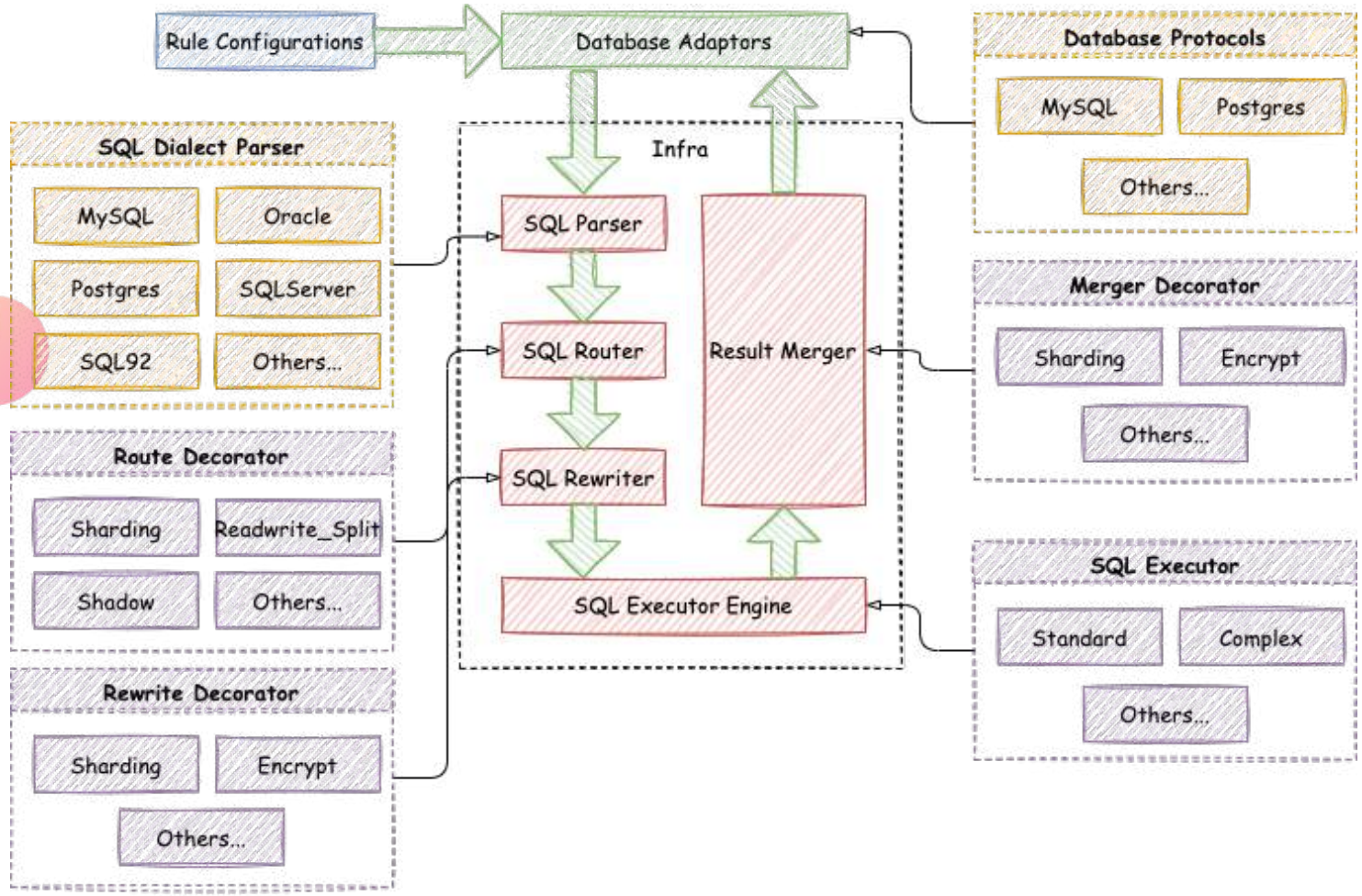
增量

增量是 ShardingSphere 的主要能力，在拦截访问数据库流量的前提下，透明化的提供增量功能。增强包含了流量的重定向（数据分片、读写分离、影子库）、流量变形（数据加密）、流量鉴权（SQL 审计、权限）、流量治理（熔断、限流）以及流量分析（可观察性、服务质量分析）等。

可插拔

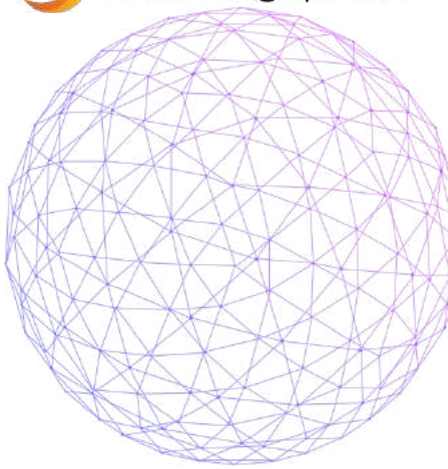
可插拔是 ShardingSphere 的设计理念，架构内核是完全面向顶层接口设计的，内核模块完全不感知具体功能的存在。它为分库分表、读写分离等每一个功能插件赋予单独部署和协同配合的能力。

内核架构



- ShardingSphere 可插拔架构提供了数十个基于 SPI 的扩展点，开发者可以十分方便的对功能进行定制化扩展；
- 按照扩展点是基于技术还是基于功能实现，可以将扩展点划分为功能扩展点和技术扩展点。
- 基于扩展点，ShardingSphere 默认实现了数据分片、读写分离、数据加密、影子库压测、高可用等功能；

目录



运行模式

内存模式

进程内
集成测试
无需清理运行痕迹

单机模式

持久化
集群无感知
本地开发

集群模式

生产环境
分布式
元数据共享
状态协调
水平扩展

覆盖开发/测试/生产部署需求

灵活可扩展

DistSQL

ShardingSphere

标准化

体系化

功能特色

- RDL（Resource & Rule Definition Language）负责资源和规则的创建、修改和删除；
- RQL（Resource & Rule Query Language）负责资源和规则的查询和展现；
- RAL（Resource & Rule Administration Language）负责 Hint、事务类型切换、分片执行计划查询等增量功能的操作。

DistSQL

ShardingSphere

Create Logic Database

1. Connect to ShardingSphere-Proxy
2. Create a logic database (schema)

```
CREATE DATABASE sharding_db;
# CREATE SCHEMA sharding_db;
```

3. Show logic databases

```
SHOW DATABASES;
```

- Output of `show databases`:

```
+--------------+
| schema_name |
+--------------+
| sharding_db |
+--------------+
1 row in set (0.01 sec)
```

4. Use the logic database

```
USE sharding_db;
```

```
schemaName: sharding_db

dataSources:
  ds_0:
    url: jdbc:mysql://127.0.0.1:3306/demo_ds_0?serverTimezone=UTC&useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 50
    minPoolSize: 1
  ds_1:
    url: jdbc:mysql://127.0.0.1:3306/demo_ds_1?serverTimezone=UTC&useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 50
    minPoolSize: 1
```

DistSQL

ShardingSphere

Add Resources

1. Add database resources for sharding_db

RDL

```
ADD RESOURCE
ds_0 (
  HOST=127.0.0.1,
  PORT=3306,
  DB=db0,
  USER=root,
  PASSWORD=root),
ds_1 (
  URL="jdbc:mysql://127.0.0.1:3306/db1?useSSL=false",
  USER=root,
  PASSWORD=root,
  PROPERTIES("maximumPoolSize"=10,"idleTimeout"="30000")
);
```

2. Show schema resources;

RQL

```
SHOW SCHEMA RESOURCES;
```

- Output of `show schema resources`:

```
+------+-------+-----------+------+------+-----------+
| name | type  | host      | port | db   | attribute |
| ds_0 | MySQL | 127.0.0.1 | 3306 | db0  | ...       |
| ds_1 | MySQL | 127.0.0.1 | 3306 | db1  | ...       |
+------+-------+-----------+------+------+-----------+
2 rows in set (0.01 sec)
```

```
schemaName: sharding_db

dataSources:
  ds_0:
    url: jdbc:mysql://127.0.0.1:3306/demo_ds_0?serverTimezone=UTC&useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 50
    minPoolSize: 1
  ds_1:
    url: jdbc:mysql://127.0.0.1:3306/demo_ds_1?serverTimezone=UTC&useSSL=false
    username: root
    password:
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 50
    minPoolSize: 1
```

DistSQL

ShardingSphere

Create Sharding Rules

1. Create sharding table rule

RDL

```
CREATE SHARDING TABLE RULE t_order (
  RESOURCES(ds_0, ds_1),
  SHARDING_COLUMN=order_id,
  TYPE(NAME=mod,PROPERTIES("sharding-count"=4)),
  GENERATED_KEY(COLUMN=another_id,TYPE(NAME=snowflake,PROPERTIES("worker-id"=1)))
);
```

2. Show sharding table rules;

RQL

```
SHOW SHARDING TABLE RULES;
# SHOW SHARDING TABLE RULE t_order;
```

- Output of `show sharding table rules;`

```
| table    | actual_data_nodes | actual_data_sources | database_strategy_type |
database_sharding_column | database_sharding_algorithm_type |
database_sharding_algorithm_props | ...
```

```yaml
rules:
- !SHARDING
  tables:
    t_order:
      actualDataNodes: ds_${0..1}.t_order_${0..1}
      tableStrategy:
        standard:
          shardingColumn: order_id
          shardingAlgorithmName: t_order_inline
      databaseStrategy:
        standard:
          shardingColumn: order_id
          shardingAlgorithmName: database_inline
      keyGenerateStrategy:
        column: order_id
        keyGeneratorName: snowflake
  shardingAlgorithms:
    database_inline:
      type: INLINE
      props:
        algorithm-expression: ds_${order_id % 2}
    t_order_inline:
      type: INLINE
      props:
        algorithm-expression: t_order_${order_id % 2}
  keyGenerators:
    snowflake:
      type: SNOWFLAKE
      props:
        worker-id: 1
```

DistSQL

ShardingSphere

Create Sharding Tables

1. Create sharding tables

```
CREATE TABLE IF NOT EXISTS t_order (
  order_id BIGINT NOT NULL AUTO_INCREMENT,
  user_id INT NOT NULL,
  status VARCHAR(50),
  PRIMARY KEY (order_id)
);
```

2. Show tables

```
SHOW TABLES;
```

- Output of `show tables`:

```
+----------------------+
| Tables_in_sharding_db |
+----------------------+
| t_order               |
+----------------------+
1 row in set (0.01 sec)
```

```
mysql>
mysql> use ds_0;
Database changed
mysql> show tables;
+----------------+
| Tables_in_ds_0 |
+----------------+
| t_order_0      |
| t_order_2      |
+----------------+
2 rows in set (0.00 sec)

mysql> use ds_1;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+----------------+
| Tables_in_ds_1 |
+----------------+
| t_order_1      |
| t_order_3      |
+----------------+
2 rows in set (0.00 sec)
```

DistSQL

Preview Or Execute SQL

1. Preview select SQL

```
PREVIEW select * from t_order;
```

RAL

- Output of `preview`:

```
+------------------+-----------------------------------------------+
| data_source_name | sql                                           |
+------------------+-----------------------------------------------+
| ds_0             | select * from t_order_0 ORDER BY order_id ASC |
| ds_0             | select * from t_order_2 ORDER BY order_id ASC |
| ds_1             | select * from t_order_1 ORDER BY order_id ASC |
| ds_1             | select * from t_order_3 ORDER BY order_id ASC |
+------------------+-----------------------------------------------+
4 rows in set (0.04 sec)
```

2. Execute SQL

```
select * from t_order;
```

- Proxy log:

```
ShardingSphere-SQL - Logic SQL: select * from t_order
ShardingSphere-SQL - SQLStatement: MySQLSelectStatement(limit=Optional.empty,
lock=Optional.empty, window=Optional.empty)
ShardingSphere-SQL - Actual SQL: ds_0 ::: select * from t_order_0 ORDER BY order_id ASC
ShardingSphere-SQL - Actual SQL: ds_0 ::: select * from t_order_2 ORDER BY order_id ASC
ShardingSphere-SQL - Actual SQL: ds_1 ::: select * from t_order_1 ORDER BY order_id ASC
ShardingSphere-SQL - Actual SQL: ds_3 ::: select * from t_order_3 ORDER BY order_id ASC
```

- 强制路由
- 弹性伸缩
- 熔断
- 刷新元数据

分布式治理

4.1.1

```
orchestration:
  orchestration_ds:
    orchestrationType: registry_center,config_center
    instanceType: zookeeper
    serverLists: localhost:2181
    namespace: orchestration
    props:
      overwrite: true
```

- Cluster 模式
- 合并注册中心/配置中心
- 完善 ZooKeeper/Etcd 支持

5.0.0

```
mode:
  type: Cluster
  repository:
    type: ZooKeeper
    props:
      namespace: governance_ds
      server-lists: localhost:2181
      retryIntervalMilliseconds: 500
      timeToLiveSeconds: 60
      maxRetries: 3
      operationTimeoutMilliseconds: 500
  overwrite: false
```

数据分片

4.1.1

```
shardingRule:
 tables:
  t_order:
   databaseStrategy:
    standard:
     shardingColumn: order_id
     preciseAlgorithmClassName: xxx
     rangeAlgorithmClassName: xxx
    complex:
     shardingColumns: year, month
     algorithmClassName: xxx
    hint:
     algorithmClassName: xxx
    inline:
     shardingColumn: order_id
     algorithmExpression: ds_${order_id % 2}
    none:
   tableStrategy:
    ...
```

5.0.0

```
rules:
 !SHARDING
  tables:
   t_order:
    databaseStrategy:
     standard:
      shardingColumn: order_id
      shardingAlgorithmName: database_inline
     complex:
      shardingColumns: year, month
      shardingAlgorithmName: database_complex
     hint:
      shardingAlgorithmName: database_hint
     none:
    tableStrategy:
     ...

  shardingAlgorithms:
   database_inline:
    type: INLINE
    props:
     algorithm-expression: ds_${order_id % 2}
  database_complex:
   type: CLASS_BASED
   props:
    strategy: COMPLEX
    algorithmClassName: xxx
  database_hint:
   type: CLASS_BASED
   props:
    strategy: HINT
    algorithmClassName: xxx
```

数据分片-自动分片

自动分片

```
rules:
 - !SHARDING
  autoTables:
   t_order:
    actualDataSources: ds_0, ds_1
    shardingStrategy:
     standard:
      shardingColumn: order_id
      shardingAlgorithmName: auto_mod
    keyGenerateStrategy:
     column: order_id
     keyGeneratorName: snowflake
 shardingAlgorithms:
  auto_mod:
   type: MOD
   props:
    sharding-count: 4
```

手动分片

```
rules:
 - !SHARDING
  tables:
   t_order:
    actualDataNodes: ds${0..1}.t_order${0..1}
    tableStrategy:
     standard:
      shardingColumn: order_id
      shardingAlgorithmName: table_inline
    dataBaseStrategy:
     standard:
      shardingColumn: user_id
      shardingAlgorithmName: database_inline
```

MOD | HASH_MOD | VOLUME_RANGE | BOUNDARY_RANGE | AUTO_INTERVAL

读写分离

ShardingSphere

4.1.1

```
masterSlaveRule:
  name: ms_ds
  masterDataSourceName: master_ds
  slaveDataSourceNames:
   - slave_ds_0
   - slave_ds_1
 loadBalanceAlgorithmType: ROUND_ROBIN
```

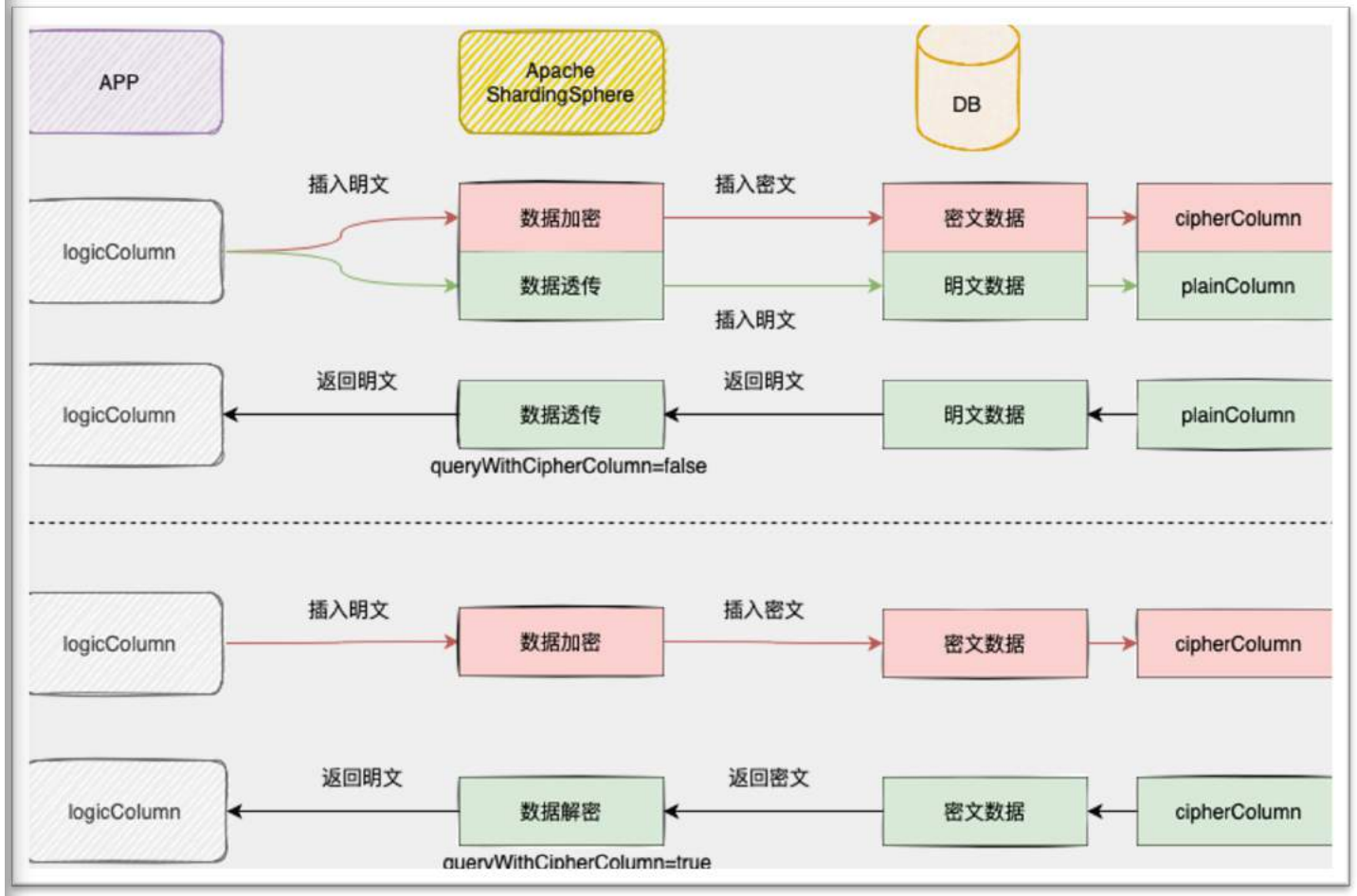
5.0.0

```
rules:
 - !READWRITE_SPLITTING
  dataSources:
   pr_ds:
    writeDataSourceName: write_ds
    readDataSourceNames:
     - read_ds_0
     - read_ds_1
    loadBalancerName: loadBalancer_1
     ...

  loadBalancers:
   loadBalancer_1:
    type: ROUND_ROBIN
```

数据加密

5.0.0

```
rules:
 - !ENCRYPT
  tables:
   t_encrypt:
    columns:
     user_id:
      plainColumn: user_plain
      cipherColumn: user_cipher
      encryptorName: aes_encryptor
     order_id:
      cipherColumn: order_cipher
      encryptorName: md5_encryptor
      queryWithCipherColumn: true
    queryWithCipherColumn: false

  encryptors:
   aes_encryptor:
    type: AES
    props:
     aes-key-value: 123456abc
   md5_encryptor:
    type: MD5
```



影子库压测

4.1.1

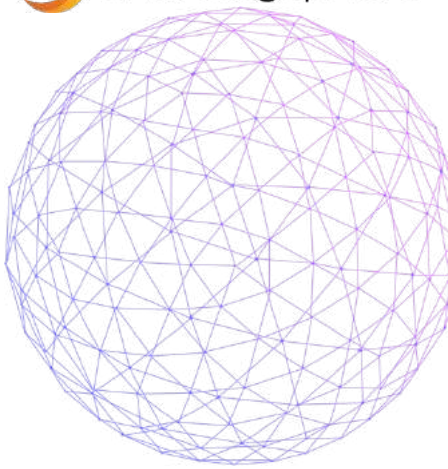
```
shadowRule:
  column: shadow
  shadowMappings:
   ds: shadow_ds
```

- 增加开关，灵活控制是否开启在线压测
- 支持表级压测规则控制
- 支持列值匹配、列正则匹配、SQL 注释匹配算法

5.0.0

```
rules:
 - !SHADOW
  enable: true
  dataSources:
   shadowDataSource:
    sourceDataSourceName: ds
    shadowDataSourceName: shadow_ds
  tables:
   t_order:
    dataSourceNames:
     - shadowDataSource
    shadowAlgorithmNames:
     - user-id-insert-match-algorithm
     - simple-hint-algorithm
 shadowAlgorithms:
  user-id-insert-match-algorithm:
   type: COLUMN_REGEX_MATCH
   props:
    operation: insert
    column: user_id
    regex: "[1]"
  simple-hint-algorithm:
   type: SIMPLE_NOTE
   props:
    shadow: true
    foo: bar
```

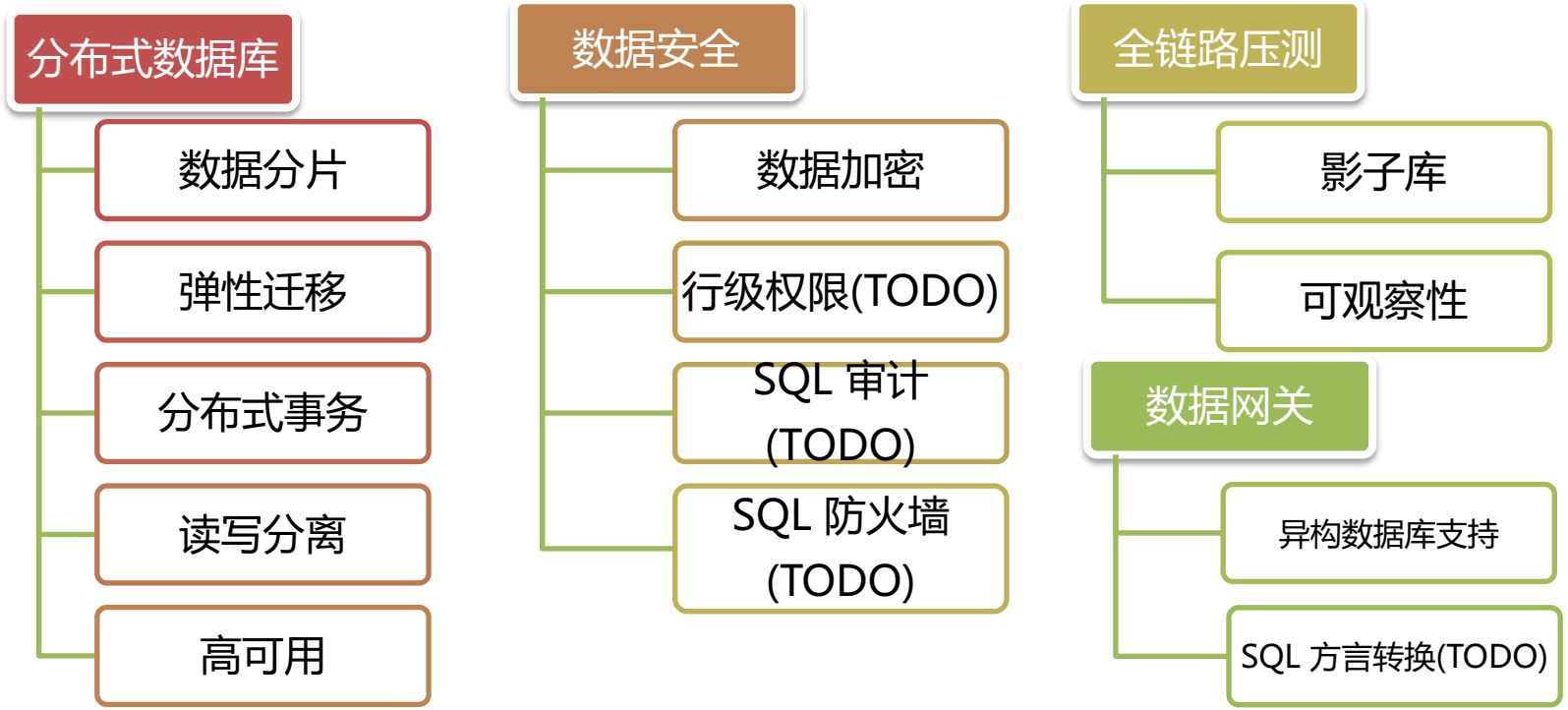
ShardingSphere



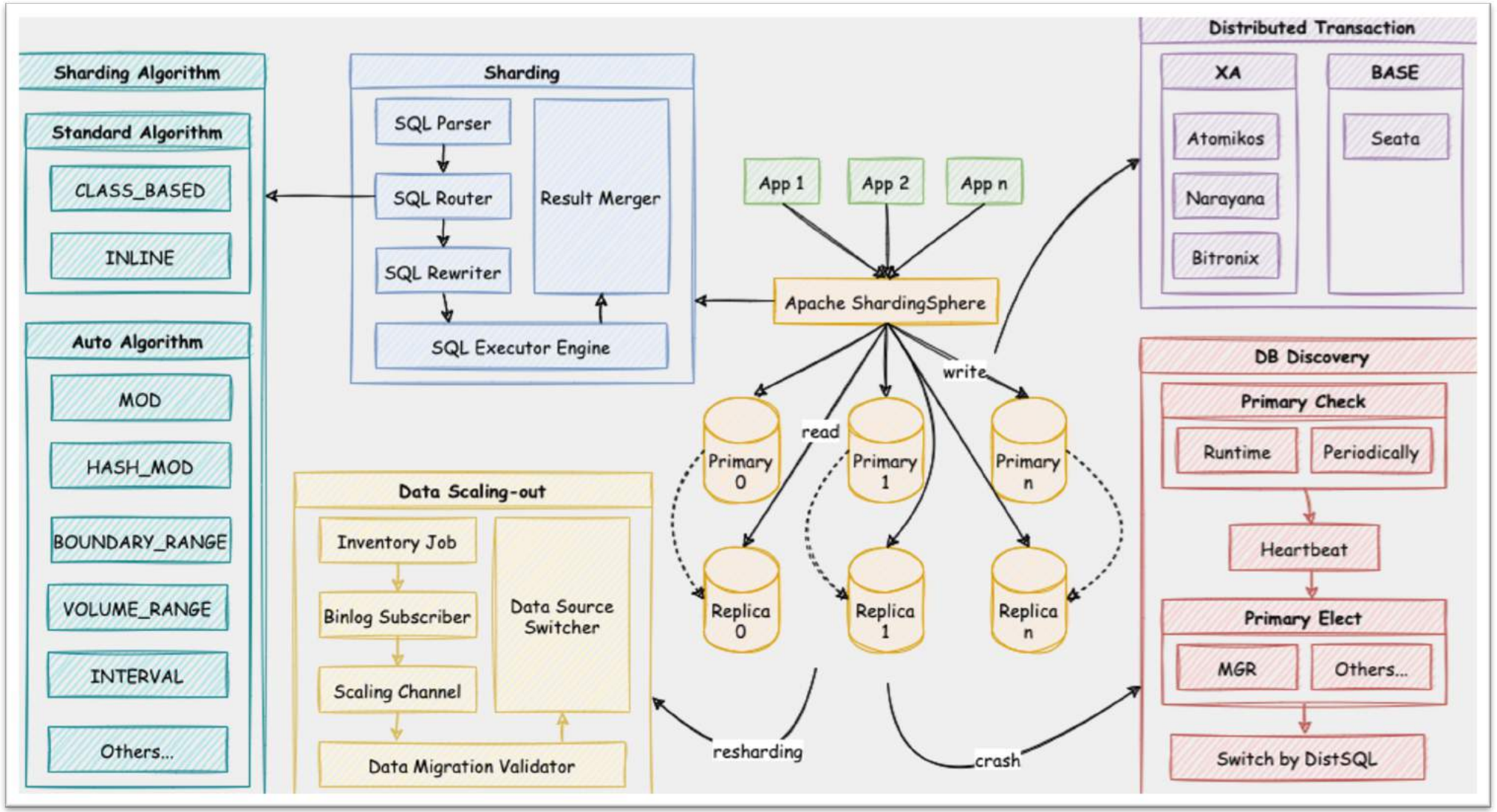
目录

解决方案

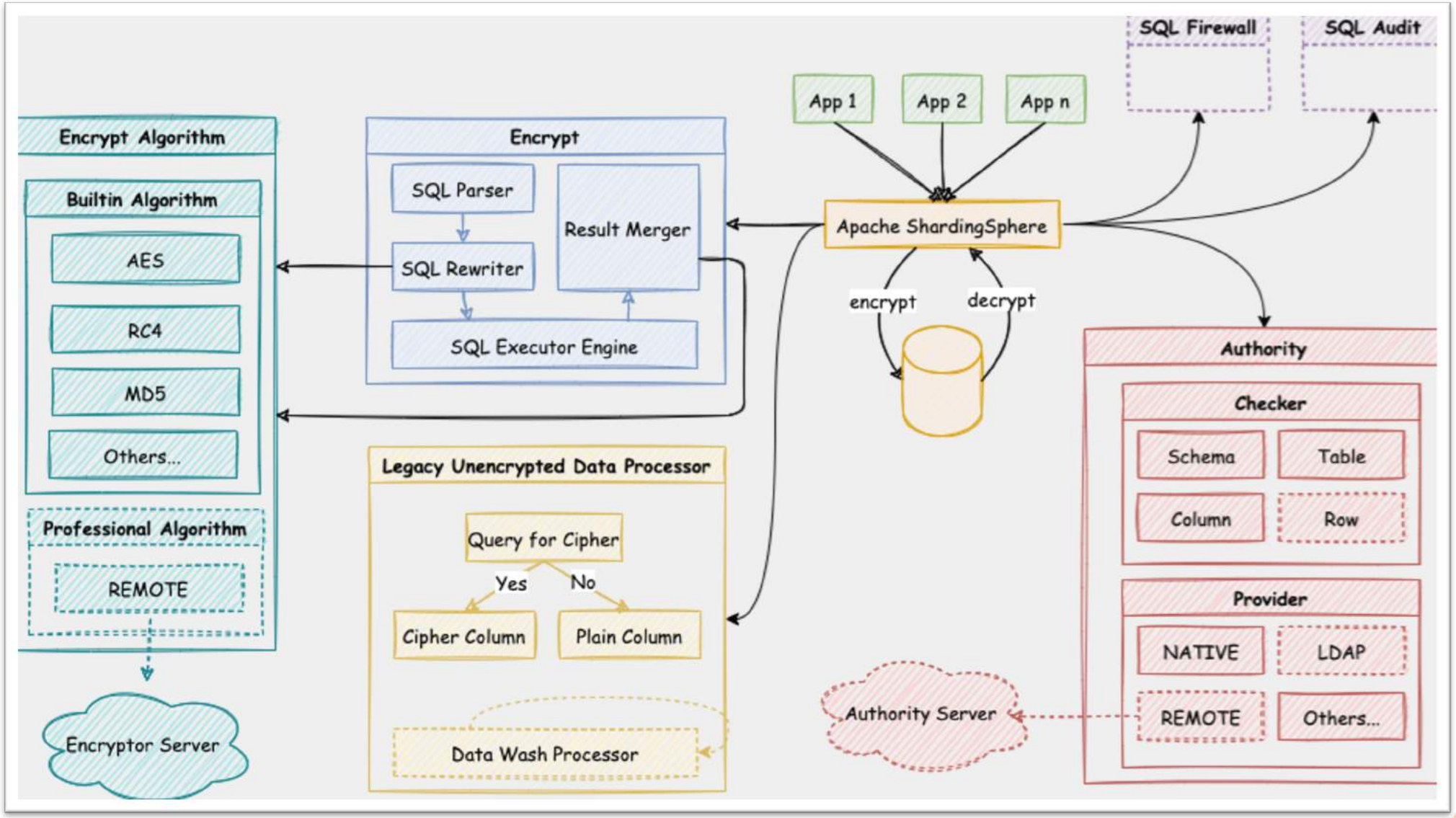
基于 Database Plus 架构以及可插拔内核，ShardingSphere 提供了丰富的功能，如数据分片、读写分离及数据加密等。基于这些丰富的功能，ShardingSphere 在产品层面也提供了分布式数据库、数据安全、数据库网关和全链路压测 4 套完善的解决方案。



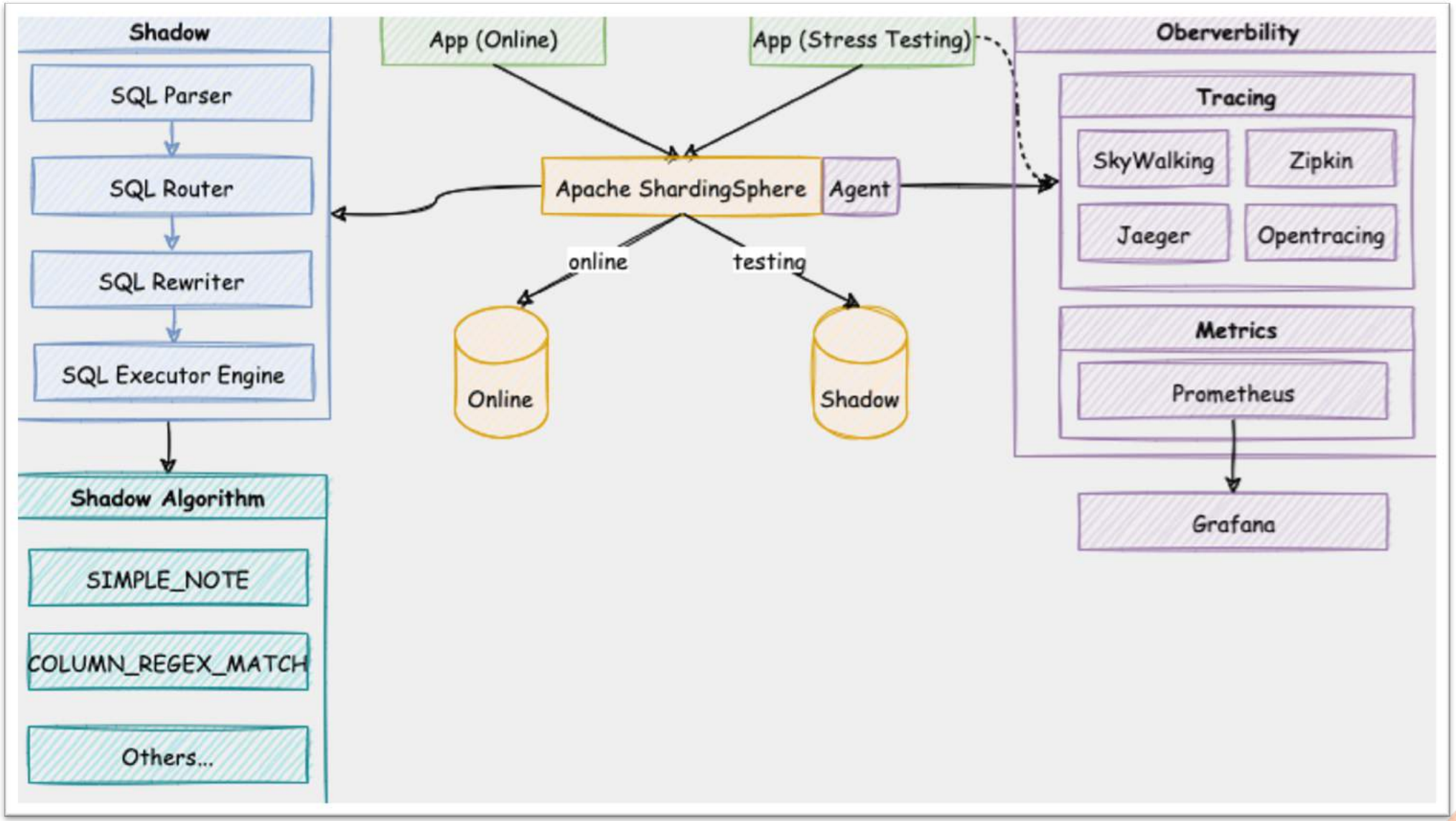
分布式数据库


Sharding Algorithm
Standard Algorithm
CLASS_BASED
INLINE
Auto Algorithm
MOD
HASH_MOD
BOUNDARY_RANGE
VOLUME_RANGE
INTERVAL
Others...
Sharding
SQL Parser
SQL Router
SQL Rewriter
SQL Executor Engine
Result Merger
App 1
App 2
App n
Apache ShardingSphere
write
read
Primary 0
Primary 1
Primary n
Replica 0
Replica 1
Replica n
Distributed Transaction
XA
Atomikos
Narayana
Bitronix
BASE
Seata
DB Discovery
Primary Check
Runtime
Periodically
Heartbeat
Primary Elect
MGR
Others...
Switch by DistSQL
Data Scaling-out
Inventory Job
Binlog Subscriber
Scaling Channel
Data Migration Validator
Data Source Switcher
resharding
crash

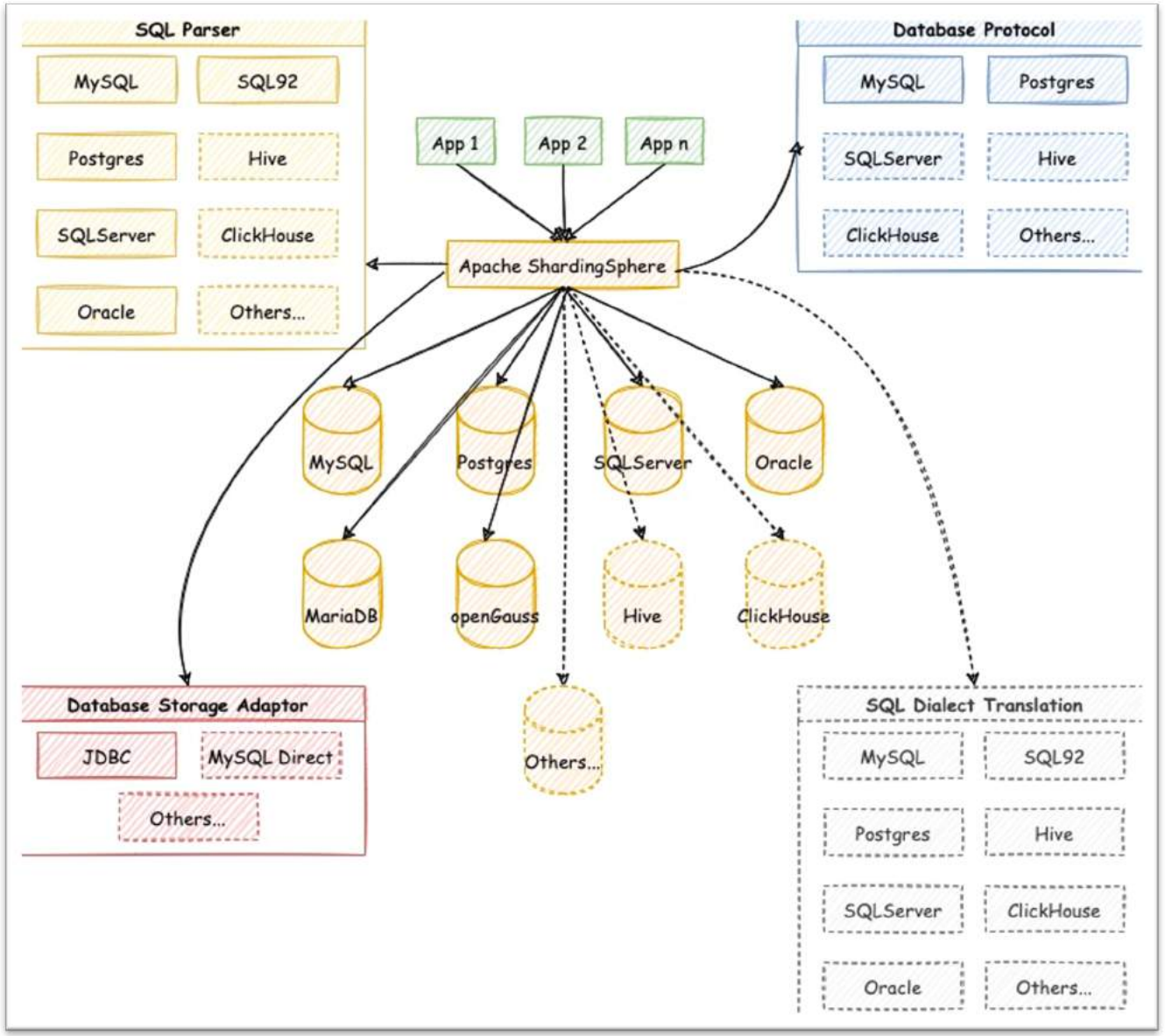
数据安全


SQL Firewall
SQL Audit
App 1
App 2
App n
Encrypt Algorithm
Builtin Algorithm
AES
RC4
MD5
Others...
Professional Algorithm
REMOTE
Encryptor Server
Encrypt
SQL Parser
Result Merger
SQL Rewriter
SQL Executor Engine
Apache ShardingSphere
encrypt
decrypt
Legacy Unencrypted Data Processor
Query for Cipher
Yes
No
Cipher Column
Plain Column
Data Wash Processor
Authority Server
Authority
Checker
Schema
Table
Column
Row
Provider
NATIVE
LDAP
REMOTE
Others...

全链路压测


Shadow
SQL Parser
SQL Router
SQL Rewriter
SQL Executor Engine
Shadow Algorithm
SIMPLE_NOTE
COLUMN_REGEX_MATCH
Others...
App (Online)
App (Stress Testing)
Apache ShardingSphere
Agent
online
testing
Online
Shadow
Oberverbility
Tracing
SkyWalking
Zipkin
Jaeger
Opentracing
Metrics
Prometheus
Grafana

数据网关


SQL Parser
MySQL
SQL92
Postgres
Hive
SQLServer
ClickHouse
Oracle
Others...
App 1
App 2
App n
Apache ShardingSphere
Database Protocol
MySQL
Postgres
SQLServer
Hive
ClickHouse
Others...
MySQL
Postgres
SQLServer
Oracle
MariaDB
openGauss
Hive
ClickHouse
Others...
Database Storage Adaptor
JDBC
MySQL Direct
Others...
SQL Dialect Translation
MySQL
SQL92
Postgres
Hive
SQLServer
ClickHouse
Oracle
Others...

ShardingSphere

谢谢观看

技术干货

加入交流群