Trino引擎在小米的应用实践

周渝豪-小米-大数据软件工程师

DataFunSummit # 2023

目录
CONTENT

01

架构定位

Trino历史

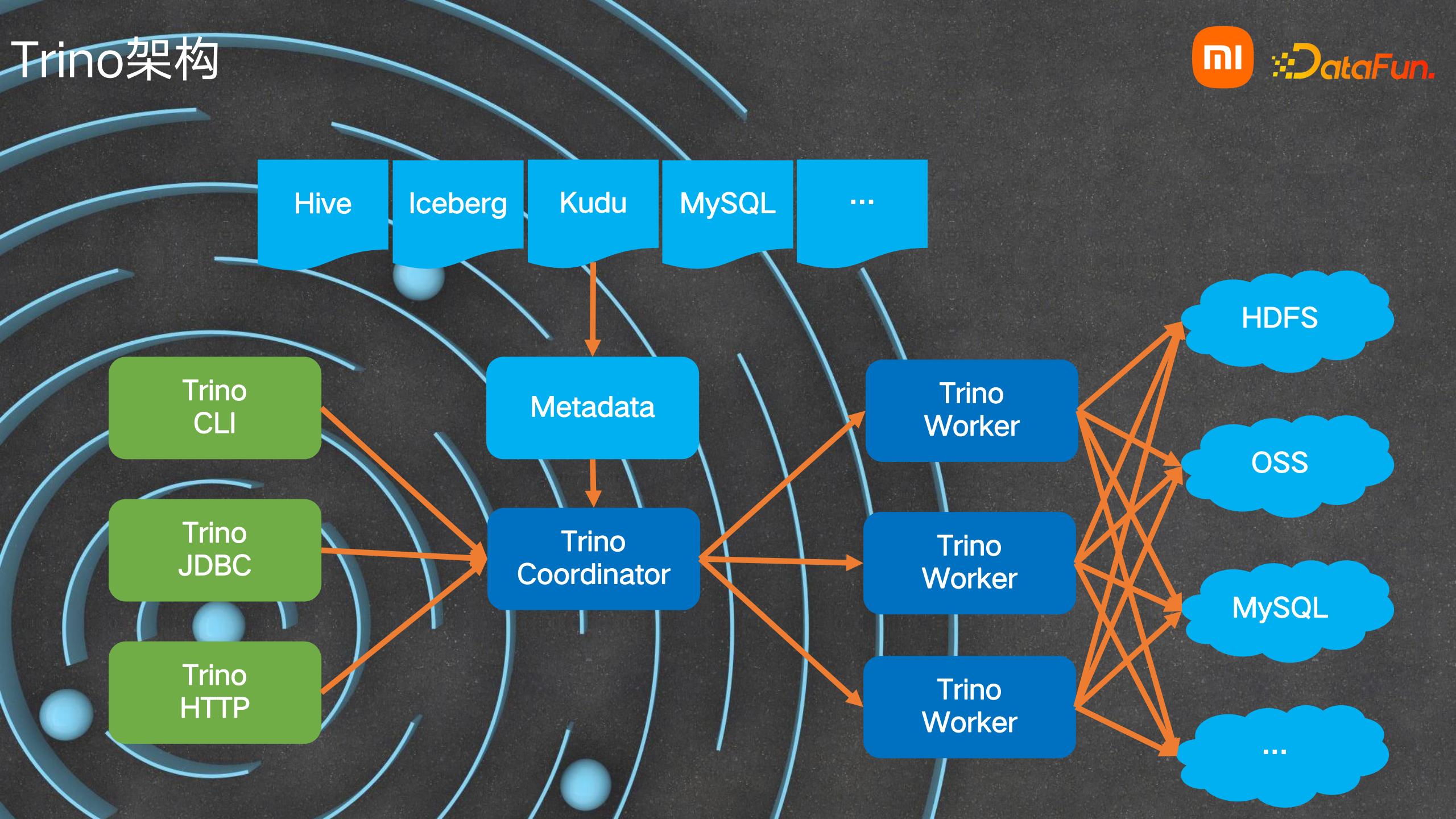
Presto

Trino

Trino, a query engine that runs at ludicrous speed

Fast distributed SQL query engine for big data analytics that helps you explore your data universe.

Trino架构


Hive
Iceberg
Kudu
MySQL
...
Trino CLI
Trino JDBC
Trino HTTP
Metadata
Trino Coordinator
Trino Worker
Trino Worker
Trino Worker
HDFS
OSS
MySQL
...

Trino优缺点

架构清晰

Master-Slave架构　　存算分离　　无依赖系统独立

速度快

全内存运算　　Pipeline模式　　动态代码生成

扩展性强

可拔插Connector　　跨源联邦查询　　方便自定义函数

内存要求高

单个节点一般32G以上

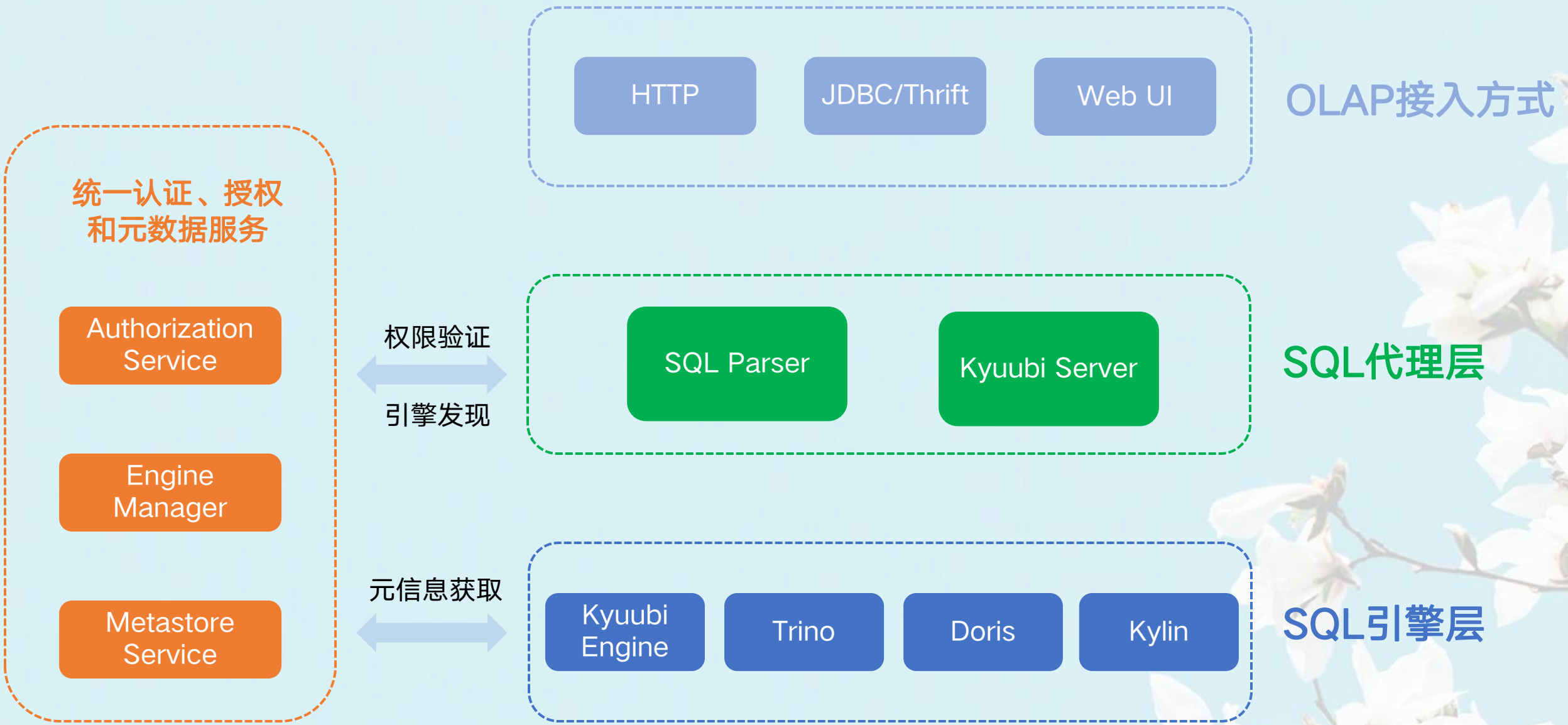
失败容忍低

基于内存模式设计

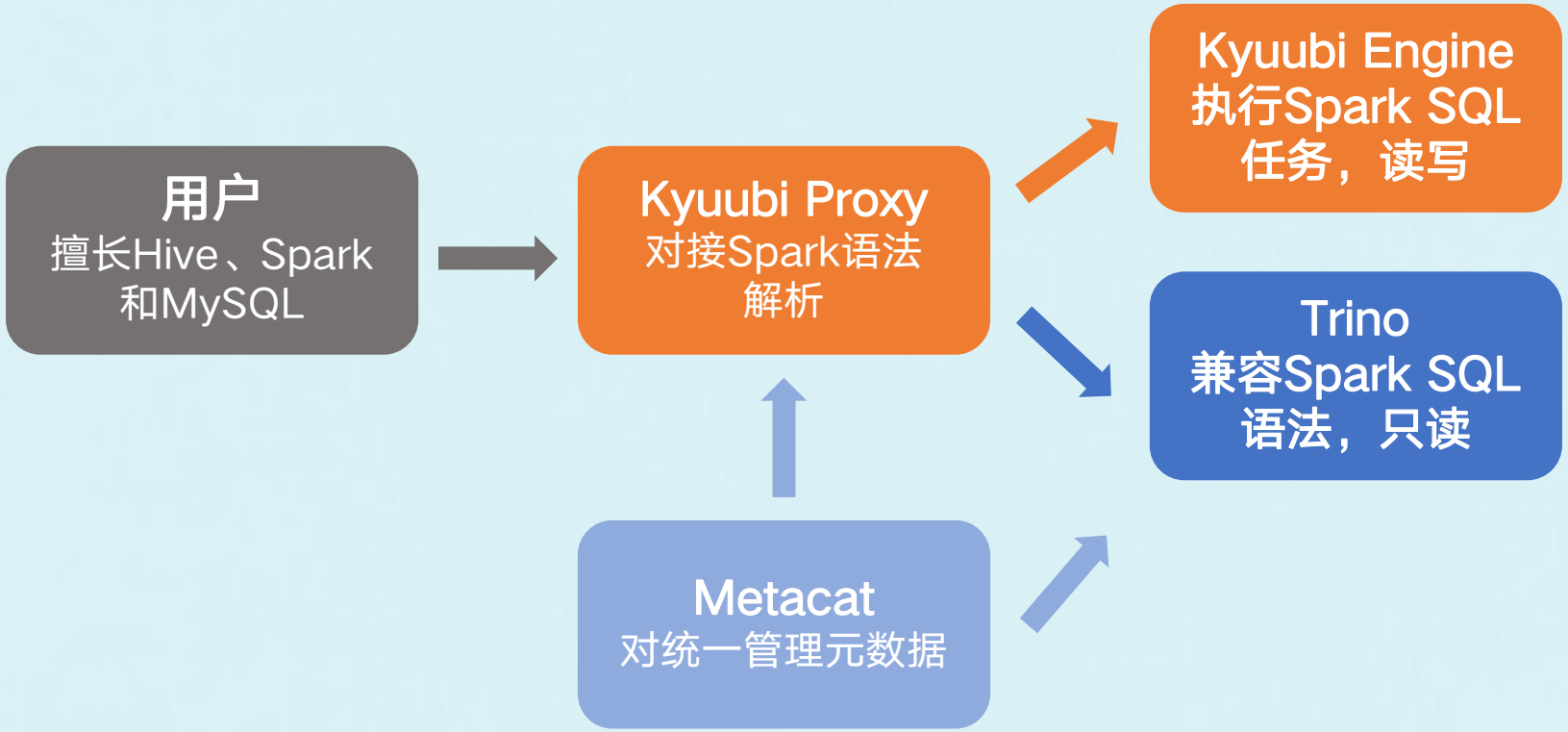
并发能力不足

单主节点加上内存限制

小米的OLAP架构


HTTP
JDBC/Thrift
Web UI
OLAP接入方式
统一认证、授权
和元数据服务
Authorization Service
Engine Manager
Metastore Service
权限验证
引擎发现
SQL Parser
Kyuubi Server
SQL代理层
元信息获取
Kyuubi Engine
Trino
Doris
Kylin
SQL引擎层

小米的Trino定位



- 统一使用Spark SQL语法
- Trino只用于查询操作
- Kyuubi负责接入和权限控制
- Metacat统一元数据管理

小米Trino目标

让大数据更快到达用户眼前。

大数据

不仅是数据量大，还有种类来源丰富：

- Hive数仓
- Kudu存储引擎
- Iceberg数据湖
- 关系型数据库

更快

更快的接入新数据源，更快的分析处理速度。

内部场景测试相比Spark SQL，Trino能够提供5-10倍的性能提升。

眼前

在小米Trino主要被用来提升用户可见部分的性能提升，比如数据预览，即席查询，统计报表等等。

小米Trino发展

紧跟社区的步伐，每年进行一次大版本升级

02

主要工作

主要工作

01

核心能力

- 兼容Spark SQL
- 优化Iceberg使用

02

扩展能力

- 动态Catalog加载
- 动态UDF加载

03

运维能力

- 审计日志和历史服务
- 集成测试和自动发布

核心能力 兼容Spark SQL

Tips: 小米统一使用Spark SQL作为标准OLAP查询语言

Spark SQL

```
SELECT ‘1’ `id`, “Bob’s Son” `name`
FROM test
WHERE array(1,2,3)[0] = 1;
```

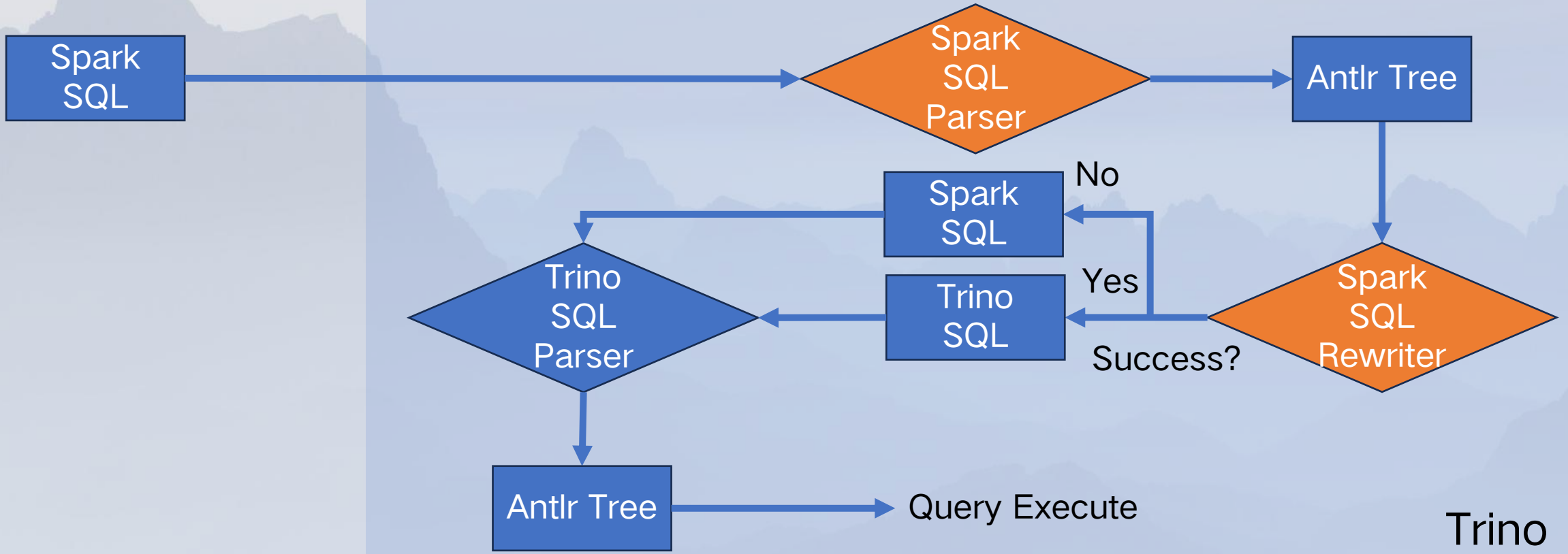
Trino SQL

```
SELECT ‘1’ “id”, ‘Bob’’s Son’ “name”
FROM test
WHERE array[1,2,3][1] = 1;
```

- 字面量和标识符的引用方式不同
  - Spark: 单双引号字面量，反引号标识符
  - Trino: 单引号字面量，双引号标识符
- 语法不同，比如Array
  - Spark: Array()
  - Trino: Array[]
- 语义不同，比如Array
  - Spark: 下标从0开始
  - Trino: 下标从1开始

Spark和Trino的SQL语法和语义都存在不同

核心能力 兼容Spark SQL SparkSqlRewriter




```
2023-10-23T15:33:15.664+0800    INFO    dispatcher-query-6
origin sql: SELECT '1' `id`, "Bob's Son" `name`
FROM test
WHERE array(1,2,3)[0] = 1
rewrite sql: SELECT '1' "id", 'Bob''s Son' "name"
FROM test
WHERE ARRAY[1, 2, 3][0] = 1
2023-10-23T15:33:15.868+0800    INFO    dispatcher-query-7
```

Spark SQL Rewriter解决了80%以上的语法兼容问题

核心能力 兼容Spark SQL 隐式转换

类型转换

- INT转成BIGINT: 加宽类型(Type widening)
- STRING转INT: 翻译类型(Type translation)

显/隐式转换

- 显示转换: 用CAST进行类型转换，比如CAST(1 AS DOUBLE)
- 隐式转换: 计算引擎自动根据需要的类型对数据进行转换，比如1/‘2’(String转换成Int)

Trino只支持加宽类型的隐式转换

Trino隐式转换支持

```
trino:default> select 1/'2';
Query 20231015_130631_01579_cn_examine01_fqujw failed: line 1:9:
Cannot apply operator: integer / varchar(1)
select 1/'2'

trino:default> set session use_spark_syntax=true;
SET SESSION
trino:default> select 1/'2';
 _col0
-------
   0.5
(1 row)
```

- session参数use_spark_syntax
- 配置级别参数use-spark-syntax
- 能够控制开启SparkSqlRewriter和隐式转换等

核心能力 兼容Spark SQL

99.6%

兼容率

尚未支持:

- ANTI/SEMI JOIN
- Hints语法
- Table-valued Functions
- 部分Spark函数

核心能力 优化Iceberg使用

Iceberg

- 适用于大型分析数据集的开放表格式
- 支持事务性，模式演进，隐式分区和行级更新等
- 适用于各种云存储和HDFS等

Trino on Iceberg问题

- 读取内存占用高，容易导致集群OOM
- 对Timestamp处理和Spark不一致(非错误)
- 表的读取存在正确性和性能问题

- 降低读取Iceberg元数据内存需求
- 优化Trino计算过程内存统计
- 支持按照session时区读取timestamp
- 支持高效读取Iceberg行级更新表
- 修复Iceberg表读取列错误问题

核心能力 优化Iceberg使用 timestamp时区

Timestamp
1970-01-01
00:00:00

Unix timestamp
0

Timestamp类型一般与Unix时间戳对应

Spark

Timestamp
1970-01-01
08:00:00

Spark会将Timestamp转换成本地时间

Iceberg

Table Schema	Parquet(Int64)
Timestamp	Int64
1970-01-01 00:00:00	0

Trino

Timestamp
1970-01-01
00:00:00

Iceberg定义使用Unix timestamp存储timestamp类型数据

Trino会直接使用Timestamp时间

为保持用户查询结果一致，Trino也修改为转换成本地时间

核心能力 优化Iceberg使用 行级更新优化

Iceberg Table `users`

id(key)	name	age
1	Alice	18
3	Bob	20

Iceberg Data File

part000.parquet

```
INSERT INTO users /*+ OPTIONS('upsert-enabled'='true') */
SELECT 1, Alice, 20;
```

Iceberg Data File

part001.parquet

id(key)	name	age
1	Alice	20

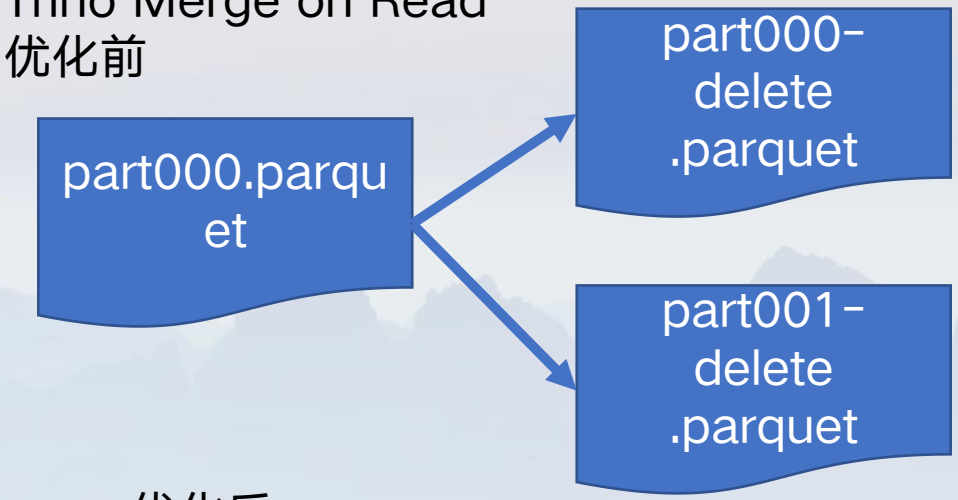
Iceberg Delete File

part000-delete.parquet

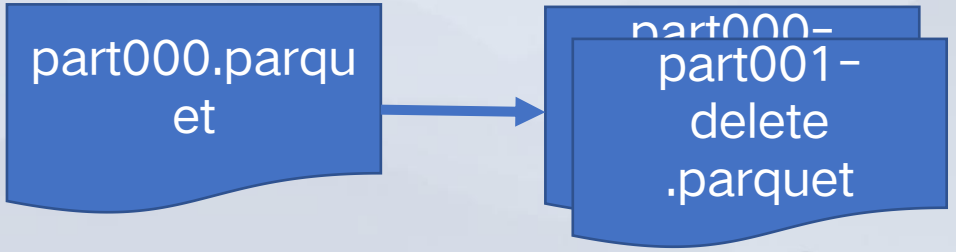
Position Delete: part000.parquet,1
Equality Delete: part000.parquet,id=1

Trino Merge on Read

优化前

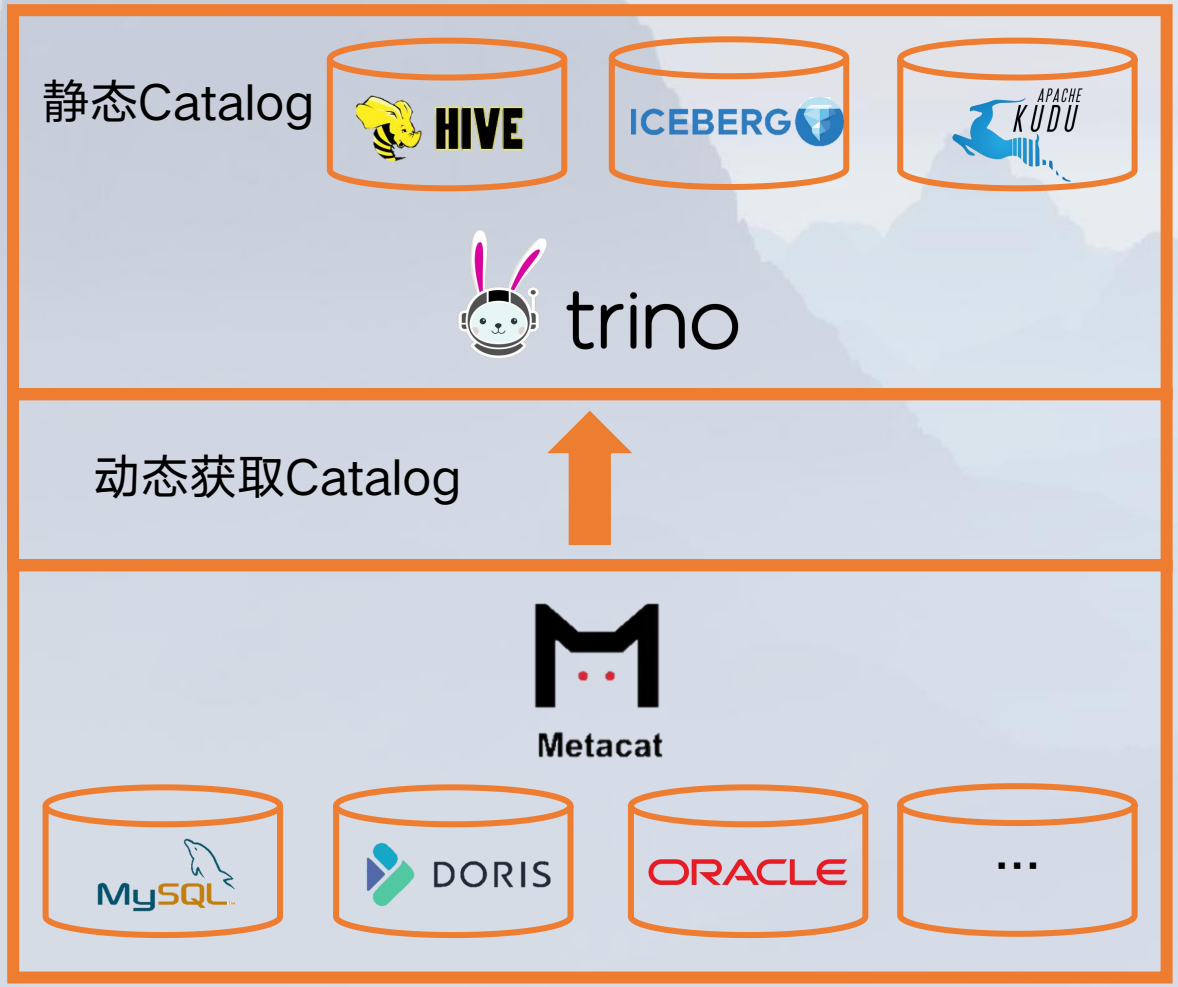


优化后



- 将相同Schema的equality-delete文件先合并再进行merge，加快读取速度。
- 部分场景能将查询耗时由数十分钟降低到几十秒。

扩展能力　动态Catalog加载



社区实现

- Worker从Coordinator动态获取Catalog
- Coordinator暂不支持动态注册Catalog
- Worker负载高时获取Catalog失败

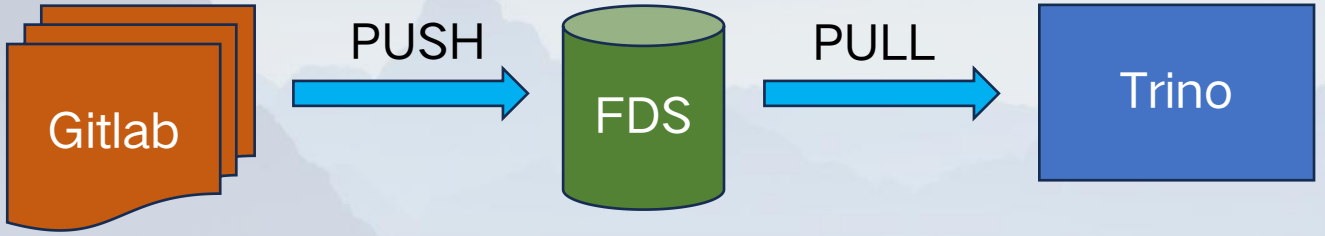
小米实现

- 所有节点从Metacat获取动态Catalog
- 零失败并发加载数千Catalog
- 启动预加载时间随Catalog数线性增加

Lazy?

扩展能力

动态UDF加载



插件模式

使用Trino可拔插的Plugin模式开发的UDF项目

可扩展

发布订阅

Gitlab-CI打包发布到对象存储，运行中引擎自动检测更新后加载

降低耦合

实时生效

流程全自动化，触发上线后整体在5min之内全集群生效

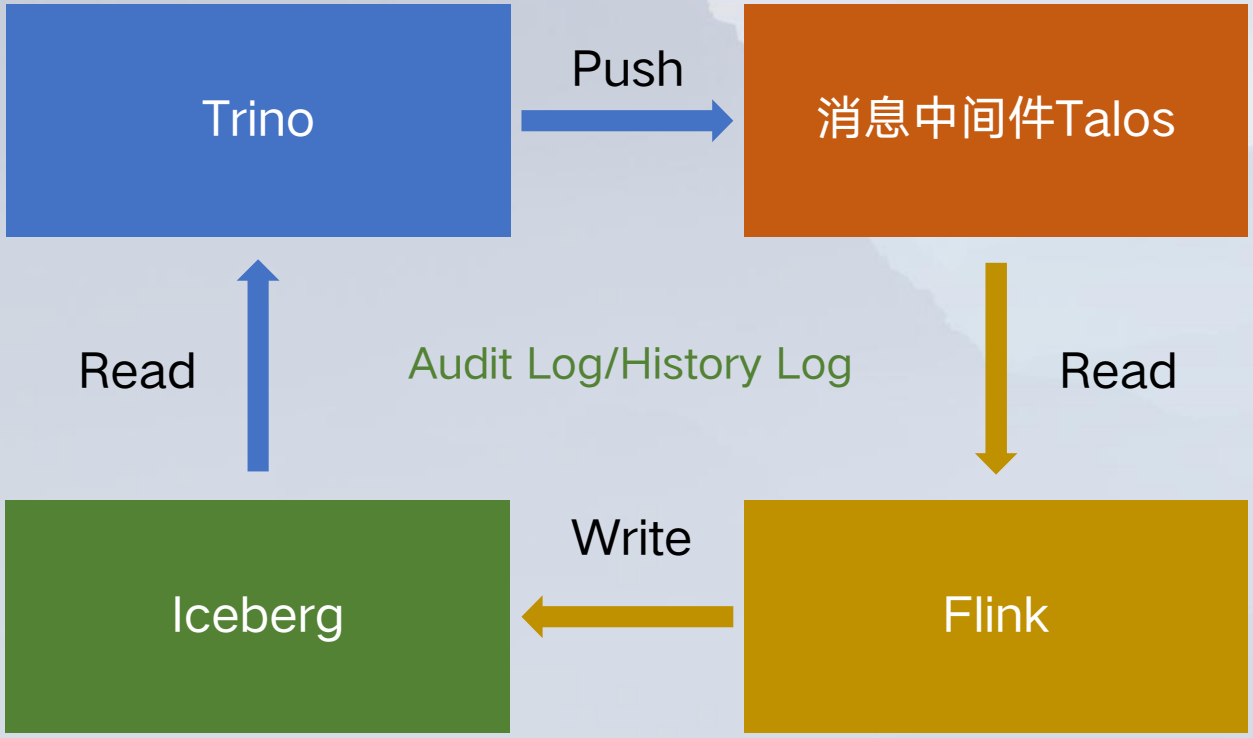
自动化

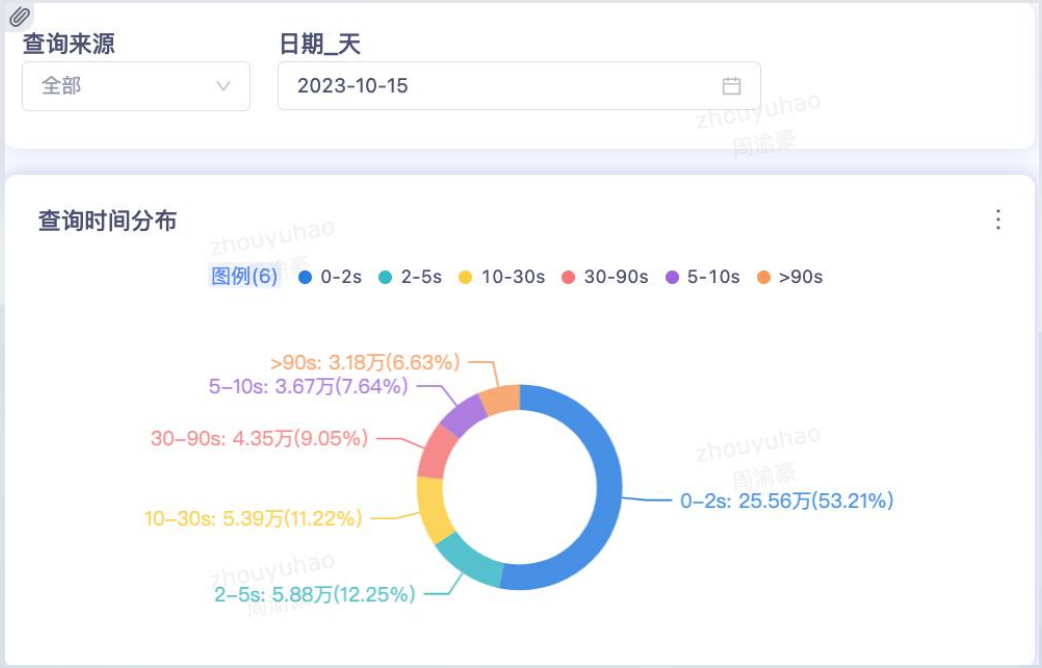
版本管理

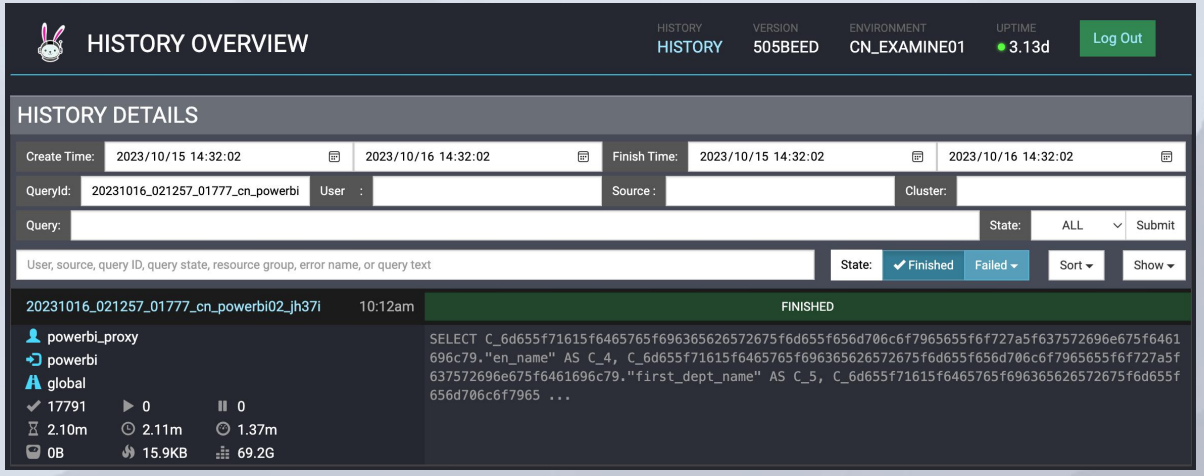
基于代码Tag进行版本管理，回退只需要重新触发Tag的流水线

可回退

运维能力 审计日志和历史服务







运维能力 集成测试和自动发布

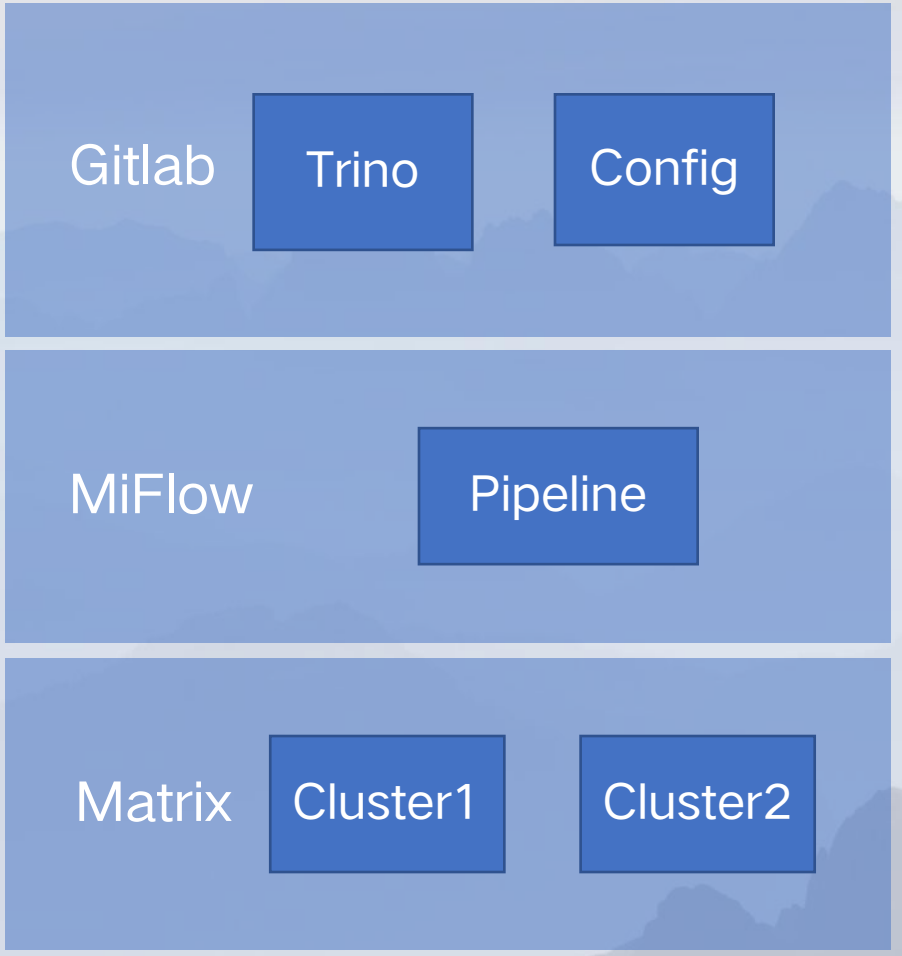
Trino-Examine项目

- 测试SQL的语法语义正确性
- 直连Trino和Kyuubi运行查询
- 从SQL文件、审计日志表中获取语句进行查询

语义测试:

采用Spark Hash算法对每行数据进行运算并求和

```
SELECT SUM(result) FROM (
    SELECT HASH(*) result FROM (
        ……
    )
);
```



其他工作

- Hive Metastore使用连接池提升元数据访问稳定性
- Hive Metastore元数据缓存提升查询效率
- 增加管理接口和Metrics指标管理集群实时状态
- 支持使用Nacos管理集群资源组配置
- 支持集群的快速重启和worker优雅滚动重启
- ...

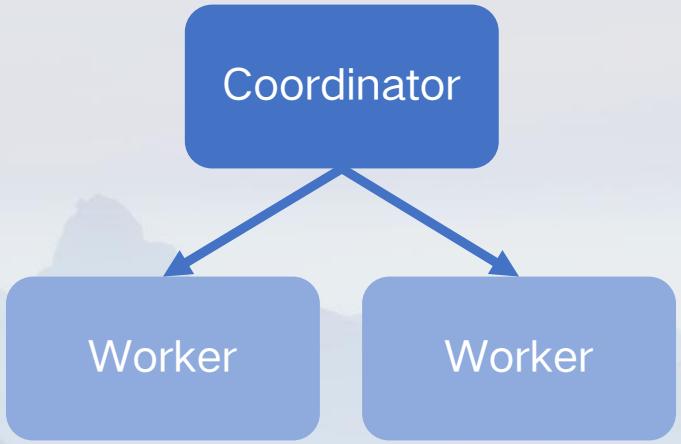
03

应用场景

应用模式 多集群模式

优点：
- 解决了单点故障，容错提升
- 集群少，便于维护
- 大集群，支持更多更复杂的查询

缺点：
- 实现复杂，容易出现不一致问题
- 大集群对Coordinator配置要求更高
- 资源隔离效果更难把握

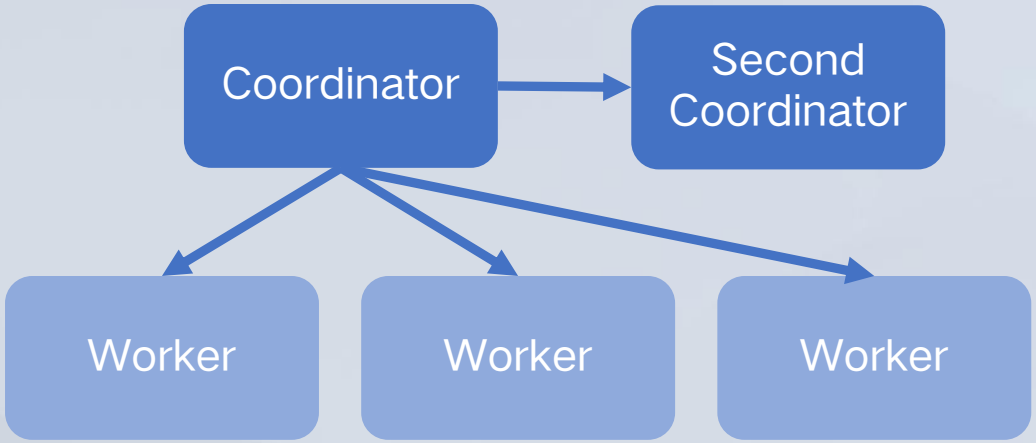


单点故障？

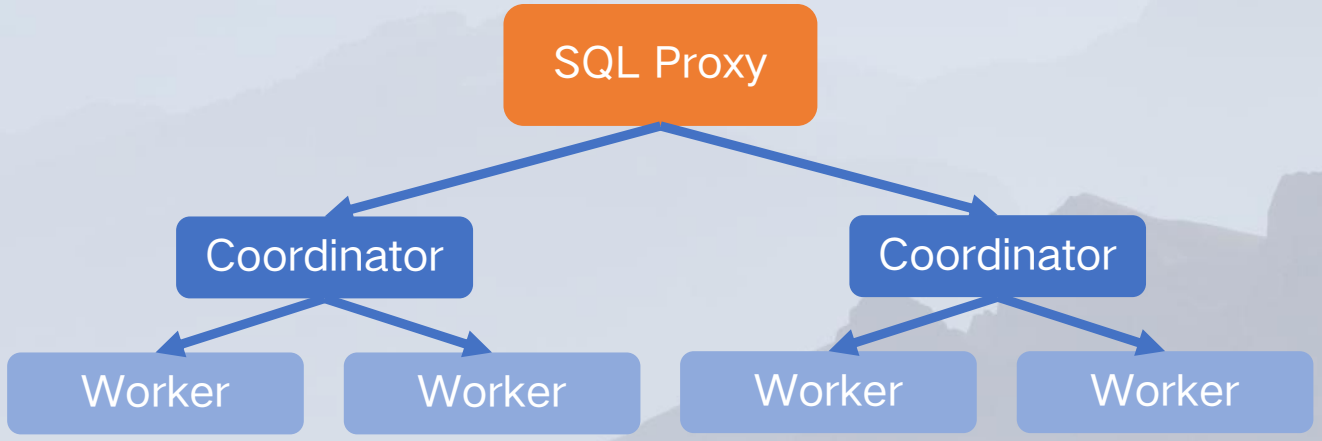
优点：
- Trino社区推荐方式，实现简单
- 多集群可以利用集群进行资源隔离
- 根据不同的需求定制集群

缺点：
- 只能通过失败重试保证容错
- 多集群的运维管理更为复杂
- 对长时间复杂查询不能保证成功率

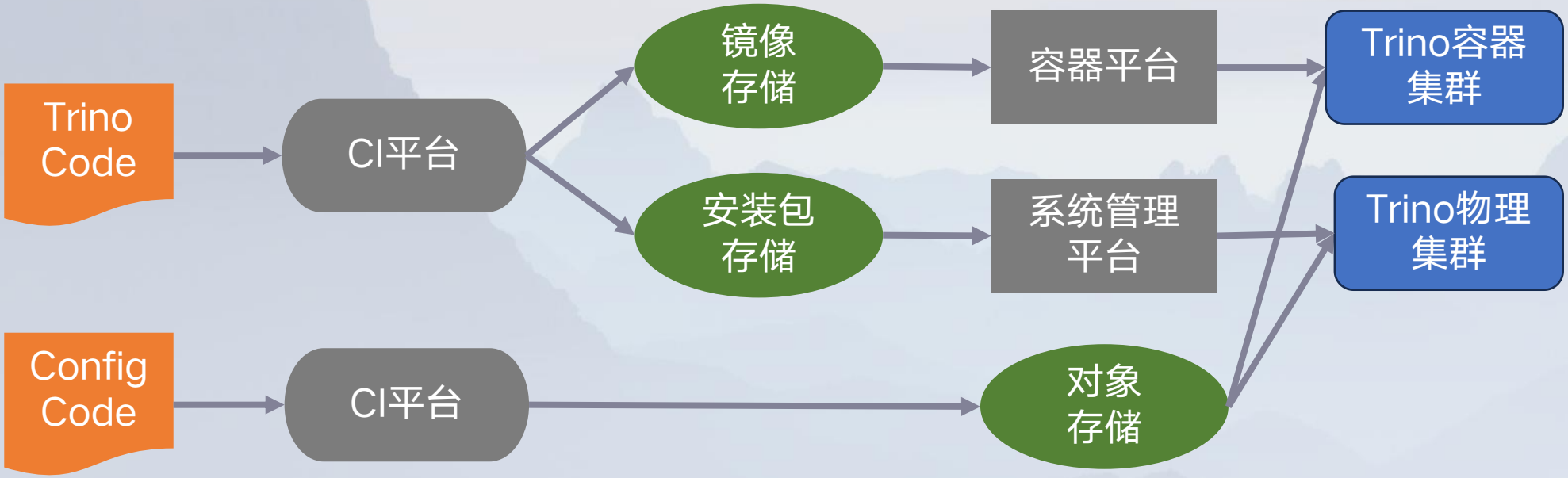
多Coordinator大集群模式



代理多集群模式



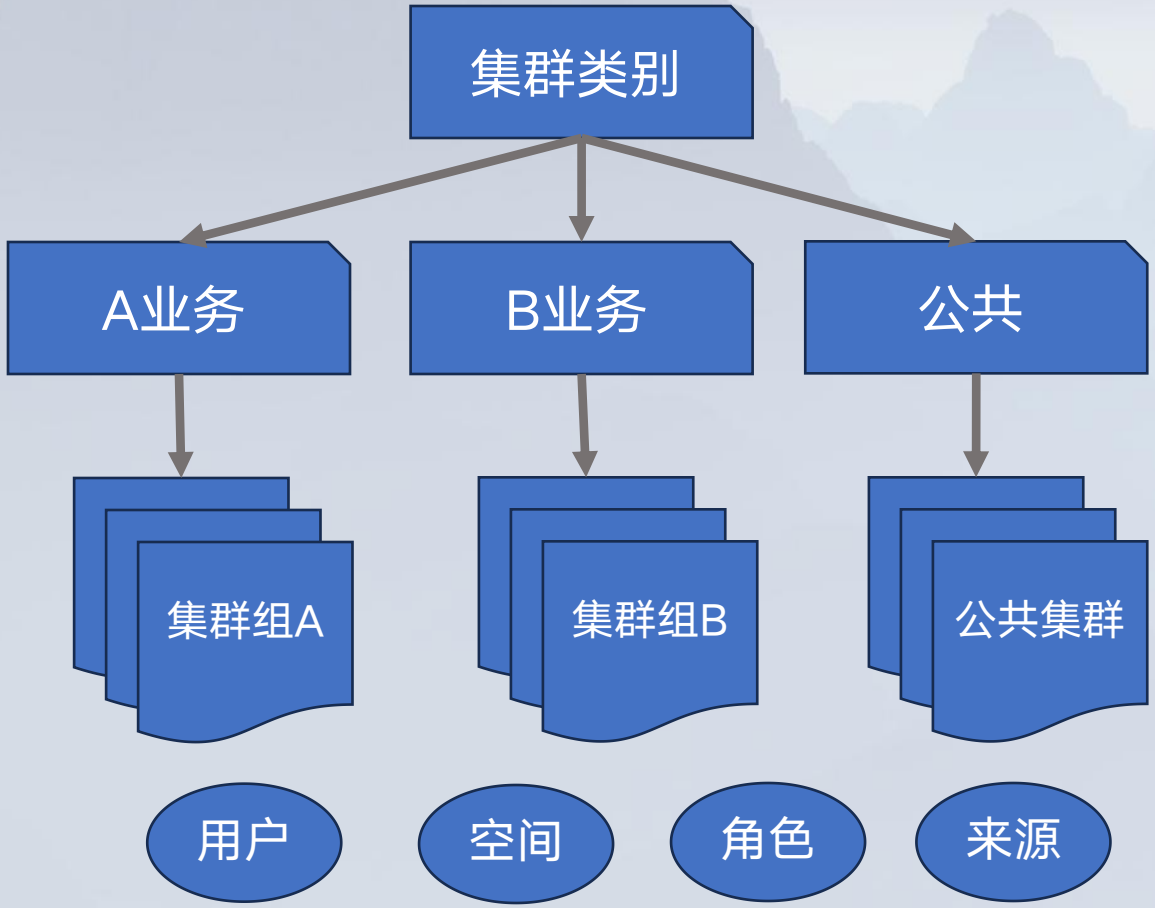
应用模式 多集群管理



开发管理

- 服务配置解耦
- 统一集群管理
- 自动发布流程

应用模式 多集群管理



集群类别：根据不同的业务需求来决定使用的一组集群

路由规则：根据不同的条件决定查询使用集群组中具体的集群

资源组配置：限制单集群的资源的占用，保证良性竞争和集群稳定性

路由和资源管理

- 按业务区分集群，保证资源的隔离
- 动态路由规则配置，支持黑白名单，保证负载均衡
- 资源组配置，单集群限制资源占用，保证稳定性

应用场景 即席查询

数据预览

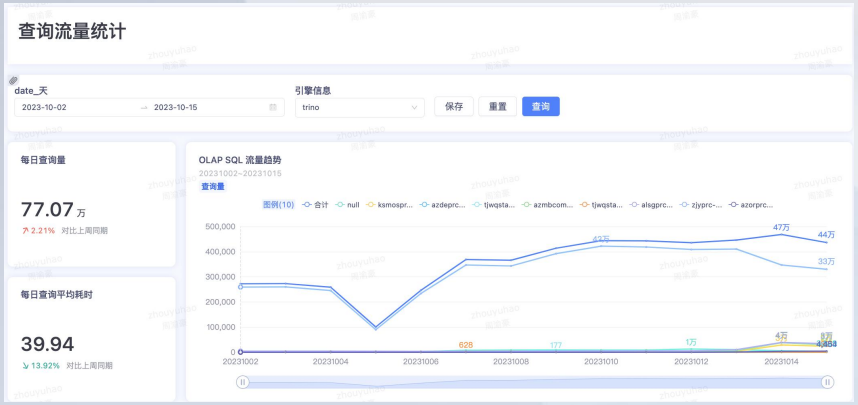
即席查询

场景说明：

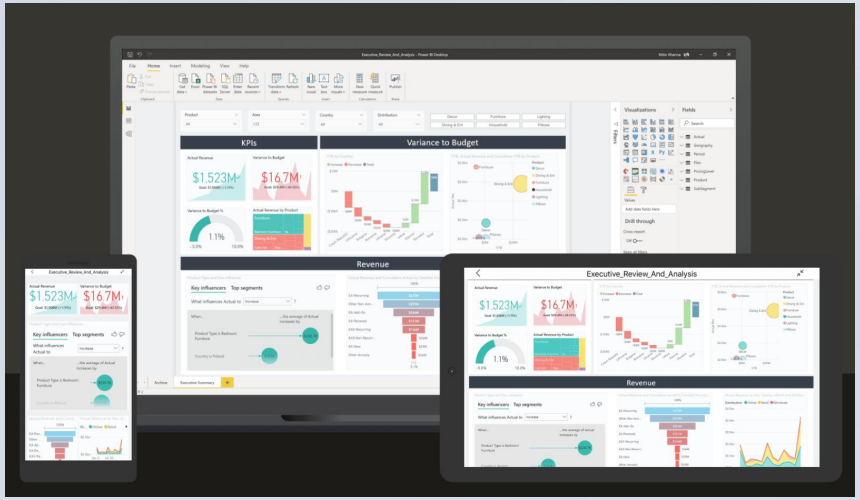
- 追求的极致的速度，秒级
- 限制查询运行时间<5min
- 独立的集群，保证资源隔离
- 集群内严格限制大查询
- 快速失败

应用场景 BI分析

小米BI数鲸



微软Power BI



场景说明：

- 较快的产出看板，分钟级
- 限制查询运行时间<10min
- 随表报数增长的集群规模
- 存在明显的周期性，定时扩缩容保证效率
- 并发较高，需要负载均衡
- 开启容错，尽可能保证成功率

应用场景 分析业务加速

小米业务示例：

- 数据质量检查
- 用户画像分析
- 数据推送任务
- 销售统计任务
- ……

场景说明：

- 根据业务不同，需要的集群规模和配置不同
- 一般查询较为固定，需要资源也因此固定
- 查询复杂度高耗时较长，小时级别，一般<1h
- 并发度可控，成功率要求高，保证产出时间
- 对集群的内存和CPU要求高，一般用物理机

04

未来规划

未来规划

感谢观看

DataFun.