百度智能云

BMR 文档

目录

功能发布记录

发布时间	功能概述
2025-06	• 发布集群巡检，支持集群、节点、服务维度问题检查，并针对性为用户提出治理建议 发布应用洞察，支持事前、事中、事后的应用问题检测和统计分析。
2025-04	• 下线上海区域。
2025-01	• 发布BMR 3.6.0，集成Paimon，升级Ranger、Flink等组件。
2024-11	• 发布 BMR 3.5.0，支持 Hadoop-存算分离集群，降低存储成本。 • 发布节点管理功能，支持缩容，提升运维灵活性。
2022-12	• 上线Infinite_spark集群类型，发布BMR Impala Infinite_spark 3.2.2镜像，集成Infinite_spark和remote_shuffle组件，Infinite_spark在开源spark框架的基础上进行优化，离线计算速度提升30%，remote_shuffle开启后提升大规模离线计算场景稳定性，镜像组件版本：infinite_spark 3.2.2、hive 3.1.3、tez 0.10.1、hdfs 3.2.3、yarn 3.2.3、mapreduce2 3.2.3、zookeeper 3.4.6、ldap 2.4.48、remote_shuffle 4.1.0、rapidfs 1.0.0.1。 • 发布BMR 2.3.3镜像版本，hive、hue组件支持中文。
2022-10	• 上线Impala集群类型，发布BMR Impala 2.12.0镜像，支持impala 2.12.0、kudu 1.10.0。 • 上线全托管集群、半托管集群两种托管方式，支持用户在创建集群时根据需要选择不同托管方式。 • 发布BMR 3.2.1镜像版本，在3.2.0镜像基础上，升级hudi版本为1.11.0。 • 发布BMR 2.3.2镜像版本，在BMR 2.3.1镜像基础上新增hudu 0.11.0。
2022-09	• 上线服务管理模块，支持用户在控制台统一管理集群，包括服务组件的启停、重启以及组件参数配置，并支持查看启停操作以及配置变更操作相关记录。 • 发布BMR 3.2.0镜像版本，在BMR 3.1.0镜像基础上新增Rapidfs 1.0.01。
2022-08	• BMR更新高可用架构，由2Master调整为3Master结构，提供更加稳定和高可用的大数据集群管控服务。 • 发布BMR 3.1.0镜像版本，集成百度侧优化改造后的spark引擎，支持spark 3.2.0。
2022-07	• BMR发布全新BMR1.6.0版本，支持Hadoop 2.8.5、Spark 2.4.8、Hive 2.3.9等版本。 • 支持删除已释放集群和修改标签和批量编辑标签功能。 • 默认不再开启dns解析按钮。 • 提供集群名称、实例ID、作业ID等复制功能。 • 修复BMR 3.0.0镜像选中spark后不关联hive的问题。

2022-06	• BMR发布全新BMR3.0.0版本，升级Hadoop为3.2.3，Spark为3.2.0，Hive 3.1.3，Tez 0.10.1，Flink 1.13.6，新增Hudi 0.10.1。
2022-05	• BMR 发布 BMR 2.3.1版本，升级Impala版本为3.4.0，Kudu版本为1.14.0。
2022-04	• BMR 发布 BMR HBase 2.2.1版本，支持HBase on BOS 读写分离集群功能，详情见HBase组件介绍。
2022-04	• BMR 发布 BMR Kafka 2.7.2 版本,同时对副本同步参数进行了调整。 • num.replica.fetchers=12 • replica.fetch.max.bytes=5242880 • replica.fetch.wait.max.ms=300
2022-03	• BMR 发布 BMR 2.3.0 镜像,支持Ooize 5.1.0与Hue 4.8.0。
2021-11	• BMR 发布 BMR 2.2.2 镜像,支持Flink 1.12.4版本。
2021-09	• BMR支持用户对Master、Core、Task、Client节点主动绑定和解绑EIP。
2021-08	• BMR 更新Hbase版本为Hbase 2.2.7版本，Hbase集群模板发布新版本：BMR HBASE 2.2.0。 • BMR ClickHouse 模板发布新版本：BMR Clickhouse 21.8.3，同时支持ClickHouse 高可用版本。 • BMR 监控大盘新增Hbase监控，支持Hbase表级数据趋势监控和告警。
2021-04	• BMR 2.2.0 版本新增组件kudu • BMR 新版安全组上线，支持用户新建集群时绑定用户自己的安全组，并且支持新建集群的安全组在VPC安全组中控制安全组规则 • BMR 上线集群标签功能，支持用户在创建时为集群添加标签，方便用户区分和管理集群，同时用户可以在云上标签管理中绑定和解绑BMR集群资源
2021-03	• BMR在苏州、上海Region 上线BMR 1.6.0 版本 支持hadoop2.8.5版本，同时适配了spark、hive、hbase、hue等组件
2021-02	• BMR 新增监控运维大盘，提供集群仪表盘和集群监控、主机监控和服务监控，支持查看主机层监控详情和主机层面的指标筛选功能 • Flink 升级版本至1.11.2 • BMR上线 BMR 2.2.0 版本,支持hadoop 3.1 版本，同时适配升级了其他组件 • BMR Kafka上线BMR Kafka 1.0.1，BMR Kafka 1.1.0，BMR Kafka 2.0.1和BMR Kafka 2.5.1多个版本
2021-01	• BMR 上线 2.0.0 版本 支持hadoop 3.1 版本 • BMR 上线 1.4.0 版本 支持hadoop 2.7.1 版本，同时适配presto、azkaban、hive、spark等组件 • BMR 上线 1.2.0 版本 支持hadoop 2.7.1 版本

2020-12	BMR 发布独立模板 ClickHouse，多用于联机分析（OLAP）场景，可提供海量数据的存储和分析，版本 BMR ClickHouse 1.0.0，ClickHouse版本为20.5.3.27
2020-04	上线计费变更功能，支持预付费与后付费集群互转
2020-03	上线配置变更功能，可以对一个正在运行的集群，对虚机的cpu和内存规格进行调整
2020-01	上线2.1.0镜像，新增kafka组件，spark版本升级到2.4.2
2019-12	新增Flink、Impala、Druid组件
2019-11	上线弹性伸缩功能，可以自动调整task节点规模
2019-10	在创建集群时，新增client节点类型，可以作为独立的集群管控节点及作业提交节点，可以提高集群的稳定性及安全性。
2019-09	在创建集群时，支持将BOS的bucket作为Hbase的deepstorage。提供数据存储可靠性及使用灵活性
2019-06	新增BMR节点并更新价格
2019-05	提供Java SDK实现创建管理集群、实例、实例组及作业等功能
2019-04	新增Scala场景教程。以使用Scala为例介绍数据计算过程通过BMR的Spark Streaming连接百度消息服务
2018-12	• 提供备用节点，提供集群的高可用性 • BMR从1.0.0版本开始支持创建安全类型的集群，即集群中的开源组件以Kerberos的安全模式启动，在这种安全环境下只有经过认证的客户端才能访问集群中的服务（如HDFS，HIVE 等） • 接入IAM主子账户鉴权，细化用户权限颗粒度，便于根据业务需要为不同职责的用户分配不同的权限 • 新增Hadoop集群管理系统。Hadoop Manger是BMR产品为用户提供的细粒度hadoop管理平台，提供集群监控、服务管理、hadoop多租户管理功能
2018-11	• 新增支持集群安全模式，提供备用节点，提供集群的高可用性 • 支持多用户访问控制。用户访问控制实现了企业或组织内部不同人员乃至不同部门协同地管理和使用BMR资源的功能。在BMR中，我们将集群和定时任务视为资源。协同工作的方式是主用户需要创建子用户，并为子用户分配访问策略

发行版本

本文介绍BMR版本的发行版本信息。

BMR 1.X系列版本信息

组件名称	BMR 1.2.0	BMR 1.4.0	BMR 1.5.0	BMR 1.6.0	备注
Hadoop	2.7.1	2.7.1	2.7.7	2.8.5	必选
Zookeeper	3.4.6	3.4.6	3.4.6	3.4.6	必选
Hadoop-Manager	1.0.0	1.0.0	1.0.0	1.0.0	必选
LDAP	-	-	-	2.4.48	必选
Spark	2.1.0	2.1.0	2.1.0	2.4.8	
Hive	1.2.0	1.2.0	1.2.0	2.3.9	
Tez	0.7.0	0.7.0	0.7.0	0.9.1	
Pig	0.15.1	0.15.1	0.7.0	-	
Hue	3.10.0	3.10.0	3.10.0	4.8.0	
HBase	1.1.2	1.2.0	1.1.2	1.4.9	
Ranger	0.5.0	0.5.0	-	-	
Azkaban	-	3.58.0	-	-	
Presto	-	0.219	-	-	
Impala	-	3.2.0	-	-	
Sqoop	-	1.4.6	-	1.4.7	
Ooize	-	-	-	5.1.0	
yarn	2.7.1	2.7.1	2.7.7	2.8.5	必选
hdfs	2.7.1	2.7.1	2.7.7	2.8.5	必选
zookeeper	3.4.6	3.4.6	3.4.6	3.4.6	必选
mapreduce2	2.7.1	2.7.1	2.7.7	2.8.5	必选

BMR 2.X系列版本信息

组件名称	BMR 2.1.1	BMR 2.2.1	BMR 2.2.2	BMR 2.3.0	BMR 2.3.1	BMR 2.3.2	BMR 2.3.3	BMR 2.3.4	BMR 2.3.5	备注
Hadoop	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	必选
Zookeeper	3.4.6	3.4.6	3.4.6	3.4.6	3.4.6	3.4.6	3.4.6	3.4.6	3.4.6	必选
Hadoop-Manager	1.0.0	1.0.0	1.0.0	1.0.0	1.0.0	1.0.0	1.0.0	1.0.0	1.0.0	必选
LDAP	2.4.48	2.4.48	2.4.48	2.4.48	2.4.48	2.4.48	2.4.48	2.4.48	2.4.48	必选
Hudi	-	-	-	-	-	0.11.0	-	0.12.2	0.12.2	必选
Spark2	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.4	2.4.8	2.4.8		
Hive	3.1.0	3.1.0	3.1.0	3.1.0	3.1.0	3.1.0	3.1.0	3.1.3		
Tez	0.9.1	0.9.1	0.9.1	0.9.1	0.9.1	0.9.1	0.9.1	0.10.1		
Pig	0.17.0	0.17.0	0.17.0	0.17.0	0.17.0	0.17.0	0.17.0	0.17.0		
Hue	-	-	-	4.8.0	4.8.0	4.8.0	4.8.0	4.8.0		
HBase	2.2.7	2.2.7	2.2.7	2.2.7	2.2.7	2.2.7	2.2.7	2.2.7		
Ranger	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0	1.2.0		
Azkaban	3.58.0	3.58.0	3.58.0	3.58.0	3.58.0	3.58.0	3.58.0	3.58.0		
Presto	0.219	0.219	0.219	0.219	0.219	0.219	0.219	0.219		
Impala	3.2.0	3.2.0	3.2.0	3.2.0	3.4.0	3.4.0	3.2.0	3.2.0		
Zeppelin	0.8.0	0.8.0	0.8.0	0.8.0	0.8.0	0.8.0	0.8.0	0.8.0		
Flink	1.11.2	1.11.2	1.12.4	1.12.4	1.12.4	1.12.4	1.12.4	1.15.3		
Airflow	1.10.11	1.10.11	1.10.11	1.10.11	1.10.11	1.10.11	1.10.11	1.10.11		
Alluxio	-	2.4.1	2.4.1	2.4.1	2.4.1	2.4.1	2.4.1	2.4.1		不支持安全模式
KUDU	-	1.13.0	1.13.0	1.13.0	1.14.0	1.14.0	1.13.0	1.13.0		集群必须为高可用架构
Ooize	-	-	-	5.1.0	5.1.0	5.1.0	5.1.0	5.1.0		
yarn	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	必选
hdfs	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	必选
mapreduce2	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	3.1.1	必选

BMR 3.X系列版本信息

组件名称	BMR 3.0.0	BMR 3.1.0	BMR 3.2.0	BMR 3.2.1	BMR 3.2.3	BMR 3.3.0	BMR 3.4.0	BMR 3.5.0	BMR 3.6.0	备注
Hadoop	3.2.3	3.2.3	3.2.3	3.2.3	3.2.3	--	--	--	--	必选
Zookeeper	3.4.6	3.4.6	3.4.6	3.4.6	3.4.6	3.4.6	3.4.6	3.4.6	3.4.6	必选
Hadoop-Manager	1.0.0	1.0.0	1.0.0	1.0.0	1.0.0	--	--	--	--	必选
LDAP	2.4.48	2.4.48	2.4.48	2.4.48	2.4.48	2.4.48	2.4.48	2.4.48	--	必选
Hudi	0.10.1	0.10.1	0.10.1	0.11.0	0.12.2	0.14.1	0.15.0	0.15.0	0.15.0	必选
Spark	3.2.0	3.2.0	3.2.0	3.2.0	3.2.0	3.2.4	3.5.1	3.5.1	--	
Hive	3.1.3	3.1.3	3.1.3	3.1.3	3.1.3	3.1.3	3.1.3	3.1.3	3.1.3	
Tez	0.10.1	0.10.1	0.10.1	0.10.1	0.10.1	0.10.2	0.10.2	0.10.2	0.10.2	
HBase	2.2.7	2.2.7	2.2.7	2.2.7	2.2.7	2.2.7	2.2.7	--	--	
Flink	1.13.6	1.13.6	1.13.6	1.13.6	1.15.3	1.15.3	1.18.1	1.18.1.2	1.20.0	
RapidFS	--	--	1.0.0.1	--	1.0.0.2	--	--	--	--	
Hdfs	3.2.3	3.2.3	3.2.3	3.2.3	3.2.3	3.2.3	3.2.3	3.2.3	3.2.3	必选
Yarn	3.2.3	3.2.3	3.2.3	3.2.3	3.2.3	3.2.3	3.2.3	3.2.3	3.2.3	必选
Mapreduce	3.2.3	3.2.3	3.2.3	3.2.3	3.2.3	3.2.3	3.2.3	3.2.3	--	必选
Spark3	3.2.0	3.2.0	3.2.0	3.2.3	3.2.3	--	--	--	3.5.1	
Iceberg	--	--	--	--	--	1.4.3	1.4.3	1.4.3	1.4.3	
Delta	--	--	--	--	--	2.0.2	3.2.0.1	3.2.0	--	
Trino	--	--	--	--	--	1.4.2	2.0.1	2.0.1	468	
Impala	--	--	--	--	--	3.2.0	3.2.0.3	3.2.0	3.2.0	
Kudu	--	--	--	--	--	1.10.1	1.10.1	1.10.1	--	
Hue	--	--	--	--	--	4.8.0	4.8.0	4.8.0	4.8.0	
Airflow	--	--	--	--	--	2.7.3	2.7.3	2.7.3	2.7.3	
Oozie	--	--	--	--	--	5.1.0	5.1.0	5.1.0	5.1.0	
Kyuubi	--	--	--	--	--	1.7.1	1.7.1	1.7.1	1.10.0	
Ranger	--	--	--	--	--	2.3.0	2.3.0	2.3.0	2.3.0	
Dolphinscheduler	--	--	--	--	--	--	3.2.0	3.2.0	3.2.0	
Hadoop-common	--	--	--	--	--	--	--	3.2.3	3.2.3	必选
Bmr-fs	--	--	--	--	--	--	--	--	3.2.3	
MapReduce2	--	--	--	--	--	--	--	--	3.2.3	
Deltalake	--	--	--	--	--	--	--	--	3.2.0	
Paimon	--	--	--	--	--	--	--	--	1.0.0	

BMR HBase 版本信息

组件名称	BMR Hbase 2.2.0	BMR Hbase2.2.1	备注
Hbase	2.2.7	2.2.7	必选
Zookeeper	3.4.6	3.4.6	必选
Hdfs	3.1.1	3.1.1	必选
Hadoop-Manager	1.0.0	1.0.0	必选
LDAP	2.4.48	2.4.48	必选
Yarn	3.1.1	3.1.1	
Mapreduce2	3.1.1	3.1.1	
Ranger	1.2.0	1.2.0	

BMR ClickHouse 版本信息

组件名称	BMR ClickHouse20.5.3	BMR ClickHouse21.12.4	BMR ClickHouse21.8.14	BMR ClickHouse22.12.6	BMR ClickHouse23.12.6	备注
clickhouse	20.5.3.27	21.12.4.1	21.8.14.5	22.12.6.22	23.12.6	
zookeeper	3.4.6	3.6.3	3.6.3	3.6.3		
hue	4.8.0	4.8.0	4.8.0	4.8.0		

BMR Impala 版本信息

组件名称	BMR Impala 2.12.0	备注
Impala	2.12.0	必选
Kudu	1.10.0	必选
Zookeeper	3.4.6	必选
Hdfs	2.7.7	必选
Hive	3.1.3	必选
Tez	0.7.0	必选
LDAP	2.4.48	必选
Hue	4.8.0	

BMR Infinite_Spark 版本信息

组件名称	BMR Infinite_Spark 3.2.2	备注
Infinite_Spark	3.2.2	必选
Hive	3.1.3	必选
Tez	0.10.1	必选
Hdfs	3.2.3	必选
Yarn	3.2.3	必选
Mapreduce2	3.2.3	必选
Zookeeper	3.4.6	必选
LDAP	2.4.48	必选
Remote_Shuffle	4.1.0	
RapidFS	1.0.0.1	

BMR Trino 版本信息

组件名称	TRINO 406	备注
hdfs	3.2.3	必选
zookeeper	3.6.3	必选
hive	3.1.3	必选
trino	406	必选
ldap	2.4.48	必选
hudi	0.12.2	

BMR Hadoop存算分离 版本信息

组件名称	BMR 3.5.0	BMR 3.6.0	备注
hadoop-common	3.2.3	3.2.3	必选
bos-hdfs	3.2.3		必选
yarn	3.2.3	3.2.3	必选
mapreduce	3.2.3		必选
zookeeper	3.4.6	3.4.6	必选
ldap	2.4.48	2.4.48	必选
hive	3.1.3	3.1.3	
tez	0.10.2	0.10.2	
spark	3.5.1		
flink	1.18.1	1.20.0	
hudi	0.15.0	0.15.0	
iceberg	1.4.3	1.4.3	
delta	3.2.0		
trino	2.0.1	468	
hue	4.8.0	4.8.0	
airflow	2.7.3	2.7.3	
oozie	5.1.0	5.1.0	
kyuubi	1.7.1	1.10.0	
ranger	2.3.0	2.3.0	
dolphinscheduler	3.2.0	3.2.0	
hdfs		3.2.3	
bmr-fs		3.2.3	
mapreduce2		3.2.3	
spark3		3.5.1	
impala		3.2.0	
deltalake		3.2.0	
paimon		1.0.0	

产品描述

产品简介

概述

MapReduce（简称“BMR”）是托管的一站式大数据平台，提供高可靠、高安全性、高性价比、易运维的分布式计算服务，涵盖 Hadoop、Spark、Hive、Flink、Presto、Druid等多种开源组件，并与百度对象存储无缝衔接，助力企业轻松高效地处理海量数据。

MapReduce支持完整的Hadoop生态：

- Hadoop：提供可靠存储HDFS以及MapReduce编程范式以便大规模并行处理数据。
- Spark：提供基于分布式内存的大规模并行处理框架，从而大大提高大数据分析性能。Spark提供了SQL查询接口、流数据处理以及机器学习。
- HBase：大规模分布式NoSQL数据库，提供随机存取大量的非结构化和半结构化的海量数据。
- ClickHouse：是一个开源的列式存储数据库管理系统，多用于联机分析（OLAP）场景，可提供海量数据的存储和分析，同时利用其数据压缩和向量化引擎的特性，能提供快速的数据搜索。

与自己搭建Hadoop集群相比，MapReduce有以下优势：

- 方便：几分钟便可创建集群，无需为节点分配、部署、优化投入时间。
- 弹性：创建任意大小的集群并动态调整集群规模，高峰期加大集群规模以提高计算能力，低峰期可对应缩减集群规模降低花费。
- 开放：完全兼容开源Hadoop/Spark社区，零成本业务迁移。
- 实惠：支持按需付费以及包年包月，计价简单而透明。
- 安全：专属私有网络，独占系统环境，确保数据安全。

MapReduce组件

- MapReduce：用于大规模数据集的分布式并行计算的编程模型，极大地方便了开发者在不会分布式并行编程的情况下，将自己的程序运行在分布式系统上。
- Spark：开源的集群计算框架。Spark通过拓展内存计算可在海量数据的迭代式计算和交互式计算中提供远快于Hadoop的运算速度。同时，Spark支持SQL请求、流数据处理、机器学习和图表处理，提高开发者效率。
- HBase：开源的、非关系型、分布式的列式数据库，为Hadoop提供NoSQL功能。
- Hive：允许使用类似于SQL语法进行数据查询，适合数据仓库的分析任务。
- Pig：是一种过程语言，可加载数据、表达转换数据以及存储最终结果，使得日志等半结构化数据变得有意义。
- Hue：为了方便管理Hadoop集群以及执行Hive或者Pig脚本而提供的一系列网页应用。
- Sqoop：用于Hadoop与传统的数据库间的数据导入和导出。
- Zeppelin：Web版的notebook，用于数据分析和可视化，可无缝对接Hive、SparkSQL等。
- ZooKeeper：提供分布式一致性锁，用于HDFS、YARN高可用，在HBase、Kafka、Druid中保证数据一致性。
- Ranger：提供基于策略的用户权限管理服务，BMR中的Ranger支持对HDFS、Hive、HBase、Kafka配置用户权限。
- Impala：为数据分析师提供的开源的OLAP数据分析引擎。Impala和Hive使用相同的元数据。
- Presto：为数据分析师提供的开源的OLAP数据分析引擎。Presto和Hive使用相同的元数据。
- Alluxio：是一个面向基于云的数据分析和人工智能的开源的数据编排技术。 它为数据驱动型应用和存储系统构建了桥梁，将数据从存储层移动到距离数据驱动型应用更近的位置从而能够更容易被访问。
- Airflow：是一个分布式的流程调度系统，在配置上可以像编程一样的方式去创作工作流，通过DAG定时和管理各种离线Job的调度平台。

高可用架构

Hadoop-3.0.0以及以上集群版本服务部署情况：

服务	组件	master1	master2	master3
HDFS	Name node	✔		
	SECONDARY_NAMENODE	✔		
	DATANODE	✔	✔	✔
	ZKFC	✔	✔	✔
YARN	ResourceManger	✔	✔	✔
	TimeLineServer			✔
MapReduce2	HistoryServer			✔
HIVE	HiveMetaStore	✔	✔	
	HiveServer2	✔	✔	
HBASE	HMasterServer	✔	✔	
Zbookkeeper	Zbookkeeper	✔	✔	✔
Spark	Spark2-HistoryServer			✔
	Spark2-ThriftServer	✔	✔	

Hadoop-3.0.0以下集群版本服务部署情况：

服务	组件	master1	master2	master3
HDFS	Name node	✔		
	JOURNALNODE	✔		
	DATANODE	✔	✔	✔
	ZKFC	✔	✔	✔
YARN	ResourceManger	✔	✔	✔
	TimeLineServer			✔
MapReduce2	HistoryServer			✔
HIVE	HiveMetaStore	✔	✔	
	HiveServer2	✔	✔	
HBASE	HMasterServer	✔	✔	
Zbookkeeper	Zbookkeeper	✔	✔	✔
Spark	Spark2-HistoryServer			✔
	Spark2-ThriftServer	✔	✔	

ClickHouse

服务	组件	master1	master2	master3
Zookeeper	ZookeeperServer	✓	✓	✓

产品功能

- 便捷易用的集群部署管理： 为用户提供简单快捷的界面化交互方式进行集群部署管理，并提供丰富全面的大数据组件，灵活满足各种使用场景。为用户提供配置多样的高性能节点机器，允许用户按需进行集群的弹性配置，以及根据业务情况进行节点的扩缩容，以最低的成本挖掘最大的数据价值。
- 作业开发： 简单友好的作业开发方式与丰富全面的作业日志及诊断调优建议，并支持多种类型的作业形式，全方位加速用户作业开发环节。允许用户创建临时集群运行定时作业，用完即释放，达到资源成本与业务需求的最优解。
- 监控报警： 提供界面化、简单易用的监控运维工具，支持集群、主机、服务等多种维度的实时监控。提供丰富全面可定义的监控策略，保障业务安全稳定运行。

产品优势

与自建Hadoop集群相比，MapReduce有以下优势：

- 易用：界面点选的操作方式，多种大数据开源组件自由组合，分钟级完成集群创建操作。
- 模板丰富：BMR支持多种集群模板，Hadoop、HBase、Hive、ClickHouse，支持多种应用场景。
- 弹性：创建任意大小的集群并动态调整集群规模，高峰期加大集群规模以提高计算能力，低峰期可对应缩减集群规模降低花费。
- 便捷管理：独创的Hadoop集群管理系统（HMS），提供丰富的集群监控、管理功能。
- 开放：完全兼容开源Hadoop/Spark社区，零成本业务迁移。
- 实惠：支持按需付费以及包年包月以及竞价实例，计价简单而透明。
- 安全：专属私有网络，独占系统环境，确保数据安全。

节点类型说明

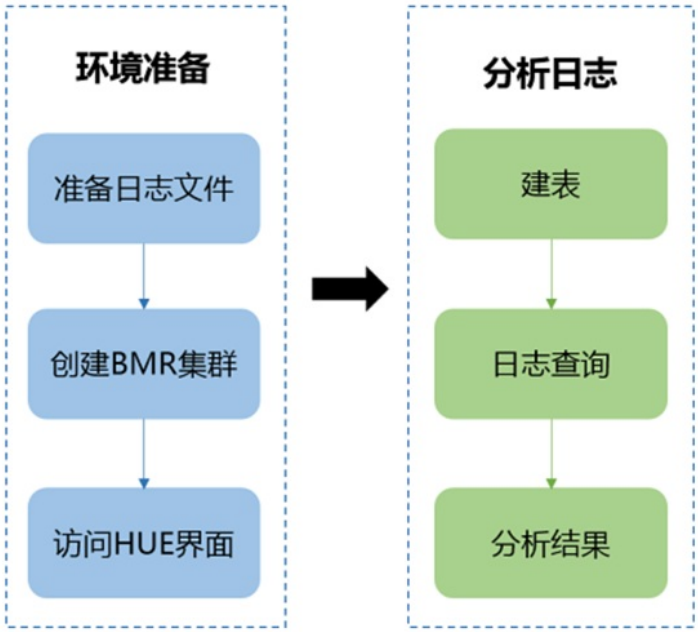
MapReduce集群提供四种类型的节点：

- Master节点：为集群管理节点，部署 NameNode、ResourceManager、HMaster 等进程，在创建集群时可以选择开启高可用模式，支持核心组件服务自动故障迁移。建议选择8核以上的节点配置。
- Core节点：为计算及数据存储节点，部署 DataNode、NodeManager、RegionServer 等进程。并且HDFS 中的数据和计算过程中的中间结果日志数据全部存储于 Core 节点中；采用存算分离架构下的计算中间结果数据也会存放在core节点中；为了数据安全，暂不支持对core节点的缩容操作。
- Task节点：为计算节点，用来补充core节点的算力，部署 NodeManger、PrestoWork 等进程。Task节点一般不用来存储数据，被计算的数据来自 Core 节点及 BOS 中，因此 Task 节点可按需进行扩容和缩容操作。
- Client节点：为独立的集群管控节点及作业提交节点。是与集群在同一VPC内的一组云服务器BCC，为用户提供独立的计算资源以实现集群的负载均衡及安全隔离，提高了集群的稳定及安全性。

应用场景

网站日志处理

网站日志包含着网站最重要的信息，通过日志分析网站，可帮助您获取用户行为以优化网站的商业价值。日益增长的日志信息需大规模处理平台的支撑，MapReduce全托Hadoop服务为高效处理海量网站日志提供了可靠依托，而且开发者在友好的界面中分析海量日志，大大降低了使用门槛。如需快速熟悉具体实现，请至使用Hive分析网站日志。



数据仓库建设

数据仓库，是为企业所有级别的决策制定过程提供所有类型数据支持的战略集合。BMR帮助您快速搭建面向主题的、完整的、稳定的、时变的自有数据仓库，为需要业务智能的企业，提供指导业务流程改进、监视时间、成本、质量以及控制。

产品定价

按需计费

按需计费价格

按需计费定价根据地区收费标准分为内地和香港，具体请参见：按需计费定价。

您可在BMR价格计算器中预设BMR配置并查看对应的价格。

> **说明**
>
> - 通用型是cpu核数与内存比例为1:4的套餐，数据盘挂载的是CDS云磁盘，适用于大多数场景。
> - 计算型是cpu核数与内存比例为1:2的套餐，数据盘挂载的是CDS云磁盘，适用于计算密集型场景。
> - 内存型是cpu核数与内存比例为1:8的套餐，数据盘挂载的是CDS云磁盘，适用于计算结果需要缓存在内存中的业务场景。
> - 本地SSD型的数据盘挂载的是本地SSD磁盘，适合频繁数据读写的场景。但是本地盘数据不保证可靠性，建议选择上述CDS云磁盘套餐，可以选择SSD云磁盘类型，在保证数据可靠性的同时获得较高的数据读写性能。

计费公式

费用=单价×节点个数×使用时长 套餐中涉及的云磁盘CDS及售价请参见云磁盘CDS

计费规则

BMR根据您选择的实例机型配置和个数，按使用时长（分钟级）实时计费。具体计费规则如下：

- 集群创建成功并能够执行提交的作业时，开始计费。您可以在集群创建过程中提交作业，或者在集群创建成功后及时提交作业。
- 集群创建成功后，最少收费时间是5分钟。但是若因为系统原因造成作业运行失败，则作业运行期间不收费。
- 按分钟计费，不足1分钟按1分钟计。
- 按小时扣费，即北京时间整点扣费并生成账单。出账单时间是当前计费周期结束后1小时内。例如，10:00-11:00的账单会在12:00之前生成，具体以系统出账

时间为准。

- 购买前需保证账户无欠款，且保证账户余额和可用代金券总和大于或等于100元。

余额不足提醒和欠费处理

余额不足提醒

- 根据您最近3天的账单金额来判断您的账户余额（含可用代金券）是否足够支付未来3天的费用，若不足以支付，系统发送续费提醒。
- 根据您最近1天的账单金额来判断您的账户余额（含可用代金券）是否足以支付未来1天的费用，若不足以支付，系统发送续费提醒。

欠费处理

- 北京时间整点检查您的账户余额是否足以支付本次BMR账单的费用（如北京时间11点整检查账户余额是否足以支付10点至11点的账单费用），若不足以支付，即为欠费，欠费时系统会发送欠费通知。
- 欠费后立即停服并释放资源，系统会发送欠费停服通知。为了不影响您的服务，建议您在使用BMR时要保证账户中的金额足够支付。

包年包月计费

概述

包年包月计费采用预付费方式，价格较按需计费更低。购买前需保证账户无欠款。

包年包月计费定价

包年包月计费定价根据地区收费标准分为内地和香港，具体请参见：包年包月计费定价。

您可在BMR价格计算器中预设BMR配置并查看对应的价格。

> **说明**
>
> - 通用型是CPU核数与内存比例为1:4的套餐，数据盘挂载的是CDS云磁盘，适用于大多数场景。
> - 计算型是CPU核数与内存比例为1:2的套餐，数据盘挂载的是CDS云磁盘，适用于计算密集型场景。
> - 内存型是CPU核数与内存比例为1:8的套餐，数据盘挂载的是CDS云磁盘，适用于计算结果需要缓存在内存中的业务场景。
> - 本地SSD型的数据盘挂载的是本地SSD磁盘，适合频繁数据读写的场景。但是本地盘数据不保证可靠性，建议选择上述CDS云磁盘套餐，可以选择SSD云磁盘类型，在保证数据可靠性的同时获得较高的数据读写性能。

计费公式

费用=价格×节点个数×使用时长

套餐中涉及的云磁盘CDS及售价请参见云磁盘CDS

到期提醒和处理

到期提醒

BMR服务到期前7天，系统会给您发送即将到期提醒通知。

到期后处理

到期后立即停服，系统会发送欠费停服通知。数据为您保留7天，期间不收取费用，7天内未充值则释放，释放前1天和释放时系统都会发送释放通知。

变更配置计费说明

MapReduce 为您提供节点升配、节点扩容、磁盘扩容、节点缩容等服务，您可在使用过程中根据您的需求，对资源进行弹性升配或降配。

- 预付费服务升配：若您希望对已购的预付费服务进行升级配置操作，预付费服务升配后，额外应付金额 =（新配置价格-原配置价格）×（合同剩余天数/合同总天数）；
- 预付费服务降配：若您希望对已购的预付费服务进行降低配置操作，预付费服务降配后，应退款金额 =（原配置价格-新配置价格）×（合同剩余天数/合同总天数）；
- 后付费服务变配：若您希望对已购的后付费服务进行变更配置操作，后付费服务变配后，立即生效(小时级)并按新的配置费用计费。

账单和用量查询

账单详情查询

1. 登录MapReduce控制台，单击上方财务按钮。

2. 进入财务中心可以查看订单管理和消费中心。

3. 在左侧导航栏，选择消费中心 > 账单明细，可以查看账单的详情，包括服务的详细信息。

退款说明

退订方式

- 预付费退订：系统将会收取您使用时长对应的消费金额，剩余服务时长的费用将退还至原账户，退款金额=订单金额-已消费金额。
- 后付费退订：系统将会收取您使用时长对应的消费金额，退订后，不再产生新的消费金额。

续费说明

续费方式

MapReduce目前支持自动续费和手动续费。使用预付费方式购买的包年包月集群，在集群状态为“运行中”或“已停服”时可续费。

表一 续费说明

续费类型	续费说明
自动续费	在**产品服务>MapReduce>集群列表**页，单击**自动续费**按钮进入续费管理。
手动续费	1.在**产品服务>MapReduce>集群列表**页，单击需续费集群对应的**续费**按钮，进入该集群的续费页。 2.选择续费的时长，单击**下一步**进入订单确认页。 3.确认订单信息和是否使用代金券，无误后点击去支付进入支付页面。根据页面提示选择支付方式完成支付即续费成功。

到期或欠费说明

到期说明

到期提醒

BMR服务到期前7天，系统会发送即将到期提醒通知。

到期后处理

到期后立即停服，系统会发送欠费停服通知。数据为您保留7天，期间不收取费用，7天内未充值则释放，释放前1天和释放时系统都会发送释放通知。

余额不足提醒和欠费处理

余额不足提醒

根据过去三天的账单消费金额来评估，系统会判断您的账户余额（包括可用的代金券）是否足以覆盖接下来三天的费用支出。如果判断结果显示余额不足，系统将自动发送续费通知。

欠费处理

北京时间整点检查您的账户余额是否足以支付本次BMR账单的费用（如北京时间11点整检查账户余额是否足以支付10点至11点的账单费用），若不足以支付，即为欠费，欠费时系统会发送欠费通知。

注意：欠费后立即停服并释放资源，系统会发送欠费停服通知。为了不影响您的服务，建议您在使用BMR时要保证账户中的金额足够支付。

转换计费方式

为了满足用户的业务需要，BMR支持按需计费（后付费）方式转包年包月计费（预付费）方式切换。

操作步骤

在产品服务>MapReduce>集群列表页，单击需续费集群对应的计费变更按钮，进入该集群的计费变更订单页。

- 后付费转预付费：进入计费变更界面，选择预付费购买时长（按月/按年，1-12月/1-3年），选择完毕后根据系统指引确认订单然后订单支付后即可变更成功

计费方式

计费方式

MapReduce支持包年包月和按需付费两种计费方式。

计费方式	描述
包年包月	也是预付费模式，预付费模式指的是在购买集群资源时即需支付全部费用，它更适合有长期需求的用户。相较于按使用量付费，预付费的价格更为优惠，并且用户购买的时长越长，所能享受到的折扣力度也越大。
按需付费	按量付费的集群模式会根据其实际的使用情况即时生成费用账单，这意味着一旦费用产生，就不会进行退款处理。如果您决定不再继续使用该按量付费的集群，您可以选择将其删除，而一旦集群被删除，相关的计费也会立即停止。

计费项

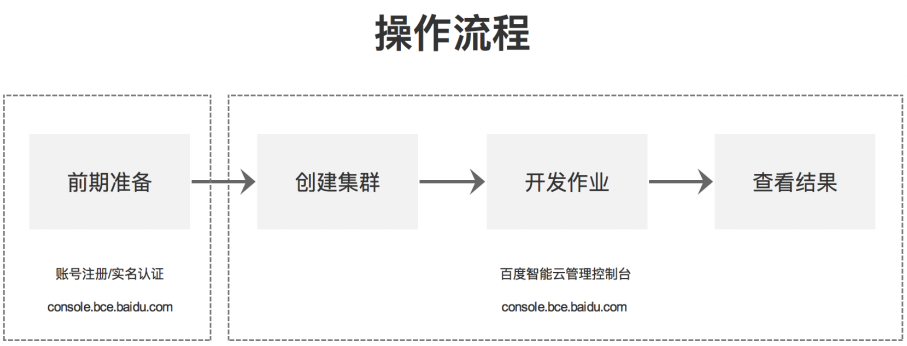
本文介绍MapReduce的计费项及计费规则。详情请查看BMR价格计算器。

计费项

计费项	计费方式	计费规则
存储空间	• 包年包月 • 按需计费	费用=单价×存储量×使用时长。
可选机型	• 包年包月 • 按需计费	费用=单价×节点个数×使用时长。 集群创建成功后，开始计费。您可以在集群创建过程中提交作业，或者在集群创建成功后及时提交作业。

快速入门

操作流程概览



环境准备

1. 注册并登录百度智能云平台，具体操作请参考注册和登录。
2. 选择**产品服务>MapReduce BMR**，进入BMR服务。
3. 使用BMR需要通过实名认证，每个账户只需认证一次。点击“立即去认证”，进入“实名认证”页面。
4. 选择“企业认证”或者“个人认证”。
5. 运用BMR的各组件做数据处理后，结果将输出到对象存储BOS服务器上，因此认证通过后需开通对象存储BOS服务并创建Bucket，并且要求您必须对该地址有写的权限，从而为输出结果提供存储路径。具体操作请参照创建Bucket。

> 注意：
>
> - 本文中所有的数据及示例程序目前只存储在“华北-北京”区域的BOS中，“华南-广州”区域的BOS没有。
> - 对象存储BOS与需要建立的BMR集群应在同一区域内。区域说明请参考区域选择说明。

创建集群

BMR提供两种创建集群的方法：创建自定义集群、使用系统预定义模板创建集群。BMR为用户保留集群的历史记录长达一年，涵盖了运行中、已释放和已终止等各种状态的集群。超过一年的历史集群记录将不再被保留。

创建自定义集群

1. 登录控制台，选择**产品服务-MapReduce BMR**，进入集群列表页。

2. （可选）选择区域，区域说明请参考区域选择说明。在不同区域创建的集群相互独立。

3. 点击**创建集群**，进入创建集群页。

选择集群配置

集群基础设置说明

表一 集群基础设置配置项说明

配置项	配置项说明
集群类型	目前MapReduce支持三种集群类型，不同类型对应不同资源规格和可选服务，请根据实际业务需要选择进行集群创建默认 Hadoop 集群类型
集群版本	BMR发行版是依据开源社区版本、客户需求进行各类服务的统一镜像，各服务的版本由集群版本决定
必选服务	指依据不同集群类型，用户必须安装的服务。例如，hadoop类型集群必选服务为：hdfs、yarn、mapreduce、zookeeper、ldap
可选服务	指依据不同集群类型，用户可自定义选择的服务。例如，hbase类型集群可选服务为：yarn、mapreduce、ranger
安全模式	开启后，集群中的组件以Kerberos安全模式启动，支持统一的 集群安全管理方案
日志	自动收集应用运行日志，支持检索和问题定位。 地址格式：bos://{bucket_name}，bucket需手动创建，folder输入后系统会自动创建

集群标签

标签支持您按各种标准（如用途、所有者或项目）对资源进行分类；每个标签包含键和值两部分。默认一个标签如有需要点击下方**添加标签**进行添加。

基础配置说明

表二 基础配置项说明

配置项名称	配置项说明
集群名称	用于区分不同集群的唯一性。
管理员密码	该密码可以用于登录创建的集群机器，请设置复杂密码，必须以字母、数字、特殊字符组合，长度在8-17个字符之间。
管理员用户名	root。用户远程SSH到集群进行管理。
短信开关	默认开启。 是否启用免费短信提醒服务，启动后系统将发送报警短信或邮件给指定的联系人。

实例组配置

付费方式和地区

1. 选择付费方式：

- 预付费：以自然年或自然月为计费周期支付订单，先付费后使用服务，价格较后付费更低
- 后付费：按分钟实时计费，按小时扣费，先使用服务后付费

2. 自动终止开启/关闭。

开启时，当最后一个作业执行完毕后，自动释放集群，否则集群会一直保持运行，需手动释放。并且可选择超时设置开启/关闭，开启后，可以设置集群的最大运行时间，如果超过，集群会被释放。提醒方式可以选择短信和邮件。

3. 当前区域展示选择后区域，如需修改购买其他地域产品，请前往主导航进行切换。

4. 选择可用区后，填写网络配置和实例组配置。

网络配置

表三 网络配置说明

配置项名称	配置项说明
内网DNS解析	内网DNS为百度云提供的独立的网络产品，可以实现域名（主机名）与ip之间的自动解析。对于不同集群间，BMR集群与BCC间，需要通过域名（主机名）进行互相访问，数据传输时，可以使用该功能。详情请查看智能云解析DNS。 开启内网DNS的BMR集群将自动关联内网DNS产品，自动加入名为novalocal的私有域（若无该私有域，将自动创建）开启内网DNS解析后，可以实现集群间数据通信。开启后BMR集群将自动关联内网DNS，若无DNS，则会自动创建。 需要注意的是： 1.开启该功能，智能云解析DNS（内网DNS）产品，将根据使用量进行收费。当前智能云解析DNS（内网DNS）处于公测免费阶段，详情请查看产品定价。 2.该功能开启后，暂不支持关闭，请合理规划集群使用用途。释放集群时，为保证其他集群（如有）的正常使用，关联的内网DNS私有域不会自动释放，请您在确定没有其他集群使用该功能时，进入智能云解析DNS控制台，将相关资源进行释放，以免带来不必要的计费。 3.当通过子用户创建BMR集群时，开启内网DNS功能，需要您进入多用户访问控制中，为该用户授予“LDFullControlPolicy”权限，否则该功能无法使用。
网络类型	若VPC未进行域名配置，会导致您无法成功创建集群，请前往VPC控制台确认VPC域名已被正确配置。
安全组	安全组具有防火墙功能，用于设置云服务器 CVM 的网络访问控制。 MapReduce自动默认一个安全组配置，该配置影响集群机器的安全组配置，非必要不进行修改。
自动终止	默认关闭。 开启时，当最后一个作业执行完毕后，自动释放集群，否则集群会一直保持运行，需手动释放。

表四 实例组配置说明

配置项名称	说明
高可用	开启后，集群拥有三个master实例，支持核心组件自动故障转移。
计算类型及实例组	MASTER节点：支持通用型、计算型、内存型。购买数量非高可用集群为1个，高可用集群为3个。 Task节点：支持通用型、计算型、内存型、本地SSD型、大数据型。购买数量默认为0个，最大可设置50个。控制台上一个集群仅能添加一组Task节点，请规划好套餐设置 CORE节点：支持通用型、计算型、内存型、本地SSD型、大数据型。购买数量默认为2个，最少设置0个，最大可设置200个。 Client节点：支持通用型、计算型、内存型。购买数量默认为0个，最大可设置10个。
系统盘	系统盘类型支持高性能云磁盘、SSD云磁盘和增强型云磁盘。容量为40-500B，ssd（免费赠送40GB，超出部分需要单独付费）。
数据盘	数据盘类型支持高性能云磁盘、SSD云磁盘和增强型云磁盘。数目默认为1，最大可设置为5。 容量：50-32765GB， ssd（SSD云磁盘 / 高性能云磁盘/增强型云磁盘）。选择说明

4. 如果出现规格售罄情况，建议尝试更换区域和可用区。在付费方式和地区切换区域和可用区，查看其它地域内是否有您的服务器。

确认订单

表五 确认订单说明

显示项	显示项说明
BMR集群信息	显示配置完成的BMR集群配置信息，确认是否无误。
代金券选择	代金券分预付费和后付费： • 预付费订单在这一步选择代金券，如有代金券可选择激活。也可进入代金券管理进行详细操作。 • 后付费订单如果您有代金券，产生账单后会优先从可用代金券中扣除；如果有折扣，会在实际产生的账单中体现。
支付订单	1.确认无误后，点击去支付后跳转订单支付界面。页面会再次展示订单详情，请您再次核对订单无误后选择具体的支付方式进行支付。 2.确认支付完成后，系统会更新订单状态，此时集群创建完毕。

使用系统预定义模板创建集群

1. 在**产品服务->MapReduce-集群模板**页中，在集群模版列表中，在操作列点击**创建集群**。具体模板创建详见集群模版。
2. 进入配置界面，输入之前创建模板时的管理员密码后点击**下一步**。
3. 完成订单确认和支付后可完成创建集群。

附录

表六 组件依赖说明

组件名称	依赖组件
ranger	ranger-plugin
yarn	hdfs、zookeeper
hdfs	zookeeper
rapidfs	hdfs、bos
mapreduce2	hdfs、yarn
tez	hdfs、yarn
spark2	hfds、yarn
spark3	hdfs、yarn
flink	hdfs、yarn
hive	hdfs、yarn、tez
hbase	hdfs、zookeeper
pig	hdfs、yarn
presto	通常和 hive 元数据搭配使用
impala	通常和 kudu 搭配使用
kudu	通常和 impala 搭配使用
kyuubi	spark
hudi	通常和 spark、flink 搭配使用
iceberg	通常和 spark、flink 搭配使用
detalake	spark
oozie	hdfs yarn

数据准备

以准备Web日志数据为例，您可以直接使用百度智能云提供的样例数据，也可根据说明构造自己的输入数据：

- 使用百度智能云提供的样例数据，路径如下：
  - 存储在“华北-北京”区域的样例数据路径为：bos://datamart-bj/web-log-10k/，仅华北区域的BMR集群可用。
  - 存储在“华南-广州”区域的样例数据路径为：bos://datamart-gz/web-log-10k/，仅华南区域的BMR集群可用。
- 根据如下说明构造自己的输入数据，并上传到对象存储BOS（具体操作详见对象存储BOS入门指南）或您本地的HDFS中。

  由Nginx产生的Web访问日志具备如下格式：

  ```
  $remote_addr - [$time_local] "$request" $status $body_bytes_sent "$http_referer"  "$http_cookie" $remote_user "$http_user_agent" $request_time $host $msec
  ```

  例如：

  ```
  10.81.78.220 - [04/Oct/2015:21:31:22 +0800] "GET /u2bmp.html?dm=37no6.com/003&ac=1510042131161237772&v=y88j6-1.0&rnd=1510042131161237772&ext_y88j6_tid=003&ext_y88j6_uid=1510042131161237772 HTTP/1.1" 200 54 "-" "-" 9CA13069CB4D7B836DC0B8F8FD06F8AF "ImgoTV-iphone/4.5.3.150815 CFNetwork/672.1.13 Darwin/14.0.0" 0.004 test.com.org 1443965482.737
  ```

开发作业

注：自2024年6月30日起，MapReduce暂不提供作业相关功能支持，可通过第三方平台EasyDAP或开源组件Airflow提交任务。

使用hadoop镜像的集群可添加的作业类型是：java，streaming。使用spark镜像的集群可添加作业类型：spark，java，streaming。集群中添加了应用后便可添加该应用的作业，即创建集群时添加了hive应用，则可创建hive作业，添加了pig应用，则可创建pig作业。

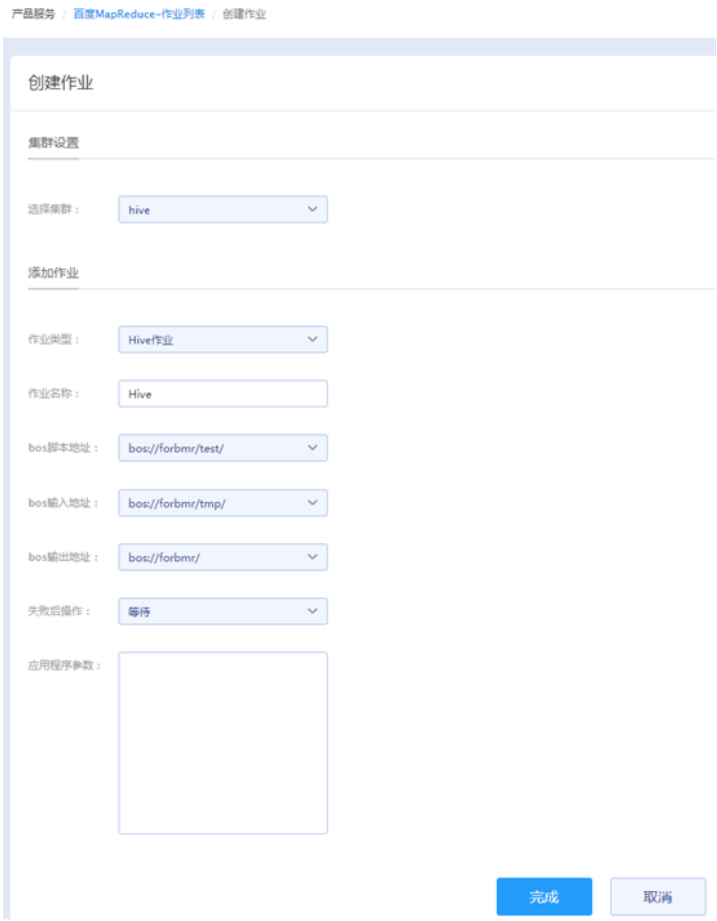
创建作业的操作步骤如下：

1. 在“产品服务>MapReduce>MapReduce-作业列表”页中，点击“创建作业”，进入创建作业页。
2. 请在创建作业页选择作业类型并配置作业类型对应的参数。以下列举了Hive作业类型对应参数配置说明：

   Hive作业：
   - 作业名称：输入作业名称，长度不可超过255个字符。
   - bos脚本地址：BOS的脚本地址必须是一个有效的BOS路径，并且指向Hive脚本。
   - bos输入地址：这个地址必须已经存在，并且您有权限读取这个地址的文件。可在脚本中通过${INPUT}引用这个地址。

- bos输出地址：写入的bucket之后的地址必须是不存在的，但您有权限对这个地址进行写操作，否则作业运行会失败。可在脚本中通过${OUTPUT}引用这个地址。
- 失败后操作：选择作业运行失败后的操作：继续（作业执行失败后，继续执行下一个作业）和等待（作业执行失败后，查看作业运行的状态，并且取消后续作业）。
- 应用程序参数：只接受两种参数类型，分别是--hiveconf key=value 和 --hivevar key=value。前一种参数是用来覆盖hive执行时的配置。后一种参数是用来声明自定义的变量，可以在脚本中通过${KEY}来引用。输入参数时，只需要输入参数本身字符串即可，用空格分隔，无需参数转义和url encode。

3. 选择适配的集群。
4. 点击“完成”，则作业创建完成。

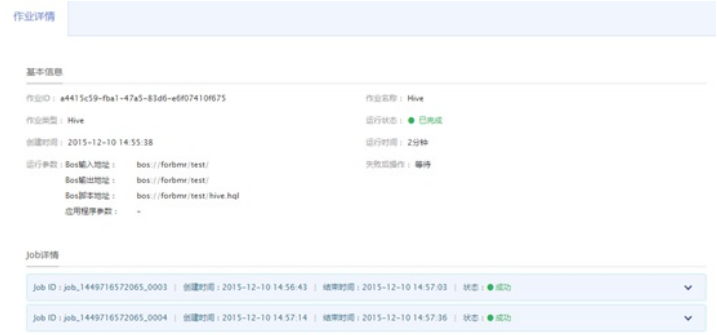


5. 当作业状态会由“等待中”更新为“运行中”状态，作业运行完毕后状态更新为“已完成”。
6. （可选）只有等待中或运行中的作业可被取消，点击“取消作业”即可。

查看结果

注：自2024年6月30日起，MapReduce暂不提供作业相关功能支持，可通过第三方平台EasyDAP或开源组件Airflow提交任务。

1. 在“产品服务>MapReduce>MapReduce-作业列表”中，点击作业名称，可查看作业基本信息。

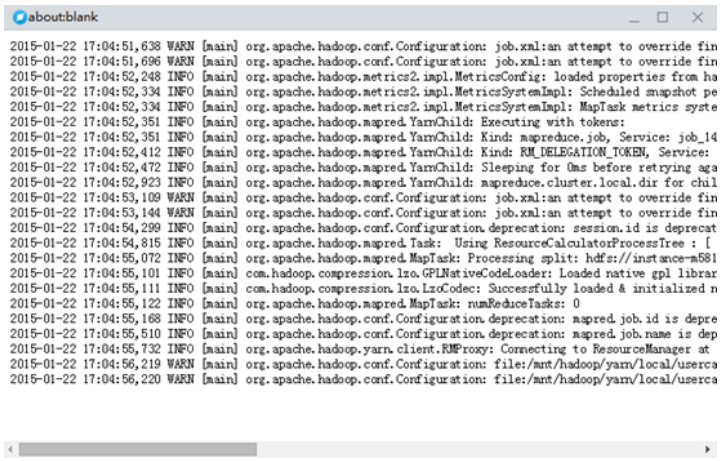


2. Hadoop将job分成若干个task进行处理，共有两种类型的task，分别为map task和reduce task。点击下拉尖括号，查看各task的任务处理情形。

3. 当task运行失败时，每个task有四次的重试机会，每次的重试信息记录在“查看Attempt”中。点击“MAP”和“REDUCE”对应的“查看Attempt”，查看attempt信息。

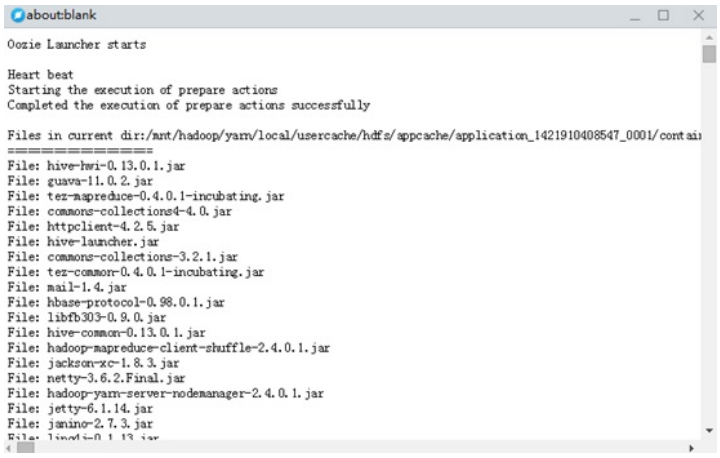
4. 点击“作业日志”，可查看作业日志，其中Spark作业无作业日志。共有三种日志类型显示，分别是syslog、stderr和stdout。

- Syslog是记录作业运行时的详细信息；



- stderr是记录运行作业时的错误信息；

- stdout记录作业运行完毕后的输出信息；



开源组件介绍

Hadoop-Streaming

Hadoop Streaming简介

本文以分析Web日志统计每日请求量为例，介绍如何在百度智能云平台使用Hadoop Streaming。

在BMR集群中，您可以使用python、shell、C++等任何您熟悉的编程语言开发Hadoop Streaming作业。Hadoop Streaming是Hadoop提供的编程工具，允许用户使用任何可执行文件或者脚本文件作为Mapper和Reducer，它将Mapper和Reducer文件封装成MapReduce作业并提交运行，让您无须打包即可快速实现MapReduce功能，满足复杂的业务需求。

程序准备

Mapper.py程序：

```
#!/usr/bin/env python
import sys
import re
separator = "\\s{1}"
ipAddr = "(\\S+)"
remoteUser = "(.*?)"
timeLocal = "\\[(.*?)\\]"
request = "(.*?)"
status = "(\\d{3})"
bodyBytesSent = "(\\S+)"
httpReferer = "(.*?)"
httpCookie = "(.*?)"
httpUserAgent = "(.*?)"
requestTime = "(\\S+)"
host = "(\\S+)"
msec = "(\\S+)"
p = ipAddr + separator + "-" + separator + timeLocal\
    + separator + request + separator + status + separator\
    + bodyBytesSent + separator + httpReferer + separator + httpCookie\
    + separator + remoteUser + separator + httpUserAgent + separator\
    + requestTime + separator + host + separator + msec
for line in sys.stdin:
    line = line.strip()
    m = re.match(p, line)
    if m is not None:
        print '%s\t%s' % (m.group(2).split(":")[0], 1)
```

Reducer.py程序：

```
#!/usr/bin/env python
import sys
from operator import itemgetter
sum_result = {}
for line in sys.stdin:
    line = line.strip()
    key, value = line.split()
    try:
        value = int(value)
        sum_result[key] = sum_result.get(key, 0) + value
    except ValueError:
        pass
sorted_sum_result = sorted(sum_result.items(), key=itemgetter(0))
for key, value in sorted_sum_result:
    print '%s\t%s'% (key, value)
```

集群准备

1. 准备数据，请参考数据准备；
2. 百度智能云环境准备；
3. 登录控制台，选择“产品服务->MapReduce BMR”，点击“创建集群”，进入集群创建页，并做如下配置：
    - 设置集群名称
    - 设置管理员密码
    - 关闭日志开关
    - 选择集群版本“BMR1.6.0”

- 选择集群类型“hadoop”

4. 请保持集群的其他默认配置不变，点击“完成”可在集群列表页查看已创建的集群，当集群状态由“初始化中”变为“空闲中”时，集群创建成功。

运行作业

1. 在“产品服务>MapReduce>MapReduce-作业列表”页中，点击“创建作业”，进入创建作业页。
2. 配置Streaming作业参数：
   - 作业类型：选择“Streaming作业”；
   - 作业名称：输入作业名称，长度不可超过255个字符；
   - Mapper：若使用您自己的代码，请上传程序至BOS或者您本地的HDFS中，并在此输入程序路径；您也可直接使用百度智能云提供的样例程序，路径如下： *华北-北京区域的BMR集群对应的样例程序路径：* `bos://bmr-public-bj/sample/streaming-1.0-mapper.py` 华南-广州区域的BMR集群对应的样例程序路径：`bos://bmr-public-gz/sample/streaming-1.0-mapper.py`
   - Reducer：若使用您自己的代码，请上传程序至BOS或者您本地的HDFS中，并在此输入程序路径；您也可直接使用百度智能云提供的样例程序，路径如下： *华北-北京区域的BMR集群对应的样例程序路径：bos://bmr-public-bj/sample/streaming-1.0-reducer.py* 华南-广州区域的BMR集群对应的样例程序路径：`bos://bmr-public-gz/sample/streaming-1.0-reducer.py`
   - BOS输入地址：`bos://bmr-public-bj/data/log/accesslog-1k.log`
   - BOS输出地址：输出路径必须具有写权限且该路径不能已存在，`bos://{your-bucket}/output`
   - 失败后操作：继续；
   - 应用程序参数：无。
3. 在“集群适配”区，选择适配的集群。
4. 点击“完成”，则作业创建成功；运行中的作业状态会由“等待中”更新为“运行中”，当作业运行完毕后，状态会更新为“已完成”，即可查看到结果了。

查看结果

您可以到您所选的存储系统（BOS或HDFS）中查看输出结果，以下是在BOS中查看输出结果的说明：

用户到bos://{your-bucket}/output路径下查看输出，如果使用系统提供的输入数据和程序，可以打开输出结果看到如下内容：

```
 03/Oct/2015   139
04/Oct/2015   375
05/Oct/2015   372
06/Oct/2015   114
```

分布式缓存导入数据

分布式缓存是Hadoop提供的文件缓存工具，它能够自动将指定的文件分发到运行Map或Reduce任务的各个节点上，并缓存到本地，供用户程序读取使用。在Map或Reduce时，如需访问通用数据，利用分布式缓存可显著提高访问效率。下面以Streaming作业为例说明如何使用Hadoop分布式缓存。

应用场景

分布式缓存通常用在当MapReduce需要依赖一些特定的版本库，或者MapReduce需要访问第三方库或文件时，若将这些库或文件安装到集群各个机器上通常比较麻烦，这时可以使用分布式缓存将其分发到集群各个节点上，程序运行完后，Hadoop自动将其删除。

例如下述的场景，运用分布式缓存会很好的解决问题：

- 分发字典文件：Map或者Reduce需要用到一些外部字典时，可使用分发字典文件，比如黑白名单、词表等。
- 自动化软件部署：MapReduce需依赖于特定版本的库时，可使用自动化软件部署，比如依赖于某个版本的PHP解释器，可以运用分布式缓存进行上传分发。

支持的文件类型

分布式缓存支持单个文件和打包文件，支持以下压缩格式：

```
 * zip
* tgz
* tar.gz
* tar
* jar
```

操作步骤

1. 您需要先将文件上传到BOS，可参考BOS上传Object；
2. 提交作业时通过指定参数，为缓存文件指定一个软链接，在Mapper和Reducer中通过调用软连接即可访问实际的文件内容;

- -files：通常用来上传单个文件，可将指定文件分发到各个Task的工作目录下，不做其他处理；如需上传多个文件，文件之间用逗号隔开；
- -archives：通常用来上传打包文件，可将指定文件分发到各个Task的工作目录下，并对后缀名为“.jar”、“.zip”，“.tar.gz”、“.tgz”的文件自动解压，默认情况下，解压后的内容存放到工作目录下名称为解压前文件名的目录中；如需上传多个打包文件，文件之间用逗号隔开。

文件类型	操作	说明
添加单个文件	输入参数：-files+文件在BOS中的位置+井号（#）+文件在缓存中的软链接名称	-files bos://bucket_name/file_name#file_name
添加打包文件	输入参数：-archives+文件在BOS中的位置+井号（#）+文件在缓存中的软链接名称	-archives bos://bucket_name/archive_name#archive_name

例如：

- 对于某一个文件mydata.txt，将其上传到BOS文件系统，路径为bos://mybucket/mydata.txt，提交MapReduce Streaming作业时在参数中指定 -files bos://mybucket/mydata.txt#data，在MapTask和ReduceTask中就可以通过 ./data 来访问所需的文件了；
- 对于某一个文件libA.so，将其打包到mylib.tar.gz文件中，并上传到BOS文件系统，路径为bos://mybucket/mylib.tar.gz，提交MapReduce Streaming作业时在参数中指定 -archives bos://mybucket/mylib.tar.gz#libs，在MapTask和ReduceTask中就可以通过./libs/libA.so来访问所需的文件了。

HBase

HBase简介

本文以分析Web日志统计每天的PV和UV为例，介绍如何在百度智能云平台使用HBase。

HBase是运行在Hadoop上的NoSQL数据库，它是一个分布式的和可扩展的大数据仓库，能够利用HDFS的分布式处理模式和Hadoop的MapReduce程序模型。HBase融合key/value存储模式带来实时查询的能力，以及通过MapReduce进行离线处理或者批处理的能力。总的来说，HBase能够让您在大量的数据中查询记录，也可获得综合分析报告。

HBase不是一个关系型数据库，需要不同的方法定义数据模型，HBase定义了一个四维数据模型：行键、列簇、列修饰符以及版本，以便获取指定数据：

- 行键：每行都有唯一的行键，行键没有数据类型，它内部被认为是一个字节数组。
- 列簇：数据在行中被组织成列簇，每行有相同的列簇，但是在行之间，相同的列簇不需要有相同的列修饰符。在引擎中，HBase将列簇存储在自身的数据文件中，因此，列簇需要事先被定义。
- 列修饰符：列簇定义真实的列，被称之为列修饰符，可认为列修饰符就是列本身。
- 版本：每列都可以有一个可配置的版本数量，可通过列修饰符的制定版本获取数据。

程序准备

您可以直接使用样例程序。也可设计自己的程序，并上传到对象存储BOS（具体操作详见对象存储BOS入门指南）。

集群准备

1. 准备数据，请参考数据准备。
2. 准备百度智能云环境。
3. 登录控制台，选择“产品服务->MapReduce BMR”，点击“创建集群”，进入集群创建页，并做如下配置：
    - 设置集群名称
    - 设置管理员密码
    - 打开日志开关
    - 选择集群类型HBASE”
    - 选择集群版本 BMR Hbase 2.2.0”。
4. 请保持集群的其他默认配置不变，点击“完成”可在集群列表页可查看已创建的集群，当集群状态由“初始化中”变为“运行中”时，集群创建成功。

> 说明：Hbase集群支持在创建集群时将HBase存储设置设置为BOS，可以获得更高的数据可靠性。`bos:<bucket-name>/<folder>`， bucket必须存在，如不存在，请新建;folder可不存在，如不存在，系统会自动创建;请确定选择的hbase目录中的任意层级，没有hbase元数据。集群创建后续无法修改该路径及切换为HDFS存储。请合理选择您的存储方式。具体设置方式，如下图所示。

运行Java作业

提取Web访问日志内容到HBase Table

1. 在“产品服务>MapReduce>MapReduce-作业列表”页中，点击“创建作业”，进入创建作业页。
2. 配置Java作业参数，具体如下：
   - 作业类型：选择“Java作业”。
   - 作业名称：输入作业名称，长度不可超过255个字符。
   - 应用程序位置：可输入样例程序路径`bos://bmr-public-data/apps/hbase/bmr-hbase-samples-1.0-SNAPSHOT.jar`。
   - 失败后操作：继续。
   - MainClass:输入`com.baidubce.bmr.hbase.samples.logextract.AccessLogExtract`。
   - 应用程序参数：输入`-D mapreduce.job.maps=6 -D mapreduce.job.reduces=2 bos://bmr-public-data/logs/accesslog-1k.log AccessTable`。（最后一个参数"AccessTable"，是HBase Table的名称。）
3. 在“集群适配”区，选择适配的集群。
4. 点击“完成”，则作业创建完成。运行中的作业状态会由“等待中”更新为“运行中”，当作业运行完毕后状态会更新为“已完成”。

统计每天的PV

1. 在“产品服务>MapReduce>MapReduce-作业列表”页中，点击“创建作业”，进入创建作业页。
2. 配置Java作业参数，具体如下：
   - 作业类型：选择“Java作业”。
   - 作业名称：输入作业名称，长度不可超过255个字符。
   - 应用程序位置：可输入样例程序路径`bos://bmr-public-data/apps/hbase/bmr-hbase-samples-1.0-SNAPSHOT.jar`。
   - 失败后操作：继续。
   - MainClass:输入`com.baidubce.bmr.hbase.samples.pv.PageView`。
   - 应用程序参数：输入`-D mapreduce.job.maps=6 -D mapreduce.job.reduces=2 AccessTable bos://${USER_BUCKET}/pv`。bos://${USER_BUCKET}/pv必须具有写权限且该路径中所指定的目录不能在bos上存在，例如，输出路径为bos://test/sqooptest，则sqooptest目录在bos上必须不存在。
3. 在“集群适配”区，选择适配的集群。
4. 点击“完成”，则作业创建完成。当作业状态会由“等待中”更新为“运行中”状态，作业运行完毕后状态更新为“已完成”。

统计每天的UV

1. 在“产品服务>MapReduce>MapReduce-作业列表”页中，点击“创建作业”，进入创建作业页。
2. 请在创建作业页选择已创建的集群、选择“Java作业”并配置对应的参数。
   - 作业名称：输入作业名称，长度不可超过255个字符。

- 应用程序位置：可输入样例程序路径bos://bmr-public-data/apps/hbase/bmr-hbase-samples-1.0-SNAPSHOT.jar。
- 失败后操作：继续。
- MainClass:输入com.baidubce.bmr.hbase.samples.uv.UniqueVisitor。
- 应用程序参数：输入-D mapreduce.job.maps=6 -D mapreduce.job.reduces=2 AccessTable bos://${USER_BUCKET}/uv。bos://${USER_BUCKET}/uv必须具有写权限且该路径中所指定的目录不能在bos上存在，例如，输出路径为bos://test/sqooptest，则sqooptest目录在bos上必须不存在。

3. 配置好作业参数后，点击“完成”，则作业创建完成。运行中的作业状态会由“等待中”更新为“运行中”，当作业运行完毕后状态会更新为“已完成”。

查看结果

bos://${USER_BUCKET}/pv/下的最终reduce结果示例：

```
 03/Oct/2015 139
05/Oct/2015 372
04/Oct/2015 375
06/Oct/2015 114
```

bos://${USER_BUCKET}/uv/下的最终reduce结果示例：

```
 03/Oct/2015 111
05/Oct/2015 212
04/Oct/2015 247
06/Oct/2015 97
```

BMR HBase读写分离集群构建

1、背景

当前hbase on bos 将存储与计算解耦，数据存储在bos 上体现了了众多运营优势。

- 优势1、HBase 根目录存储在 bos 中（HBase hfile存储文件和元信息）此数据在集群外部持续存在且可跨可用区访问。
- 优势2、较之前使用hdfs 3副本的冗余空间占用，大大节省了的存储空间。
- 优势3、避免集群资源的浪费，您可以针对计算要求而非数据规模要求调整 bmr 集群的大小，避免存储空间大但是计算需求小的情况存在资源浪费的情况。

如果您的主集群在批量加载、写入和压缩期间处于高负载状态，但是您同时又有一些历史数据分析查询的业务急需要处理。怎么办？ 传统的方法你可能需要集群的扩容并配置group来保证集群业务之间的隔离。这样您可能为了满足自己一些读取的需求预留出一定的资源从而增加集群资源的成本，同时增加了运维的成本。如果您有这些烦恼，那么您可以replica cluster 功能创建辅助集群，以实现负载分担，将读取负载与写入负载分离开来，从而确保您满足读取服务要求，同时围绕成本和性能进行优化。

2、优点

使用read replica cluster 模式有如下几点的优点：

1、读写分离。写负载和读负载分布在不同的集群中，避免了业务在同一个集群的情况下相互的性能影响。提升读写性能，减少运维成本。

2、减少集群资源的使用成本，primary 集群可以只关注当前需要写的table，对一些不需要写入的table 可以直接disable掉，减少集群的压力。replica cluster可以只针对当前需要读取的表进行分析，避免无关的table占用过多的资源，减少使用成本。

3、即读即建，读完可删。如果不需要继续使用的话可以直接释放掉，减少使用成本。

当前功能replica cluster 与 primary cluster 的数据存在一定时间的gap（小时级），所以replica cluster 推荐读取更改不频繁的历史数据表。

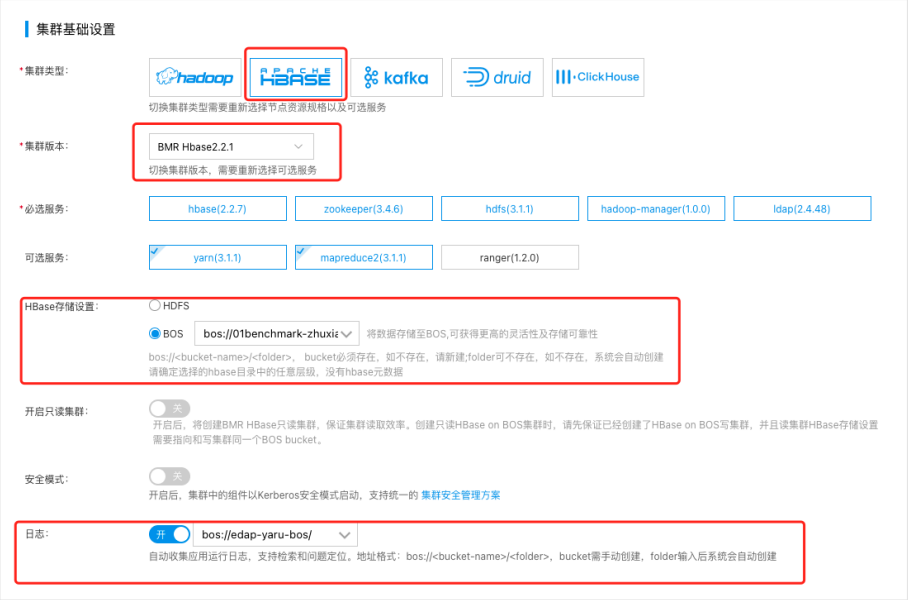
创建集群

读写分离实现需要分别创建写集群和读集群，从而实现读写集群分离的构建。

创建写集群

1. 登录控制台，选择“产品服务->MapReduce BMR”，点击“创建集群”，进入集群创建页，并做如下配置：
    - 设置集群名称
    - 设置管理员密码
    - 打开日志开关
    - 选择集群类型HBASE”
    - 选择集群版本 BMR Hbase 2.2.1”。
2. 请保持集群的其他默认配置不变，点击“完成”可在集群列表页可查看已创建的集群，当集群状态由“初始化中”变为“运行中”时，集群创建成功。

> 说明：Hbase集群支持在创建集群时将HBase存储设置设置为BOS，可以获得更高的数据可靠性。bos:<bucket-name>/<folder>， bucket必须存在，如不存在，请新建;folder可不存在，如不存在，系统会自动创建;请确定选择的hbase目录中的任意层级，没有hbase元数据。集群创建后续无法修改该路径及切换为HDFS存储。请合理选择您的存储方式。具体设置方式，如下图所示。

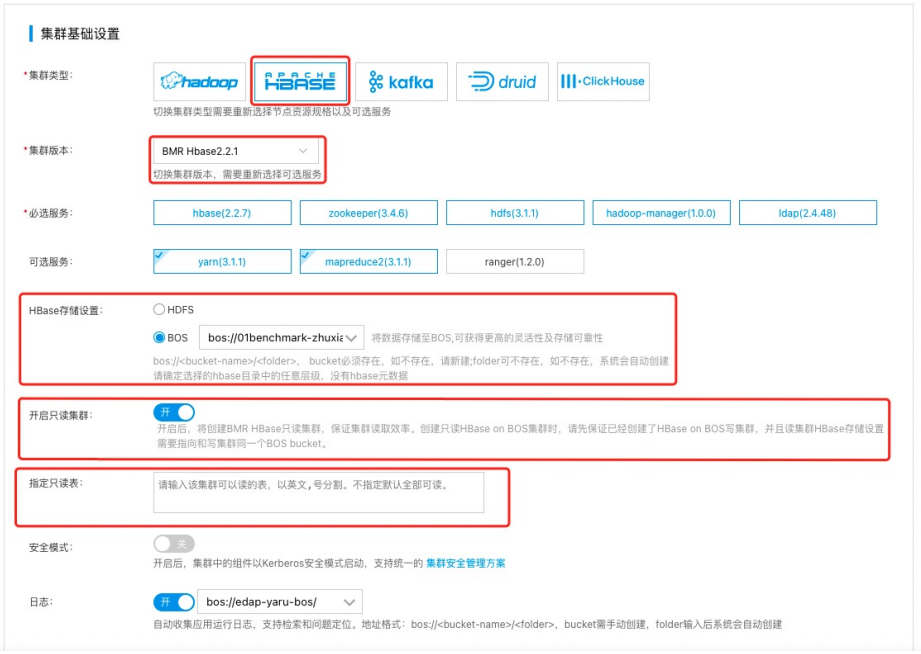


创建只读集群

1. 登录控制台，选择“产品服务->MapReduce BMR”，点击“创建集群”，进入集群创建页，并做如下配置：
    - 设置集群名称
    - 设置管理员密码
    - 打开日志开关
    - 选择集群类型HBASE”
    - 选择集群版本 BMR Hbase 2.2.1”。
    - 将HBase存储设置指向写集群统一BOS bucket
    - 打开只读集群开关
    - 指定要读取的表的名称，不填写默认可以读取写集群中所有的表。

> 开启后，将创建BMR HBase只读集群，保证集群读取效率。创建只读HBase on BOS集群时，请先保证已经创建了HBase on BOS写集群， 并且读集群HBase存储设置需要指向和写集群同一个BOS bucket。

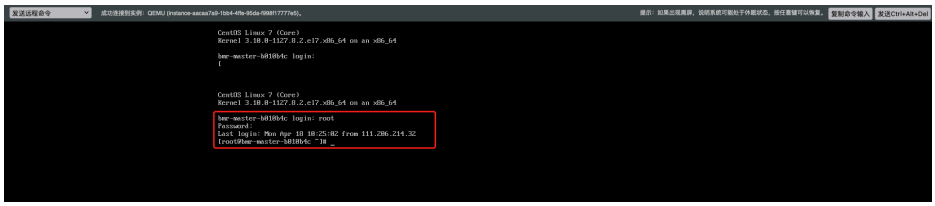
2. 请保持集群的其他默认配置不变，点击“完成”可在集群列表页可查看已创建的集群，当集群状态由“初始化中”变为“运行中”时，集群创建成功。



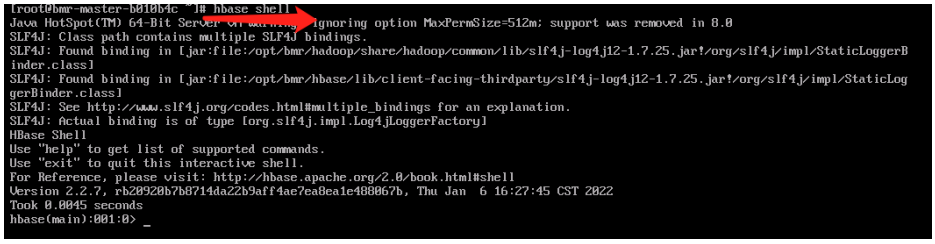
> 只读集群可以get和scan指定的表，但是禁止namespace、table、写入操作。

在写集群中创建HBASE表并插入数据

1、登录到写集群的master节点上，可以通过内网IP或者绑定EIP的方式访问集群主机或者点击主机名称，直接跳转到BCC console用VNC控制台登录。本实例采用VNC控制台登录。输入创建集群时的密码即可，用户名默认为：root。



2、输入命令：`hbase shell`，进入hbase命令行模式。



3、创建表，执行命令：`create 'Student','StuInfo','Grades'`

```
hbase(main):001:0> create 'Student','StuInfo','Grades'
Created table Student
Took 8.8896 seconds
=> Hbase::Table - Student
```

4、插入数据

```
put 'Student', '0001', 'StuInfo:Name', 'test'
put 'Student', '0001', 'Grades:Name', '1'
put 'Student', '0001', 'StuInfo:Sex', 'man'
```

```
hbase(main):005:0> put 'Student', '0001', 'StuInfo:Name', 'test'
Took 0.2462 seconds
hbase(main):006:0> put 'Student', '0001', 'Grades:Name', '1'
Took 0.0121 seconds
hbase(main):007:0> put 'Student', '0001', 'StuInfo:Sex', 'man'
Took 0.0203 seconds
hbase(main):008:0>
```

5、查看插入的数据。 `scan 'Student'`

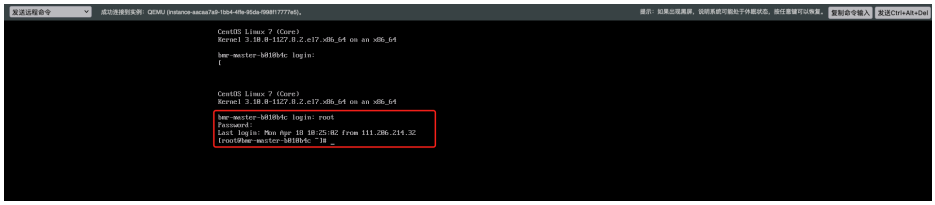
```
hbase(main):008:0> scan 'Student'
ROW                             COLUMN+CELL
 0001                           column=Grades:Name, timestamp=1650249793121, value=1
 0001                           column=StuInfo:Name, timestamp=1650249772009, value=test
 0001                           column=StuInfo:Sex, timestamp=1650249805556, value=man
1 row(s)
Took 0.0663 seconds
hbase(main):009:0>
```

在只读集群中查询数据

> 只读集群读取写集群中的历史数据时可以及时读取，但是读取写集群新创建的表或者插入的新数据有大约4小时左右的延迟，如需读取新创建的表或者是新插入表中的数据，请在写集群中刷新元数据或对应表，参考命令如下：flush 'meta' 或者 flush '表名'。

```
[root@bmr-master-b010b4c ~]# hbase shell
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=512m; support was removed in 8.0
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/bmr/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerB
inder.class]
SLF4J: Found binding in [jar:file:/opt/bmr/hbase/lib/client-facing-thirdparty/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLog
gerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.2.7, rb20920b7b0714da22b9aff4ae7ea8ea1e480867b, Thu Jan  6 16:27:45 CST 2022
Took 0.0044 seconds
hbase(main):001:0> flush 'Student'
Took 0.6182 seconds
hbase(main):002:0>
```

1、登录到只读集群的master节点上，可以通过内网IP或者绑定EIP的方式访问集群主机或者点击主机名称，直接跳转到BCC console用VNC控制台登录。本实例采用VNC控制台登录。输入创建集群时的密码即可，用户名默认为：root。



2、输入命令：`hbase shell`，进入hbase命令行模式。

```
[root@bmr-master-b010b4c ~]# hbase shell
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=512m; support was removed in 8.0
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/bmr/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerB
inder.class]
SLF4J: Found binding in [jar:file:/opt/bmr/hbase/lib/client-facing-thirdparty/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLog
gerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.2.7, rb20920b7b0714da22b9aff4ae7ea8ea1e480867b, Thu Jan  6 16:27:45 CST 2022
Took 0.0045 seconds
hbase(main):001:0> _
```

3、查询数据。`scan 'Student'`

```
hbase(main):008:0> scan 'Student'
ROW                             COLUMN+CELL
 0001                           column=Grades:Name, timestamp=1650249793121, value=1
 0001                           column=StuInfo:Name, timestamp=1650249772009, value=test
 0001                           column=StuInfo:Sex, timestamp=1650249805556, value=man
1 row(s)
Took 0.0192 seconds
hbase(main):009:0>
```

Sqoop

Sqoop简介

本样例场景是：通过Sqoop将RDS上的数据导入Hive，Hive中的数据表的location为BOS路径，Hive数据表的partition为dt（string），根据dt指定日期，区分每一天的导入数据。

Sqoop是用来将Hadoop和关系型数据库中的数据相互转移的工具，可通过Hadoop的MapReduce将关系型数据库（MySQL、Oracle、Postgres等）中的数据导入到HDFS中，也可以将HDFS的数据导进到关系型数据库中。实现过程如下：

1. 读取要导入数据的表结构，生成运行类，默认是QueryResult，打成jar包，然后提交给Hadoop。
2. Hadoop根据作业要求通过MapReduce来执行Import命令：
    1. 切分数据，即DataSplit。
    2. 写入范围，以便读取。
    3. 读取写入的范围。
    4. 创建RecordReader从数据库中读取数据。

5. 创建Map。
6. RecordReader一行一行从关系型数据库中读取数据，设置好Map的Key和Value，交给Map。
7. 运行map。最后生成的Key是行数据。

Sqoop导入数据

您可通过sqoop把关系型数据库RDS中的数据导入到BOS、HDFS、HBase或Hive中。具体操作如下：

从RDS关系型数据库导入数据至BOS中

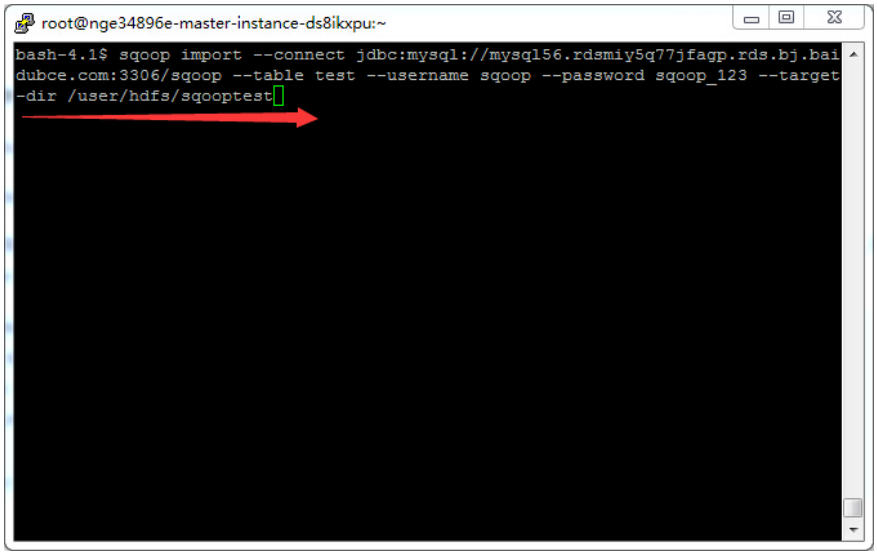
1. 通过SSH连接到主节点，请参考SSH连接到集群。
2. 输入命令：su hdfs。切换到HDFS用户。
3. 执行以下格式的命令：`sqoop import --connect jdbc:mysql://address:port/db_name --table table_name --username XX --password XX --target-dir XX`

   示例：sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --target-dir bos://abc/sqooptest
4. 执行后，可至bos中查看执行结果。BOS上查看执行结果的示例如下：

从RDS关系型数据库导入数据至HDFS中

1. 执行**从RDS关系型数据库导入数据至BOS中**的步骤1至2。
2. 执行以下格式的命令：`sqoop import --connect jdbc:mysql://address:port/db_name --table table_name --username XX --password XX --target-dir XX`

   示例：sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --target-dir /user/hdfs/sqooptest
3. 执行后，可至HDFS中查看执行结果。



参数	参数说明
address和port	RDS数据库实例的地址和端口号。请至RDS实例的基本信息中获取，可参考连接RDS实例。
db_name	需导入数据所在数据库的名称。如需创建关系型数据库RDS实例，请参考创建数据库。
table_name	需导入数据的数据表的名称。如需创建数据表，请先登录到关系型数据库RDS实例中创建，请参考连接RDS实例。
--username和--password	需导入数据所在数据库的账号和密码。请至RDS实例中获取信息，请参考创建账号。
--target-dir	数据导入的目的地址，即BOS或HDFS的路径。 注意：若指定BOS路径，则路径中所指定的目录不能在bos上存在，例如，导入路径bos://test/sqooptest中的sqooptest目录在bos上必须不存在。

从RDS关系型数据库导入数据至HBase中

1. 通过SSH连接到主节点，请参考SSH连接到集群。
2. 输入命令：su hdfs。切换到HDFS用户。
3. Sqoop导入数据到HBase有两种情况:
   - 导入数据到HBase已经存在的表中，执行以下格式的命令：sqoop import --connect jdbc:mysql://address:port/db_name --table table_name --username XX --password XX --hbase-table hbase_table_name --column-family hbase_column_family --hbase-row-key row_key。示例：sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --hbase-table sqoop_hbase -- column-family message1 --hbase-row-key id。
   - 导入数据到HBase中不存在的表中，执行以下格式的命令：sqoop import --connect jdbc:mysql://address:port/db_name --table table_name --username XX --password XX --hbase-table hbase_table_name --column-family hbase_column_family --hbase-row-key row_key --hbase-create-table。示例：sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --hbase-table sqoop_hbase -- column-family message1 --hbase-row-key id --hbase-create-table。
4. 执行后，可至HBase中查看执行结果，执行结果示例如下:

```
 hbase(main):003:0>scan "sqoop_hbase"
ROW  COLUMN+CELL
 1  column=message1:address,timestamp=1439794756361,value=shanghai
 1  column=message1:favor,timestamp=1439794756361,value=lanqiu
 1  column=message1:name,timestamp=1439794756361,value=xiaoming
 1  column=message1:sex,timestamp=1439794756361,value=1
 1  column=message1:address,timestamp=1439794756361,value=beijing
 1  column=message1:favor,timestamp=1439794756361,value=pingpong
 1  column=message1:name,timestamp=1439794756361,value=xiaohong
 1  column=message1:sex,timestamp=1439794756361,value=2
```

参数	参数说明
address和port	RDS数据库实例的地址和端口号。请至RDS实例的基本信息中获取，可参考连接RDS实例。
db_name	数据源所在数据库的名称。如需创建关系型数据库RDS实例，请参考创建数据库。
table_name	数据源在所数据表的名称。如需创建数据表，请先登录到关系型数据库RDS实例中创建，请参考连接RDS实例。
--username和--password	数据源所在数据库的账号和密码。请至RDS实例中获取信息，请参考创建账号。
--hbase-table	数据导入的目的数据表的表名，即HBase中数据表的表名。
--column-family	目的数据表的列簇，即HBase中数据表的列簇。 注意：sqoop一次只能指定一个列簇。
--hbase-row-key	数据源所在RDS数据表中作为hbase row key的字段。

从RDS关系型数据库导入数据至Hive中

1. 通过SSH连接到主节点，请参考SSH连接到集群。
2. 输入命令：su hdfs。切换到HDFS用户。
3. Sqoop导入数据到Hive有三种情况:
   - Hive中不存在与关系型数据库RDS同名的数据表，执行以下格式的命令：sqoop import --connect jdbc:mysql://address:port/db_name --table table_name --username XX --password XX --hive-import。示例：示例：sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --hive-import。
   - 默认情况下，sqoop会将hive中的表名设定为导入数据的数据表名，如不想使用默认表名，使用--hive-table指定新数据表，执行以下格式的命令：sqoop import --connect jdbc:mysql://address:port/db_name --table table_name --username XX --password XX --hive-import --hive-table XX。示例：sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --hive-import --hive-table sqoop_hive。

- Hive中存在与关系型数据库RDS同名的数据表，需要通过--hive-table指定数据表，并且加上--hive-overwrite参数，执行以下格式的命令：sqoop import --connect jdbc:mysql://address:port/db_name --table table_name --username XX --password XX --hive-import --hive-table XX --hive-overwrite。使用RDS数据库中的另一张表test_export，指定hive中已经存在的表sqoop_hive。示例：sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --hive-import --hive-table sqoop_hive -- hive-overwirte。

4. 执行后，可至Hive中查看执行结果，执行结果示例如下：

```
 hive>select * from test;
OK
1 xiaoming 1 shanghai lanqiu
2 xiaohong 2 beijing  pingpong
```

参数	参数说明
address和port	RDS数据库实例的地址和端口号。请至RDS实例的基本信息中获取，可参考连接RDS实例。
db_name	数据源所在数据库的名称。如需创建关系型数据库RDS实例，请参考创建数据库。
table_name	数据源在所数据表的名称。如需创建数据表，请先登录到关系型数据库RDS实例中创建，请参考连接RDS实例。
--username和--password	数据源所在数据库的账号和密码。请至RDS实例中获取信息，请参考创建账号。
--hive-import	导入数据至hive中。
--hive-table	数据导入的目的数据表的表名，即Hive中数据表的表名。
--hive-overwrite	覆盖Hive中与关系型数据库RDS同名的数据表。注意：如果将非INT类型转换为INT类型，导入数据可能不正确。

Sqoop导出数据

您可通过Sqoop把BOS或HDFS的数据导出至关系型数据库RDS中。具体操作如下：

从BOS中导出数据至RDS关系型数据库

1.在关系型数据库RDS中创建相应的数据表，请注意数据表字段类型与导出数据需要一致，否则在导出过程中可能出现异常。数据需要连接到RDS数据库进行创建，请参考连接RDS实例。

2. 通过SSH连接到主节点，请参考SSH连接到集群。

3. 输入命令：su hdfs。切换到HDFS用户。

4. 执行命令：`sqoop export --connect jdbc:mysql://address:port/db_name --table export_table_name --username XX --password XX --export-dir XX`

示例：`sqoop export --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --export-dir bos://abc/sqooptest`

5.执行后，在RDS关系数据库PHP Admin界面可以看到执行结果，如下图所示：

从HDFS中导出数据至RDS关系型数据库

1.执行**从BOS中导出数据至RDS关系型数据库**的步骤1至3。 2.执行命令：sqoop export --connect jdbc:mysql://address:port/db_name --table export_table_name --username XX --password XX --export-dir XX

示例：sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --export-dir /user/hdfs/sqooptest

3.执行**从BOS中导出数据至RDS关系型数据库**的步骤5。

参数	参数说明
address和port	RDS数据库实例的地址和端口号。请至RDS实例的基本信息中获取，可参考连接RDS实例。
db_name	需导出数据所在数据库的名称。如需创建关系型数据库RDS实例，请参考创建数据库。
table_name	需导出数据的数据表的名称。如需创建数据表，请先登录到关系型数据库RDS实例中创建，请参考连接RDS实例。
--username和--password	需导出数据所在数据库的账号和密码。请至RDS实例中获取信息，请参考创建账号。
--export-dir	数据导出的目的地址，即BOS或HDFS的路径。

Pig

Pig简介

本文以分析Web日志统计每天的PV和UV为例，介绍如何在百度智能云平台使用Pig。

Pig是基于Hadoop的大规模数据分析平台，把类SQL的数据分析请求转换为一系列经过优化处理的MapReduce运算。Pig适用于大量的并行进程，因此可处理大规模数据集，而且Pig为复杂的海量数据并行计算提供了一个简单的操作和编程接口，便于写入和维护，可为实现不同的目的创建自己的进程。

Pig的数据类型，不仅支持包、元组和映射等高级概念，还支持简单的数据类型，如int、long、float、double、chararray和bytearray。

程序准备

您可以直接使用样例程序。

```
 lines = LOAD '${INPUT}';
fields = FOREACH lines GENERATE FLATTEN(REGEX_EXTRACT_ALL($0, '(\\S+)\\s+-\\s+\\[(.*?)\\]\\s+\\"(.*)\\"\\s+(\\d{3})\\s+(.*)\\s+\\"
(.*)\\"\\s+\\"(.*)\\"\\s+(.*)\\s+\\"(.*)\\"\\s+(\\S+)\\s+(\\S+)\\s+(\\S+)'));
access_logs = FOREACH fields GENERATE $0 AS remote_ip, ToString(ToDate($1, 'dd/MMM/yyyy:HH:mm:ss Z'), 'dd/MMM/yyyy') AS date;
groups = GROUP access_logs BY date;
pv = FOREACH groups GENERATE group, COUNT(access_logs);
STORE pv INTO '${OUTPUT}';
```

创建集群

1. 准备数据，请参考数据准备。
2. 准备百度智能云环境。
3. 登录控制台，选择“产品服务->MapReduce BMR”，点击“创建集群”，进入集群创建页，并做如下配置：
   - 设置集群名称
   - 设置管理员密码
   - 关闭日志开关
   - 选择镜像版本“BMR 1.0.0(hadoop 2.7)”
   - 选择内置模板“hadoop”。
4. 请保持集群的其他默认配置不变，点击“完成”可在集群列表页可查看已创建的集群，当集群状态由“初始化中”变为“空闲中”时，集群创建成功。

运行Pig作业

1. 在“产品服务>MapReduce>MapReduce-作业列表”页中，点击“创建作业”，进入创建作业页。
2. 配置Pig作业参数，具体如下：
   - 作业类型：选择“Pig作业”。
   - 作业名称：输入作业名称，长度不可超过255个字符。
   - bos脚本地址：可输入样例程序路径`bos://bmr-public-data/apps/pig/AccessLogAnalyzer.pig`。
   - bos输入地址：可输入样例数据路径`bos://bmr-public-data/logs/accesslog-1k.log`
   - bos输出地址：输入`bos://{yourbucket}/output`，该路径必须具有写权限且路径中所指定的目录不能在bos上存在，例如，输出路径为bos://test/sqooptest，则sqooptest目录在bos上必须不存在。
   - 失败后操作：继续。
   - 应用程序参数：无。
3. 在“集群适配”区，选择适配的集群。
4. 点击“完成”，则作业创建完成。运行中的作业状态会由“等待中”更新为“运行中”，当作业运行完毕后状态会更新为“已完成”。

查看结果

可以去bos://{yourbukcet}/output路径下查看输出。如果使用系统提供的输入数据和程序，可以看到如下结果:

```
 03/Oct/2015   139
04/Oct/2015   375
05/Oct/2015   372
06/Oct/2015   114
```

Hue

Hue简介

本文以网站日志分析来介绍可Web访问的Hue服务。开发者可以在Web界面中通过SQL语句就能分析海量日志，大大降低了使用门槛。

Hue为Hadoop数据分析提供了图形界面系统，仅使用浏览器便能够在Hadoop平台上导入数据、处理数据以及分析数据。

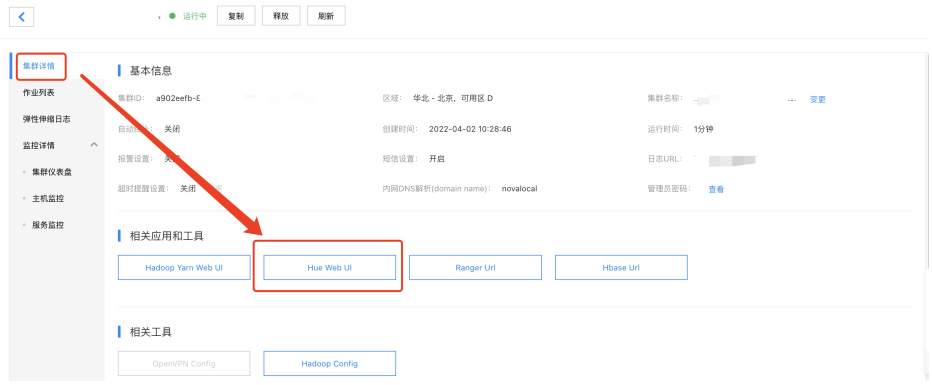
Hue 3.10.0使用

创建集群

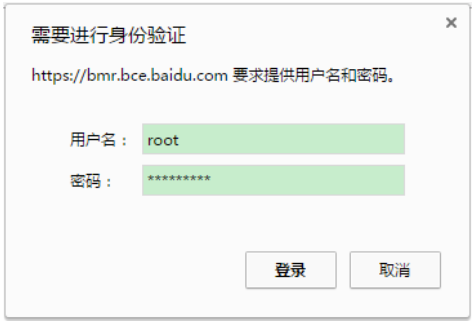
1. 准备数据，请参考数据准备。
2. 准备百度智能云环境。
3. 登录控制台，选择“产品服务->MapReduce BMR”，点击“创建集群”，进入集群创建页，并做如下配置：
   - 选择集群类型【hadoop】
   - 选择集群版本“BMR 1.2.0(hadoop 2.7.1)”
   - 在可选服务中选择需要的服务，如spark(2.1.0)/hive(1.2.0)/hue(3.10.0)/tez(0.7.0)
   - 打开日志开关
   - 设置集群名称
   - 设置管理员密码
   - 点击下一步，选择对于的实例，支付即可。
4. 请保持集群的其他默认配置不变，点击“提交订单”。支付订单后，可在集群列表页可查看已创建的集群，当集群状态由“启动中”变为“运行中”时，集群创建成功。

登录Hue Web界面

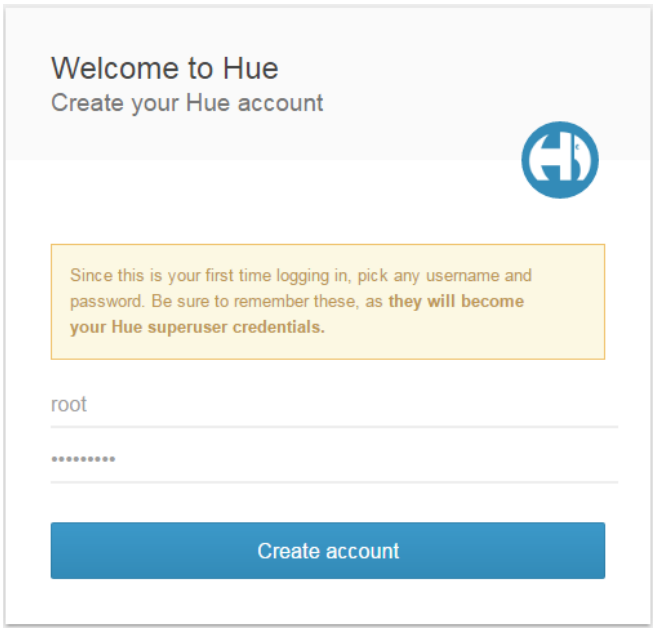
1. 登录控制台，选择“产品服务->MapReduce BMR”，点击已创建的集群，进入该集群详情页。
2. 在“相关应用”栏中点击“Hue Web UI”。



3. 在弹出的认证页面中输入创建集群时设置的用户名和密码，并点击“登录”。



4. 创建您登录Hue服务的用户名和密码，输入后点击“Create Account”后进入Hue Web界面。



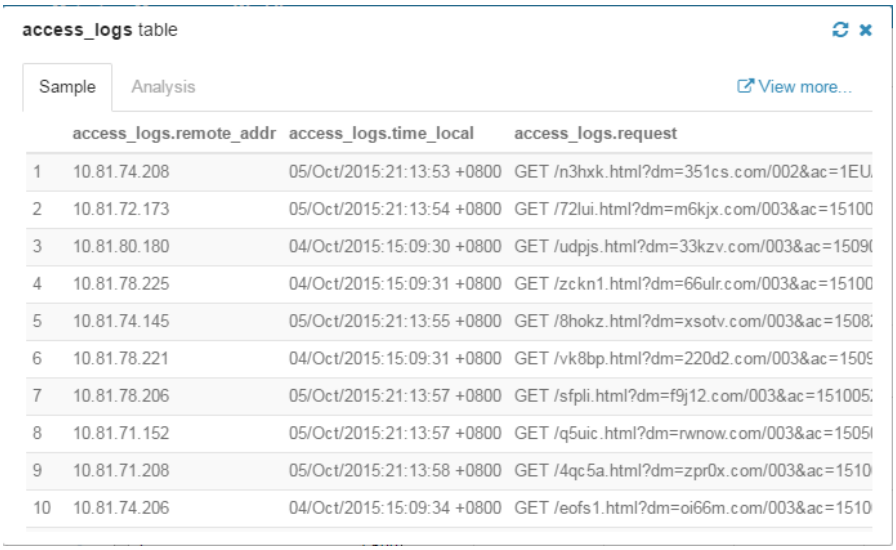
建表

1. 在分析之前，首先需要根据网站日志建立一张Hive表。在Hue菜单栏中选择“Query Editors>Hive”，并输入以下SQL语句：

```
 DROP TABLE IF EXISTS access_logs;
CREATE EXTERNAL TABLE access_logs(
    remote_addr STRING comment 'client IP',
    time_local STRING comment 'access time',
request STRING comment 'request URL',
status STRING comment 'HTTP status',
body_bytes_sent STRING comment 'size of response body',
http_referer STRING comment 'referer',
http_cookie STRING comment 'cookies',
remote_user STRING comment 'client name',
http_user_agent STRING comment 'client browser info',
request_time STRING comment 'consumed time of handling request',
host STRING comment 'server host',
msec STRING comment 'consumed time of writing logs'
)
COMMENT 'web access logs'
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
"input.regex" = "([0-9\\.]+) - \\[([^\\]]+)\\] \"([^\"]*)\" ([\\d]+) ([\\d]*) \"([^\"]*)\" \"([^\"]*)\" ([\\S]+) \"([^\"]*)\" ([0-9\\.]+) ([\\S]+) ([0-9\\.]+)"
)
STORED AS TEXTFILE
LOCATION "bos://datamart-bj/web-log-10k";
```

2. 输入语句后点击执行按钮，Hive会重建access_logs表，然后通过正则表达式来解析日志文件。

3. 成功创建access_logs表之后，点击左侧“Table”栏中的刷新按钮，找到access_logs表并预览示例数据：



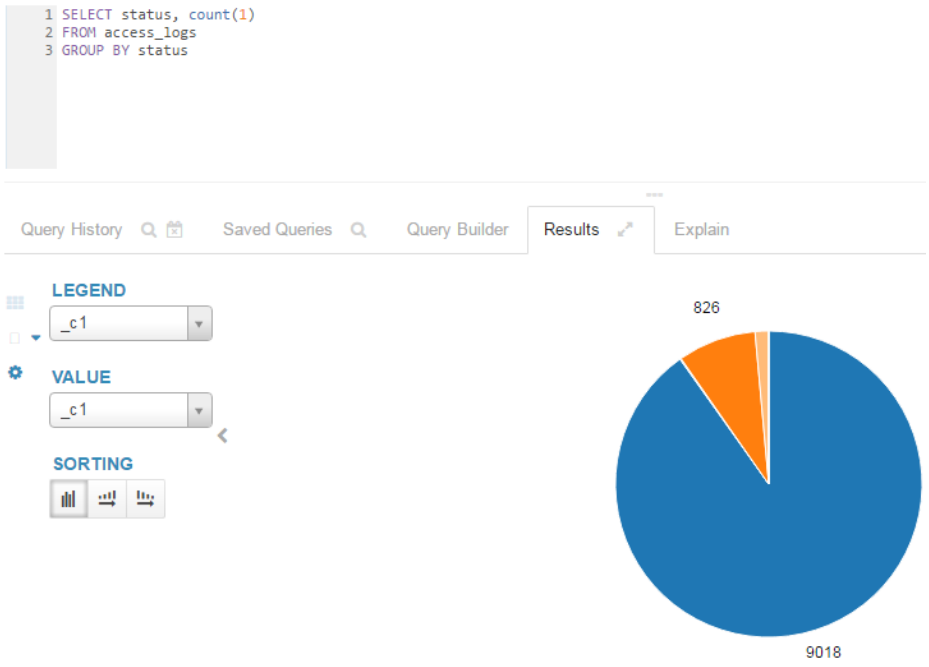
查询

定了表之后，便可以进行查询了。

- 若统计网页请求的结果，可使用以下语句：

```
SELECT status, count(1)  FROM access_logs  GROUP BY status
```

查询结果可切换到图表页，还可以以饼图的形式可视化数据，如下图所示：

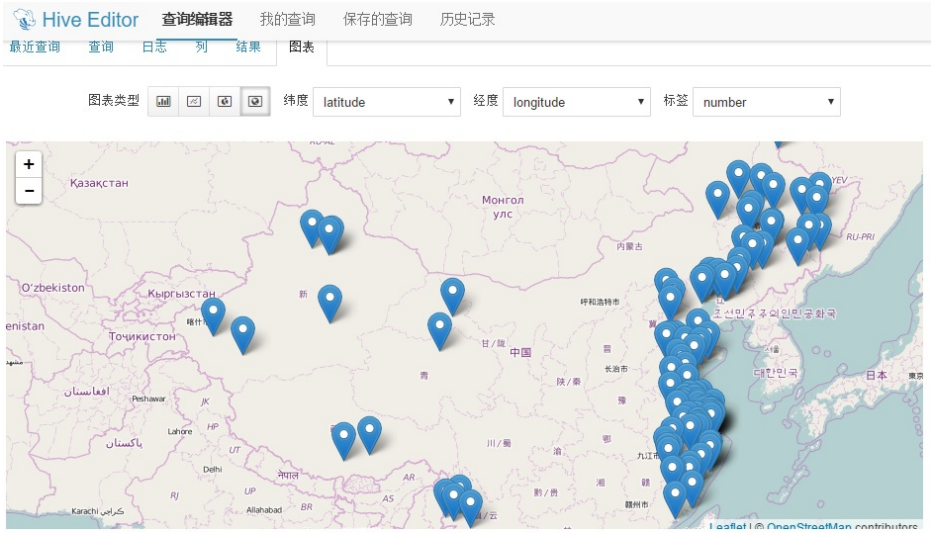


- 若想了解网页访问量最大的时段，可使用下面的语句：

```
 SELECT hour(from_unixtime(unix_timestamp(time_local, 'dd/MMMM/yyyy:HH:mm:ss Z'))) as hour, count(1) as pv
FROM access_logs
GROUP BY hour(from_unixtime(unix_timestamp(time_local, 'dd/MMMM/yyyy:HH:mm:ss Z')))
```

查询结果可切换到图表页，或以柱状图来更直观的查看结果：

在百度智能云，不仅可通过Hue使用Hive，还可通过Hue轻松制作地图的可视化效果，如下图所示，根据日志信息生成用户地图。欲了解详情，请联系bce@baidu.com。



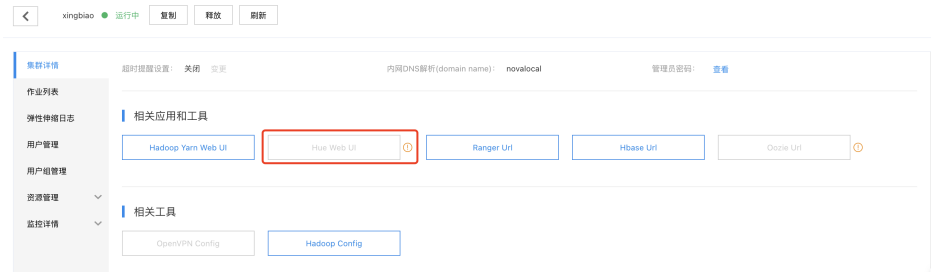
Hue 4.8.0使用

创建集群

1. 准备数据，请参考数据准备。
2. 准备百度智能云环境。
3. 登录控制台，选择“产品服务->MapReduce BMR”，点击“创建集群”，进入集群创建页，并做如下配置：
   - 选择集群类型【hadoop】
   - 选择集群版本“BMR 2.3.0(hadoop 3.1.1)”
   - 在可选服务中选择需要的服务，如spark(2.4.4)/hive(3.1.0)/hue(4.8.0)/tez(0.9.1)
   - 打开日志开关，填写对应的存放日志的BOS bucket
   - 设置集群名称
   - 设置管理员密码
   - 点击下一步，选择对于的实例，支付即可。
4. 请保持集群的其他默认配置不变，点击“提交订单”。支付订单后，可在集群列表页可查看已创建的集群，当集群状态由“启动中”变为“运行中”时，集群创建成功。

登录Hue Web界面

1. 登录控制台，选择“产品服务->MapReduce BMR”，点击已创建的集群，进入该集群详情页。
2. 在“相关应用和工具”栏中可以查看“Hue Web UI”。



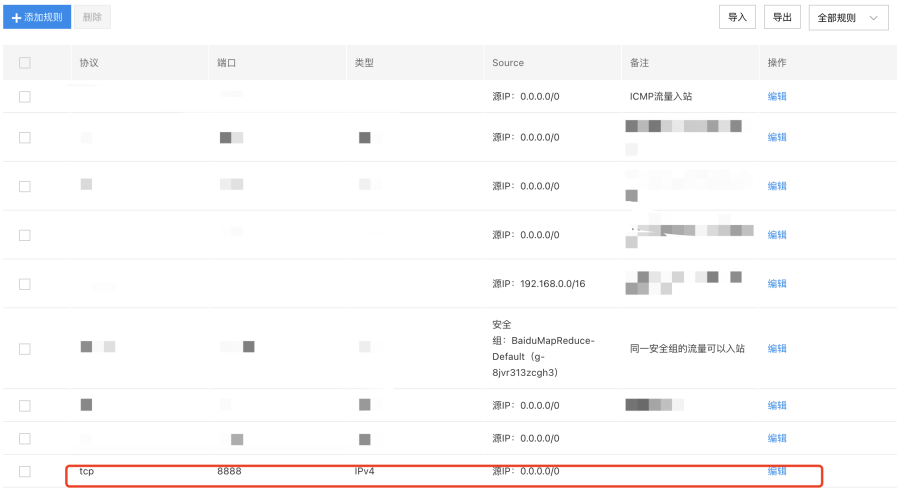
Hue4.8.0默认不可点击，需要在给对应的master实例绑定EIP后并且开通对应的端口后才可以访问。 具体需要绑定EIP的实例ID可以将鼠标指针上浮在Hue Web UI叹号处查看。绑定EIP见绑定EIP。若无EIP，请先于产品【网络——弹性公网IP EIP】创建EIP。详见：创建EIP实例。

3.给BMR安全组添加8888端口访问规则。 在给对应的节点绑定eip后，在集群详情-网络处可以查看对应BMR安全组ID。

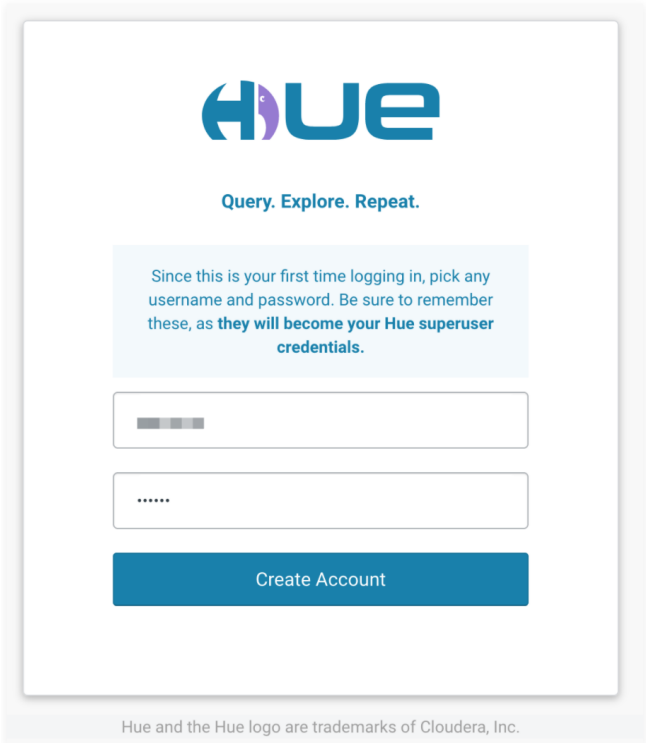
复制对应的BMR安全组ID，进入VPC-安全组中，找到对应的BMR安全组。

点击添加规则，选择HTTP，规则类型为IPV4,协议为tcp，端口范围为8888，source建议填写为需要访问BMR集群的IP或者IP段，不建议为all，本示例为方便演示选择all（0.0.0.0/0）。

添加完成后如下图所示：



4. 完成上述工作后，在BMR集群详情页面中点击Hue Web UI即可访问。创建您登录Hue服务的用户名和密码，输入后点击“Create Account”后进入Hue Web界面。



运维相关

hue 忘记管理员密码怎么办? 重置管理员密码操作如下:

1.登录集群主master节点,使用hue用户(su hue)

2.执行:/opt/bmr/hue/build/env/bin/hue shell

3.依次运行以下命令:

In [1]: from django.contrib.auth.models import User

In [2]: a = User.objects.get(username='用户名')

In [3]: a.is_staff = True

In [4]: a.is_superuser = True

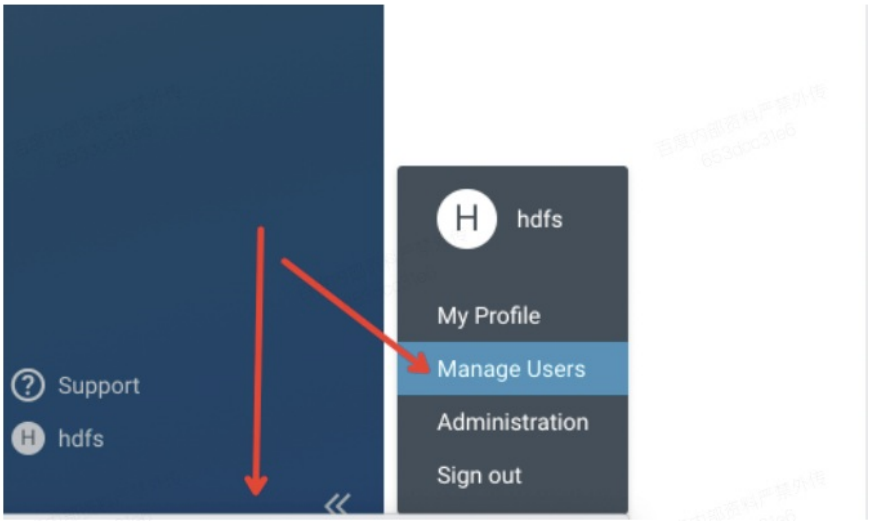
In [5]: a.set_password('xxxxxx')

In [6]: a.save()

In [7]:quit()

重置成功，密码为第五步执行的xxxxxx，用户可以自行从新设置管理员密码。

hue 使用登录 hue 的用户提交作业，新增用户步骤 在hue web 页面左下角进入 Manage Users



点击右上角 Add user，依次按照引导进行即可。

更多使用方式

请参考Hue官方文档。

Presto

Presto简介

Presto是一个分布式SQL查询引擎，用于查询分布在一个或多个不同数据源中的大数据集。Presto通过使用分布式查询，可以快速高效的完成海量数据的查询，并提供了Web UI页面方便用户查看任务查询详情与服务运行状态。

创建集群

登录百度云控制台，选择“产品服务->MapReduce BMR”，点击“创建集群”，进入集群创建页，BMR2.0.0及以上版本已支持 Presto 组件集成，购置集群时勾选 Presto 组件即可, 如下图所示:

使用简介

Presto支持本地和远程两种操作方式。

本地连接presto

登录集群机器

输入命令，进入presto终端交互页面：presto --catalog XXX --schema XXX ，示例如下： （参数含义：--catalog 是使用的数据源配置；--schema 用户选择查询的schema）

```
 presto --catalog hive --schema test
presto:test>
```

执行查询命令，示例如下:

```
 presto:test> select * from t2;
  id
---------
 1
(1 row)

Query 20190517_075918_00019_hs9fa, FINISHED, 1 node
Splits: 17 total, 17 done (100.00%)
0:01 [1 rows, 208B] [1 rows/s, 324B/s]
```

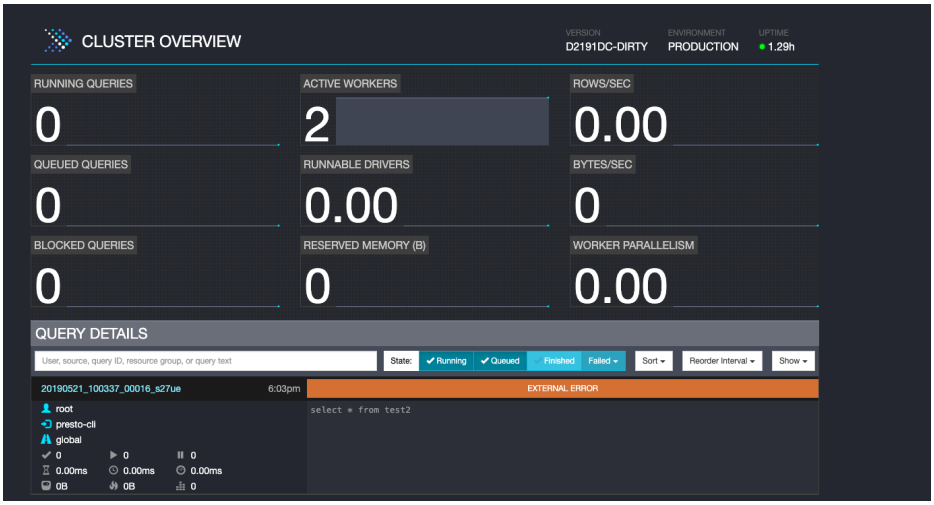
远程连接presto

将集群机器/opt/bmr/presto/bin目录下的presto-cli-0.219-executable.jar拷贝到本地 配置OpenVPN 执行连接命令：./presto-cli-0.219-executable.jar --server internal_ip:8089 --catalog XXX --schema XXX 即可连接到presto服务

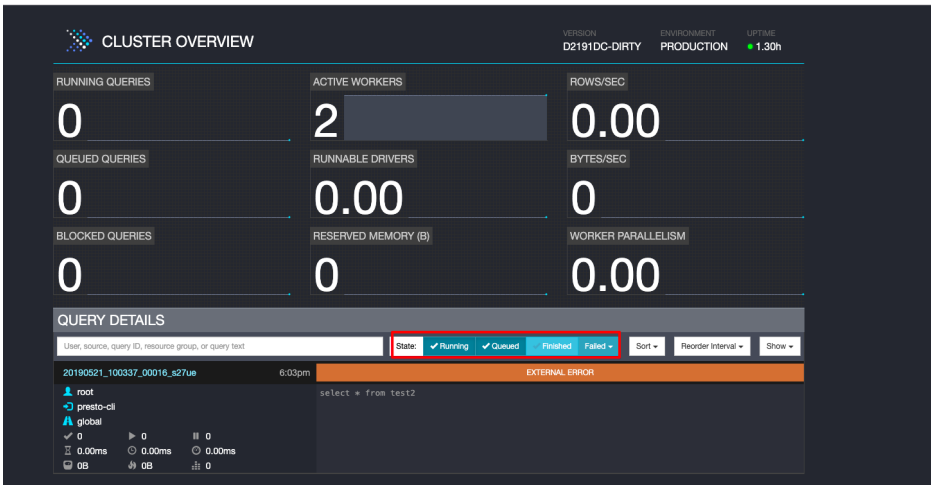
Presto UI界面

访问步骤：

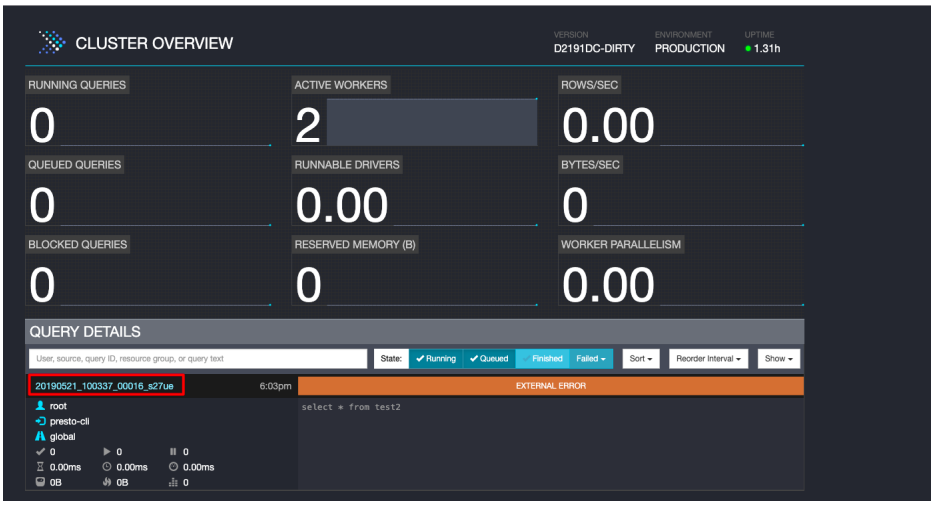
1. 配置OpenVPN
2. 在Master节点上执行 hostname 命令获取主机名
3. 访问http://hostname:port (master_hostname:8089) 即可查看presto的监控页面，提供服务监控与查询详情等信息：

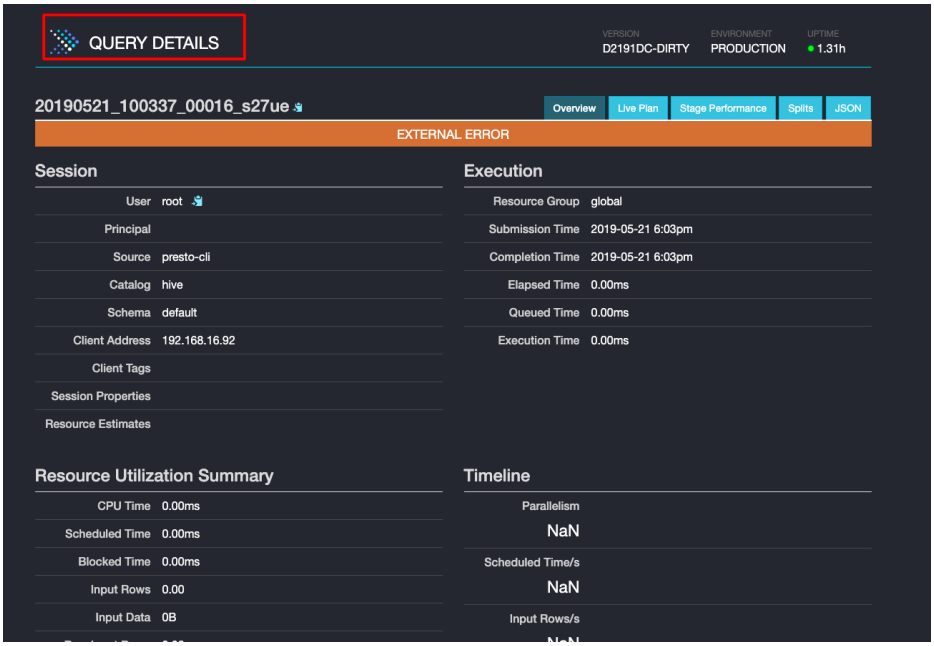


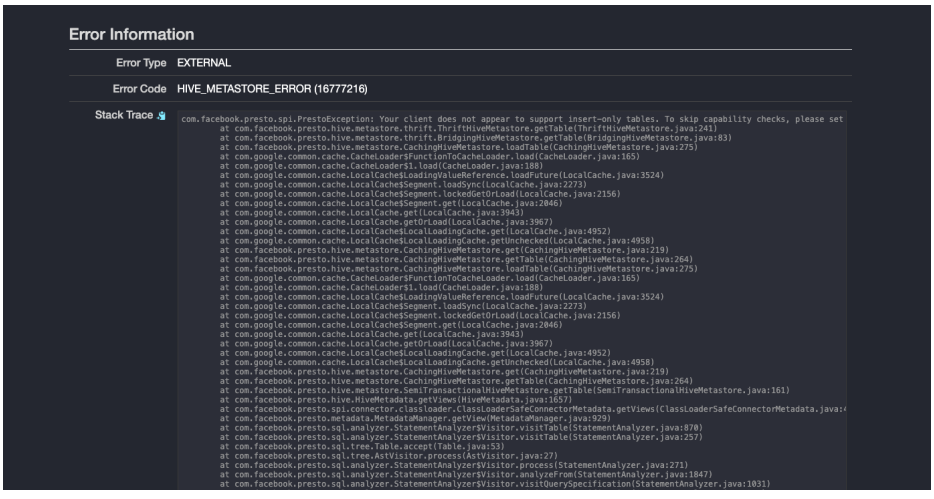
4. 可以通过页面的state选项可以对查询任务进行筛选：



5. 点击查询编号，可以进一步查看查询详情；对于失败的任务，也可以在查询详情页面查看相关日志：







注意事项

由于Hive3 对managed tables实现了全控制，所以Persto不支持Hive的managed table查询、更改等操作；若需要通过Presto对Hive table进行查询等操作，请通过Hive创建External tables。 https://github.com/prestodb/presto/issues/12484 示例如下： 通过hive创建external tables，通过presto进行查询，查询成功：

```
 # 连接hive，创建外表
hive> create external table test1 (id bigint) stored as textfile;
hive> insert into test1 values(123);

# 连接Presto，查询
presto:test> select * from test1;
 id
-----
 123
(1 row)
```

通过hive创建managed tables，通过presto进行查询，查询失败：

```
 # 连接hive，创建managed table
hive> create table test2(id bigint) stored as textfile;

# 连接Presto，查询
presto:test> select * from test2;
Query XXX failed: XXX
```

Zeppelin

Zeppelin简介

zeppelin 是一个交互式数据分析工具，可支持spark、sql等数据分析工具（详细介绍，请参考zeppelin 官网）。

本文将介绍如何在zeppelin上链接配置hiveserver2，来介绍zeppelin上sql的基本使用。

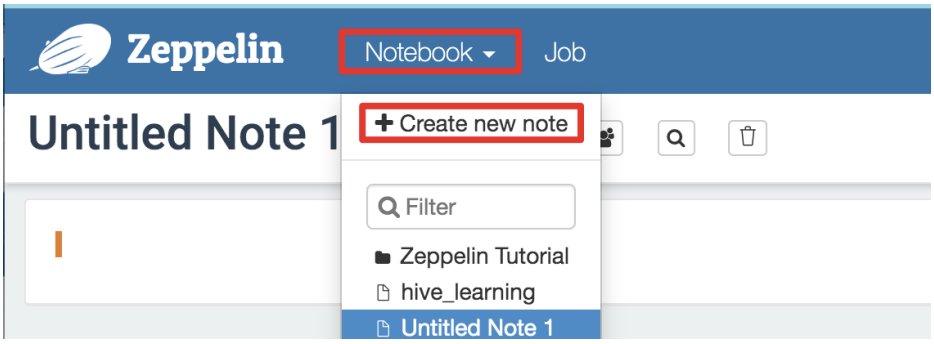
集群准备

准备百度智能云环境。

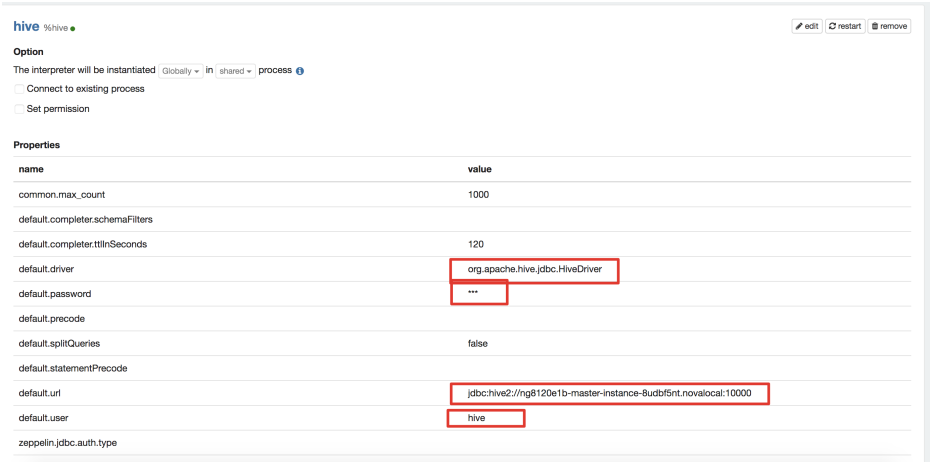
1. 登录控制台（百度智能云登录平台），选择“产品服务->MapReduce BMR”，点击“创建集群”，进入集群创建页，并做如下配置：
    - 设置集群名称
    - 设置管理员密码
    - 关闭日志开关（如果打开，需要选择存放日志用的bos目录，bos目录的bucket必须已经存在
    - 选择镜像版本“BMR 2.0(hadoop 3.1)“ （只有BMR2.0 及以上版本的zeppelin方可用）
    - 选择内置模板“zeppelin” （默认会自动勾选hive；如果需要使用spark，请手动勾选spark组件）
    - 高可用开关默认打开，可选择关闭HA模式
    - 集群网络和安全设置保持默认即可
    - 点击下一步，选择各个组的机器配置(master节点建议cpu核数 >= 8, 内存 >= 16G)和机器数量(master节点跟上一步中的高可用模式打开或者关闭有关）

保持其他配置为默认值，点击下一步后，再请点击“去支付”可在集群列表页可查看已创建的集群，当集群状态由“初始化中”变为“空闲中”时，集群创建成功。

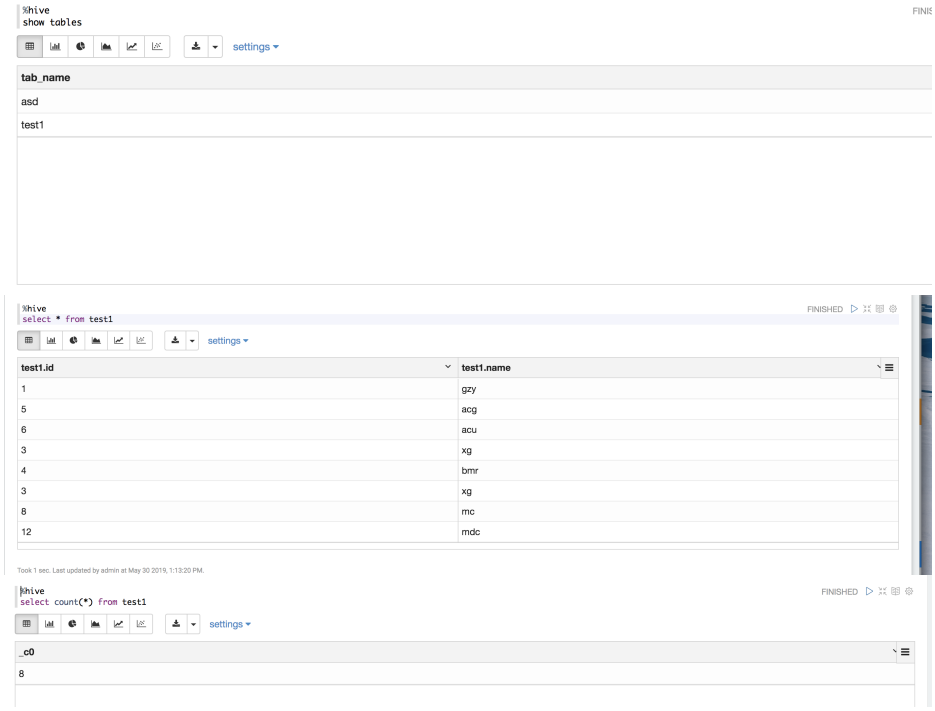
2. 访问集群
    - 首先参考访问集群建立本地浏览器能访问集群的网络环境（可以是ssh方式也可以是openvpn方式）
    - 登录集群master节点，在终端输入hostname命令可得到集群的fqdn名称(称作hostname_master)
    - 浏览器输入$hosname_master:9995即可链接到zeppelin UI界面
3. 使用zeppelin
    - login 默认账户名和密码是admin/admin
    - 新建notebook命名为hive



- 关键参数配置(选择jdbc group，配置hive时，要配置四个选项：driver, user, passwd, jdbc connection url)



- 执行命令



参考文档：

1. http://zeppelin.apache.org/docs/0.8.0/index.html
2. https://zeppelin.apache.org/

Druid

Druid简介

Druid是一个高性能的实时数据分析系统，由MetaMarkets公司在2012开源，专门为OLAP场景而设计。Druid遵从Lambda架构，支持批量和实时两种方式导入数据，并提供高性能的数据查询。

集群准备

Druid模版

登录百度云控制台，选择“产品服务->百度MapReduce BMR”，点击“创建集群”，进入集群创建页。购置集群时选择Druid模版即可，另外Druid的元数据存储支持本地MySQL和云数据库RDS MySQL。如下图所示:

节点类型

MASTER节点 集群的主节点

BMR的Master节点上部署Druid的Overlord、Coordinator、Broker和Router，供提交任务、查看数据源和数据查询。

CORE节点 有hdfs部署的计算节点

TASK节点 无hdfs部署的计算节点，适合扩展

为了更容易扩展节点，BMR的Core节点部署Druid的Historical和MiddleManager, Task节点部署Druid的Broker和MiddleManager。可以按照需求变更Core和Task的节点数目。

> Druid对内存的要求较高，建议使用4核16GB（或更高配置）的节点配置。在Druid集群内部，建议不要使用MapReduce跑其他开源组件的作业。

使用简介（批量索引）

Druid使用的端口如下：

Druid节点类型	端口
Broker	8590
Overlord	8591
Coordinator	8592
Router	8593
Historical	8594
MiddleManager	8595

Druid提供Http访问，每个请求都会返回结果，如果使用curl命令没有任何返回，可以加上-v查看具体的响应信息。

1. 远程登录到创建好的集群中

> `ssh root@[master节点公网ip]`
> 使用创建集群时输入的密码

2. 切换成hdfs用户，在HDFS上创建quickstart目录

> `hdfs dfs -mkdir /user/druid/quickstart`

3. 将示例数据文件wikiticker-2015-09-12-sampled.json.gz拷贝到HDFS

> `hdfs dfs -copyFromLocal /opt/bmr/druid/quickstart/wikiticker-2015-09-12-sampled.json.gz /user/druid/quickstart`

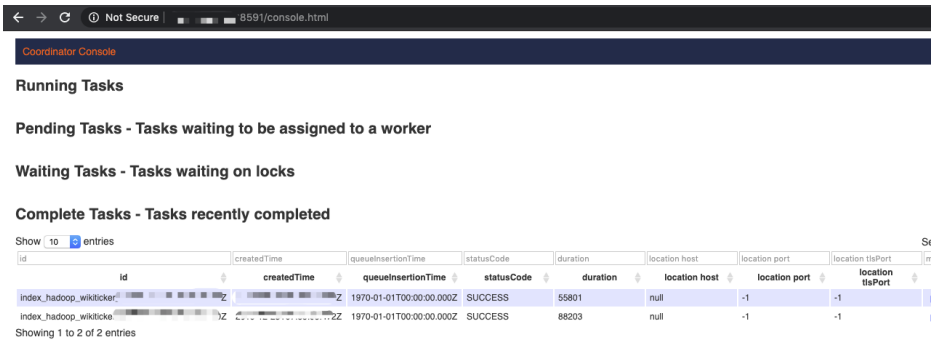
4. 向Overlord提交批量索引任务

> `curl -v -X POST -H 'Content-Type: application/json' -d @/opt/bmr/druid/quickstart/wikiticker-index.json http://xxx-master-instance-xxx-1（hostname）:8591/druid/indexer/v1/task`
> 成功返回结果为{"task":"index_hadoop_wikiticker_yyyy-MM-ddThh:mm:ss.xxZ"}

如果使用的是高可用集群，Overlord只有一个为Active，有效的**hostname**可能是xxx-master-instance-xxx-2。

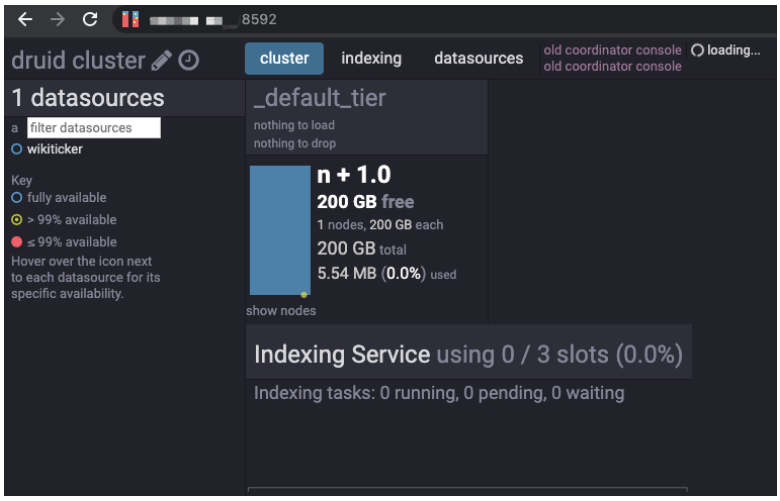
5. 使用Web UI查看任务执行情况

> 使用ssh -ND [本地未使用端口] root@[公网ip] 并对浏览器做相关配置，例如Chrome浏览器通过Swichysharp配置了VPN之后（通过SSH-Tunnel访问集群），或者使用OpenVPN，可通过Master节点的**内网ip**加上Overlord端口（8591）查看结果，如下图所示：



6. 等到索引任务执行成功，查看数据源

> 使用Web UI：配置好VPN后，通过Master节点的**内网ip**加上Coordinator端口（8592）查看结果，如下图所示：



使用Http：

`curl -v -X GET http://xxx-master-instance-xxx-1:8592/druid/coordinator/v1/datasources`

成功返回结果为["wikiticker"]

如果使用的是高可用集群，Coordinator只有一个为Active，有效的hostname可能是xxx-master-instance-xxx-2。

7. 查询数据

> 当Overlord Web UI显示任务成功或者查看数据源显示有wikiticker的时候，可以使用Broker查询数据：
> `curl -v -X 'POST' -H 'Content-Type:application/json' -d @/opt/bmr/druid/quickstart/wikiticker-top-pages.json http://[master/task hostname或ip]:8590/druid/v2?pretty`

Tips: 如果需要使用自己的Druid Spec JSON，建议在~目录下编辑，并在本地做好备份。Spec的编写请参考Apache Druid官方文档。

使用Kafka Indexing Service

Druid提供了Kafka Indexing Service摄入实时数据，详情参考Druid官方文档。BMR托管的Druid可支持百度消息服务BMS、BMR内部和放在同一VPC下的Kafka集群。本示例使用百度消息服务BMS演示，通过其他方式部署的Kafka集群步骤类似。

1. 使用百度消息服务BMS创建主题 `kafka_demo`，然后下载BMS的证书，用于后续创建Kafka生产者和消费者，参考BMS文档
2. 在~目录（/home/hdfs），将下载完毕的BMS证书上传并解压到此处，然后scp到其他节点

```
unzip -d kafka-key kafka-key.zip
scp -rp kafka-key root@[节点ip或hostname]:~/
```

3. 在~目录（/home/hdfs）下编写Druid Spec文件kafka_demo.json，BMS的主题需要带上创建后生成的前缀

查看client.properties文件，替换kafka_demo.json里的ssl.keystore.password

```
{
  "type": "kafka",
  "dataSchema": {
    "dataSource": "wiki_kafka_demo",
    "parser": {
      "type": "string",
      "parseSpec": {
        "format": "json",
        "timestampSpec": {
          "column": "time",
          "format": "auto"
        },
        "dimensionsSpec": {
          "dimensions": [
            "channel",
            "cityName",
            "comment",
            "countryIsoCode",
            "countryName",
            "isAnonymous",
            "isMinor",
            "isNew",
            "isRobot",
            "isUnpatrolled",
            "metroCode",
            "namespace",
            "page",
            "regionIsoCode",
            "regionName",
            "user",
            { "name": "added", "type": "long" },
            { "name": "deleted", "type": "long" },
            { "name": "delta", "type": "long" }
          ]
        }
      }
    },
    "metricsSpec" : [],
    "granularitySpec": {
      "type": "uniform",
      "segmentGranularity": "DAY",
      "queryGranularity": "NONE",
      "rollup": false
    }
  },
  "tuningConfig": {
    "type": "kafka",
    "reportParseExceptions": false
  },
  "ioConfig": {
    "topic": "[accountID]__kafka_demo",
    "replicas": 2,
    "taskDuration": "PT10M",
    "completionTimeout": "PT20M",
    "consumerProperties": {
      "bootstrap.servers": "[kafka_address]:[kafka_port]",
      "security.protocol": "SSL",
      "ssl.truststore.password": "kafka",
      "ssl.truststore.location": "/home/hdfs/kafka-key/client.truststore.jks",
      "ssl.keystore.location": "/home/hdfs/kafka-key/client.keystore.jks",
      "ssl.keystore.password": "******"
    }
  }
}
```

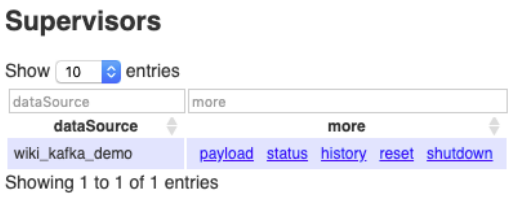
> **若使用的不是开启SSL的Kafka集群，consumerProperties只需要配置bootstrap.servers即可，例如BMR集群中的Kafka。**

4. 向Overlord提交Kafka Supervisor任务

```
curl -XPOST -H'Content-Type: application/json' -d @kafka_demo.json http://[overlord_ip]:8591/druid/indexer/v1/supervisor
```

成功返回：

{"id":"wiki_kafka_demo"}，Overlord Web UI上也可以看到结果：

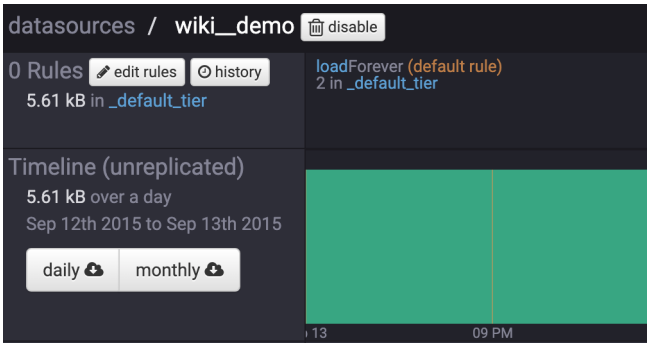


5. 本地目录解压BMS证书压缩包，使用client.propertie启动Kafka生产者并发送消息

```
sh bin/kafka-console-producer.sh --producer.config client.properties --topic <accountID>__kafka_demo --sync --broker-list [kafka_address]:[kafka_port]

{"time":"2015-09-12T23:58:31.643Z","channel":"#en.wikipedia","cityName":null,"comment":"Notification: tagging for deletion of [[File:Axintele
team.gif]].
([[WP:TW|TW]])","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":false,"isUnpatrolled":false,"metroCo
r talk","page":"User talk:AlexGeorge33","regionIsoCode":null,"regionName":null,"user":"Sir Sputnik","delta":1921,"added":1921,"deleted":0}
{"time":"2015-09-12T23:58:33.743Z","channel":"#vi.wikipedia","cityName":null,"comment":"clean up using
[[Project:AWB|AWB]]","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":true,"isUnpatrolled":false,"met
m finisterrae","regionIsoCode":null,"regionName":null,"user":"ThitxongkhoiAWB","delta":18,"added":18,"deleted":0}
{"time":"2015-09-12T23:58:35.732Z","channel":"#en.wikipedia","cityName":null,"comment":"Put info into infobox, succession box, split sections a
bit","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":false,"isUnpatrolled":false,"metroCode":null,"na
l Holley","regionIsoCode":null,"regionName":null,"user":"Mr. Matté","delta":3427,"added":3427,"deleted":0}
{"time":"2015-09-12T23:58:38.531Z","channel":"#uz.wikipedia","cityName":null,"comment":"clean up, replaced: ozbekcha → o´zbekcha, olchami →
o´lchami (3) using
[[Project:AWB|AWB]]","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":true,"isNew":false,"isRobot":true,"isUnpatrolled":false,"metr
}
{"time":"2015-09-12T23:58:41.619Z","channel":"#it.wikipedia","cityName":null,"comment":"/* Gruppo Discovery Italia
*/","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":false,"isUnpatrolled":true,"metroCode":null,"nar
y TV","regionIsoCode":null,"regionName":null,"user":"CiosI","delta":4,"added":4,"deleted":0}
{"time":"2015-09-12T23:58:43.304Z","channel":"#en.wikipedia","cityName":null,"comment":"/* Plot
*/","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":false,"isUnpatrolled":false,"metroCode":null,"na
r Moon (film)","regionIsoCode":null,"regionName":null,"user":"Siddharth Mehrotra","delta":0,"added":0,"deleted":0}
{"time":"2015-09-12T23:58:46.732Z","channel":"#en.wikipedia","cityName":null,"comment":"/* Jeez */ delete (also fixed Si Trew's syntax error, as I
think he meant to make a link
there)","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":false,"isUnpatrolled":false,"metroCode":null,
s for discussion/Log/2015 September 12","regionIsoCode":null,"regionName":null,"user":"JaykeBird","delta":293,"added":293,"deleted":0}
{"time":"2015-09-
12T23:58:48.729Z","channel":"#fr.wikipedia","cityName":null,"comment":"MàJ","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":f
s articles/Nouveau","regionIsoCode":null,"regionName":null,"user":"Gemini1980","delta":76,"added":76,"deleted":0}
{"time":"2015-09-12T23:58:51.785Z","channel":"#vi.wikipedia","cityName":null,"comment":"clean up using
[[Project:AWB|AWB]]","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":true,"isUnpatrolled":false,"met
a yezonis","regionIsoCode":null,"regionName":null,"user":"ThitxongkhoiAWB","delta":10,"added":10,"deleted":0}
{"time":"2015-09-12T23:58:53.898Z","channel":"#vi.wikipedia","cityName":null,"comment":"clean up using
[[Project:AWB|AWB]]","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":true,"isUnpatrolled":false,"met
a triverticillata","regionIsoCode":null,"regionName":null,"user":"ThitxongkhoiAWB","delta":36,"added":36,"deleted":0}
```

6. 通过Coordinator Web UI查看实时数据源



跨集群使用Druid

Druid支持通过跨集群和其他Hadoop组件联合使用，如Hadoop、Hive、Spark等。在集群创建页将多个集群创建在同一个私有网络（VPC）之下，curl命令或者Http客户端就可以指定IP和端口向Druid各组件发送跨集群的请求。

集群基础设置

> 目前只支持通过IP地址访问同一个VPC下的另一个集群的主机。

使用Hive简化Druid操作

在Hive中可以通过指定IP地址和端口访问Druid节点。

本节提供一个能够运行的示例，更多使用详情可参考Apache Hive官方文档https://cwiki.apache.org/confluence/display/Hive/Druid+Integration

示例一：查询

推荐使用Hive查询Druid数据，可以简化Druid常用的查询，不用为了简单的查询而去编写复杂的JSON文件。

1. 创建一个Hive集群，登录Hive集群

> `ssh root@[hive集群公网ip]`
> 使用创建hive集群时输入的密码

2. 指定Druid集群Broker的IP地址和端口

> hive> `SET hive.druid.broker.address.default=x.x.x.x:8590;`

3. 创建外表（确保按照之前的示例，创建了`wikiticker`数据源）

> hive> `CREATE EXTERNAL TABLE druid_hive_demo_01`
> `STORED BY 'org.apache.hadoop.hive.druid.DruidStorageHandler'`
> `TBLPROPERTIES ("druid.datasource" = "wikiticker");`

4. 查看元信息

> hive> `DESCRIBE FORMATTED druid_hive_demo_01;`
> 其中维度列是string类型，指标列是bigint类型，时间戳字段__time是timestamp with local time zone类型

5. 查询数据

hive> `SELECT `__time`, count, page, `user`, added FROM druid_hive_demo_01 LIMIT 10;`

> 注意：使用Hive无法扫描Druid全表，如上面的例子不使用limit会报错

__time是Druid表（数据源）的时间戳字段，在Hive中需要用反引号转义，返回结果如下：

```
 2015-09-12 08:46:58.771 Asia/Shanghai 1 36 Talk:Oswald Tilghman GELongstreet
2015-09-12 08:47:00.496 Asia/Shanghai 1 17 Rallicula PereBot
2015-09-12 08:47:05.474 Asia/Shanghai 1 0 Peremptory norm 60.225.66.142
2015-09-12 08:47:08.77 Asia/Shanghai 1 18 Apamea abruzzorum Cheers!-bot
2015-09-12 08:47:11.862 Asia/Shanghai 1 18 Atractus flammigerus ThitxongkhoiAWB
2015-09-12 08:47:13.987 Asia/Shanghai 1 18 Agama mossambica ThitxongkhoiAWB
2015-09-12 08:47:17.009 Asia/Shanghai 1 0 Campanya dels Balcans (1914-1918) Jaumellecha
2015-09-12 08:47:19.591 Asia/Shanghai 1 345 Talk:Dani Ploeger New Media Theorist
2015-09-12 08:47:21.578 Asia/Shanghai 1 121 User:WP 1.0 bot/Tables/Project/Pubs WP 1.0 bot
2015-09-12 08:47:25.821 Asia/Shanghai 1 18 Agama persimilis ThitxongkhoiAWB
```

示例二：导入历史数据

可以使用已有的Hive表，通过建外表将HDFS数据导入到Druid中，不必编写复杂的Druid Ingestion Spec文件。注意，必须指定时间戳字段`__time`，且时间戳类型为timestamp with local time zone。

1. 进入~目录，下载示例数据文件，创建示例Hive表

```
 wget http://files.grouplens.org/datasets/movielens/ml-100k.zip
unzip ml-100k.zip

hive>
CREATE TABLE u_data (
    userid INT,
    movieid INT,
    rating INT,
    unixtime STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH 'ml-100k/u.data'
OVERWRITE INTO TABLE u_data;
```

2. 设置Hive属性，关联Druid

```
 // 获取druid集群的元数据库信息：ssh root@public_ip 'cat /etc/druid/conf/_common/common.runtime.properties | grep
"druid.metadata.storage.connector"'
// 跨集群访问druid集群的mysql，需要使用ip地址，若使用hostname需要更新hive集群的hosts文件
// 需要删除外表时开启，可以清空Druid数据: SET external.table.purge=true;
SET hive.druid.metadata.uri=jdbc:mysql://[druid_mysql_ip]:3306/druid?characterEncoding=UTF-8;
SET hive.druid.metadata.username=[druid_user];
SET hive.druid.metadata.password=[druid_passwd];
SET hive.druid.broker.address.default=[druid_broker_ip]:8590;
SET hive.druid.overlord.address.default=[druid_overlord_ip]:8591;
SET hive.druid.coordinator.address.default=[druid_coordinator_ip]:8592;
SET hive.druid.storage.storageDirectory=hdfs://[hive_hdfs_ip]:8020/user/druid/warehouses;
```

3. 方法一：先建表再导入数据

```
 CREATE EXTERNAL TABLE druid_hive_demo_02
(`__time` TIMESTAMP WITH LOCAL TIME ZONE, userid STRING, moveid STRING, rating INT)
STORED BY 'org.apache.hadoop.hive.druid.DruidStorageHandler'
TBLPROPERTIES (
    "druid.segment.granularity" = "MONTH",
    "druid.query.granularity" = "DAY"
);

INSERT INTO druid_hive_demo_02
SELECT
    cast (from_unixtime(cast(unixtime as int)) as timestamp with local time zone)  as `__time`,
    cast(userid as STRING) userid,
    cast(movieid as STRING) moveid,
    cast(rating as INT) rating
FROM u_data;
```

方法二，使用CTAS语句创建Druid表并导入数据

```
 // 首先打开Hive外表的CTAS开关
SET hive.ctas.external.tables=true;

CREATE EXTERNAL TABLE druid_hive_demo_02
STORED BY 'org.apache.hadoop.hive.druid.DruidStorageHandler'
TBLPROPERTIES (
    "druid.segment.granularity" = "MONTH",
    "druid.query.granularity" = "DAY")
AS
SELECT
    cast (from_unixtime(cast(unixtime as int)) as timestamp with local time zone)  as `__time`,
    cast(userid as STRING) userid,
    cast(movieid as STRING) moveid,
    cast(rating as INT) rating
FROM u_data;
```

> 如果CTAS语句执行失败，创建的表仍然存在，可以继续使用INSERT语句插入数据，或者删除该表并重新执行。

4. 查询结果

```
hive> SELECT * FROM druid_hive_demo_02 LIMIT 10;
OK
1997-09-19 08:00:00.0 Asia/Shanghai 259 1074 3
1997-09-19 08:00:00.0 Asia/Shanghai 259 108  4
1997-09-19 08:00:00.0 Asia/Shanghai 259 117  4
1997-09-19 08:00:00.0 Asia/Shanghai 259 173  4
1997-09-19 08:00:00.0 Asia/Shanghai 259 176  4
1997-09-19 08:00:00.0 Asia/Shanghai 259 185  4
1997-09-19 08:00:00.0 Asia/Shanghai 259 200  4
1997-09-19 08:00:00.0 Asia/Shanghai 259 210  4
1997-09-19 08:00:00.0 Asia/Shanghai 259 255  4
1997-09-19 08:00:00.0 Asia/Shanghai 259 286  4
```

示例三：导入Kafka实时数据

1. 使用百度消息服务BMS创建主题 `kafka_demo`，并下载证书scp到druid集群各节点，参考BMS文档。

`scp -rp kafka-key root@[druid集群的节点ip或hostname]:~/`

2. 创建Hive表，通过属性关联BMS Kafka

```
// 获取druid集群的元数据库信息：ssh hdfs@public_ip 'cat /etc/druid/conf/_common/common.runtime.properties | grep "druid.metadata.storage.connector"'
// 跨集群访问druid集群的mysql，需要使用ip地址，若使用hostname需要更新hive集群的hosts文件
SET hive.druid.metadata.uri=jdbc:mysql://[druid_mysql_ip]:3306/druid?characterEncoding=UTF-8;
SET hive.druid.metadata.username=[druid_user];
SET hive.druid.metadata.password=[druid_passwd];
SET hive.druid.broker.address.default=[druid_broker_ip]:8590;
SET hive.druid.overlord.address.default=[druid_overlord_ip]:8591;
SET hive.druid.coordinator.address.default=[druid_coordinator_ip]:8592;
SET hive.druid.storage.storageDirectory=hdfs://[hive_hdfs_ip]:8020/user/druid/warehouses;
// Kafka Supervisor配置前缀是druid.kafka.ingestion
CREATE EXTERNAL TABLE druid_kafka_demo
(
    `__time`          TIMESTAMP WITH LOCAL TIME ZONE,
    `channel`         STRING,
    `cityName`        STRING,
    `comment`         STRING,
    `countryIsoCode`  STRING,
    `isAnonymous`     STRING,
    `isMinor`         STRING,
    `isNew`           STRING,
    `isRobot`         STRING,
    `isUnpatrolled`   STRING,
    `metroCode`       STRING,
    `namespace`       STRING,
    `page`            STRING,
    `regionIsoCode`   STRING,
    `regionName`      STRING,
    `added`           INT,
    `user`            STRING,
    `deleted`         INT,
    `delta`           INT
)
STORED BY 'org.apache.hadoop.hive.druid.DruidStorageHandler'
TBLPROPERTIES (
    "kafka.bootstrap.servers" = "[kafka_address]:[kafka_port]",
    "kafka.security.protocol" = "SSL",
    "kafka.ssl.truststore.password" = "kafka",
    "kafka.ssl.truststore.location" =  "/home/hdfs/kafka-key/client.truststore.jks",
    "kafka.ssl.keystore.location" =  "/home/hdfs/kafka-key/client.keystore.jks",
    "kafka.ssl.keystore.password" =  "******",
    "kafka.topic" = "[account_id]__kafka_demo",
    "druid.kafka.ingestion.useEarliestOffset" = "true",
    "druid.kafka.ingestion.maxRowsInMemory" = "5",
    "druid.kafka.ingestion.startDelay" = "PT1S",
    "druid.kafka.ingestion.period" = "PT1S",
    "druid.kafka.ingestion.taskDuration" = "PT10M",
    "druid.kafka.ingestion.completionTimeout" = "PT20M",
    "druid.kafka.ingestion.consumer.retries" = "2"
);
```

> **如果使用的不是开启了SSL的Kafka，可移除ssl相关配置，例如BMR集群里的Kafka**

3. 启动Kafka生产者发送数据

```
 sh bin/kafka-console-producer.sh --producer.config client.properties --topic druid_kafka_demo --sync --broker-list [kafka_address]:[kafka_port]

{"__time":"2015-09-12T23:58:31.643Z","channel":"#en.wikipedia","cityName":null,"comment":"Notification: tagging for deletion of [[File:Axintele
team.gif]].
([[WP:TW|TW]])","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":false,"isUnpatrolled":false,"metroCo
r talk","page":"User talk:AlexGeorge33","regionIsoCode":null,"regionName":null,"user":"Sir Sputnik","delta":1921,"added":1921,"deleted":0}
{"__time":"2015-09-12T23:58:33.743Z","channel":"#vi.wikipedia","cityName":null,"comment":"clean up using
[[Project:AWB|AWB]]","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":true,"isUnpatrolled":false,"met
m finisterrae","regionIsoCode":null,"regionName":null,"user":"ThitxongkhoiAWB","delta":18,"added":18,"deleted":0}
{"__time":"2015-09-12T23:58:35.732Z","channel":"#en.wikipedia","cityName":null,"comment":"Put info into infobox, succession box, split sections a
bit","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":false,"isUnpatrolled":false,"metroCode":null,"na
l Holley","regionIsoCode":null,"regionName":null,"user":"Mr. Matté","delta":3427,"added":3427,"deleted":0}
{"__time":"2015-09-12T23:58:38.531Z","channel":"#uz.wikipedia","cityName":null,"comment":"clean up, replaced: ozbekcha → o´zbekcha, olchami →
o´lchami (3) using
[[Project:AWB|AWB]]","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":true,"isNew":false,"isRobot":true,"isUnpatrolled":false,"metr
}
{"__time":"2015-09-12T23:58:41.619Z","channel":"#it.wikipedia","cityName":null,"comment":"/* Gruppo Discovery Italia
*/","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":false,"isUnpatrolled":true,"metroCode":null,"nar
y TV","regionIsoCode":null,"regionName":null,"user":"CiosI","delta":4,"added":4,"deleted":0}
{"__time":"2015-09-12T23:58:43.304Z","channel":"#en.wikipedia","cityName":null,"comment":"/* Plot
*/","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":false,"isUnpatrolled":false,"metroCode":null,"na
r Moon (film)","regionIsoCode":null,"regionName":null,"user":"Siddharth Mehrotra","delta":0,"added":0,"deleted":0}
{"__time":"2015-09-12T23:58:46.732Z","channel":"#en.wikipedia","cityName":null,"comment":"/* Jeez */ delete (also fixed Si Trew's syntax error, as I
think he meant to make a link
there)","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":false,"isUnpatrolled":false,"metroCode":null,
s for discussion/Log/2015 September 12","regionIsoCode":null,"regionName":null,"user":"JaykeBird","delta":293,"added":293,"deleted":0}
{"__time":"2015-09-
12T23:58:48.729Z","channel":"#fr.wikipedia","cityName":null,"comment":"MàJ","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":f
s articles/Nouveau","regionIsoCode":null,"regionName":null,"user":"Gemini1980","delta":76,"added":76,"deleted":0}
{"__time":"2015-09-12T23:58:51.785Z","channel":"#vi.wikipedia","cityName":null,"comment":"clean up using
[[Project:AWB|AWB]]","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":true,"isUnpatrolled":false,"met
a yezonis","regionIsoCode":null,"regionName":null,"user":"ThitxongkhoiAWB","delta":10,"added":10,"deleted":0}
{"__time":"2015-09-12T23:58:53.898Z","channel":"#vi.wikipedia","cityName":null,"comment":"clean up using
[[Project:AWB|AWB]]","countryIsoCode":null,"countryName":null,"isAnonymous":false,"isMinor":false,"isNew":false,"isRobot":true,"isUnpatrolled":false,"met
a triverticillata","regionIsoCode":null,"regionName":null,"user":"ThitxongkhoiAWB","delta":36,"added":36,"deleted":0}
```

4. 可以通过DDL控制Kafka导入数据的启停

```
 // 只有druid.kafka.ingestion为START，Hive才能够提交实时任务到Druid
ALTER TABLE druid_kafka_demo SET TBLPROPERTIES('druid.kafka.ingestion' = 'START');
ALTER TABLE druid_kafka_demo SET TBLPROPERTIES('druid.kafka.ingestion' = 'STOP');
ALTER TABLE druid_kafka_demo SET TBLPROPERTIES('druid.kafka.ingestion' = 'RESET');
```

5. 当第一个Segment上传成功，Coordinator显示出数据源后即可使用Hive查询数据，见示例一

Impala

Impala简介

Impala是Cloudera公司主导开发的MPP架构的查询系统，它提供SQL语义，能够快速的查询存储在HDFS、HBASE中的数据。此外Impala使用与Hive相同的元数据、SQL语法、ODBC驱动。

创建集群

登录百度云控制台，选择“产品服务->MapReduce BMR”，点击“创建集群”，进入集群创建页。BMR2.0.0及以上版本已支持 Impala 组件集成，购置集群时勾选 Impala 组件即可，如下图所示:

使用简介

1. 远程登录到创建好的集群

```
 ssh root@$public_ip
使用创建集群时输入的密码
```

2. 准备数据，可以参考数据准备。上传日志文件到HDFS中。

```
 hadoop dfs -get bos://datamart-gz/web-log-10k/accesslog-10k.log ./
hadoop dfs -put accesslog-10k.log /tmp/test
```

3. 在impala-shell中执行命令建表

   - 在shell中输入impala-shell

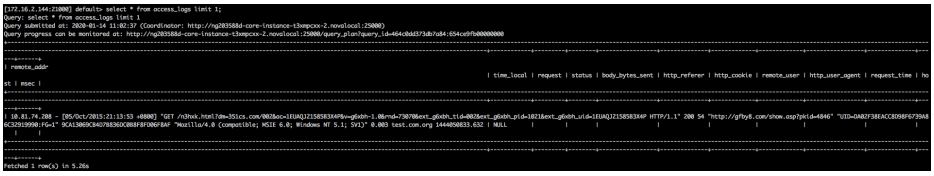
     说明：impala-shell默认连接到localhost上impalad的21000端口。BMR集群默认只在core、task节点上安装impalad服务。

     如果在master节点上执行impala-shell，需要使用-i <host:port>参数指定安装了impalad的host。更多可用参数可通过impala-shell -h查看。

   - 执行如下建表语句

```
 CREATE EXTERNAL TABLE `access_logs`(
`remote_addr` string COMMENT 'client IP',
`time_local` string COMMENT 'access time',
`request` string COMMENT 'request URL',
`status` string COMMENT 'HTTP status',
`body_bytes_sent` string COMMENT 'size of response body',
`http_referer` string COMMENT 'referer',
`http_cookie` string COMMENT 'cookies',
`remote_user` string COMMENT 'client name',
`http_user_agent` string COMMENT 'client browser info',
`request_time` string COMMENT 'consumed time of handling request',
`host` string COMMENT 'server host',
`msec` string COMMENT 'consumed time of writing logs')
COMMENT 'web access logs'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION '/tmp'
```

4. 建表成功后，可以使用SQL语句查询结果。如果使用提供的样例数据和建表语句，可以看到如下结果。



参考

1. Apache Impala Guide
2. Impala Home

Alluxio

Alluxio简介

1.什么是Alluxio
Alluxio 是世界上第一个面向基于云的数据分析和人工智能的开源的数据编排技术。 它为数据驱动型应用和存储系统构建了桥梁，将数据从存 储层移动到距离数据驱动型应用更近的位置从而能够更容易被访问。 这还使得应用程序能够通过一个公共接口连接到许多存储系统。 Alluxio内 存至上的层次化架构使得数据的访问速度能比现有方案快几个数量级。 在大数据生态系统中，Alluxio 位于数据驱动框架或应用（如 Apache Spark、Presto、Tensorflow、Apache HBase、Apache Hive 或 Apache Flink）和各种持久化存储系统（如 Amazon S3、Google Cloud Storage、OpenStack Swift、HDFS、GlusterFS、IBM Cleversafe、EMC ECS、Ceph、NFS 、Minio和Baidu BOS）之间。 Alluxio 统一了存储在这些不同存储系统中的数据，为其上层数据驱动型应用提供统一的客户端 API 和全局命名空间。 Alluxio 项目源自 UC Berkeley 的 AMPLab，在伯克利数据分析栈 (Berkeley Data Analytics Stack, BDAS) 中扮演数据访问层的角色。 它以 Apache License 2.0 协议的方式开源。 Alluxio 是发展最快的开源大数据项目之一，已经吸引了超过 300 个组织机构的1000多名贡献者参与到 Alluxio 的开发中，包括 阿里巴巴、 Alluxio、 百度、 CMU、 Google、 IBM、 Intel、 南京大学、 Red Hat、 腾讯、 UC Berkeley、 和 Yahoo。

2. 优势
   通过简化应用程序访问其数据的方式（无论数据是什么格式或位置），Alluxio 能够帮助克服从数据中提取信息所面临的困难。Alluxio 的优势包括：

- 内存速度 I/O：Alluxio 能够用作分布式共享缓存服务，这样与 Alluxio 通信的计算应用程序可以透明地缓存频繁访问的数据（尤其是从远程位置），以提供内存级 I/O 吞吐率。此外，Alluxio的层次化存储机制能够充分利用内存、固态硬盘或者磁盘，降低具有弹性扩张特性的数据驱动型应用的成本开销。
- 简化云存储和对象存储接入：与传统文件系统相比，云存储系统和对象存储系统使用不同的语义，这些语义对性能的影响也不同于传统文件系统。在云存储和

对象存储系统上进行常见的文件系统操作（如列出目录和重命名）通常会导致显著的性能开销。当访问云存储中的数据时，应用程序没有节点级数据本地性或跨应用程序缓存。将 Alluxio 与云存储或对象存储一起部署可以缓解这些问题，因为这样将从 Alluxio 中检索读取数据，而不是从底层云存储或对象存储中检索读取。

- 简化数据管理：Alluxio 提供对多数据源的单点访问。除了连接不同类型的数据源之外，Alluxio 还允许用户同时连接同一存储系统的不同版本，如多个版本的 HDFS，并且无需复杂的系统配置和管理。
- 应用程序部署简易：Alluxio 管理应用程序和文件或对象存储之间的通信，将应用程序的数据访问请求转换为底层存储接口的请求。Alluxio 与 Hadoop 生态系统兼容，现有的数据分析应用程序，如 Spark 和 MapReduce 程序，无需更改任何代码就能在 Alluxio 上运行。

3. 技术创新

- 全局命名空间：Alluxio 能够对多个独立存储系统提供单点访问，无论这些存储系统的物理位置在何处。这提供了所有数据源的统一视图和应用程序的标准接口。有关详细信息，请参阅统一命名空间文档。
- 智能多层级缓存：Alluxio 集群能够充当底层存储系统中数据的读写缓存。可配置自动优化数据放置策略，以实现跨内存和磁盘（SSD/HDD）的性能和可靠性。缓存对用户是透明的，使用缓冲来保持与持久存储的一致性。有关详细信息，请参阅 缓存功能文档。
- 服务器端 API 翻译转换：Alluxio支持工业界场景的API接口，例如HDFS API, S3 API, FUSE API, REST API。它能够透明地从标准客户端接口转换到任何存储接口。Alluxio 负责管理应用程序和文件或对象存储之间的通信，从而消除了对复杂系统进行配置和管理的需求。文件数据可以看起来像对象数据，反之亦然。

创建集群

1. 准备数据，请参考数据准备。
2. 准备百度智能云环境。
3. 登录控制台，选择“产品服务->MapReduce BMR”，点击“创建集群”，进入集群创建页，并做如下配置：
   - 设置集群名称
   - 设置管理员密码
   - 选择镜像版本“BMR 2.2.0(hadoop 3.1)”
   - 选择内置模板“hadoop”。
   - 选择组件时勾选alluxio、spark、hive、hbase等组件。
4. 请保持集群的其他默认配置不变，点击“完成”可在集群列表页可查看已创建的集群，当集群状态由“启动中”变为“运行中”时，集群创建成功。

使用Alluxio

1. 首先将你需要通过alluxio访问的bos路径挂载到集群内的alluxio上，具体操作如下：
   - 登录集群虚机
   - 在虚机上执行：alluxio fs mount /bos_path1 bos://testbucket/produce/，挂载所需的bos路径
   - 通过alluxio查看：alluxio fs ls /bos_path1
   - 通过alluxio对挂载路径的bos文件进行改动

     ```
     i)  alluxio fs rm  /bos_path1/a.txt
    ii) alluxio fs copyFromLocal    b.txt   /bos_path1/
     ```

2. MR作业支持alluxio schema

   ```
   hadoop jar /opt/bmr/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.1.1.jar wordcount   alluxio:///bos_path1/b.txt
   alluxio:///wordcount_output
   ```

3. spark支持alluxio schema

   ```
   val sc1 = sc.textFile("alluxio:///bos_path1/b.txt")
   val d1 = sc1.map(line => line + line)
   d1.saveAsTextFile("alluxio:///output")
   ```

4. hive支持alluxio schema

- 创建表

```
CREATE TABLE `default.table_alluxio_t`(
  `message` string)
PARTITIONED BY (
  `dt` string,
  `ip` string,
  `inode` string)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\u0001'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  'alluxio:///table_alluxio_t/'
TBLPROPERTIES (
  'transient_lastDdlTime'='1590749263');
```

- hive支持sql查询location是alluxio schema的表/partition

Alluxio官网手册

Alluxio概览

快速上手指南

Kudu

1.什么是Kudu

Kudu是一个用于结构化数据的开源存储引擎，它支持低延迟的随机访问，以及高效的分析存取模式. Kudu使用水平partition和副本技术来将数据分布式化，每个partition的副本用Raft协议同步，保证了低平均恢复时间和低长尾延迟. Kudu围绕着Hadoop生态圈设计，支持多种存取方式如Apache Impala，Apache Spark和MapReduce。

此外，Kudu还有更多优化的特点：

- OLAP 工作的快速处理。
- 与 MapReduce，Spark 和其他 Hadoop 生态系统组件集成。
- 与 Apache Impala（incubating）紧密集成，使其与 Apache Parquet 一起使用 HDFS 成为一个很好的可变的替代方案。
- 强大而灵活的一致性模型，允许您根据每个 per-request（请求选择）一致性要求，包括 strict-serializable（严格可序列化）一致性的选项。
- 针对同时运行顺序和随机工作负载的情况性能很好。
- High availability（高可用性）。Tablet server *和* Master 使用 Raft Consensus Algorithm 来保证节点的高可用，确保只要有一半以上的副本可用，该 tablet *便可用于读写。例如，如果* 3 个副本中有 2 *个或* 5 *个副本中的* 3 *个可用，则该* tablet *可用。即使在* leader tablet *出现故障的情况下，读取功能也可以通过* read-only（只读的）follower tablets 来进行服务。
- 结构化数据模型。

2.Kudu常见的几个应用场景

- 实时更新的应用。刚刚到达的数据就马上要被终端用户使用访问到。
- 时间序列相关的应用，需要同时支持：
  - 根据海量历史数据查询。
  - 必须非常快地返回关于单个实体的细粒度查询。
- 实时预测模型的应用，支持根据所有历史数据周期地更新模型。
- 有关这些和其他方案的更多信息，请参阅 Example Use Cases。

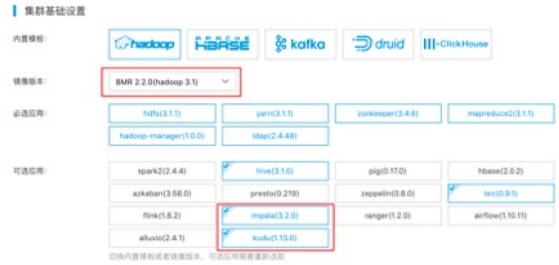
3.Kudu使用

准备百度智能云环境。

登录控制台，选择“产品服务->MapReduce BMR”，点击“创建集群”，进入集群创建页，并做如下配置：

1. 创建一个bmr2.2.0镜像的集群，选择kudu+impala

选择了kudu，则集群必须有2个master节点和至少3个core节点。（由于task节点可以进行弹性伸缩，故而默认不启动tserver进行kudu数据存储。）



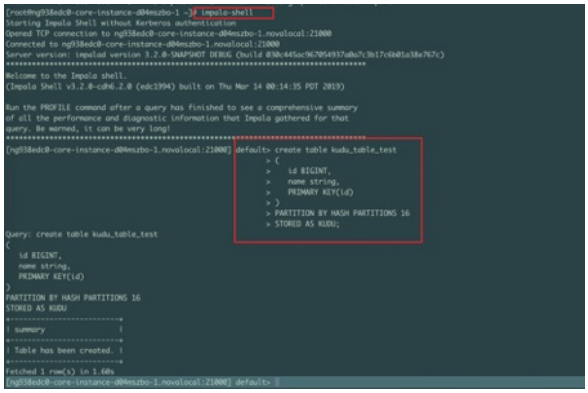
2.登录控制台

创建完成之后，登录master节点，ssh到impalad所在的host，通过impala-shell进入impala控制台

2.1 通过impala创建kudu表

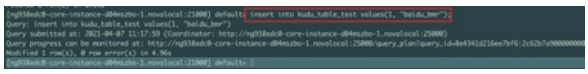
```
 create table kudu_table_test
(
  id BIGINT,
  name string,
  PRIMARY KEY(id)
)
PARTITION BY HASH PARTITIONS 16
STORED AS KUDU;
```

具体如下图所示:



2.2 insert

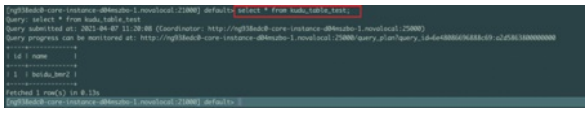
```
insert into kudu_table_test values(1, "baidu_bmr");
```



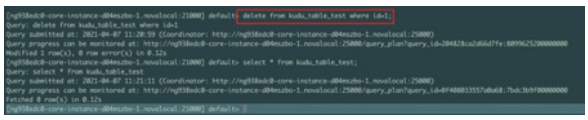
2.3 update

```
update kudu_table_test set name='baidu_bmr2' where id=1;
```

2.4 select `select * from kudu_table_test;`



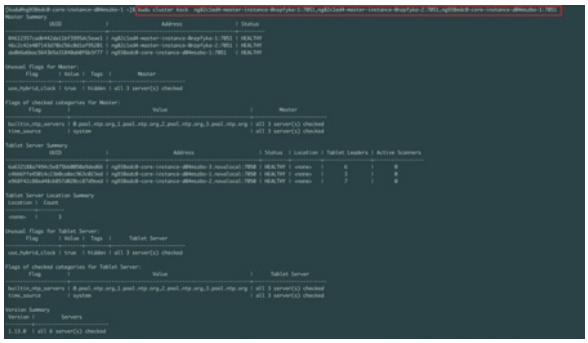
2.5 delete `delete from kudu_table_test where id=1;`



3、kudu cli使用

同时可以在集群上通过kudu cli对kudu进行管理。 **3.1 查看kudu集群状态** `kudu cluster ksck  ng82c1ed4-master-instance-0nzpfyka-1:7051,ng82c1ed4-master-instance-0nzpfyka-2:7051,ng938edc0-core-instance-d04mszbo-1:7051`

需要指定三个kudu master的地址



3.2 rebalance

```
kudu cluster rebalance ng82c1ed4-master-instance-0nzpfyka-1:7051,ng82c1ed4-master-instance-0nzpfyka-2:7051,ng938edc0-core-instance-d04mszbo-1:7051
```

当kudu集群上tserver数据分配不合理时，可以手动通过kudu cli进行数据rebalance

3.3 其他kudu cli可以查询kudu官方文档 kudu官方文档

Ooize

什么是Ooize

Oozie是一个用于管理Apache Hadoop作业的工作流调度程序系统。

- Oozie Workflow job是由多个Action组成的有向无环图（DAG）。
- Oozie Coordinator job是根据时间（频率）和数据可用性触发的可重复执行的Oozie Workflow job（简单讲就是根据时间或数据条件，规划workflow的执行）。
- Oozie与Hadoop技术栈的项目集成，支持多种类型的Hadoop作业（例如Java map-reduce，Streaming map-reduce，Pig，Hive，Sqoop和Distcp，Spark）以及系统特定的工作（例如Java程序和shell脚本）。
- Oozie是一个可水平扩展，可靠和可使用扩展插件（scalable, reliable and extensible）的系统。

Ooize常见的几个应用场景

1、统一调度hadoop系统中常见的mr任务启动、 hdfs操作、 shell调度、 hive操作等。

2、使得复杂的依赖关系、时间触发、事件触发使用xml语言进行表达，开发效率提高。

3、一组任务使用一个DAG来表示，使用图形表达流程逻辑更加清晰。

4、支持很多种任务调度，能完成大部分hadoop任务处理。

5、程序定义支持EL常量和函数，表达更加丰富。

Oozie的功能模块介绍

模块

- Workflow：顺序执行流程节点，支持fork（分支多个节点），join（合并多个节点为一个）。
- Coordinator：定时触发workflow。
- Bundle：绑定多个Coordinator。

Workflow常用节点

- 控制流节点（Control Flow Nodes）：控制流节点一般都是定义在工作流开始或者结束的位置，比如start,end,kill等。以及提供工作流的执行路径机制，如decision，fork，join等。
- 动作节点（Action Nodes）：负责执行具体动作的节点，比如：拷贝文件，执行某个Shell脚本等等。

Ooize使用

准备百度智能云环境。

登录控制台，选择“产品服务->MapReduce BMR”，点击“创建集群”，进入集群创建页，并做如下配置：

1. 创建一个bmr2.3.0镜像的集群，选择hue+oozie 用户可以根据需求选择对应的服务，选择完成后填写集群的基础信息和实例规格，点击购买即可，建议选择新建高可用集群。

2.使用Ooize调度作业

****1.命令行调度****

1.1本地创建任务文件夹(比如 hive)，编写 hive/job.properties

```
vim hive/job.properties
```

将配置中host改为所在集群机器host

```
#
# 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.
#
nameNode=hdfs://master-22c8a0f:8020
resourceManager=master-22c8a0f:8050
queueName=default
jdbcURL=jdbc:hive2://master-22c8a0f:10000/default
examplesRoot=examples
oozie.use.system.libpath=true
oozie.wf.application.path=${nameNode}/user/${user.name}/${examplesRoot}/apps/hive2
~
~
~
~
```

```
 nameNode=hdfs://master-22c8a0f:8020
resourceManager=master-22c8a0f:8050
queueName=default
jdbcURL=jdbc:hive2://master-22c8a0f:10000/default
examplesRoot=examples
oozie.use.system.libpath=true
oozie.wf.application.path=${nameNode}/user/${user.name}/${examplesRoot}/apps/hive
```

1.2 编写 workflow.xml

script 填写sql文件名称

```
-->
<workflow-app xmlns="uri:oozie:workflow:1.0" name="hive2-wf">
    <start to="hive2-node"/>

    <action name="hive2-node">
        <hive2 xmlns="uri:oozie:hive2-action:1.0">
            <resource-manager>${resourceManager}</resource-manager>
            <name-node>${nameNode}</name-node>
            <prepare>
                <delete path="${nameNode}/user/${wf:user()}/${examplesRoot}/output-data/hive2"/>
                <mkdir path="${nameNode}/user/${wf:user()}/${examplesRoot}/output-data"/>
            </prepare>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>${queueName}</value>
                </property>
            </configuration>
            <jdbc-url>${jdbcURL}</jdbc-url>
            <script>show.q</script>
            <param>INPUT=/user/${wf:user()}/${examplesRoot}/input-data/table</param>
            <param>OUTPUT=/user/${wf:user()}/${examplesRoot}/output-data/hive2</param>
        </hive2>
        <ok to="end"/>
        <error to="fail"/>
    </action>

    <kill name="fail">
        <message>Hive2 (Beeline) action failed, error message[${wf:errorMessage(wf:lastErrorNode())}]</message>
```

1.3 编写任务脚本

```
vim hive/show.q
create database hivetest;
create table hivetest.student(id int, name string, age int) row format delimited fields terminated by ',' STORED AS TEXTFILE;
insert into hivetest.student values(1, 'name1', 12);
```

1.4上传文件夹到 job.properties 中 oozie.wf.application.path 指定的 hdfs 路径

```
hdfs dfs -put -f hive/ /user/hdfs/examples/apps/
```

1.5执行命令提交 oozie 任务

```
/opt/bmr/oozie/bin/oozie job -oozie http://{oozie-server eip}:11000/oozie -config hive/job.properties  -run
```

提交后会得到 oozie jobId

```
[hdfs@master-22c8a0f apps]$ /opt/bmr/oozie/bin/oozie job -oozie http://127.0.0.1:11000/oozie -config hive2/job.properties  -run
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/bmr/oozie/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/bmr/oozie/lib/slf4j-log4j12-1.6.6.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only errors to the console. Set system property 'log4j2.debug' to show Log4j2 internal in
itialization logging.
job: 0000079-211118162829239-bmr-oozie-W
[hdfs@master-22c8a0f apps]$ /opt/bmr/oozie/bin/oozie job -oozie http://127.0.0.1:11000/oozie -info 0000079-211118162829239-bmr-oozie-W
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/bmr/oozie/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/bmr/oozie/lib/slf4j-log4j12-1.6.6.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only errors to the console. Set system property 'log4j2.debug' to show Log4j2 internal in
itialization logging.
Job ID : 0000079-211118162829239-bmr-oozie-W
```

1.6使用 oozie jobId 可以查看任务运行详情

```
/opt/bmr/oozie/bin/oozie job -oozie http://{oozie-server eip}:11000/oozie -info 0000079-211118162829239-bmr-oozie-W
```

```
[hdfs@master-22c8a0f apps]$ /opt/bmr/oozie/bin/oozie job -oozie http://127.0.0.1:11000/oozie -info 0000079-211118162829239-bmr-oozie-W
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/bmr/oozie/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/bmr/oozie/lib/slf4j-log4j12-1.6.6.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only errors to the console. Set system property 'log4j2.debug' to show Log4j2 internal in
itialization logging.
Job ID : 0000079-211118162829239-bmr-oozie-W
------------------------------------------------------------------------------------------------------------------------------------
Workflow Name : hive2-wf
App Path      : hdfs://master-22c8a0f:8020/user/hdfs/examples/apps/hive2
Status        : SUCCEEDED
Run           : 0
User          : hdfs
Group         : -
Created       : 2021-11-22 07:11 GMT
Started       : 2021-11-22 07:11 GMT
Last Modified : 2021-11-22 07:12 GMT
Ended         : 2021-11-22 07:12 GMT
CoordAction ID: -

Actions
------------------------------------------------------------------------------------------------------------------------------------
ID                                                                            Status    Ext ID                 Ext Status Err Code
------------------------------------------------------------------------------------------------------------------------------------
0000079-211118162829239-bmr-oozie-W@:start:                                   OK        -                      OK         -
------------------------------------------------------------------------------------------------------------------------------------
0000079-211118162829239-bmr-oozie-W@hive2-node                                OK        application_1637224014323_0093SUCCEEDED  -
------------------------------------------------------------------------------------------------------------------------------------
0000079-211118162829239-bmr-oozie-W@end                                       OK        -                      OK         -
------------------------------------------------------------------------------------------------------------------------------------
```

- [更多详细样例查看 apache oozie 官网]

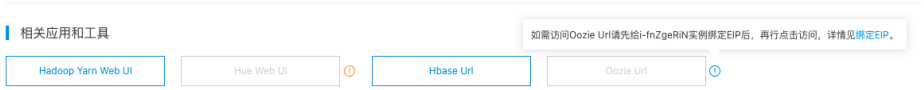
2.hue调度

使用 hue 交互式开发 oozie 任务，请参考Hue官方开发示例。

3.访问Ooize URL

1.登录控制台，选择“产品服务->MapReduce BMR”，点击已创建的集群，进入该集群详情页。

2.在“相关应用和工具”栏中可以查看“Ooize Url”。



Ooize Url 默认不可点击，需要在给对应的master实例绑定EIP后并且开通对应的端口后才可以访问。 具体需要绑定EIP的实例ID可以将鼠标指针上浮在Ooize Url 叹号处查看。绑定EIP见绑定EIP。若无EIP，请先于产品【网络——弹性公网IP EIP】创建EIP。详见：创建EIP实例。

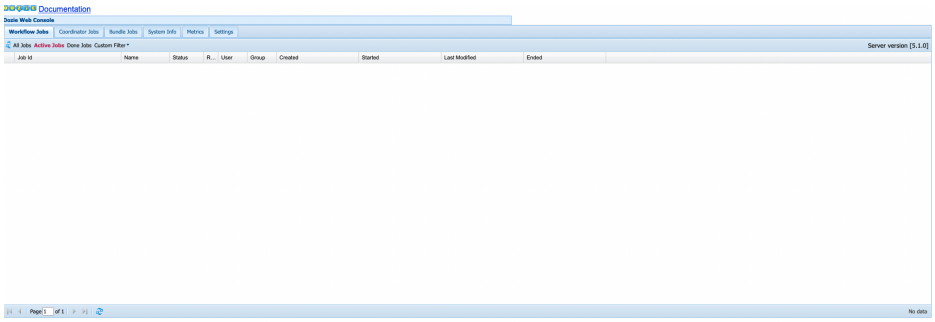
3.给BMR安全组添加11000端口访问规则。 在给对应的节点绑定eip后，在集群详情-网络处可以查看对应BMR安全组ID。

复制对应的BMR安全组ID，进入VPC-安全组中，找到对应的BMR安全组。

点击添加规则，选择HTTP，规则类型为IPV4,协议为tcp，端口范围为11000，source建议填写为需要访问BMR集群的IP或者IP段，不建议为all，本示例为方便演示选择all（0.0.0.0/0）。

添加完成后如下图所示：

4.完成上述工作后，在BMR集群详情页面中点击Ooize Url即可访问。



ClickHouse

快速入门

访问模式

通过BLB访问ClickHouse集群

前提条件

用户已创建BMR Clickhouse集群。

操作步骤

具体配置流程如下，参考https://cloud.baidu.com/doc/BLB/s/cjwvxnr91

1. 创建BLB实例，可根据业务负载购买对应类型和规格的BLB实例
2. 配置clickhouse实例
3. 添加监听端口号

通过JDBC访问ClickHouse集群

获取主机的 IP 地址，步骤如下：

1. 登录 MapReduce 控制台，进入集群列表
2. 单击集群名称，进入集群详情，在实例信息部分展开实例组。在公网IP/内网IP可以查看ClickHouse集群的IP地址。
3. 配置JDBC以访问ClickHouse集群。

基础操作

表操作

本地表

- 创建本地表：

```
CREATE TABLE `check_local` (
 `Id` UInt16,
 `Name` String,
 `CreateDate` Date)
ENGINE = MergeTree()
PARTITION BY CreateDate
ORDER BY Id;
```

- 本地表插入数据：

```
 insert into check_local (Id, Name, CreateDate) values (1, 'aa', '2020-01-01');
```

- 本地表查询数据：

```
select * from check_local;
```

分布式表

- 在默认集群上批量建立本地表：

```
CREATE TABLE `check_local2` ON CLUSTER default_cluster (
`Id` UInt16,
`Name` String,
`CreateDate` Date)
ENGINE = MergeTree()
PARTITION BY CreateDate
ORDER BY Id;
```

备注：ClickHouse集群支持分布式DDL语句，即在DDL语句上加上ON CLUSTER <cluster_neme>的语法，使得该DDL语句执行一次便可在所有实例上创建该表。默认集群名字为default_cluster。

- 创建分布式表：

```
CREATE TABLE dis_check_all ON CLUSTER default_cluster
AS check_local2
ENGINE = Distributed(default_cluster, default, check_local2, rand());
```

- 分布式表插入语法同本地表：

```
insert into dis_check_all (Id, Name, CreateDate)values (1,'aa','2020-01-01');
```

或者

```
insert into dis_check_all values (1,'aa','2020-01-01');
```

- 分布式表查询：

```
select * from dis_check_all;
```

副本表

注意，如果创建的不是副本表（replicated table），数据完整性，在宕机、故障的时候无法保证。

请在不添加参数的情况下创建表。例如：

```
ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/table_name', '{replica}', ver)
```

```
ENGINE = ReplicatedMergeTree
```

前缀`Replicated`添加到表引擎名称。例如：`ReplicatedMergeTree`。

- *MergeTree参数

zoo_path：ClickHouse Keeper中表的路径。

replica_name：ClickHouse Keeper中的副本名称。

other_parameters：用于创建复制版本的引擎的参数，例如`ReplacingMergeTree`中的版本。

例如：

```
CREATE TABLE table_name
(
    EventDate DateTime,
    CounterID UInt32,
    UserID UInt32,
    ver UInt16
)
ENGINE = ReplicatedReplacingMergeTree('/clickhouse/tables/{layer}-{shard}/table_name', '{replica}', ver)
PARTITION BY toYYYYMM(EventDate)
ORDER BY (CounterID, EventDate, intHash32(UserID))
SAMPLE BY intHash32(UserID);
```

弃用语法示例：

```
 CREATE TABLE table_name
(
    EventDate DateTime,
    CounterID UInt32,
    UserID UInt32
) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/table_name', '{replica}', EventDate, intHash32(UserID), (CounterID, EventDate,
intHash32(UserID), EventTime), 8192);
```

如示例所示，这些参数可以包含大括号内的替换值。替换值取自配置文件的宏部分。

例如：

```
 <macros>
    <shard>02</shard>
    <replica>example05-02-1</replica>
</macros>
```

ClickHouse Keeper 中表的路径对于每个复制表应该是唯一的。不同分片上的表应该有不同的路径。在这种情况下，路径由以下部分组成：

/clickhouse/tables/是常用前缀。{shard}将扩展为分片标识符。table_name是 ClickHouse Keeper 中表的节点名称。最好将其与表名相同。它是明确定义的，因为与表名不同，它在 RENAME 查询后不会改变。 提示：您也可以在前面添加数据库名称table_name。例如db_name.table_name

可以使用两个内置的替换{database}和{table}，它们分别展开为表名和数据库名（除非在宏部分定义了这些宏）。因此，动物园管理员路径可以指定为“/clickhouse/tables/{shard}/{database}/{table}”。使用这些内置替换时，请小心表重命名。ClickHouse Keeper中的路径无法更改，当重命名表时，宏将展开到其他路径，表将引用ClickHouse Keeper不存在的路径，并将进入只读模式。

副本名称标识同一表的不同副本。您可以使用服务器名称，如示例所示。名称只需要在每个分片中是唯一的。

您可以显式定义参数，而不是使用替换。这可能便于测试和配置小型集群。但是，在这种情况下，您不能使用分布式DDL查询（ON CLUSTER）。

在处理大型集群时，我们建议使用替换，因为它们可以降低出错的概率。

您可以在服务器配置文件中为副本表引擎指定默认参数。例如：

```
 <default_replica_path>/clickhouse/tables/{shard}/{database}/{table}</default_replica_path>
<default_replica_name>{replica}</default_replica_name>
```

在这种情况下，可以在创建表时省略参数：

```
 CREATE TABLE table_name (
    x UInt32
) ENGINE = ReplicatedMergeTree
ORDER BY x;
```

或者：

```
 CREATE TABLE table_name (
    x UInt32
) ENGINE = ReplicatedMergeTree('/clickhouse/tables/{shard}/{database}/table_name', '{replica}')
ORDER BY x;
```

对每个副本运行CREATE TABLE查询。此查询创建一个新的复制表，或向现有表添加一个新副本。

如果在表已包含其他副本上的一些数据后添加新副本，则在运行查询后，数据将从其他副本复制到新副本。换句话说，新的复制品会与其他复制品同步。

要删除副本，请运行DROP TABLE。但是，只会删除一个副本，即位于运行查询的服务器上的副本。

- 数据丢失后恢复

如果其中一台服务器的所有数据和元数据都消失了，请按照以下步骤进行恢复：

1. 在服务器上安装 ClickHouse。如果使用替换，请在包含分片标识符和副本的配置文件中正确定义替换。
2. 如果您有未复制的表必须在服务器上手动复制，请从副本（在目录中/var/lib/clickhouse/data/db_name/table_name/）复制其数据。
3. /var/lib/clickhouse/metadata/从副本中复制位于的表定义。如果在表定义中明确定义了分片或副本标识符，请更正它，使其与此副本相对应。（或者，启动服务器并ATTACH TABLE执行 中的 .sql 文件应包含的所有查询/var/lib/clickhouse/metadata/。）
4. 要开始恢复，请创建具有任何内容的 ClickHouse Keeper 节点/path_to_table/replica_name/flags/force_restore_data，或运行命令来恢复所有复制的表：sudo -u clickhouse touch /var/lib/clickhouse/flags/force_restore_data 然后启动服务器（如果已在运行，则重新启动）。数据将从副本中下载。
5. 另一种恢复选项是从 ClickHouse Keeper () 中删除有关丢失副本的信息，然后按照“创建复制表/path_to_table/replica_name”中所述再次创建副本。

恢复期间网络带宽不受限制。如果您要同时恢复多个副本，请注意这一点。

客户端登录

操作步骤

1. 远程登录到创建好的集群

```
ssh root@$public_ip
```

2. 登录ClickHouse客户端

```
 su - clickhouse
clickhouse-client -m -u admin --password 集群密码
```

表一 clickhouse-client常用参数

参数	描述
-h	主机名
-d	数据库名
-m	客户端支持多行SQL输入以分号结尾，不指定该参数默认以回车作为SQL结尾
-u	账户，默认admin
--password	默认创建集群时的密码

其他client参数可以执行以下命令进行查看：

```
clickhouse-client --help
```

创建ClickHouse集群

操作步骤

1. 登录百度智能云控制台，选择产品服务>百度MapReduce BMR，进入集群管理界面。

2. 单击创建集群，进入集群创建页。购置集群时勾选 ClickHouse 组件即可。详见创建集群。

数据迁移同步

从本地数据导入

Parquet格式

Parquet格式

Parquet 是一种高效的文件格式，以列为单位存储数据。ClickHouse 提供对 Parquet 文件的读取和写入支持。

从 Parquet 导入

1. 在加载数据之前，我们可以使用file()函数来探索示例 parquet 文件结构：

```
DESCRIBE TABLE file('data.parquet', Parquet);
```

2. 使用Parquet作为第二个参数，因此 ClickHouse 知道文件格式。这将打印具有以下类型的列：

```
┌─name─┬─type──────────────┬─default_type─┬─default_expression─┬─comment─┬─codec_expression─┬─ttl_expression─┐
│ path │ Nullable(String)  │              │                    │         │                  │                │
│ date │ Nullable(String)  │              │                    │         │                  │                │
│ hits │ Nullable(Int64)   │              │                    │         │                  │                │
└──────┴───────────────────┴──────────────┴────────────────────┴─────────┴──────────────────┴────────────────┘
```

3. 还可以在实际导入数据之前利用SQL的所有功能搜索文件：

```
 SELECT *
FROM file('data.parquet', Parquet)
LIMIT 3;
```

```
┌─path──────────────────────────────────────────────────┬──date─┬─hits─┐
│ Akiba_Hebrew_Academy      │ 2017-08-01 │  241 │
│ Aegithina_tiphia          │ 2018-02-01 │   34 │
│ 1971-72_Utah_Stars_season │ 2016-10-01 │    1 │
└─────────────────────────────────────────────────┴────────┴──────┘
```

导入到现有表

1. 创建一个表，将 Parquet 数据导入其中：

```
CREATE TABLE sometable
(
    `path` String,
    `date` Date,
    `hits` UInt32
)
ENGINE = MergeTree
ORDER BY (date, path);
```

2. 使用以下FROM INFILE句子导入数据：

```
INSERT INTO sometable
FROM INFILE 'data.parquet' FORMAT Parquet;

SELECT *
FROM sometable
LIMIT 5;
```

```
┌─path──────────────────────────────────────────────────┬──date─┬─hits─┐
│ 1988_in_philosophy          │ 2015-05-01 │   70 │
│ 2004_Green_Bay_Packers_season │ 2015-05-01 │  970 │
│ 24_hours_of_lemans          │ 2015-05-01 │   37 │
│ 25604_Karlin                │ 2015-05-01 │   20 │
│ ASCII_ART                   │ 2015-05-01 │    9 │
└─────────────────────────────────────────────────┴────────┴──────┘
```

请注意ClickHouse如何自动将Parquet字符串（在date列中）转换为Date类型。这是因为 ClickHouse 根据目标表中的类型自动进行类型转换。

将本地文件插入到远程服务器

如果要将本地 Parquet 文件插入到远程 ClickHouse 服务器，可以通过将文件内容导入来实现clickhouse-client，如下所示：

```
clickhouse client -q "INSERT INTO sometable FORMAT Parquet" < data.parquet
```

从Parquet文件创建新表

1. 从Parquet文件创建新表：

```
CREATE TABLE imported_from_parquet
ENGINE = MergeTree
ORDER BY tuple() AS
SELECT *
FROM file('data.parquet', Parquet)
```

2. 自动从给定的Parquet文件创建和填充表格：

```
DESCRIBE TABLE imported_from_parquet;
```

```
┌─name─┬─type─────────────┬─default_type─┬─default_expression─┬─comment─┬─codec_expression─┬─ttl_expression─┐
│ path │ Nullable(String) │              │                    │         │                  │                │
│ date │ Nullable(String) │              │                    │         │                  │                │
│ hits │ Nullable(Int64)  │              │                    │         │                  │                │
└──────┴──────────────────┴──────────────┴────────────────────┴─────────┴──────────────────┴────────────────┘
```

默认情况下，ClickHouse 对列名、类型和值的要求很严格。但有时，我们可以在导入过程中跳过不存在的列或不支持的值。

导出为Parquet格式

1. 要将任何表或查询结果导出到Parquet文件，可以使用以下INTO OUTFILE句子：

```
 SELECT *
FROM sometable
INTO OUTFILE 'export.parquet'
FORMAT Parquet
```

ClickHouse和Parquet数据类型

1. ClickHouse 和 Parquet 数据类型大部分相同，但仍有一点不同。例如，ClickHouse 将DateTime类型导出为 Parquets' int64。如果将其导入回 ClickHouse，将看到：

```
 SELECT * FROM file('time.parquet', Parquet);
```

```
 ┌─n─┬──────time─┐
│ 0 │ 1673622611 │
│ 1 │ 1673622610 │
│ 2 │ 1673622609 │
│ 3 │ 1673622608 │
│ 4 │ 1673622607 │
└───┴────────────┘
```

2.在这种情况下可以使用类型转换：

```
 SELECT
  n,
  toDateTime(time)                <--- int to time
FROM file('time.parquet', Parquet);
```

```
 ┌─n─┬─toDateTime(time)────┐
│ 0 │ 2023-01-13 15:10:11 │
│ 1 │ 2023-01-13 15:10:10 │
│ 2 │ 2023-01-13 15:10:09 │
│ 3 │ 2023-01-13 15:10:08 │
│ 4 │ 2023-01-13 15:10:07 │
└───┴─────────────────────┘
```

CSV and TSV

CSV and TSV

ClickHouse 支持从 CSV 导入数据和将数据导出到 CSV。由于 CSV 文件可能具有不同的格式细节，包括标题行、自定义分隔符和转义符号，ClickHouse 提供了格式和设置来有效解决每种情况。

从CSV文件导入数据

1. 在导入数据之前，创建一个具有相关结构的表：

```
 CREATE TABLE sometable
(
  `path` String,
  `month` Date,
  `hits` UInt32
)
ENGINE = MergeTree
ORDER BY tuple(month, path)
```

2. 要将数据从CSV 文件导入到sometable表中，我们可以将文件直接传输到 clickhouse-client：

```
 clickhouse-client -q "INSERT INTO sometable FORMAT CSV" < data_small.csv
```

3. 注意，使用FORMAT CSV让 ClickHouse知道我们正在提取 CSV 格式的数据。或者，我们可以使用FROM INFILE句子从本地文件加载数据。这里，我们使用了FORMAT CSV句子，以便 ClickHouse 能够理解文件格式。我们还可以使用url()函数直接从 URL 加载数据，或者使用s3()函数从 S3 文件加载数据。

```
 INSERT INTO sometable
FROM INFILE 'data_small.csv'
FORMAT CSV
```

带标题的 CSV 文件

1. 假设CSV文件包含标题：

```
head data-small-headers.csv
```

```
"path","month","hits"
"Akiba_Hebrew_Academy","2017-08-01",241
"Aegithina_tiphia","2018-02-01",34
```

2. 要从此文件导入数据，我们可以使用CSVWithNames格式：

```
clickhouse-client -q "INSERT INTO sometable FORMAT CSVWithNames" < data_small_headers.csv
```

在这种情况下，ClickHouse 在从文件导入数据时会跳过第一行。

具有自定义分隔符的 CSV 文件

如果CSV文件使用逗号以外的分隔符，可以使用format_csv_delimiter选项来设置相关符号：

```
SET format_csv_delimiter = ';'
```

现在，当从CSV文件导入时，;符号将用作分隔符而不是逗号。

跳过 CSV 文件中的行

有时，我们可能会在从 CSV 文件导入数据时跳过一定数量的行。这可以使用input_format_csv_skip_first_lines选项完成：

```
SET input_format_csv_skip_first_lines = 10
```

在这种情况下，将跳过 CSV 文件的前十行：

```
SELECT count(*) FROM file('data-small.csv', CSV)
```

```
┌─count()─┐
│     990 │
└─────────┘
```

该文件有1000行，但由于要求跳过前 10 行，因此ClickHouse仅加载了990 行。

处理CSV文件中的NULL值

根据生成文件的应用程序，空值可以采用不同的编码方式。默认情况下，ClickHouse`\N`在 CSV 中将其用作空值。但我们可以使用format_csv_null_representation选项进行更改。

1. 假设有以下 CSV 文件：

```
> cat nulls.csv
Donald,90
Joe,Nothing
Nothing,70
```

2.如果我们从此文件加载数据，ClickHouse 将把它视为`Nothing`一个字符串（这是正确的）：

```
SELECT * FROM file('nulls.csv')
```

```
┌─c1──────┬─c2──────┐
│ Donald  │ 90      │
│ Joe     │ Nothing │
│ Nothing │ 70      │
└─────────┴─────────┘
```

3. 如果希望ClickHouse将其视为`Nothing，NULL`可以使用以下选项定义：

```
SET format_csv_null_representation = 'Nothing'
```

现在已经知道`NULL`它位于期望的位置了：

```
SELECT * FROM file('nulls.csv')
```

```
┌─c1─────┬─c2─┐
│ Donald │ 90 │
│ Joe    │ NULL │
│ NULL   │ 70 │
└────────┴────┘
```

TSV（制表符分隔）文件

制表符分隔数据格式被广泛用作数据交换格式。要将数据从TSV 文件加载到ClickHouse，请使用TabSeparated格式：

```
clickhouse-client -q "INSERT INTO sometable FORMAT TabSeparated" < data_small.tsv
```

还有TabSeparatedWithNames格式，允许处理带有标题的TSV文件。并且，与CSV一样，我们可以使用input_format_tsv_skip_first_lines选项跳过前X行。

原始TSV

有时，TSV文件保存时没有转义制表符和换行符。我们应该使用TabSeparatedRaw来处理此类文件。

导出至CSV

1. 前面示例中的任何格式也可用于导出数据。要将表（或查询）中的数据导出为 CSV 格式，我们使用相同的FORMAT句子。

```
SELECT *
FROM sometable
LIMIT 5
FORMAT CSV
```

```
"Akiba_Hebrew_Academy","2017-08-01",241
"Aegithina_tiphia","2018-02-01",34
"1971-72_Utah_Stars_season","2016-10-01",1
"2015_UEFA_European_Under-21_Championship_qualification_Group_8","2015-12-01",73
"2016_Greater_Western_Sydney_Giants_season","2017-05-01",86
```

2. 要向CSV文件添加标题，需要使用CSVWithNames格式：

```
SELECT *
FROM sometable
LIMIT 5
FORMAT CSVWithNames
```

```
"path","month","hits"
"Akiba_Hebrew_Academy","2017-08-01",241
"Aegithina_tiphia","2018-02-01",34
"1971-72_Utah_Stars_season","2016-10-01",1
"2015_UEFA_European_Under-21_Championship_qualification_Group_8","2015-12-01",73
"2016_Greater_Western_Sydney_Giants_season","2017-05-01",86
```

将导出的数据保存到CSV文件

要将导出的数据保存到文件，可以使用INTO…OUTFILE句子：

```
SELECT *
FROM sometable
INTO OUTFILE 'out.csv'
FORMAT CSVWithNames
```

```
36838935 rows in set. Elapsed: 1.304 sec. Processed 36.84 million rows, 1.42 GB (28.24 million rows/s., 1.09 GB/s.)
```

使用自定义分隔符导出CSV

1. 如果想要使用逗号以外的分隔符，我们可以使用format_csv_delimiter设置选项：

```
SET format_csv_delimiter = '|'
```

2. 现在ClickHouse将使用|作为 CSV 格式的分隔符：

```
 SELECT *
FROM sometable
LIMIT 5
FORMAT CSV
```

```
 "Akiba_Hebrew_Academy"|"2017-08-01"|241
"Aegithina_tiphia"|"2018-02-01"|34
"1971-72_Utah_Stars_season"|"2016-10-01"|1
"2015_UEFA_European_Under-21_Championship_qualification_Group_8"|"2015-12-01"|73
"2016_Greater_Western_Sydney_Giants_season"|"2017-05-01"|86
```

导出适用于Windows的CSV

如果希望CSV文件在Windows环境中正常工作，我们应该考虑启用output_format_csv_crlf_end_of_line选项。这将用作`\r\n`换行符而不是`\n`：

```
SET output_format_csv_crlf_end_of_line = 1;
```

CSV文件的架构推断

在许多情况下，可能会使用未知的CSV文件，因此我们必须探索要使用哪些类型的列。默认情况下，Clickhouse 将尝试根据对给定CSV文件的分析来猜测数据格式。这称为“模式推断”。可以使用语句与file()`DESCRIBE`函数配对来探索检测到的数据类型：

```
DESCRIBE file('data-small.csv', CSV)
```

```
┌─name─┬─type─────────────┬─default_type─┬─default_expression─┬─comment─┬─codec_expression─┬─ttl_expression─┐
│ c1   │ Nullable(String) │              │                    │         │                  │                │
│ c2   │ Nullable(Date)   │              │                    │         │                  │                │
│ c3   │ Nullable(Int64)  │              │                    │         │                  │                │
└──────┴──────────────────┴──────────────┴────────────────────┴─────────┴──────────────────┴────────────────┘
```

ClickHouse可以有效地猜测CSV文件的列类型。如果不想让ClickHouse猜测，可以使用以下选项禁用此功能：

```
SET input_format_csv_use_best_effort_in_schema_inference = 0
```

在这种情况下，所有列类型都将被视为`String`。

导出和导入具有明确列类型的CSV

ClickHouse还允许在使用CSVWithNamesAndTypes（和其他 *WithNames 格式系列）导出数据时明确设置列类型：

```
 SELECT *
FROM sometable
LIMIT 5
FORMAT CSVWithNamesAndTypes
```

```
 "path","month","hits"
"String","Date","UInt32"
"Akiba_Hebrew_Academy","2017-08-01",241
"Aegithina_tiphia","2018-02-01",34
"1971-72_Utah_Stars_season","2016-10-01",1
"2015_UEFA_European_Under-21_Championship_qualification_Group_8","2015-12-01",73
"2016_Greater_Western_Sydney_Giants_season","2017-05-01",86
```

此格式将包含两个标题行 - 一个包含列名，另一个包含列类型。这将允许 ClickHouse（和其他应用程序）在从此类文件加载数据时识别列类型：

```
DESCRIBE file('data_csv_types.csv', CSVWithNamesAndTypes)
```

```
┌─name──┬─type───┬─default_type─┬─default_expression─┬─comment─┬─codec_expression─┬─ttl_expression─┐
│ path  │ String │              │                    │         │                  │                │
│ month │ Date   │              │                    │         │                  │                │
│ hits  │ UInt32 │              │                    │         │                  │                │
└───────┴────────┴──────────────┴────────────────────┴─────────┴──────────────────┴────────────────┘
```

现在 ClickHouse 根据（第二）标题行来识别列类型。

自定义定界符、分隔符和转义规则

在复杂的情况下，文本数据可以以高度自定义的方式格式化，但仍具有结构。ClickHouse 针对这种情况具有特殊的CustomSeparated格式，允许设置自定义转义规则、分隔符、行分隔符和开始/结束符号。

假设文件中有以下数据：

```
row('Akiba_Hebrew_Academy';'2017-08-01';241),row('Aegithina_tiphia';'2018-02-01';34),...
```

可以看到，各行用包裹row()，各行用分隔;。在这种情况下，我们可以使用以下设置从此文件中读取数据：

```
SET format_custom_row_before_delimiter = 'row(';
SET format_custom_row_after_delimiter = ')';
SET format_custom_field_delimiter = ';';
SET format_custom_row_between_delimiter = ',';
SET format_custom_escaping_rule = 'Quoted';
```

可以从自定义格式的文件中加载数据：

```
SELECT *
FROM file('data_small_custom.txt', CustomSeparated)
LIMIT 3
```

```
┌─c1────────────────────────────────────────────────┬─────────c2─┬──c3─┐
│ Akiba_Hebrew_Academy                              │ 2017-08-01 │ 241 │
│ Aegithina_tiphia                                  │ 2018-02-01 │  34 │
│ 1971-72_Utah_Stars_season                         │ 2016-10-01 │   1 │
└───────────────────────────────────────────────────┴────────────┴─────┘
```

还可以使用CustomSeparatedWithNames来正确导出和导入标头。搜索正则表达式和模板格式来处理更复杂的情况。

处理大型CSV文件

1. CSV文件可能很大，ClickHouse可以高效处理任何大小的文件。大型文件通常经过压缩，ClickHouse可以解决这个问题，无需在处理前解压。可以COMPRESSION在插入期间使用句子：

```
INSERT INTO sometable
FROM INFILE 'data_csv.csv.gz'
COMPRESSION 'gzip' FORMAT CSV
```

2. 如果COMPRESSION省略子句，ClickHouse 仍将尝试根据文件扩展名猜测文件压缩。可以使用相同的方法将文件直接导出为压缩格式，这将创建一个压缩data_csv.csv.gz文件。

```
SELECT *
FROM for_csv
INTO OUTFILE 'data_csv.csv.gz'
COMPRESSION 'gzip' FORMAT CSV
```

SQL转储

SQL转储

ClickHouse 可以通过多种方式轻松集成到OLTP数据库基础架构中。一种方法是使用SQL转储在其他数据库和ClickHouse之间传输数据。

创建SQL转储

1. 可以使用SQLInsert以 SQL格式转储数据。ClickHouse将在INSERT INTO <table name> VALUES(...表单中写入数据并使用output_format_sql_insert_table_name设置选项作为表名：

```
SET output_format_sql_insert_table_name = 'some_table';
SELECT * FROM some_data
INTO OUTFILE 'dump.sql'
FORMAT SQLInsert
```

2. 可以通过禁用选项来省略列名output_format_sql_insert_include_column_names

```
SET output_format_sql_insert_include_column_names = 0
```

3.现在可以将dump.sql文件提供给另一个OLTP数据库：

```
mysql some_db < dump.sql
```

4. 假设该some_table表存在于some_dbMySQL 数据库中。

某些 DBMS 可能会限制单个批次中可以处理的值的数量。默认情况下，ClickHouse 将创建 65k 个值批次，但可以使用以下output_format_sql_insert_max_batch_size选项进行更改：

```
`SET output_format_sql_insert_max_batch_size = 1000;`
```

导出一组值

ClickHouse具有Values格式，它与SQLInsert类似，但省略了一部分INSERT INTO table VALUES并仅返回一组值：

```
SELECT * FROM some_data LIMIT 3 FORMAT Values
```

```
('Bangor_City_Forest','2015-07-01',34),('Alireza_Afzal','2017-02-01',24),('Akhaura-Laksam-Chittagong_Line','2015-09-01',30)
```

从SQL转储插入数据

要读取SQL转储，可以使用MySQLDump ：

```
 SELECT *
FROM file('dump.sql', MySQLDump)
LIMIT 5
```

```
 ┌─path──────────────────────────┬──────month─┬─hits─┐
 Bangor_City_Forest            2015-07-01    34 
 Alireza_Afzal                 2017-02-01    24 
 Akhaura-Laksam-Chittagong_Line  2015-09-01    30 
 1973_National_500               2017-10-01    80 
 Attachment                    2017-09-01  1356 
└───────────────────────────────┴────────────┴──────┘
```

默认情况下，ClickHouse将跳过未知列（由input_format_skip_unknown_fields选项控制）并处理转储中找到的第一个表的数据（以防多个表被转储到单个文件中）。将跳过DDL语句。要将数据从 MySQL转储加载到表中（mysql.sql文件）：

```
 INSERT INTO some_data
FROM INFILE 'mysql.sql' FORMAT MySQLDump
```

还可以从MySQL转储文件中自动创建表：

```
 CREATE TABLE table_from_mysql
ENGINE = MergeTree
ORDER BY tuple() AS
SELECT *
FROM file('mysql.sql', MySQLDump)
```

这里table_from_mysql根据ClickHouse自动推断的结构创建了一个表。ClickHouse 可以根据数据检测类型，或者在可用时使用 DDL：

```
 DESCRIBE TABLE table_from_mysql;
```

```
 ┌─name──┬─type─────────────────┬─default_type─┬─default_expression─┬─comment─┬─codec_expression─┬─ttl_expression─
┐
 path   Nullable(String)                                                                
 month  Nullable(Date32)                                                                
 hits   Nullable(UInt32)                                                                
└───────┴──────────────────────┴──────────────┴────────────────────┴─────────┴──────────────────┴────────────────
┘
```

JSON

加载 JSON

在本节中，我们假设 JSON 数据为NDJSON（换行符分隔的JSON）格式，在ClickHouse中称为JSONEachRow。这是加载 JSON 的首选格式，因为它简洁且能有效利用空间，但其他格式也支持输入和输出。

考虑以下JSON示例，它表示来自Python PyPI 数据集的一行：

```
 {
  "date": "2022-11-15",
  "country_code": "ES",
  "project": "clickhouse-connect",
  "type": "bdist_wheel",
  "installer": "pip",
  "python_minor": "3.9",
  "system": "Linux",
  "version": "0.3.0"
}
```

为了将此JSON对象加载到 ClickHouse，必须定义表模式。下面显示了一个简单的模式，其中JSON 键映射到列名：

```
 CREATE TABLE pypi (
  `date` Date,
  `country_code` String,
  `project` String,
  `type` String,
  `installer` String,
  `python_minor` String,
  `system` String,
  `version` String
)
ENGINE = MergeTree
ORDER BY (project, date)
```

ClickHouse 可以加载多种格式的 JSON 数据，并自动根据扩展名和内容推断类型。我们可以使用S3 函数读取上表的 JSON 文件：

```
 SELECT *
FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/pypi/json/*.json.gz')
LIMIT 1
┌────────date─┬─country_code─┬─project────────────┬─type─────────┬─installer──┬─python_minor─┬─sys
┐
│ 2022-11-15 │ CN           │ clickhouse-connect │ bdist_wheel │ bandersnatch │              │        │ 0.2.8   │
└────────────┴──────────────┴────────────────────┴──────────────┴────────────┴──────────────┴────────┴─────────
┘

 1 row in set. Elapsed: 1.232 sec.
```

请注意，我们不需要指定文件格式。相反，我们使用glob模式来读取`*.json.gz`存储桶中的所有文件。ClickHouse 会自动`JSONEachRow`根据文件扩展名和内容推断格式为 (ndjson)。如果 ClickHouse 无法检测到格式，可以通过参数函数手动指定格式。

```
 SELECT * FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/pypi/json/*.json.gz', JSONEachRow)
```

要加载这些文件中的行，可以使用`INSERT INTO SELECT`：

```
 INSERT INTO pypi SELECT * FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/pypi/json/*.json.gz')
Ok.

0 rows in set. Elapsed: 10.445 sec. Processed 19.49 million rows, 35.71 MB (1.87 million rows/s., 3.42 MB/s.)

SELECT * FROM pypi LIMIT 2

┌────────date─┬─country_code─┬─project────────────┐
│ 2022-05-26 │ CN           │ clickhouse-connect │
│ 2022-05-26 │ CN           │ clickhouse-connect │
└────────────┴──────────────┴────────────────────┘

2 rows in set. Elapsed: 0.005 sec. Processed 8.19 thousand rows, 908.03 KB (1.63 million rows/s., 180.38 MB/s.)
```

`FORMAT`也可以使用子句以内联方式加载行，例如：

INSERT INTO pypi FORMAT JSONEachRow {"date":"2022-11-15","country_code":"CN","project":"clickhouse-connect","type":"bdist_wheel","installer":"bandersnatch","python_minor":"","system":"","version":"0.2.8"}

JSON架构推断

ClickHouse 可以自动确定 JSON 数据的结构。这可用于直接查询 JSON 数据（例如使用磁盘`clickhouse-local`或 S3 存储桶），和/或在将数据加载到 ClickHouse 之前自动创建模式。

何时使用类型推断

- 一致的结构- 您要从中推断类型的数据包含您感兴趣的所有列。类型推断后添加的附加列的数据将被忽略并且无法查询。
- 一致的类型- 特定列的数据类型需要兼容。

检测类型

之前的示例使用了 NDJSON 格式的Python PyPI 数据集的简单版本。在本节中，我们将探索具有嵌套结构的更复杂的数据集 -包含 250 万篇学术论文的arXiv数据集。此数据集中的每一行都以 NDJSON 格式分发，代表一篇已发表的学术论文。示例行如下所示：

```
{
 "id": "2101.11408",
 "submitter": "Daniel Lemire",
 "authors": "Daniel Lemire",
 "title": "Number Parsing at a Gigabyte per Second",
 "comments": "Software at https://github.com/fastfloat/fast_float and\n  https://github.com/lemire/simple_fastfloat_benchmark/",
 "journal-ref": "Software: Practice and Experience 51 (8), 2021",
 "doi": "10.1002/spe.2984",
 "report-no": null,
 "categories": "cs.DS cs.MS",
 "license": "http://creativecommons.org/licenses/by/4.0/",
 "abstract": "With disks and networks providing gigabytes per second ....\n",
 "versions": [
  {
   "created": "Mon, 11 Jan 2021 20:31:27 GMT",
   "version": "v1"
  },
  {
   "created": "Sat, 30 Jan 2021 23:57:29 GMT",
   "version": "v2"
  }
 ],
 "update_date": "2022-11-07",
 "authors_parsed": [
  [
   "Lemire",
   "Daniel",
   ""
  ]
 ]
}
```

这些数据需要比前面的示例复杂得多的架构。我们在下面概述了定义此架构的过程，并引入了诸如和`Tuple`之类的复杂类型`Array`。

该数据集存储在公共 S3 存储桶中`s3://datasets-documentation/arxiv/arxiv.json.gz`。

您可以看到上面的数据集包含嵌套的 JSON 对象。虽然用户应该起草并版本化他们的架构，但推断允许从数据中推断类型。这允许自动生成架构 DDL，避免手动构建它并加速开发过程。

使用命令中的s3函数`DESCRIBE`显示将被推断的类型。

```
DESCRIBE TABLE s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/arxiv/arxiv.json.gz')
SETTINGS describe_compact_output = 1
```

```
┌─name──────────┬─type──────────────────────────────────────────────────────────────┐
│ id            │ Nullable(String)                                                  │
│ submitter     │ Nullable(String)                                                  │
│ authors       │ Nullable(String)                                                  │
│ title         │ Nullable(String)                                                  │
│ comments      │ Nullable(String)                                                  │
│ journal-ref   │ Nullable(String)                                                  │
│ doi           │ Nullable(String)                                                  │
│ report-no     │ Nullable(String)                                                  │
│ categories    │ Nullable(String)                                                  │
│ license       │ Nullable(String)                                                  │
│ abstract      │ Nullable(String)                                                  │
│ versions      │ Array(Tuple(created Nullable(String),version Nullable(String)))   │
│ update_date   │ Nullable(Date)                                                    │
│ authors_parsed │ Array(Array(Nullable(String)))                                   │
└───────────────┴───────────────────────────────────────────────────────────────────┘
```

避免空值

> 您可以看到很多列被检测为 Nullable。我们不建议在不是绝对需要时使用 Nullable 类型。您可以使用schema_inference_make_columns_nullable来控制应用 Nullable 时的行为。

可以看到，大多数列已自动检测为String，update_date列正确检测为Date。versions列已创建为用于Array(Tuple(created String, version String))存储对象列表，authors_parsed定义为Array(Array(String))用于嵌套数组。

查询 JSON

可以依靠模式推理来就地查询JSON数据。下面,我们找到了每年的顶级作者，利用了日期和数组被自动检测的事实。

```
 SELECT
   toYear(update_date) AS year,
   authors,
   count() AS c
FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/arxiv/arxiv.json.gz')
GROUP BY
   year,
   authors
ORDER BY
   year ASC,
   c DESC
LIMIT 1 BY year

┌─year─┬─authors─────────────────────────────────────────────────────────────────────────┬───c─┐
│ 2007 │ The BABAR Collaboration, B. Aubert, et al                                       │  98 │
│ 2008 │ The OPAL collaboration, G. Abbiendi, et al                                      │  59 │
│ 2009 │ Ashoke Sen                                                                      │  77 │
│ 2010 │ The BABAR Collaboration, B. Aubert, et al                                       │ 117 │
│ 2011 │ Amelia Carolina Sparavigna                                                      │  21 │
│ 2012 │ ZEUS Collaboration                                                              │ 140 │
│ 2013 │ CMS Collaboration                                                               │ 125 │
│ 2014 │ CMS Collaboration                                                               │  87 │
│ 2015 │ ATLAS Collaboration                                                             │ 118 │
│ 2016 │ ATLAS Collaboration                                                             │ 126 │
│ 2017 │ CMS Collaboration                                                               │ 122 │
│ 2018 │ CMS Collaboration                                                               │ 138 │
│ 2019 │ CMS Collaboration                                                               │ 113 │
│ 2020 │ CMS Collaboration                                                               │  94 │
│ 2021 │ CMS Collaboration                                                               │  69 │
│ 2022 │ CMS Collaboration                                                               │  62 │
│ 2023 │ ATLAS Collaboration                                                             │ 128 │
│ 2024 │ ATLAS Collaboration                                                             │ 120 │
└──────┴─────────────────────────────────────────────────────────────────────────────────┴─────┘

18 rows in set. Elapsed: 20.172 sec. Processed 2.52 million rows, 1.39 GB (124.72 thousand rows/s., 68.76 MB/s.)
```

模式推断允许我们查询JSON文件而无需指定模式，从而加速临时数据分析任务。

创建表

1. 我们可以依靠模式推断来创建表的模式。以下`CREATE AS EMPTY`命令将推断表的 DDL 并创建表。这不会加载任何数据，以下示例使用的文件来创建表。

```
 CREATE TABLE arxiv
ENGINE = MergeTree
ORDER BY update_date EMPTY
AS SELECT *
FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/arxiv/arxiv.json.gz')
SETTINGS schema_inference_make_columns_nullable = 0
```

2.为了确认表模式，我们使用以下`SHOW CREATE TABLE`命令：

```
 SHOW CREATE TABLE arxiv

CREATE TABLE arxiv
(
    `id` String,
    `submitter` String,
    `authors` String,
    `title` String,
    `comments` String,
    `journal-ref` String,
    `doi` String,
    `report-no` String,
    `categories` String,
    `license` String,
    `abstract` String,
    `versions` Array(Tuple(created String, version String)),
    `update_date` Date,
    `authors_parsed` Array(Array(String))
)
ENGINE = MergeTree
ORDER BY update_date
SETTINGS index_granularity = 8192
```

以上是此数据的正确架构。架构推断基于对数据进行采样并逐行读取数据。根据格式提取列值，使用递归解析器和启发式方法确定每个值的类型。架构推断中从数据中读取的最大行数和字节数由设置`input_format_max_rows_to_read_for_schema_inference`(默认为 25000)
和`input_format_max_bytes_to_read_for_schema_inference`(默认为 32MB) 控制。

使用函数创建表

使用函数创建表，如下所示：

```
 CREATE TABLE arxiv
ENGINE = MergeTree
ORDER BY update_date EMPTY
AS SELECT *
FROM format(JSONEachRow, '{"id":"2101.11408","submitter":"Daniel Lemire","authors":"Daniel Lemire","title":"Number Parsing at a Gigabyte per
Second","comments":"Software at https://github.com/fastfloat/fast_float and","doi":"10.1002/spe.2984","report-no":null,"categories":"cs.DS
cs.MS","license":"http://creativecommons.org/licenses/by/4.0/","abstract":"Withdisks and networks providing gigabytes per second ","versions":
[{"created":"Mon, 11 Jan 2021 20:31:27 GMT","version":"v1"},{"created":"Sat, 30 Jan 2021 23:57:29 GMT","version":"v2"}],"update_date":"2022-11-
07","authors_parsed":[["Lemire","Daniel",""]]}') SETTINGS schema_inference_make_columns_nullable = 0

SHOW CREATE TABLE arxiv

CREATE TABLE arxiv
(
    `id` String,
    `submitter` String,
    `authors` String,
    `title` String,
    `comments` String,
    `doi` String,
    `report-no` String,
    `categories` String,
    `license` String,
    `abstract` String,
    `versions` Array(Tuple(created String, version String)),
    `update_date` Date,
    `authors_parsed` Array(Array(String))
)
ENGINE = MergeTree
ORDER BY update_date
```

加载JSON数据

1. 前面的命令创建了一个可以加载数据的表。现在，您可以使用以下命令将数据插入表中`INSERT INTO SELECT`：

```
 INSERT INTO arxiv SELECT *
FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/arxiv/arxiv.json.gz')

0 rows in set. Elapsed: 38.498 sec. Processed 2.52 million rows, 1.39 GB (65.35 thousand rows/s., 36.03 MB/s.)
Peak memory usage: 870.67 MiB.
```

2.加载后，我们可以查询数据，可选择使用格式`PrettyJSONEachRow`以显示其原始结构中的行：

```
 SELECT *
FROM arxiv
LIMIT 1
FORMAT PrettyJSONEachRow

{
    "id": "0704.0004",
    "submitter": "David Callan",
    "authors": "David Callan",
    "title": "A determinant of Stirling cycle numbers counts unlabeled acyclic",
    "comments": "11 pages",
    "journal-ref": "",
    "doi": "",
    "report-no": "",
    "categories": "math.CO",
    "license": "",
    "abstract": "  We show that a determinant of Stirling cycle numbers counts unlabeled acyclic\nsingle-source automata.",
    "versions": [
        {
            "created": "Sat, 31 Mar 2007 03:16:14 GMT",
            "version": "v1"
        }
    ],
    "update_date": "2007-05-23",
    "authors_parsed": [
        [
            "Callan",
            "David"
        ]
    ]
}

1 row in set. Elapsed: 0.009 sec.
```

设计 JSON 架构

虽然可以使用架构推断来为JSON数据建立初始架构并就地查询 JSON 数据文件（例如在 S3 中），但用户应该致力于为其数据建立优化的版本化架构。我们在下面讨论用于建模 JSON 结构的选项。

静态 JSON 与动态 JSON

定义 JSON 架构的主要任务是确定每个键值的适当类型。我们建议用户对 JSON 层次结构中的每个键递归应用以下规则，以确定每个键的适当类型。

1. **原始类型**-如果键的值是原始类型，无论它是子对象的一部分还是位于根上，请确保根据通用架构设计最佳实践和类型优化规则选择其类型。原始数组（例如，`phone_numbers`如下所示）可以建模为`Array(<type>)`。`Array(String)`。
2. **静态与动态**- 如果键的值是一个复杂对象，即一个对象或一个对象数组，则确定它是否会发生变化。很少有新键的对象，可以通过模式更改来预测和处理新键的添加和处理新键的添加 `ALTER TABLE ADD COLUMN`，可以被视为静态的。这包括在某些 JSON 文档中可能只提供部分键的对象。经常添加新键和/或不可预测的新键的对象应被视为动态的。

重要提示：上述规则应递归应用。如果确定某个键的值是动态的，则无需进一步评估，并可遵循处理动态对象中的指导原则。如果对象是静态的，则继续评估子键，直到遇到键值是原始键或动态键为止。

3. 为了说明这些规则，使用以下代表一个 JSON 示例：

```
{
 "id": 1,
 "name": "Clicky McCliickHouse",
 "username": "Clicky",
 "email": "clicky@clickhouse.com",
 "address": [
  {
   "street": "Victor Plains",
   "suite": "Suite 879",
   "city": "Wisokyburgh",
   "zipcode": "90566-7771",
   "geo": {
    "lat": -43.9509,
    "lng": -34.4618
   }
  }
 ],
 "phone_numbers": ["010-692-6593", "020-192-3333"],
 "website": "clickhouse.com",
 "company": {
  "name": "ClickHouse",
  "catchPhrase": "The real-time data warehouse for analytics",
  "labels": {
   "type": "database systems",
   "founded": "2021"
  }
 },
 "dob": "2007-03-31",
 "tags": {
  "hobby": "Databases",
  "holidays": [
   {
    "year": 2024,
    "location": "Azores, Portugal"
   }
  ],
  "car": {
   "model": "Tesla",
   "year": 2023
  }
 }
}
```

4.应用这些规则：

*根键name、username、email、website可表示为 类型String。列phone_numbers是 类型的数组原语Array(String)，分别具有dob和id类型Date和UInt32。

- 不会向address对象添加新键（只有新地址对象），因此可以将其视为静态的。如果我们递归，则除 之外的所有子列都可以被视为基元（和类型String）geo。这也是一个具有两列的静态结构Float32，lat和lon。
- 该tags列是动态的。我们假设可以向任何类型和结构的此对象添加新的任意标签。
- 该company对象是静态的，并且始终最多包含指定的 3 个键。子键name和catchPhrase的类型为String。键labels是动态的。我们假设可以向此对象添加新的任意标签。值始终是字符串类型的键值对。

处理静态对象

1. 建议使用命名元组来处理静态对象`Tuple`。可以使用元组数组`Array(Tuple)`来保存对象数组。在元组本身中，应使用相同的规则定义列及其各自的类型。这可以导致嵌套元组来表示嵌套对象，如下所示。

为了说明这一点，我们使用前面的 JSON 人员示例，省略动态对象：

```
 {
  "id": 1,
  "name": "Clicky McCliickHouse",
  "username": "Clicky",
  "email": "clicky@clickhouse.com",
  "address": [
   {
    "street": "Victor Plains",
    "suite": "Suite 879",
    "city": "Wisokyburgh",
    "zipcode": "90566-7771",
    "geo": {
     "lat": -43.9509,
     "lng": -34.4618
    }
   }
  ],
  "phone_numbers": ["010-692-6593", "020-192-3333"],
  "website": "clickhouse.com",
  "company": {
   "name": "ClickHouse",
   "catchPhrase": "The real-time data warehouse for analytics"
  },
  "dob": "2007-03-31"
}
```

2. 该表的架构如下所示：

```
 CREATE TABLE people
(
    `id` Int64,
    `name` String,
    `username` String,
    `email` String,
    `address` Array(Tuple(city String, geo Tuple(lat Float32, lng Float32), street String, suite String, zipcode String)),
    `phone_numbers` Array(String),
    `website` String,
    `company` Tuple(catchPhrase String, name String),
    `dob` Date
)
ENGINE = MergeTree
ORDER BY username
```

注意`company`列是如何定义为的`Tuple(catchPhrase String, name String)`。该`address`字段使用`Array(Tuple)`带有嵌套的`Tuple`来表示`geo`列。

3.可以将JSON按照其当前结构插入到此表中：

```
 INSERT INTO people FORMAT JSONEachRow
{"id":1,"name":"Clicky McCliickHouse","username":"Clicky","email":"clicky@clickhouse.com","address":[{"street":"Victor Plains","suite":"Suite
879","city":"Wisokyburgh","zipcode":"90566-7771","geo":{"lat":-43.9509,"lng":-34.4618}}],"phone_numbers":["010-692-6593","020-192-
3333"],"website":"clickhouse.com","company":{"name":"ClickHouse","catchPhrase":"The real-time data warehouse for analytics"},"dob":"2007-03-31"}
```

4.在上面的例子中，有最少的数据。但如下所示，可以通过句点分隔的名称来查询元组字段：

```
 SELECT
   address.street,
   company.name
FROM people

┌─address.street────┬─company.name─┐
│ ['Victor Plains'] │ ClickHouse   │
└───────────────────┴──────────────┘
```

5.请注意`address.street`列是如何作为 来返回的`Array`。要按位置查询数组内的特定对象，应在列名后指定数组偏移量。例如，要从第一个地址访问street：

```
 SELECT address.street[1] AS street
FROM people

┌─street─────────────┐
│ Victor Plains │
└────────────────────┘

1 row in set. Elapsed: 0.001 sec.
```

元组的主要缺点是子列不能用于排序键。因此，以下操作将失败：

```
 CREATE TABLE people
(
    `id` Int64,
    `name` String,
    `username` String,
    `email` String,
    `address` Array(Tuple(city String, geo Tuple(lat Float32, lng Float32), street String, suite String, zipcode String)),
    `phone_numbers` Array(String),
    `website` String,
    `company` Tuple(catchPhrase String, name String),
    `dob` Date
)
ENGINE = MergeTree
ORDER BY company.name

Code: 47. DB::Exception: Missing columns: 'company.name' while processing query: 'company.name', required columns: 'company.name' 'company.name'.
(UNKNOWN_IDENTIFIER)
```

> **排序键中的元组**
> 虽然元组列不能用于排序键，但可以使用整个元组。虽然可能，但这很少有意义。

处理默认值

即使JSON对象是结构化的，它们通常也很稀疏，只提供了已知键的子集。幸运的是，该`Tuple`类型不需要 JSON 有效负载中的所有列。如果未提供，则将使用默认值。

1. 考虑之前的表和下面的`people`稀疏 JSON，缺少键`suite`、`geo`和`phone_numbers` `catchPhrase`:

```
{
 "id": 1,
 "name": "Clicky McCliickHouse",
 "username": "Clicky",
 "email": "clicky@clickhouse.com",
 "address": [
  {
    "street": "Victor Plains",
    "city": "Wisokyburgh",
    "zipcode": "90566-7771"
  }
 ],
 "website": "clickhouse.com",
 "company": {
   "name": "ClickHouse"
 },
 "dob": "2007-03-31"
}
```

2.可以从下面看到这一行可以成功插入：

```
 INSERT INTO people FORMAT JSONEachRow
{"id":1,"name":"Clicky McCliickHouse","username":"Clicky","email":"clicky@clickhouse.com","address":[{"street":"Victor
Plains","city":"Wisokyburgh","zipcode":"90566-7771"}],"website":"clickhouse.com","company":{"name":"ClickHouse"},"dob":"2007-03-31"}

Ok.

1 row in set. Elapsed: 0.002 sec.
```

3.查询此单行，可以看到省略的列（包括子对象）使用了默认值：

```
 SELECT *
FROM people
FORMAT PrettyJSONEachRow

{
    "id": "1",
    "name": "Clicky McCliickHouse",
    "username": "Clicky",
    "email": "clicky@clickhouse.com",
    "address": [
        {
            "city": "Wisokyburgh",
            "geo": {
                "lat": 0,
                "lng": 0
            },
            "street": "Victor Plains",
            "suite": "",
            "zipcode": "90566-7771"
        }
    ],
    "phone_numbers": [],
    "website": "clickhouse.com",
    "company": {
        "catchPhrase": "",
        "name": "ClickHouse"
    },
    "dob": "2007-03-31"
}

1 row in set. Elapsed: 0.001 sec.
```

区分empty和null

如果用户需要区分值为空和未提供，则可以使用Nullable类型。除非绝对必要，否则应避免这样做，因为这会对此类列的存储和查询性能产生负面影响。

处理新列

虽然当JSON键是静态的时候，结构化方法是最简单的，但如果可以规划对模式的更改，即提前知道新键，并且可以相应地修改模式，则仍然可以使用此方法。

请注意，ClickHouse 默认会忽略有效负载中提供但架构中不存在的 JSON 键。考虑以下修改后的 JSON 有效负载并添加了一个`nickname`键：

```
 {
  "id": 1,
  "name": "Clicky McCliickHouse",
  "nickname": "Clicky",
  "username": "Clicky",
  "email": "clicky@clickhouse.com",
  "address": [
   {
     "street": "Victor Plains",
     "suite": "Suite 879",
     "city": "Wisokyburgh",
     "zipcode": "90566-7771",
     "geo": {
       "lat": -43.9509,
       "lng": -34.4618
     }
   }
  ],
  "phone_numbers": ["010-692-6593", "020-192-3333"],
  "website": "clickhouse.com",
  "company": {
    "name": "ClickHouse",
    "catchPhrase": "The real-time data warehouse for analytics"
  },
  "dob": "2007-03-31"
}
```

忽略键即可成功插入此JSON`nickname`：

```
 INSERT INTO people FORMAT JSONEachRow
{"id":1,"name":"Clicky McCliickHouse","nickname":"Clicky","username":"Clicky","email":"clicky@clickhouse.com","address":[{"street":"Victor Plains","suite":"Suite 879","city":"Wisokyburgh","zipcode":"90566-7771","geo":{"lat":-43.9509,"lng":-34.4618}}],"phone_numbers":["010-692-6593","020-192-3333"],"website":"clickhouse.com","company":{"name":"ClickHouse","catchPhrase":"The real-time data warehouse for analytics"},"dob":"2007-03-31"}

Ok.

1 row in set. Elapsed: 0.002 sec.
```

可以使用命令将列添加到架构中`ALTER TABLE ADD COLUMN`。可以通过子句指定默认值`DEFAULT`，如果在后续插入期间未指定默认值，则将使用该默认值。不存在此值的行（因为它们是在创建之前插入的）也将返回此默认值。如果没有`DEFAULT`指定值，则将使用该类型的默认值。

例如：

```
 -- insert initial row (nickname will be ignored)
INSERT INTO people FORMAT JSONEachRow
{"id":1,"name":"Clicky McCliickHouse","nickname":"Clicky","username":"Clicky","email":"clicky@clickhouse.com","address":[{"street":"Victor Plains","suite":"Suite 879","city":"Wisokyburgh","zipcode":"90566-7771","geo":{"lat":-43.9509,"lng":-34.4618}}],"phone_numbers":["010-692-6593","020-192-3333"],"website":"clickhouse.com","company":{"name":"ClickHouse","catchPhrase":"The real-time data warehouse for analytics"},"dob":"2007-03-31"}

-- add column
ALTER TABLE people
    (ADD COLUMN `nickname` String DEFAULT 'no_nickname')

-- insert new row (same data different id)
INSERT INTO people FORMAT JSONEachRow
{"id":2,"name":"Clicky McCliickHouse","nickname":"Clicky","username":"Clicky","email":"clicky@clickhouse.com","address":[{"street":"Victor Plains","suite":"Suite 879","city":"Wisokyburgh","zipcode":"90566-7771","geo":{"lat":-43.9509,"lng":-34.4618}}],"phone_numbers":["010-692-6593","020-192-3333"],"website":"clickhouse.com","company":{"name":"ClickHouse","catchPhrase":"The real-time data warehouse for analytics"},"dob":"2007-03-31"}

-- select 2 rows
SELECT id, nickname FROM people

┌─id─┬─nickname─────┐
│  2 │ Clicky       │
│  1 │ no_nickname  │
└────┴──────────────┘

2 rows in set. Elapsed: 0.001 sec.
```

处理动态对象

1. 有两种推荐的方法来处理动态对象：

- Map(String,V)类型
- 带有 JSON 函数的字符串

2. 可以应用以下规则来确定最合适的。

- 如果对象高度动态，没有可预测的结构并且包含任意嵌套对象，则用户应使用该`String`类型。可以使用 JSON 函数在查询时提取值，如下所示。
- 如果对象用于存储任意键，且大多为一种类型，请考虑使用该`Map`类型。理想情况下，唯一键的数量不应超过几百个。`Map`对于具有子对象的对象，也可以考虑使用该类型，前提是后者的类型一致。通常，我们建议`Map`将该类型用于标签和标记，例如日志数据中的 Kubernetes pod 标签。

> **应用对象级方法**
> 不同的技术可以应用于同一架构中的不同对象。有些对象可以用String和其他来最好地解决`Map`。请注意，一旦`String`使用类型，就不需要做出进一步的架构决策。相反，可以将子对象嵌套在键中`Map`，如下所示 - 包括`String`表示JSON。

使用字符串

对于使用动态 JSON 的用户来说，使用上述结构化方法处理数据通常不可行，因为动态 JSON 可能会发生变化，或者其架构不太为人所理解。为了获得绝对的灵活性，用户可以简单地将 JSON 存储为Strings，然后根据需要使用函数提取字段。这与将 JSON 作为结构化对象处理完全相反。这种灵活性会产生成本，并带来重大缺点 - 主要是查询语法复杂性增加以及性能下降。

如前所述，对于原始person对象，我们无法确保列的结构`tags`。我们插入原始行（还包括`company.labels`，暂时忽略），将`Tags`列声明为`String`：

```
CREATE TABLE people
(
    `id` Int64,
    `name` String,
    `username` String,
    `email` String,
    `address` Array(Tuple(city String, geo Tuple(lat Float32, lng Float32), street String, suite String, zipcode String)),
    `phone_numbers` Array(String),
    `website` String,
    `company` Tuple(catchPhrase String, name String),
    `dob` Date,
    `tags` String
)
ENGINE = MergeTree
ORDER BY username

INSERT INTO people FORMAT JSONEachRow
{"id":1,"name":"Clicky McCliickHouse","username":"Clicky","email":"clicky@clickhouse.com","address":[{"street":"Victor Plains","suite":"Suite 879","city":"Wisokyburgh","zipcode":"90566-7771","geo":{"lat":-43.9509,"lng":-34.4618}}],"phone_numbers":["010-692-6593","020-192-3333"],"website":"clickhouse.com","company":{"name":"ClickHouse","catchPhrase":"The real-time data warehouse for analytics","labels":{"type":"database systems","founded":"2021"}},"dob":"2007-03-31","tags":{"hobby":"Databases","holidays":[{"year":2024,"location":"Azores, Portugal"}],"car":{"model":"Tesla","year":2023}}}

Ok.
1 row in set. Elapsed: 0.002 sec.
```

可以选择该tags列并看到JSON已作为字符串插入：

```
SELECT tags
FROM people

┌─tags─────────────────────────────────────────────────────────────────────────────────────────────────────────────────┐
│ {"hobby":"Databases","holidays":[{"year":2024,"location":"Azores, Portugal"}],"car":{"model":"Tesla","year":2023}} │
└──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘

1 row in set. Elapsed: 0.001 sec.
```

JSONExtract函数可用于从此 JSON 中检索值。请考虑以下简单示例：

```
SELECT JSONExtractString(tags, 'holidays') as holidays FROM people

┌─holidays─────────────────────────────────────────┐
│ [{"year":2024,"location":"Azores, Portugal"}] │
└──────────────────────────────────────────────────┘

1 row in set. Elapsed: 0.002 sec.
```

请注意，函数既需要对String列的引用tags，也需要提取 JSON 中的路径。嵌套路径需要嵌套函数，例如JSONExtractUInt(JSONExtractString(tags, 'car'), 'year')提取列。可以通过函数JSON_QUERY和JSON_VALUEtags.car.year简化嵌套路径的提取。

考虑数据集中的极端情况arxiv，我们将整个body视为一个String。

```
CREATE TABLE arxiv (
 body String
)
ENGINE = MergeTree ORDER BY ()
```

要插入此模式，我们需要使用以下JSONAsString格式：

```
INSERT INTO arxiv SELECT *
FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/arxiv/arxiv.json.gz', 'JSONAsString')

0 rows in set. Elapsed: 25.186 sec. Processed 2.52 million rows, 1.38 GB (99.89 thousand rows/s., 54.79 MB/s.)
```

假设我们希望统计按年份发布的论文数量。将查询与结构化版本的架构进行对比，并与仅使用字符串的情况进行比较：

```
 -- using structured schema
SELECT
    toYear(parseDateTimeBestEffort(versions.created[1])) AS published_year,
    count() AS c
FROM arxiv_v2
GROUP BY published_year
ORDER BY c ASC
LIMIT 10

┌─published_year─┬─────c─┐
│           1986 │     1 │
│           1988 │     1 │
│           1989 │     6 │
│           1990 │    26 │
│           1991 │   353 │
│           1992 │  3190 │
│           1993 │  6729 │
│           1994 │ 10078 │
│           1995 │ 13006 │
│           1996 │ 15872 │
└────────────────┴───────┘

10 rows in set. Elapsed: 0.264 sec. Processed 2.31 million rows, 153.57 MB (8.75 million rows/s., 582.58 MB/s.)

-- using unstructured String

SELECT
    toYear(parseDateTimeBestEffort(JSON_VALUE(body, '$.versions[0].created'))) AS published_year,
    count() AS c
FROM arxiv
GROUP BY published_year
ORDER BY published_year ASC
LIMIT 10

┌─published_year─┬─────c─┐
│           1986 │     1 │
│           1988 │     1 │
│           1989 │     6 │
│           1990 │    26 │
│           1991 │   353 │
│           1992 │  3190 │
│           1993 │  6729 │
│           1994 │ 10078 │
│           1995 │ 13006 │
│           1996 │ 15872 │
└────────────────┴───────┘

10 rows in set. Elapsed: 1.281 sec. Processed 2.49 million rows, 4.22 GB (1.94 million rows/s., 3.29 GB/s.)
Peak memory usage: 205.98 MiB.
```

注意这里使用 xpath 表达式来按方法过滤`JSON JSON_VALUE(body, '$.versions[0].created')`。

字符串函数比使用索引的显式类型转换慢得多（> 10 倍）。上述查询始终需要全表扫描并处理每一行。虽然这些查询在像这样的小型数据集上仍然很快，但在较大的数据集上性能会下降。

这种方法的灵活性显然是以性能和语法为代价的，因此它只应该用于模式中高度动态的对象。

简单的JSON的函数

上述示例使用了 JSON* 系列函数。这些函数利用了基于simdjson 的完整 JSON 解析器，该解析器解析严谨，并能区分嵌套在不同级别的相同字段。这些函数能够处理语法正确但格式不正确的 JSON，例如键之间的双空格。

有一组更快、更严格的函数可用。这些`simpleJSON*`函数主要通过对 JSON 的结构和格式做出严格的假设来提供潜在的卓越性能。具体来说：

- 字段名称必须是常量
- 字段名称的一致编码，例如`simpleJSONHas('{"abc":"def"}', 'abc') = 1`，但是`visitParamHas('{"\\u0061\\u0062\\u0063":"def"}', 'abc') = 0`
- 字段名称在所有嵌套结构中都是唯一的。嵌套级别之间没有区别，匹配也是无差别的。如果有多个匹配字段，则使用第一个匹配的字段。
- 字符串文字之外不得有特殊字符。这包括空格。以下内容无效，无法解析。

```
 {"@timestamp": 893964617, "clientip": "40.135.0.0", "request": {"method": "GET",
"path": "/images/hm_bg.jpg", "version": "HTTP/1.0"}, "status": 200, "size": 24736}
```

而以下内容将正确解析：

```
 {"@timestamp":893964617,"clientip":"40.135.0.0","request":{"method":"GET",
"path":"/images/hm_bg.jpg","version":"HTTP/1.0"},"status":200,"size":24736}
```

simpleJSON* 在某些情况下，如果性能至关重要并且您的 JSON 满足上述要求，这些可能是合适的。下面显示了使用函数重写的早期查询的示例：

```
 SELECT
   toYear(parseDateTimeBestEffort(simpleJSONExtractString(simpleJSONExtractRaw(body, 'versions'), 'created'))) AS published_year,
   count() AS c
FROM arxiv
GROUP BY published_year
ORDER BY published_year ASC
LIMIT 10

┌─published_year─┬─────c─┐
│           1986 │     1 │
│           1988 │     1 │
│           1989 │     6 │
│           1990 │    26 │
│           1991 │   353 │
│           1992 │  3190 │
│           1993 │  6729 │
│           1994 │ 10078 │
│           1995 │ 13006 │
│           1996 │ 15872 │
└────────────────┴───────┘

10 rows in set. Elapsed: 0.964 sec. Processed 2.48 million rows, 4.21 GB (2.58 million rows/s., 4.36 GB/s.)
Peak memory usage: 211.49 MiB.
```

上述代码使用simpleJSONExtractString来提取created密钥，利用了我们只需要发布日期的第一个值这一事实。在这种情况下，函数的限制simpleJSON* 对于性能的提升是可以接受的。

使用 Map

如果对象用于存储主要为一种类型的任意键，请考虑使用该Map类型。理想情况下，唯一键的数量不应超过几百个。我们建议Map将该类型用于标签和标记，例如日志数据中的 Kubernetes pod 标签。虽然这是一种表示嵌套结构的简单方法，Map但它有一些明显的局限性：

- 所有字段必须属于同一类型。
- 访问子列需要特殊的映射语法，因为字段不作为列存在；整个对象是一列。
- 访问子列会加载整个Map值，即所有同级值及其各自的值。对于较大的地图，这可能会导致严重的性能损失。

> **字符串键**
> 当将对象建模为Map时，String使用 key 来存储JSON 键名称。因此映射将始终为Map(String, T)，其中T取决于数据。

原始值

最简单的应用Map是当对象包含与值相同的原始类型时。在大多数情况下，这涉及使用String值的类型T。

1. 考虑我们之前的Person JSON，其中company.labels对象被确定为动态的。重要的是，我们只希望将String类型的键值对添加到此对象。因此，我们可以将其声明为Map(String, String)：

```
 CREATE TABLE people
(
   `id` Int64,
   `name` String,
   `username` String,
   `email` String,
   `address` Array(Tuple(city String, geo Tuple(lat Float32, lng Float32), street String, suite String, zipcode String)),
   `phone_numbers` Array(String),
   `website` String,
   `company` Tuple(catchPhrase String, name String, labels Map(String,String)),
   `dob` Date,
   `tags` String
)
ENGINE = MergeTree
ORDER BY username
```

2. 可以插入原始的完整 JSON 对象：

```
 INSERT INTO people FORMAT JSONEachRow
{"id":1,"name":"Clicky McCliickHouse","username":"Clicky","email":"clicky@clickhouse.com","address":[{"street":"Victor Plains","suite":"Suite 879","city":"Wisokyburgh","zipcode":"90566-7771","geo":{"lat":-43.9509,"lng":-34.4618}}],"phone_numbers":["010-692-6593","020-192-3333"],"website":"clickhouse.com","company":{"name":"ClickHouse","catchPhrase":"The real-time data warehouse for analytics","labels":{"type":"database systems","founded":"2021"}},"dob":"2007-03-31","tags":{"hobby":"Databases","holidays":[{"year":2024,"location":"Azores, Portugal"}],"car":{"model":"Tesla","year":2023}}}

Ok.

1 row in set. Elapsed: 0.002 sec.
```

3. 在请求对象中查询这些字段需要使用映射语法，例如：

`SELECT company.labels FROM people

┌─company.labels─┐ │ {'type':'database systems','founded':'2021'} │ └────────────────┘

1 row in set. Elapsed: 0.001 sec.

SELECT company.labels['type'] AS type FROM people

┌─type─┐ │ database systems │ └──────┘

1 row in set. Elapsed: 0.001 sec.`

对象值

该Map类型也可以考虑用于具有子对象的对象，前提是后者的类型具有一致性。

1. 假设tags我们persons对象的键需要一致的结构，其中每个子对象tag都有一个name和time列。此类JSON文档的简化示例可能如下所示：

```
 {
  "id": 1,
  "name": "Clicky McCliickHouse",
  "username": "Clicky",
  "email": "clicky@clickhouse.com",
  "tags": {
    "hobby": {
      "name": "Diving",
      "time": "2024-07-11 14:18:01"
    },
    "car": {
      "name": "Tesla",
      "time": "2024-07-11 15:18:23"
    }
  }
}
```

2. Map(String, Tuple(name String, time DateTime))可以用如下所示的模型来建模：

```
 CREATE TABLE people
(
    `id` Int64,
    `name` String,
    `username` String,
    `email` String,
    `tags` Map(String, Tuple(name String, time DateTime))
)
ENGINE = MergeTree
ORDER BY username

INSERT INTO people FORMAT JSONEachRow
{"id":1,"name":"Clicky McCliickHouse","username":"Clicky","email":"clicky@clickhouse.com","tags":{"hobby":{"name":"Diving","time":"2024-07-11
14:18:01"},"car":{"name":"Tesla","time":"2024-07-11 15:18:23"}}}

Ok.

1 row in set. Elapsed: 0.002 sec.

SELECT tags['hobby'] AS hobby
FROM people
FORMAT JSONEachRow

{"hobby":{"name":"Diving","time":"2024-07-11 14:18:01"}}

1 row in set. Elapsed: 0.001 sec.
```

3. 在这种情况下，映射的应用通常很少见，并且建议对数据进行重构，以使动态键名不包含子对象。例如，可以将上述内容重构如下，以允许使用`Array(Tuple(key String, name String, time DateTime)`。

```
 {
  "id": 1,
  "name": "Clicky McCliickHouse",
  "username": "Clicky",
  "email": "clicky@clickhouse.com",
  "tags": [
   {
    "key": "hobby",
    "name": "Diving",
    "time": "2024-07-11 14:18:01"
   },
   {
    "key": "car",
    "name": "Tesla",
    "time": "2024-07-11 15:18:23"
   }
  ]
}
```

导出 JSON

1. 几乎任何用于导入的 JSON 格式都可以用于导出。最流行的是`JSONEachRow`：

```
 SELECT * FROM sometable FORMAT JSONEachRow
```

```
 {"path":"Bob_Dolman","month":"2016-11-01","hits":245}
{"path":"1-krona","month":"2017-01-01","hits":4}
{"path":"Ahmadabad-e_Kalij-e_Sofla","month":"2017-01-01","hits":3}
```

2. 或者可以`JSONCompactEachRow`通过跳过列名来节省磁盘空间：

```
 SELECT * FROM sometable FORMAT JSONCompactEachRow
```

```
 ["Bob_Dolman", "2016-11-01", 245]
["1-krona", "2017-01-01", 4]
["Ahmadabad-e_Kalij-e_Sofla", "2017-01-01", 3]
```

将数据类型覆盖为字符串

ClickHouse 尊重数据类型并将根据标准导出 JSON。但如果我们需要将所有值编码为字符串，则可以使用JSONStringsEachRow格式：

```
SELECT * FROM sometable FORMAT JSONStringsEachRow
```

```
 {"path":"Bob_Dolman","month":"2016-11-01","hits":"245"}
{"path":"1-krona","month":"2017-01-01","hits":"4"}
{"path":"Ahmadabad-e_Kalij-e_Sofla","month":"2017-01-01","hits":"3"}
```

现在，hits数字列被编码为字符串。所有 JSON* 格式都支持导出为字符串，只需探索JSONStrings*并JSONCompactStrings*格式化：

```
SELECT * FROM sometable FORMAT JSONCompactStringsEachRow
```

```
 ["Bob_Dolman", "2016-11-01", "245"]
["1-krona", "2017-01-01", "4"]
["Ahmadabad-e_Kalij-e_Sofla", "2017-01-01", "3"]
```

导出元数据和数据

1. 应用程序中流行的通用JSON格式不仅会导出结果数据，还会导出列类型和查询统计信息：

```
SELECT * FROM sometable FORMAT JSON
```

```
{
  "meta":
  [
    {
      "name": "path",
      "type": "String"
    },
    ...
  ],

  "data":
  [
    {
      "path": "Bob_Dolman",
      "month": "2016-11-01",
      "hits": 245
    },
    ...
  ],

  "rows": 3,

  "statistics":
  {
    "elapsed": 0.000497457,
    "rows_read": 3,
    "bytes_read": 87
  }
}
```

2. JSONCompact格式将打印相同的元数据，但对数据本身使用压缩形式：

```
SELECT * FROM sometable FORMAT JSONCompact
```

```
{
  "meta":
  [
    {
      "name": "path",
      "type": "String"
    },
    ...
  ],

  "data":
  [
    ["Bob_Dolman", "2016-11-01", 245],
    ["1-krona", "2017-01-01", 4],
    ["Ahmadabad-e_Kalij-e_Sofla", "2017-01-01", 3]
  ],

  "rows": 3,

  "statistics":
  {
    "elapsed": 0.00074981,
    "rows_read": 3,
    "bytes_read": 87
  }
}
```

考虑JSONStrings或JSONCompactStrings变体将所有值编码为字符串。

导出JSON数据和结构的压缩方法

1. 获取数据及其结构的更有效的方法是使用JSONCompactEachRowWithNamesAndTypes格式：

```
SELECT * FROM sometable FORMAT JSONCompactEachRowWithNamesAndTypes
```

```
["path", "month", "hits"]
["String", "Date", "UInt32"]
["Bob_Dolman", "2016-11-01", 245]
["1-krona", "2017-01-01", 4]
["Ahmadabad-e_Kalij-e_Sofla", "2017-01-01", 3]
```

2. 这将使用紧凑的 JSON 格式，前面有两个标题行，其中包含列名和类型。然后可以使用此格式将数据导入另一个 ClickHouse 实例（或其他应用程序）。

将JSON导出到文件

1. 要将导出的 JSON 数据保存到文件，可以使用INTO OUTFILE子句：

```
SELECT * FROM sometable INTO OUTFILE 'out.json' FORMAT JSONEachRow
```

```
36838935 rows in set. Elapsed: 2.220 sec. Processed 36.84 million rows, 1.27 GB (16.60 million rows/s., 572.47 MB/s.)
```

2. ClickHouse 仅用了 2 秒就将近 3700 万条记录导出到 JSON 文件。我们还可以使用COMPRESSION子句导出以动态启用压缩：

```
SELECT * FROM sometable INTO OUTFILE 'out.json.gz' FORMAT JSONEachRow
```

```
36838935 rows in set. Elapsed: 22.680 sec. Processed 36.84 million rows, 1.27 GB (1.62 million rows/s., 56.02 MB/s.)
```

3. 需要更多时间来完成，但会生成更小的压缩文件：

```
 2.2G    out.json
576M    out.json.gz
```

处理其他JSON格式

前面加载JSON数据的示例假设使用JSONEachRow(ndjson)。我们在下面提供了加载其他常见格式的JSON的示例。

JSON 对象数组

1. JSON 数据最流行的形式之一是在 JSON 数组中拥有 JSON 对象列表，如下例所示：

```
 > cat list.json
[
  {
    "path": "Akiba_Hebrew_Academy",
    "month": "2017-08-01",
    "hits": 241
  },
  {
    "path": "Aegithina_tiphia",
    "month": "2018-02-01",
    "hits": 34
  },
  ...
]
```

2.为这种数据创建一个表：

```
 CREATE TABLE sometable
(
    `path` String,
    `month` Date,
    `hits` UInt32
)
ENGINE = MergeTree
ORDER BY tuple(month, path)
```

3. 要导入 JSON 对象列表，我们可以使用一种格式（从list.jsonJSONEachRow文件插入数据）：

```
 INSERT INTO sometable
FROM INFILE 'list.json'
FORMAT JSONEachRow
```

4. 使用FROM INFILE子句从本地文件加载数据，并且可以看到导入成功：

```
 SELECT *
FROM sometable
```

```
 ┌─path──────────────────────────────────────────────────┬──────month─┬─hits─┐
│ 1971-72_Utah_Stars_season │ 2016-10-01 │    1 │
│ Akiba_Hebrew_Academy      │ 2017-08-01 │  241 │
│ Aegithina_tiphia          │ 2018-02-01 │   34 │
└──────────────────────────────────────────────────┴────────────┴──────┘
```

处理NDJSON（行分隔的JSON）

1. 许多应用程序可以以 JSON 格式记录数据，以便每个日志行都是一个单独的 JSON 对象，就像在此文件中一样：

```
 cat object-per-line.json
```

```
 {"path":"1-krona","month":"2017-01-01","hits":4}
{"path":"Ahmadabad-e_Kalij-e_Sofla","month":"2017-01-01","hits":3}
{"path":"Bob_Dolman","month":"2016-11-01","hits":245}
```

2. 相同的JSONEachRow格式可以处理这样的文件：

```
 INSERT INTO sometable FROM INFILE 'object-per-line.json' FORMAT JSONEachRow;
SELECT * FROM sometable;
```

```
 ┌─path──────────────────────────────────────────────┬──────month─┬─hits─┐
│ Bob_Dolman                │ 2016-11-01 │  245 │
│ 1-krona                   │ 2017-01-01 │    4 │
│ Ahmadabad-e_Kalij-e_Sofla │ 2017-01-01 │    3 │
└──────────────────────────────────────────────────┴────────────┴──────┘
```

JSON对象键

1. 在某些情况下，JSON对象列表可以编码为对象属性而不是数组元素：

```
cat objects.json
```

```
{
 "a": {
   "path":"April_25,_2017",
   "month":"2018-01-01",
   "hits":2
 },
 "b": {
   "path":"Akahori_Station",
   "month":"2016-06-01",
   "hits":11
 },
 ...
}
```

2. ClickHouse 可以使用以下格式从此类数据中加载数据`JSONObjectEachRow`：

```
 INSERT INTO sometable FROM INFILE 'objects.json' FORMAT JSONObjectEachRow;
SELECT * FROM sometable;
```

```
 ┌─path──────────────────────┬──────month─┬─hits─┐
│ Abducens_palsy  │ 2016-05-01 │   28 │
│ Akahori_Station │ 2016-06-01 │   11 │
│ April_25,_2017  │ 2018-01-01 │    2 │
└─────────────────────┴────────────┴──────┘
```

指定父对象键值

1. 假设还想将父对象键中的值保存到表中。 在这种情况下，可以使用以下选项来定义要保存键值的列的名称：

```
SET format_json_object_each_row_column_for_object_name = 'id'
```

2. 现在，可以使用函数检查要从原始JSON文件中加载哪些数据`file()`：

```
SELECT * FROM file('objects.json', JSONObjectEachRow)
```

```
 ┌─id─┬─path──────────────────────┬──────month─┬─hits─┐
│ a  │ April_25,_2017  │ 2018-01-01 │    2 │
│ b  │ Akahori_Station │ 2016-06-01 │   11 │
│ c  │ Abducens_palsy  │ 2016-05-01 │   28 │
└────┴─────────────────────┴────────────┴──────┘
```

> 请注意该id列是如何正确地填充键值的。

JSON数组

1. 有时，为了节省空间，JSON文件会以数组而不是对象的形式进行编码。在这种情况下，我们处理JSON 数组列表：

```
cat arrays.json
```

```
 ["Akiba_Hebrew_Academy", "2017-08-01", 241],
["Aegithina_tiphia", "2018-02-01", 34],
["1971-72_Utah_Stars_season", "2016-10-01", 1]
```

2. 在这种情况下，ClickHouse 将加载此数据并根据数组中的顺序将每个值归属于相应的列。我们`JSONCompactEachRow`为此使用格式：

```
SELECT * FROM sometable
```

```
 ┌─c1──────────────────────┬──────────c2─┬─c3─┐
│ Akiba_Hebrew_Academy    │ 2017-08-01 │ 241 │
│ Aegithina_tiphia        │ 2018-02-01 │  34 │
│ 1971-72_Utah_Stars_season │ 2016-10-01 │   1 │
└─────────────────────────┴────────────┴─────┘
```

从JSON数组导入单个列

1. 在某些情况下，数据可以按列而不是按行进行编码。在这种情况下，父 JSON 对象包含具有值的列。查看以下文件：

```
cat columns.json
```

```
{
 "path": ["2007_Copa_America", "Car_dealerships_in_the_USA", "Dihydromyricetin_reductase"],
 "month": ["2016-07-01", "2015-07-01", "2015-07-01"],
 "hits": [178, 11, 1]
}
```

2. ClickHouse使用`JSONColumns`以下格式来解析数据：

```
SELECT * FROM file('columns.json', JSONColumns)
```

```
┌─path──────────────────────┬──────month─┬─hits─┐
│ 2007_Copa_America         │ 2016-07-01 │  178 │
│ Car_dealerships_in_the_USA │ 2015-07-01 │   11 │
│ Dihydromyricetin_reductase │ 2015-07-01 │    1 │
└────────────────────────────┴────────────┴──────┘
```

3. 当处理列数组而不是使用格式的对象时，也支持更紧凑的格式`JSONCompactColumns`：

```
SELECT * FROM file('columns-array.json', JSONCompactColumns)
```

```
┌─c1──────────────────┬──────────c2─┬─c3─┐
│ Heidenrod           │ 2017-01-01 │ 10 │
│ Arthur_Henrique     │ 2016-11-01 │ 12 │
│ Alan_Ebnother       │ 2015-11-01 │ 66 │
└─────────────────────┴────────────┴────┘
```

保存JSON对象

1. 在某些情况下，您可能希望将JSON对象保存到单个String（或 JSON）列中，而不是对其进行解析。这在处理不同结构的JSON对象列表时很有用。让我们以这个文件为例，其中父列表中有多个不同JSON对象：

```
cat custom.json
```

```
[
 {"name": "Joe", "age": 99, "type": "person"},
 {"url": "/my.post.MD", "hits": 1263, "type": "post"},
 {"message": "Warning on disk usage", "type": "log"}
]
```

2.我们希望将原始 JSON 对象保存到下表中：

```
CREATE TABLE events
(
    `data` String
)
ENGINE = MergeTree
ORDER BY ()
```

3. 现在可以使用格式将文件中的数据加载到该表中`JSONAsString`以保留JSON对象而不是解析它们：

```
INSERT INTO events (data)
FROM INFILE 'custom.json'
FORMAT JSONAsString
```

4. 可以使用JSON函数来查询已保存的对象：

```
SELECT
    JSONExtractString(data, 'type') AS type,
    data
FROM events
```

```
┌─type───┬─data────────────────────────────────────────────────────────────────────┐
│ person │ {"name": "Joe", "age": 99, "type": "person"}                           │
│ post   │ {"url": "/my.post.MD", "hits": 1263, "type": "post"}                   │
│ log    │ {"message": "Warning on disk usage", "type": "log"}                    │
└────────┴────────────────────────────────────────────────────────────────────────┘
```

JSONAsString请注意，如果我们有JSON对象每行格式的文件（通常与JSONEachRow格式一起使用），那么它可以完美地工作。

嵌套对象的架构

在处理嵌套 JSON 对象的情况下，我们可以另外定义模式并使用复杂类型（Array、Object Data Type或Tuple）来加载数据：

```
SELECT *
FROM file('list-nested.json', JSONEachRow, 'page Tuple(path String, title String, owner_id UInt16), month Date, hits UInt32')
LIMIT 1
```

```
┌─page──────────────────────────────────────────┬──────month─┬─hits─┐
│ ('Akiba_Hebrew_Academy','Akiba Hebrew Academy',12) │ 2017-08-01 │  241 │
└───────────────────────────────────────────────┴────────────┴──────┘
```

访问嵌套的JSON对象

1. 可以通过启用以下设置选项来引用嵌套的JSON键：

```
SET input_format_import_nested_json = 1
```

2. 这能够使用点符号引用嵌套的JSON对象键（记住用反引号符号包装它们才能起作用）：

```
SELECT *
FROM file('list-nested.json', JSONEachRow, '`page.owner_id` UInt32, `page.title` String, month Date, hits UInt32')
LIMIT 1
```

```
┌─page.owner_id─┬─page.title─────────┬──────month─┬─hits─┐
│            12 │ Akiba Hebrew Academy │ 2017-08-01 │  241 │
└───────────────┴────────────────────┴────────────┴──────┘
```

这样，可以展平嵌套的JSON对象或使用一些嵌套的值将它们保存为单独的列。

跳过未知列

1. 默认情况下，ClickHouse在导入JSON数据时会忽略未知列。我们来尝试将原始文件导入到没有该列的表中month：

```
CREATE TABLE shorttable
(
   `path` String,
   `hits` UInt32
)
ENGINE = MergeTree
ORDER BY path
```

2. 这样仍然可以将原始的包含3列的JSON数据插入到该表中：

```
INSERT INTO shorttable FROM INFILE 'list.json' FORMAT JSONEachRow;
SELECT * FROM shorttable
```

```
┌─path──────────────────────────────────────────┬─hits─┐
│ 1971-72_Utah_Stars_season │    1 │
│ Aegithina_tiphia          │   34 │
│ Akiba_Hebrew_Academy      │  241 │
└───────────────────────────────────────────────┴──────┘
```

3. ClickHouse将在导入时忽略未知列。可以使用input_format_skip_unknown_fields设置选项禁用此功能：

```
 SET input_format_skip_unknown_fields = 0;
INSERT INTO shorttable FROM INFILE 'list.json' FORMAT JSONEachRow;
```

```
 Ok.
Exception on client:
Code: 117. DB::Exception: Unknown field found while parsing JSONEachRow format: month: (in file/uri /data/clickhouse/user_files/list.json): (at row 1)
```

3. 如果 JSON 和表列结构不一致，ClickHouse将会抛出异常。

BSON

ClickHouse 允许导出和导入BSON编码文件的数据。一些DBMS使用此格式，例如MongoDB数据库。

1. 要导入BSON数据，使用BSONEachRow格式。从这个BSON文件导入数据：

```
 SELECT * FROM file('data.bson', BSONEachRow)
```

```
 ┌─path──────────────────────────────────────────────┬─month─┬─hits─┐
│ Bob_Dolman                                        │ 17106 │  245 │
│ 1-krona                                           │ 17167 │    4 │
│ Ahmadabad-e_Kalij-e_Sofla                         │ 17167 │    3 │
└───────────────────────────────────────────────────┴───────┴──────┘
```

2. 还可以使用相同的格式导出到BSON文件：

```
 SELECT *
FROM sometable
INTO OUTFILE 'out.bson'
FORMAT BSONEachRow
```

此后，我们将数据导出到`out.bson`文件。

对JSON进行建模的其他方法

使用嵌套

Nested类型可用于对很少发生变化的静态对象进行建模，从而为`Tuple`和提供替代方案`Array(Tuple)`。我们通常建议避免对JSON使用此类型，因为它的行为通常令人困惑。主要好处`Nested`是子列可用于对键进行排序。

下面提供了一个使用Nested类型对静态对象进行建模的示例。请考虑以下JSON中的简单日志条目：

```
 {
 "timestamp": 897819077,
 "clientip": "45.212.12.0",
 "request": {
   "method": "GET",
   "path": "/french/images/hm_nav_bar.gif",
   "version": "HTTP/1.0"
 },
 "status": 200,
 "size": 3305
}
``

We can declare the `request` key as `Nested`. Similar to `Tuple`, we are required to specify the sub columns.

```sql
-- default
SET flatten_nested=1
CREATE table http
(
   timestamp Int32,
   clientip     IPv4,
   request Nested(method LowCardinality(String), path String, version LowCardinality(String)),
   status       UInt16,
   size         UInt32,
) ENGINE = MergeTree() ORDER BY (status, timestamp);
```

flatten_nested

该设置`flatten_nested`控制嵌套的行为。
```

1. flatten_nested=1

值1（默认值）不支持任意嵌套级别。使用此值，最容易将嵌套数据结构视为长度相同的多个Array列。字段“方法”、“路径”和“版本”都是单独的“数组（类型）”列，它们有一个关键约束：方法、路径和版本字段的长度必须相同。如果我们使用SHOW CREATE TABLE，如下所示：

```
 SHOW CREATE TABLE http

CREATE TABLE http
(
    `timestamp` Int32,
    `clientip` IPv4,
    `request.method` Array(LowCardinality(String)),
    `request.path` Array(String),
    `request.version` Array(LowCardinality(String)),
    `status` UInt16,
    `size` UInt32
)
ENGINE = MergeTree
ORDER BY (status, timestamp)
```

下面，插入到这个表中：

```
 SET input_format_import_nested_json = 1;
INSERT INTO http
FORMAT JSONEachRow
{"timestamp":897819077,"clientip":"45.212.12.0","request":
[{"method":"GET","path":"/french/images/hm_nav_bar.gif","version":"HTTP/1.0"}],"status":200,"size":3305}
```

这里需要注意几个要点：

- 我们需要使用设置input_format_import_nested_json将JSON作为嵌套结构插入。如果没有这个，我们需要将JSON展平，即

```
 INSERT INTO http FORMAT JSONEachRow
{"timestamp":897819077,"clientip":"45.212.12.0","request":{"method":["GET"],"path":["/french/images/hm_nav_bar.gif"],"version":
["HTTP/1.0"]},"status":200,"size":3305}
```

- 嵌套字段method、 path和version需要作为 JSON 数组传递，即

```
 {
  "@timestamp": 897819077,
  "clientip": "45.212.12.0",
  "request": {
    "method": [
      "GET"
    ],
    "path": [
      "/french/images/hm_nav_bar.gif"
    ],
    "version": [
      "HTTP/1.0"
    ]
  },
  "status": 200,
  "size": 3305
}
```

可以使用点符号来查询列：

```
 SELECT clientip, status, size, `request.method` FROM http WHERE has(request.method, 'GET');

┌─clientip─────┬─status─┬─size─┬─request.method─┐
│ 45.212.12.0 │    200 │ 3305 │ ['GET']        │
└──────────────┴────────┴──────┴────────────────┘
1 row in set. Elapsed: 0.002 sec.
```

Array请注意，子列的使用意味着可以利用完整的呼吸数组函数ARRAY JOIN，包括子句 - 如果您的列有多个值，则很有用。

2. flatten_nested=0

这允许任意级别的嵌套，并且意味着嵌套的列保留为单个Tuples 数组 - 实际上它们变得相同Array(Tuple)。

这是使用 JSON 的首选方法，通常也是最简单的方法Nested。如下所示，它仅要求所有对象都是列表。

下面，我们重新创建表格并重新插入一行：

```
 CREATE TABLE http
(
    `timestamp` Int32,
    `clientip` IPv4,
    `request` Nested(method LowCardinality(String), path String, version LowCardinality(String)),
    `status` UInt16,
    `size` UInt32
)
ENGINE = MergeTree
ORDER BY (status, timestamp)

SHOW CREATE TABLE http

-- note Nested type is preserved.
CREATE TABLE default.http
(
    `timestamp` Int32,
    `clientip` IPv4,
    `request` Nested(method LowCardinality(String), path String, version LowCardinality(String)),
    `status` UInt16,
    `size` UInt32
)
ENGINE = MergeTree
ORDER BY (status, timestamp)

INSERT INTO http
FORMAT JSONEachRow
{"timestamp":897819077,"clientip":"45.212.12.0","request":
[{"method":"GET","path":"/french/images/hm_nav_bar.gif","version":"HTTP/1.0"}],"status":200,"size":3305}
```

这里需要注意几个要点：

- input_format_import_nested_json无需插入。
- 类型Nested保存在 中SHOW CREATE TABLE。此列下面实际上是Array(Tuple(Nested(method LowCardinality(String), path String, version LowCardinality(String))))
- 因此，我们需要将其插入request为数组，即：

```
 {
  "timestamp": 897819077,
  "clientip": "45.212.12.0",
  "request": [
   {
    "method": "GET",
    "path": "/french/images/hm_nav_bar.gif",
    "version": "HTTP/1.0"
   }
  ],
  "status": 200,
  "size": 3305
}
```

可以再次使用点符号来查询列：

```
 SELECT clientip, status, size, `request.method` FROM http WHERE has(request.method, 'GET');

┌─clientip─┬─status─┬─size─┬─request.method─┐
│ 45.212.12.0 │    200 │ 3305 │ ['GET']        │
└──────────┴────────┴──────┴────────────────┘
1 row in set. Elapsed: 0.002 sec.
```

使用成对数组

1. 成对数组在将JSON表示为字符串的灵活性与更结构化方法的性能之间实现了平衡。该架构非常灵活，因为任何新字段都可能添加到根中。然而，这需要更复杂的查询语法，并且与嵌套结构不兼容。

例如，请参考下表：

```
 CREATE TABLE http_with_arrays (
  keys Array(String),
  values Array(String)
)
ENGINE = MergeTree  ORDER BY tuple();
```

2. 要插入此表，需要将JSON构造为键和值的列表。以下查询说明了如何使用JSONExtractKeysAndValues来实现此目的：

```
 SELECT
   arrayMap(x -> (x.1), JSONExtractKeysAndValues(json, 'String')) AS keys,
   arrayMap(x -> (x.2), JSONExtractKeysAndValues(json, 'String')) AS values
FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/http/documents-01.ndjson.gz', 'JSONAsString')
LIMIT 1
FORMAT Vertical

Row 1:
──────
keys:   ['@timestamp','clientip','request','status','size']
values: ['893964617','40.135.0.0','{"method":"GET","path":"/images/hm_bg.jpg","version":"HTTP/1.0"}','200','24736']

1 row in set. Elapsed: 0.416 sec.
```

3. 请注意，请求列仍然是一个以字符串表示的嵌套结构。我们可以向根插入任何新键。我们还可以在JSON本身中存在任意差异。要插入到我们的本地表中，请执行以下操作：

```
 INSERT INTO http_with_arrays
SELECT
   arrayMap(x -> (x.1), JSONExtractKeysAndValues(json, 'String')) AS keys,
   arrayMap(x -> (x.2), JSONExtractKeysAndValues(json, 'String')) AS values
FROM s3('https://datasets-documentation.s3.eu-west-3.amazonaws.com/http/documents-01.ndjson.gz', 'JSONAsString')

0 rows in set. Elapsed: 12.121 sec. Processed 10.00 million rows, 107.30 MB (825.01 thousand rows/s., 8.85 MB/s.)
```

4. 查询此结构需要使用indexOf函数来识别所需键的索引（应与值的顺序一致）。这可用于访问值数组列，即values[indexOf(keys, 'status')]。我们仍然需要请求列的JSON解析方法 - 在本例中为simpleJSONExtractString。

```
 SELECT toUInt16(values[indexOf(keys, 'status')])                    as status,
     simpleJSONExtractString(values[indexOf(keys, 'request')], 'method') as method,
     count()                                                         as c
FROM http_with_arrays
WHERE status >= 400
 AND toDateTime(values[indexOf(keys, '@timestamp')]) BETWEEN '1998-01-01 00:00:00' AND '1998-06-01 00:00:00'
GROUP by method, status ORDER BY c DESC LIMIT 5;

┌─status─┬─method─┬─────c─┐
│    404 │ GET    │ 11267 │
│    404 │ HEAD   │   276 │
│    500 │ GET    │   160 │
│    500 │ POST   │   115 │
│    400 │ GET    │    81 │
└────────┴────────┴───────┘

5 rows in set. Elapsed: 0.383 sec. Processed 8.22 million rows, 1.97 GB (21.45 million rows/s., 5.15 GB/s.)
Peak memory usage: 51.35 MiB.
```

从Kafka同步数据

从Kafka到ClickHouse

首先，我们关注最常见的用例：使用Kafka表引擎将数据从Kafka插入ClickHouse。

Kafka表引擎允许ClickHouse直接从Kafka主题读取数据。虽然该引擎对于查看主题消息很有用，但其设计仅允许一次性检索，即当向表发出查询时，它会从队列中使用数据并增加消费者偏移量，然后再将结果返回给调用者。实际上，如果不重置这些偏移量，就无法重新读取数据。

为了从表引擎读取中持久保存这些数据，我们需要一种捕获数据并将其插入另一个表的方法。基于触发器的物化视图本身就提供了此功能。物化视图启动对表引擎的读取，接收批量文档。TO子句确定数据的目标 - 通常是Merge Tree系列的表。此过程如下所示：

准备

如果您已填充目标主题的数据，则可以调整以下内容以用于您的数据集。或者，此处提供了一个示例 Github数据集。此数据集在下面的示例中使用，与此处提供的完整数据集相比，它使用了简化的架构和行子集（具体而言，我们限制为与ClickHouse 存储库有关的Github 事件），以简洁起见。这仍然足以使大多数随数据集发布的查询正常工作。

配置

1. 如果您要连接到安全的Kafka，则此步骤是必需的。这些设置不能通过SQL DDL命令传递，必须在 ClickHouse config.xml中配置。我们假设您正在连接到SASL安全实例。这是与Confluent Cloud交互时最简单的方法。

```
<clickhouse>
 <kafka>
    <sasl_username>username</sasl_username>
    <sasl_password>password</sasl_password>
    <security_protocol>sasl_ssl</security_protocol>
    <sasl_mechanisms>PLAIN</sasl_mechanisms>
 </kafka>
</clickhouse>
```

2. 将上述代码片段放入 conf.d/ 目录下的新文件中，或将其合并到现有配置文件中。

3. 再创建一个名为的数据库以`KafkaEngine`供在本教程中使用：

```
CREATE DATABASE KafkaEngine;
```

4.创建数据库后，需要切换到该数据库：

```
USE KafkaEngine;
```

创建目标

准备目标表。在下面的示例中，为了简洁起见，我们使用简化的 GitHub 架构。

```
 CREATE TABLE github
(
    file_time DateTime,
    event_type Enum('CommitCommentEvent' = 1, 'CreateEvent' = 2, 'DeleteEvent' = 3, 'ForkEvent' = 4, 'GollumEvent' = 5, 'IssueCommentEvent' = 6,
'IssuesEvent' = 7, 'MemberEvent' = 8, 'PublicEvent' = 9, 'PullRequestEvent' = 10, 'PullRequestReviewCommentEvent' = 11, 'PushEvent' = 12,
'ReleaseEvent' = 13, 'SponsorshipEvent' = 14, 'WatchEvent' = 15, 'GistEvent' = 16, 'FollowEvent' = 17, 'DownloadEvent' = 18, 'PullRequestReviewEvent' =
19, 'ForkApplyEvent' = 20, 'Event' = 21, 'TeamAddEvent' = 22),
    actor_login LowCardinality(String),
    repo_name LowCardinality(String),
    created_at DateTime,
    updated_at DateTime,
    action Enum('none' = 0, 'created' = 1, 'added' = 2, 'edited' = 3, 'deleted' = 4, 'opened' = 5, 'closed' = 6, 'reopened' = 7, 'assigned' = 8, 'unassigned' =
9, 'labeled' = 10, 'unlabeled' = 11, 'review_requested' = 12, 'review_request_removed' = 13, 'synchronize' = 14, 'started' = 15, 'published' = 16, 'update'
= 17, 'create' = 18, 'fork' = 19, 'merged' = 20),
    comment_id UInt64,
    path String,
    ref LowCardinality(String),
    ref_type Enum('none' = 0, 'branch' = 1, 'tag' = 2, 'repository' = 3, 'unknown' = 4),
    creator_user_login LowCardinality(String),
    number UInt32,
    title String,
    labels Array(LowCardinality(String)),
    state Enum('none' = 0, 'open' = 1, 'closed' = 2),
    assignee LowCardinality(String),
    assignees Array(LowCardinality(String)),
    closed_at DateTime,
    merged_at DateTime,
    merge_commit_sha String,
    requested_reviewers Array(LowCardinality(String)),
    merged_by LowCardinality(String),
    review_comments UInt32,
    member_login LowCardinality(String)
) ENGINE = MergeTree ORDER BY (event_type, repo_name, created_at)
```

创建并填充主题

以下示例创建一个具有与合并树表相同架构的表引擎。这并不是严格要求的，因为您可以在目标表中拥有别名或临时列。设置很重要；但是 - 请注意使用作为`JSONEachRow`从 Kafka 主题使用 JSON 的数据类型。值`github`和`clickhouse`分别代表主题名称和消费者组名称。主题实际上可以是值列表。

```
 CREATE TABLE github_queue
(
    file_time DateTime,
    event_type Enum('CommitCommentEvent' = 1, 'CreateEvent' = 2, 'DeleteEvent' = 3, 'ForkEvent' = 4, 'GollumEvent' = 5, 'IssueCommentEvent' = 6,
'IssuesEvent' = 7, 'MemberEvent' = 8, 'PublicEvent' = 9, 'PullRequestEvent' = 10, 'PullRequestReviewCommentEvent' = 11, 'PushEvent' = 12,
'ReleaseEvent' = 13, 'SponsorshipEvent' = 14, 'WatchEvent' = 15, 'GistEvent' = 16, 'FollowEvent' = 17, 'DownloadEvent' = 18, 'PullRequestReviewEvent' =
19, 'ForkApplyEvent' = 20, 'Event' = 21, 'TeamAddEvent' = 22),
    actor_login LowCardinality(String),
    repo_name LowCardinality(String),
    created_at DateTime,
    updated_at DateTime,
    action Enum('none' = 0, 'created' = 1, 'added' = 2, 'edited' = 3, 'deleted' = 4, 'opened' = 5, 'closed' = 6, 'reopened' = 7, 'assigned' = 8, 'unassigned' =
9, 'labeled' = 10, 'unlabeled' = 11, 'review_requested' = 12, 'review_request_removed' = 13, 'synchronize' = 14, 'started' = 15, 'published' = 16, 'update'
= 17, 'create' = 18, 'fork' = 19, 'merged' = 20),
    comment_id UInt64,
    path String,
    ref LowCardinality(String),
    ref_type Enum('none' = 0, 'branch' = 1, 'tag' = 2, 'repository' = 3, 'unknown' = 4),
    creator_user_login LowCardinality(String),
    number UInt32,
    title String,
    labels Array(LowCardinality(String)),
    state Enum('none' = 0, 'open' = 1, 'closed' = 2),
    assignee LowCardinality(String),
    assignees Array(LowCardinality(String)),
    closed_at DateTime,
    merged_at DateTime,
    merge_commit_sha String,
    requested_reviewers Array(LowCardinality(String)),
    merged_by LowCardinality(String),
    review_comments UInt32,
    member_login LowCardinality(String)
)
  ENGINE = Kafka('kafka_host:9092', 'github', 'clickhouse',
        'JSONEachRow') settings kafka_thread_per_consumer = 0, kafka_num_consumers = 1;
```

我们将在下面讨论引擎设置和性能调整。此时，对表进行简单的选择应该会读取一些行。请注意，这将使消费者偏移量向前移动，从而防止在没有重置的github_queue情况下重新读取这些行。注意限制和必需参数stream_like_engine_allow_direct_select.

创建物化视图

物化视图将连接之前创建的两个表，从 Kafka 表引擎读取数据并将其插入目标合并树表。我们可以进行许多数据转换。我们将进行简单的读取和插入。使用 * 假定列名相同（区分大小写）。

```
 CREATE MATERIALIZED VIEW github_mv TO github AS
SELECT *
FROM github_queue;
```

在创建时，物化视图会连接到 Kafka 引擎并开始读取：将行插入目标表。此过程将无限期地继续，后续插入 Kafka 的消息将被使用。您可以随时重新运行插入脚本以将更多消息插入Kafka。

确认已插入数据

1. 确认目标表中存在数据：

```
 SELECT count() FROM github;
```

2.应该可以看到200,000行：

```
 ┌─count()─┐
│  200000 │
└─────────┘
```

常见操作

停止并重新启动消息

1. 要停止消息消费，可以分离 Kafka 引擎表：

```
 DETACH TABLE github_queue;
```

2. 这不会影响消费者组的偏移量。要重新开始消费并从之前的偏移量继续，请重新附加表。

```
 ATTACH TABLE github_queue;
```

添加Kafka元数据

在将原始 Kafka 消息导入 ClickHouse 后，跟踪其中的元数据会很有用。例如，我们可能想知道我们已使用了多少特定主题或分区。为此，Kafka 表引擎公开了几个虚拟列。通过修改我们的架构和物化视图的 select 语句，这些可以作为目标表中的列保留下来。

1. 首先，在向目标表添加列之前，我们执行上面描述的停止操作。

```
 DETACH TABLE github_queue;
```

2. 下面我们添加信息列来识别源主题和该行源自的分区。

```
ALTER TABLE github
 ADD COLUMN topic String,
 ADD COLUMN partition UInt64;
```

3. 接下来，我们需要确保虚拟列按要求映射。虚拟列以 为前缀。虚拟列的完整列表可在此处_找到。

要使用虚拟列更新我们的表，我们需要删除物化视图，重新连接 Kafka 引擎表，然后重新创建物化视图。

```
 DROP VIEW github_mv;
```

```
 ATTACH TABLE github_queue;
```

```
 CREATE MATERIALIZED VIEW github_mv TO github AS
SELECT *, _topic as topic, _partition as partition
FROM github_queue;
```

4. 新使用的行应该具有元数据。

```
 SELECT actor_login, event_type, created_at, topic, partition
FROM github
LIMIT 10;
```

结果如下：

actor_login	事件类型	创建时间	主题	分割
IgorMinar	提交评论事件	2011-02-12 02:22:00	github	0
queeup	提交评论事件	2011-02-12 02:23:23	github	0
IgorMinar	提交评论事件	2011-02-12 02:23:24	github	0
IgorMinar	提交评论事件	2011-02-12 02:24:50	github	0
IgorMinar	提交评论事件	2011-02-12 02:25:20	github	0
dapi	提交评论事件	2011-02-12 06:18:36	github	0
sourcerebels	提交评论事件	2011-02-12 06:34:10	github	0
jamierumbelow	提交评论事件	2011-02-12 12:21:40	github	0
jpn	提交评论事件	2011-02-12 12:24:31	github	0
Oxonium	提交评论事件	2011-02-12 12:31:28	github	0修改 Kafka 引擎设置

修改 Kafka 引擎设置

我们建议删除 Kafka 引擎表并使用新设置重新创建它。在此过程中不需要修改物化视图 - 重新创建 Kafka 引擎表后，消息消费将恢复。

调试问题

身份验证问题等错误不会在对 Kafka 引擎 DDL 的响应中报告。为了诊断问题，我们建议使用主 ClickHouse 日志文件 clickhouse-server.err.log。可以通过配置启用底层 Kafka 客户端库librdkafka的进一步跟踪日志记录。

```
 <kafka>
  <debug>all</debug>
</kafka>
```

处理格式错误的消息

Kafka 经常被用作数据的“垃圾场”。这会导致主题包含混合的消息格式和不一致的字段名称。避免这种情况，并利用 Kafka Streams 或 ksqlDB 等 Kafka 功能来确保消息在插入 Kafka 之前格式正确且一致。如果这些选项不可行，ClickHouse 有一些功能可以提供帮助。

- 将消息字段视为字符串。如果需要，可以在物化视图语句中使用函数来执行清理和转换。这不应代表生产解决方案，但可能有助于一次性提取。
- 如果您使用 JSONEachRow 格式从主题使用 JSON，请使用设置input_format_skip_unknown_fields。默认情况下，在写入数据时，如果输入数据包含目标表中不存在的列，ClickHouse 会抛出异常。但是，如果启用此选项，这些多余的列将被忽略。同样，这不是生产级解决方案，可能会让其他人感到困惑。
- 考虑设置kafka_skip_broken_messages。这要求用户指定每个块对格式错误的消息的容忍度 - 在 kafka_max_block_size 的上下文中考虑。如果超出此容忍度（以绝对消息为单位），则通常的异常行为将恢复，并且其他消息将被跳过。

传递语义和重复

Kafka 表引擎具有至少一次语义。在几种已知的罕见情况下，可能会出现重复。例如，可以从 Kafka 读取消息并成功插入 ClickHouse。在提交新的偏移量之前，与 Kafka 的连接已丢失。在这种情况下需要重试该块。可以使用分布式表或 ReplicatedMergeTree 作为目标表对块进行重复数据删除。虽然这减少了重复行的可能性，但它依赖于相同的块。诸如 Kafka 重新平衡之类的事件可能会使此假设无效，从而在极少数情况下导致重复。

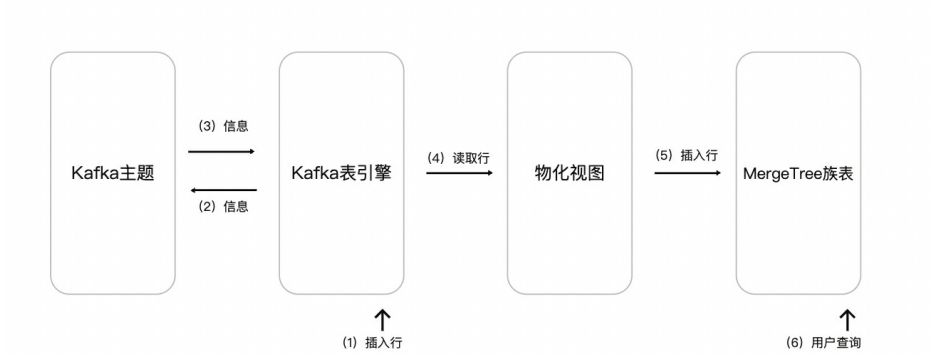
基于Quorum的插入数据

在 ClickHouse 中需要更高交付保证的情况下，您可能需要基于仲裁的插入。这无法在物化视图或目标表上设置。但是，可以为用户配置文件设置它，例如：

```
 <profiles>
 <default>
  <insert_quorum>2</insert_quorum>
 </default>
</profiles>
```

从ClickHouse到Kafka

尽管使用情况较少，但 ClickHouse 数据也可以保存在 Kafka 中。例如，我们将手动将行插入 Kafka 表引擎。该数据将由同一 Kafka 引擎读取，其物化视图将数据放入 Merge Tree 表中。最后，我们演示了在 Kafka 插入中应用物化视图从现有源表中读取表。体现为：



步骤

直接插入行

1. 首先，确认目标表的数量。

```
SELECT count() FROM github;
```

您应该有 200,000 行：

```
┌─count()─┐
│  200000 │
└─────────┘
```

2. 现在将 GitHub 目标表中的行插入回 Kafka 表引擎 github_queue。请注意我们如何利用 JSONEachRow 格式并将选择限制为 100。

```
INSERT INTO github_queue SELECT * FROM github LIMIT 100 FORMAT JSONEachRow
```

3.重新计算 GitHub 中的行数以确认它已增加 100。如上图所示，行已通过 Kafka 表引擎插入 Kafka，然后由同一引擎重新读取并通过我们的物化视图插入到 GitHub 目标表中。

```
SELECT count() FROM github;
```

您应该会看到另外 100 行：

```
┌─count()─┐
│  200100 │
└─────────┘
```

使用物化视图

1. 当文档插入表中时，我们可以利用物化视图将消息推送到 Kafka 引擎（和主题）。当行插入 GitHub 表时，会触发物化视图，这会导致行重新插入 Kafka 引擎和新主题。再次说明这一点：

2. 创建一个新的 Kafka 主题`github_out`或等效主题。确保 Kafka 表引擎`github_out_queue`指向此主题。

```
 CREATE TABLE github_out_queue
(
    file_time DateTime,
    event_type Enum('CommitCommentEvent' = 1, 'CreateEvent' = 2, 'DeleteEvent' = 3, 'ForkEvent' = 4, 'GollumEvent' = 5, 'IssueCommentEvent' = 6,
'IssuesEvent' = 7, 'MemberEvent' = 8, 'PublicEvent' = 9, 'PullRequestEvent' = 10, 'PullRequestReviewCommentEvent' = 11, 'PushEvent' = 12,
'ReleaseEvent' = 13, 'SponsorshipEvent' = 14, 'WatchEvent' = 15, 'GistEvent' = 16, 'FollowEvent' = 17, 'DownloadEvent' = 18, 'PullRequestReviewEvent' =
19, 'ForkApplyEvent' = 20, 'Event' = 21, 'TeamAddEvent' = 22),
    actor_login LowCardinality(String),
    repo_name LowCardinality(String),
    created_at DateTime,
    updated_at DateTime,
    action Enum('none' = 0, 'created' = 1, 'added' = 2, 'edited' = 3, 'deleted' = 4, 'opened' = 5, 'closed' = 6, 'reopened' = 7, 'assigned' = 8, 'unassigned' =
9, 'labeled' = 10, 'unlabeled' = 11, 'review_requested' = 12, 'review_request_removed' = 13, 'synchronize' = 14, 'started' = 15, 'published' = 16, 'update'
= 17, 'create' = 18, 'fork' = 19, 'merged' = 20),
    comment_id UInt64,
    path String,
    ref LowCardinality(String),
    ref_type Enum('none' = 0, 'branch' = 1, 'tag' = 2, 'repository' = 3, 'unknown' = 4),
    creator_user_login LowCardinality(String),
    number UInt32,
    title String,
    labels Array(LowCardinality(String)),
    state Enum('none' = 0, 'open' = 1, 'closed' = 2),
    assignee LowCardinality(String),
    assignees Array(LowCardinality(String)),
    closed_at DateTime,
    merged_at DateTime,
    merge_commit_sha String,
    requested_reviewers Array(LowCardinality(String)),
    merged_by LowCardinality(String),
    review_comments UInt32,
    member_login LowCardinality(String)
)
  ENGINE = Kafka('host:port', 'github_out', 'clickhouse_out',
           'JSONEachRow') settings kafka_thread_per_consumer = 0, kafka_num_consumers = 1;
```

3. 现在创建一个新的物化视图`github_out_mv`以指向 GitHub 表，并在触发时将行插入上述引擎。因此，GitHub 表的添加内容将被推送到我们的新 Kafka 主题。

```
 CREATE MATERIALIZED VIEW github_out_mv TO github_out_queue AS
SELECT file_time, event_type, actor_login, repo_name,
       created_at, updated_at, action, comment_id, path,
       ref, ref_type, creator_user_login, number, title,
       labels, state, assignee, assignees, closed_at, merged_at,
       merge_commit_sha, requested_reviewers, merged_by,
       review_comments, member_login
FROM github
FORMAT JsonEachRow;
```

4. 如果您将原始 github 主题（作为Kafka 到 ClickHouse的一部分创建）插入，文档将神奇地出现在“github_clickhouse”主题中。使用原生 Kafka 工具确认这一点。例如，下面，我们使用kcat将 100 行插入到Confluent Cloud 托管主题的 github 主题中：

```
 head -n 10 github_all_columns.ndjson |
kcat -P \
  -b <host>:<port> \
  -t github
  -X security.protocol=sasl_ssl \
  -X sasl.mechanisms=PLAIN \
  -X sasl.username=<username> \
  -X sasl.password=<password>
```

阅读该`github_out`主题应该可以确认消息已送达。

```
 kcat -C \
  -b <host>:<port> \
  -t github_out \
  -X security.protocol=sasl_ssl \
  -X sasl.mechanisms=PLAIN \
  -X sasl.username=<username> \
  -X sasl.password=<password> \
  -e -q |
wc -l
```

注意事项

1. 通过 Kafka 消费者组，多个 ClickHouse 实例可以从同一主题读取数据。每个消费者将以 1:1 的映射分配到主题分区。使用 Kafka 表引擎扩展 ClickHouse 消费时，请考虑集群中的消费者总数不能超过主题上的分区数。因此，请确保提前为主题适当配置分区。
2. 可以将多个 ClickHouse 实例全部配置为使用相同的消费者组 ID（在创建 Kafka 表引擎时指定）从主题读取数据。因此，每个实例将从一个或多个分区读取数据，并将段插入到其本地目标表中。反过来，可以将目标表配置为使用 ReplicatedMergeTree 来处理数据重复。只要有足够的 Kafka 分区，这种方法就可以使用 ClickHouse 集群扩展 Kafka 读取。
3. 在寻求提高 Kafka 引擎表吞吐量性能时，请考虑以下几点：

- 性能将根据消息大小、格式和目标表类型而有所不同。单个表引擎上 100k 行/秒应该被认为是可以实现的。默认情况下，消息以块的形式读取，由参数 kafka_max_block_size 控制。默认情况下，它设置为 max_insert_block_size ，默认为 1,048,576。除非消息非常大，否则几乎总是应该增加这个值。500k 到 1M 之间的值并不罕见。测试并评估对吞吐量性能的影响。
- 可以使用 kafka_num_consumers 增加表引擎的消费者数量。但是，默认情况下，除非 kafka_thread_per_consumer 从默认值 1 更改，否则插入将在单个线程中线性化。将其设置为 1 以确保并行执行刷新。请注意，创建具有 N 个消费者（并且 kafka_thread_per_consumer=1）的 Kafka 引擎表在逻辑上等同于创建 N 个 Kafka 引擎，每个引擎都有一个物化视图并且 kafka_thread_per_consumer=0。
- 增加消费者并非免费操作。每个消费者都维护自己的缓冲区和线程，这会增加服务器的开销。请意识到消费者的开销，并尽可能先在整个集群中线性扩展。
- 如果 Kafka 消息的吞吐量是可变的并且延迟是可以接受的，请考虑增加 stream_flush_interval_ms 以确保刷新更大的块。
- background_message_broker_schedule_pool_size设置执行后台任务的线程数。这些线程用于 Kafka 流式传输。此设置在 ClickHouse 服务器启动时应用，无法在用户会话中更改，默认为 16。如果您在日志中看到超时，则可能需要增加此值。
- 为了与 Kafka 通信，使用了 librdkafka 库，该库本身会创建线程。因此，大量的 Kafka 表或消费者可能会导致大量的上下文切换。要么将此负载分散到整个集群中，尽可能只复制目标表，要么考虑使用表引擎从多个主题读取 - 支持值列表。可以从单个表中读取多个物化视图，每个视图都过滤特定主题的数据。

4. Kafka_max_wait_ms-重试前从Kafka读取消息的等待时间（毫秒）。设置为用户配置文件级别，默认为5000。

来自底层librdkafka的所有设置也可以放置在kafka元素内的ClickHouse配置文件中——设置名称应该是XML元素，用下划线替换句点，例如：

```
 <clickhouse>
  <kafka>
     <enable_ssl_certificate_verification>false</enable_ssl_certificate_verification>
  </kafka>
</clickhouse>
```

从Flink导入

使用Flink ClickHouse 连接器进行导入。

表一 连接器选择

选项		默认	类型	描述
网址	必填	none	String	格式的 ClickHouse jdbc url clickhouse://<host>:<port>
用户名	选填	none	String	如果指定了“用户名”和“密码”，则必须同时指定。
密码	选填	none	String	ClickHouse 密码。
数据库名称	选填	default	String	ClickHouse 数据库名称。
表名	必填	none	String	ClickHouse 表名。
使用本地	选填	false	Boolean	在分布式表引擎的情况下直接读取/写入本地表。
sink.flush 间隔	选填	1000	Integer	最大刷新大小，超过此大小将会刷新数据。
sink.flush 间隔	选填	1s	Duration	在此刷新间隔时间内，异步线程将刷新数据。
sink.max-重试次数	选填	3	Integer	将记录写入数据库失败时的最大重试次数。
sink.update 策略	选填	update	String	将 UPDATE_AFTER 类型的记录转换为更新/插入语句或者直接丢弃它，可用：更新、插入、丢弃。
sink.partition-策略	选填	balanced	String	分区策略：平衡（循环）、哈希（分区键）、随机（随机）。
接收器.分区键	选填	none	String	用于哈希策略的分区键。
接收器分片使用表定义	选填	false	Boolean	分片策略与分布式表定义一致，若设置为true，则会覆盖sink.partition-strategy和sink.partition-key 的配置。
sink.ignore-删除	选填	true	Integer	是否忽略删除语句。
接收器并行性	选填	none	String	为接收器定义自定义并行性。
扫描.分区.列	选填	none	Integer	用于对输入进行分区的列名。
扫描分区号	选填	none	Long	分区的数量。
扫描分区下限	选填	none	Long	第一个分区的最小值。
是否忽略主键	选填	true	Boolean	使用 ClickHouseCatalog 创建表时是否忽略主键。
特性	选填	none	String	这可以设置并传递clickhouse-jdbc配置。
查找缓存	选填	none	String	该查询表的缓存策略，包括NONE和PARTIAL（暂不支持FULL）
查找部分缓存访问后过期	选填	none	Duration	访问后缓存中的条目过期的持续时间，超过此时间，最旧的行将会过期。
查找部分缓存写入后过期	选填	none	Duration	写入后缓存中的条目过期的持续时间，超过此时间，最旧的行将会过期。
查找缓存最大行数	选填	none	Long	查找缓存的最大行数，超过此值，最旧的行将会过期。
查找部分缓存缺少键	选填	true	Boolean	标记缓存丢失的密钥，默认为 true
查找最大重试数	选填	3	Integer	查找数据库失败时的最大重试次数。

更新/删除数据注意事项：

1. 分布式表不支持更新/删除语句，如果要使用更新/删除报表，请确保将记录写入本地表或将use local设置为true。

2. 数据由主键更新和删除，在分区表中使用时请注意这一点。

数据类型映射

Flink 类型	ClickHouse 类型
CHAR	String
VARCHAR	String / IP / UUID
STRING	String / Enum
BOOLEAN	UInt8
BYTES	FixedString
DECIMAL	Decimal / Int128 / Int256 / UInt64 / UInt128 / UInt256
TINYINT	Int8
SMALLINT	Int16 / UInt8
INTEGER	Int32 / UInt16 / Interval
BIGINT	Int64 / UInt32
FLOAT	Float32
DOUBLE	Float64
DATE	Date
TIME	DateTime
TIMESTAMP	DateTime
TIMESTAMP_LTZ	DateTime
INTERVAL_YEAR_MONTH	Int32
INTERVAL_DAY_TIME	Int64
ARRAY	Array
MAP	Map
ROW	Not supported
MULTISET	Not supported
RAW	Not supported

Maven依赖关系

该项目未发布到maven中央存储库，在使用之前，需要部署/安装到自己的存储库，步骤如下：

```
  # clone the project
git clone https://github.com/itinycheng/flink-connector-clickhouse.git

 # enter the project directory
cd flink-connector-clickhouse/

 # display remote branches
git branch -r

 # checkout the branch you need
git checkout $branch_name

 # install or deploy the project to our own repository
mvn clean install -DskipTests
mvn clean deploy -DskipTests
```

```
 <dependency>
   <groupId>org.apache.flink</groupId>
   <artifactId>flink-connector-clickhouse</artifactId>
   <version>1.16.0-SNAPSHOT</version>
</dependency>
```

使用步骤

1. 创建和读/写表：

```
 -- register a clickhouse table `t_user` in flink sql.
CREATE TABLE t_user (
    `user_id` BIGINT,
    `user_type` INTEGER,
    `language` STRING,
    `country` STRING,
    `gender` STRING,
    `score` DOUBLE,
    `list` ARRAY<STRING>,
    `map` Map<STRING, BIGINT>,
    PRIMARY KEY (`user_id`) NOT ENFORCED
) WITH (
    'connector' = 'clickhouse',
    'url' = 'clickhouse://{ip}:{port}',
    'database-name' = 'tutorial',
    'table-name' = 'users',
    'sink.batch-size' = '500',
    'sink.flush-interval' = '1000',
    'sink.max-retries' = '3'
);

-- read data from clickhouse
SELECT user_id, user_type from t_user;

-- write data into the clickhouse table from the table `T`
INSERT INTO t_user
SELECT cast(`user_id` as BIGINT), `user_type`, `lang`, `country`, `gender`, `score`, ARRAY['CODER', 'SPORTSMAN'], CAST(MAP['BABA', cast(10 as BIGINT),
'NIO', cast(8 as BIGINT)] AS MAP<STRING, BIGINT>) FROM T;
```

2. 创建和使用ClickHouseCatalog:

- Scala

```
 val tEnv = TableEnvironment.create(setting)

val props = new util.HashMap[String, String]()
props.put(ClickHouseConfig.DATABASE_NAME, "default")
props.put(ClickHouseConfig.URL, "clickhouse://127.0.0.1:8123")
props.put(ClickHouseConfig.USERNAME, "username")
props.put(ClickHouseConfig.PASSWORD, "password")
props.put(ClickHouseConfig.SINK_FLUSH_INTERVAL, "30s")
val cHcatalog = new ClickHouseCatalog("clickhouse", props)
tEnv.registerCatalog("clickhouse", cHcatalog)
tEnv.useCatalog("clickhouse")

tEnv.executeSql("insert into `clickhouse`.`default`.`t_table` select...");
```

- Java

```
 TableEnvironment tEnv = TableEnvironment.create(setting);

Map<String, String> props = new HashMap<>();
props.put(ClickHouseConfig.DATABASE_NAME, "default")
props.put(ClickHouseConfig.URL, "clickhouse://127.0.0.1:8123")
props.put(ClickHouseConfig.USERNAME, "username")
props.put(ClickHouseConfig.PASSWORD, "password")
props.put(ClickHouseConfig.SINK_FLUSH_INTERVAL, "30s");
Catalog cHcatalog = new ClickHouseCatalog("clickhouse", props);
tEnv.registerCatalog("clickhouse", cHcatalog);
tEnv.useCatalog("clickhouse");

tEnv.executeSql("insert into `clickhouse`.`default`.`t_table` select...");
```

- SQL

```
 > CREATE CATALOG clickhouse WITH (
   'type' = 'clickhouse',
   'url' = 'clickhouse://127.0.0.1:8123',
   'username' = 'username',
   'password' = 'password',
   'database-name' = 'default',
   'use-local' = 'false',
   ...
);

> USE CATALOG clickhouse;
> SELECT user_id, user_type FROM `default`.`t_user` limit 10;
> INSERT INTO `default`.`t_user` SELECT ...;
```

从MySQL导入和同步

本页介绍了MySQL与ClickHouse集成的两个方面：

- 使用MySQL表引擎，从MySQL表中读取
- 使用MaterializedMySQL数据库引擎，将MySQL中的数据库与ClickHouse中的数据库同步

使用MySQL表引擎将ClickHouse连接到MySQL

MySQL表引擎允许您将ClickHouse连接到MySQL。SELECT和INSERT语句可以在ClickHouse或MySQL表中执行。本文说明了如何使用MySQL表引擎的基本方法。

配置MySQL

1. 在MySQL中创建一个数据库：

```
 CREATE DATABASE db1;
```

2. 创建表：

```
 CREATE TABLE db1.table1 (
  id INT,
  column1 VARCHAR(255)
);
```

3. 插入示例行：

```
 INSERT INTO db1.table1
  (id, column1)
VALUES
  (1, 'abc'),
  (2, 'def'),
  (3, 'ghi');
```

4.创建一个用户以从ClickHouse连接：

```
 CREATE USER 'mysql_clickhouse'@'%' IDENTIFIED BY 'Password123!';
```

5.根据需要授予特权。（出于演示目的，mysql_clickhouse用户被授予管理员权限。）

```
 GRANT ALL PRIVILEGES ON *.* TO 'mysql_clickhouse'@'%';
```

在ClickHouse中定义表

1. 创建一个使用MySQL表引擎的ClickHouse表：

```
 CREATE TABLE mysql_table1 (
  id UInt64,
  column1 String
)
ENGINE = MySQL('mysql-host.domain.com','db1','table1','mysql_clickhouse','Password123!')
```

表一 最小参数

参数	描述	举例
host	主机名或 IP	mysql-host.domain.com
database	mysql 数据库名称	db1
table	mysql 表名	table1
user	连接 mysql 的用户名	mysql_clickhouse
password	连接mysql的密码	密码123！

测试集成

1. 在MySQL中，插入一个示例行：

```
 INSERT INTO db1.table1
 (id, column1)
VALUES
 (4, 'jkl');
```

2. 请注意，MySQL表中的现有行以及您刚才添加的新行都在ClickHouse表中：

```
 SELECT
   id,
   column1
FROM mysql_table1
```

您可以看有四行：

```
 Query id: 6d590083-841e-4e95-8715-ef37d3e95197

┌─id─┬─column1─┐
│  1 │ abc     │
│  2 │ def     │
│  3 │ ghi     │
│  4 │ jkl     │
└────┴─────────┘

4 rows in set. Elapsed: 0.044 sec.
```

3. 在ClickHouse表中添加一行：

```
 INSERT INTO mysql_table1
 (id, column1)
VALUES
 (5,'mno')
```

4. 请注意，MySQL中出现了新行：

```
 mysql> select id,column1 from db1.table1;
```

可以看到新的一行：

```
 +------+---------+
| id   | column1 |
+------+---------+
|    1 | abc     |
|    2 | def     |
|    3 | ghi     |
|    4 | jkl     |
|    5 | mno     |
+------+---------+
5 rows in set (0.01 sec)
```

在ClickHouse中复制MySQL数据库

MaterializedMySQL数据库引擎允许您在ClickHouse中定义一个数据库，该数据库包含MySQL数据库中的所有现有表以及这些表中的所有数据。在MySQL方面，DDL和DML操作可以继续进行，ClickHouse检测到这些更改并充当MySQL数据库的副本。

本文演示了如何配置MySQL和ClickHouse来实现。

配置MySQL

1. 配置MySQL数据库以允许复制和本机身份验证。ClickHouse仅适用于本机密码身份验证。将以下条目添加到/etc/my.cnf：

```
default_authentication_plugin = mysql_native_password
gtid_mode = ON
enforce_gtid_consistency = ON
```

2. 创建一个用户从ClickHouse连接：

```
CREATE USER clickhouse_user IDENTIFIED BY 'ClickHouse_123';
```

3. 向新用户授予所需的权限。出于演示目的，此处授予了完全管理员权限：

```
GRANT ALL PRIVILEGES ON *.* TO 'clickhouse_user'@'%';
```

4. 在MySQL中创建一个数据库：

```
CREATE DATABASE db1;
```

5. 创建一个表：

```
CREATE TABLE db1.table_1 (
  id INT,
  column1 VARCHAR(10),
  PRIMARY KEY (`id`)
) ENGINE = InnoDB;
```

6. 插入一些示例行：

```
INSERT INTO db1.table_1
 (id, column1)
VALUES
 (1, 'abc'),
 (2, 'def'),
 (3, 'ghi');
```

配置ClickHouse

1. 设置参数：

```
set allow_experimental_database_materialized_mysql = 1;
```

2. 创建一个使用MaterializedMySQL数据库引擎的数据库：

```
CREATE DATABASE db1_mysql
ENGINE = MaterializedMySQL(
 'mysql-host.domain.com:3306',
 'db1',
 'clickhouse_user',
 'ClickHouse_123'
);
```

表二 参数说明

参数	描述	举例
host	主机名或 IP	mysql-host.domain.com
database	mysql 数据库名称	db1
table	mysql 表名	table1
user	连接 mysql 的用户名	mysql_clickhouse
password	连接mysql的密码	密码123！

测试集成

1. 在MySQL中，插入一个示例行：

```
 INSERT INTO db1.table_1
 (id, column1)
VALUES
 (4, 'jkl');
```

2. 请注意，ClickHouse表中出现了新行：

```
 SELECT
   id,
   column1
FROM db1_mysql.table_1
```

响应如下：

```
 Query id: d61a5840-63ca-4a3d-8fac-c93235985654

┌─id─┬─column1─┐
│  1 │ abc     │
└────┴─────────┘
┌─id─┬─column1─┐
│  4 │ jkl     │
└────┴─────────┘
┌─id─┬─column1─┐
│  2 │ def     │
└────┴─────────┘
┌─id─┬─column1─┐
│  3 │ ghi     │
└────┴─────────┘

4 rows in set. Elapsed: 0.030 sec.
```

3.假设MySQL中的表被修改了。在MySQL中为db1.table_1添加一列：

```
 alter table db1.table_1 add column column2 varchar(10) after column1;
```

4. 在修改后的表中插入一行：

```
 INSERT INTO db1.table_1
 (id, column1, column2)
VALUES
 (5, 'mno', 'pqr');
```

5.ClickHouse中的表格现在有了新列和新行：

```
 SELECT
   id,
   column1,
   column2
FROM db1_mysql.table_1
```

前几行第2列将为NULL：

```
Query id: 2c32fd15-3c83-480b-9bfc-cba5d932d674

Connecting to localhost:9000 as user default.
Connected to ClickHouse server version 22.2.2 revision 54455.

┌─id─┬─column1─┬─column2─┐
│  3 │ ghi     │ NULL    │
└────┴─────────┴─────────┘
┌─id─┬─column1─┬─column2─┐
│  2 │ def     │ NULL    │
└────┴─────────┴─────────┘
┌─id─┬─column1─┬─column2─┐
│  1 │ abc     │ NULL    │
│  5 │ mno     │ pqr     │
└────┴─────────┴─────────┘
┌─id─┬─column1─┬─column2─┐
│  4 │ jkl     │ NULL    │
└────┴─────────┴─────────┘

5 rows in set. Elapsed: 0.017 sec.
```

从Spark导入

将Apache Spark与ClickHouse集成

连接Apache Spark和ClickHouse有两种主要方式：

- Spark连接器-Spark连接器实现了DataSourceV2，并具有自己的目录管理。截至今天，这是集成ClickHouse和Spark的推荐方式。
- Spark JDBC-使用JDBC数据源集成Spark和ClickHouse。

Spark连接器

此连接器利用ClickHouse特定的优化，如高级分区和谓词下推，来提高查询性能和数据处理。该连接器基于ClickHouse的官方JDBC连接器，并管理自己的目录。

必要条件

- Java 8 or 17
- Scala 2.12 or 2.13
- Apache Spark 3.3 or 3.4 or 3.5

兼容性

版本	兼容的 Spark 版本	ClickHouse JDBC 版本
main	Spark 3.3, 3.4, 3.5	0.6.3
0.8.0	Spark 3.3, 3.4, 3.5	0.6.3
0.7.3	Spark 3.3, 3.4	0.4.6
0.6.0	Spark 3.3	0.3.2-patch11
0.5.0	Spark 3.2, 3.3	0.3.2-patch11
0.4.0	Spark 3.2, 3.3	Not depend on
0.3.0	Spark 3.2, 3.3	Not depend on
0.2.1	Spark 3.2	Not depend on
0.1.2	Spark 3.2	Not depend on

下载库

二进制JAR的名称模式是：

```
clickhouse-spark-runtime-${spark_binary_version}_${scala_binary_version}-${version}.jar
```

您可以在Maven中央存储库中找到所有可用的已发布JAR，在Sonatype OSS快照存储库中可以找到所有每日构建的SNAPSHOT JAR。

作为依赖项导入

- Gradle

```
 dependencies {
  implementation("com.clickhouse.spark:clickhouse-spark-runtime-{{ spark_binary_version }}_{{ scala_binary_version }}:{{ stable_version }}")
  implementation("com.clickhouse:clickhouse-jdbc:{{ clickhouse_jdbc_version }}:all") { transitive = false }
}
```

如果要使用SNAPSHOT版本，请添加以下存储库：

```
 repositries {
  maven { url = "https://s01.oss.sonatype.org/content/repositories/snapshots" }
}
```

- Maven

```
 <dependency>
  <groupId>com.clickhouse.spark</groupId>
  <artifactId>clickhouse-spark-runtime-{{ spark_binary_version }}_{{ scala_binary_version }}</artifactId>
  <version>{{ stable_version }}</version>
</dependency>
<dependency>
  <groupId>com.clickhouse</groupId>
  <artifactId>clickhouse-jdbc</artifactId>
  <classifier>all</classifier>
  <version>{{ clickhouse_jdbc_version }}</version>
  <exclusions>
    <exclusion>
      <groupId>*</groupId>
      <artifactId>*</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

如果要使用SNAPSHOT版本，请添加以下存储库。

```
 <repositories>
  <repository>
    <id>sonatype-oss-snapshots</id>
    <name>Sonatype OSS Snapshots Repository</name>
    <url>https://s01.oss.sonatype.org/content/repositories/snapshots</url>
  </repository>
</repositories>
```

使用Spark SQL

注意：对于仅使用SQL的用例，建议将Apache Kyuubi用于生产环境。

启动Spark SQL命令行界面

```
 $SPARK_HOME/bin/spark-sql \
  --conf spark.sql.catalog.clickhouse=com.clickhouse.spark.ClickHouseCatalog \
  --conf spark.sql.catalog.clickhouse.host=${CLICKHOUSE_HOST:-127.0.0.1} \
  --conf spark.sql.catalog.clickhouse.protocol=http \
  --conf spark.sql.catalog.clickhouse.http_port=${CLICKHOUSE_HTTP_PORT:-8123} \
  --conf spark.sql.catalog.clickhouse.user=${CLICKHOUSE_USER:-default} \
  --conf spark.sql.catalog.clickhouse.password=${CLICKHOUSE_PASSWORD:-} \
  --conf spark.sql.catalog.clickhouse.database=default \
  --jars /path/clickhouse-spark-runtime-{{ spark_binary_version }}_{{ scala_binary_version }}:{{ stable_version }}.jar,/path/clickhouse-jdbc-{{
clickhouse_jdbc_version }}-all.jar
```

以下论点：

```
   --jars /path/clickhouse-spark-runtime-{{ spark_binary_version }}_{{ scala_binary_version }}:{{ stable_version }}.jar,/path/clickhouse-jdbc-{{
clickhouse_jdbc_version }}-all.jar
```

可以替换为以下代码，以避免将JAR复制到Spark客户端节点。

```
   --repositories https://{maven-cental-mirror or private-nexus-repo} \
  --packages com.clickhouse.spark:clickhouse-spark-runtime-{{ spark_binary_version }}_{{ scala_binary_version }}:{{ stable_version
}},com.clickhouse:clickhouse-jdbc:{{ clickhouse_jdbc_version }}:all
```

操作

基本操作，例如创建数据库、创建表、写表、读表等。

```
 spark-sql> use clickhouse;
Time taken: 0.016 seconds

spark-sql> create database if not exists test_db;
Time taken: 0.022 seconds

spark-sql> show databases;
default
system
test_db
Time taken: 0.289 seconds, Fetched 3 row(s)

spark-sql> CREATE TABLE test_db.tbl_sql (
        >   create_time TIMESTAMP NOT NULL,
        >   m           INT       NOT NULL COMMENT 'part key',
        >   id          BIGINT    NOT NULL COMMENT 'sort key',
        >   value       STRING
        > ) USING ClickHouse
        > PARTITIONED BY (m)
        > TBLPROPERTIES (
        >   engine = 'MergeTree()',
        >   order_by = 'id',
        >   settings.index_granularity = 8192
        > );
Time taken: 0.242 seconds

spark-sql> insert into test_db.tbl_sql values
        > (timestamp'2021-01-01 10:10:10', 1, 1L, '1'),
        > (timestamp'2022-02-02 10:10:10', 2, 2L, '2')
        > as tabl(create_time, m, id, value);
Time taken: 0.276 seconds

spark-sql> select * from test_db.tbl_sql;
2021-01-01 10:10:10 1   1   1
2022-02-02 10:10:10 2   2   2
Time taken: 0.116 seconds, Fetched 2 row(s)

spark-sql> insert into test_db.tbl_sql select * from test_db.tbl_sql;
Time taken: 1.028 seconds

spark-sql> insert into test_db.tbl_sql select * from test_db.tbl_sql;
Time taken: 0.462 seconds

spark-sql> select count(*) from test_db.tbl_sql;
6
Time taken: 1.421 seconds, Fetched 1 row(s)

spark-sql> select * from test_db.tbl_sql;
2021-01-01 10:10:10 1   1   1
2021-01-01 10:10:10 1   1   1
2021-01-01 10:10:10 1   1   1
2022-02-02 10:10:10 2   2   2
2022-02-02 10:10:10 2   2   2
2022-02-02 10:10:10 2   2   2
Time taken: 0.123 seconds, Fetched 6 row(s)

spark-sql> delete from test_db.tbl_sql where id = 1;
Time taken: 0.129 seconds

spark-sql> select * from test_db.tbl_sql;
2022-02-02 10:10:10 2   2   2
2022-02-02 10:10:10 2   2   2
2022-02-02 10:10:10 2   2   2
Time taken: 0.101 seconds, Fetched 3 row(s)
```

使用 Spark Shell

启动 Spark Shell

```
$SPARK_HOME/bin/spark-shell \
--conf spark.sql.catalog.clickhouse=com.clickhouse.spark.ClickHouseCatalog \
--conf spark.sql.catalog.clickhouse.host=${CLICKHOUSE_HOST:-127.0.0.1} \
--conf spark.sql.catalog.clickhouse.protocol=http \
--conf spark.sql.catalog.clickhouse.http_port=${CLICKHOUSE_HTTP_PORT:-8123} \
--conf spark.sql.catalog.clickhouse.user=${CLICKHOUSE_USER:-default} \
--conf spark.sql.catalog.clickhouse.password=${CLICKHOUSE_PASSWORD:-} \
--conf spark.sql.catalog.clickhouse.database=default \
--jars /path/clickhouse-spark-runtime-{{ spark_binary_version }}_{{ scala_binary_version }}:{{ stable_version }}.jar,/path/clickhouse-jdbc-{{ clickhouse_jdbc_version }}-all.jar
```

以下论点：

```
  --jars /path/clickhouse-spark-runtime-{{ spark_binary_version }}_{{ scala_binary_version }}:{{ stable_version }}.jar,/path/clickhouse-jdbc-{{ clickhouse_jdbc_version }}-all.jar
```

可以替换为以下代码，以避免将JAR复制到Spark客户端节点。

```
  --repositories https://{maven-cental-mirror or private-nexus-repo} \
--packages com.clickhouse.spark:clickhouse-spark-runtime-{{ spark_binary_version }}_{{ scala_binary_version }}:{{ stable_version }},com.clickhouse:clickhouse-jdbc:{{ clickhouse_jdbc_version }}:all
```

Shell基本操作

基本操作，例如创建数据库、创建表、写表、读表等。

```
 scala> spark.sql("use clickhouse")
res0: org.apache.spark.sql.DataFrame = []

scala> spark.sql("create database test_db")
res1: org.apache.spark.sql.DataFrame = []

scala> spark.sql("show databases").show
+---------+
|namespace|
+---------+
|  default|
|   system|
|  test_db|
+---------+

scala> spark.sql("""
     | CREATE TABLE test_db.tbl (
     |   create_time TIMESTAMP NOT NULL,
     |   m           INT       NOT NULL COMMENT 'part key',
     |   id          BIGINT    NOT NULL COMMENT 'sort key',
     |   value       STRING
     | ) USING ClickHouse
     | PARTITIONED BY (m)
     | TBLPROPERTIES (
     |   engine = 'MergeTree()',
     |   order_by = 'id',
     |   settings.index_granularity = 8192
     | )
     | """)
res2: org.apache.spark.sql.DataFrame = []

scala> :paste
// Entering paste mode (ctrl-D to finish)

spark.createDataFrame(Seq(
    ("2021-01-01 10:10:10", 1L, "1"),
    ("2022-02-02 10:10:10", 2L, "2")
)).toDF("create_time", "id", "value")
    .withColumn("create_time", to_timestamp($"create_time"))
    .withColumn("m", month($"create_time"))
    .select($"create_time", $"m", $"id", $"value")
    .writeTo("test_db.tbl")
    .append

// Exiting paste mode, now interpreting.

scala> spark.table("test_db.tbl").show
+-------------------+---+---+-----+
|        create_time|  m| id|value|
+-------------------+---+---+-----+
|2021-01-01 10:10:10|  1|  1|    1|
|2022-02-02 10:10:10|  2|  2|    2|
+-------------------+---+---+-----+

scala> spark.sql("DELETE FROM test_db.tbl WHERE id=1")
res3: org.apache.spark.sql.DataFrame = []

scala> spark.table("test_db.tbl").show
+-------------------+---+---+-----+
|        create_time|  m| id|value|
+-------------------+---+---+-----+
|2022-02-02 10:10:10|  2|  2|    2|
+-------------------+---+---+-----+
```

执行ClickHouse原生SQL。

```
 scala> val options = Map(
    |     "host" -> "clickhouse",
    |     "protocol" -> "http",
    |     "http_port" -> "8123",
    |     "user" -> "default",
    |     "password" -> ""
    | )

scala> val sql = """
    | |CREATE TABLE test_db.person (
    | |  id    Int64,
    | |  name  String,
    | |  age Nullable(Int32)
    | |)
    | |ENGINE = MergeTree()
    | |ORDER BY id
    | """.stripMargin

scala> spark.executeCommand("com.clickhouse.spark.ClickHouseCommandRunner", sql, options)

scala> spark.sql("show tables in clickhouse_s1r1.test_db").show
+---------+---------+-----------+
|namespace|tableName|isTemporary|
+---------+---------+-----------+
|  test_db|   person|      false|
+---------+---------+-----------+

scala> spark.table("clickhouse_s1r1.test_db.person").printSchema
root
 |-- id: long (nullable = false)
 |-- name: string (nullable = false)
 |-- age: integer (nullable = true)
```

支持的数据类型

本节概述了Spark和ClickHouse之间的数据类型映射。下表提供了从ClickHouse读取数据到Spark以及将Spark数据插入ClickHouse时转换数据类型的快速参考。

将数据从ClickHouse读取到Spark

ClickHouse 数据类型	Spark 数据类型	是否支持	是原始的	备注
Nothing	NullType	✓	是	
Bool	BooleanType	✓	是	
UInt8, Int16	ShortType	✓	是	
Int8	ByteType	✓	是	
UInt16,Int32	IntegerType	✓	是	
UInt32,Int64, UInt64	LongType	✓	是	
Int128,UInt128, Int256, UInt256	DecimalType(38, 0)	✓	是	
Float32	FloatType	✓	是	
Float64	DoubleType	✓	是	
String, JSON, UUID, Enum8, Enum16, IPv4, IPv6	StringType	✓	是	
FixedString	BinaryType, StringType	✓	是	由配置控制`READ_FIXED_STRING_AS`
Decimal	DecimalType	✓	是	精度和规模高达`Decimal128`
Decimal32	DecimalType(9, scale)	✓	是	
Decimal64	DecimalType(18, scale)	✓	是	
Decimal128	DecimalType(38, scale)	✓	是	
Date, Date32	DateType	✓	是	
DateTime, DateTime32, DateTime64	TimestampType	✓	是	
Array	ArrayType	✓	是	数组元素类型也会被转换
Map	MapType	✓	是	钥匙仅限于`StringType`
IntervalYear	YearMonthIntervalType(Year)	✓	是	
IntervalMonth	YearMonthIntervalType(Month)	✓	是	
IntervalDay, IntervalHour, IntervalMinute, IntervalSecond	DayTimeIntervalType	✓	否	使用特定间隔类型
Object		✗		
Nested		✗		
Tuple		✗		
Point		✗		
Polygon		✗		
MultiPolygon		✗		
Ring		✗		
IntervalQuarter		✗		
IntervalWeek		✗		
Decimal256		✗		
AggregateFunction		✗		
SimpleAggregateFunction		✗		

将Spark中的数据插入ClickHouse

Spark 数据类型	ClickHouse 数据类型	是否支持	Is Primitive	备注
BooleanType	UInt8	✓	Yes	
ByteType	Int8	✓	Yes	
ShortType	Int16	✓	Yes	
IntegerType	Int32	✓	Yes	
LongType	Int64	✓	Yes	
FloatType	Float32	✓	Yes	
DoubleType	Float64	✓	Yes	
StringType	String	✓	Yes	
VarcharType	String	✓	Yes	
CharType	String	✓	Yes	
DecimalType	Decimal(p, s)	✓	Yes	Precision and scale up to `Decimal128`
DateType	Date	✓	Yes	
TimestampTypeArrayType (list, tuple, or array)	DateTime	✓	Yes	
ArrayType (list, tuple, or array)	Array	✓	No	Array element type is also converted
MapType	Map	✓	No	Keys are limited to `StringType`
Object		✗		
Nested		✗		

Spark JDBC

读取数据

```
public static void main(String[] args) {
    // Initialize Spark session
    SparkSession spark = SparkSession.builder().appName("example").master("local").getOrCreate();

    // JDBC connection details
    String jdbcUrl = "jdbc:ch://localhost:8123/default";
    Properties jdbcProperties = new Properties();
    jdbcProperties.put("user", "default");
    jdbcProperties.put("password", "123456");

    // Load the table from ClickHouse
    Dataset<Row> df = spark.read().jdbc(jdbcUrl, "example_table", jdbcProperties);

    // Show the DataFrame
    df.show();

    // Stop the Spark session
    spark.stop();
  }
```

写入数据

```
    public static void main(String[] args) {
        // Initialize Spark session
        SparkSession spark = SparkSession.builder().appName("example").master("local").getOrCreate();

        // JDBC connection details
        String jdbcUrl = "jdbc:ch://localhost:8123/default";
        Properties jdbcProperties = new Properties();
        jdbcProperties.put("user", "default");
        jdbcProperties.put("password", "******");
        // Create a sample DataFrame
        StructType schema = new StructType(new StructField[]{
                DataTypes.createStructField("id", DataTypes.IntegerType, false),
                DataTypes.createStructField("name", DataTypes.StringType, false)
        });

        List<Row> rows = new ArrayList<Row>();
        rows.add(RowFactory.create(1, "John"));
        rows.add(RowFactory.create(2, "Doe"));

        Dataset<Row> df = spark.createDataFrame(rows, schema);

        df.write()
                .mode(SaveMode.Append)
                .jdbc(jdbcUrl, "my_table", jdbcProperties);
        // Show the DataFrame
        df.show();

        // Stop the Spark session
        spark.stop();
    }
```

将自建ClickHouse数据迁移到云ClickHouse中

本工具是为给云上bmr Clickhouse集群做上云或下云数据迁移而准备，采用点对点的方式进行迁移，支持高并行、断点续传。适用于数据量较大、数据表较多且只需保证源和目标集群间数据最终一致的场景。

环境准备

1. 本工具使用python3开发，因此执行节点上需要部署python3环境，版本最低要求为3.6，推荐3.7（附件有python安装包以及已经集成好的python环境）。
2. 迁移过程需要保存迁移元数据信息，元数据库选择的是mysql（每个bmr集群会自带一个mysql实例，无需另外搭建），因此需要安装python的mysql connector依赖，附件的python_venv中已经集成。
3. 迁移需要连接源集群实例和目标集群实例，因此需要安装clickhouse-client依赖，附件的python_venv中已经集成.
4. (可选)为了不影响原来的python环境，推荐使用附件中的python_venv集成环境来执行迁移。
5. 启动虚拟环境后再启动迁移工具。

```
[root@core-f36e4ac-01 ~]# source python3_venv/bin/activate
(python3_venv) [root@core-f36e4ac-01 ~]# python --version
Python 3.6.7
(python3_venv) [root@core-f36e4ac-01 ~]#
```

启动方式

启动之前需要将迁移任务录入元数据库，此信息对应的mysql 数据表为migrator.jobs，如下所示：

```
mysql> select * from migrator.jobs;
+--------+-------------+----------+----------+-------------+-----------------+----------+----------+--------------+
| job_id | src_host    | src_port | src_user | src_password | dst_host        | dst_port | dst_user | dst_password |
+--------+-------------+----------+----------+-------------+-----------------+----------+----------+--------------+
|      1 | 10.160.39.8 |     9000 | default  | 7@3c3a9*    | core-f36e4ac-01 |     9000 | admin    | X*gongbAg9g  |
|      2 | 10.160.39.7 |     9000 | default  | 7@3c3a9*    | core-f36e4ac-02 |     9000 | admin    | X*gongbAg9g  |
|      3 | 10.160.39.16 |     9000 | default  | 7@3c3a9*    | core-f36e4ac-03 |     9000 | admin    | X*gongbAg9g  |
|      4 | 10.160.39.3 |     9000 | default  | 7@3c3a9*    | core-f36e4ac-04 |     9000 | admin    | X*gongbAg9g  |
|      5 | 10.160.39.13 |     9000 | default  | 7@3c3a9*    | core-f36e4ac-05 |     9000 | admin    | X*gongbAg9g  |
|      6 | 10.160.39.2 |     9000 | default  | 7@3c3a9*    | core-f36e4ac-06 |     9000 | admin    | X*gongbAg9g  |
|      7 | 10.160.39.10 |     9000 | default  | 7@3c3a9*    | core-f36e4ac-07 |     9000 | admin    | X*gongbAg9g  |
|      8 | 10.160.39.6 |     9000 | default  | 7@3c3a9*    | core-f36e4ac-08 |     9000 | admin    | X*gongbAg9g  |
|      9 | 10.160.39.12 |     9000 | default  | 7@3c3a9*    | core-f36e4ac-09 |     9000 | admin    | X*gongbAg9g  |
|     10 | 10.160.39.15 |     9000 | default  | 7@3c3a9*    | core-f36e4ac-10 |     9000 | admin    | X*gongbAg9g  |
```

表一 字段说明

字段	字段说明
job_id	job的标识
src_host	源集群实例主机名或ip
src_port	源集群实例端口
src_user	源集群实例用户名
src_password	源集群实例密码
dst_host	目标集群实例主机名
dst_port	目标集群实例端口
dst_user	目标集群用户名
dst_password	目标集群密码

表二 参数说明

参数名称	参数说明
hostname	直接使用bmr提供的hostname，例如：core-0cb9d33-01，切记不要使用ip或者alias后的hostnmae。
taskID	当前job的task的标识，是为了提高并行度，和taskSize配合使用可以在一个job上执行更高并行度的迁移，加快迁移速度。
taskSize	当前job总共的task数，一般和taskID配合使用。
runType	当前迁移任务的类型，目前支持的类型有如下几种： • migration（既迁移库表结构也迁移数据） • migration_schema （只迁移库表结构，并对表数据做快照） • migration_data （只迁移数据，不迁移库表结构，前提是表快照已经完成） • check （校验数据） • drop_schema （删除目标实例上的数据库表） • migration_mv （删除目标实例上的物化视图）

使用实例

1. 执行命令：

```
python ck_migrator.py  core-0cb9d33-01  0  5   migration_schema
```

上述命令参数意义为：在core-0cb9d33-01 这个节点上执行job，task 并行度为5(taskID 为0，1，2，3，4)，当前task id 为0，任务类型迁移库表结构（migration_schema）

2. 上述命令执行后会显示：Will migrator all schemas!，执行完成后会显示以下信息：Migrator all shcema done!如果中间失败了可以重新拉起task进程，参数不变，不用担心数据或表会重复。

注意事项

1. 在迁移过程中不要对源集群的库表执行ddl操作，例如增删表字段等
2. 如果有物化视图需要单独迁移
3. mysql元数据库访问的用户名和密码指定在工具代码内的如图位置，可以依据实际情况进行修改，若不清楚当前集群mysql的访问密码可以咨询bmr值班同学。

4.如果想过滤自定义的表或者库，可以在工具内的如下位置进行修改：

在filter里面加上不想迁移的表名即可（直接修改代码），如下图。修改后迁移工具将不会迁移表'za_search_info' 和表 'deeptrace_info'。

运维相关操作

ClickHouse集群缩容

前提条件

已完成 ClickHouse 集群的创建，具体信息可查看 “创建 ClickHouse 集群” 的相关内容。

注意事项

- 当集群处于初始化或释放阶段，或者集群因欠费而停止服务时，无法处理扩容缩容请求。
- 集群节点core数量大于2支持缩容。
- 缩容后,您的物化视图数据可能会受到影响,请谨慎缩容。
- 缩容期间，正在被释放的节点可读不可写，其他节点可读可写。

操作步骤

1. 在产品服务>MapReduce>集群列表页，单击被调整集群对应节点管理按钮，进入该集群的节点管理列表页。
2. 节点管理支持对节点进行扩容、缩容、规格变更和磁盘变更。
3. 目前 ClickHouse 支持对core节点进行扩缩容，缩容方式为按分片缩容。单击节点处缩容按钮在容配置处对分片数量进行填写。
4. 填写完成后系统会跳转进行缩容检测界面，展示检测是否成功和预计被释放实例。

> 节点数量 = 副本数量 *分片数量，请调整副本配置和分片数量确定Core节点数量

ClickHouse集群扩容

前提条件

已完成 ClickHouse 集群的创建，具体信息可查看 “创建 ClickHouse 集群” 的相关内容。

注意事项

- 当集群处于初始化或释放阶段，或者集群因欠费而停止服务时，无法处理扩容缩容请求。
- 确认当前集群中没有正在进行的大规模数据迁移或长时间运行的查询任务，避免在扩容过程中因这些操作影响集群的性能或造成数据不一致。

操作步骤

1. 在产品服务>MapReduce>集群列表页，单击被调整集群对应节点管理按钮，进入该集群的节点管理列表页。
2. 节点管理支持对节点进行扩容、规格变更和磁盘变更。
3. 目前 ClickHouse 支持对core节点进行扩缩容，单击扩容按钮在扩容配置处对实例数目进行增加，最大实例数为200，最低为2。
4. 扩容结束后点击确认变更，系统跳转完成页。注意：实例变更将在5-15分钟内生效，同时集群状态变为“调整中”。

日志配置说明

前提条件

已创建BMR集群，且选择了ClickHouse服务。

Clickhouse控制台日志配置

在 ClickHouse 服务配置页面的服务配置区域，您只需点击 server - config 页签，接着在搜索区域输入 “logger.”，就能在该页面查看或修改所有相关的日志配置项。

表一 参数说明

参数	描述
logger.level	日志的等级，默认等级为information。可以配置的等级从严格到宽松依次为 • none：关闭日志。 • fatal：致命信息。 • critical：危险信息。 • error：错误信息。 • warning：警告信息。 • notice：普通但需要注意的信息。 • information（默认值）：重要或者您感兴趣的信息。 • debug：调试信息。 • trace：程序执行路径跟踪信息。
logger.size	日志文件的大小。当文件达到该参数设置的值时，ClickHouse会将其存档并重命名，并创建一个新的日志文件。默认值为1000M。
logger.count	存档的ClickHouse日志文件个数。当存档的日志文件个数达到该参数设置的值时，ClickHouse会将最早的存档删除。默认值为10。

ClickHouse客户端日志配置

通过配置客户端日志，您便能够接收来自服务端的日志，默认情况下接收的是fatal级别的日志。

1. 可以用 SSH 方式登录集群，具体操作详情请参考 “登录集群” 相关内容。

2. 操作步骤示例。

- 对每次执行的日志进行查看。

执行以下命令，进入ClickHouse客户端。

```
clickhouse-client -h <core节点ID> -m
```

您可以执行以下命令，设置参数send_logs_level查看每次执行的日志。

```
set send_logs_level='debug';
```

返回信息：

```
 SET send_logs_level = 'debug'
Ok.
0 rows in set. Elapsed: 0.002 sec.
```

- 在启动ClickHouse客户端时，您可以执行以下命令，将日志保存到指定的文件中。

```
clickhouse-client -h core-1-1 -m --send_logs_level=trace --log-level=trace --server_logs_file='/tmp/query.log'
```

监控告警配置

集群报警

报警配置

BMR的报警配置在BCM侧进行配置，BMR的报警配置分为BMR事件报警配置和BMR指标报警配置：

配置类型	类型说明
BMR事件报警配置	针对BMR中监控对象(比如主机和组件进程)运行状态(比如down/up)的事件报警配置。 操作步骤： 1.在**产品服务->云监控BCM**页中，在侧边导航点击**事件监控**，参考BCM的事件监控说明，配置BMR的事件报警策略。 2.配置主机运行状态的事件报警策略，产品类型需要选择**MapReduce BMR**, 事件名称选择主机宕和主机宕恢复。
BMR指标报警配置	针对BMR中监控对象指标阈值的报警配置，比如CPU利用率，磁盘利用率超过阈值报警配置。 1.在**产品服务->云监控 BCM**页中，点击**实例组**，参考BCM的实例组说明，配置BMR的实例组以及实例组的报警策略。 2.创建完实例组后，参考BCM的添加实例组报警策略 创建实例组的指标报警策略。

报警管理

当您需要监控各云服务资源的使用和运行情况时，您可以对已接入BCM的云服务设置合理的报警策略，包括对于资源设置性能消耗类指标的阈值报警，也可以对实例或服务的状态即事件监控设置事件报警。 同时针对站点监控中的探测点、应用监控中的实例和自定义监控中的监控项也可以配置合理的报警策略。

添加报警策略

- 指标监控

1. 登录百度智能云，选择**云监控BCM**，在左侧导航栏中点击**报警管理—报警策略—指标报警**，进入报警策略列表页面。
2. 点击**添加策略**按钮，进入创建报警策略页面，填写表单信息完成指标监控策略创建，填写产品类型时需要选择**MapReduce BMR**。

- 事件监控

1. 登录百度智能云，选**择云监控BCM**，在左侧导航栏中点击报警管理—报警策略—云产品事件，进入报警策略列表页面。
2. 点击添加策略按钮，进入“创建报警策略”页面，填写相应的表单信息完成事件监控策略创建。填写产品类型时需要选择**MapReduce BMR**。

报警策略操作

1. 登录百度智能云，选择云监控BCM，在左侧导航栏中点击**报警管理—报警策略**，进入报警策略列表页面。
2. 点击操作列的复制、编辑、删除、启用、禁用按钮，您可以对单个报警策略进行**复制、修改、启用、禁用**或**删除**操作。 勾选策略名称前的复选框，您可以对报警规则进行批量删除，启用，禁用操作。

查看报警策略详情

1. 登录百度智能云，选择**云监控BCM**，在左侧导航栏中点击**报警管理—报警策略**，进入报警策略列表页面。
2. 点击**报警策略名称**链接，您可以查看当前报警策略的详情信息。

说明：为方便您对策略进行编辑操作，在报警策略详情界面也提供了复制、编辑、删除和启用/禁用按钮，您可在查看详情的同时直接在此页面进行相关操作。

> 说明：为方便您对策略进行编辑操作，在报警策略详情界面也提供了复制、编辑、删除和启用/禁用按钮，您可在查看详情的同时直接在此页面进行相关操作。

禁用/启用报警通知

1. 登录百度智能云，选择**云监控BCM**，在左侧导航栏中点击**报警管理—报警策略**，进入报警策略列表页面。
2. 在通知状态列进行操作，展示“ON”则报警通知开启，“OFF”则报警通知关闭。

报警回调

通过报警回调，可实现将BCM云监控的报警通知发送到您指定的URL。您可以自由、灵活的处理各类告警消息，BCM支持通过 HTTP/HTTPS协议 的 POST 请求推送到可访问公网 URL ，您可基于回调接口推送的报警信息做进一步的处理。如需通过企业微信、钉钉、如流等办公软件接收报警通知，请参见webhook使用说明。

操作步骤

报警回调功能的入口有三处：统一的创建报警策略入口、云服务下单个实例创建报警策略入口和创建报警通知模版入口。下面将具体描述报警回调的操作步骤：

表一 报警回调操作步骤

报警回调入口	具体步骤
统一的创建报警策略入口	1.在左侧导航栏中点击**报警管理—报警策略**，在云产品监控的策略列表下，点击**添加策略**。 2.在创建策略页面，点击**报警回调**按钮开启，输入公网可访问的 URL 地址。
云服务下单个实例创建报警策略入口	1.在左侧导航栏中点击云产品监控，点击要查看的云产品，进入该云产品的实例列表页面。如查看云服务器BCC监控数据，点击云服务器监控，进入“云服务器列表”页面。然后选择对应的实例点击进入报警策略页面。 2.在实例报警策略页面，点击添加策略。 3.在“创建策略”页面，开启报警回调，输入公网可访问的 URL 地址。
创建报警通知模版入口	1.在左侧导航栏中点击**报警管理—报警模版**，在报警动作列表页面，点击添加模版。 2.在添加通知模版页面，接口回调一栏，输入公网可访问的 URL 地址。

webhook使用说明

表二 操作步骤说明

使用方式	使用步骤
企业微信	1. 登录企业微信，打开需要接收告警通知的企业微信群。 2. 添加群机器人后，复制webhook地址，参考操作步骤填写到“报警回调”中即可。 3. 配置成功后，当报警通知被触发时，您可以在企业微信群收到报警通知。
钉钉	1. 登录钉钉，打开需要接收告警通知的钉钉群，添加群机器人。 2. 填写表单，“安全设置”模块勾选“自定义关键词”选项，建议填写“报警”作为关键词。
如流	1. 登录如流，打开需要接收告警通知的如流群。 2. 群内添加如流机器人，复制webhook地址，参考操作步骤填写到**报警回调**中即可。 3. 配置成功后，当报警通知被触发时，您可以在如流群收到报警通知。

POST方式参数说明

表四 指标报警POST方式参数说明

参数	说明
alertId	告警ID
userId	账号ID
alarmName	报警策略名称
scope	云产品名称
policyType	策略类型（指标报警和事件报警之一，Metric代表是指标报警，Event代表事件报警）
alertStartTimestamp	发生告警的时间戳
region	报警对象所在的地域
monitoringObject	发生报警的对象
alarmLevel	报警等级状态。根据实际情况返回严重、通知、重要、警告四种状态中的一种
formula	报警条件
currentValue	报警发生或恢复时监控项的当前值
taskTimestamp	报警回调发送时间
signature	签名

表五 事件报警POST方式参数说明

参数	说明
alarmName	报警策略名称
scope	云产品名称
alertStartTimestamp	发生告警的时间戳
alertContent	事件详情
taskTimestamp	报警回调发送时间
signature	签名

URL回调实例，下面是URL回调的使用实例，BCM发起的POST方式URL回调请求：

```
 POST http://127.0.0.1:8201/callback
请求Body("Content-Type": "application/json")：
{
    "alarmStatus":"报警-异常",
    "alertId":"19925050-3f77-4839-bae7-6a5f721aae0c",
    "userId":"your_user_id",
    "alarmName":"test_bcc_alarm",
    "scope":"BCE_BCC",
    "policyType":"Metric",
    "alertStartTimestamp":1698982559,
    "region":"北京",
    "monitoringObject":"i-6nfua8xc/bcc-test-bj/-(公)/192.168.16.12(内)",
    "alarmLevel":"重要",
    "formula":"CPU使用率1分钟平均值 > -1 %",
    "currentValue":"CPU使用率=0.50%",
    "taskTimestamp":1698982642,
    "signature":"88e647b853e480046632a5eb9fef70f5"
}
```

在callback.java文件中接收POST参数并对消息进行校验：

```
 // 从发送来的POST请求中解析出alertId、taskTimestamp 、signature这3个参数。使用alertId、token（创建报警策略时生成的token）和taskTimestamp 这3个参数字符串连接并用MD5算法加密后的值来校验消息。
如果校验成功，则说明此消息为百度云发出，否则为非法请求，不予处理。其中taskTimestamp可以用来做过期验证，如果时间戳与用户当前时间时间间隔大于某个周期（如10分钟），则用户可自行丢弃请求。

if (md5(alertId + token + taskTimestamp) == signature) {
    ..........
}
```

报警历史

当报警发生后，您可以在报警历史页面通过产品类型、报警等级、当前状态等条件筛选想要关注的报警信息。

查看报警历史

1. 登录百度智能云，选择**云监控BCM**，在左侧导航栏中点击**报警管理—报警历史**，进入报警历史列表页面。

2. 切换tab页，可以分别查看云产品监控、站点监控、自定义监控、应用监控的报警历史信息。

查看报警详情

1. 登录百度智能云，选择**云监控BCM**，在左侧导航栏中点击**报警管理—报警历史**，进入报警历史列表页面。

2. 在报警历史页面，点击报警内容打开您要查看的报警事件的详情页面。

3. 在报警事件详情页面，可以查看该报警事件的基本信息，数据监控详情及该报警事件的状态变更历史。

Clickhouse常用指标

指标说明

指标名称	推荐阈值
DelayedInserts	10
MaxPartCountForPartition	300
ReplicasSumQueueSize	10
ReplicasMaxMergesInQueue	5
ReplicasMaxAbsoluteDelay	100
ReplicasMaxQueueSize	10
MemoryResident	53687091200
ReplicasSumMergesInQueue	10
Query	16
BackgroundSchedulePoolTask	300
OpenFileForRead	2048
OpenFileForWrite	512
QueryPreempted	5
ZooKeeperWatch	5000
ZooKeeperRequest	100

常见问题

如何配置数据索引

ClickHouse是一款高性能的列式数据库，通过其独特的索引机制，实现了极高的查询效率和吞吐量。 ClickHouse中的索引类型丰富多样，每种类型适用于不同的场景。以下是ClickHouse中的主要索引类型及其选择策略：

一、主要索引类型

1. 主键索引（Primary Key Index）

- 定义：主键索引是最常用的索引类型之一，用于唯一标识每条记录。在ClickHouse中，主键索引可 以加速数据的查找和聚合操作。
- 特点：主键索引由稀疏索引和段内数据排序组成。数据写入时按照主键排序（如果指定了ORDER BY），每次插入新的数据块时，ClickHouse会在磁盘上生成新的稀疏索引。每个稀疏索引条目对应 一个数据块中的首行，索引条目记录该块的首个主键值以及其在数据文件中的位置。
- 使用场景：适用于需要唯一标识记录的场景，如用户ID、订单号等。

2. 排序键索引（Sorting Key Index）

- 定义：排序键索引是根据指定的列对数据进行排序存储的索引类型。
- 特点：在ClickHouse中，可以通过指定ORDER BY关键字来创建排序键索引。排序键索引与主键索 引密切相关，默认情况下主键与排序键相同。排序键索引可以加速按排序键进行查询和聚合操作。
- 使用场景：适用于需要按特定列进行排序查询的场景，如按时间戳排序的事件日志查询。

3. 辅助索引（Secondary Index，也称为跳数索引或数据跳过索引）

- 定义：辅助索引用于加速查询列数据，提高查询效率。在ClickHouse中，辅助索引允许在查询时跳 过那些不相关的数据块，从而减少不必要的磁盘I/O。
- 类型：

类型	类型说明
minmax索引	记录每个数据段的最小值和最大值。对于范围查询，如 WHERE column > x
AND column < y ，ClickHouse可以跳过不在范围内的块。	
bloom_filter索引	用于高基数的数据列，如字符串或ID字段。布隆过滤器可以快速过滤掉不
可能匹配的数据块。	
tokenbf_v1索引	一种适用于包含大量词汇的字段（如文本）的布隆过滤器索引，能够对包含
某个词的查询进行加速。	
ngrambf_v1索引	基于n-gram的布隆过滤器索引，适用于需要精确匹配短字符串的场景。
set索引	记录每个数据段中特定列的唯一值集合，适用于精确匹配的场景。
inverted索引	倒排索引，适用于需要快速查找某个值在哪些数据块中存在的场景。

- 使用场景：适用于列上具有高度稀疏性或数据分布不均匀的场景，以及需要加速范围查询、精确匹 配等操作的场景。

4. 稠密索引（Dense Index）

- 定义：稠密索引存储了每一条记录的索引信息，可以提高范围查询的速度。
- 特点：与稀疏索引不同，稠密索引会为每一行数据都创建一个索引条目。虽然稠密索引在理论上可 以提供更快的查询速度，但由于其占用空间较大，且会增加数据插入、更新和删除的开销，因此在 ClickHouse中并不常用。
- 使用场景：适用于数据分布稠密、需要频繁进行范围查询的场景，但考虑到其性能和空间开销，实 际应用中需谨慎使用。

二、索引选择策略

在选择ClickHouse的索引类型时，需要综合考虑多个因素，包括数据特点、查询需求、性能要求等。以下是一些建议：

1. 根据查询需求选择索引类型：

- 如果查询中经常涉及主键列的查找和聚合操作，应优先考虑使用主键索引。
- 如果查询中经常需要进行范围查询，如按时间戳筛选数据，可以考虑使用minmax索引或稠密索引 （但需注意稠密索引的空间开销）。
- 如果查询中涉及对高基数列（如字符串或ID字段）的精确匹配，可以考虑使用bloom_filter索引。

2. 根据数据特点优化索引设计：

- 避免在低基数（即值的数量很少）的列上创建索引，因为这样的索引往往效果不佳。
- 当多个列经常一起用于查询条件时，可以考虑创建复合索引（通过ORDER BY子句指定多个列）。

3. 控制索引的大小和数量：

- 索引会占用额外的存储空间，并且会增加数据插入、更新和删除的开销。因此，在创建索引时要注 意控制索引的大小和数量，避免过度索引。
- 可以通过设置 GRANULARITY 参数来控制索引的粒度，从而平衡索引的大小和查询性能，一般建 议大小为8192。

4. 定期维护索引：

- 随着数据的插入、更新和删除，索引可能会变得碎片化，导致查询性能下降。定期对索引进行维 护，如重建索引、合并索引等，可以保持索引的高效性。

5. 结合使用多种索引类型：

- 在某些复杂查询场景中，可能需要结合使用多种索引类型来达到最佳的查询性能。例如，在按时间 戳排序的事件日志查询中，可以同时使用主键索引（基于用户ID）和minmax索引（基于时间戳） 来加速查询。

综上所述，ClickHouse提供了多种索引类型以满足不同的查询需求和数据特点。在选择索引类型时，需要根据实际情况进行综合考虑和优化设计。

如何定时删除数据

定期删除是在建表语句的ttl里面定义，另外在配置里面也可以定义ttl的转存。

用这个语句定义 query_log的保存天数。示例：

```
 ALTER TABLE system.query_log
MODIFY TTL event_time + INTERVAL 7 DAY;
```

Kerberos

概述

BMR支持创建开启Kerberos认证的集群，在创建集群时打开安全模式开关。在这种高安全级别的集群中，所有开源组件均采用Kerberos安全模式启动，确保只有经过Kerberos认证的客户端能够访问集群提供的服务（例如HDFS）。

背景信息

集群开启Kerberos之后：

- 客户端：可以对可信任的客户端提供认证，使得可信任客户端能够正确提交作业，恶意用户无法伪装成其他用户侵入到集群当中，能够有效防止恶意冒充客户端提交作业的情况。
- 服务端：集群中的服务都是可以信任的，集群服务之间使用密钥进行通信，避免了冒充服务的情况。

开启Kerberos能够提升集群的安全性，但是也会增加用户使用集群的复杂度：

- 开启Kerberos前需要用户对Kerberos的原理、使用有一些了解，才能更好地使用Kerberos。
- 由于集群服务间的通信加入了Kerberos认证机制，认证过程会有一些轻微的时间消耗，相同作业相较于没有开启Kerberos的同规格集群执行速度会有一些下降。

Kerberos身份认证原理

Kerberos是一种基于对称密钥技术的身份认证协议，可以为其他服务提供身份认证功能，且支持SSO（即客户端身份认证后，可以访问多个服务，例如HBase和HDFS）。

Kerberos组成内容如下：

- KDC：Kerberos的服务端程序。
- Client：需要访问服务的用户（Principal），KDC和Service会对用户的身份进行认证。
- Service：集成了Kerberos的服务。例如，HDFS、YARN和HBase。

Kerberos协议认证过程如下图所示。

Kerberos协议认证过程主要有以下两个阶段：

- 第一阶段：KDC对Client进行身份认证

当客户端用户（Principal）访问一个集成了Kerberos的服务之前，需要先通过KDC的身份认证。

如果身份认证通过，则客户端会获取到一个TGT（Ticket Granting Ticket），后续就可以使用该TGT去访问集成了Kerberos的服务。

- 第二阶段：Service对Client进行身份认证

当用户获取TGT后，就可以继续访问Service服务。

使用TGT以及需要访问的服务名称（例如HDFS）去KDC获取SGT（Service Granting Ticket），然后使用SGT去访问Service。Service会利用相关信息对Client进行身份认证，认证通过后就可以正常访问Service服务。

集群互信配置

工作准备

注意事项：

- 进行互信配置的两个集群的 nameservice 需要不同，BMR 默认 nameservice 为 bmr-cluster

- 本文以 BMR 跨域访问 CDH 集群中的服务为例。配置完成后，BMR 在获取到本集群 KDC 授予的TGT（Ticket Granting Ticket）后，能够跨域访问 CDH 中的服务。
- 本文配置的跨域互信是单向的，即 CDH 无法跨域访问 BMR 上的服务，如果需要实现双向跨域互信，按照同样的方法交换配置即可。
- 如需配置外部集群(即CDH集群)与BMR集群间的互信，除本文说明操作外还需要修改 BMR 安全组规则使外部集群可访问 BMR 集群 Kerberos 服务。

在进行配置前，需获知集群 hostname 或 ip 以及 realm 信息，以 master 节点获取 hostname 和 realm 为例，具体操作步骤如下。

1. 登录 BMR 集群的 master 节点；
2. 执行`hostname`命令获取 hostname；或通过接口可直接获取集群hostname列表，参考 实例操作接口；
3. 在 master 节点的 /etc/krb5.conf 文件中获取 realm，执行，其中 default_realm 即为所需的集群 realm。

```
cat /etc/krb5.conf
```

为方便表述实际使用方式，在本文中，规定所使用的 BMR 集群与 CDH 集群信息如下：

集群	hostname	realm
BMR	master-8932187	BAIDU.COM
CDH	test-cdh5-1	CDHTEST.COM

添加跨域认证 Principal

注意事项

- 实现 BMR 集群和 CDH 集群之间的跨域互信，例如使用 BMR 集群的客户端访问 CDH 集群中的服务，两个集群 realm 需要同时拥有名为 krbtgt/CDHTEST.COM.COM@BAIDU.COM的 principal。
- 默认情况下互信是单向的，CDH集群的客户端如需访问BMR集群的服务，两个REALM则需要有krbtgt/BAIDU.COM@CDHTEST.COM的principal。两个两次操作需要保证密码，version number和加密方式一致。

具体操作步骤如下：

1. 使用 root 用户，SSH 登录 BMR 集群的 master 节点；
2. 执行以下命令,添加 princ，其中 krbtgt/CDHTEST.COM@BAIDU.COM 为跨域访问的共同 principal；

```
kadmin.local -q 'add_principal -e "aes128-cts:normal des3-hmac-sha1:normal arcfour-hmac:normal" krbtgt/CDHTEST.COM@BAIDU.COM'
```

3. 按照提示输入两次密码（建议为集群密码），在两个集群中添加Principal时使用的密码必须一致；
4. 登录 CDH 集群，重复上述步骤1~2。

配置principal和user的映射关系

1. SSH 登录 BMR 集群的 master 节点；
2. 执行以下命令，获取 HDFS 配置文件 core-site 中的hadoop.security.auth_to_local值：

```
cat /etc/hadoop/conf/core-site.xml
```

3. 将步骤2中获取的值，追加到 CDH 集群的 HDFS 配置文件 hadoop.security.auth_to_local 值末尾；
4. 配置完毕后，重启 CDH 集群的HDFS；
5. 配置完成后，在 CDH 集群节点上执行 hadoop org.apache.hadoop.security.HadoopKerberosName princ查看是否能成功解析用户。

配置krb5.conf

1. SSH 登录 BMR 集群的 master 节点；
2. 执行以下命令，修改 BMR 集群上krb5.conf的配置信息。注：一个集群中所有节点的 krb5.conf 配置一致：

```
vim /etc/krb5.conf
```

3. 修改krb5.conf中的[domain_realm],将 CDH 及BMR集群的节点 hostname 和realm做映射

修改前格式如下：

```
[domain_realm]
 baidu.com = BAIDU.COM
```

修改后示例如下：

```
[domain_realm]
 baidu.com = BAIDU.COM
 core-8932187-3 = BAIDU.COM
 core-8932187-2 = BAIDU.COM
 core-8932187-1 = BAIDU.COM
 master-8932187 = BAIDU.COM
 test-cdh5-1.baidu.com = CDHTEST.COM
 test-cdh5-2.baidu.com = CDHTEST.COM
 test-cdh5-3.baidu.com = CDHTEST.COM
```

4. 修改 krb5.conf 中的 [realms] ，将 CDH的 KDC Server(配置在 CDH 集群的 krb5.conf 中)配置到 BMR 的 realms 中

修改前格式如下：

```
[realms]
 BAIDU.COM = {
  #kadmin instance_name
  admin_server = master-8932187
  #kdc instance_name
  kdc = master-8932187
  database_module = openldap_ldapconf
 }
```

修改后格式如下：

```
[realms]
 BAIDU.COM = {
  #kadmin instance_name
  admin_server = master-8932187
  #kdc instance_name
  kdc = master-8932187
  database_module = openldap_ldapconf
 }
 HWLPOC.COM = {
  kdc = test-cdh5-1.baidu.com
  admin_server = test-cdh5-1.baidu.com
 }
```

5. krb5.conf中新增[capaths]，krb5.conf 中 默认没有 [capaths] 相关配置。需新增配置，示例如下：

```
[capaths]
 CDHTEST.COM = {
  BAIDU.COM = .
 }
```

6. 同步修改好的krb5.conf配置信息复制到BMR集群所有节点

a.SSH 登录 BMR 集群master节点

b.执行如下命令：

```
 vim instance
 #把BMR集群所有节点（除了当前）hostname写入文件中，保存
ansible -i instance all -m copy -a "src=/etc/krb5.conf dest=/etc/krb5.conf"
```

7. 重启Kerberos服务

a.在master节点上执行如下命令

```
 systemctl restart krb5kdc.service
systemctl restart kadmin.service
```

b.在master节点上执行如下命令

```
 [libdefaults]
  kdc_timeout = 3000
  max_retries = 3
  udp_preference_limit = 1
  renew_lifetime = 7d
  forwardable = true
  #cluster realm
  default_realm = BAIDU.COM
  ticket_lifetime = 24h
  dns_lookup_realm = false
  dns_lookup_kdc = false
  default_ccache_name = /tmp/krb5cc_%{uid}
  #default_tgs_enctypes = aes des3-cbc-sha1 rc4 des-cbc-md5
  #default_tkt_enctypes = aes des3-cbc-sha1 rc4 des-cbc-md5

[domain_realm]
  #cluster realm
  baidu.com = BAIDU.COM
  core-8932187-3 = BAIDU.COM
  core-8932187-2 = BAIDU.COM
  core-8932187-1 = BAIDU.COM
  master-8932187 = BAIDU.COM
  test-cdh5-1.baidu.com = CDHTEST.COM
  test-cdh5-2.baidu.com = CDHTEST.COM
  test-cdh5-3.baidu.com = CDHTEST.COM

[logging]
  default = FILE:/mnt/bmr/log/kerberos/krb5kdc.log
  admin_server = FILE:/mnt/bmr/log/kerberos/kadmind.log
  kdc = FILE:/mnt/bmr/log/kerberos/krb5kdc.log

[realms]
  BAIDU.COM = {
    #kadmin instance_name
    admin_server = master-8932187
    #kdc instance_name
    kdc = master-8932187
    database_module = openldap_ldapconf
  }
  CDHTEST.COM = {
    kdc = test-cdh5-1.baidu.com
    admin_server = test-cdh5-1.baidu.com
  }

[dbdefaults]
  ldap_kerberos_container_dn = cn=krbcontainer,dc=bmr.sh,dc=com

[dbmodules]
  openldap_ldapconf = {
    db_library = kldap
    #openldap ip
    ldap_servers = ldap://172.18.0.33
    ldap_kerberos_container_dn = cn=krbcontainer,dc=bmr.sh,dc=com
    ldap_kdc_dn = uid=krb5kdc,ou=People,dc=bmr.sh,dc=com
    ldap_kadmind_dn = uid=kadmind,ou=People,dc=bmr.sh,dc=com
    #kerberos-ldap key file
    ldap_service_password_file = /etc/krb5.ldap
    ldap_conns_per_server = 5
 }
[capaths]
  CDHTEST.COM = {
    BAIDU.COM = .
  }
```

修改BMR集群的/etc/hosts

1. SSH 登录 BMR 集群的 master 节点；
2. 复制 CDH 集群节点 /etc/hosts 中的 CDH 集群节点信息，追加到 BMR 当前节点的 /etc/hosts 文件中。

```
 vim instance
 #把 BMR 所有节点（除了当前）hostname写入文件中，保存
ansible -i cores all -m copy -a "src=/etc/hosts dest=/etc/hosts"
```

互信验证

BMR 集群访问 CDH 集群的 HDFS 文件，示例如下：

```
 #查看本集群hdfs
hdfs dfs -ls /

 #查看CDH集群hdfs
 #test-cdh5-1.baidu.com为CDH集群的namenode所在节点
hdfs dfs -ls hdfs://test-cdh5-1.baidu.com:8020/
```

Hive

基础使用

Hive连接方式

本文介绍 BMR 集群如何连接 Hive 提交 Hive SQL，具体方式包括 Hive 客户端、Beeline 客户端、Java。

前提条件

- 已创建 BMR 集群，且选择了 Hive 服务，创建集群详情请参见创建集群。
- 用户如需使用安全模式集群，在创建集群时，需将安全模式开关设置为开启。

通过Hive客户端连接

普通集群

1. SSH登录集群，参考SSH连接到集群。
2. 执行 `hive` 命令即可完成连接。

安全模式集群

1. SSH登录集群，参考SSH连接到集群。
2. 执行以下命令进行验证：

```
kinit -kt /etc/emr/hive-conf/keytab/hive.keytab hive/<主节点名称>@BAIDU.COM
```

3. 执行 `hive` 命令完成连接。

通过Beeline客户端连接

普通集群

1. SSH登录集群，参考SSH连接到集群。
2. 执行以下命令完成连接：

```
beeline -u jdbc:hive2://<主节点名称>:10000
```

安全模式集群

1. SSH登录集群，参考SSH连接到集群。
2. 执行以下命令进行认证：

```
kinit -kt /etc/security/keytabs/hive.service.keytab hive/<主节点名称>@BAIDU.COM
```

3. 执行以下命令连接 Beeline：

```
beeline -u "jdbc:hive2://<主节点名称>:10000/;principal=hive/<主节点名称>@BAIDU.COM
```

通过Java连接

注意事项

1. 在pom.xml文件中配置项目依赖（hadoop-common和hive-jdbc）。本示例新增的项目依赖如下所示。

```
<dependencies>
    <dependency>
        <groupId>org.apache.hive</groupId>
        <artifactId>hive-jdbc</artifactId>
        <version>3.1.3</version>
    </dependency>
    <dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-common</artifactId>
        <version>3.2.3</version>
    </dependency>
</dependencies>
```

2. 编写代码，连接HiveServer2并操作Hive表数据。示例代码如下所示。

```
import java.sql.*;

publicclassApp
{
    privatestaticStringdriverName="org.apache.hive.jdbc.HiveDriver";

    publicstaticvoidmain(String[] args)throws SQLException {

        try {
            Class.forName(driverName);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        Connectioncon= DriverManager.getConnection(
            "jdbc:hive2://master-2044d6e-2:10000", "root", "");//其中 master-2044d6e-2 为节点名称

        Statementstmt= con.createStatement();

        Stringsql="select * from sample_tbl limit 10";
        ResultSetres= stmt.executeQuery(sql);

        while (res.next()) {
            System.out.println(res.getString(1) + "\t" + res.getString(2));
        }

    }
}
```

3. 打包项目工程，生成JAR包，上传JAR包至 BMR 集群 Master 节点。

4. 测试验证JAR包正常运行：

```
java -jar bmr-hiveserver2-1.0.jar
```

Hive基础操作

本文介绍如何通过 Hive 在 BMR 集群上进行库、表操作

前提条件

- 已创建 BMR 集群，且选择了 Hive 服务，创建集群详情请参见创建集群。

库操作

注意：示例中的数据库名称以testdb为例介绍。

1. 创建库

```
create database if not exists testdb;
```

当返回信息包含OK时，表示创建库testdb成功。

2. 查看库

```
desc database testdb;
```

3. 使用数据库

```
use testdb;
```

4. 删除库

```
drop database if exists testdb;
```

当返回信息包含OK时，表示删除库成功。

表操作

注意：示例中的数据表名称以t为例介绍。

1. 创建表

```
create table if not exists t (id bigint, value string);
```

当返回信息包含OK时，表示创建表table成功。

2. 查看表信息

```
desc formatted t;
```

3. 查看所有表

```
show tables;
```

返回信息如下所示：

```
OK
t
```

4. 删除表

```
drop table if exists t;
```

当返回信息包含OK时，表示删除表成功。

SQL操作

1. 插入记录

```
insert into table t select 1, 'value-1';
```

当返回信息包含OK时，表示插入信息成功。

2. 查询表中的信息

```
select * from t limit 10;
```

3. 聚合操作

```
select value, count(id) from t group by value;
```

开发指南

自定义函数（UDF）

Hive 具备诸多内建函数，能够满足您的计算需求。此外，您还可创建自定义函数（UDF），以满足多样化的计算需求。UDF 的使用方式与常见的内建函数相近。本文将向您介绍自定义函数的开发及使用流程。

背景信息

表一 UDF分类说明

UDF分类	描述
UDF（User Defined Scalar Function）	自定义标量函数，通常称为UDF。其输入与输出是一对一的关系，即读入一行数据，写出一条输出值。
UDTF（User Defined Table-valued Function）	自定义表值函数，用来解决一次函数调用输出多行数据场景的，也是唯一一个可以返回多个字段的自定义函数。
UDAF（User Defined Aggregation Function）	自定义聚合函数，其输入与输出是多对一的关系，即将多条输入记录聚合成一条输出值，可以与SQL中的Group By语句联合使用。

开发UDF

1. 使用IDE，创建Maven工程。

工程基本信息如下，您可以自定义groupId和artifactId。

```
 <groupId>org.example</groupId>
<artifactId>hiveudf</artifactId>
<version>1.0-SNAPSHOT</version>
```

2. 添加pom依赖。

```
 <dependency>
    <groupId>org.apache.hive</groupId>
    <artifactId>hive-exec</artifactId>
    <version>2.3.7</version>
    <exclusions>
      <exclusion>
        <groupId>org.pentaho</groupId>
        <artifactId>*</artifactId>
      </exclusion>
    </exclusions>
</dependency>
```

3. 构建一个类，使其继承自Hive UDF类。

类名可以自定义，本文示例中类名为MyUDF。

```
 package org.example;

import org.apache.hadoop.hive.ql.exec.UDF;

/**
 * Hello world!
 *
 */
public class MyUDF extends UDF
{
    public String evaluate(final String s) {
        if (s == null) { return null; }
        return s + ":HelloWorld";
    }
}
```

4. 将自定义的代码打成JAR包。

在pom.xml所在目录，执行如下命令制作JAR包。

```
 mvn clean package -DskipTests
```

target目录下会出现hiveudf-1.0-SNAPSHOT.jar的JAR包，即代表完成了UDF开发工作。

使用UDF

1. 将生成的JAR包上传到集群root目录。

2. 上传JAR包至HDFS。

a.通过SSH方式登录集群，详情请参见登录集群。

b.执行以下命令，将JAR包上传到HDFS。

```
 hadoop fs -put hiveudf-1.0-SNAPSHOT.jar /user/hive/warehouse/
```

可以通过`hadoop fs -ls /user/hive/warehouse/`命令，查看是否上传成功。待返回信息如下所示表示上传成功。

```
Found 1 items
-rw-r--r--   1 xx xx 2668 2021-06-09 14:13 /user/hive/warehouse/hiveudf-1.0-SNAPSHOT.jar
```

3. 创建UDF函数。

a.运行以下命令，进入到Hive命令行。

```
hive
```

b.运行以下命令，利用生成的 JAR 包创建函数。

```
create function myfunc as "org.example.MyUDF" using jar "hdfs:///user/hive/warehouse/hiveudf-1.0-SNAPSHOT.jar";
```

代码中的`myfunc`是UDF函数的名称，`org.example.MyUDF`是开发UDF中创建的类，`hdfs:///user/hive/warehouse/hiveudf-1.0-SNAPSHOT.jar`为上传JAR包到HDFS的路径。

当出现以下信息时，表示创建成功。

```
Added [/private/var/folders/2s/wzzsgpn13rn8rl_0fc4xxkc00000gp/T/40608d4a-a0e1-4bf5-92e8-b875fa6a1e53_resources/hiveudf-1.0-SNAPSHOT.jar] to class path
Added resources: [hdfs:///user/hive/warehouse/myfunc/hiveudf-1.0-SNAPSHOT.jar]
```

4. 执行以下命令，使用UDF函数。

该函数与内置函数使用方式一样，直接使用函数名称即可访问。

```
select myfunc("abc");
```

返回如下信息。

```
OK
abc:HelloWorld
```

实践操作

Hive操作HBase外表

创建 Hbase 表

```
hbase shell
```

创建 hbase 表 hbase_hive_table：

```
hbase(main):004:0> create 'hbase_hive_table', 'cf'
Created table hbase_hive_table
Took 1.3137 seconds
```

Hive 操作

进入 hive 环境

```
hive
```

1. 设置引擎为 mr MapReduce 引擎 和本集群的 Hbase 环境已经调好，如果使用其他集群的 Hbase，可以用 add file hbase-site.xml 添加其他集群的配置文件（需要服务器打通）。TEZ 引擎需要额外的 Hbase 相关的配置，需使用 add jar 的方式把 hbase 的相关 jar 包放到执行环境里。

```
set hive.execution.engine=mr;
```

2. 创建外部表

```
CREATE EXTERNAL TABLE hbase_hive_table (key int, value string)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf:val")
TBLPROPERTIES ("hbase.table.name" = "hbase_hive_table", "hbase.mapred.output.outputtable" = "hbase_hive_table");
```

3. 插入数据

```
INSERT OVERWRITE TABLE HBASE_HIVE_TABLE values(98, 'abc');
```

4. 检索数据

```
select * from HBASE_HIVE_TABLE;
OK
98 abc
Time taken: 0.246 seconds, Fetched: 1 row(s)
```

Join 测试

1. 创建表

```
create table t1(t1_c1 int, t1_c2 string);
insert into t1 values(1,'t1_key1'),(2,'t1_key2'),(3,'t1_key3');

create table t2(t2_c1 int, t2_c2 string);
insert into t2 values(2,'t2_key2'),(3,'t2_key3'),(4,'t2_key4');
```

2. 插入数据

```
INSERT OVERWRITE TABLE HBASE_HIVE_TABLE
select t1_c1, concat(t1_c2, t2_c2)
from t1 join t2 on t1_c1 = t2_c1;
```

3. 检索结果

```
select * from HBASE_HIVE_TABLE;
OK
2 t1_key2t2_key2
3 t1_key3t2_key3
98 abc
Time taken: 0.141
```

Hive迁移

操作步骤

本文档描述如何把 Hive 数据库从一个 Hadoop 集群迁移到另一个 Hadoop 集群。

本文档假定新集群的 Hive 元数据库的内容可以清空。

1. 停止老集群 Hive 集群 的 hive-metastore 和 hive-server2

停止方法：使用命令`systemctl stop hive-metastore`或者通过 ambari 操作。

```
systemctl stop hive-metastore
systemctl stop hive-server2
```

停止 hive metastore 后，集群所有的访问 hive 的任务都会报错。

2. 新集群 Hive 停止所有的 hive-metastore 和 hive-server2

```
systemctl stop hive-metastore
systemctl stop hive-server2
```

3. 备份 MySQL 数据库

```
mysqldump -h ${MYSQL_HOST} -uhive '-p${MYSQL_PASSWD}' hive > hive-metastore.bak
```

4. 恢复 MySQL数据库到新的地址

登录新的数据库：

```
mysql -h ${NEW_MYSQL_HOST} -uhive '-p${MYSQL_PASSWD}' hive
```

如果目标数据库已经存在则先删除：

```
drop database hive;
```

创建目标数据库：

```
create database hive;
```

导入数据到新的数据库。

```
mysql -h ${NEW_MYSQL_HOST} -uhive '-p${MYSQL_PASSWD}' hive < hive-metastore.bak
```

${NEW_MYSQL_HOST} 是数据库新的地址。

5. 新 Hive 集群修改 MetaStore 服务器所在的 hive-site.xml

按需要修改以下配置，如果 MYSQL 数据库信息没有改变，则不需要修改。

```
  <property>
   <name>javax.jdo.option.ConnectionDriverName</name>
   <value>com.mysql.jdbc.Driver</value>
  </property>

  <property>
   <name>javax.jdo.option.ConnectionPassword</name>
   <value>${MYSQL_PASSWD}</value>
  </property>

  <property>
   <name>javax.jdo.option.ConnectionURL</name>
   <value>jdbc:mysql://${NEW_MYSQL_HOST}/hive?createDatabaseIfNotExist=true&characterEncoding=UTF-8</value>
  </property>
  <property>
   <name>javax.jdo.option.ConnectionUserName</name>
   <value>hive</value>
  </property>
```

6. 升级 MetaStore 到新集群的版本（选做

如果新集群的 Hive 版本比老集群的 Hive 版本新，执行此步骤。如果一致，则跳过此步骤。新集群的 Hive 版本不得小于老集群的 Hive 版本。 在新集群 metastore 所在服务器，使用 hive 用户下执行以下操作, ${NEW_MYSQL_HOST}是新 mysql 数据库所在的服务器 IP 地址。 如下是把 hive 从 1.2 升级到 3.1。

```
 cd /opt/bmr/hive/scripts/metastore/upgrade/mysql
mysql -uhive '-p${MYSQL_PASSWD}' -h ${NEW_MYSQL_HOST} hive
```

进入 MySQL 之后，执行以下 SQL 命令。

```
 source upgrade-1.2.0-to-2.0.0.mysql.sql;
source upgrade-2.0.0-to-2.1.0.mysql.sql;
source upgrade-2.1.0-to-2.2.0.mysql.sql;
source upgrade-2.2.0-to-2.3.0.mysql.sql;
source upgrade-2.3.0-to-3.0.0.mysql.sql;
source upgrade-3.0.0-to-3.1.0.mysql.sql;
```

7. 复制老集群的数据到新集群

复制老集群管理表数据到新集群

此步骤要求所有管理表都在老集群参数`hive.metastore.warehouse.dir`设置的目录下。

```
 <property>
  <name>hive.metastore.warehouse.dir</name>
  <value>/warehouse/tablespace/managed/hive</value>
</property>
```

执行命令：

```
 hadoop fs -rm -r hdfs://${new-cluster}/${hive.metastore.warehouse.dir}
hadoop distcp hdfs://${old-cluster}/${hive.metastore.warehouse.dir} hdfs://${new-cluster}/${hive.metastore.warehouse.dir
```

注意：目标路径 hdfs://${new-cluster}/${hive.metastore.warehouse.dir} 必须不存在，否则会拷贝到它的子目录。

一般在新集群中执行 distcp 命令。如果在新集群执行 distcp 命令，需要在新集群设置老集群的配置，并且所有新集群的服务器需要识别老集群的服务器，反之亦然。

如果不能用 hdfs 协议访问老集群，可以使用 hftp 或者 webhdfs 协议。

复制老集群外部表数据到新集群

此步骤要求所有管理表都在老集群参数hive.metastore.warehouse.external.dir设置的目录下，或者统一的目录。

```
 <property>
  <name>hive.metastore.warehouse.external.dir</name>
  <value>/warehouse/tablespace/external/hive</value>
</property>
```

执行命令：

```
 hadoop fs -rm -r hdfs://${new-cluster}/${hive.metastore.warehouse.external.dir}
hadoop distcp hdfs://${old-cluster}/${hive.metastore.warehouse.external.dir} hdfs://${new-cluster}/${hive.metastore.warehouse.external.dir}
```

一般在新集群中执行 distcp 命令。如果在新集群执行 distcp 命令，需要在新集群设置老集群的配置，并且所有新集群的服务器需要识别老集群的服务器，反之亦然。

如果不能用 hdfs 协议访问老集群，可以使用 hftp 或者 webhdfs 协议。

8. 修改元数据库数据的位置

mysql -h 172.18.0.148 -uhive '-pxxx' hive

```
 -- managed db location
update DBS SET DB_LOCATION_URI=replace(DB_LOCATION_URI, 'hdfs://${old-cluster}/${hive.metastore.warehouse.dir}','hdfs://${new-cluster}/${hive.metastore.warehouse.dir}');
-- external db location
update DBS SET DB_LOCATION_URI=replace(DB_LOCATION_URI, 'hdfs://${old-cluster}/${hive.metastore.warehouse.external.dir}', 'hdfs://${new-cluster}/${hive.metastore.warehouse.external.dir}' );
-- managed table and partition location
UPDATE SDS SET LOCATION = REPLACE(LOCATION, 'hdfs://${old-cluster}/${hive.metastore.warehouse.dir}','hdfs://${new-cluster}/${hive.metastore.warehouse.dir}');
-- external table partition location
update SDS SET LOCATION=replace(LOCATION, 'hdfs://${old-cluster}/${hive.metastore.warehouse.external.dir}', 'hdfs://${new-cluster}/${hive.metastore.warehouse.external.dir}');
```

9. 重启新集群的 metastore 和 hiveserver

进入新集群的 metastore 和 hiveserver 对应的服务器，以 root 身份执行以下命令：

```
 systemctl start hive-metastore;
systemctl start hive-server2;
```

10. 进行测试

查询老集群的表，看数据是否正确。

迁移示例

老集群 hdfs 地址为：hdfs://master-9331fa9:8020，hive 版本为 hive 1.2.0，mysql 元数据库地址为 master-9331fa9 新集群 hdfs 地址为：hdfs://master-b882990:8020，hive 版本为 hive 3.1.0，mysql 元数据库地址为 master-b882990（IP：172.18.0.148）

在老集群创建相应的库和表：

```
 create database managed;
use managed;
create table t(c1 string) stored as textfile;
load data local inpath '/etc/profile' overwrite into table t;
create table tp(c1 string) partitioned by (pt string)stored as textfile;
load data local inpath '/etc/profile' overwrite into table tp partition(pt='profile');

create database e_db location '/warehouse/tablespace/external/hive/e_db.db';
use e_db;
create table e(c1 string) stored as textfile;
load data local inpath '/etc/hosts' overwrite into table e;
create table ep(c1 string) partitioned by (pt string)stored as textfile;
load data local inpath '/etc/hosts' overwrite into table ep partition(pt='hosts');
```

1. 停止老集群 hive 集群 的 hive-metastore 和 hive-server2

在老集群的 master-9331fa9 的 root 账号执行以下命令：

```
systemctl stop hive-metastore
systemctl stop hive-server2
```

2. 新集群 hive 停止所有的 hive-metastore 和 hive-server2

在老集群的 master-b882990 的 root 账号执行以下命令：

```
systemctl stop hive-metastore
systemctl stop hive-server2
```

3. 备份 mysql 数据库

```
mysqldump -h master-9331fa9 -uhive '-pxxx' hive > hive-metastore.bak
```

4. 恢复 mysql数据库到新的地址

登录新的数据库：

```
mysql -h 172.18.0.148 -uhive '-pxxx' hive
```

目标数据库已经存在，先删除：

```
drop database hive;
```

创建目标数据库

```
create database hive;
```

恢复 MYSQL:

```
mysql -h 172.18.0.148 -uhive '-pxxx' hive < hive-metastore.bak
```

5. 新集群修改 metastore 服务器所在的 hive-site.xml

新集群连接 MYSQL 数据库的地址没有改变，跳过此步。

6. 升级 metastore 到 新集群的版本

如下是把 hive 从 1.2 升级到 3.1。

```
cd /opt/bmr/hive/scripts/metastore/upgrade/mysql
mysql -h 172.18.0.148 -uhive '-pxxx' hive
```

进入 MYSQL 之后，执行以下 SQL 命令：

```
source upgrade-1.2.0-to-2.0.0.mysql.sql;
source upgrade-2.0.0-to-2.1.0.mysql.sql;
source upgrade-2.1.0-to-2.2.0.mysql.sql;
source upgrade-2.2.0-to-2.3.0.mysql.sql;
source upgrade-2.3.0-to-3.0.0.mysql.sql;
source upgrade-3.0.0-to-3.1.0.mysql.sql;
quit;
```

7. 复制老集群的数据到新集群

复制老集群管理表数据到新集群

以 hive 账号执行以下命令：

```
hadoop fs -rm -r hdfs://master-b882990/warehouse/tablespace/managed/hive
hadoop distcp  hdfs://master-9331fa9:8020/apps/hive/warehouse hdfs://master-b882990:8020/warehouse/tablespace/managed/
```

复制老集群外部表数据到新集群

以 hive 账号执行以下命令:

```
 hadoop fs -rm -r /warehouse/tablespace/external/hive
hadoop distcp  hdfs://master-9331fa9:8020/warehouse/tablespace/external/hive hdfs://master-b882990:8020/warehouse/tablespace/external/hive
```

8. 修改元数据库数据的位置

```
 mysql -h 172.18.0.148 -uhive '-pxxx' hive
```

进入 MYSQL 环境执行以下命令：

```
 -- managed db location
update DBS SET DB_LOCATION_URI=replace(DB_LOCATION_URI, 'hdfs://master-9331fa9:8020/apps/hive/warehouse','hdfs://master-b882990:8020//warehouse/tablespace/managed/hive');
-- external db location
update DBS SET DB_LOCATION_URI=replace(DB_LOCATION_URI, 'hdfs://master-9331fa9:8020/warehouse/tablespace/external/hive','hdfs://master-b882990:8020/warehouse/tablespace/external/hive');
-- managed table and partition location
UPDATE SDS SET LOCATION = REPLACE(LOCATION, 'hdfs://master-9331fa9:8020/apps/hive/warehouse','hdfs://master-b882990:8020/warehouse/tablespace/managed/hive');
-- external table partition location
update SDS SET LOCATION=replace(LOCATION, 'hdfs://master-9331fa9:8020/warehouse/tablespace/external/hive','hdfs://master-b882990:8020/warehouse/tablespace/external/hive');
```

9. 重启新集群的 metastore 和 hiveserver

进入新集群的 metastore 和 hiveserver 对应的服务器，以 root 身份执行以下命令：

```
 systemctl start hive-metastore;
systemctl start hive-server2;
```

10. 验证

验证正常。

Spark

基础使用

Spark SQL 基础操作

Spark SQL允许用户直接运用SQL语句对数据进行操作，在此过程中，Spark会负责对SQL语句进行解析、优化以及执行。

以下示例展示了如何使用Spark SQL进行读取文件。示例如下：

- 示例1：Spark支持多种数据格式，本示例读取了JSON格式文件的数据，并输出为Parquet格式。

```
 val peopleDF = spark.read.json("examples/student.json")
peopleDF.write.parquet("student.parquet")
```

- 示例2：借助 SQL 从 parquetFile 表中提取出年龄处于 13 岁至 19 岁区间的年轻人的名字，接着将这些数据转换为 DataFrame。之后，利用 Map 操作把名字处理成可读的形式，最后输出处理后的结果。

```
 val namesDF = spark.sql("SELECT name FROM parquetFile WHERE age BETWEEN 13 AND 19")
namesDF.map(attributes => "Name: " + attributes(0)).show()
```

启动Spark Shell

Spark的Shell是一款功能强大的交互式数据分析工具，它为学习API提供了一种简便途径。在Spark中，你既能够运用Scala语言，也能够使用Python语言。

操作步骤：

1. 通过SSH方式连接集群，详情请参见登录集群。

2. 执行以下命令，启动Spark Shell。

```
 spark-shell
```

RDD基础操作

Spark是以弹性分布式数据集（RDD）这一概念为核心构建的，RDD是能够进行并行操作且具备容错能力的元素集合。在Spark里，创建RDD有两种途径，一是通过

集合来创建，二是借助外部数据集进行构建。像共享文件系统、HDFS、HBase或者任何提供Hadoop InputFormat的数据集，都可用于构建RDD。

1. 创建RDD示例：

- 通过集合来创建RDD

```
 val data = Array(1, 2, 3, 4, 5)
val distData = sc.parallelize(data)
```

- 通过外部数据集构建RDD

```
 val distFile = sc.textFile("data.txt")
```

当 RDD 成功构建完成后，可以针对它开展一系列操作，诸如 Map 和 Reduce 等操作。

比如，执行以下代码时，会先从外部存储系统读取一个文本文件，利用该文件构建出一个RDD。接着，借助RDD的Map算子对其进行运算，从而得到文本文件中每一行的长度。最后，再通过Reduce算子进行计算，得出文本文件中各行长度的总和。

```
 val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

一般来说，Spark RDD有两种常见操作，分别是Transform操作和Action操作。Transform操作不会马上执行，而是要等到执行Action操作时才会真正执行。

- Transform操作

操作	描述
map()	参数是函数，函数应用于RDD每一个元素，返回值是新的RDD。
flatMap()	参数是函数，函数应用于RDD每一个元素，拆分元素数据，变成迭代器，返回值是新的RDD。
filter()	参数是函数，函数会过滤掉不符合条件的元素，返回值是新的RDD。
distinct()	没有参数，将RDD里的元素进行去重操作。
union()	参数是RDD，生成包含两个RDD所有元素的新RDD。
intersection()	参数是RDD，求出两个RDD的共同元素。
subtract()	参数是RDD，去掉原RDD里和参数RDD里相同的元素。
cartesian()	参数是RDD，求两个RDD的笛卡尔积。

- Action操作

操作	描述
collect()	返回RDD所有元素。
count()	返回RDD中的元素个数。
countByValue()	返回各元素在RDD中出现的次数。
reduce()	并行整合所有RDD数据，例如求和操作。
fold(0)(func)	和reduce()功能一样，但是fold带有初始值。
aggregate(0)(seqOp,combop)	和reduce()功能一样，但是返回的RDD数据类型和原RDD不一样。
foreach(func)	对RDD每个元素都是使用特定函数。

PySpark基础操作

1. SSH登录集群，参考SSH连接到集群

2. 执行以下命令，进入PySpark交互式环境：

```
 pyspark
```

3. 初始化SparkSession

```
 from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
```

4. 创建DataFrame

```
 from datetime import datetime, date
import pandas as pd
from pyspark.sql import Row

df = spark.createDataFrame([
    (1, 2., 'string1', date(2000, 1, 1), datetime(2000, 1, 1, 12, 0)),
    (2, 3., 'string2', date(2000, 2, 1), datetime(2000, 1, 2, 12, 0)),
    (3, 4., 'string3', date(2000, 3, 1), datetime(2000, 1, 3, 12, 0))
],schema='a long, b double, c string, d date, e timestamp')
```

DataFrame创建完成后，可以通过各种类型的transform算子完成数据计算。

SparkSQL UDF 基础操作

Spark SQL具备众多内建函数，能够满足您的计算需求。同时，您还可以通过创建自定义函数（UDF）来满足各种不同的计算需求。UDF在使用方式上和普通的内建函数相近。本文将为您介绍在Spark SQL中使用Hive自定义函数的具体流程。

前提条件

已在Hive中创建了UDF，详情请参见开发UDF。

使用Hive UDF

1. 使用文件传输工具，上传生成的JAR包至集群任意目录（本文以test目录为例）。

2. 上传JAR包至HDFS或BOS（本文以HDFS为例）。

a.通过SSH方式登录集群，详情请参见登录集群。

b.执行以下命令，上传JAR包到HDFS:

```
hadoop fs -put /test/hiveudf-1.0-SNAPSHOT.jar /user/hive/warehouse/
```

您可以通过hadoop fs -ls /user/hive/warehouse/命令，查看是否上传成功。待返回信息如下所示表示上传成功。

```
 Found 1 items
-rw-r--r--   1 xx xx 2668 2021-06-09 14:13 /user/hive/warehouse/hiveudf-1.0-SNAPSHOT.jar
```

4. 执行以下命令，使用UDF函数。

该函数与内置函数使用方式一样，直接使用函数名称即可访问。

```
 select myfunc("abc");
```

返回如下信息。

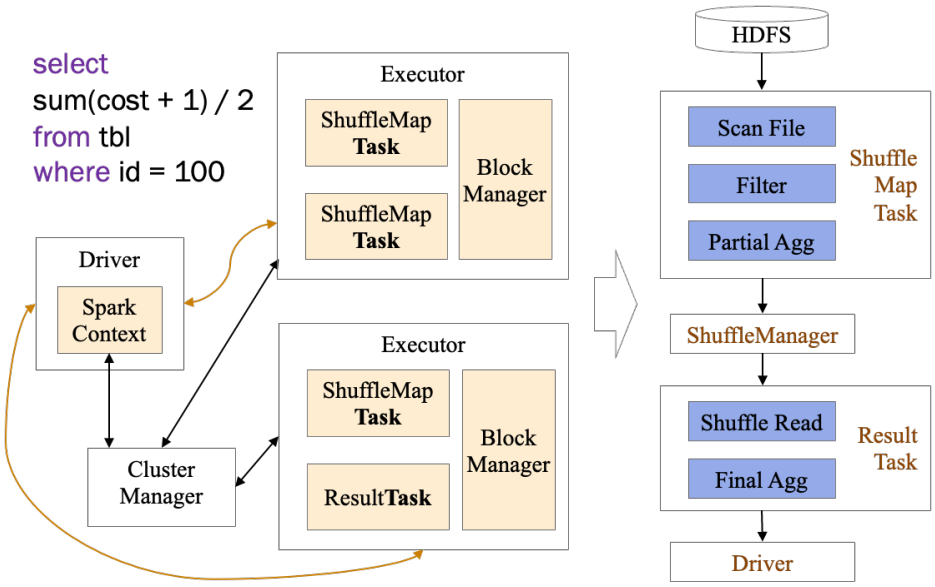
```
 OK
abc:HelloWorld
```

引擎增强

Spark Native Engine

截止目前，Spark是一个运行在JVM上的大数据计算引擎，其计算性能依然有很大的提升空间。我们通过引入ClickHouse计算引擎实现了Spark性能的巨大提升，同时还能保证和现有Spark的兼容性。

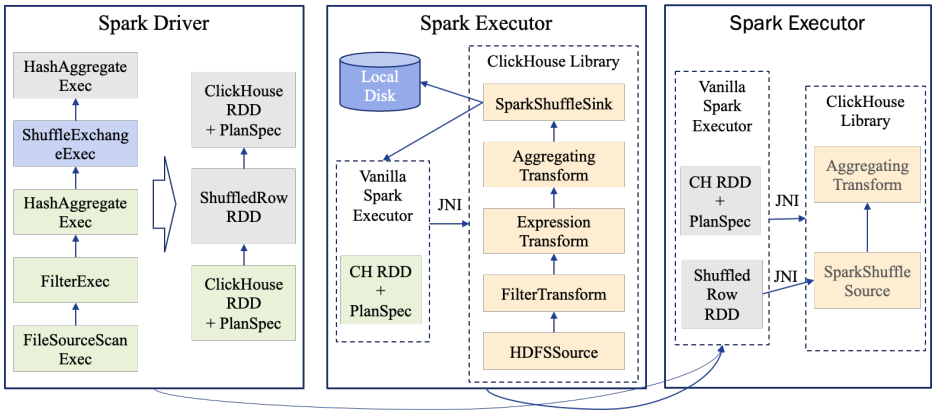
基本原理 我们先以一段简单的聚合SQL为例，看下Spark是如何执行的



我们的SQL代码经过解析先在Driver中生成了执行计划，执行计划优化后最终转化为一个个的RDD，每个RDD和其要处理的分片信息最终会发往对应的Executor，以Task为执行单元来执行对应的计算逻辑。这里可以将一个Task当做Executor中的一个线程，这个线程里完成了主要的计算逻辑。在Task中实际执行的Scan、Filter、Agg等计算逻辑就是由对应的RDD生成的。

实际上ClickHouse中也有对应于Scan、Filter、Agg这样的计算逻辑，所以我们可以将每个Task中的实际计算逻辑交给ClickHouse来执行，凭借ClickHouse强大的计算性能实现Spark性能的提升。我们这里称使用ClickHouse加速以后的Spark引擎为SNE（Spark Native Engine）。

下面是这段SQL使用SNE执行时的执行计划以及每个Task中Java和ClickHouse的调用关系。



可以看到我们使用JNI将执行计划信息发给了ClickHouse，在整体分布式框架上，依然保持了Spark现有的架构。

使用方法 使用SNE的方法非常简单，只需要增加如下配置即可：

```
spark.sql.execution.clickhouse true
```

由于我们目前还没有支持所有Spark特性，对于不支持的情况，我们会自动回退到原生Spark来执行用户的请求。另外我们支持了如下配置可以强制使用SNE执行而不回滚

```
spark.sql.clickhouse.fallback false
```

这个配置的默认值是`true`，就是会自动回滚。

进展 在数据类型方面，我们目前已经支持了常见的基础类型：BooleanType, ByteType, IntegerType, LongType, FloatType, DoubleType, DecimalType, TimestampType, DateType, StringType。

在算子级别的覆盖度见下表：

Operator	支持度
FileSourceScanExec	支持
FilterExec	支持
ProjectExec	支持
HashAggregateExec	支持
SortExec	支持
BroadcastHashJoinExec	支持
ShuffledHashJoinExec	支持
SortMergeJoinExec	支持
CartesianProductExec	支持
BroadcastNestedLoopJoinExec	支持
WindowExec	支持
GlobalLimitExec	支持
LocalLimitExec	支持
CollectLimitExec	支持
TakeOrderedAndProjectExec	支持
ExpandExec	支持
UnionExec	支持
ShuffleExchangeExec	支持
BroadcastExchangeExec	支持
SubqueryBroadcastExec	支持
DataWritingCommandExec	不支持
BatchScanExec	不支持
ObjectHashAggregateExec	不支持
SortAggregateExec	不支持
CoalesceExec	不支持
GenerateExec	不支持
RangeExec	不支持
SampleExec	不支持
CustomShuffleReaderExec	不支持
InMemoryTableScanExec	不支持
AggregateInPandasExec	不支持
ArrowEvalPythonExec	不支持
FlatMapGroupsInPandasExec	不支持
MapInPandasExec	不支持
WindowInPandasExec	不支持
Velox2Row	不支持
Velox2Arrow	不支持

这里我们虽然只支持了一半多一点的算子，但是在SQL场景，基本上都已经都能覆盖到了。在SQL场景，还有两类SQL需要特别说明一下：

- 我们虽然不支持DataWritingCommandExec算子，但是对于包括了该算子的SQL，比如`insert into table_x select ...`，我们会将除了DataWritingCommandExec以外的算子全部执行在ClickHouse上，最后调用原生Spark的DataWritingCommandExec来完成计算。所以除了DataWritingCommandExec以外的查询算子依然可以享受到ClickHouse的性能提速。
- 还有Scan相关的算子，比如BatchScanExec、InMemoryTableScanExec，如果SQL用到了这些算子，目前会将整个SQL回退到原生Spark执行。这些算子我们接下来会尽快支持。

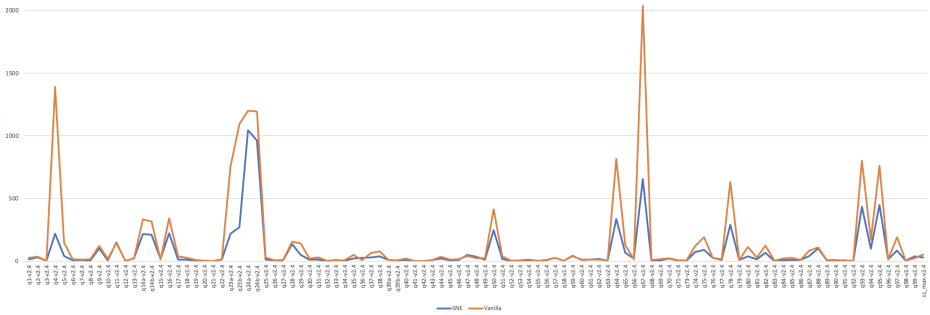
相对于原生Spark的内建函数，我们目前先支持了其中较常用的部分，剩余函数会逐步支持。目前支持的函数如下：

函数	说明
cast	
coalesce	
isNull	

isNotNull	
abs	
year	
month	
day/dayofmonth	
dayofweek	
weekofyear	
hour	
minute	
second	
quarter	
dayofyear	
upper/ucase	
!/not	
~	
cbrt	
cos	
floor	
ceil/ceiling	
sqrt	
length/char_length/character_length	暂不支持BinaryType
log2	
log10	
+	
-	
*	
/	
%/mod	暂不支持DecimalType
and	
or	
=/==	
<	
>	
<=	
>=	
datediff	
date_add	
date_sub	
add_months	
date_format	
like	
round	
shiftright	
shiftleft	
&	
|	
^	
pow/power	
instr	

substr/substring	
regexp_extract	
locate/position	
regexp_replace	
concat	
when	
in	
parse_url	不支持3个参数的版本，另外第二个参数只支持常量

在性能方面，在5T规模的TPC-DS数据集下，相对于社区版Spark的性能对比见下图



其中原生Spark总的执行时间是15257秒，SNE总的执行时间为7779秒。

Flink

基础使用

简介

Apache Flink 是一个开源的框架和分布式处理引擎，用于在无边界和有边界数据流上进行有状态的计算。它能够处理实时数据流和批处理数据，具有高吞吐量、低延迟和容错性强的特点。Flink 能在所有常见的集群环境中运行，并能以内存速度和任意规模进行计算。

常见应用场景：

1. 事件驱动型应用：

- 反欺诈：实时检测异常交易行为。
- 异常检测：监控系统状态，实时发现异常。

2. 数据分析应用：

- 流数据分析：实时分析数据流，提取有价值的信息。
- 实时报表分析：实时生成报表，展示关键指标的变化。

3. 数据管道与ETL：

- 实时ETL：Flink 提供丰富的Connector，支持多种数据源和数据Sink，能够实时处理数据管道。
- 实时数仓：支持分钟级或秒级的数据更新，便于实时查询和分析。

作业提交

1. 登录百度智能云控制台，选择“产品>MapReduce BMR”，单击“创建集群”，进入集群创建页，可选服务中勾选 Flink 服务。

> 注意：BMR2.1.1及以上版本支持 Flink。不同BMR版本对应支持的Flink组件版本也不同，具体支持版本以选择BMR版本后可选服务中组件版本为准。

2. SSH登录集群 ，参考SSH连接到集群。

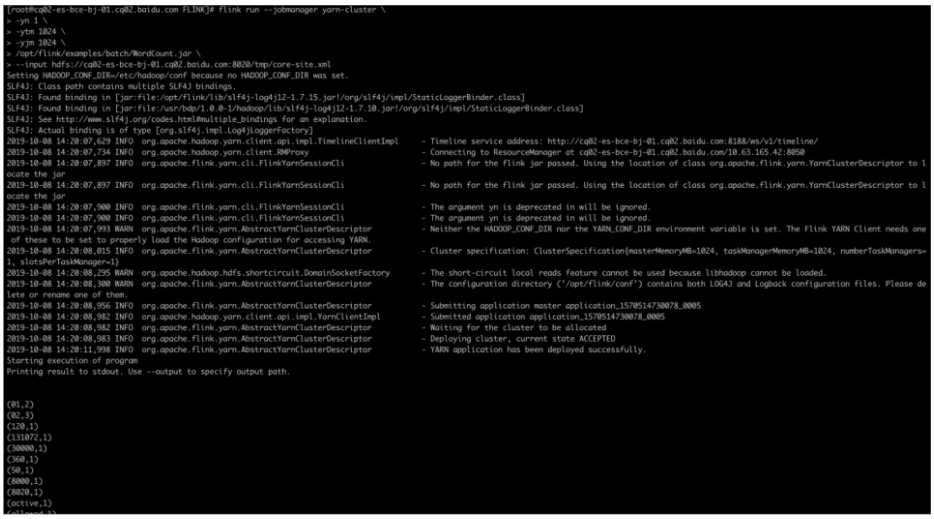
3. 执行以下命令，上传文件至HDFS，本示例以 Flink 作业示例 WordCount 为例:

```
hdfs dfs -put /etc/hadoop/conf/core-site.xml /tmp
```

4. 执行以下命令，提交作业：

```
 flink run --jobmanager yarn-cluster \
-yn 1 \
-ytm 1024 \
-yjm 1024 \
/opt/bmr/flink/examples/batch/WordCount.jar \
--input hdfs://$ACTIVE_NAMENODE_HOSTNAME:$PORT/tmp/core-site.xml \
--output hdfs://$ACTIVE_NAMENODE_HOSTNAME:$PORT/tmp/out
```

5. 结果查看



查看作业状态

1. 登录百度云控制台，进入 BMR 集群列表，单击集群名称/ID>>集群详情>>相关应用和工具>>Hadoop Yarn Web UI，进入 Web UI。
2. 单击 Flink 任务的 Application ID。
3. 进入详情页面后，单击 Tracking URL 的链接。
4. 进入 Apache Flink Dashboard 页面，即可查看作业的状态。

使用示例

通过BMR的Flink消费百度消息服务BMS。本文以使用Scala为例，Flink版本1.8.2,线上Kafka版本2.1。具体步骤如下：

第一步 创建Topic并下载百度消息服务的证书

（本步骤详情请参考文档 Spark流式应用场景）

下载证书：

第二步 编写业务代码

```
package com.baidu.inf.flink

import java.util.Properties

import org.apache.flink.api.common.serialization.SimpleStringSchema
import org.apache.flink.streaming.api.scala.{StreamExecutionEnvironment, _}
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer
import org.apache.kafka.common.serialization.StringDeserializer
import org.slf4j.LoggerFactory

object CloudFlinkConsumeKafkaDemo {
  private val logger = LoggerFactory.getLogger(this.getClass)

  def main(args: Array[String]): Unit = {
    logger.info("************ Flink Consume Kafka Demo start **************")
    if (args.length < 7) {
      logger.error(" Parameters Are Missing , " +
        "Needs : <topic> " +
        "<groupId> " +
        "<brokerHosts> " +
        "<truststore_location> " +
        "<truststore_pass> " +
        "<keystore_location> " +
        "<keystore_pass>")
      System.exit(-1)
    }
    val Array(topic, groupId, brokerHosts,
    truststore_location, truststore_pass,
    keystore_location, keystore_pass, _*) = args

    val env = StreamExecutionEnvironment.getExecutionEnvironment
    env.setParallelism(2)
    env.getConfig.disableSysoutLogging

    val kafkaProperties = new Properties()
    kafkaProperties.setProperty("bootstrap.servers", brokerHosts)
    kafkaProperties.setProperty("key.deserializer", classOf[StringDeserializer].getName)
    kafkaProperties.setProperty("value.deserializer", classOf[StringDeserializer].getName)
    kafkaProperties.setProperty("group.id", groupId)
    kafkaProperties.setProperty("auto.offset.reset", "latest")
    kafkaProperties.setProperty("serializer.class", "kafka.serializer.StringEncoder")
    kafkaProperties.setProperty("security.protocol", "SSL")
    kafkaProperties.setProperty("ssl.truststore.location", truststore_location)
    kafkaProperties.setProperty("ssl.truststore.password", truststore_pass)
    kafkaProperties.setProperty("ssl.keystore.location", keystore_location)
    kafkaProperties.setProperty("ssl.keystore.password", keystore_pass)
    kafkaProperties.setProperty("enable.auto.commit", "true")

    val ds = env.addSource(
      new FlinkKafkaConsumer[String](topic,
        new SimpleStringSchema(),
        kafkaProperties))

    ds.print()
    env.execute()
  }
}
```

第三步 编译代码，打成可执行Jar文件，上传到服务器上

（注：要保证第一步下载的证书文件在集群每个节点上相同的路径下都存在）

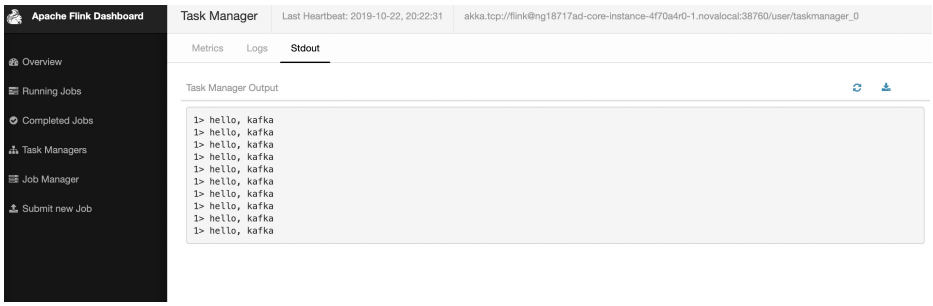
运行作业示例：

```
flink run --jobmanager yarn-cluster -yn 1 -ytm 1024 -yjm 1024 /root/flink-streaming-1.0-SNAPSHOT-jar-with-dependencies.jar
"676c4bb9b72c49c7bd3b089c181af9ec__demo02" "group1" "kafka.fsh.baidubce.com:9091" "/tmp/client.truststore.jks" "kafka"
"/tmp/client.keystore.jks" "0yw0ckrt"
```

第四步 消息队列中生产一些消息，在Flink作业监控页面上查看对应输出

通过Tunnel登录到集群的Yarn页面上（通过SSH-Tunnel访问集群）

在yarn console找到对应作业的application的单击application名称，进入作业详情页面：
（在Flink的原生页面上，点击TaskManagers > Stdout，查看作业运行情况）



Trino

概述

Trino简介

Trino 是一个开源的分布式SQL查询引擎，专为执行大规模数据集上的查询而设计，尤其在数据存储与计算分离的场景下表现出色。旨在提供快速的SQL查询能力，支持数据湖、数据仓库等多种数据存储。

Trino的基本特性

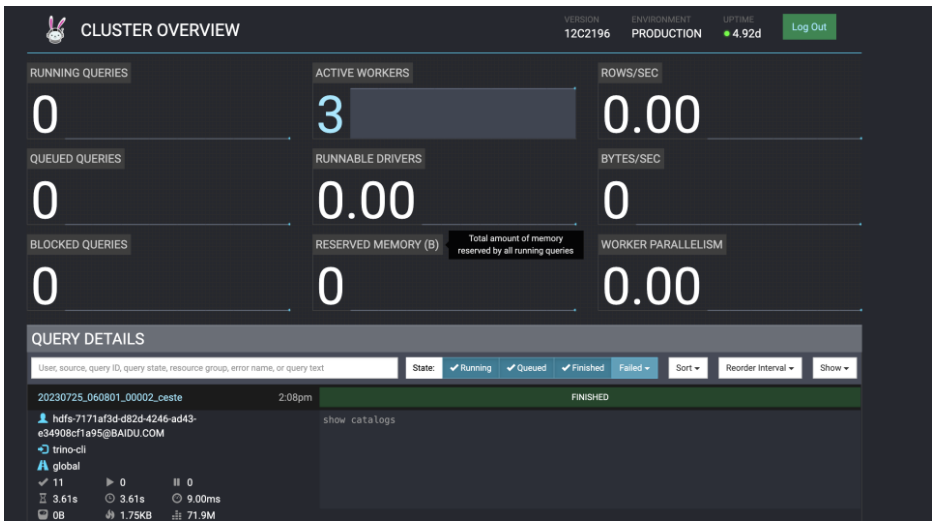
1. 高性能：分布式架构和内存内计算确保查询的快速执行。
2. 扩展性：能够扩展到数百甚至数千台节点，处理PB级的数据量。
3. 灵活性：支持多种数据源，包括关系型数据库、NoSQL数据库和对象存储。
4. 查询优化：Trino提供了多种查询优化策略，如广播连接和分布式哈希连接，以提升性能。

Trino的应用场景

1. 数据湖查询
2. 多源数据整
3. 弹性扩展
4. 实时数据处理
5. 大规模数据集的交互式分析
6. 企业大数据平台的构建

Trino UI

通过 百度云 BMR 控制台 集群列表>集群详情>相关应用和工具>Trino Url，可看到服务监控与查询详情等信息。



基础使用

前提条件

- 已创建 Hadoop 类型集群，并选择了 Trino 服务，创建集群详情请参见创建集群。或者 已创建 Trino 类型集群。
- 如需使用 EDAP 统一元数据管理，请开通 EDAP 产品使用权限。

元数据管理

Trino支持有两种元数据管理方式，一种是使用 DEFAULT 元数据管理，一种是使用 EDAP 元数据管理。

- DEFAULT 元数据管理：默认的数据源是本集群 Hive Meta Store，若要查询其它数据源，需要在集群上手动添加 connector 配置文件到 ${TRINO_HOME}/etc/catalog目录下并同步到所有节点，然后重启 Trino 服务，参考官网文档。
- EDAP 元数据管理：当使用EDAP 元数据管理时，需要在edap中添加数据源，trino服务对数据源实行动态加载，不需要重启服务。

连接Trino进行查询

元数据管理为DEFAULT

1. SSH登录集群 Core / Coordinator节点 ，参考SSH连接到集群
2. 执行以下命令，切换用户：

```
su hdfs
```

3. 执行以下命令进入交互式命令行:

```
/opt/bmr/trino/bin/trino \
--server https://{coordinator_host}:8089
```

4. 执行以下命令，连接到数据库:

```
use hive.default;
```

5. 创建表示例:

```
create table test(id bigint, name varchar);
```

6. 插入数据示例:

```
insert into test values(1,'aa'),(2,'bb');
```

7. 查询数据示例:

```
select * from test;
```

元数据管理为EDAP

1. SSH登录集群 Core / Coordinator节点 ，参考SSH连接到集群
2. 执行以下命令，切换用户：

```
su hdfs
```

3. 执行以下命令进入交互式命令行:

```
/opt/bmr/trino/bin/trino \
--server http://{coordinator_host}:8089 \
--catalog edap_physical_tbl \
--extra-credential access_key={accessKey} \     // accessKey的是/etc/hadoop/conf/core-site.xml fs.bos.access.key
--extra-credential secret_key={accessSecret} \  // accessSecret的是/etc/hadoop/conf/core-site.xml fs.bos.secret.access.key
--extra-credential session_token_key="{token}"   // token的是/etc/hadoop/conf/core-site.xml fs.bos.session.token.key
```

4. 执行以下命令，连接到数据库：

```
use edap_physical_tbl.{database};
```

5. 查询 EDAP 管理的表示例：

```
select * from test;
```

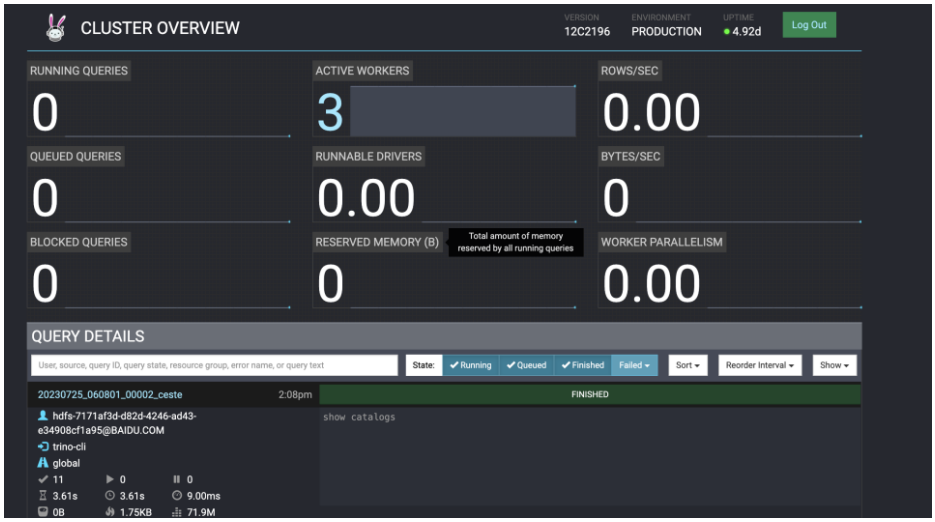
连接安全模式集群

BMR 支持创建安全模式的 Hadoop / Trino 集群，连接安全模式集群时，需要使用https协议访问7778端口，还需要添加安全认证参数。示例如下：

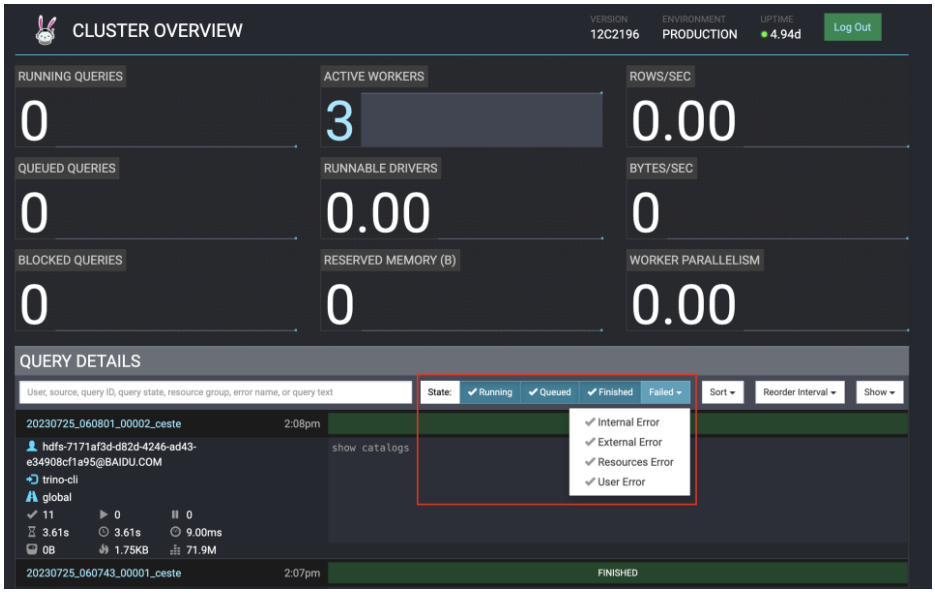
```
 /opt/bmr/trino/bin/trino \
// ${coordinator_host}请使用CanonicalHostName，同/opt/bmr/trino/etc/config.properties http-server.authentication.krb5.principal-hostname
--server https://{coordinator_host}:7778 \
--krb5-config-path /etc/krb5.conf  \
// 可以执行`klist -kt /etc/security/keytabs/hdfs.headless.keytab`获取
--krb5-principal hdfs@BAIDU.COM  \
--krb5-keytab-path /etc/security/keytabs/hdfs.headless.keytab  \
--krb5-remote-service-name trino  \
--keystore-path /etc/security/keytabs/keystore.jks  \
--keystore-password {http-server.https.keystore.key} \  // http-server.https.keystore.key的值在/opt/bmr/trino/etc/config.properties中
--user hdfs@BAIDU.COM \
```

查看日志

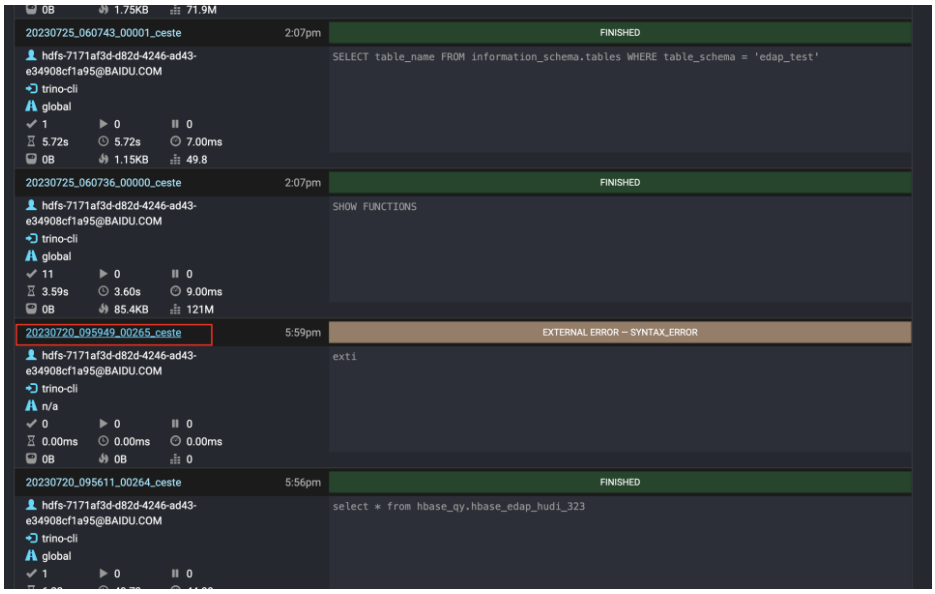
1. 通过 百度云 BMR 控制台 集群列表>>集群详情>>相关应用和工具>>Trino Url，进入Trino UI。

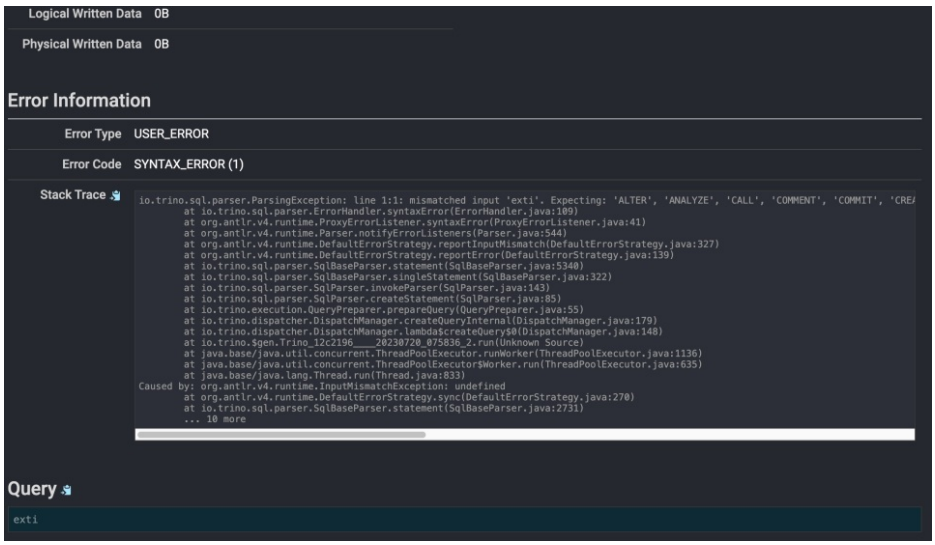


2. 通过页面的state选项可以对查询任务进行筛选。



3. 点击查询编号，可以进一步查看查询详情和报错日志.





Ranger

ranger概述

Apache Ranger 提供集中式的权限管理框架，可以对Hadoop生态中的HDFS/HIVE/YARN 等组件提供细粒度的权限访问控制，并且提供了Web UI页面方便管理员进行操作。

Ranger简介

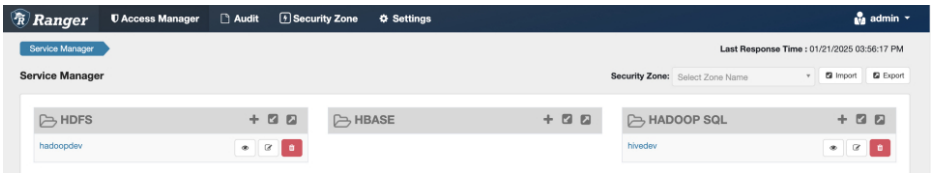
Apache Ranger 是一个为大数据平台提供集中化安全管理的开源框架，专门用于确保 Hadoop 生态系统中的数据安全。以下是 Ranger 的主要组件及其作用。

1. Ranger Admin（管理服务）：是 Ranger 的核心组件，提供了一个图形化用户界面（Web UI），用于集中管理安全策略和用户权限。管理员可以通过该界面创建、修改和删除访问控制策略，并定义基于角色的权限（RBAC）。
2. Ranger Plugins（插件架构）：Ranger 通过插件架构与多个 Hadoop 组件集成，如 HDFS、Hive、HBase、Kafka、Storm、YARN 等。插件被部署到这些组件中，负责拦截数据请求，并根据 Ranger Admin 定义的策略执行权限检查。
3. Ranger UserSync（用户同步服务）：将外部身份验证系统（如 LDAP、Active Directory）中的用户和组同步到 Ranger。通过 UserSync，管理员可以将企业用户的身份管理与 Ranger 的权限控制集成，自动同步企业中已有的用户和组，无需手动管理用户信息。
4. Ranger KMS（密钥管理服务）：用于管理和保护数据加密的密钥。通过 Ranger KMS，管理员可以控制数据加密和解密的权限，并对加密操作进行集中管理和审计。

访问Ranger UI

1. 登录控制台，选择产品服务-MapReduce BMR，进入集群列表页；
2. 点击集群名称/ID，进入集群详情页；
3. 点击相关应用和工具-Ranger Url，进入Ranger Web UI；

初始用户名/密码：admin/admin



权限策略配置

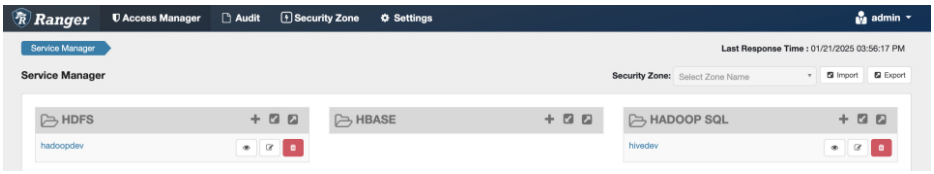
用户在创建集群时，选择了 Ranger 服务，则 BMR 自动为用户进行权限相关的Service 配置，用户可通过 Ranger UI 调整权限策略

前提条件

已创建 BMR 集群，且选择了 Ranger 服务，创建集群详情请参见创建集群.

操作步骤

1. 登录 Ranger UI，参考 Ranger概述。
2. 在 Ranger UI 页面，单击配置好的 Service，例如图中 hadoopdev。



3. 单击右上角的Add New Policy，根据您的实际需求配置相关参数。

参数	描述
Policy Name	策略名称，用户自定义。
Resource Path	资源路径。例如，/user/dev。
recursive	子目录或文件是否集成权限。
Select Group	指定添加此策略的用户组。
Select User	指定添加此策略的用户。
Permissions	选择授予的权限。例如，Write和Execute。

4. 单击Add，完成权限策略添加。

注意事项

- HDFS、YARN、HIVE（HADOOP SQL）、TRINO、BOS 均可按上述操作步骤进行配置；
- Flink、Spark 复用 HIVE 权限策略，无需额外配置

Paimon

Flink示例

前提条件

已完成创建 BMR 集群，并且配置了 Paimon、Flink 组件，详情请参见创建集群。

注意事项

- BMR Flink 不使用 Hive Metastore 元数据，可以使用文件系统存储元数据，可通过 Hive 和 Spark 操作.

操作示例

1. SSH登录集群，参考SSH连接到集群；
2. 参考以下命令，启动 /opt/bmr/flink/bin/yarn-session.sh：

```
 su - hdfs
/opt/bmr/flink/bin/yarn-session.sh
```

3. 执行 flink-sql a. 在新的命令行窗口，重复执行上述命令 注意：如果是存算分离类型的 BMR 集群，warehouse 地址需改为 BOS 地址，参考以下命令。

```
 CREATE CATALOG my_catalog WITH (
   'type'='paimon',
   'warehouse'='hdfs:///warehouse/paimon/flink'
);

USE CATALOG my_catalog;

-- create a word count table
CREATE TABLE word_count (
    word STRING PRIMARY KEY NOT ENFORCED,
    cnt BIGINT
);
insert into word_count values('abc', 1);
-- 等一会，或者点开 yarn 界面，找到 application, 进入 application Master，点击 Running jobs 和 Completed Jobs。看到 insert 语句执行完就可以。
select * from word_count;
```

4. 通过 Flink 创建 paimon 表，参考以下命令:

```
 CREATE TABLE `my_catalog`.`default`.`word_count` (
  `word` VARCHAR(2147483647) NOT NULL,
  `cnt` BIGINT,
  CONSTRAINT `PK_word` PRIMARY KEY (`word`) NOT ENFORCED
) WITH (
  'path' = 'hdfs:/warehouse/paimon/flink/default.db/word_count'
)
```

5. 在 Hive 中查询示例如下：注意需要创建外表。

```
 CREATE EXTERNAL TABLE external_test_table
STORED BY 'org.apache.paimon.hive.PaimonStorageHandler'
LOCATION 'hdfs:/warehouse/paimon/flink/default.db/word_count';
select * from external_test_table;
```

Spark示例

前提条件

已完成创建 BMR 集群，并且配置了 Paimon、Spark 组件，详情请参见创建集群。

注意事项

- Paimon 的 JAR 文件已存放到 ${SPARK_HOME}/jars 目录；
- 默认使用 Hive Catalog；
- Hive 用户可以直接使用 Spark 创建的表；
- 启动 Spark 不需要添加 Paimon 相关参数。

操作示例

1. SSH登录集群，参考SSH连接到集群；
2. 执行以下命令查看结果：

```
 -- 用hive以外的用户时需要在ranger配置权限
spark-sql --master local[2]
```

```
 USE paimon;
USE default;
drop table if exists spark_paimon;
-- 如果没有 bucket，对 spark 没有影响，hive  可以读，但是不能写入。
create table spark_paimon (
    id int,
    name string
) tblproperties (
    'primary-key' = 'id',
    'bucket' = '4'
);

INSERT INTO spark_paimon VALUES (1, 'spark-paimon-1'), (2, 'spark-paimon-2');

select * from spark_paimon;
```

操作结果

Hive示例

前提条件

已完成创建 BMR 集群，并且配置了 Paimon、Hive 组件，详情请参见创建集群。

注意事项

- 由于 Paimon 已经放到 ${HIVE_HOME}/auxlib 目录，所以不需要使用 add jar 就可以直接操作 spark 创建的 paimon 表。
- Hive 不能使用 TEZ 引擎插入数据，并且需要关闭 cbo。参考以下命令：

```
 set hive.execution.engine=mr;
set hive.cbo.enable=false;
```

操作示例

Hive表

1. SSH登录集群，参考SSH连接到集群；
2. 执行以下命令查看结果：

```
 drop table if exists hive_paimon;
CREATE TABLE hive_paimon(
    id INT COMMENT 'The id field',
    name STRING COMMENT 'The name field',
    PRIMARY KEY (id) NOT ENFORCED

) CLUSTERED BY (id) INTO 4 BUCKETS
STORED BY 'org.apache.paimon.hive.PaimonStorageHandler';

INSERT INTO hive_paimon VALUES (3, 'hive-Paimon-3');

select * from hive_paimon;
```

Spark 创建的表

1. SSH登录集群，参考SSH连接到集群；
2. 执行以下命令查看结果：

```
 show tables;
select * from spark_paimon;
INSERT INTO spark_paimon  values (3, 'spark-Paimon-3');
select * from spark_paimon;
```

StarRocks示例

前提条件

- 已完成创建 BMR 集群，并且配置了 Paimon、Hive 组件，详情请参见创建集群。

- 已完成创建 StarRocks 是自建集群，将 BMR 集群 core-site.xml 和 hdfs-site.xml 配置文件 已复制到 StarRocks 集群 FE 和 FE 的 conf 下。

注意事项

- Paimon Catalog 仅支持查询 Paimon 数据，不支持针对 Paimon 的写/删操作。

StarRocks读示例

1. SSH登录集群，参考SSH连接到集群；
2. 创建 paimon 表，参考以下命令：

```
spark-sql --master local[2]
```

```
USE paimon;
USE default;
drop table if exists spark_paimon;
-- 如果没有 bucket，对 spark 没有影响，hive 可以读，但是不能写入。
create table spark_paimon (
    id int,
    name string
) tblproperties (
    'primary-key' = 'id',
    'bucket' = '4'
);

INSERT INTO spark_paimon VALUES (1, 'spark-paimon-1'), (2, 'spark-paimon-2');

select * from spark_paimon;
```

3. 结果显示如下：

```
2 spark-paimon-2
1 spark-paimon-1
```

4. StarRocks 创建 Catalog，读取 Paimon 表，参考以下命令：

```
CREATE EXTERNAL CATALOG srpaimon PROPERTIES
(
    "type" = "paimon",
    "paimon.catalog.type" = "hive",
    "paimon.catalog.warehouse" = "hdfs://bmr-cluster/warehouse/tablespace/managed/hive/spark_paimon",
    "hive.metastore.uris" = "thrift://master-fba3efd-1:9083,thrift://master-fba3efd-2:9083"
);
SET CATALOG srpaimon;
use default;
 select * from spark_paimon;
```

5. 结果显示如下：

```
+------+---------------+
| id   | name          |
+------+---------------+
|    1 | spark-paimon-1 |
|    2 | spark-paimon-2 |
+------+---------------+
```

导入 Paimon 示例

1. 创建 StarRocks 表，并插入数据，参考以下命令：

```
CREATE DATABASE srpaimontest;
use srpaimontest;
create table srtest (
    id int,
    name string
);
INSERT INTO srtest VALUES (3, 'spark-paimon-3'), (4, 'spark-paimon-4');
select * from srtest;
```

2. 结果显示如下：

```
+------+---------------+
| id   | name          |
+------+---------------+
|    4 | spark-paimon-4 |
|    3 | spark-paimon-3 |
+------+---------------+
```

3. 导入 paimon 表数据，参考以下命令：

```
INSERT INTO srtest SELECT * FROM srpaimon.`default`.`spark_paimon`;
select * from srtest;
```

4. 结果显示如下：

```
+------+---------------+
| id   | name          |
+------+---------------+
|    1 | spark-paimon-1 |
|    2 | spark-paimon-2 |
|    3 | spark-paimon-3 |
|    4 | spark-paimon-4 |
+------+---------------+
```

联合查询示例

前提条件

已完成创建 BMR 集群，并且配置了 Paimon、Spark 组件，详情请参见创建集群。

操作示例

基于Hive元信息联合查询

1. SSH登录集群，参考SSH连接到集群。

2. 创建 Paimon 表，参考以下命令：

```
spark-sql
USE paimon;
USE default;
drop table if exists spark_paimon;
-- 如果没有 bucket，对 spark 没有影响，hive 可以读，但是不能写入。
create table spark_paimon (
    id int,
    name string
) tblproperties (
    'primary-key' = 'id',
    'bucket' = '4'
);

INSERT INTO spark_paimon VALUES (1, 'spark-paimon-1'), (2, 'spark-paimon-2');

select * from spark_paimon;
```

3. 结果显示如下：

```
+------+---------------+
| id   | name          |
+------+---------------+
|    1 | spark-paimon-1 |
|    2 | spark-paimon-2 |
+------+---------------+
```

4. 创建 Hive 表，参考以下命令：

```
 use default;
CREATE TABLE hive_table (
    id INT,
    age INT
);

INSERT INTO hive_table VALUES (1, 10), (2, 20);
select * from hive_table;
```

5. 结果显示如下：

```
 OK
1 10
2 20
```

6. Hive 执行联合查询，参考以下命令：

```
 SELECT a.id, a.age, b.name
FROM hive_table a
JOIN spark_paimon b
ON a.id = b.id;
```

7. 结果显示如下：

```
 OK
2 20 spark-paimon-2
1 10 spark-paimon-1
```

8. Spark 执行联合查询 ，参考以下命令:

```
 SELECT a.id, a.age, b.name
FROM spark_catalog.default.hive_table a
JOIN paimon.default.spark_paimon b
ON a.id = b.id;
```

9. 结果显示如下：

```
 1 10 spark-paimon-1
2 20 spark-paimon-2
```

基于Filesystem元信息联合查询

1. SSH登录集群，参考SSH连接到集群；

2. 创建 Paimon 表，参考以下命令：

```
 SET spark.sql.catalog.paimon_fs=org.apache.paimon.spark.SparkCatalog;
SET spark.sql.catalog.paimon_fs.warehouse=hdfs://bmr-cluster/warehouse/paimon/spark;
SET spark.sql.catalog.paimon_fs.metastore=filesystem;

use paimon_fs;
create table fs_paimon (
    id int,
    name string
) tblproperties (
    'primary-key' = 'id',
    'bucket' = '4'
);

DESCRIBE FORMATTED  fs_paimon;
```

3. 结果显示如下：

```
 id                  int
name                 string

##### Metadata Columns
__paimon_file_path   string
__paimon_row_index   bigint
__paimon_partition   struct<>
__paimon_bucket      int

##### Detailed Table Information
Name                 default.fs_paimon
Type                 MANAGED
Location             hdfs://bmr-cluster/warehouse/paimon/spark/default.db/fs_paimon
Provider             paimon
Owner                hive
Table Properties     [bucket=4,path=hdfs://bmr-cluster/warehouse/paimon/spark/default.db/fs_paimon,primary-key=id]
```

4. 设置元信息，参考以下命令:

```
 SET spark.sql.catalog.paimon_fs.metastore;
```

5. 插入数据，参考以下命令：

```
 INSERT INTO fs_paimon VALUES (1, 'fs-paimon-1'), (2, 'fs-paimon-1');
select * from fs_paimon;
```

6. 执行联合查询（spark_table 为 spark-sql 创建）：

```
 SELECT a.id, a.age, b.name
FROM spark_catalog.default.spark_table a
JOIN paimon_fs.default.fs_paimon b
ON a.id = b.id;
```

7. 结果显示如下：

```
 1 10 fs-paimon-1
2 20 fs-paimon-1
```

8. 执行联合查询（hive_table 为hive创建）：

```
 SELECT a.id, a.age, b.name
FROM spark_catalog.default.hive_table a
JOIN paimon_fs.default.fs_paimon b
ON a.id = b.id;
```

9. 结果显示如下：

```
 1 10 fs-paimon-1
2 20 fs-paimon-1
```

操作指南

集群配置

创建集群

BMR提供两种创建集群的方法：创建自定义集群、使用系统预定义模板创建集群。BMR为用户保留集群的历史记录长达一年，涵盖了运行中、已释放和已终止等各种状态的集群。超过一年的历史集群记录将不再被保留。

创建自定义集群

1. 登录控制台，选择**产品服务-MapReduce BMR**，进入集群列表页。

2. （可选）选择区域，区域说明请参考区域选择说明。在不同区域创建的集群相互独立。

3. 点击**创建集群**，进入创建集群页。

选择集群配置

集群基础设置说明

表一 集群基础设置配置项说明

配置项	配置项说明
集群类型	目前MapReduce支持三种集群类型，不同类型对应不同资源规格和可选服务，请根据实际业务需要选择进行集群创建默认 Hadoop 集群类型
集群版本	BMR发行版是依据开源社区版本、客户需求进行各类服务的统一镜像，各服务的版本由集群版本决定
必选服务	指依据不同集群类型，用户必须安装的服务。例如，hadoop类型集群必选服务为：hdfs、yarn、mapreduce、zookeeper、ldap
可选服务	指依据不同集群类型，用户可自定义选择的服务。例如，hbase类型集群可选服务为：yarn、mapreduce、ranger
安全模式	开启后，集群中的组件以Kerberos安全模式启动，支持统一的 集群安全管理方案
日志	自动收集应用运行日志，支持检索和问题定位。 地址格式：bos://{bucket_name}，bucket需手动创建，folder输入后系统会自动创建

集群标签

标签支持您按各种标准（如用途、所有者或项目）对资源进行分类；每个标签包含键和值两部分。默认一个标签如有需要点击下方**添加标签**进行添加。

基础配置说明

表二 基础配置项说明

配置项名称	配置项说明
集群名称	用于区分不同集群的唯一性。
管理员密码	该密码可以用于登录创建的集群机器，请设置复杂密码，必须以字母、数字、特殊字符组合，长度在8-17个字符之间。
确认管理员密码	再次输入管理员密码。
管理员用户名	root。用户远程SSH到集群进行管理。
短信开关	默认开启。 是否启用免费短信提醒服务，启动后系统将发送报警短信给指定的联系人。

实例组配置

付费方式和地区

1. 选择付费方式：

- 预付费：以自然年或自然月为计费周期支付订单，先付费后使用服务，价格较后付费更低
- 后付费：按分钟实时计费，按小时扣费，先使用服务后付费

2. 自动终止开启/关闭。

开启时，当最后一个作业执行完毕后，自动释放集群，否则集群会一直保持运行，需手动释放。并且可选择超时设置开启/关闭，开启后，可以设置集群的最大运行时间，如果超过，集群会被释放。

3. 当前区域展示选择后区域，如需修改购买其他地域产品，请前往主导航进行切换。

4. 选择可用区后，填写网络配置和实例组配置。

网络配置

表三 网络配置说明

配置项名称	配置项说明
内网DNS解析	内网DNS为百度云提供的独立的网络产品，可以实现域名（主机名）与ip之间的自动解析。对于不同集群间，BMR集群与BCC间，需要通过域名（主机名）进行互相访问，数据传输时，可以使用该功能。详情请查看智能云解析DNS。 开启内网DNS的BMR集群将自动关联内网DNS产品，自动加入名为novalocal的私有域（若无该私有域，将自动创建）开启内网DNS解析后，可以实现集群间数据通信。开启后BMR集群将自动关联内网DNS，若无DNS，则会自动创建。 需要注意的是： 1.开启该功能，智能云解析DNS（内网DNS）产品，将根据使用量进行收费。当前智能云解析DNS（内网DNS）处于公测免费阶段，详情请查看产品定价。 2.该功能开启后，暂不支持关闭，请合理规划集群使用用途。释放集群时，为保证其他集群（如有）的正常使用，关联的内网DNS私有域不会自动释放，请您在确定没有其他集群使用该功能时，进入智能云解析DNS控制台，将相关资源进行释放，以免带来不必要的计费。 3.当通过子用户创建BMR集群时，开启内网DNS功能，需要您进入多用户访问控制中，为该用户授予“LDFullControlPolicy”权限，否则该功能无法使用。
网络类型	若VPC未进行域名配置，会导致您无法成功创建集群，请前往VPC控制台确认VPC域名已被正确配置。
安全组	安全组具有防火墙功能，用于设置云服务器 CVM 的网络访问控制。 MapReduce自动默认一个安全组配置，该配置影响集群机器的安全组配置，非必要不进行修改。
自动终止	默认关闭。 开启时，当最后一个作业执行完毕后，自动释放集群，否则集群会一直保持运行，需手动释放。

表四 实例组配置说明

配置项名称	说明
高可用	开启后，集群拥有三个master实例，支持核心组件自动故障转移。
计算类型及实例组	MASTER节点：支持通用型、计算型、内存型。购买数量非高可用集群为1个，高可用集群为3个。 Task节点：支持通用型、计算型、内存型、本地SSD型、大数据型。购买数量默认为0个，最大可设置50个。控制台上一个集群仅能添加一组Task节点，请规划好套餐设置 CORE节点：支持通用型、计算型、内存型、本地SSD型、大数据型。购买数量默认为2个，最少设置2个，最大可设置200个。 Client节点：支持通用型、计算型、内存型。购买数量默认为0个，最大可设置10个。
系统盘	系统盘类型支持高性能云磁盘、SSD云磁盘和增强型云磁盘。容量为40-500GB，ssd（免费赠送40GB，超出部分需要单独付费）。
数据盘	数据盘类型支持高性能云磁盘、SSD云磁盘和增强型云磁盘。数目默认为1，最大可设置为5。 容量：50-32765GB， ssd（SSD云磁盘 / 高性能云磁盘/增强型云磁盘）。选择说明

4. 如果出现规格售罄情况，建议尝试更换区域和可用区。在付费方式和地区切换区域和可用区，查看其它地域内是否有您的服务器。

确认订单 表五 确认订单说明

显示项	显示项说明
BMR集群信息	显示配置完成的BMR集群配置信息，确认是否无误。
代金券选择	代金券分预付费和后付费： • 预付费订单在这一步选择代金券，如有代金券可选择激活。也可进入代金券管理进行详细操作。 • 后付费订单如果您有代金券，产生账单后会优先从可用代金券中扣除；如果有折扣，会在实际产生的账单中体现。
支付订单	1.确认无误后，点击去支付后跳转订单支付界面。页面会再次展示订单详情，请您再次核对订单无误后选择具体的支付方式进行支付。 2.确认支付完成后，系统会更新订单状态，此时集群创建完毕。

使用系统预定义模板创建集群

1. 在**产品服务->MapReduce-集群模板**页中，在集群模版列表中，在操作列点击**创建集群**。具体模板创建详见集群模版。
2. 进入配置界面，输入之前创建模板时的管理员密码后点击**下一步**。
3. 完成订单确认和支付后可完成创建集群。

附录

表六 组件依赖说明

组件名称	依赖组件
ranger	ranger-plugin
yarn	hdfs、zookeeper
hdfs	zookeeper
rapidfs	hdfs、bos
mapreduce2	hdfs、yarn
tez	hdfs、yarn
spark2	hfds、yarn
spark3	hdfs、yarn
flink	hdfs、yarn
hive	hdfs、yarn、tez
hbase	hdfs、zookeeper
pig	hdfs、yarn
presto	通常和 hive 元数据搭配使用
impala	通常和 kudu 搭配使用
kudu	通常和 impala 搭配使用
kyuubi	spark
hudi	通常和 spark、flink 搭配使用
iceberg	通常和 spark、flink 搭配使用
detalake	spark
oozie	hdfs yarn

配置已有集群

对于已创建的集群，用户可以在产品服务中的**MapReduce-集群列表**页面查看所有集群的状态和详情。并对集群做一些具体操作，下面是关于所有操作的具体描述。

集群详情

1.登录MapReduce控制台，在集群列表中选择对应的**集群 ID/名称**点击进入集群详情页。

2.在详情界面的集群详情分别可以查看关于集群的基本信息、配置信息、相关工具、网络和实例信息，实例信息可以展开查看详细信息。

修改集群名称

目前MapReduce支持两种方式修改集群名称：

1.在**产品服务-MapReduce-集群列表**页，在集群名称/ID列，鼠标移入集群名称直到出现右侧的变更图标并单击，在弹出的对话框内输入集群的新名称，点击**确定**即可。

2.在**产品服务-MapReduce-集群列表**页，点击**集群名称**进入集群详情页，单击集群名称对应的**变更**按钮，在弹出的对话框内输入集群的新名称，点击**确定**即可。

集群续费

MapReduce目前支持**自动续费**和**手动续费**。使用预付费方式购买的包年包月集群，在集群状态为“运行中”和“已停服”时可续费。

表一 续费说明

续费类型	续费说明
自动续费	进入集群列表界面，在页面第一行找到**自动续费**点击进入续费管理。
手动续费	1.在**产品服务-MapReduce-集群列表**页，点击需续费集群对应的**续费**按钮，进入该集群的续费页。 2.选择续费的时长，点击**下一步**进入订单确认页。 3.确认订单信息和是否使用代金券，无误后点击去支付进入支付页面。根据页面提示选择支付方式完成支付即续费成功。

计费变更

为了满足用户的业务需要，BMR支持对集群实例的计费方式进行灵活切换。按需计费（后付费）方式与包年包月计费（预付费）方式可以相互切换。

在**产品服务-MapReduce-集群列表页**，点击需续费集群对应的计费变更按钮，进入该集群的计费变更订单页。计费变更分预付费转后付费或后付费转预付费：

- 预付费转后付费：进入计费变更界面，选择完毕后根据系统指引确认订单然后订单支付后即可变更成功
- 后付费转预付费：进入计费变更界面，选择预付费购买时长（按月/按年，1-12月/1-3年），选择完毕后根据系统指引确认订单然后订单支付后即可变更成功

规模调整

1. 在**产品服务->MapReduce-集群列表**页，点击被调整集群对应的**规模调整**操作，进入该集群的变更配置页。
2. 变更类型支持数目变更、规格变更、磁盘变更和新增实例类型。

表二 变更类型说明

变更类型	变更类型说明
数目变更	master不支持变更数目、core仅支持扩容、task和client支持扩缩容。
规格变更	本地盘实例不支持规格变更。变更需要重启实例，业务会短暂中断，建议在业务不繁忙时操作。
磁盘变更	master和core仅支持扩容、task和client支持扩缩容。
新增实例类型	支持新增未被添加的实例组类型。

3. 规模调整结束后点击确认变更，付费方式不同确认界面也不相同：

- 后付费方式购买的集群：已完成集群规模变更
- 预付费方式购买的集群：进入确认订单页，确认订单价格和是否使用可用代金券后完成支付即可

> 说明

- 预付费方式购买的包年包月集群，不允许扩增到期时间小于24小时的集群
- 串行处理扩容缩容的请求，因此处理完一个扩容缩容的请求后，才接受下一个扩容缩容请求
- 集群在初始化或者释放中的时候，以及欠费停服务的集群，是不能处理扩容缩容请求的
- 扩增集群节点时可以释放集群，扩增的节点也均被释放
- 集群扩增节点的过程中不影响作业的提交和运行
- 新增加的task节点上没有datanode服务，属于任务节点
- 在集群运行作业期间提交的缩减task节点的请求，会在作业运行完之后处理

复制集群

1. 在**产品服务-MapReduce-集群列表**页，点击被复制集群对应的**复制**按钮，进入该集群的配置页。
2. 若完全复制集群，点击“完成”即可；也可更改集群的设置后点击“完成”。建议更改日志文件的储存地址，便于区分集群日志。

释放集群

释放集群仅可释放付费集群，不可释放预付费集群。如需释放预付费集群请提交工单支持。--链接

操作步骤

1.目前支持两种入口释放集群：

- 进入MapReduce登陆台界面，在集群列表中找到需要释放的集群，单击操作列中的释放按钮确认后可释放集群。
- 找到集群详情中右上角释放按钮，点击确认后即可释放。

2.集群的状态由“运行中”变成“已终止”，最后是“已终止”意味着集群释放成功。

删除已释放集群

用户已释放的BMR集群仍然会保留在集群列表中，方便用户回溯和查看历史集群的一些信息，但是用户也可手动删除已释放的集群。

注意：删除集群列表中的集群可以删除【已终止】和【已终止但有错误】状态的集群，【已终止但有错误】状态为方便用户和BMR追溯创建集群失败原因会保留。处于【运行中】的集群不可删除。

弹性伸缩

大数据处理及分析场景下，常常需要根据业务情况的变化动态调整集群的task节点数量，这样可以在保证作业顺利完成的同时，降低您的成本。弹性伸缩功能支持按时间规则或者按指标规则进行集群task节点规模的调整。适用于以下场景：

- 业务规模具有时间周期规律，存在明显的波峰与波谷，例如特定时间的日报表、周报表等处理分析场景
- 业务变化不具有时间规律，但需要保证重要作业的及时运行，需要根据集群负载指标动态调整集群规模

配置节点套餐

1. 选择**产品服务>数据分析>MapReduce>集群**进入集群列表页面。
2. 点击所选集群后方操作列中的**弹性伸缩**按钮，进入管理弹性伸缩页面。

3. 在扩容节点配置栏中，点击**新增节点类型**，在弹框中选择节点套餐，并配置系统盘及数据盘，点击确认。
4. 节点配置选择后，可点击**修改**或**删除**，重新配置节点套餐。节点套餐配置完成后，点击**保存**，则完成节点套餐配置操作。
5. 释放所有扩容节点：在弹性伸缩页面，点击页面右上角的**释放所有节点**，即可释放当前已扩容的节点，并且所有规则将处于已失效的状态，不会触发伸缩操作。

> 说明：节点套餐一经保存后，在规则运行期间不允许修改套餐。若要修改套餐，需要点击页面右上角的**重置并释放所有节点**，将已扩容的节点释放并重置套餐类型。

管理伸缩规则

配置规则

伸缩规则分为**时间规则**与**指标规则**两种类型，两种规则间互相独立，不可同时生效。规则切换后，原有规则将会失效不再触发，已扩容的节点不会被释放，直到触发新的缩容规则。点击**规则类型名称**后，可进行规则切换。

1.单击**创建扩容规则**可以开始创建规则，下面详细介绍不同规则的具体配置说明。

注意：创建规则前请先配置节点套餐保存。

- 时间规则

配置项	配置项说明
规则名称	必须以字母开头，支持字母、数字、特殊字符- _ /，长度在1-65个字符之间。
规则类型	不可选，与当前规则类型相同。
执行次数	可选重复执行或仅执行一次。
执行时间	重复执行时，先选择重复周期“每天”、“每周”、“每月”，再按照时-分-秒的格式选择执行的时间点；仅执行一次时，选择具体日期，再按照时-分-秒的格式选择执行的时间点。
伸缩行为	伸缩行为只会在限制的节点数范围内执行。

- 指标规则

配置项	配置项说明
规则名称	必须以字母开头，支持字母、数字、特殊字符- _ /，长度在1-65个字符之间。
规则类型	不可选，与当前规则类型相同。
集群指标项	选择所要监控的集群指标项；选择指标数据的统计规则，当前可选平均值、最大值、最小值。
统计周期	评估所选集群指标项是否满足条件的时间周期，例如当5分钟统计周期内，Yarn内存使用量的平均值>80%时，触发规则。
阈值	选择运算符号“>”、“<”；配置阈值数字。
伸缩行为	为该条规则触发时扩容节点数量。
集群指标项定义	Yarn： • YarnMemoryPercentage：Yarn内存使用百分比 • YarnAppsPending：Yarn中等待运行的任务数 Cluster： • CpuUsagePercentage：集群平均cpu使用百分比

修改规则

修改规则需要进入弹性伸缩界面，在具体规则可以选择不同操作项（失效、编辑、删除）对规则进行修改。

表一 操作项具体说明

操作项	操作说明
失效/生效	点击操作列中的失效/生效，可切换该条规则的状态。
编辑	点击操作列中的编辑，可重新编辑规则。
删除	点击操作列中的删除，可删除该条规则。
规则状态	有两种状态。“生效中”，为该规则可被正常触发并执行相应操作；“已失效”，为该规则不可被触发。

查看伸缩日志

1. 选择**产品服务>MapReduce>集群列表**，点击集群名称进入集群详情页面。
2. 在侧边导航**点击弹性伸缩日志**按钮，查看日志记录。

表二 字段说明

字段名称	字段说明
开始时间	扩容、缩容操作开始的执行时间。
结束时间	扩容、缩容操作结束的时间。若操作失败，则展示“无”。
规则名称	所触发的规则名称。
节点类型	实例配置时选择的实例类型。
伸缩行为	该条操作实际执行的节点数量。
执行状态	该条操作执行的结果。
执行完节点数目	该条操作执行完后的弹性伸缩task节点总数。
操作	单击查看详情按钮，可查看伸缩规则的具体信息，包括规则类型、伸缩行为、执行方式和执行时间。

集群安全模式

BMR从1.0.0版本开始支持创建安全类型的集群，即集群中的开源组件以Kerberos的安全模式启动，在这种安全环境下只有经过认证的客户端才能访问集群中的服务(如HDFS，HIVE 等)。

创建安全集群

在集群创建页面进行集群配置时打开安全模式按钮即可。

图一 开启集群安全模式

Kerberos身份认证原理

Kerberos是一种基于对称密钥技术的身份认证协议，它作为一个独立的第三方的身份认证服务，可以为其它服务提供身份认证功能，且支持SSO(即客户端身份认证后，可以访问多个服务如HBase/HDFS等)。

Kerberos协议过程主要有两个阶段，第一个阶段是KDC完成对Client的身份认证，第二个阶段是Service对Client的身份认证。

表一 身份认证说明

阶段	阶段说明
KDC 对 Client 的身份认证	当客户端用户(principal)访问一个集成了Kerberos的服务之前，首先需要通过KDC（kerberos的服务端程序）的身份认证。客户端用户通过 KDC 服务的身份认证后，会拿到一个TGT凭证(Ticket Granting Ticket，默认时效为24小时)，后续访问集成了 kerberos 的服务时会使用该凭证。
Service 对 Client 的身份认证	当客户端用户访问集成 kerberos 的 Service服务时，它会使用TGT以及需要访问的服务名称(如HDFS)去KDC换取TGS(Ticket Granting Service)，然后使用TGS去访问Service，Service会利用相关信息对Client进行身份认证，认证通过后便可以正常访问Service服务。

Kerberos 常用命令指南

管理员登录

登录到管理员账户: 如果在BMR集群节点上，root 账户可以直接通过kadmin.local命令登录：

```
 $ kadmin -p admin/admin
Password for admin/admin@BAIDU.COM: (BMR集群默认密码为：hadoop)
```

增删改查账户

在管理员的状态下使用addprinc/delprinc/modprinc/listprincs 命令管理用户：

```
 ##### 增加 allen 用户
kadmin: addprinc allen
Enter password for principal "allen@BAIDU.COM":
Re-enter password for principal "allen@BAIDU.COM":
Principal "allen@BAIDU.COM" created.

##### 为用户 allen 生成 keytab: 使用xst命令或者ktadd命令
kadmin: ktadd -k /etc/security/keytabs/allen.keytab allen
Entry for principal allen with kvno 2, encryption type aes256-cts-hmac-sha1-96 added to keytab WRFILE:/etc/security/keytabs/allen.keytab.
Entry for principal allen with kvno 2, encryption type aes128-cts-hmac-sha1-96 added to keytab WRFILE:/etc/security/keytabs/allen.keytab.
```

查看当前认证的用户

```
 $ klist
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: admin/admin@BAIDU.COM

Valid starting       Expires              Service principal
2018-09-27T12:06:07  2018-09-28T12:06:07  krbtgt/BAIDU.COM@BAIDU.COM
```

通过 keytab 文件的方式认证用户

```
kinit -kt /etc/security/keytabs/allen.keytab allen
```

删除当前认证用户的缓存

```
kdestroy
```

BMR安全集群实践

1.初次使用 KDC 服务时，需要集群管理员登录 master 机器，执行 kadmin.local 命令创建客户端用户的 principal及密码(或者生成 keytab 文件，且注意该文件的属主及访问权限)。

2.因为 BMR 集群使用的 KDC 类型为 MIT KDC，创建的客户端用户 principal 最好与 UNIX 账户存在一一映射关系（尤其在部署有 Ranger 组件的 BMR 集群中，这源于 BMR 中的 Ranger UserSync 使用的是 UNIX 账户同步机制）。 例如，管理员首先在 BMR 集群各节点中创建 UNIX 用户 allen：

```
 $ useradd allen
```

3.管理员新增一个名为 allen/_HOST@BAIDU.COM 的 principal，如下所示：

```
[root@ng6c79765-master-instance-fh2qdy0g ~]# kadmin.local
Authenticating as principal jack/admin@BAIDU.COM with password.
kadmin.local:  addprinc allen/ng6c79765-master-instance-fh2qdy0g.novalocal@BAIDU.COM
WARNING: no policy specified for allen/ng6c79765-master-instance-fh2qdy0g.novalocal@BAIDU.COM; defaulting to no policy
Enter password for principal "allen/ng6c79765-master-instance-fh2qdy0g.novalocal@BAIDU.COM":
Re-enter password for principal "allen/ng6c79765-master-instance-fh2qdy0g.novalocal@BAIDU.COM":
Principal "allen/ng6c79765-master-instance-fh2qdy0g.novalocal@BAIDU.COM" created.
kadmin.local:  q
[root@ng6c79765-master-instance-fh2qdy0g ~]#
```

4.登录 allen 账户（allen 为 UNIX 账户，如果有向集群YARN队列资源提交作业的需求，在启动安全模式下，还需要保障集群中各个节点已存在该账户，否则提交的 yarn 任务会提示用户不存在而运行失败），执行 kinit 指令获取初始有效凭证 TGT（默认有效期为24h）。

```
[allen@ng6c79765-master-instance-fh2qdy0g ~]$ klist
klist: No credentials cache found (filename: /tmp/krb5cc_1018)
[allen@ng6c79765-master-instance-fh2qdy0g ~]$ kinit -p allen/ng6c79765-master-instance-fh2qdy0g.novalocal
Password for allen/ng6c79765-master-instance-fh2qdy0g.novalocal@BAIDU.COM:
[allen@ng6c79765-master-instance-fh2qdy0g ~]$ klist
Ticket cache: FILE:/tmp/krb5cc_1018
Default principal: allen/ng6c79765-master-instance-fh2qdy0g.novalocal@BAIDU.COM

Valid starting       Expires              Service principal
2018-11-15T10:34:55  2018-11-16T10:34:55  krbtgt/BAIDU.COM@BAIDU.COM
[allen@ng6c79765-master-instance-fh2qdy0g ~]$
```

5.完成 kerberos 认证之后，接下来就可以正常访问集群中的服务了（比如 HDFS、YARN、HIVE等）。

用户管理

概述

实际业务处理场景中，经常遇到需要将BMR集群资源及集群中的数据、服务等在多个组织及部门间共享的场景，需要能够做到集群中细粒度的计算资源及访问权

限管控。 对于云上资源（集群、作业）粒度的管理，请参见多用户访问控制。 用户管理功能提供了BMR集群中从账号、认证、授权、审计四个纬度的统一安全方案，满足企业级细粒度多租户及安全管控需求。并且支持与IAM账号做关联打通，易于使用。

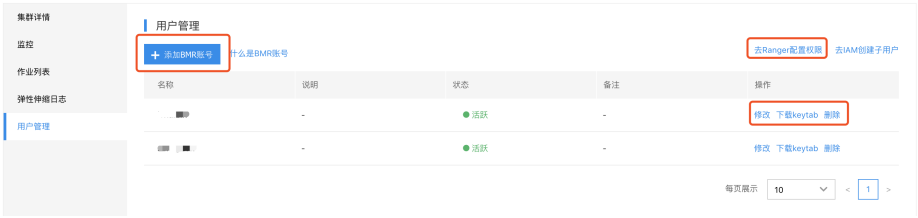
管理BMR账号

BMR账号是BMR集群中LDAP、Kerberos、Linux、Ranger等几个模块的打通的同用户名的身份集合。

> **说明**
>
> - BMR账号所关联的身份会根据集群中实际启动的组件情况决定。
> - 启动Kerberos，请参见集群安全模式。
> - LDAP为默认启动组件，当前已支持hadoop、hbase模版。
> - Ranger为创建集群时的可选组件。

1. 在“产品服务->MapReduce-集群列表”页，点击集群名称，进入该集群的集群详情页。
2. 在集群详情页，点击左侧“用户管理”按钮，管理BMR账号。
3. 点击“添加BMR账号”，可以选择已有的IAM子用户，为该用户创建集群中同用户名的BMR账号。
4. 点击“修改”，可以修改BMR账号的密码及备注信息。
5. 点击“下载keytab”，可以下载该账号对应的keytab，用作集群中提交作业时使用。
6. 点击“删除”，可以删除该BMR账号。
7. 点击表格右上角“去Ranger配置权限”，为创建的BMR账号配置在集群中的组件及数据访问权限。请注意，若不配置权限，默认无法使用组件及访问数据。Ranger的初始账号密码为admin、rangeradmin123，请管理员用户及时修改密码。

> **说明**
>
> - 开启Ranger的情况下，需要为创建的BMR账号在Ranger中设置相关策略，否则无法在console以及集群中提交作业等。Ranger配置详情请查看Ranger配置。
> - 只有admin账号才可在Ranger中为其他BMR账号配置权限，请注意及时修改并保存好密码。
> - 若在多用户访问控制中，将IAM子用户的状态设置为“禁用”，则该IAM子用户关联的BMR账号无法进行任何操作。



安全组

安全组是在VPC网络内为BCC实例和DCC专属实例、负载均衡实例、云数据库实例中创建的安全防火墙，定义IP＋端口的入站和出站访问策略，从而提高云服务器、负载均衡、云数据库等实例的安全性。安全组分为普通安全组和企业安全组，功能概述如下：

表一 功能说明

功能	普通安全组	企业安全组
安全组无任何规则	入方向：拒绝所有访问请求 出方向：拒绝所有访问请求	入方向：拒绝所有访问请求 出方向：拒绝所有访问请求
新建安全组默认规则	入方向：拒绝所有访问请求 出方向：允许所有访问请求	入方向：拒绝所有访问请求 出方向：允许所有访问请求
安全组规则策略	仅支持允许策略	支持允许、拒绝策略
设置规则优先级	纯白名单机制，无优先级	取值范围1-1000，可以与已有条目重复。数值越小，优先级越高，规则匹配顺序为按优先级由高到低匹配。若拒绝与允许的优先级相同，则拒绝优先
Source或Target选择安全组	支持	不支持
关联实例数量	无限制	无限制
实例关联多个安全组	所有规则逐一匹配，有一条规则允许则允许	将关联的多个企业安全组中的所有规则按优先级重新排序，按优先级大小匹配规则，若允许拒绝策略的优先级相同，则拒绝优先
使用场景	适用于运维成本更低的场景	适用于网络控制更精细化的场景

创建安全组

1.选择**产品服务>私有网络 VPC**，在左侧导航栏选择**安全组**。

2.点击**创建安全组**，进入“创建安全组”界面。

3.选择所在网络，根据要求输入安全组名称和描述。

4.选择端口设置方式，入站和出站的规则设置分为两个独立页签，规则设置有如下方式：

- “允许访问所有端口”为OFF状态，可选择“添加规则”进行入站和出站的规则设置，设置时可以通过选择右侧的快捷模板进行快捷设置。
- “允许访问所有端口”为OFF状态且没有添加规则，将导致云服务器无法与外界通信。此时只能通过远程VNC登录访问云服务器，因此请慎重选择该选项。
- 允许访问所有端口”为ON状态，将使云服务器的所有服务端口完全暴露在网络环境下，此时可能会面临一定的安全风险，因此请慎重选择该选项。

5.点击“确定”，弹出“安全组详情”页面，完成安全组的创建。

使用安全组

1.登录百度智能云官网，点击上方的“产品”，选择**智能大数据>数据计算>MapRedcue**。

2.进入MapRedce详情页，首次使用需要为BMR服务进行授权开通相应的服务，选择开通服务即可。

3.在进入MapReduce产品详情页之后，点击创建集群，在创建集群页面的基础配置中可以看到相应的安全组配置项。如果您已经是百度智能云的用户并且在BMR集群同一个VPC下拥有其他安全组，用户也可以选择自己的安全组。但是用户自选的安全组需要符合BMR集群安全组使用的基本要求才能成功创建集群，否则将无法成功创建BMR集群。

4.在基础配置中可以看到安全组配置信息，建议用户使用默认的BMR安全组。

图一 集群配置安全组界面

5.在完成配置支付订单完成后，在集群列表中点击集群名称进入集群详情，可在网络配置查看安全组的名称。

编辑安全组

用户可对安全组名称、描述、端口设置和关联服务器等信息进行编辑。

注意：为了保障系统安全，建议用户不要更改“默认安全组”的相关配置，如果需要其他权限的安全组机制，可以新建安全组并与云服务的实例进行绑定。

1. 选择**产品服务>私有网络 VPC**，在左侧导航栏选择**安全组**。
2. 点击**安全组名称**，进入安全组详情页面。
3. 针对需要修改规则的“协议”，点击对应操作后的**编辑**按钮可以进行修改。

安全组规则

导入

用户可将导出的安全组规则文件导入到安全组中，用于安全组的快速创建及恢复。

1. 选择**产品服务>私有网络 VPC**，在左侧导航栏选择**安全组**。
2. 点击**安全组名称**，进入安全组详情页面。
3. 点击**导入**，选择已有.csv格式的安全组规则文件。
4. 点击**确认**，导入安全组规则。

导出

用户可将安全组规则导出，用于本地备份。

1. 选择**产品服务>私有网络 VPC**，在左侧导航栏选择**安全组**。
2. 点击**安全组名称**，进入安全组详情页面。
3. 点击**导出**，将规则文件保存到本地。

修改安全组规则

在成功创建BMR集群后，如果用户想要修改BMR使用的默认安全组规则，可以登录控制台，选择**私有网络VPC**，选择相应的安全组，并对其进行规则编辑。

例如开通22端口，操作步骤如下：

1.选择**私有网络vpc**，选择进入相应BMR集群的安全组中进行修改。BMR集群安全组ID可以在BMR集群详情页中进行查看。

2.点击进入VPC安全组中，可以搜索查看相应的BMR集群的安全组。

3.点击**安全组名称**—**BaiduMapReduce-Default** 进入安全组详情。在入站规则中点击添加规则，添加22端口即可，添加完成后可以通过SSH方式使用22端口进行登录访问集群。

4.配置完成后，即可通过22端口成功登录主节点，用户可对集群进行相关操作。也可以在VCP安全组中查看BaiduMapReduce-Default的具体规则。

复制安全组

用户可通过复制安全组功能，快速创建一组同样规则的安全组。

1. 选择**产品服务>私有网络 VPC**，在左侧导航栏选择**安全组**，进入安全组列表页。
2. 在需要复制的安全组后，点击操作列的**复制**按键，弹出“复制安全组”的界面。
3. 点击**确定**，完成安全组信息的复制。

删除安全组

当用户不需要该安全组时，可删除安全组。用户可对未关联的自定义安全组进行直接删除；已关联的安全组需要先取消关联然后才可进行删除。

1. 选择**产品服务>私有网络 VPC**，在左侧导航栏选择**安全组**，进入安全组列表页。
2. 勾选需要删除安全组，点击**删除**按钮。
3. （可选）取消和云服务器实例的关联，具体请参考取消关联安全组。
4. 未关联的安全组，点击“确定”，直接删除该安全组信息。

> 说明： 用户也可以批量删除安全组。在安全组列表中选中需要删除的安全组点击“删除”即可。

关联安全组

用户创建安全组后还需要将该安全组关联绑定到对应的云服务器上，这样该云服务器才能按安全组规则的设定实现网络访问控制功能。可以从安全组侧关联实例，也可以从实例侧关联安全组。

安全组侧关联实例

1. 选择**产品服务>私有网络 VPC**，在左侧导航栏选择**安全组**。

2. 点击**安全组名称**，进入安全组详情页面。

3. 在关联实例列表上方点击**关联云服务器**按键进行关联。

实例侧关联安全组

以关联BCC操作为例，给出详细的操作步骤供参考。

1. 选择**产品服务>云服务器BCC**，可看到用户已创建的云服务器列表。

2. 选择需要关联安全组的实例，当勾选多个实例时，系统将进行批量关联。

3. 点击**关联安全组**按钮，弹出“关联安全组”对话框。

4. 选择需要关联的安全组名称，点击**确定**，完成云服务器实例和安全组的关联。

取消关联安全组

当某个BCC实例需要切换到其它安全组时，需要取消原有安全组的关联关系。一个BCC实例必须关联至少一个安全组。可以从安全组侧取消关联实例，也可以从实例侧取消关联安全组。

安全组侧取消关联实例

1. 选择**产品服务>私有网络VPC**，在左侧导航栏选择**安全组**。

2. 点击**安全组名称**，进入安全组详情页面。

3. 在关联实例列表操作列，点击**取消关联**按键进行关联。

实例侧取消关联安全组

1.选择**产品服务>云服务器BCC**，进入“实例列表”界面。

2.在实例名称下，选择相应的链接，进入实例详情页签。

3.选择**安全组**页签，下拉页面到关联的安全组列表区域。

4.点击**取消关联**，取消实例和相应安全组之间的关联。

EIP

BMR集群中使用EIP

弹性公网IP EIP (Elastic IP) 作为一个独立的商品为用户提供公网带宽服务。 EIP的主要用途包括： 通过EIP实例，用户可以获取公网带宽服务。 用户可灵活配置EIP实例的计费模式，包括按需按带宽付费、按需按流量付费和包年包月按带宽付费三种。 用户可将EIP实例与任意BCC或BLB实例绑定或解绑，匹配用户的不同业务场景。详见：弹性公网 EIP

BMR在创建完集群后，用户在与集群外部进行数据交互或传输时或者进行大数据组件的WEB UI访问时，需要拥有公网IP支持访问和数据传输，此时需要使用EIP。BMR支持用户对所有类型的BMR节点绑定和解绑EIP。BMR中使用的EIP是从客户同一个Region下的EIP列表中获取的，BMR默认不再赠送master节点1M的EIP。如果您还未创建EIP，请先去EIP中创建。详见：创建EIP实例

EIP绑定

在创建完BMR集群后，如果用户想要绑定和解绑EIP，在集群详情的实例信息中绑定和解绑即可。建议用户为大数据创建的EIP只在BMR console内进行解绑，在BMR内解绑EIP不会主动为您释放该EIP，需要您在EIP console中主动释放该EIP。如果您在EIP侧console中释放了与BMR节点绑定的EIP，该节点将无法访问外网。以下是具体操作步骤：

1.在MapReduce服务集群列表中，单击**集群名称**，跳转到集群详情，找到实例信息查看网络IP信息。

2.找到要绑定公网IP的节点，点击**绑定公网IP**，单击确认即可。在集群详情实例信息中可看到该节点已绑定的公网IP地址。

EIP解绑

1.在集群列表点击**集群名称**，进入集群详情。在实例信息部分找到需要解绑到EIP，单击**解除绑定**后确认解除即可解除成功。

2.在公网ip那栏即可看到该实例已经解除公网IP绑定。

集群审计

概述

云审计（BCT，Baidu CloudTrail），是支持对云账户进行监管、合规性检查、操作审核和风险审核的服务。借助 BCT，BMR 可以记录基础设施中的相关操作。 用户可登录 BCT 产品查看事件详情，参考BCT操作指南。

事件

当前 BMR 支持通过 BCT 查看以下操作事件：

操作者	事件类型	事件说明	事件名称	资源类型
主账号/子用户	console	创建集群	CreateCluster	集群
主账号/子用户	console	续费集群	RenewCluster	集群
主账号/子用户	console	变配集群	ResizeCluster	集群
主账号/子用户	console	转后付费	ToPsotpayCluster	集群
主账号/子用户	console	转预付费	ToPrepayCluster	集群
主账号/子用户	console	释放集群	TerminateCluster	集群
主账号/子用户	console	重启服务	Restart {服务名}	服务
主账号/子用户	console	停止服务	Stop {服务名}	服务
主账号/子用户	console	启动服务	Start {服务名}	服务
主账号/子用户	console	停止组件	Stop {组件名}	组件
主账号/子用户	console	启动组件	Start {组件名}	组件
主账号/子用户	console	重启组件	Restart {组件名}	组件
主账号/子用户	console	修改组件配置	UpdatetConfig	组件
主账号/子用户	console	重启虚机	Restart {虚机id}	节点

Hive元数据说明

概述

元数据（Meta Data）在大数据中是描述数据的数据，提供有关数据的信息，帮助用户理解和管理数据。元数据包括有关数据的属性、结构、来源、格式、质量、安全性以及与其他数据的关系等信息。元数据在大数据环境中起到了关键的作用，帮助组织更好地利用和管理其数据资产。 目前大数据领域，常用的元数据服务由Hive Metastore来提供。

Hive元数据

1. 登录控制台，选择产品服务-MapReduce BMR，进入集群列表页。
2. （可选）选择区域，区域说明请参考区域选择说明。在不同区域创建的集群相互独立。
3. 点击创建集群，进入创建集群页。
4. 集群类型选择 Hadoop，可选服务选择 Hive，显示 Hive 元数据配置，支持 DEFALUT（Hive Metasore）、MySQL、EDAP元数据。

Hive Metastore

Hive Metastore最初是 Hive 的关键组件，管理元数据。Hive Metastore需要一个数据库来存储真正的数据。一个集群可以有多个 Hive Metastore，以此提高 Hive Metastore 的高可用性。如果期望多个集群使用同一个元数据库，则数据文件必须存在各个集群都能访问的地方，例如对象存储。

在 BMR 集群中，Hive Metastore 的数据库可以是集群本地MySQL，也可以是外部数据库，例如云数据库RDS，亦可以存在EDAP中。

EDAP 元数据

目前 BMR-3.2.3 及以上发行版支持 EDAP 元数据，支持区域包括 北京、广州、苏州、保定。使用 EDAP 元数据进行统一管理，可以为用户带来以下优势：

1. EDAP元数据稳定性更高，采用中心化管理，不必担心集群出问题导致元数据丢失。
2. EDAP元数据可以让元数据生命周期超越集群生命周期，多个集群共用元数据，并且延伸出更多的管理功能。
3. EDAP元数据+存算分离可以使整个BMR集群变成一个无状态的计算集群，可以随时缩容、释放来节约成本，帮助客户降本增效。

Trino元数据

1. 登录控制台，选择产品服务-MapReduce BMR，进入集群列表页。
2. （可选）选择区域，区域说明请参考区域选择说明。在不同区域创建的集群相互独立。
3. 点击创建集群，进入创建集群页。
4. 集群类型选择 Hadoop，可选服务选择 Trino，显示 Trino 元数据配置，支持 DEFALUT、EDAP元数据。

注意：如您选择DEFAUT，需登录集群手动更改 Trino Catalog 配置，参考官网文档。

集群管理

集群指标

集群仪表盘

表一 集群仪表盘指标说明

指标英文名称（metric name)	指标中文名称	单位	维度
cluster_YARNResource_precent	YARN计算资源使用率	%	ClusterId
cluster_YARNVCoreResource	YARN计算资源(VCore)	个	ClusterId
cluster_YARNMemoryResource	YARN计算资源(内存)	GB	ClusterId
cluster_HDFSResourceCapacity_precent	HDFS存储资源使用率	%	ClusterId
cluster_cpu_used_percent	集群平均CPU利用率	%	ClusterId
cluster_disk_used_percent	集群平均磁盘利用率	%	ClusterId
cluster_disk_max_partition_used_percent	集群最大磁盘利用率	%	ClusterId
cluster_net_in_bitps	集群总网络入速率	KB	ClusterId
cluster_net_out_bitps	集群总网络出速率	KB	ClusterId
cluster_HDFSResourceCapacity	HDFS存储资源	%	ClusterId
cluster_mem_used_percent	集群平均内存利用率	%	ClusterId
cluster_disk_total_size	集群磁盘总容量	GB	ClusterId
cluster_disk_total_free	集群磁盘总空闲容量	GB	ClusterId
cluster_disk_total_used	集群磁盘总使用容量	GB	ClusterId

主机监控仪表盘

表二 主机监控仪表盘指标说明

指标英文名称（metric name)	指标中文名称	单位	维度
cpu_user	用户CPU利用率	%	InstanceId
cpu_sys	系统CPU利用率	%	InstanceId
cpu_idle	CPU空闲率	%	InstanceId
cpu_wait_io	等待IOCPU时间比率	%	InstanceId
mem_total	内存总量	GB	InstanceId
mem_used	内存使用量	GB	InstanceId
mem_free	内存空闲量	GB	InstanceId
mem_cached	文件系统内存cache值	GB	InstanceId
mem_buffers	块设备读写内存缓冲量	GB	InstanceId
mem_used_percent	内存利用率	%	InstanceId
swap_total	交换分区总量	GB	InstanceId
swap_free	交换分区空闲量	GB	InstanceId
swap_used	交换分区使用量	GB	InstanceId
swap_used_percent	交换分区使用率	%	InstanceId
disk_max_partition_used_percent	最大磁盘分区利用率	%	InstanceId
disk_total_size	磁盘总空间量	GB	InstanceId
disk_total_free	磁盘总空闲量	GB	InstanceId
disk_total_used	磁盘总使用量	GB	InstanceId
disk_total_used_percent	磁盘总使用率	%	InstanceId
disk_total_write_kb	磁盘总写速率	KB/s	InstanceId
disk_total_read_kb	磁盘总读速率	KB/s	InstanceId
disk_size	单块磁盘总容量	GB	InstanceId,DiskName
disk_free	单块磁盘空闲量	GB	InstanceId,DiskName
disk_used	单块磁盘使用量	GB	InstanceId,DiskName
disk_used_percent	单块磁盘使用率	%	InstanceId,DiskName
disk_write_kb	单块盘写速率	KB/s	InstanceId,DiskName
disk_read_kb	单块盘读速率	KB/s	InstanceId,DiskName
disk_io_util	单块盘io使用率	%	InstanceId,DiskName
disk_max_partition_io_util	最大磁盘io使用率	%	InstanceId
fd_limitation	整机的fd上限	个	InstanceId
fd_used	整机已使用的fd个数	个	InstanceId
fd_used_percent	整机的fd使用率	%	InstanceId
loadavg5	机器负载	个	InstanceId
net_total_in_bitps	整机网卡总接受速率	KB/s	InstanceId
net_total_out_bitps	整机网卡总发送速率	KB/s	InstanceId
net_tcp_curr_estab	已建立的TCP连接数	个	InstanceId
net_total_sockets_used	socket连接句柄总数	个	InstanceId
net_tcp_close_wait	CLOSE_WAIT状态连接数	个	InstanceId
host_connect_status	主机连接状态	--	InstanceId
bmr_agent_connect_status	bmr-agent连接状态	--	InstanceId

服务监控

HDFS

表三 HDFS服务指标说明

指标英文名称（metric name)	指标中文名称	单位	维度
dfs_FSNamesystem_BlockCapacity	block的总容量	个	ServiceId
dfs_FSNamesystem_BlocksTotal	block的当前容量	个	ServiceId
dfs_FSNamesystem_CapacityRemainingGB	HDFS文件系统剩余的容量	GB	ServiceId
dfs_FSNamesystem_CapacityTotalGB	HDFS文件系统总体容量	GB	ServiceId
dfs_FSNamesystem_CapacityUsedGB	HDFS文件系统已使用的容量	GB	ServiceId
dfs_FSNamesystem_CorruptBlocks	已损坏的block数量	个	ServiceId
dfs_FSNamesystem_ExcessBlocks	多余的block	个	ServiceId
dfs_FSNamesystem_ExpiredHeartbeats	超时的心跳	个	ServiceId
dfs_FSNamesystem_FilesTotal	文件总数	个	ServiceId
dfs_FSNamesystem_LastCheckpointTime	最近一次做checkpoint的时间	datatime	ServiceId
dfs_FSNamesystem_LastWrittenTransactionId	最近一次写入的transactionid	个	ServiceId
dfs_FSNamesystem_MillisSinceLastLoadedEdits	距离上一次加载edit的时间	ms	ServiceId
dfs_FSNamesystem_MissingBlocks	丢失的block数量	个	ServiceId
dfs_FSNamesystem_UnderReplicatedBlocks	副本个数不够的block	个	ServiceId
dfs_FSNamesystem_PendingDataNodeMessageCount	datanode的请求被queue在standby namenode的个数	个	ServiceId
dfs_FSNamesystem_PendingDeletionBlocks	未被验证的block个数	个	ServiceId
dfs_FSNamesystem_PendingReplicationBlocks	等待被备份的block个数	个	ServiceId
dfs_FSNamesystem_PostponedMisreplicatedBlocks	被推迟处理的错误备份的block个数	个	ServiceId
dfs_FSNamesystem_ScheduledReplicationBlocks	排定要备份的block个数	个	ServiceId
dfs_FSNamesystem_TotalLoad	namenode的Xceiver个数	个	ServiceId
dfs_FSNamesystem_TransactionsSinceLastLogRoll	从上次roll editlog起到现在新的transcation的个数	个	ServiceId
dfs_FSNamesystem_CapacityUsed_percent	HDFS容量使用率	%	ServiceId
dfs_FSNamesystem_NumLiveDataNodes	DataNode正常节点数	个	ServiceId
dfs_FSNamesystem_NumDeadDataNodes	DataNode异常节点数	个	ServiceId
dfs_FSNamesystem_VolumeFailuresTotal	DataNode坏卷数	个	ServiceId
dfs_namenode_Safemode	安全模式	个	ServiceId

HDFS NameNode

表四 HDFS组件指标说明

指标英文名称（metric name)	指标中文名称	单位	维度
dfs_namenode_MemHeapCommitted	heap已提交的内存	MB	ComponentId
dfs_namenode_MemHeapMaxM	heap总内存	MB	ComponentId
dfs_namenode_MemHeapUsedM	heap使用的内存	MB	ComponentId
dfs_namenode_MemMaxM	最大内存	MB	ComponentId
dfs_namenode_MemNonHeapCommittedM	非堆内存提交	MB	ComponentId
dfs_namenode_MemNonHeapMaxM	最大非堆内存	MB	ComponentId
dfs_namenode_MemNonHeapUsedM	非堆内存使用	MB	ComponentId
dfs_namenode_SafeModeTime	safemode时间	ms	ComponentId
dfs_namenode_AddBlockOps	写入block次数	次	ComponentId
dfs_namenode_BlockReportAvgTime	block report的平均时间次数	ms	ComponentId
dfs_namenode_BlockReportNumOps	block report的次数	次	ComponentId
dfs_namenode_CreateFileOps	创建文件次数	次	ComponentId
dfs_namenode_DeleteFileOps	删除文件次数	次	ComponentId
dfs_namenode_FileInfoOps	查看文件info次数	次	ComponentId
dfs_namenode_FilesCreated	已创建的文件个数	个	ComponentId
dfs_namenode_FilesDeleted	已删除的文件个数	个	ComponentId
dfs_namenode_FilesInGetListingOps	getlist操作次数	次	ComponentId
dfs_namenode_FilesRenamed	重命名文件个数	个	ComponentId
dfs_namenode_FsImageLoadTime	fsimage加载时间	ms	ComponentId
dfs_namenode_GetAdditionalDatanodeOps	GetAdditionalDatanode操作次数	次	ComponentId
dfs_namenode_GetBlockLocations	获取block位置操作次数	次	ComponentId
dfs_namenode_GetListingOps	getListing操作次数	次	ComponentId
dfs_namenode_SyncsAvgTime	将操作同步为editlog的平均时间	ms	ComponentId
dfs_namenode_SyncsNumOps	将操作同步为editlog的次数	次	ComponentId
dfs_namenode_TransactionsAvgTime	transcation的平均时间	ms	ComponentId
dfs_namenode_TransactionsBatchedInSync	transcation在flush时发现已经被sync的情况的次数	次	ComponentId
dfs_namenode_TransactionsNumOps	transcation的个数	个	ComponentId

HDFS DataNode

表五 HDFS DataNode组件指标说明

指标英文名称（metric name)	指标中文名称	单位	维度
dfs_datanode_MemHeapCommittedM	heap已提交的内存	MB	ComponentId
dfs_datanode_MemHeapMaxM	heap总内存	MB	ComponentId
dfs_datanode_MemHeapUsedM	heap使用的内存	MB	ComponentId
dfs_datanode_MemMaxM	最大内存	MB	ComponentId
dfs_datanode_MemNonHeapCommittedM	非堆内存提交	MB	ComponentId
dfs_datanode_MemNonHeapMaxM	最大非堆内存	MB	ComponentId
dfs_datanode_MemNonHeapUsedM	非堆内存使用	MB	ComponentId
dfs_datanode_BlockReportsAvgTime	向namenode汇报block的平均时间	ms	ComponentId
dfs_datanode_BlockReportsNumOps	向namenode汇报block的次数	次	ComponentId
dfs_datanode_BlocksRead	从硬盘读块的次数	次	ComponentId
dfs_datanode_BlocksRemoved	删除块的个数	次	ComponentId
dfs_datanode_BlocksReplicated	备份块操作的个数	个	ComponentId
dfs_datanode_BlocksVerified	验证块的次数	次	ComponentId
dfs_datanode_BlocksWritten	写入块的个数	个	ComponentId
dfs_datanode_BytesRead	读出总字节	bytes	ComponentId
dfs_datanode_BytesWritten	写入总字节	bytes	ComponentId
dfs_datanode_CopyBlockOpAvgTime	复制块的平均时间	ms	ComponentId
dfs_datanode_CopyBlockOpNumOps	复制块的次数	次	ComponentId
dfs_datanode_HeartbeatsAvgTime	向namenode汇报的平均时间	ms	ComponentId
dfs_datanode_HeartbeatsNumOps	向namenode汇报的次数	次	ComponentId
dfs_datanode_ReadBlockOpAvgTime	读数据块的平均时间	ms	ComponentId
dfs_datanode_ReadBlockOpNumOps	读数据块的次数	次	ComponentId
dfs_datanode_ReadsFromLocalClient	本地读取的次数	次	ComponentId
dfs_datanode_ReadsFromRemoteClient	远程读取的次数	次	ComponentId
dfs_datanode_WriteBlockOpAvgTime	写数据块的平均时间	ms	ComponentId
dfs_datanode_WriteBlockOpNumOps	写数据块的次数	次	ComponentId
dfs_datanode_WritesFromLocalClient	写本地的次数	次	ComponentId
dfs_datanode_WritesFromRemoteClient	写远程的次数	次	ComponentId
dfs_datanode_PacketAckRoundTripTimeNanosAvgTime	包确认平均时间	ms	ComponentId
dfs_datanode_PacketAckRoundTripTimeNanosNumOps	包确认次数	次	ComponentId
dfs_datanode_FlushNanosAvgTime	文件系统flush平均时间	ms	ComponentId
dfs_datanode_FlushNanosNumOps	文件系统flush平均时间	ms	ComponentId
dfs_datanode_ReplaceBlockOpAvgTime	块替换平均时间	ms	ComponentId
dfs_datanode_ReplaceBlockOpNumOps	块替换次数	次	ComponentId
dfs_datanode_SendDataPacketBlockedOnNetworkNanosAvgTime	网络上发送块平均时间	ms	ComponentId
dfs_datanode_SendDataPacketBlockedOnNetworkNanosNumOps	网络上发生块次数	次	ComponentId
dfs_datanode_SendDataPacketTransferNanosAvgTime	网络上发送包平均时间	ms	ComponentId
dfs_datanode_SendDataPacketTransferNanosNumOps	网络上发送包个数	个	ComponentId
dfs_datanode_FsStat_Capacity	DataNode容量	GB	ComponentId
dfs_datanode_FsStat_DfsUsed	DataNode使用量	GB	ComponentId
dfs_datanode_FsStat_NumFailedVolumes	坏卷数量	GB	ComponentId

HDFS JOURNALNODE

表六 HDFS JOURNALNODE组件指标说明

指标英文名称（metric name)	指标中文名称	单位	维度
status	进程运行状态	state	ComponentId
proc cpu usage	进程cpu利用率	%	ComponentId
proc mem usage	进程内存利用率	%	ComponentId
jvm gc o	jvm gc o	%	ComponentId
jvm_gc E	jvm_gc E	%	ComponentId
jvm_gc M	jvm_gc M	%	ComponentId
jvm gc ccs	jvm gc ccs	%	ComponentId
jvm gc yGcT	jvm gc yGcT	%	ComponentId
jvm gc FGCT	jvm gc FGCT	%	ComponentId
jvm_gc_GCT	jvm_gc_GCT	%	ComponentId
jvm_gc_YGC	jvm_gc_YGC	%	ComponentId
jvm_gc_FGC	jvm_gc_FGC	%	ComponentId

YARN

表七 YARN服务指标说明

指标英文名称（metric name)	指标中文名称	单位	维度
yarn_ClusterMetrics_NumActiveNMs	活的nodemanager个数	个	ServiceId
yarn_ClusterMetrics_NumLostNMs	丢失的nodemanager个数	个	ServiceId
yarn_ClusterMetrics_NumUnhealthyNMs	不健康的nodemanager个数	个	ServiceId
yarn_QueueMetrics_TotalMB	总内存	GB	ServiceId
yarn_QueueMetrics_TotalVCores	总vcores	个	ServiceId
yarn_QueueMetrics_ActiveApplications	活跃的task的个数	个	ServiceId
yarn_QueueMetrics_ActiveUsers	活跃的用户个数	个	ServiceId
yarn_QueueMetrics_AggregateContainersAllocated	总共分配的container个数	个	ServiceId
yarn_QueueMetrics_AggregateContainersReleased	总共释放的container个数	个	ServiceId
yarn_QueueMetrics_AllocatedContainers	已经分配的container个数	个	ServiceId
yarn_QueueMetrics_AllocatedMB	已经分配的内存	GB	ServiceId
yarn_QueueMetrics_AllocatedVCores	已分配的vcore	个	ServiceId
yarn_QueueMetrics_AppsCompleted	已完成的task数	个	ServiceId
yarn_QueueMetrics_AppsPending	挂起的task数	个	ServiceId
yarn_QueueMetrics_AppsRunning	运行的task数	个	ServiceId
yarn_QueueMetrics_AppsSubmitted	已经提交的task数	个	ServiceId
yarn_QueueMetrics_AvailableMB	可用的内存	GB	ServiceId
yarn_QueueMetrics_AvailableVCores	可用的vcore	个	ServiceId
yarn_QueueMetrics_PendingContainers	挂起的container数	个	ServiceId
yarn_QueueMetrics_PendingMB	挂起的内存	GB	ServiceId
yarn_QueueMetrics_PendingVCores	挂起的vcore	个	ServiceId
yarn_QueueMetrics_running_0	运行时间在0-60分钟内的task个数	个	ServiceId
yarn_QueueMetrics_running_1440	运行时间在1440分钟以上的task个数	个	ServiceId
yarn_QueueMetrics_running_300	运行时间在300-1440分钟内的task个数	个	ServiceId
yarn_QueueMetrics_running_60	运行时间在60-300分钟内的task个数	个	ServiceId
yarn_QueueMetrics_AllocatedMem_precent	分配内存占比	%	ServiceId
yarn_QueueMetrics_AllocatedVCore_precent	分配VCore占比	%	ServiceId

YARN TimeLineServer

表八 yarn TimeLineServer组件指标说明

指标英文名称（metric name）	指标中文名称	单位	维度
yarn_timeline_GetEntitiesOps	TimelineServer获取批量entities操作数	次	ComponentId
yarn_timeline_GetEntitiesTimeAvgTime	TimelineServer获取批量entities平均时间	ms	ComponentId
yarn_timeline_GetEntityOps	TimelineServer获取entity操作数	次	ComponentId
yarn_timeline_GetEntityTimeAvgTime	TimelineServer获取entity平均时间	ms	ComponentId
yarn_timeline_GetEventsOps	TimelineServer获取批量events操作数	次	ComponentId
yarn_timeline_GetEventsTimeAvgTime	TimelineServer获取批量evnets平均时间	ms	ComponentId
yarn_timeline_PostEntitiesOps	TimelineServer更新批量entities操作数	次	ComponentId
yarn_timeline_PostEntitiesTimeAvgTime	TimelineServer更新批量entities的平均时间	ms	ComponentId
yarn_timeline_PutDomainOps	TimelineServer更新Domain操作数	次	ComponentId
yarn_timeline_PutDomainTimeAvgTime	TimelineServer更新Domain平均时间	ms	ComponentId
yarn_timeline_GetDomainOps	TimelineServer获取Domain操作数	次	ComponentId
yarn_timeline_GetDomainTimeAvgTime	TimelineServer获取Domain平均时间	ms	ComponentId
yarn_timeline_GetDomainsOps	TimelineServer批量获取Domains操作数	次	ComponentId
yarn_timeline_GetDomainsTimeAvgTime	TimelineServer批量获取Domains平均时间	ms	ComponentId

YARN ResourceManager

表九 YARN ResourceManager指标说明

指标英文名称（metric name）	指标中文名称	单位	维度
status	进程运行状态	state	ComponentId
proc cpu usage	进程cpu利用率	%	ComponentId
proc mem usage	进程内存利用率	%	ComponentId
jvm gc o	jvm gc o	%	ComponentId
jvm_gc E	jvm_gc E	%	ComponentId
jvm_gc M	jvm_gc M	%	ComponentId
jvm gc ccs	jvm gc ccs	%	ComponentId
jvm gc yGcT	jvm gc yGcT	%	ComponentId
jvm gc FGCT	jvm gc FGCT	%	ComponentId
jvm_gc_GCT	jvm_gc_GCT	%	ComponentId
jvm_gc_YGC	jvm_gc_YGC	%	ComponentId
jvm_gc_FGC	jvm_gc_FGC	%	ComponentId

YARN HISTORY_SERVER

表十 YARN HISTORY_SERVER指标说明

指标英文名称（metric name）	指标中文名称	单位	维度
status	进程运行状态	state	ComponentId
proc cpu usage	进程cpu利用率	%	ComponentId
proc mem usage	进程内存利用率	%	ComponentId
jvm gc o	jvm gc o	%	ComponentId
jvm_gc E	jvm_gc E	%	ComponentId
jvm_gc M	jvm_gc M	%	ComponentId
jvm gc ccs	jvm gc ccs	%	ComponentId
jvm gc yGcT	jvm gc yGcT	%	ComponentId
jvm gc FGCT	jvm gc FGCT	%	ComponentId
jvm_gc_GCT	jvm_gc_GCT	%	ComponentId
jvm_gc_YGC	jvm_gc_YGC	%	ComponentId
jvm_gc_FGC	jvm_gc_FGC	%	ComponentId

YARN NodeManager

表十一 YARN NodeManager指标说明

指标英文名称（metric name）	指标中文名称	单位	维度
status	进程运行状态	state	ComponentId
proc cpu usage	进程cpu利用率	%	ComponentId
proc mem usage	进程内存利用率	%	ComponentId
jvm gc o	jvm gc o	%	ComponentId
jvm_gc E	jvm_gc E	%	ComponentId
jvm_gc M	jvm_gc M	%	ComponentId
jvm gc ccs	jvm gc ccs	%	ComponentId
jvm gc yGcT	jvm gc yGcT	%	ComponentId
jvm gc FGCT	jvm gc FGCT	%	ComponentId
jvm_gc_GCT	jvm_gc_GCT	%	ComponentId
jvm_gc_YGC	jvm_gc_YGC	%	ComponentId
jvm_gc_FGC	jvm_gc_FGC	%	ComponentId

HIVE

HiveServer2

表十二 HIVE组件指标说明1

指标英文名称（metric name）	指标中文名称	单位	维度
hive_hs2_active_sessions	当前活跃Session数	个	ComponentId
hive_hs2_open_sessions	当前打开的Session数	个	ComponentId
hive_hs2_open_connections	当前打开的连接数	个	ComponentId
hive_hs2_cumulative_connection_count	累计连接数	个	ComponentId
hive_hs2_active_calls_api_runTasks	当前Runtask请求数	个	ComponentId
hive_hs2_compiling_queries	执行编译的平均时间	ms	ComponentId
hive_hs2_executing_queries	执行查询的平均时间	ms	ComponentId
hive_hs2_submitted_queries	提交查询的平均时间	ms	ComponentId
hive_hs2_succeeded_queries	服务启动后成功的查询数	个	ComponentId
hive_hs2_sql_operation_active_user	当前活跃用户数	个	ComponentId
hive_hs2_completed_sql_operation_FINISHED	已结束的SQL总数	个	ComponentId
hive_hs2_sql_operation_PENDING	SQL任务处于PEEDING状态平均时间	ms	ComponentId
hive_hs2_sql_operation_RUNNING	SQL任务处于RUNNING状态平均时间	ms	ComponentId
status	进程运行状态	state	ComponentId
proc cpu usage	进程cpu利用率	%	ComponentId
proc mem usage	进程内存利用率	%	ComponentId
jvm gc o	jvm gc o	%	ComponentId
jvm_gc E	jvm_gc E	%	ComponentId
jvm_gc M	jvm_gc M	%	ComponentId
jvm gc ccs	jvm gc ccs	%	ComponentId
jvm gc yGcT	jvm gc yGcT	%	ComponentId
jvm gc FGCT	jvm gc FGCT	%	ComponentId
jvm_gc_GCT	jvm_gc_GCT	%	ComponentId
jvm_gc_YGC	jvm_gc_YGC	%	ComponentId
jvm_gc_FGC	jvm_gc_FGC	%	ComponentId

HiveMetastore

表十三 HIVE组件指标说明2

指标英文名称（metric name)	指标中文名称	单位	维度
hive_metastore_active_calls_drop_table	当前活跃DropTable请求数	次	ComponentId
hive_metastore_api_alter_table	AlterTable请求平均时间	ms	ComponentId
hive_metastore_api_alter_table_with_environment_context	AlterTableWithEnvContext请求平均时间	ms	ComponentId
hive_metastore_api_create_table	CreateTable请求平均时间	ms	ComponentId
hive_metastore_api_create_table_with_environment_context	CreateTableWithEnvContext请求平均时间	ms	ComponentId
hive_metastore_api_drop_table	DropTable请求平均时间	ms	ComponentId
hive_metastore_api_drop_table_with_environment_context	DropTableWithEnvContext请求平均时间	ms	ComponentId
hive_metastore_api_get_table	GetTable请求平均时间	ms	ComponentId
hive_metastore_api_get_table_req	GetTableReq请求平均时间	ms	ComponentId
hive_metastore_api_get_table_objects_by_name_req	GetTableObjectsByName请求平均时间	ms	ComponentId
hive_metastore_api_get_tables	GetTables请求平均时间	ms	ComponentId
hive_metastore_api_get_tables_by_type	GetTablesByType请求平均时间	ms	ComponentId
hive_metastore_api_get_multi_table	GetMultiTable请求平均时间	ms	ComponentId
hive_metastore_api_get_table_statistics_req	GetTableStatistics请求平均时间	ms	ComponentId
hive_metastore_api_get_all_databases	GetAllDatabases请求平均时间	ms	ComponentId
hive_metastore_api_get_database	GetDatabase请求平均时间	ms	ComponentId
hive_metastore_api_get_databases	GetDatabases请求平均时间	ms	ComponentId
hive_metastore_api_get_all_functions	GetAllFunctions请求平均时间	ms	ComponentId

ZOOKEEPER

表十四 ZOOKEEPER组件指标说明

指标英文名称（metric name)	指标中文名称	单位	维度
zk_avg_latency	平均响应延迟	ms	ComponentId
zk_max_latency	最大响应延迟	ms	ComponentId
zk_min_latency	最小响应延迟	ms	ComponentId
zk_packets_received	收包数	个	ComponentId
zk_packets_sent	发包数	个	ComponentId
zk_num_alive_connections	活跃连接数	个	ComponentId
zk_outstanding_requests	堆积请求数	个	ComponentId
zk_server_state	主从状态	个	ComponentId
zk_znode_count	znode数	个	ComponentId
zk_watch_count	watch数	个	ComponentId
zk_ephemerals_count	临时节点数	个	ComponentId
zk_approximate_data_size	近似数据总和大小	bytes	ComponentId
zk_open_file_descriptor_count	打开文件描述符数	个	ComponentId
zk_max_file_descriptor_count	最大文件描述符数	个	ComponentId
status	进程运行状态	state	ComponentId
proc cpu usage	进程cpu利用率	%	ComponentId
proc mem usage	进程内存利用率	%	ComponentId
jvm gc o	jvm gc o	%	ComponentId
jvm_gc E	jvm_gc E	%	ComponentId
jvm_gc M	jvm_gc M	%	ComponentId
jvm gc ccs	jvm gc ccs	%	ComponentId
jvm gc yGcT	jvm gc yGcT	%	ComponentId
jvm gc FGCT	jvm gc FGCT	%	ComponentId
jvm_gc_GCT	jvm_gc_GCT	%	ComponentId
jvm_gc_YGC	jvm_gc_YGC	%	ComponentId
jvm_gc_FGC	jvm_gc_FGC	%	ComponentId

HBASE

表十五 HBASE服务指标说明

指标英文名称（metric name)	指标中文名称	单位	维度
hbase_ritCount	处于RIT的Region个数	个	ServiceId
hbase_ritCountOverThreshold	处于超时的RIT的Region个数	个	ServiceId
hbase_ritOldestAge	RIT的最长时间	ms	ServiceId
hbase_averageLoad	平均负载	个	ServiceId
hbase_numRegionServers	活动的RS数量	个	ServiceId
hbase_numDeadRegionServers	停止的RS数量	个	ServiceId
hbase_clusterRequests	集群总请求数量	个	ServiceId
hbase_mergePlanCount	Merge计划数	个	ServiceId
hbase_splitPlanCount	Split计划数	个	ServiceId
hbase_receivedBytes	接受字节数	Bytes	ServiceId
hbase_sentBytes	发送字节数	Bytes	ServiceId
hbase_queueSize	排队队列大小	个	ServiceId
hbase_numCallsInGeneralQueue	普通队列调用数	次	ServiceId
hbase_numCallsInReplicationQueue	副本队列调用数	次	ServiceId
hbase_numCallsInPriorityQueue	优先队列调用数	次	ServiceId
hbase_numOpenConnections	保持的链接数的大小	个	ServiceId
hbase_numActiveHandler	活跃的handler	个	ServiceId
hbase_numGeneralCallsDropped	丢失的普通请求数	次	ServiceId
hbase_numLifoModeSwitches	栈模式切换数	次	ServiceId
hbase_authenticationSuccesses	认证成功数	次	ServiceId
hbase_authenticationFailures	认证失败次数	次	ServiceId
hbase_authenticationFallbacks	认证退却次数	次	ServiceId
hbase_authorizationSuccesses	授权成功次数	次	ServiceId
hbase_authorizationFailures	授权失败数	次	ServiceId
hbase_exceptions_RegionMovedException	Region状态迁移错误数	次	ServiceId
hbase_exceptions_multiResponseTooLarge	接收到多个相应超出限定阈值	次	ServiceId
hbase_exceptions_RegionTooBusyException	RegionServer任务过多导致错误的数量	次	ServiceId
hbase_exceptions_FailedSanityCheckException	FailedSanityCheckException	次	ServiceId
hbase_exceptions_UnknownScannerException	未知扫描错误	次	ServiceId
hbase_exceptions_OutOfOrderScannerNextException	乱序扫描错误	次	ServiceId
hbase_exceptions_NotServingRegionException	NotServingRegionException	次	ServiceId
hbase_exceptions_callQueueTooBig	等待队列满错误	次	ServiceId
hbase_exceptions_ScannerResetException	扫描器重置错误	次	ServiceId
hbase_exceptions	总错误数	次	ServiceId
hbase_ProcessCallTime_num_ops	总操作数	次	ServiceId
hbase_ProcessCallTime_min	处理时间最小值	ms	ServiceId
hbase_ProcessCallTime_max	处理时间最大值	ms	ServiceId
hbase_ProcessCallTime_mean	处理时间平均值	ms	ServiceId
hbase_QueueCallTime_num_opsHBASE_REGIONSERVER	队列调用次数	次	ServiceId
hbase_QueueCallTime_min	调用最短时间	ms	ServiceId
hbase_QueueCallTime_max	调用最长时间	ms	ServiceId
hbase_QueueCallTime_mean	调用平均时间	ms	ServiceId

HBASE_REGIONSERVER

表十六 HBASE组件指标说明1

指标英文名称（metric name)	指标中文名称	单位	维度

hbase_rs_averageRegionSize	Region平均大小	Bytes	ComponentId
hbase_rs_regionCount	Region个数	个	ComponentId
hbase_rs_percentFilesLocalSecondaryRegions	Region副本本地化	%	ComponentId
hbase_rs_hlogFileCount	WAL文件数量	个	ComponentId
hbase_rs_hlogFileSize	WAL文件大小	Bytes	ComponentId
hbase_rs_memStoreSize	Memstore大小	MB	ComponentId
hbase_rs_storeCount	Store个数	个	ComponentId
hbase_rs_storeFileCount	Storefile个数	个	ComponentId
hbase_rs_storeFileSize	Storefile 大小	MB	ComponentId
hbase_rs_storeFileIndexSize	storeFileIndexSize	Bytes	ComponentId
hbase_rs_staticIndexSize	staticIndexSize	Bytes	ComponentId
hbase_rs_staticBloomSize	staticBloomSize	Bytes	ComponentId
hbase_rs_flushedCellsSize	flush到磁盘的大小	Bytes	ComponentId
hbase_rs_Append_mean	Append_mean	ms	ComponentId
hbase_rs_Replay_mean	Append_mean	ms	ComponentId
hbase_rs_Get_mean	Append_mean	ms	ComponentId
hbase_rs_updatesBlockedTime	updatesBlockedTime	ms	ComponentId
hbase_rs_FlushTime_num_ops	RS写磁盘次数	次	ComponentId
hbase_rs_splitQueueLength	split操作队列请求数	个	ComponentId
hbase_rs_compactionQueueLength	compaction操作队列请求数	个	ComponentId
hbase_rs_totalRequestCount	总请求数	次	ComponentId
hbase_rs_readRequestCount	读请求数	次	ComponentId
hbase_rs_writeRequestCount	写请求数	次	ComponentId
hbase_rs_compactedCellsCount	合并cell个数	个	ComponentId
hbase_rs_majorCompactedCellsCount	大合并cell个数	个	ComponentId
hbase_rs_splitRequestCount	region分裂请求次数	次	ComponentId
hbase_rs_splitSuccessCount	region分裂成功次数	次	ComponentId
hbase_rs_slowGetCount	请求完成时间超过1000ms的次数	次	ComponentId
hbase_rs_authenticationFailures	RPC认证失败次数	次	ComponentId
hbase_rs_authenticationSuccesses	RPC认证成功次数	次	ComponentId
hbase_rs_numOpenConnections	RPC打开的连接数	个	ComponentId
hbase_rs_exceptions_FailedSanityCheckException	FailedSanityCheckException	次	ComponentId
hbase_rs_exceptions_NotServingRegionException	NotServingRegionException	次	ComponentId
hbase_rs_exceptions_OutOfOrderScannerNextException	OutOfOrderScannerNextException	次	ComponentId
hbase_rs_exceptions_RegionMovedException	RegionMovedException	次	ComponentId
hbase_rs_exceptions_RegionTooBusyException	RegionTooBusyException	次	ComponentId
hbase_rs_exceptions_UnknownScannerException	UnknownScannerException	次	ComponentId
hbase_rs_exceptions	Exceptions	次	ComponentId
hbase_rs_numActiveHandler	RPC句柄数	个	ComponentId
hbase_rs_numCallsInPriorityQueue	numCallsInPriorityQueue	个	ComponentId
hbase_rs_numCallsInReplicationQueue	numCallsInReplicationQueue	个	ComponentId
hbase_rs_numCallsInGeneralQueue	numCallsInGeneralQueue	个	ComponentId
hbase_rs_receivedBytes	接受字节数	Bytes	ComponentId
hbase_rs_sentBytes	发送字节数	Bytes	ComponentId
hbase_rs_queueSize	排队队列大小	个	ComponentId
hbase_rs_blockCacheSize	block缓存大小	Bytes	ComponentId
hbase_rs_blockCacheFreeSize	block缓存剩余大小	Bytes	ComponentId
hbase_rs_blockCacheCount	block缓存命中次数	Bytes	ComponentId
hbase_rs_blockCacheCountHitPercent	block缓存命中率	%	ComponentId

hbase_rs_blockCacheExpressHitPercent	block缓存打开命中率	%	ComponentId
status	进程运行状态	state	ComponentId
proc cpu usage	进程cpu利用率	%	ComponentId
proc mem usage	进程内存利用率	%	ComponentId
jvm gc o	jvm gc o	%	ComponentId
jvm_gc E	jvm_gc E	%	ComponentId
jvm_gc M	jvm_gc M	%	ComponentId
jvm gc ccs	jvm gc ccs	%	ComponentId
jvm gc yGcT	jvm gc yGcT	%	ComponentId
jvm gc FGCT	jvm gc FGCT	%	ComponentId
jvm_gc_GCT	jvm_gc_GCT	%	ComponentId
jvm_gc_YGC	jvm_gc_YGC	%	ComponentId
jvm_gc_FGC	jvm_gc_FGC	%	ComponentId

HBASE_TABLE

表十七 HBASE组件指标说明2

指标英文名称（metric name)	指标中文名称	单位	维度
hbase_tb_tableSize	hbase_tableSize	Bytes	TopicId
hbase_tb_storeFileSize	hbase_storeFileSize	Bytes	TopicId
hbase_tb_readRequestCount	hbase_readRequestCount	个	TopicId
hbase_tb_writeRequestCount	hbase_writeRequestCount	个	TopicId
hbase_tb_totalRequestCount	hbase_totalRequestCount	个	TopicId
hbase_tb_memstoreSize	hbase_memstoreSize	Bytes	TopicId

CLICKHOUSE

CLICKHOUSE_SERVER

表十八 CLICKHOUSE组件监控说明

ck_server_event_InsertQuery	ck_event_InsertQuery	次	ComponentId
ck_server_event_InsertedRows	ck_event_InsertedRows	条	ComponentId
ck_server_event_DelayedInserts	ck_event_DelayedInserts	条	ComponentId
ck_server_event_RejectedInserts	ck_event_RejectedInserts	条	ComponentId
ck_server_event_MergedRows	ck_event_MergedRows	行	ComponentId
ck_server_metrics_BackgroundPoolTask	ck_metrics_BackgroundPoolTask	个	ComponentId
ck_server_metrics_Merge	ck_metrics_Merge	次	ComponentId
ck_server_metrics_MemoryTrackingForMerges	ck_metrics_MemoryTrackingForMerges	bytes	ComponentId
ck_server_metrics_PartMutation	ck_metrics_PartMutation	个	ComponentId
ck_server_event_Query	ck_event_Query	次	ComponentId
ck_server_event_SelectQuery	ck_event_SelectQuery	次	ComponentId
ck_server_event_FailedQuery	ck_event_FailedQuery	次	ComponentId
ck_server_event_SlowRead	ck_event_SlowRead	个	ComponentI

ck_server_event_SlowRead	ck_event_SlowRead	个	d
ck_server_metrics_MemoryTracking	ck_metrics_MemoryTracking	bytes	ComponentId
ck_server_event_MarkCacheHits	ck_event_MarkCacheHits	个	ComponentId
ck_server_event_MarkCacheMisses	ck_event_MarkCacheMisses	个	ComponentId
ck_server_metrics_ReadonlyReplica	ck_metrics_ReadonlyReplica	个	ComponentId
ck_server_metrics_ReplicatedFetch	ck_metrics_ReplicatedFetch	个	ComponentId
ck_server_metrics_ReplicatedSend	ck_metrics_ReplicatedSend	个	ComponentId
ck_server_event_ZooKeeperTransactions	ck_event_ZooKeeperTransactions	个	ComponentId
ck_server_metrics_ZooKeeperSession	ck_metrics_ZooKeeperSession	个	ComponentId
ck_server_metrics_ZooKeeperWatch	ck_metrics_ZooKeeperWatch	个	ComponentId
ck_server_metrics_Query	ck_metrics_Query	次	ComponentId
ck_server_metrics_ReplicatedChecks	ck_metrics_ReplicatedChecks	个	ComponentId
ck_server_metrics_BackgroundMovePoolTask	ck_metrics_BackgroundMovePoolTask	个	ComponentId
ck_server_metrics_BackgroundSchedulePoolTask	ck_metrics_BackgroundSchedulePoolTask	个	ComponentId
ck_server_metrics_BackgroundBufferFlushSchedulePoolTask	ck_metrics_BackgroundBufferFlushSchedulePoolTask	个	ComponentId
ck_server_metrics_BackgroundDistributedSchedulePoolTask	ck_metrics_BackgroundDistributedSchedulePoolTask	个	ComponentId
ck_server_metrics_CacheDictionaryUpdateQueueBatches	ck_metrics_CacheDictionaryUpdateQueueBatches	个	ComponentId
ck_server_metrics_CacheDictionaryUpdateQueueKeys	ck_metrics_CacheDictionaryUpdateQueueKeys	个	ComponentId
ck_server_metrics_DiskSpaceReservedForMerge	ck_metrics_DiskSpaceReservedForMerge	bytes	ComponentId
ck_server_metrics_DistributedSend	ck_metrics_DistributedSend	个	ComponentId
ck_server_metrics_QueryPreempted	ck_metrics_QueryPreempted	个	ComponentId
ck_server_metrics_TCPConnection	ck_metrics_TCPConnection	个	ComponentId
ck_server_metrics_MySQLConnection	ck_metrics_MySQLConnection	个	ComponentId
ck_server_metrics_HTTPConnection	ck_metrics_HTTPConnection	个	ComponentId
ck_server_metrics_InterserverConnection	ck_metrics_InterserverConnection	个	ComponentId
ck_server_metrics_PostgreSQLConnection	ck_metrics_PostgreSQLConnection	个	ComponentId
ck_server_metrics_OpenFileForRead	ck_metrics_OpenFileForRead	个	ComponentId
ck_server_metrics_OpenFileForWrite	ck_metrics_OpenFileForWrite	个	ComponentId

ck_server_metrics_Read	ck_metrics_Read	个	ComponentId
ck_server_metrics_Write	ck_metrics_Write	个	ComponentId
ck_server_metrics_SendScalars	ck_metrics_SendScalars	个	ComponentId
ck_server_metrics_SendExternalTables	ck_metrics_SendExternalTables	个	ComponentId
ck_server_metrics_QueryThread	ck_metrics_QueryThread	个	ComponentId
ck_server_metrics_MemoryTrackingInBackgroundProcessingPool	ck_metrics_MemoryTrackingInBackgroundProcessingPool	bytes	ComponentId
ck_server_metrics_MemoryTrackingInBackgroundMoveProcessingPool	ck_metrics_MemoryTrackingInBackgroundMoveProcessingPool	bytes	ComponentId
ck_server_metrics_MemoryTrackingInBackgroundSchedulePool	ck_metrics_MemoryTrackingInBackgroundSchedulePool	bytes	ComponentId
ck_server_metrics_MemoryTrackingInBackgroundBufferFlushSchedulePool	ck_metrics_MemoryTrackingInBackgroundBufferFlushSchedulePool	bytes	ComponentId
ck_server_metrics_MemoryTrackingInBackgroundDistributedSchedulePool	ck_metrics_MemoryTrackingInBackgroundDistributedSchedulePool	bytes	ComponentId
ck_server_metrics_EphemeralNode	ck_metrics_EphemeralNode	个	ComponentId
ck_server_metrics_ZooKeeperRequest	ck_metrics_ZooKeeperRequest	个	ComponentId
ck_server_metrics_DelayedInserts	ck_metrics_DelayedInserts	个	ComponentId
ck_server_metrics_ContextLockWait	ck_metrics_ContextLockWait	个	ComponentId
ck_server_metrics_StorageBufferRows	ck_metrics_StorageBufferRows	个	ComponentId
ck_server_metrics_StorageBufferBytes	ck_metrics_StorageBufferBytes	bytes	ComponentId
ck_server_metrics_DictCacheRequests	ck_metrics_DictCacheRequests	个	ComponentId
ck_server_metrics_Revision	ck_metrics_Revision	个	ComponentId
ck_server_metrics_VersionInteger	ck_metrics_VersionInteger	版本	ComponentId
ck_server_metrics_RWLockWaitingReaders	ck_metrics_RWLockWaitingReaders	个	ComponentId
ck_server_metrics_RWLockWaitingWriters	ck_metrics_RWLockWaitingWriters	个	ComponentId
ck_server_metrics_RWLockActiveReaders	ck_metrics_RWLockActiveReaders	个	ComponentId
ck_server_metrics_RWLockActiveWriters	ck_metrics_RWLockActiveWriters	个	ComponentId
ck_server_metrics_GlobalThread	ck_metrics_GlobalThread	个	ComponentId
ck_server_metrics_GlobalThreadActive	ck_metrics_GlobalThreadActive	个	ComponentId
ck_server_metrics_LocalThread	ck_metrics_LocalThread	个	ComponentId
ck_server_metrics_LocalThreadActive	ck_metrics_LocalThreadActive	个	ComponentId
ck_server_metrics_DistributedFilesToInsert	ck_metrics_DistributedFilesToInsert	个	ComponentI

			d
ck_server_event_FailedSelectQuery	ck_event_FailedSelectQuery	次	ComponentId
ck_server_event_FailedInsertQuery	ck_event_FailedInsertQuery	次	ComponentId
ck_event_FileOpen	ck_event_FileOpen	个	ComponentId
ck_server_event_Seek	ck_event_Seek	次	ComponentId
ck_server_event_ReadBufferFromFileDescriptorRead	ck_event_ReadBufferFromFileDescriptorRead	个	ComponentId
ck_server_event_ReadBufferFromFileDescriptorReadBytes	ck_event_ReadBufferFromFileDescriptorReadBytes	bytes	ComponentId
ck_server_event_WriteBufferFromFileDescriptorWrite	ck_event_WriteBufferFromFileDescriptorWrite	个	ComponentId
ck_server_event_WriteBufferFromFileDescriptorWriteFailed	ck_event_WriteBufferFromFileDescriptorWriteFailed	个	ComponentId
ck_server_event_WriteBufferFromFileDescriptorWriteBytes	ck_event_WriteBufferFromFileDescriptorWriteBytes	bytes	ComponentId
ck_server_event_ReadCompressedBytes	ck_event_ReadCompressedBytes	bytes	ComponentId
ck_server_event_CompressedReadBufferBlocks	ck_event_CompressedReadBufferBlocks	个	ComponentId
ck_server_event_CompressedReadBufferBytes	ck_event_CompressedReadBufferBytes	bytes	ComponentId
ck_server_event_IOBufferAllocs	ck_event_IOBufferAllocs	个	ComponentId
ck_server_event_IOBufferAllocBytes	ck_event_IOBufferAllocBytes	bytes	ComponentId
ck_server_event_ArenaAllocChunks	ck_event_ArenaAllocChunks	个	ComponentId
ck_server_event_ArenaAllocBytes	ck_event_ArenaAllocBytes	bytes	ComponentId
ck_server_event_FunctionExecute	ck_event_FunctionExecute	个	ComponentId
ck_server_event_TableFunctionExecute	ck_event_TableFunctionExecute	个	ComponentId
ck_server_event_CreatedReadBufferOrdinary	ck_event_CreatedReadBufferOrdinary	个	ComponentId
ck_server_event_DiskReadElapsedMicroseconds	ck_event_DiskReadElapsedMicroseconds	μs	ComponentId
ck_server_event_DiskWriteElapsedMicroseconds	ck_event_DiskWriteElapsedMicroseconds	μs	ComponentId
ck_server_event_NetworkReceiveElapsedMicroseconds	ck_event_NetworkReceiveElapsedMicroseconds	μs	ComponentId
ck_server_event_NetworkSendElapsedMicroseconds	ck_event_NetworkSendElapsedMicroseconds	μs	ComponentId
ck_server_event_ReplicatedPartFetches	ck_event_ReplicatedPartFetches	个	ComponentId
ck_server_event_ReplicatedPartMerges	ck_event_ReplicatedPartMerges	个	ComponentId
ck_server_event_InsertedBytes	ck_event_InsertedBytes	bytes	ComponentId
ck_server_event_DelayedInsertsMilliseconds	ck_event_DelayedInsertsMilliseconds	ms	ComponentId

ck_server_event_ZooKeeperInit	ck_event_ZooKeeperInit	个	ComponentId
ck_server_event_ZooKeeperList	ck_event_ZooKeeperList	个	ComponentId
ck_server_event_ZooKeeperCreate	ck_event_ZooKeeperCreate	个	ComponentId
ck_server_event_ZooKeeperWaitMicroseconds	ck_event_ZooKeeperWaitMicroseconds	μs	ComponentId
ck_server_event_ZooKeeperBytesSent	ck_event_ZooKeeperBytesSent	bytes	ComponentId
ck_server_event_ZooKeeperBytesReceived	ck_event_ZooKeeperBytesReceived	bytes	ComponentId
ck_server_event_ReadBackoff	ck_event_ReadBackoff	个	ComponentId
ck_server_event_ReplicaPartialShutdown	ck_event_ReplicaPartialShutdown	个	ComponentId
ck_server_event_SelectedParts	ck_event_SelectedParts	个	ComponentId
ck_server_event_SelectedRanges	ck_event_SelectedRanges	个	ComponentId
ck_server_event_SelectedMarks	ck_event_SelectedMarks	个	ComponentId
ck_server_event_Merge	ck_event_Merge	次	ComponentId
ck_server_event_MergedUncompressedBytes	ck_event_MergedUncompressedBytes	bytes	ComponentId
ck_server_event_MergesTimeMilliseconds	ck_event_MergesTimeMilliseconds	ms	ComponentId
ck_server_event_MergeTreeDataWriterRows	ck_event_MergeTreeDataWriterRows	行	ComponentId
ck_server_event_MergeTreeDataWriterUncompressedBytes	ck_event_MergeTreeDataWriterUncompressedBytes	bytes	ComponentId
ck_server_event_MergeTreeDataWriterCompressedBytes	ck_event_MergeTreeDataWriterCompressedBytes	bytes	ComponentId
ck_server_event_MergeTreeDataWriterBlocks	ck_event_MergeTreeDataWriterBlocks	个	ComponentId
ck_server_event_MergeTreeDataWriterBlocksAlreadySorted	ck_event_MergeTreeDataWriterBlocksAlreadySorted	个	ComponentId
ck_server_event_CannotRemoveEphemeralNode	ck_event_CannotRemoveEphemeralNode	个	ComponentId
ck_server_event_RegexpCreated	ck_event_RegexpCreated	个	ComponentId
ck_server_event_ContextLock	ck_event_ContextLock	个	ComponentId
ck_server_event_RWLockAcquiredReadLocks	ck_event_RWLockAcquiredReadLocks	个	ComponentId
ck_server_event_RWLockAcquiredWriteLocks	ck_event_RWLockAcquiredWriteLocks	个	ComponentId
ck_server_event_RWLockReadersWaitMilliseconds	ck_event_RWLockReadersWaitMilliseconds	ms	ComponentId
ck_server_event_RealTimeMicroseconds	ck_event_RealTimeMicroseconds	μs	ComponentId
ck_server_event_UserTimeMicroseconds	ck_event_UserTimeMicroseconds	μs	ComponentId
ck_server_event_SystemTimeMicroseconds	ck_event_SystemTimeMicroseconds	μs	ComponentId

ck_server_event_SoftPageFaults	ck_event_SoftPageFaults	个	ComponentId
ck_server_event_OSIOWaitMicroseconds	ck_event_OSIOWaitMicroseconds	μs	ComponentId
ck_server_event_OSCPUWaitMicroseconds	ck_event_OSCPUWaitMicroseconds	μs	ComponentId
ck_server_event_OSCPUVirtualTimeMicroseconds	ck_event_OSCPUVirtualTimeMicroseconds	个	ComponentId
ck_server_event_OSReadBytes	ck_event_OSReadBytes	bytes	ComponentId
ck_server_event_OSWriteBytes	ck_event_OSWriteBytes	bytes	ComponentId
ck_server_event_OSReadChars	ck_event_OSReadChars	个	ComponentId
ck_server_event_OSWriteChars	ck_event_OSWriteChars	个	ComponentId
ck_server_event_CreatedHTTPConnections	ck_event_CreatedHTTPConnections	个	ComponentId
ck_server_event_QueryProfilerSignalOverruns	ck_event_QueryProfilerSignalOverruns	个	ComponentId
ck_server_event_CreatedLogEntryForMerge	ck_event_CreatedLogEntryForMerge	个	ComponentId
ck_server_async_metrics_NumberOfTables	ck_async_metrics_NumberOfTables	个	ComponentId
ck_server_async_metrics_NumberOfDatabases	ck_async_metrics_NumberOfDatabases	个	ComponentId
ck_server_async_metrics_MaxPartCountForPartition	ck_async_metrics_MaxPartCountForPartition	个	ComponentId
ck_server_async_metrics_ReplicasSumQueueSize	ck_async_metrics_ReplicasSumQueueSize	个	ComponentId
ck_server_async_metrics_ReplicasMaxMergesInQueue	ck_async_metrics_ReplicasMaxMergesInQueue	个	ComponentId
ck_server_async_metrics_MemoryShared	ck_async_metrics_MemoryShared	bytes	ComponentId
ck_server_async_metrics_MemoryCode	ck_async_metrics_MemoryCode	bytes	ComponentId
ck_server_async_metrics_ReplicasMaxAbsoluteDelay	ck_async_metrics_ReplicasMaxAbsoluteDelay	个	ComponentId
ck_server_async_metrics_ReplicasMaxQueueSize	ck_async_metrics_ReplicasMaxQueueSize	个	ComponentId
ck_server_async_metrics_MemoryVirtual	ck_async_metrics_MemoryVirtual	bytes	ComponentId
ck_server_async_metrics_MarkCacheBytes	ck_async_metrics_MarkCacheBytes	bytes	ComponentId
ck_server_async_metrics_CompiledExpressionCacheCount	ck_async_metrics_CompiledExpressionCacheCount	个	ComponentId
ck_server_async_metrics_ReplicasSumMergesInQueue	ck_async_metrics_ReplicasSumMergesInQueue	个	ComponentId
ck_server_async_metrics_UncompressedCacheBytes	ck_async_metrics_UncompressedCacheBytes	个	ComponentId
ck_async_metrics_ReplicasSumInsertsInQueue	ck_async_metrics_ReplicasSumInsertsInQueue	个	ComponentId
ck_server_async_metrics_MarkCacheFiles	ck_async_metrics_MarkCacheFiles	个	ComponentId

ck_server_async_metrics_MemoryDataAndStack	ck_async_metrics_MemoryDataAndStack	bytes	ComponentId
ck_server_async_metrics_MemoryResident	ck_async_metrics_MemoryResident	bytes	ComponentId
ck_server_async_metrics_ReplicasMaxInsertsInQueue	ck_async_metrics_ReplicasMaxInsertsInQueue	个	ComponentId
ck_server_async_metrics_ReplicasMaxRelativeDelay	ck_async_metrics_ReplicasMaxRelativeDelay	个	ComponentId
ck_server_async_metrics_UncompressedCacheCells	ck_async_metrics_UncompressedCacheCells	个	ComponentId

IMPALA

IMPALA_CATALOG

表十九 IMPALA组件监控指标说明1

指标英文名称（metric name）	指标中文名称	单位	维度
impala_catalog_impala_thrift_server_CatalogService_connections_in_use	当前活跃连接数	个	ComponentId
impala_catalog_thread_manager_total_threads_created	catalogd进程线程创建数量	个	ComponentId
impala_catalog_memory_total_used	catalogd进程内存总使用量	byte	ComponentId
impala_catalog_memory_rss	catalogd进程物理内存使用量	byte	ComponentId

IMPALA_STATE_STORE

表二十 IMPALA组件监控指标说明2

指标英文名称（metric name）	指标中文名称	单位	维度
impala_statestore_thread_manager_total_threads_created	statestored进程线程创建数量	个	ComponentId
impala_statestore_memory_total_used	statestored进程内存总使用量	byte	ComponentId
impala_statestore_memory_rss	statestored进程物理内存使用量	byte	ComponentId

IMPALA_IMPALAD

表二十一 IMPALA组件监控指标说明3

指标英文名称（metric name）	指标中文名称	单位	维度
impalad_metrics_impala_server_query_durations_ms_90th	已完成查询操作耗时时间的90分位点	ms	ComponentId
impalad_metrics_impala_server_query_durations_ms_99_9th	已完成查询操作耗时时间的99.9分位点	ms	ComponentId
impalad_metrics_impala_server_ddl_durations_ms_90th	已完成DDL操作耗时时间的90分位点	ms	ComponentId
impalad_metrics_impala_server_ddl_durations_ms_99_9th	已完成DDL操作耗时时间的99分位点	ms	ComponentId
impalad_metrics_impala_thrift_server_backend_connections_in_use	ThriftServer后端当前活跃连接数	个	ComponentId
impalad_metrics_thread_manager_total_threads_created	impalad进程线程创建数量	个	ComponentId
impalad_metrics_memory_total_used	impalad进程内存总使用量	byte	ComponentId
impalad_metrics_impala_server_num_fragments	已完成fragment总数	个	ComponentId

KUDU

KUDU_TSERVER

表二十二 KUDU组件监控指标说明1

指标英文名称（metric name)	指标中文名称	单位	维度
kudu_tserver_inbound_connections_socket_stats_pacing_rate_max	tserver网络传入每毫秒流量最大值	kps	ComponentId
kudu_tserver_inbound_connections_socket_stats_pacing_rate_min	tserver网络传入每毫秒流量最小值	kps	ComponentId
kudu_tserver_inbound_connections_socket_stats_rtt_max	tserver网络传入往返时间最大值	ms	ComponentId
kudu_tserver_inbound_connections_socket_stats_rtt_min	tserver网络传入往返时间最小值	ms	ComponentId
kudu_tserver_inbound_connections_socket_stats_rttvar_max	tserver网络传入往返时间平均偏差最大值	ms	ComponentId
kudu_tserver_inbound_connections_socket_stats_rttvar_min	tserver网络传入往返时间平均偏差最小值	ms	ComponentId
kudu_tserver_outbound_connections_socket_stats_pacing_rate_max	tserver网络传出每毫秒流量最大值	kps	ComponentId
kudu_tserver_outbound_connections_socket_stats_pacing_rate_min	tserver网络传出每毫秒流量最小值	kps	ComponentId
kudu_tserver_outbound_connections_socket_stats_rtt_max	tserver网络传出往返时间最大值	ms	ComponentId
kudu_tserver_outbound_connections_socket_stats_rtt_min	tserver网络传出往返时间最小值	ms	ComponentId
kudu_tserver_outbound_connections_socket_stats_rttvar_max	tserver网络传出往返时间平均偏差最大值	ms	ComponentId
kudu_tserver_outbound_connections_socket_stats_rttvar_min	tserver网络传出往返时间平均偏差最小值	ms	ComponentId
kudu_tserver_active_scanners	处于active状态的scanner数量	个	ComponentId
kudu_tserver_block_cache_usage	tserver进程块缓存占用的内存	byte	ComponentId
kudu_tserver_cpu_stime	tserver进程的总系统 CPU 时间	s	ComponentId
kudu_tserver_cpu_utime	tserver进程的用户 CPU 总时间	s	ComponentId
kudu_tserver_glog_error_messages	tserver进程中发出的 ERROR 级日志消息数	次	ComponentId
kudu_tserver_memrowset_size	内存中存储行	行	ComponentId
kudu_tserver_num_rowsets_on_disk	硬盘中存储行	行	ComponentId
kudu_tserver_op_apply_queue_length_percentile_99	操作队列长度的99分位数	个	ComponentId
kudu_tserver_op_apply_queue_time_percentile_99	操作在队列的等待时间的99分位数	ms	ComponentId
kudu_tserver_op_apply_run_time_percentile_99	操作执行时间的99分位数	ms	ComponentId
kudu_tserver_reactor_load_percent_percentile_99	reactor线程负载的99分位数	个	ComponentId
kudu_tserver_rows_deleted	节点删除 Row 的数量	行	ComponentId
kudu_tserver_rows_inserted	节点插入 Row 的数量	行	ComponentId
kudu_tserver_rows_updated	节点更新 Row 的数量	行	ComponentId
kudu_tserver_rows_upserted	节点 Upserted Row 的数量	行	ComponentId
kudu_tserver_scanner_duration_percentile_99	scanner耗费时间的99分位数	ms	ComponentId
kudu_tserver_tablets_num_failed	失败的 tablet 个数	个	ComponentId
kudu_tserver_tablets_num_running	当前正在运行的 tablet 个数	个	ComponentId
kudu_tserver_tablets_num_shutdown	当前关闭的 tablet 个数	个	ComponentId
kudu_tserver_tablets_num_stopped	当前停止的 tablet 个数	个	ComponentId
kudu_tserver_tcmalloc_current_total_thread_cache_bytes	tserver线程TCMalloc正在使用的内存	byte	ComponentId
kudu_tserver_threads_running	tablet server线程数	个	ComponentId
status	进程运行状态	状态	
proc_cpu_usage	进程cpu利用率	%	
proc_mem_usage	进程内存利用率	%	

KUDU_MASTER

表二十三 KUDU组件监控指标说明2

指标英文名称（metric name)	指标中文名称	单位	维度
kudu_master_data_dirs_failed	失败的数据目录个数	个	ComponentId
kudu_master_data_dirs_full	full状态的数据目录个数	个	ComponentId
kudu_master_glog_error_messages	master进程中发出的 ERROR 级日志消息数	次	ComponentId
kudu_master_glog_warning_messages	master进程中发出的 WARNING 级日志消息数	次	ComponentId
kudu_master_rpc_connections_accepted	master进程RPC请求接收的数量	个	ComponentId
kudu_master_rpc_incoming_queue_time_percentile_99	master进程RPC队列的等待时间的99分位数	ms	ComponentId
kudu_master_rpcs_queue_overflow	master进程RPC队列溢出次数	次	ComponentId
kudu_master_rpcs_timed_out_in_queue	master进程RPC等待超时	ms	ComponentId
kudu_master_threads_running	master 线程数	个	ComponentId
kudu_master_inbound_connections_socket_stats_pacing_rate_max	master进程网络传入每毫秒流量最大值	kps	ComponentId
kudu_master_inbound_connections_socket_stats_pacing_rate_min	master进程网络传入每毫秒流量最小值	kps	ComponentId
kudu_master_inbound_connections_socket_stats_rtt_max	master进程网络传入每毫秒流量最小值	ms	ComponentId
kudu_master_inbound_connections_socket_stats_rtt_min	master进程网络传入往返时间最小值	ms	ComponentId
kudu_master_inbound_connections_socket_stats_rttvar_max	master进程网络传入往返时间平均偏差最大值	ms	ComponentId
kudu_master_inbound_connections_socket_stats_rttvar_min	master进程网络传入往返时间平均偏差最小值	ms	ComponentId
kudu_master_outbound_connections_socket_stats_pacing_rate_max	master进程网络传出每毫秒流量最大值	kps	ComponentId
kudu_master_outbound_connections_socket_stats_pacing_rate_min	master进程网络传出每毫秒流量最小值	kps	ComponentId
kudu_master_outbound_connections_socket_stats_rtt_max	master进程网络传出往返时间最大值	ms	ComponentId
kudu_master_outbound_connections_socket_stats_rtt_min	master进程网络传出往返时间最小值	ms	ComponentId
kudu_master_outbound_connections_socket_stats_rttvar_max	master进程网络传出往返时间平均偏差最大值	ms	ComponentId
kudu_master_outbound_connections_socket_stats_rttvar_min	master进程网络传出往返时间平均偏差最小值	ms	ComponentId

资源管理

YARN管理

1.进入MapReduce集群列表界面，点击**集群名称**进入详情界面。

2.选择侧边导航**资源管理**进入资源管理界面。资源管理包括Yarn资源队列管理、HDFS存储资源管理。

在资源管理页面的Yarn页面，每个队列拥有一个名为root的默认资源池。在列表页可以选择不同的调度类型。调度类型分为两种：**Capacity Scheduler**和**FAIR Scheduler**。

表一 调度类型说明

调度类型	类型说明
Capacity Schedule	多个组织共享一个Hadoop集群，每个组织可以分配到全部集群资源的一部分。每个组织被配置一个专门的队列，每个队列被配置为可以使用一定的集群资源。队列可以进一步按层次划分，这样每个组织内的不同用户能够共享该组织队列所分配的资源。在一个队列内，使用FIFO调度策略对应用进行调整。
FAIR Scheduler	该调度方式旨在为所有运行的应用公平分配资源。下面解释资源是如何在队列之间公平共享的。想象两个用户A和B，分别拥有自己的队列。A启动一个作业，在B没有需求时A会分配到全部可用资源；当A的作业仍在运行时B启动一个作业，一段时间后，A和B的作业都将使用一半的集群资源。这时，如果B启动第二个作业且其他作业仍在运行，那么第二个作业将与B的其他作业（这里是B的第一个作业）公平的共享B所对应队列的资源。由于这里最终A有一个作业，B有两个作业，所以最终的资源分配是：B的每个作业将占用四分之一的集群资源，合计为一半的资源，而A的作业占用一半的资源。这就是资源在用户之间的公平共享。

表二 资源池字段说明

配置参数项	Hadoop YARN对应参数项	说明
名称	-	指资源池队列的名称，注意同一层级的资源池名称不能重复（必填）
资源份额	yarn.scheduler.capacity..capacity	指该资源池队列占用资源份额，取值范围为0-100（必填）
最大资源份额	yarn.scheduler.capacity..maximum-capacity	资源池限制的最大值，指队列占用资源的最大百分比
单个用户限制比例	yarn.scheduler.capacity..user-limit-factor	允许用户获取的队列资源，取值范围0-1.0（默认情况下值为1.0，确保单个用户不超过队列配置的资源）
最大内存	yarn.scheduler.capacity..maximum-allocation-mb	Resource Manager分配给单个容器的最大内存
最大虚核数	yarn.scheduler.capacity..maximum-allocation-vcores	Resource Manager分配给单个容器的最大虚拟内核
最大应用数	yarn.scheduler.capacity..maximum-applications	队列允许处于运行和挂起状态的最大应用数
Application Master最大资源占比	yarn.scheduler.capacity..maximum-am-resource-percent	可运行application master的最大资源百分比
状态	yarn.scheduler.capacity..state	队列的运行状态。如果队列处于STOPPED状态，则新的application将不能提交给该队列以及其子队列

创建子队列

对创建好的资源池可以实现编辑、删除、创建子队列的操作。

1. 在资源管理yarn队列管理中点击资源池操作列的**创建子队列**按钮，进入创建界面。

2. 根据系统指引填写相关配置。注意：每个节点下的子队列的权重和必须等于100%。

表二 子队列配置说明

配置参数项	Hadoop YARN对应参数项	说明
子队列名称	-	指资源池队列的名称，注意同一层级的资源池名称不能重复（必填）
权重	weight	资源池队列占用的资源份额权重
单个用户限制比例	yarn.scheduler.capacity..user-limit-factor	允许用户获取的队列资源，取值范围0-1.0（默认情况下值为1.0，确保单个用户不超过队列配置的资源）
用户最低资源保证	-	每个用户最低可用资源
最大资源份额	yarn.scheduler.capacity..maximum-capacity	资源池限制的最大值，指队列占用资源的最大百分比
最大分配内存	yarn.scheduler.capacity..maximum-allocation-mb	Resource Manager分配给单个容器的最大内存
最大分配虚核	yarn.scheduler.capacity..maximum-allocation-vcores	Resource Manager分配给单个容器的最大虚拟内核
最大应用数	yarn.scheduler.capacity..maximum-applications	队列允许处于运行和挂起状态的最大应用数
Application Master 最大占比	maxAMShare	运行application master的资源份额占比（注：当该值为-1时，表示禁用该特性，AMShare不做检查）
队列运行状态	-	分为running和stopped

HDFS管理

在侧边导航选择**资源管理-HDFS存储资源管理**，可查看已创建的test用户对应的空间目录以及配额等内容，可以查看到当前用户使用的情况。

1. 点击**创建目录**，选择租户并且按要求填写路径后确定目录创建成功。

2. 选择对应目录操作项**修改配额**按钮，即可修改用户的配额。系统每隔10分钟自动同步配置到集群。也可以点击**同步配置到集群**进行同步。同步后无需重启HDFS服务即可生效。

- Namespace配额：用户目录下能存在的目录数和文件数的配额，默认值200
- 存储空间配额：用户目录存储空间大小，默认值2000M

3. 点击**删除**按钮可对目录进行删除，弹窗再次确认后即可删除成功。

监控报警

集群监控

集群监控为用户提供实时监控和管理集群状态、性能功能以及资源使用情况，以确保集群稳定运行。本章节详细介绍了关于查看监控指标的位置和操作。监控指标说明详见集群指标。

集群仪表盘

1. 在集群列表中点击集群名称进入详情页，侧边导航找到**监控详情-集群仪表盘**。
2. 集群仪表盘支持对时间范围进行筛选查看，右上角点击指标筛选还可以对指标进行筛选。筛选类型分为**常用**和**其他**，根据需要筛选指标项。

主机监控

1. 主机监控列表可以看到具体主机运行状态和具体指标值，点击主机名称可进入详情查看具体指标值的图例形态。主机监控位于集群仪表盘下方，操作步骤可以参照集群仪表盘。
2. 列表支持主机名称、实例id两种类型搜索。状态支持筛选（全部、已停止、运行中），并且指标可排序。

服务监控

1. 进入集群监控详情页，选择侧边导航**服务监控**，能查看当前集群下的服务(已部署服务)的指标数据展示和概览信息。同时，可以进行时间选择和指标筛选。
2. 服务下方是对应的组件，并且可看到组件部署对应的主机，以及主机的进程状态和各使用率。支持按照主机名进行筛选，进程状态可选（全部、进行中和已停止），且支持使用率排序。

集群报警

报警配置

BMR的报警配置都是在BCM侧进行配置的，BMR的报警配置分为BMR事件报警配置和BMR指标报警配置：

配置类型	类型说明
BMR事件报警配置	针对BMR中监控对象(比如主机和组件进程)运行状态(比如down/up)的事件报警配置。 操作步骤： 1.在**产品服务->云监控BCM**页中，在侧边导航点击**事件监控**，参考BCM的事件监控说明，配置BMR的事件报警策略。 2.配置主机运行状态的事件报警策略，产品类型需要选择**MapReduce BMR**, 事件名称选择主机宕和主机宕恢复。
BMR指标报警配置	针对BMR中监控对象指标阈值的报警配置，比如CPU利用率，磁盘利用率超过阈值报警配置。 1.在**产品服务->云监控 BCM**页中，点击**实例组**，参考BCM的实例组说明，配置BMR的实例组以及实例组的报警策略。 2.创建完实例组后，参考BCM的添加实例组报警策略 创建实例组的指标报警策略。

报警管理

当您需要监控各云服务资源的使用和运行情况时，您可以对已接入BCM的云服务设置合理的报警策略，包括对于资源设置性能消耗类指标的阈值报警，也可以对实例或服务的状态即事件监控设置事件报警。 同时针对站点监控中的探测点、应用监控中的实例和自定义监控中的监控项也可以配置合理的报警策略。

添加报警策略

- 指标监控

1. 登录百度智能云，选择**云监控BCM**，在左侧导航栏中点击**报警管理—报警策略—指标报警**，进入报警策略列表页面。
2. 点击**添加策略**按钮，进入创建报警策略页面，填写表单信息完成指标监控策略创建，填写产品类型时需要选择**MapReduce BMR**。

- 事件监控

1. 登录百度智能云，选**择云监控BCM**，在左侧导航栏中点击报警管理—报警策略—云产品事件，进入报警策略列表页面。
2. 点击添加策略按钮，进入“创建报警策略”页面，填写相应的表单信息完成事件监控策略创建。填写产品类型时需要选择**MapReduce BMR**。

报警策略操作

1. 登录百度智能云，选择云监控BCM，在左侧导航栏中点击**报警管理—报警策略**，进入报警策略列表页面。
2. 点击操作列的复制、编辑、删除、启用、禁用按钮，您可以对单个报警策略进行**复制、修改、启用、禁用**或**删除**操作。 勾选策略名称前的复选框，您可以对报警规则进行批量删除，启用，禁用操作。

查看报警策略详情

1. 登录百度智能云，选择**云监控BCM**，在左侧导航栏中点击**报警管理—报警策略**，进入报警策略列表页面。
2. 点击**报警策略名称**链接，您可以查看当前报警策略的详情信息。

说明：为方便您对策略进行编辑操作，在报警策略详情界面也提供了复制、编辑、删除和启用/禁用按钮，您可在查看详情的同时直接在此页面进行相关操作。

> 说明：为方便您对策略进行编辑操作，在报警策略详情界面也提供了复制、编辑、删除和启用/禁用按钮，您可在查看详情的同时直接在此页面进行相关操作。

禁用/启用报警通知

1. 登录百度智能云，选择**云监控BCM**，在左侧导航栏中点击**报警管理—报警策略**，进入报警策略列表页面。

2. 在通知状态列进行操作，展示“ON”则报警通知开启，“OFF”则报警通知关闭。

报警回调

通过报警回调，可实现将BCM云监控的报警通知发送到您指定的URL。您可以自由、灵活的处理各类告警消息，BCM支持通过 HTTP/HTTPS协议 的 POST 请求推送到可访问公网 URL ，您可基于回调接口推送的报警信息做进一步的处理。如需通过企业微信、钉钉、如流等办公软件接收报警通知，请参见webhook使用说明。

操作步骤

报警回调功能的入口有三处：统一的创建报警策略入口、云服务下单个实例创建报警策略入口和创建报警通知模版入口。下面将具体描述报警回调的操作步骤：

表一 报警回调操作步骤

报警回调入口	具体步骤
统一的创建报警策略入口	1.在左侧导航栏中点击**报警管理—报警策略**，在云产品监控的策略列表下，点击**添加策略**。 2.在创建策略页面，点击**报警回调**按钮开启，输入公网可访问的 URL 地址。
云服务下单个实例创建报警策略入口	1.在左侧导航栏中点击云产品监控，点击要查看的云产品，进入该云产品的实例列表页面。如查看云服务器BCC监控数据，点击云服务器监控，进入“云服务器列表”页面。然后选择对应的实例点击进入报警策略页面。 2.在实例报警策略页面，点击添加策略。 3.在“创建策略”页面，开启报警回调，输入公网可访问的 URL 地址。
创建报警通知模版入口	1.在左侧导航栏中点击**报警管理—报警模版**，在报警动作列表页面，点击添加模版。 2.在添加通知模版页面，接口回调一栏，输入公网可访问的 URL 地址。

webhook使用说明

表二 操作步骤说明

使用方式	使用步骤
企业微信	1. 登录企业微信，打开需要接收告警通知的企业微信群。 2. 添加群机器人后，复制webhook地址，参考操作步骤填写到“报警回调”中即可。 3. 配置成功后，当报警通知被触发时，您可以在企业微信群收到报警通知。
钉钉	1. 登录钉钉，打开需要接收告警通知的钉钉群，添加群机器人。 2. 填写表单，“安全设置”模块勾选“自定义关键词”选项，建议填写“报警”作为关键词。
如流	1. 登录如流，打开需要接收告警通知的如流群。 2. 群内添加如流机器人，复制webhook地址，参考操作步骤填写到**报警回调**中即可。 3. 配置成功后，当报警通知被触发时，您可以在如流群收到报警通知。

POST方式参数说明

表四 指标报警POST方式参数说明

参数	说明
alertId	告警ID
userId	账号ID
alarmName	报警策略名称
scope	云产品名称
policyType	策略类型（指标报警和事件报警之一，Metric代表是指标报警，Event代表事件报警）
alertStartTimestamp	发生告警的时间戳
region	报警对象所在的地域
monitoringObject	发生报警的对象
alarmLevel	报警等级状态。根据实际情况返回严重、通知、重要、警告四种状态中的一种
formula	报警条件
currentValue	报警发生或恢复时监控项的当前值
taskTimestamp	报警回调发送时间
signature	签名

表五 事件报警POST方式参数说明

参数	说明
alarmName	报警策略名称
scope	云产品名称
alertStartTimestamp	发生告警的时间戳
alertContent	事件详情
taskTimestamp	报警回调发送时间
signature	签名

URL回调实例，下面是URL回调的使用实例，BCM发起的POST方式URL回调请求：

```
 POST http://127.0.0.1:8201/callback
请求Body("Content-Type": "application/json") :
{
    "alarmStatus":"报警-异常",
    "alertId":"19925050-3f77-4839-bae7-6a5f721aae0c",
    "userId":"your_user_id",
    "alarmName":"test_bcc_alarm",
    "scope":"BCE_BCC",
    "policyType":"Metric",
    "alertStartTimestamp":1698982559,
    "region":"北京",
    "monitoringObject":"i-6nfua8xc/bcc-test-bj/-(公)/192.168.16.12(内)",
    "alarmLevel":"重要",
    "formula":"CPU使用率1分钟平均值 > -1 %",
    "currentValue":"CPU使用率=0.50%",
    "taskTimestamp":1698982642,
    "signature":"88e647b853e480046632a5eb9fef70f5"
}
```

在callback.java文件中接收POST参数并对消息进行校验：

```
 // 从发送来的POST请求中解析出alertId、taskTimestamp 、signature这3个参数。使用alertId、token（创建报警策略时生成的token）和taskTimestamp 这3个参数字符串连接并用MD5算法加密后的值来校验消息。
如果校验成功，则说明此消息为百度云发出，否则为非法请求，不予处理。其中taskTimestamp可以用来做过期验证，如果时间戳与用户当前时间时间间隔大于某个周期（如10分钟），则用户可自行丢弃请求。

if (md5(alertId + token + taskTimestamp) == signature) {
    ..........
}
```

报警历史

当报警发生后，您可以在报警历史页面通过产品类型、报警等级、当前状态等条件筛选想要关注的报警信息。

查看报警历史

1. 登录百度智能云，选择**云监控BCM**，在左侧导航栏中点击**报警管理—报警历史**，进入报警历史列表页面。

2. 切换tab页，可以分别查看云产品监控、站点监控、自定义监控、应用监控的报警历史信息。

查看报警详情

1. 登录百度智能云，选择**云监控BCM**，在左侧导航栏中点击**报警管理—报警历史**，进入报警历史列表页面。

2. 在报警历史页面，点击报警内容打开您要查看的报警事件的详情页面。

3. 在报警事件详情页面，可以查看该报警事件的基本信息，数据监控详情及该报警事件的状态变更历史。

节点管理

本章节旨在指导您进行节点管理操作，涵盖各关键环节，帮助您熟练掌握相关技能。

前提条件

已创建BMR集群，详情请参照创建集群。

操作步骤

进入节点管理：

1. 在**产品服务>MapReduce>集群列表页**，单击被调整集群对应节点管理按钮，进入该集群的节点管理列表页。

2. 查看节点的详情，对于节点的实例详情以列表形式展示。展示信息包括申请节点数量和运行数量。

表一 节点管理列表项说明

列表项	列表项说明
实例ID	节点在系统中的唯一标识，用于准确区分和定位各个节点实例，方便系统管理和用户识别特定节点。
主机名称	节点所在主机的名称。
实例状态	节点当前的运行状态。分为运行中、变更中、变更失败、重启中、启动中、已停止。
公网IP	节点在公共网络中的 IP 地址。
内网IP	节点在内部网络中的 IP 地址
操作项	• VPC:单击 “VPC” 可远程连接节点进行操作。 • 重启：单击 “重启” 可重启对应节点。重启实例，实例中部署的服务会被重新拉起，但可能会造成您历史数据的丢失，请谨慎操作。 • 绑定公网 IP：单击 “绑定公网 IP” 可为未分配公网 IP 的节点绑定。

3. 节点管理支持对节点进行扩缩容、规格变更、新增实例类型和磁盘变更。

表二 变更类型说明

变更类型	变更类型说明
数目变更	masker不支持变更数目、clickhouse集群core支持扩缩容，其他类型core仅支持缩容、task和client支持扩缩容。
规格变更	本地盘实例不支持规格变更。变更需要重启实例，业务会短暂中断，建议在业务不繁忙时操作。
扩容	单击扩容按钮在扩容配置处对实例数目进行增加，最大实例数为200，最低为2。扩容结束后点击确认变更，系统跳转完成页。注意：实例变更将在5-15分钟内生效，同时集群状态变为“调整中”。参照集群扩容。
缩容	目前 ClickHouse 支持对core节点进行扩缩容，缩容方式为按分片缩容。单击节点处缩容按钮在容配置处对分片数量进行填写。参照集群缩容。
磁盘变更	masker和core仅支持扩容、task和client支持扩缩容。支持cds类型变更，支持选择价格更高的磁盘类。数量的调整支持增加，不支持减少。本地盘不支持变更
新增实例类型	单击新增实例类型按钮，进入新增实例类型界面。支持新增未被添加的实例组类型。

访问集群

SSH连接到集群

应用场景

在集群运行期间与主节点交互。例如，登录主节点运行交互式查询，检查日志文件，使用在主节点上运行的应用程序监控性能，调试集群问题等。

使用安全外壳协议（SSH）连接到主节点可实现监控集群并与集群交互。您可在主节点上发出Linux命令，以交互的形式运行HBase、Hive和Pig等应用程序，浏览目录和读取日志文件等。

操作步骤

1. 在集群创建成功后，集群列表页点击集群，进入集群详情页，在节点信息点击**Master节点**，获取Master节点的公网IP，SSH端口默认22。为用户集群安全性考虑，BMR新建集群默认情况下不开启22端口，如果用户需要通过22端口访问集群，可以前往VPC的安全组中进行规则的修改，添加22端口，再进行访问。

具体操作流程如下：

a.在产品列表找到私有网络vpc点击进入。

b.选择进入相应BMR集群的安全组中进行修改，BMR集群安全组ID可以在BMR集群详情页中进行查看。

c.点击进入VPC安全组中，可以搜索查看相应的BMR集群的安全组。

d.点击安全组名称： BaiduMapReduce-Default 进入安全组详情，在入站规则中点击添加规则，添加22端口即可，添加完成后可以通过SSH方式使用22端口进行登录访问集群。

2. 在集群运行期间，连接Master节点。集群IP地址和端口号可至站内信中查看，用户名和密码是创建集群时设置的用户名和密码。

表一 登录方式说明

登录方式	登录方式说明
Linux环境	通过 ssh username@eip -p [端口号]命令连接Master节点。其中，eip为Master节点的公网IP地址，username是root，密码是创建集群时设置的管理员密码。内网用户建议通过Master内网IP地址进行访问。
Windows环境	通过SSH客户端（Putty、SecureCRT及Xshell等）登录，使用SSH协议进行登录。

3. 验证通过后，成功登录主节点，用户可对集群进行相关操作。

访问集群-openVPN访问集群

使用BMR集群的VPN服务，需配置OpenVPN Client，本章介绍在Mac OS X、Linux和Windows操作系统上配置OpenVPN Client的过程。

Mac OS X

Mac OS X（10.11.2）具体配置操作步骤如下：

1. 下载Tunnelblick的安装文件。

- 官方网址：https://github.com/Tunnelblick/Tunnelblick/releases/tag/v3.8.0。
- 下载地址：https://bmr.bj.bcebos.com/tools/openvpn/mac/Tunnelblick_3.8.0_build_5370.dmg。

2. 安装Tunnelblick工具。选择默认安装即可。
3. 下载OpenVPN Client配置文件。登录控制台，选择**产品服务->MapReduce-集群列表**，进入集群列表页，点击**集群名称**进入集群详情页，在“工具下载”区点击下载**openvpn-conf.zip**。
4. 解压“openvpn-conf.zip”文件后，进入目录“openvpn-conf/client/”，双击**client.ovpn**后，Tunnelblick启动安装OpenVPN Client配置，如下图所示，请选择分享配置给**所有用户**或**只是我**。

图一 分享配置确认界面

5. Tunnelblick完成OpenVPN Client的配置安装后，点击右下角的**链接**即可。
6. 打开浏览器，可使用Master节点的内网IP访问集群服务，请在集群详情页的**节点信息**区获取内网IP。

Linux（centos6.5）

Linux（centos6.5）具体配置操作步骤如下：

准备工作

安装lzo和libpam：

```
 # 下载lzo源码包，下载地址：http://bmr.bj.bcebos.com/tools/openvpn/linux/lzo-2.09.tar.gz
tar zxf lzo-2.09.tar.gz
cd lzo-2.09
./configure
make
make install
# 下载libpam源码包，下载地址：http://bmr.bj.bcebos.com/tools/openvpn/linux/Linux-PAM-1.2.1.tar.bz2
tar xvf Linux-PAM-1.2.1.tar.bz2
cd Linux-PAM-1.2.1
./configure
make
make install
```

通过yum安装OpenVPN Client

1. 下载并安装OpenVPN。执行命令`yum install openvpn`。
2. 下载OpenVPN Client配置文件并解压。登录控制台，选择“产品服务->MapReduce-集群列表”，进入集群列表页，点击集群名称进入集群详情页，在“工具下载”区点击下载“openvpn-conf.zip”并解压。

3. 复制“openvpn-conf/client/”目录下的所有文件至OpenVPN的安装目录“/etc/openvpn”。命令如下：

```
 cp client/* /etc/openvpn
service openvpn start
```

源码安装OpenVPN Client

1. 下载OpenVPN源码。

- 源码下载地址:http://bmr.bj.bcebos.com/tools/openvpn/linux/openvpn-2.3.10.tar.gz。

2. 解压并安装。命令如下： tar zxf openvpn-2.3.10.tar.gz cd openvpn-2.3.10 ./configure # 若是非root用户，请加入--prefix来指定安装路径 make make install

3. 下载OpenVPN Client配置文件并解压。登录控制台，选择**产品服务->MapReduce-集群列表**，进入集群列表页，点击**集群名称**进入集群详情页，在工具下载区点击下载**openvpn-conf.zip**，下载后解压。

4. 进入OpenVPN Client目录，启动OpenVPN。命令如下：

```
 cd openvpn-conf/client
/usr/sbin/openvpn --config client.conf # 若是非root用户，请使用sudo
```

Windows

以Windows 7（64bit）为例。

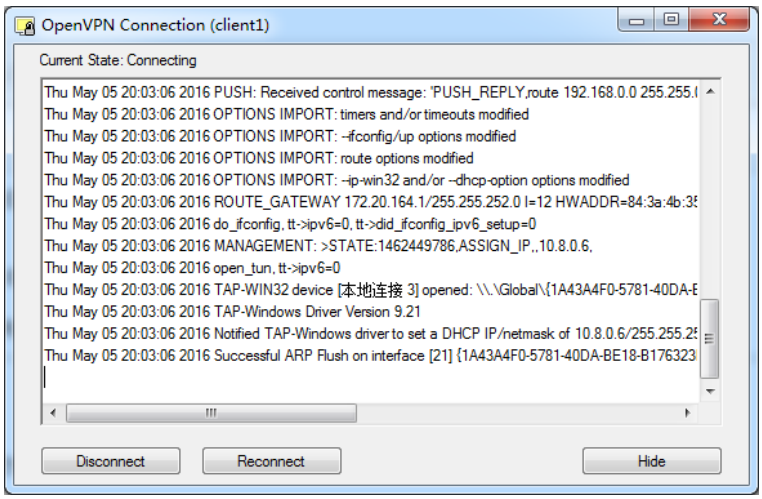
1. 下载并安装OpenVPN。打开http://bmr.bj.bcebos.com/tools/openvpn/windows/openvpn-install-2.3.10-I002-x86_64.exe，下载后安装，选择默认安装即可。

2. 下载OpenVPN Client配置文件。登录控制台，选择**产品服务->MapReduce-集群列表**，进入集群列表页，点击**集群名称**进入集群详情页，在工具下载区点击下载**openvpn-conf.zip**。

3. 解压**openvpn-conf.zip**文件后，复制“openvpn-conf/client/”目录下的所有文件至OpenVPN的安装目录“OpenVPN\config”内。

4. 在OpenVPN的安装目录中打开程序**openvpnserv.exe**，自动连接OpenVPN Server，如下图所示：

图二 连接界面



使用OpenVPN提交Hadoop作业

本章介绍如何使用OpenVPN在Linux、Windows和Mac OS X操作系统中提交Hadoop作业。

使用客户端提交作业时，需在系统中设置环境变量，即“HADOOP_USER_NAME=hdfs”，或在MapReduce作业中配置，即在程序第一行加上“System.setProperty("HADOOP_USER_NAME","hdfs")”，可实现以hdfs用户的身份提交作业。

Linux

在Linux操作系统下通过OpenVPN提交Hadoop作业。

1. 下载Hadoop Client。

- 下载地址：http://bmr.bj.bcebos.com/tools/hadoop/hadoop-2.6.0-SNAPSHOT.tar.gz

2. 下载Hadoop配置文件。

登录控制台，选择**产品服务->MapReduce-集群列表**，进入集群列表页，点击集群名称进入集群详情页，在工具下载区点击下载**hadoop-conf.zip**。

3. 解压“hadoop-conf.zip”。

替换“hadoop-2.6.0-SNAPSHOT/etc/hadoop/”下的所有文件为“conf”中的配置文件。

4. 由于hadoop脚本中使用了配置文件中的java路径，可能和客户端机器不一致。

如果客户端机器已经将java加入过path中，请修改bin/hadoop文件:cd至目录hadoop-2.6.0-SNAPSHOT/bin执行命令vi hadoop，修改exec $JAVA $JAVA_HEAP_MAX $HADOOP_OPTS $CLASS "$@"为exec java $JAVA_HEAP_MAX $HADOOP_OPTS $CLASS "$@"。

5. 提交hadoop作业。

   cd至目录hadoop-2.6.0-SNAPSHOT/bin执行hadoop jar {jar file} {main class path} {parameters}。如提示文件压缩的类找不到，是指镜像缺少这个类，则需在core-site.xml中的“io.compression.codecs”找到该类，并去掉该异常类。

Windows

使用Eclipse在Windows操作系统通过OpenVPN中提交Hadoop作业的过程如下：

配置Hadoop Client。

1. Hadoop Client下载地址：http://bmr.bj.bcebos.com/tools/hadoop/hadoop-2.6.0-SNAPSHOT.tar.gz。下载后解压。
2. 点击下载：https://github.com/srccodes/hadoop-common-2.2.0-bin/archive/master.zip。下载后解压，替换“\hadoop-2.6.0-SNAPSHOT\bin\”中的文件为“\hadoop-common-2.2.0-bin-master\bin\”中的文件。

> 注意：
> 如果系统的jdk是1.8，而集群是1.7，作业打包提交后，可能会出现版本冲突的异常，必须卸载java1.8，重新下载1.7版本安装。

配置Hadoop配置文件

1. 登录控制台，选择**产品服务->MapReduce-集群列表**，进入集群列表页，点击集群名称进入集群详情页，在工具下载区点击下载**hadoop-conf.zip**。
2. 解压后在mapred-site.xml中加入如下代码确保系统一致：

```
  <property>
    <name>mapred.remote.os</name>
    <value>Linux</value>
  </property>
  <property>
    <name>mapreduce.app-submission.cross-platform</name>
    <value>true</value>
  </property>
```

3. 从下列中选择一种方式创建工程。

表一 创建方式说明

创建方式	创建步骤
创建普通java工程	a.复制配置文件“conf”中的所有文件至“src”下。 b.添加Hadoop Client的“etc/share”目录下的如下文件至依赖中： mapreduce/.jar;mapreduce/lib/.jar;hdfs/.jar;hdfs/lib/.jar;yarn/.jar;yarn/lib/.jar;common/*.jar;common/lib/*.jar。
创建Maven项目	a.复制配置文件“conf”中的所有文件至“/main/resources”下。 b.在pom.xml中配置hadoop-hdfs，hadoop-common，hadoop-mapreduce-client-core，hadoop-mapreduce-client-common，hadoop-mapreduce-client-jobclient。版本须与集群保持一致。

4. 编写主函数。主函数中需要增加hadoop镜像路径的配置以及作业文件位置的配置。main函数可参考如下代码：

```
 public static void main(String[] args) throws Exception {
String hadoop_home = “”;//hadoop mirror path
String jar_path = “”;
String input_path = “”;
String output_path = “”;
String job_name = “”;

System.setProperty("hadoop.home.dir", hadoop_home);

Configuration conf = new Configuration();
conf.set("mapreduce.job.jar", jar_path);

Job job = new Job(conf, job_name);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);

job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);

FileInputFormat.addInputPath(job, new Path(input_path));
FileOutputFormat.setOutputPath(job, new Path(output_path));

job.waitForCompletion(true);
}
```

5. 编译提交。

表二 编译提交步骤

编译类型	操作步骤
提交普通java工程	a.右键点击项目选择“export jar”使用普通打包方式即可。 b.右键点击项目选择run as java application，即可向集群提交作业。
提交Maven项目	a.右键点击项目选择run as maven install，生成的jar文件会存放在Maven项目包所在目录的target文件夹中。 b.复制target中的jar包以备引用，否则会出现文件正在被占用的异常。

Mac

与Linux相同，请参考Linux。

访问集群服务页面

访问集群Hadoop Yarn服务页面

通过Web UI可访问Hadoop Yarn，具体操作如下：

1. 打开**产品服务>MapReduce>MapReduce-集群列表**，点击已创建的集群，进入实例详情页面。
2. 点击服务页面栏中的**Hadoop Yarn**对应的链接。
3. 在弹出的认证页面中输入创建集群时设置的用户名和密码即可访问集群服务页面。

访问集群的Hue Web页面

BMR提供了两种访问Hue服务页面的方式：Web UI、SSH Tunnel。在BMR中，您可以选择添加Hue应用实现以下操作：编辑与执行Hive脚本和查看集群HDFS。下面将详细介绍两种访问Hue Web页面的方式。

通过Web UI访问Hue Web页面

1. 在创建集群中设置软件配置时，添加Hue和Hive应用。
2. 打开**产品服务>MapReduce>MapReduce-集群列表**，点击已创建的集群，进入实例详情页面。
3. 点击服务页面栏中的**Hue**对应的链接。
4. 在弹出的认证页面中输入创建集群时设置的用户名和密码即可访问集群服务页面。

通过SSH Tunnel访问Hue Web页面

环境准备

表一 环境准备说明

准备步骤	步骤说明
1.建立SSH Tunnel	Linux环境下： 通过 ssh -ND [local_port][username]@[eip] -p [端口号]命令连接Master节点。执行命令后即与Master节点建立了SSH Tunnel，且Master节点不返回响应信息。 -eip：Master节点的公网IP地址。 -local_port：选择一个本机未被使用的端口。 -端口号：SSH端口。 Windows环境下： a.下载并配置Putty b.选择SSH>Tunnels，输入一个本机没有被使用的端口，并选择“Dynamic”和“IPv4”，保持“Destination”为空，点击“Add”按钮添加。
2.配置浏览器	操作步骤： a.配置Firefox，选择“远程DNS”选项，以便正确解析主机。 b.配置Chrome，将SOCKS代理设置为127.0.0.1。

操作步骤

1.在创建集群中设置软件配置时，添加Hue和Hive应用。

2.建立SSH Tunnel并配置浏览器，具体操作请参考[准备](#通过SSH Tunnel访问集群的环境准备)中的所有步骤。

3.代理设置完后，在Master结点上执行命令hostname获取hostname。

4.通过 `http://hostname:8888/` 访问Hue Console。

访问集群的Ganglia页面

在BMR的集群中集成了Ganglia服务，您可访问Ganglia的Web前端。Ganglia是UC Berkeley发起的一个开源集群监视项目，设计用于测量数以千计的节点。Ganglia的核心包含gmond、gmetad以及一个Web前端。主要是用来监控系统性能，如：CPU、mem、硬盘利用率，I/O负载、网络流量情况等，通过曲线很容易见到每个节点的工作状态，对合理调整、分配系统资源，提高系统整体性能起到重要作用。

通过SSH Tunnel可访问Ganglia服务页面，具体操作如下：

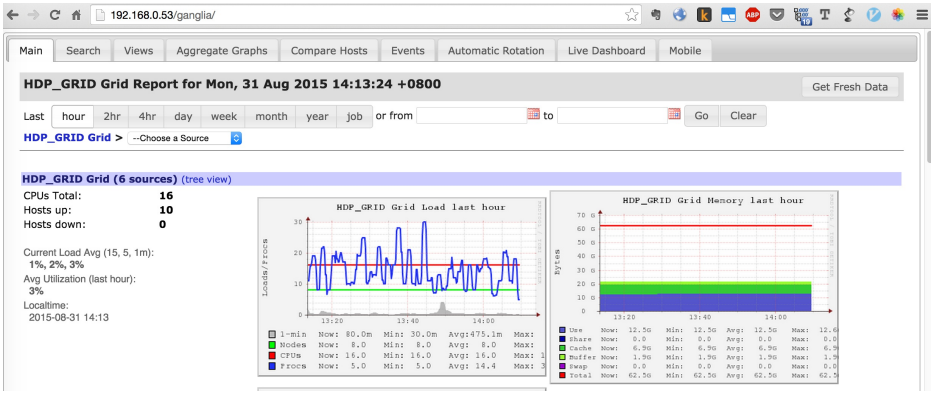
1. 建立SSH Tunnel并配置浏览器，具体操作请参考上一章环境准备中的所有步骤。

2. 代理设置完成后，可通过以下两个地址访问Ganglia服务：

   - master_name/ganglia，master_name即为master节点的hostname。如下图所示，“ng175fe8a-master-instance-8y6oeh6l”是master结点的hostname：
   - 内网IP/ganglia，内网IP可以在控制台的“实例详情”获取。

3. Ganglia服务页面如下图所示：

表二 Ganglia服务界面



服务管理

用户可在服务管理页面查看当前集群已安装服务运行状态（支持按照全部状态、运行中和已停止筛选）、WEB UI链接，支持对服务进行启动、停止、重启操作。

1.点击具体服务名称，进入服务详情可查看具体组件列表、部署拓扑、配置管理和配置历史。

表一 详细信息说明

详情信息名称	说明
组件	展示当前服务的所有组件主机数量、停止数运行数，用户可以对任一组件进行启动、停止、重启等操作。
部署拓扑	1.支持按照组件类型进行筛选列表，查看组件的状态（支持按照全部状态、运行中和已停止筛选）以及组件名称和一些具体信息。 2.点击组件操作列可对组件进行不同操作（启动、停止和重启）。
配置管理	在配置管理界面可以直观查看组件对应的配置项和配置项内容。除此之外还可以对组件进行操作。 1.选择相应组件，点击页面新增配置项按钮，填写配置项名称和配置项内容后点击确定。 2.确认完成选择保存相关配置文件后，请点击配置下发按钮，否则编辑后的内容不会生效。
配置历史	查看每次提交记录的备注，操作支持查看不同配置版本间的区别。支持按照时间范围进行筛选，同时可以根据配置文件或者操作人搜索历史记录。

2.服务管理详情界面上方导航栏下拉可选择不同服务，点击不同服务可进行页面跳转，查看和管理不同服务。

图一 服务选择界面

管理作业

创建作业

注：自2024年6月30日起，MapReduce暂不提供作业相关功能支持，可通过第三方平台EasyDAP或开源组件Airflow提交任务。

使用hadoop镜像的集群可添加的作业类型是：java，streaming。使用spark镜像的集群可添加作业类型：spark，java，streaming。集群中添加了应用后便可添加该应用的作业，即创建集群时添加了hive应用，则可创建hive作业，添加了pig应用，则可创建pig作业。

> BMR将为用户保留一年的作业历史记录，超过此期限的作业历史将被清空，请注意保存和复制相关作业。

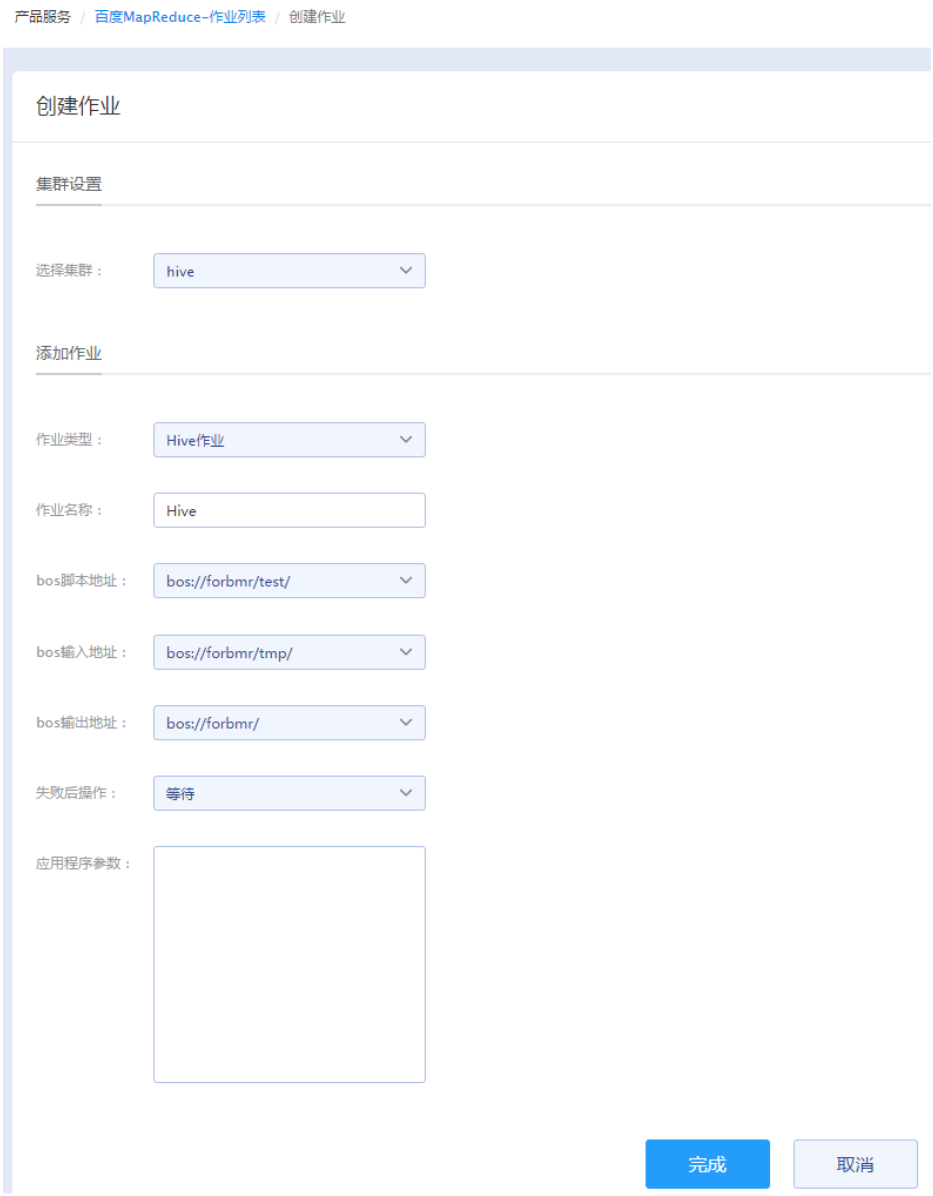
创建作业的操作步骤如下：

1. 在“产品服务>MapReduce>MapReduce-作业列表”页中，点击“创建作业”，进入创建作业页。
2. 请在创建作业页选择作业类型并配置作业类型对应的参数。以下列举了所有作业类型对应参数配置说明：

- Streaming作业
  - 作业名称：输入作业名称，长度不可超过255个字符。
  - Mapper：Mapper是负责把您输入的作业分解成多个作业，对作业进行处理。Mapper对应的输入地址是应用程序在BOS中的地址。
  - Reducer：Reducer是负责把分解后多任务处理的结果汇总起来。Reducer对应的输入地址是应用程序在BOS中的地址。
  - bos输入地址：这个地址必须已经存在，并且您有权限读取这个地址的文件。
  - bos输出地址：写入的bucket之后的地址必须是不存在的，但您有权限对这个地址进行写操作，否则作业运行会失败。
  - 失败后操作：选择作业运行失败后的操作：继续（作业执行失败后，继续执行下一个作业）和等待（作业执行失败后，查看作业运行的状态，并且取消后续作业）。
  - 应用程序参数：除了以上五种streaming program作业必须输入的参数外，若您还有其他参数的设置，请在参数输入框中以空格为分隔符输入参数配置。用户输入参数时，只需要输入参数本身字符串即可，用空格分隔，无需参数转义和url encode。
- Java作业
  - 作业名称：输入作业名称，长度不可超过255个字符。
  - 应用程序位置：输入JAR包在bos上的地址。
  - 失败后操作：选择作业运行失败后的操作：继续（作业执行失败后，继续执行下一个作业）和等待（作业执行失败后，查看作业运行的状态，并且取消后续作业）。
  - MainClass：写入org.apache.MyClass指定主程序的类名。
  - 应用程序参数：输入参数，且参数不做任何修改传给MainClass中的main函数。输入参数时，只需要输入参数本身字符串即可，用空格分隔，无需参数转义和url encode。

- Spark作业
  - 作业名称：输入作业名称，长度不可超过255个字符。
  - 应用程序位置：输入JAR包在bos上的地址。
  - 失败后操作：选择作业运行失败后的操作：继续（作业执行失败后，继续执行下一个作业）和等待（作业执行失败后，查看作业运行的状态，并且取消后续作业）。
  - Spark-submit：Spark的系统参数，Spark在判定Spark-submit输入时遵循如下规则：
    - 当对以下的参数进行多次设置时，只有最后一次设置才会生效：--name、--driver-memory、--driver-java-options、--driver-library-path、--driver-class-path、--executor-memory、--executor-cores、--queue、--num-executors、--properties-file、--jars、--files、--archives。
    - 不可以指定的参数有：--master、--deploy-mode、--py-files、--driver-cores、--total-executor-cores、--supervise、--help。对于不可以指定的参数，若指定了不会生效。
    - 输入参数时，只需要输入参数本身字符串即可，用空格分隔，无需参数转义和url encode。
    - 应用程序参数：输入自定义参数。
- Pig作业
  - 作业名称：输入作业名称，长度不可超过255个字符。
  - bos脚本地址：bos的脚本地址必须是一个有效的bos路径，并且指向Hive脚本。
  - bos输入地址：这个地址必须已经存在，并且您有权限读取这个地址的文件。可在脚本中通过${INPUT}引用这个地址。
  - bos输出地址：写入的bucket之后的地址必须是不存在的，但您有权限对这个地址进行写操作，否则作业运行会失败。可在脚本中通过${OUTPUT}引用这个地址。
  - 失败后操作：选择作业运行失败后的操作：继续（作业执行失败后，继续执行下一个作业）和等待（作业执行失败后，查看作业运行的状态，并且取消后续作业）。
  - 应用程序参数：输入以下指定的参数进行相关的配置：-D key=value指定配置，-p KEY=VALUE指定变量，也可加入自定义参数。输入参数时，只需要输入参数本身字符串即可，用空格分隔，无需参数转义和url encode。
- Hive作业
  - 作业名称：输入作业名称，长度不可超过255个字符。
  - bos脚本地址：BOS的脚本地址必须是一个有效的BOS路径，并且指向Hive脚本。
  - bos输入地址：这个地址必须已经存在，并且您有权限读取这个地址的文件。可在脚本中通过${INPUT}引用这个地址。
  - bos输出地址：写入的bucket之后的地址必须是不存在的，但您有权限对这个地址进行写操作，否则作业运行会失败。可在脚本中通过${OUTPUT}引用这个地址。
  - 失败后操作：选择作业运行失败后的操作：继续（作业执行失败后，继续执行下一个作业）和等待（作业执行失败后，查看作业运行的状态，并且取消后续作业）。
  - 应用程序参数：只接受两种参数类型，分别是--hiveconf key=value 和 --hivevar key=value。前一种参数是用来覆盖hive执行时的配置。后一种参数是用来声明自定义的变量，可以在脚本中通过${KEY}来引用。输入参数时，只需要输入参数本身字符串即可，用空格分隔，无需参数转义和url encode。

3. 选择适配的集群。
4. 点击“完成”，则作业创建完成。



5. 当作业状态会由“等待中”更新为“运行中”状态，作业运行完毕后状态更新为“已完成”。

6. （可选）只有等待中或运行中的作业可被取消，点击“取消作业”即可。

使用远程文件

如果您的作业参数需要依赖本地文件，可以选择使用“附加文件”功能，将远程文件映射到本地路径，即可直接使用远程文件。例如hadoop中的-libjars参数只支持本地文件，通过添加附加文件参数就可以让-libjars使用BOS上的文件，您只需将文件上传至BOS，Hadoop作业即可读取到文件。

需要注意的是，在应用程序参数中使用的文件名需要和本地文件路径设置的文件名保持一致。例如，附加文件：远程文件路径bos://path/to/testA.jar，本地文件路径testB.jar，应用程序参数-libjars testB.jar。

- 操作步骤：

在创建作业页面，填写“附加文件”中的“远程路径”和“本地路径”；“远程路径”对应的是BOS上的文件路径，“本地路径”对应的是作业参数中需要使用的文件名。

- 查看详情：

作业详情页面的“运行参数”中显示有附加文件的远程路径和本地路径。

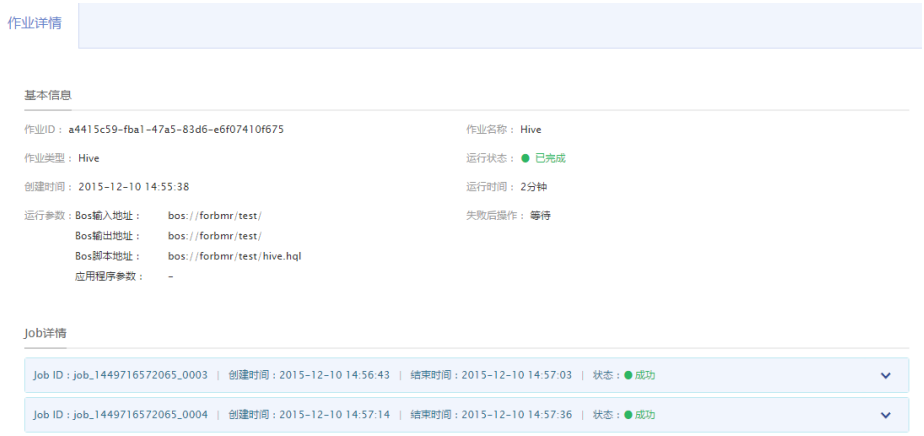
集群详情中的作业列表也有附加文件的信息。

- **注意** 定时任务的作业中添加附加文件，步骤与上述一致。

查看作业

注：自2024年6月30日起，MapReduce暂不提供作业相关功能支持，可通过第三方平台EasyDAP或开源组件Airflow提交任务。

1. 在“产品服务>MapReduce>MapReduce-作业列表”中，点击作业名称，可查看作业基本信息。

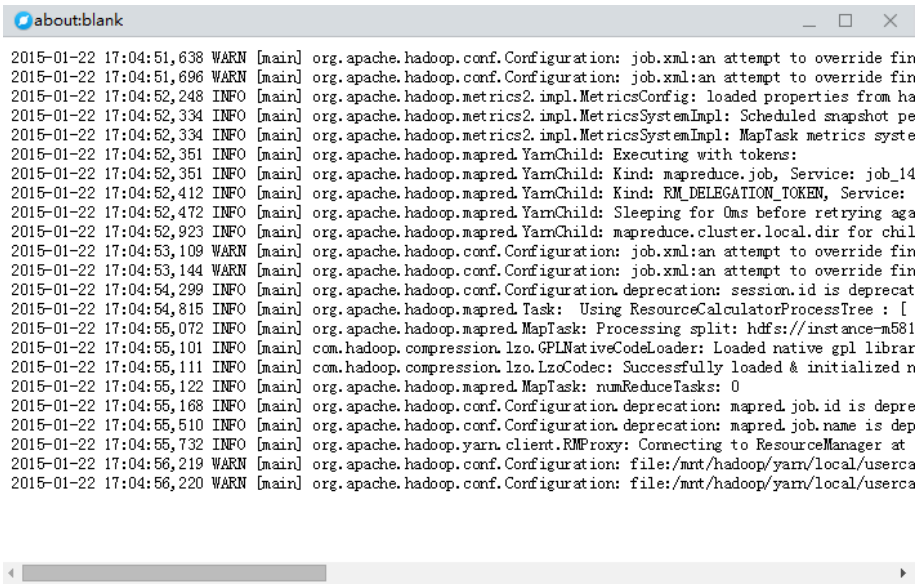


2. Hadoop将job分成若干个task进行处理，共有两种类型的task，分别为map task和reduce task。点击下拉尖括号，查看各task的任务处理情形。

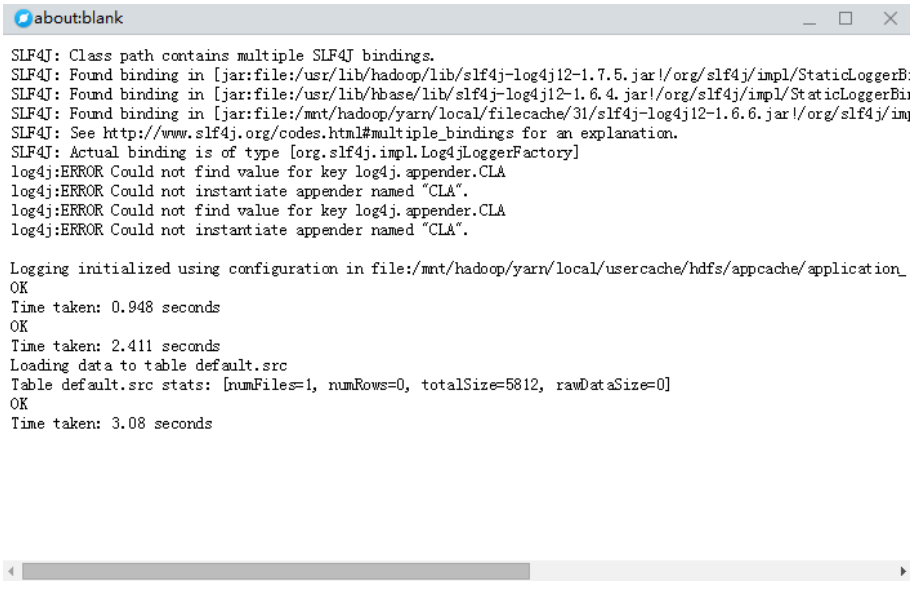
3. 当task运行失败时，每个task有四次的重试机会，每次的重试信息记录在“查看Attempt”中。点击“MAP”和“REDUCE”对应的“查看Attempt”，查看attempt信息。

4. 点击“作业日志”，可查看作业日志，其中Spark作业无作业日志。共有三种日志类型显示，分别是syslog、stderr和stdout。

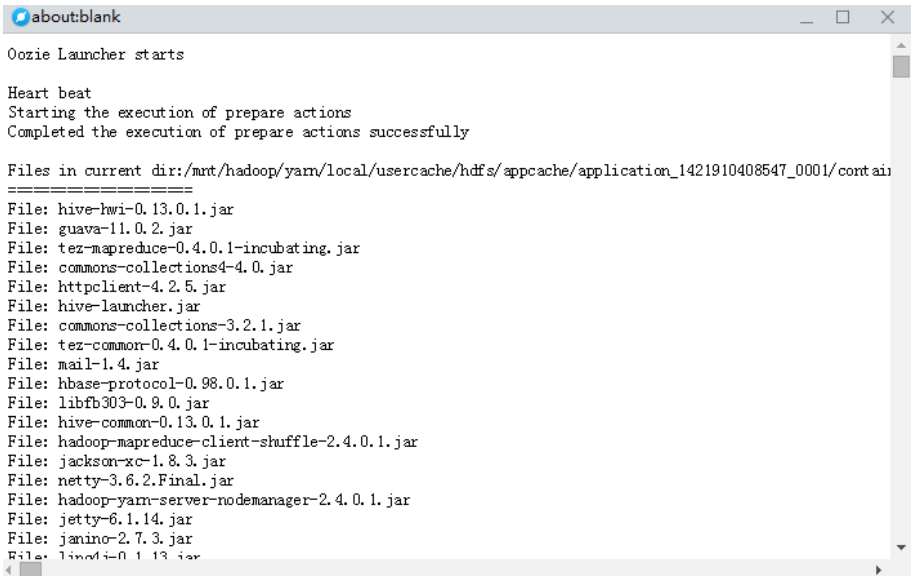
- Syslog是记录作业运行时的详细信息；



- stderr是记录运行作业时的错误信息；



- stdout记录作业运行完毕后的输出信息；



诊断、调优

注：自2024年6月30日起，MapReduce暂不提供作业相关功能支持，可通过第三方平台EasyDAP或开源组件Airflow提交任务。

目的

- 诊断运行失败的作业，在日志中定位失败的原因，精确定位到您的程序中错误的位置。
- 调优运行成功的作业，基于经验评价作业的配置和参数的合理性，给予您调优的建议。

适用范围

诊断或调优Hadoop MR作业，Hadoop Streaming作业，Spark作业。后续会增加Hive、Pig、HBase的诊断或调优。

查看失败作业的诊断信息

1. 在集群作业页面，点击已失败作业的作业名称可查看该作业详情。

2. 在作业详情页面，点击“诊断”，进入诊断页面。

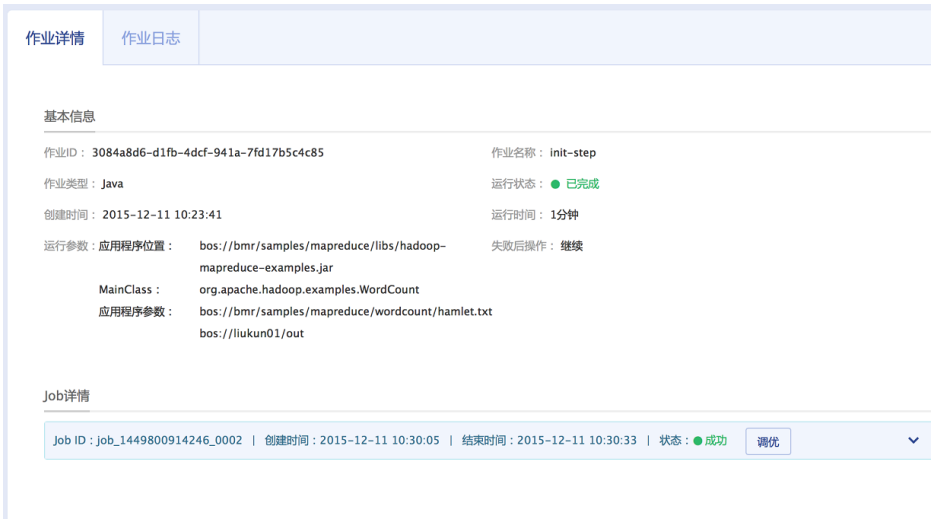
如上图所示，诊断的内容包含了用户的配置或者程序出错的信息，以及给出的建议。其中，如果出错的原因是在您的程序中，诊断功能还会扫描用户的代码中的异常栈或错误栈，剔除框架中可能对您定位问题无用的错误信息，直接定位到您程序中错误的代码。如果您想要了解更加具体的错误，还可以查看bos中的错误文件信息。

作业 / Jobs / Diagnosis

问题名称	数量	原因	建议
越界错误	1	具体的原因: Caused by: java.lang.ArrayIndexOutOfBoundsException: 2: at com.baidu.inf.WordCount$.main(WordCount.scala:26) at com.baidu.inf.WordCount.main(WordCount.scala)	存在越界错误，检查程序中的数据获取

查看成功作业的调优信息

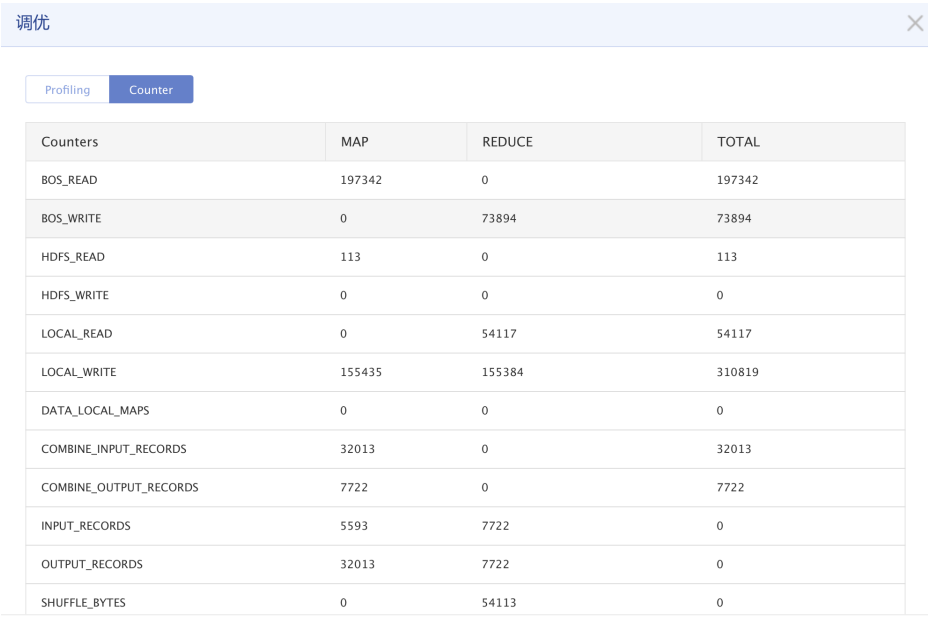
1. 在集群作业页面，点击已完成作业的作业名称可查看该作业详情。



2. 在作业详情页面，点击“调优”，进入调优页面。

在集群中预设的参数是针对大多数作业的调优权衡的结果，不同作业的最优参数也不同，BMR调优会根据您作业的信息给予合理的作业参数调整。在BMR中内置了Map/Reduce任务数均衡性、内存溢写检查、Reduce分桶均匀度等十多项调优规则，在作业运行过程中收集集群环境数据和作业运行的信息，在作业结束时根据收集到的信息计算这十多种规则的评分情况，一旦超过阈值则给予警告，列出未通过的原因以及调优的方向。未通过的原因通常是因为作业参数设置不合理，或者套餐选择不合理，BMR的调优会自动收集您设置的配置参数和系统参数，对比不同参数对作业运行的影响并给予建议。例如在Map/Reduce任务数均衡性的检查中，BMR会权衡每个Map和Reduce任务所处理的数据量、任务处理时间，根据百度积累多年来的经验值给出合适的Map和Reduce的任务数量建议。

3. 查看counter信息。在作业结束后可查看不同counter的信息，根据这些信息，您可以更方便地了解到作业执行时的情况，以及针对这些信息对程序做出相应的改进。



定时任务

注：自2024年6月30日起，MapReduce暂不提供作业相关功能支持，可通过第三方平台EasyDAP或开源组件Airflow提交任务。

简介

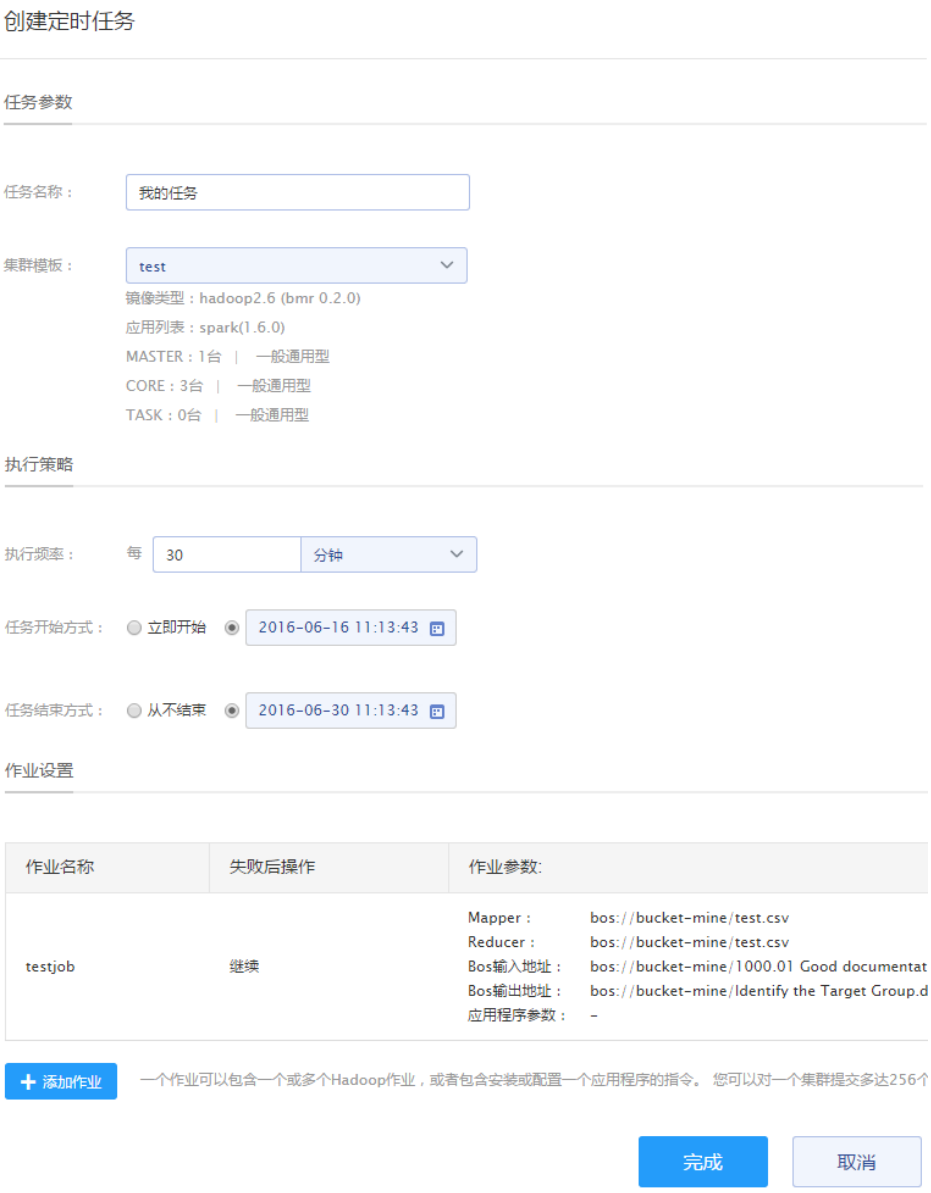
通过定时任务您可定时启动集群运行作业。需预先规划时间策略，并依据时间策略存储输入数据，再创建定时任务，并可对已创建的定时任务修改时间策略。

规划准备

1. 请预先规划时间策略，即自动启动集群运行作业的时间。本文以自2015年12月11日19点55分至2015年12月18日19点55分期间每半小时自动启动集群为例。
2. 请确定数据在BOS中的存储地址，并根据规划好的时间策略存储输入数据。依照例子中的时间策略，地址分别是bos://{bucket_name}/input/201512111955/data.txt，bos://{bucket_name}/input/201512112025/data.txt，bos://{bucket_name}/input/201512112055/data.txt，如此类推。

创建定时任务

1. 在“产品服务>MapReduce>MapReduce-定时任务”页中，点击“创建任务”，进入创建定时任务页。
2. 配置“任务参数”区：
   - 在“任务名称”栏输入任务名称。
   - 在“集群模板”栏选择设置定时任务的集群模板。
3. 配置“执行策略”区：
   - 在“执行频率”栏，可设置间隔的分钟数、小时数和天数。其中，最小间隔不能小于5分钟。
   - 在“任务开始方式”栏，选择“立即开始”则即刻开始执行任务，选择“Y-M-D H:M:S”则指定开始任务的具体时间点。
   - 在“任务结束方式”栏，选择“永不结束”则任务无结束时间，选择“Y-M-D H:M:S”则指定结束任务的具体时间点。
4. 在“作业设置”区，点击“添加作业”，在弹出的对话框内参照创建作业的步骤2的信息说明设置作业参数，其中输入bos输入\输出地址时依照时间策略输入带有%Y%m%d%H%M字串的地址。依照例子中的时间策略，bos输入地址是：bos://{Bucket_name}/input/%Y%m%d%H%M/data.txt。bos输出地址是：bos://{Bucket_name}/output/%Y%m%d%H%M，作业设置完毕后请点击“确定”完成定时任务的作业添加。
5. 点击“完成”，定时任务创建完毕。



6. 系统启动集群时可根据输入/输出地址自动匹配字符串对应时间的文件夹。即2015年12月11日15:38启动集群时调用地址是bos://{bucket_name}/input/201512111538/data.txt的数据，运行结果自动输出至地址是bos://{Bucket_name}/output/201512111538文件夹内。
7. 可在“产品服务>MapReduce>MapReduce-定时任务”页中查看已创建的任务。
8. 点击已创建定时任务对应的“查看执行历史”可查看任务执行记录。
9. （可选）点击“停止”，可暂停该任务，点击“开启”，可重新启动该任务。

修改定时任务

对已创建的定时任务，您可修改时间策略和作业。具体操作如下：

1. 在“产品服务>MapReduce>MapReduce-定时任务”页中，点击已创建的定时任务，打开“定时任务详情”页。
2. 修改时间策略：在“执行详情”区，点击“调整执行策略”，在弹出的对话框中设定新的执行策略后，点击“确定”即可。
3. 修改作业：在“作业信息”区，可修改或删除已有作业，点击作业名称对应的“编辑”或“删除”按钮即可。也可添加新的作业，点击“添加作业”，在弹出的对话框内参照创建作业的步骤2的信息说明设置作业参数。

权限管理

多用户访问控制

多用户访问控制，主要用于帮助用户管理云账户下资源的访问权限，适用于企业内的不同角色，可以对不同的工作人员赋予使用产品的不同权限，当您的企业存在多用户协同操作资源时，推荐您使用多用户访问控制。

概论

适用场景

多用户访问控制，主要用于帮助用户管理云账户下资源的访问权限，适用于企业内的不同角色，可以对不同的工作人员赋予使用产品的不同权限，当您的企业存在多用户协同操作资源时，推荐您使用多用户访问控制。 适用于下列使用场景：

- 中大型企业客户：对公司内多个员工授权管理
- 偏技术型vendor或SAAS的平台商：对代理客户进行资源和权限管理
- 中小开发者或小企业：添加项目成员或协作者，进行资源管理

访问策略

多用户访问控制实现了企业或组织内部不同人员乃至不同部门协同地管理和使用BMR资源的功能。在BMR中，我们将集群和定时任务视为资源。协同工作的方式是主用户需要创建子用户，并为子用户分配访问策略。 目前访问控制策略分为权限和资源两个维度。其中，权限是指用户可执行的动作，如创建、删除，查看，修改等；资源是指集群或定时任务。权限又被分为粗粒度和细粒度两种，粗粒度权限是指不限定具体的资源的权限，一个粗粒度权限可对应多个细粒度权限；而细粒度权限需要选定某一个或多个资源。被赋予某种策略的子用户将会具有相应资源的相应访问权限。比如：

```
 ACL{ //该策略为对全部资源具有READ粗粒度权限；
"resource": ["*"], //资源
"permission": [ //权限
        "READ"
]
} ;


ACL{ //该策略为对某个集群具有ScaleCluster（扩缩容）细粒度权限；
"resource": ["cluster/ clusterID"], //资源
"permission": [ //权限
        " ScaleCluster"
]
}。
```

> 说明：
>
> - BMR所有资源均归属于主用户，子用户所创建或删除的资源均计费于主用户之下。

- 未对集群模板做权限控制，对集群模板的操作，子用户与主用户权限相同。
- 每次策略变动将在5分钟后生效。

权限模型

表一 权限模型说明

粗粒度权限名称	描述	对应细粒度权限名称	描述
READ	仅使用BMR的权限	ReadCluster	读取集群
-	-	ReadStep	查看作业详情
-	-	CreateStep	创建作业
-	-	CancelStep	终止作业
-	-	ReadExecutionPlan	读取定时任务
-	-	UpdateExecutionPlan	更新定时任务包括调整执行策略、添加、删除定时任务上的作业
LIST_ALL_STEP	拉取作业列表	无	-
CREATE_DELETE	仅创建和删除BMR资源的权限	无	-
-	-	CreateCluster	创建BMR集群
-	-	DeleteCluster	删除BMR集群
OPERATE	仅运维BMR的权限	RenameCluster	更改集群名称
-	-	ScaleCluster	变更集群配置，即扩缩容
-	-	SwitchExecutionPlan	启停定时任务
FULL_CONTROL	具有完全控制管理BMR的权限	无	-

选择一个粗粒度权限策略表示选择了该粗粒度权限对应的全部细粒度权限。从命名中可以看出，粗粒度权限名为全部大写字符，细粒度权限名为驼峰式命名。

> 说明：

- LIST_ALL_STEP粗粒度权限可访问全部作业列表，包括集群页面下作业列表；
- READ粗粒度权限，或指定集群ID的ReadStep细粒度权限仅可访问集群页面下作业列表，同时该权限可查看作业详情。

子用户操作

创建子用户

主账号用户登录后在控制台选择**多用户访问控制**进入用户管理页面。

1. 在左侧导航栏选择**用户管理**，选择**子用户**，点击**创建子用户**。

2. 在弹出的创建子用户对话框中，完成填写用户名等信息后确认，返回子用户管理列表区可以查看到刚刚创建的子用户。

子用户登录

主账号完成对子用户的授权后，可以将链接发送给子用户；子用户可以通过IAM用户登录链接登录主账号的管理控制台，根据被授权的策略对主账户资源进行操作和查看。

配置策略

权限策略有两种类型**系统权限策略**和**自定义权限策略**。可选择系统策略或自定义策略来查看权限策略列表，分别实现BMR的产品级权限和资源级权限控制。

创建策略

1.点击**创建策略**，选择策略类型（按策略生成器创建、按标签创建、按策略语法创建和按资源组创建）

表三 策略创建类型说明

策略创建类型名称	策略创建类型说明
策略生成器	选择服务、权限以及区域下实例，生成策略。 1.登录云控制台，鼠标移到右上角头像处，进入多用户访问控制>策略管理; 2.点击**创建策略**，弹窗中选择按策略生成器创建； 3.填写基本信息中的策略名称和描述； 4.填写权限配置，如需添加权限点击下方添加权限按钮，为当前策略再增加一条权限。 -选择服务：即需要选择的产品名称 -选择方式：可视化编辑或策略语法编辑
标签	根据你为服务实例创建的便签筛选资源，生成实例。 1.登录云控制台，鼠标移到右上角头像处，进入多用户访问控制>策略管理; 2.点击**创建策略**，弹窗中选择按标签创建； 3.填写基本信息中的策略名称和说明； 4.填写权限配置： -选择标签：选择你需要的标签键值对，如果没有标签，可以选择点击还没有标签？点击创建标签链接跳转到标签管理； -选择服务：选择已支持标签的服务类型，查看哪些产品线已支持基于标签的授权，具体可参考使用IAM的产品； -选择操作：已选服务的权限操作，统一为只读、运维和管理权限； -资源范围：展示已选择服务的资源列表，如未命中任何实际资源，则代表全局所有资源，在未来的时间范围内，如果你将当前标签关联到实际的资源实例，该资源实例将受到当前自定义策略的控制。
策略语法	编写策略语法，生成策略。 1. 登录云控制台，鼠标移到右上角头像处，进入**多用户访问控制>策略管理**; 2. 点击**创建策略**，弹窗中选择按策略语法创建； 3. 填写基本信息中的策略名称和说明； 4. 填写权限配置： -策略模板：选择已有系统策略模板 -策略内容：输入策略内容后点击完成
资源组	根据你在资源管理创建的资源分组筛选资源，生成策略。 1. 登录云控制台，鼠标移到右上角头像处，进入**多用户访问控制>策略管理**; 2. 点击**创建策略**，弹窗中选择按资源组创建； 3. 填写基本信息中的策略名称和说明； 4. 填写权限配置后确定创建成功。

用户授权

在用户管理->子用户管理列表页的对应子用户的操作列选择添加权限，并为用户选择系统权限或自定义策略进行授权。

> 说明： 如果在不修改已有策略规则的情况下修改某子用户的权限，只能通过删除已有的策略并添加新的策略来实现，不能取消勾选已经添加过的策略权限。

相关服务授权

为使子用户能够正常使用BMR服务，还需要为其授予百度智能云内其他相关服务的权限，相关服务的权限策略及影响范围如下：

表四 相关服务权限策略及影响范围

相关服务策略名称	描述	影响范围
FCOrderAccessPolicy	完全控制订单服务权限	查看集群和创建集群
VpcFullControlPolicy	完全控制VPC服务权限	创建集群
SubnetFullControlPolicy	完全控制子网服务权限	创建集群
BosFullAccessPolicy	完全控制BOS服务权限	创建集群时，选择将日志存储在BOS中
SecurityGroupFullController	完全控制安全组服务权限	创建集群时，选择将日志存储在BOS中

用户管理

实际业务处理场景中，经常遇到需要将BMR集群资源及集群中的数据、服务等在多个组织及部门间共享的场景，需要能够做到集群中细粒度的计算资源及访问权限管控。 对于云上资源（集群、作业）粒度的管理，请参见多用户访问控制。 用户管理功能提供了BMR集群中从账号、认证、授权、审计四个纬度的统一安全方案，满足企业级细粒度多租户及安全管控需求。并且支持与IAM账号做关联打通，易于使用。

新增用户

1.在集群列表中点击**集群名称**进入集群管理界面，选择侧边导航**用户管理**

2.在用户管理界面点击**添加BMR账号**按钮，根据需要填写配置项后点击确定。

管理BMR用户

1.在用户列表找到需要编辑的用户，点击操作列**修改**按钮

2.进入编辑界面，按要求输入密码并在此确认后点击确定即可编辑成功。

- 点击删除按钮，可以删除所选的BMR账号
- 安全模式开启时可以点击**下载keytab**，下载该账号对应的keytab，用作集群中提交作业时使用。

集群模板

操作步骤

1. 在**产品服务-MapReduce-集群模板**页中，点击**创建模板**，进入创建模板页。
2. 完成"选择集群配置"、"实例组配置"，具体操作请参考创建自定义集群的步骤。
3. 点击**完成**，模板创建成功。可在集群模板页查看已创建的模板。
4. 在集群模板界面选择已创建的模版，在操作列选择**创建集群**。

保存集群为模板

1. 在MapReduce集群列表中，选择对应的集群点击操作列**保存为模版**即可保存集群为模板。
2. 查看保存的模板可以在侧边导航选择集群模板查看。

实践操作

导入数据

Sqoop导入数据

在使用BMR添加作业之前，用户需要将被分析的数据上传到BOS中，具体操作请参考BOS上传Object。

您可通过Sqoop把关系型数据库RDS中的数据导入到BOS、HDFS、HBase或Hive中。具体操作如下：

从RDS关系型数据库导入数据至BOS中

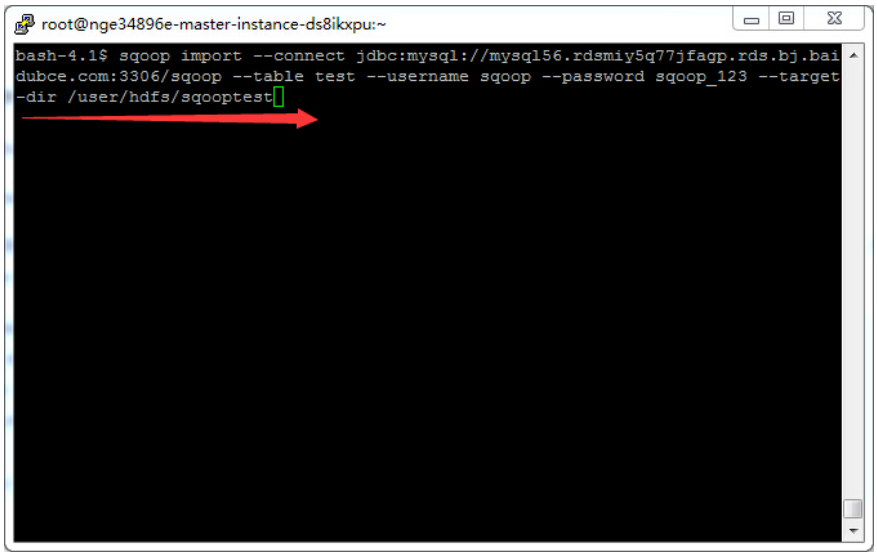
1. 通过SSH连接到主节点，请参考SSH连接到集群。
2. 输入命令：su hdfs。切换到HDFS用户。
3. 执行以下格式的命令： sqoop import --connect jdbc:mysql://address:port/db_name --table table_name --username XX --password XX --target-dir XX

   示例：sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --target-dir bos://abc/sqooptest
4. 执行后，可至BOS中查看执行结果。BOS上查看执行结果的示例如下：

从RDS关系型数据库导入数据至HDFS中

1. 执行从RDS关系型数据库导入数据至BOS中的步骤1至2。
2. 执行以下格式的命令： sqoop import --connect jdbc:mysql://address:port/db_name --table table_name --username XX --password XX --target-dir XX

   示例：sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --target-dir /user/hdfs/sqooptest
3. 执行后，可至HDFS中查看执行结果。



参数	参数说明
address和port	RDS数据库实例的地址和端口号。请至RDS实例的基本信息中获取，可参考连接RDS实例
db_name	需导入数据所在数据库的名称。如需创建关系型数据库RDS实例，请参考创建数据库。
table_name	需导入数据的数据表的名称。如需创建数据表，请先登录到关系型数据库RDS实例中创建，请参考创建RDS实例。
--username和--password	需导入数据所在数据库的账号和密码。请至RDS实例中获取信息，请参考创建账号。
--target-dir	数据导入的目的地址，即BOS或HDFS的路径。 注意：若指定BOS路径，则路径中所指定的目录不能在bos上存在，例如，导入路径bos://test/sqooptest中的sqooptest目录在bos上必须不存在。

从RDS关系型数据库导入数据至HBase中

1. 通过SSH连接到主节点，请参考SSH连接到集群。
2. 输入命令：su hdfs。切换到HDFS用户。
3. Sqoop导入数据到HBase有两种情况:
   - 导入数据到HBase已经存在的表中，执行以下格式的命令： sqoop import --connect jdbc:mysql://address:port/db_name --table table_name --username XX --password XX --hbase-table hbase_table_name --column-family hbase_column_family --hbase-row-key row_key。 示例： sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --hbase-table sqoop_hbase -- column-family message1 --habse-row-key id。
   - 导入数据到Hbase中不存在的表中，执行以下格式的命令： sqoop import --connect jdbc:mysql://address:port/db_name --table table_name --username XX --password XX --hbase-table hbase_table_name --column-family hbase_column_family --hbase-row-key row_key --hbase-create-table。 示例： sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --hbase-table sqoop_hbase -- column-family message1 --habse-row-key id --hbase-create-table。
4. 执行后，可至HBase中查看执行结果，执行结果示例如下:

```
 hbase(main):003:0>scan "sqoop_hbase"
ROW  COLUMN+CELL
 1  column=message1:address,timestamp=1439794756361,value=shanghai
 1  column=message1:favor,timestamp=1439794756361,value=lanqiu
 1  column=message1:name,timestamp=1439794756361,value=xiaoming
 1  column=message1:sex,timestamp=1439794756361,value=1
 1  column=message1:address,timestamp=1439794756361,value=beijing
 1  column=message1:favor,timestamp=1439794756361,value=pingpong
 1  column=message1:name,timestamp=1439794756361,value=xiaohong
 1  column=message1:sex,timestamp=1439794756361,value=2
```

参数	参数说明
address和port	RDS数据库实例的地址和端口号。请至RDS实例的基本信息中获取，可参考连接RDS实例
db_name	需导入数据所在数据库的名称。如需创建关系型数据库RDS实例，请参考创建数据库。
table_name	需导入数据的数据表的名称。如需创建数据表，请先登录到关系型数据库RDS实例中创建，请参考创建RDS实例。
--username和--password	需导入数据所在数据库的账号和密码。请至RDS实例中获取信息，请参考创建账号。
--hbase-table	数据导入的目的数据表的表名，即HBase中数据表的表名。
--column-family	目的数据表的列簇，即HBase中数据表的列簇。 注意：sqoop一次只能指定一个列簇。
--hbase-row-key	数据源所在RDS数据表中作为hbase row key的字段。

从RDS关系型数据库导入数据至Hive中

1. 通过SSH连接到主节点，请参考SSH连接到集群。
2. 输入命令：su hdfs。切换到HDFS用户。
3. Sqoop导入数据到Hive有三种情况:
   - Hive中不存在与关系型数据库RDS同名的数据表，执行以下格式的命令： sqoop import --connect jdbc:mysql://address:port/db_name --table table_name --username XX --password XX --hive-import 示例： sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --hive-import
   - 默认情况下，sqoop会将hive中的表名设定为导入数据的数据表名，如不想使用默认表名，使用--hive-table指定新数据表，执行以下格式的命令： sqoop import --connect jdbc:mysql://address:port/db_name --table table_name --username XX --password XX --hive-import --hive-table XX 示例： sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --hive-import --hive-table sqoop_hive
   - Hive中存在与关系型数据库RDS同名的数据表，需要通过--hive-table指定数据表，并且加上--hive-overwrite参数，执行以下格式的命令： sqoop import --connect jdbc:mysql://address:port/db_name --table table_name --username XX --password XX --hive-import --hive-table XX --hive-overwrite 使用RDS数据库中的另一张表test_export，指定hive中已经存在的表sqoop_hive，示例： sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test_export --username sqoop --password sqoop_123 --hive-import --hive-table sqoop_hive -- hive-overwirte。
4. 执行后，可至Hive中查看执行结果，执行结果示例如下：

```
 hive>select * from test;
OK
1 xiaoming 1 shanghai lanqiu
2 xiaohong 2 beijing  pingpong
```

参数	参数说明
address和port	RDS数据库实例的地址和端口号。请至RDS实例的基本信息中获取，可参考连接RDS实例
db_name	需导入数据所在数据库的名称。如需创建关系型数据库RDS实例，请参考创建数据库。
table_name	需导入数据的数据表的名称。如需创建数据表，请先登录到关系型数据库RDS实例中创建，请参考创建RDS实例。
--username和--password	需导入数据所在数据库的账号和密码。请至RDS实例中获取信息，请参考创建账号。
--hive-import	导入数据至hive中。
--hive-table	数据导入的目的数据表的表名，即Hive中数据表的表名。
--hive-overwrite	覆盖Hive中与关系型数据库RDS同名的数据表。注意：如果将非INT类型转换为INT类型，导入数据可能不正确。

存储数据至HBase

准备

使用HBase存储数据需正确安装HBase，操作详情如下：

1. 安装HBase。请在创建集群时安装HBase，具体操作请参考创建集群。HBase安装成功后，HBase的Master部署在BMR的Master节点，Region Server部署在BMR的Core节点。
2. 登录HBase Master节点。即连接到BMR集群的Master节点，具体操作请参考SSH连接到集群。
3. 登录HBase Master节点后，通过SSH登录BMR的Core节点：Core节点的内网地址作为SSH连接的IP地址，端口号请至站内信获取。

读写数据

可通过以下步骤完成读写数据操作：

1. 连接HBase。登录HBase Master节点，执行命令：hbase shell。

2. 建表。

   示例：若建立一个名为'test'的表，这个表包含一个名为'family'的列族，请执行命令：hbase(main):001:0> create 'test', 'family'。

3. 列出所有表。执行命令：hbase(main):001:0> list。

4. 写入数据。

   示例：若设置行'row1'、列' family:col1'对应的数据为'val1'，请执行命令：hbase(main):001:0> put 'test', 'row1', 'family:col1', 'val1'

5. 读取数据。

   示例：

   - 若读取行'row1'的全部数据，请执行命令：hbase(main):001:0> get 'test', 'row1'
   - 若读取行'row1'、列'family:col1'对应的数据，请执行命令:hbase(main):001:0> get 'test', 'row1', 'family:col1'

配置HBase

1. 登录HBase的Master节点。
2. 在Master节点的/etc/hbase/conf目录下，找到配置文件hbase-sit.xml和配置JVM和GC参数的hbase-env.sh脚本，修改相应的参数并保存配置文件。
3. 修改完配置文件后，请重启HBase Master节点和Region Server使配置生效。
   - Master的重启命令如下：su hbase -c '/opt/bmr/hbase/bin/hbase-daemon.sh --config /etc/hbase/conf restart master'
   - Region Server的重启命令如下：su hbase -c '/opt/bmr/hbase/bin/hbase-daemon.sh --config /etc/hbase/conf restart regionserver'

使用Hive访问HBase

应用场景

在BMR中，用户可以结合HBase使用Hive。使用Hive访问HBase时，Hive和HBase可以在同一集群上，也可以分别存在于不同集群上。通常建议用户将HBase和Hive运行于不同集群中，因为这样可以让HBase更充分的利用集群资源。

将Hive连接到HBase后，可以通过在Hive中创建外部表对HBase中存储的数据进行访问。

> **说明：** 用户只能将Hive集群连接到单个的HBase集群。

操作步骤

1. Hive连接到HBase。
   1. 通过控制台创建BMR集群，并添加Hive应用。请参考创建集群。
   2. 通过SSH连接到主节点。请参考SSH连接到集群。
   3. 复制HBase集群的`/etc/hosts`内容至Hive集群的`/etc/hosts`中。
   4. 执行命令`hive`启动Hive程序。
   5. 连接Hive集群上的HBase客户端与包含数据的HBase集群。执行命令`set hbase.zookeeper.quorum=hbase_cluster_master_ip`。

      > **说明：**
      > - `hbase_cluster_master_ip`为Hbase服务所在集群的主节点内网IP，Hive和Hbase必须是同一用户的两个集群。
      > - 若Hive和Hbase在同一集群中，此步骤可省略。

2. Hive访问HBase数据。 从Hive提示符中运行时，以下示例创建了一个外部表，此表引用了存储在名为`hive_hbase`的HBase表上的数据。然后，您可直接引用Hive语句中的`pagecounts_hbase`，查询和修改存储在HBase集群上的数据。

```
 CREATE EXTERNAL TABLE IF NOT EXISTS pagecounts_hbase (rowkey STRING, pageviews STRING, bytes STRING)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
   WITH SERDEPROPERTIES ('hbase.columns.mapping' = ':key,f:c1,f:c2')
    TBLPROPERTIES ('hbase.table.name' = 'hive_hbase');

select count(*) from pagecounts_hbase;
```

查看HBase日志

请在HBase Master和Region server节点的/var/log/hbase目录下查看HBase日志。

编译Maven项目

Maven项目包样例

百度智能云提供了以下组件的Maven项目样例代码，您可通过GitHub克隆代码至本地设计自己的程序：MapReduce

Linux环境下使用命令行编译Maven项目

以Ubuntu 14.04环境为例，介绍Maven的安装和编译。

1. 安装JDK。

   1). Maven依赖Java运行环境，因此使用Maven之前需要确认正确安装JDK1.4及以上的版本。执行命令：`sudo apt-get install openjdk-7-jdk`。

   2). 设置JAVA_HOME环境变量：`export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64/`。

   3). 验证JDK安装正确：java -version。若显示如下内容则安装正确：

   ```
   java version "1.7.0_60"
   Java(TM) SE Runtime Environment (build 1.7.0_60-b19)
   Java HotSpot(TM) 64-Bit Server VM (build 24.60-b09, mixed mode)
   ```

2. 安装Maven。

   1). 下载Maven并安装：sudo apt-get install maven。 2). 验证Maven安装正确：mvn -version。若显示如下内容则安装正确：

   ```
   Apache Maven 3.0.5
   Maven home: /usr/share/maven
   Java version: 1.7.0_95, vendor: Oracle Corporation
   Java home: /usr/lib/jvm/java-7-openjdk-amd64/jre
   Default locale: en_US, platform encoding: UTF-8
   OS name: "linux", version: "3.13.0-32-generic", arch: "amd64", family: "unix"
   ```

3. 本地安装Git:`sudo apt-get install git`。

4. 编译Maven项目生成jar文件。本文以编译百度智能云提供的样例Maven项目包MapReduce为例。

   1). 通过GitHub克隆Maven项目包至本地：`git clone https://github.com/BCEBIGDATA/bmr-sample-java.git`。

   2). cd至源文件所在目录，即“/{yourPath}/bmr-sample-java-master/mapreduce”。

   3). 执行编译命令mvn clean install生成jar包，生成的jar文件会存放在Maven项目包所在目录的target文件夹中，即“/{yourPath}/bmr-sample-java-master/bmr-sample-java-master/mapreduce/target/mapreduce-1.0-SNAPSHOT.jar”。

Windows环境下使用命令行编译Maven项目

介绍Maven的安装和编译。

1. 安装JDK。

   1). Maven依赖Java运行环境，因此使用Maven之前需要确认正确安装JDK1.4及以上的版本，请阅读Maven官网的bin包对应的Java版本并下载安装JDK。本示例中采用JDK1.7版本，本地安装路径是C:\Program Files\Java\jdk1.7.0_79。下载地址：http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html。

   2). 设置环境变量。打开“计算机>属性>高级系统设置>环境变量”，在“系统变量”栏点击“新建”，Java环境变量：

   - 变量名：JAVA_HOME
   - 变量值：C:\Program Files\Java\jdk1.7.0_79

2. 安装Maven。

   1). Maven官方网站下载Maven的bin。

   2). Maven压缩包解压至本地。本例解压至D:/Maven，目录结构如下：

   ```
   D:\Maven的目录
   |-- bin
   |-- conf
   |-- core
   |-- lib
   |-- local
   ```

   3). 设置环境变量。打开“计算机>属性>高级系统设置>环境变量”，在“系统变量”栏找到“PATH”点击“编辑”，在变量值的最后输入“;%MAVEN_HOME%\bin”。点击“新建”，新建Maven环境变量：

   - 变量名：MAVEN，变量值：%MAVEN_HOME%\bin
   - 变量名：MAVEN_HOME，变量值：D:/Maven

     4). 在本机DOS环境下输入：mvn -v，若显示如下内容，则安装正确。

```
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-17+08:00)
Maven home: D:\Maven204\bin\..
Java version: 1.7.0_79, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.7.0_79\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"
```

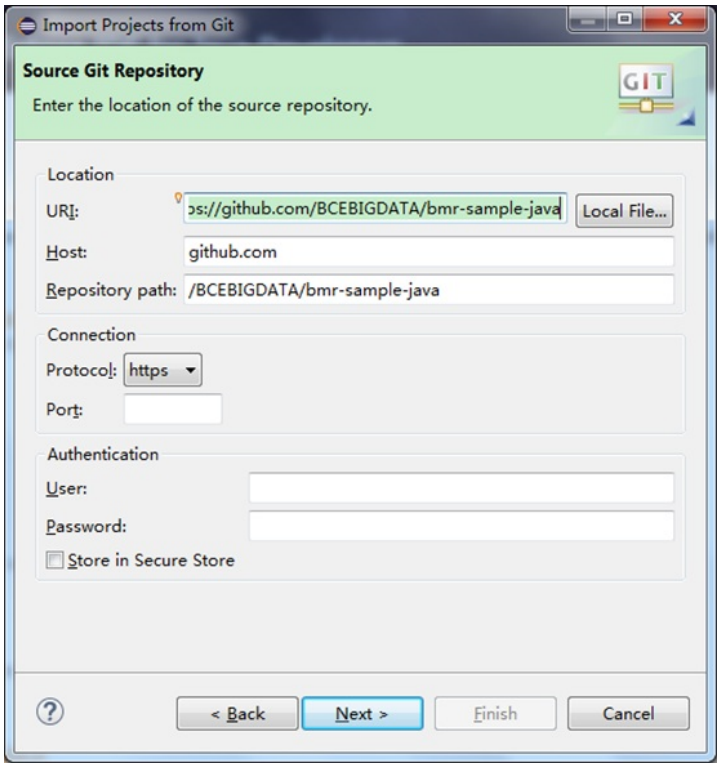
3. 编译Maven项目生成jar文件。本文以编译样例Maven项目包MapReduce为例。

1). 下载百度智能云提供的示例代码至本地。浏览器中打开https://github.com/BCEBIGDATA/bmr-sample-java。点击“Clone or download>Download ZIP”，下载至本地后解压。 2). 本机DOS环境下cd至源文件所在目录，即“{yourPath}\bmr-sample-java-master\mapreduce”。 3). 执行编译命令mvn clean install。 4). 生成的jar文件会存放在Maven项目包所在目录的target文件夹中，即“{yourPath}\bmr-sample-java-master\bmr-sample-java-master\mapreduce\target\mapreduce-1.0-SNAPSHOT.jar”。

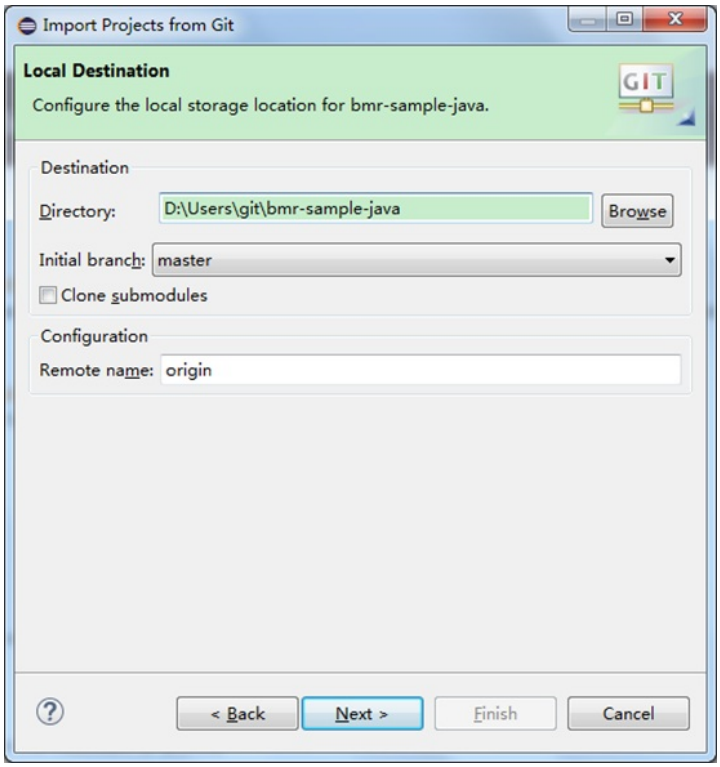
使用Eclipse编译Maven项目

本文介绍使用Eclipse编译百度智能云提供的样例Maven项目包，项目包存放地址：https://github.com/BCEBIGDATA/bmr-sample-java。具体操作如下：

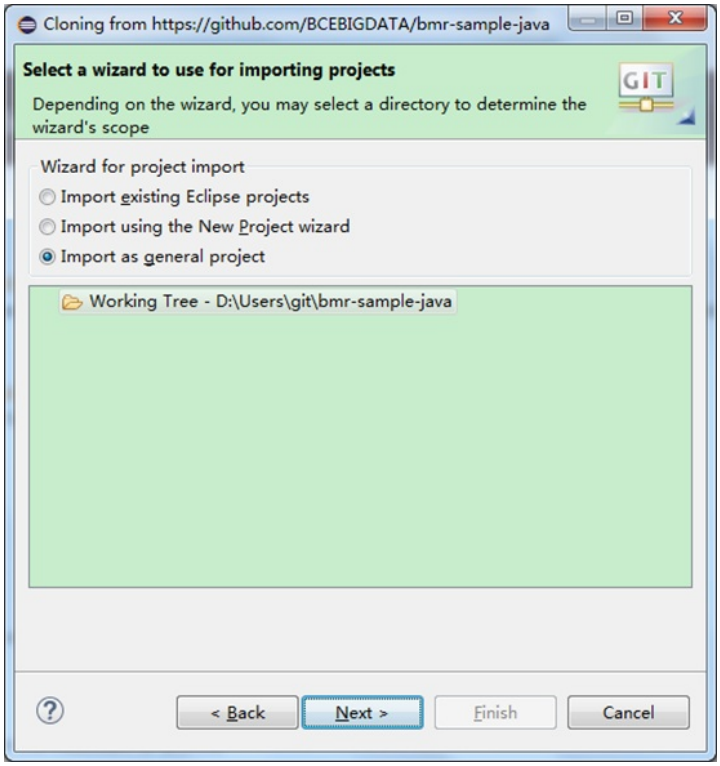
1. 官网下载Eclipse。本文以Eclipse Neon.1a Release (4.6.1)为例。
2. 打开Eclipse，菜单栏选择“Window>Preferences>Java>Install JRE”，确保勾选了系统安装的JDK。
3. 导入百度智能云提供的示例项目。
    1. Eclipse的菜单栏选择“File>Import”，选择“Git>Projects from Git”，点击“Next”。
    2. 选择“Clone URI”，点击“Next”。
    3. 在URI栏中输入需克隆的Git地址：https://github.com/BCEBIGDATA/bmr-sample-java。点击“Next”。



    4. 保持默认配置即创建本地仓库master，点击“Next”即可。
    5. Directory栏输入本地仓库的存储地址“..{yourPath}\bmr-sample-java”，点击“Next”。



6. 选择“Import as general project”后，系统会自动识别本地仓库地址“..{yourPath}\bmr-sample-java”，点击“Next”。



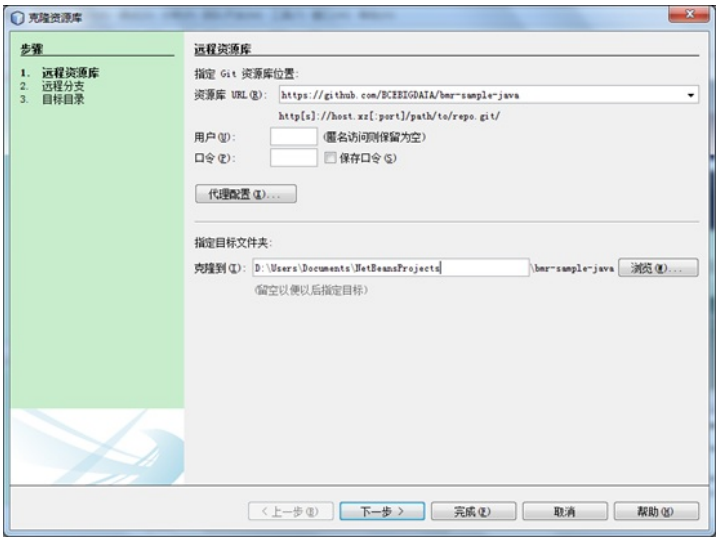
7. 保持默认的项目名称“bmr-sample-java”，点击“Finish”，即可完成项目导入。

4. 编译Maven项目输出jar文件。选择导入的Maven项目“mapreduce>pom.xml”后，菜单栏打开“Run>Run As>Maven install"即启动编译。
5. 编译成功后，生成的jar文件会存放在本地Maven项目所在目录的target文件夹中，即“{yourPath}\bmr-sample-java\mapreduce\target\mapreduce-1.0-SNAPSHOT.jar”。

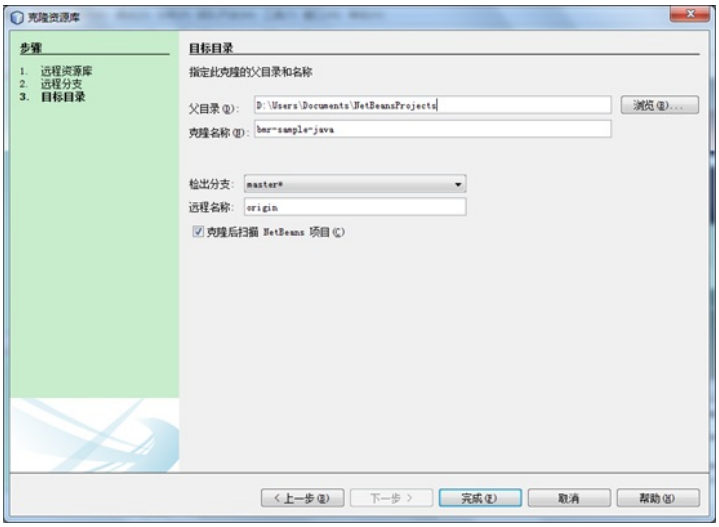
使用Netbeans编译Maven项目

1. 下载并安装Netbeans6.7或以上版本，其中已经集成了Maven。
2. 导入百度智能云提供的示例项目。

   1). 打开Netbeans，菜单栏选择“团队开发（M）>Git(G)>克隆”，在资源库URL栏输入百度智能云提供的示例地址：https://github.com/BCEBIGDATA/bmr-sample-java。并指定本地仓库地址“..{yourPath}\bmr-sample-java”，点击“下一步”。



2). 保持默认配置即创建本地仓库master，点击“下一步”。 3). 保持默认配置，点击“完成”即可。



4). 系统启动代码克隆，完成后请在弹出的对话框中选择“打开项目”，即可在左侧项目栏中查看已导入的项目。

3. 编译Maven项目输出jar文件。

1). 打开mapreduce模块。选择左侧项目栏中的“bmr>模块>mapreduce”，右键选择“打开项目”，即可打开Maven项目mapreduce。

2). 选择已打开的Maven项目mapreduce，点击菜单栏中的“运行>清理并构建项目”。

4. 编译成功后，生成的jar文件会存放在本地Maven项目所在目录的target文件夹中，即“{yourPath}\bmr-sample-java\mapreduce\target\mapreduce-1.0-SNAPSHOT.jar”。

Sqoop导入导出数据

导出数据

Sqoop导出数据

在使用BMR对数据进行分析之后，分析的结果数据被保存在BOS的指定目录下，用户可以进入BOS中将结果数据导出，具体操作请参考BOS下载Object。

您可通过Sqoop把BOS或HDFS的数据导出至关系型数据库RDS中。具体操作如下：

从BOS中导出数据至RDS关系型数据库

1. 在关系型数据库RDS中创建相应的数据表，请注意数据表字段类型与导出数据需要一致，否则在导出过程中可能出现异常。数据需要连接到RDS数据库进行创建，请参考连接RDS实例。
2. 通过SSH连接到主节点，请参考SSH连接到集群。
3. 输入命令：su hdfs。切换到HDFS用户。
4. 执行命令： sqoop export --connect jdbc:mysql://address:port/db_name --table export_table_name --username XX --password XX --export-dir XX 示例： sqoop export --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --export-dir bos://abc/sqooptest
5. 执行后，在RDS关系数据库PHP Admin界面可以看到执行结果，如下图所示：

从HDFS中导出数据至RDS关系型数据库

1. 执行从BOS中导出数据至RDS关系型数据库的步骤1至3。
2. 执行命令： sqoop export --connect jdbc:mysql://address:port/db_name --table export_table_name --username XX --password XX --export-dir XX 示例：sqoop import --connect jdbc:mysql://mysql56.rdsmiy5q77jfaqp.rds.bj.baidubce.com:3306/sqoop --table test --username sqoop --password sqoop_123 --export-dir /user/hdfs/sqooptest
3. 执行从BOS中导出数据至RDS关系型数据库的步骤5。

参数	参数说明
address和port	RDS数据库实例的地址和端口号。请至RDS实例的基本信息中获取，可参考连接RDS实例。
db_name	需导出数据所在数据库的名称。如需创建关系型数据库RDS实例，请参考创建数据库。
table_name	需导出数据的数据表的名称。如需创建数据表，请先登录到关系型数据库RDS实例中创建，请参考连接RDS实例。
--username和--password	需导出数据所在数据库的账号和密码。请至RDS实例中获取信息，请参考创建数据库账号。
--export-dir	数据导出的目的地址，即BOS或HDFS的路径。

场景教程

定时分析日志数据

概览

通过定时任务创建BMR集群，分析日志数据，定时释放集群，为用户大大节约了使用成本。

需求场景

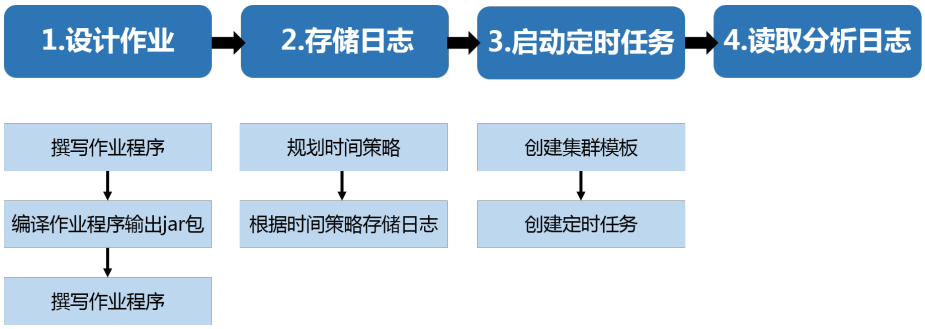
对于业务稳定且有规律的用户，日志的峰值和低谷的规律一般是固定的。对于有规律的日志业务场景，用户只需要在特定的时间段内用集群进行分析即可，其余时间无需使用集群。传统的大数据集群一旦构建则无法释放或者需要人工手动释放，使用成本较高。解决这一场景下的日志分析十分必要。

方案概述

通过定时任务您可定时启动集群运行作业，适用于对已有数据有规律的定时分析并获取结果。优势如下：

- 省时：任务只需一次创建，一键启停。
- 省钱：集群随任务的启停而启停。
- 省力：托管式集群，无需部署和运维。

本文通过定时启动集群运行MapReduce作业分析网站日志以统计每天的访问量，介绍定时任务的实现过程：



示例日志

示例日志是Nginx日志，存储在对象存储服务BOS的公共可读的路径中：

- 存储在“华北-北京”区域的样例数据仅华北区域的BMR集群可用，路径如下：
  - bos://datamart-bj/access-log/201701102000/access.log
  - bos://datamart-bj/access-log/201701112000/access.log
  - bos://datamart-bj/access-log/201701122000/access.log
  - bos://datamart-bj/access-log/201701132000/access.log
  - bos://datamart-bj/access-log/201701142000/access.log
- 存储在“华南-广州”区域的样例数据仅华南区域的BMR集群可用，路径如下：
  - bos://datamart-gz/access-log/201701102000/access.log
  - bos://datamart-gz/access-log/201701112000/access.log
  - bos://datamart-gz/access-log/201701122000/access.log
  - bos://datamart-gz/access-log/201701132000/access.log
  - bos://datamart-gz/access-log/201701142000/access.log

关于百度智能云的区域说明，请参考区域选择说明。

设计作业

1. 撰写作业程序。本文使用的MapReduce样例程序的代码已上传至：https://github.com/BCEBIGDATA/bmr-sample-java，您可通过GitHub克隆代码至本地设计自己的程序。
2. 编译程序生成jar包，具体可参考编译Maven项目。
3. 上传编译生成的jar包到对象存储BOS（具体操作详见对象存储BOS入门指南）。

存储日志

1. 规划时间策略如下：自2017年1月10日至1月14日，每天20时分析前一天的日志数据。
2. 准备日志数据。您可直接使用百度智能云提供的示例日志，在熟悉定时任务后，可参考数据准备选择您自己的日志数据。

启动定时任务

创建集群模板

1. 登录控制台，选择“产品服务->MapReduce BMR”，点击“集群模板”，进入模板列表页。
2. 点击“创建模板”，在“集群基础设置”区做如下配置：
   - 集群模板：输入模板名称“timedtask”。
   - 日志：选择集群日志的存储路径。
   - 高级设置：打开“自动终止”开关。
3. 在“集群配置”区，选择镜像版本BMR 1.0.0(hadoop 2.7)，并选择模板“hadoop”。
4. 其他设置可保持默认设置，点击“完成”即可。
5. 点击已创建的集群模板，可查看模板详情如下：

创建定时任务

1. 在“产品服务->MapReduce BMR”页，点击“定时任务”，进入定时任务列表页。
2. 点击“创建定时任务”，在“任务参数”区输入任务名称，并选择已创建的集群模板“timedtask”。
3. 在“执行频率”区，设置执行频率为“每1天”，并指定开始任务时间点“2017年1月10日20:00:00”，任务结束时间点“2017年1月10日20:00:00”。
4. 在“作业设置”区，点击“添加作业”，做如下配置：
    - 作业类型：选择“java作业”。
    - 作业名称：输入“timedtaskjob”。
    - 应用程序位置：若使用您自行编译的程序，请上传程序jar包至BOS或者您本地的HDFS中，并在此输入程序路径；您也可直接使用百度智能云提供的样例程序，路径如下：
        - 华北-北京区域的BMR集群对应的样例程序路径：bos://bmr-public-bj/sample/mapreduce-1.0-SNAPSHOT.jar。
        - 华南-广州区域的BMR集群对应的样例程序路径：bos://bmr-public-gz/sample/mapreduce-1.0-SNAPSHOT.jar。
    - 失败后操作：继续。
    - MainClass：输入“com.baidu.cloud.bmr.mapreduce.AccessLogAnalyzer”。
    - 应用程序参数：指定输入数据的路径、结果输出的路径（可选BOS或HDFS），其中输出路径必须具有写权限且该路径不能已存在。以样例日志作为输入数据，BOS作为输出路径为例，输入如下：
        - 华北-北京区域的BMR集群对应的参数：bos://datamart-bj/access-log/201701102000/access.log bos://{your-bucket}/output/%Y%m%d%H%M。
        - 华南-广州区域的BMR集群对应的参数：bos://datamart-gz/access-log/201701102000/access.log bos://{your-bucket}/output/%Y%m%d%H%M。

        > 说明：
        > - 请替换{your-bucket}为您自己的bucket名字。
        > - 系统启动集群时可根据输入/输出地址自动匹配字串“%Y%m%d%H%M”对应时间的文件。即2017年11月28日20:00启动集群时调用地址是bos://datamart-bj/access-log/201701102000/access.log的数据，运行结果自动输出至地址是bos://{your-bucket}/output/201711282000/文件夹内。
5. 点击“确定”完成定时任务的作业添加。
6. 点击“完成”，定时任务创建完毕。
7. 可在“产品服务>MapReduce>MapReduce-定时任务”页中查看已创建的任务，如下图所示：

读取分析结果

2017年1月10日至1月14日，每天20时系统会自动启动集群并运行作业，作业运行结束后集群自动释放，您可至bos://{your-bucket}/output/路径下查看每次任务的执行结果。下图是第一次任务的分析结果：

相关产品

BMR（MapReduce）、对象存储（BOS）

使用Hive分析网站日志

概览

网站日志包含用户访问信息，通过日志分析我们可以了解网站的访问量、网页访问次数、网页访问人数、频繁访问时段等等，以便获取用户行为以优化网站的商业价值。由于网站每天会产生海量的日志，非常适合使用MapReduce（简称BMR）这样的托管Hadoop服务。同时，BMR集成了Hive和Hue，开发者可在浏览器中与

Hadoop集群交互，分析处理数据，完成创建数据集、执行Hive查询等操作，大大降低了使用门槛。

需求场景

网站PV/UV日志分析

WEB服务网站每天都会有大量的用户访问，相关的用户行为，访问量，访问频次以及用户行为等数据具有很大的商业价值，可以用于用户画像的构建以及用户行为的预测等。

方案概述

示例日志

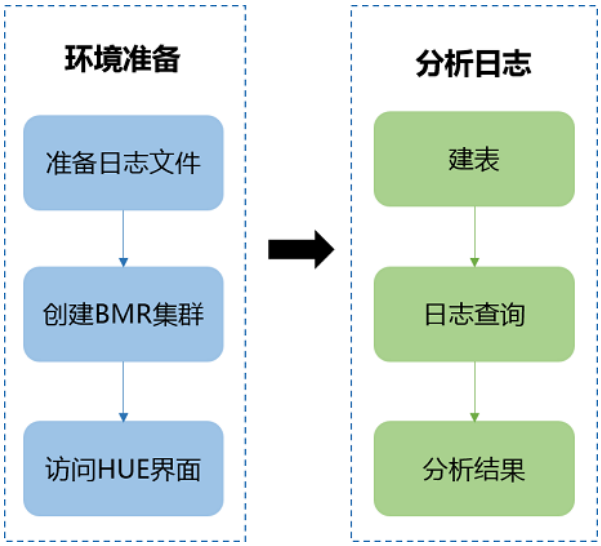
示例日志是Nginx日志，存储在对象存储服务BOS的公共可读的路径中：

- 存储在“华北-北京”区域的样例数据路径为：bos://datamart-bj/web-log-10k/，仅华北区域的BMR集群可用。
- 存储在“华南-广州”区域的样例数据路径为：bos://datamart-gz/web-log-10k/，仅华南区域的BMR集群可用。

关于百度智能云的区域说明，请参考区域选择说明。

分析过程总览

使用BMR分析Niginx日志的过程如下：



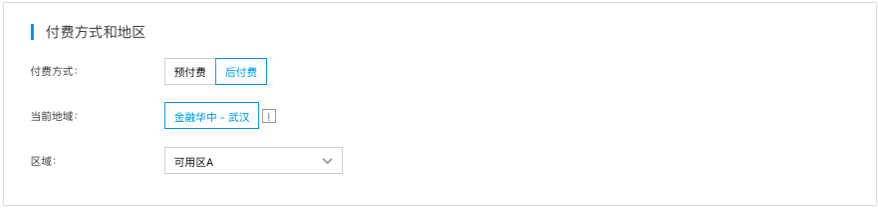
环境准备

准备日志文件

您可跳过此步直接使用百度智能云提供的示例日志。在熟悉日志分析后，可参考数据准备选择您自己的日志数据。

创建BMR集群

1. 打开“产品服务>MapReduce>MapReduce-集群列表”，点击“创建集群”，进入集群配置页面。
2. 选择付费方式和地区，可选择后付费或预付费，地域和区域（可用区）根据客户需求自行选择即可。



3. 在“集群基础设置”区选择集群类型"hadoop"，并且选择 BMR 1.2.0版本，并且勾选HUE 、Hive、Spark等需要的服务。

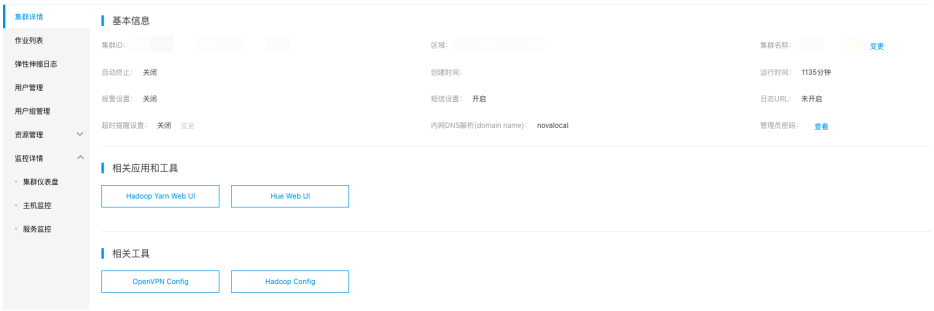
4. 用户可以对BMR集群打标签，用于区分不同集群。也可默认不填写。

5.在"基础设置"中设置集群名称，密码和网络等信息，然后点击下一步。

6. 根据用户实际计算和存储需求，选择实例类型规格，点击“提交订单”。支付订单后，集群会在五至十分钟左右创建完成。

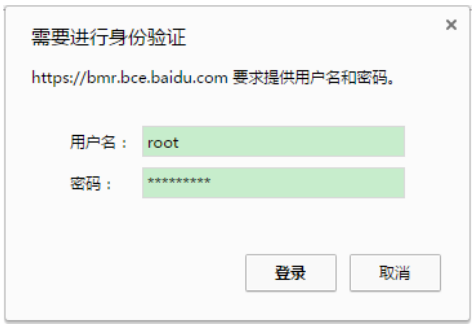
访问Hue Web界面

1. 打开“产品服务>MapReduce>MapReduce-集群列表”，点击已创建的集群，进入实例详情页面。

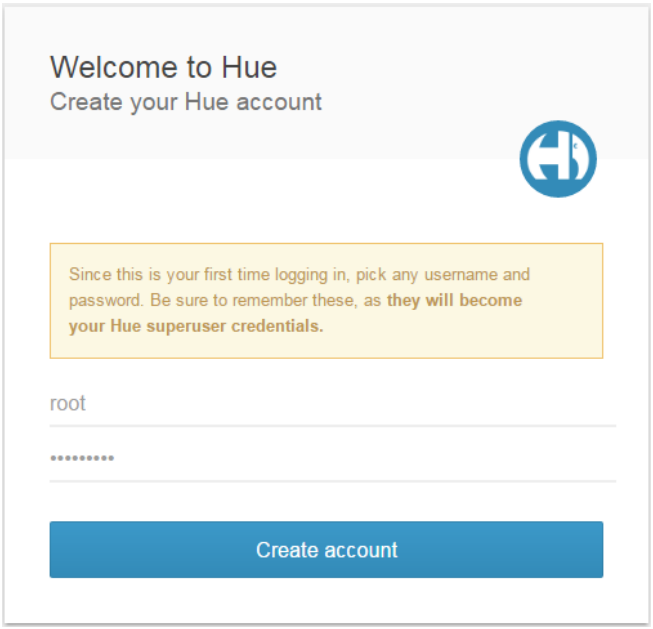


2. 在“相关应用”栏中点击“Hue Web UI”。

3. 在弹出的认证页面中输入创建集群时设置的用户名和密码，用户名默认为root，密码为创建集群时的密码，并点击“登录”。



4. 首次登录HUE WEB UI，需要创建您登录Hue服务的用户名和密码，输入后点击“Create Account”后进入Hue Web界面。



分析网站日志

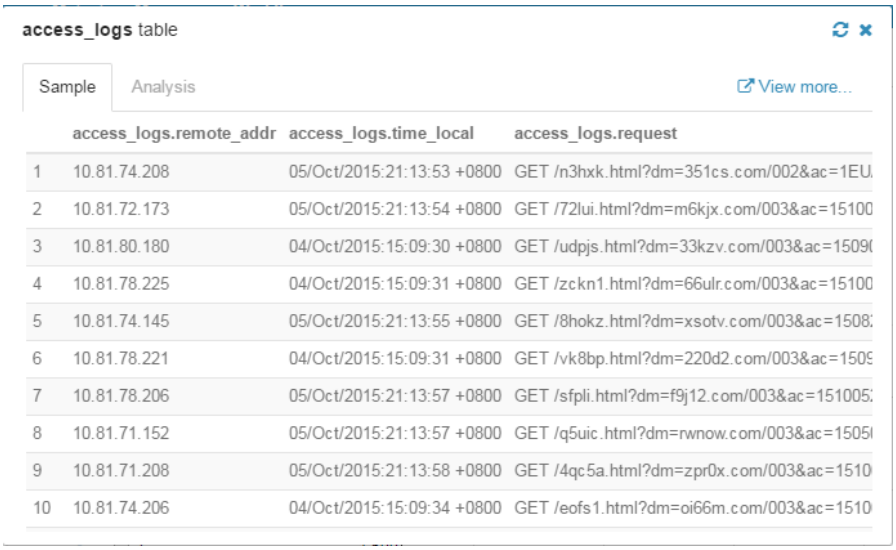
建表

1. 在分析之前，首先需要根据网站日志建立一张Hive表。在Hue菜单栏中选择“查询编辑器”>“Hive”，并输入以下SQL语句：

```
 DROP TABLE IF EXISTS access_logs;
CREATE EXTERNAL TABLE access_logs(
    remote_addr STRING comment 'client IP',
    time_local STRING comment 'access time',
    request STRING comment 'request URL',
    status STRING comment 'HTTP status',
    body_bytes_sent STRING comment 'size of response body',
    http_referer STRING comment 'referer',
    http_cookie STRING comment 'cookies',
    remote_user STRING comment 'client name',
    http_user_agent STRING comment 'client browser info',
    request_time STRING comment 'consumed time of handling request',
    host STRING comment 'server host',
    msec STRING comment 'consumed time of writing logs'
)
COMMENT 'web access logs'
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
    "input.regex" = "([0-9\\.]+) - \\[([^\\]]+)\\] \"([^\"]*)\" ([\\d]+) ([\\d]*) \"([^\"]*)\" \"([^\"]*)\" ([\\S]+) \"([^\"]*)\" ([0-9\\.]+) ([\\S]+) ([0-
9\\.]+)"
 )
STORED AS TEXTFILE
LOCATION "bos://datamart-bj/web-log-10k";
```

2. 输入语句后点击左侧的三角符号执行命令，这样，Hive会重建access_logs表，然后通过正则表达式来解析日志文件。

3. 成功创建access_logs表之后，点击Hive Editor左侧的刷新按钮，找到access_logs表并预览示例数据：



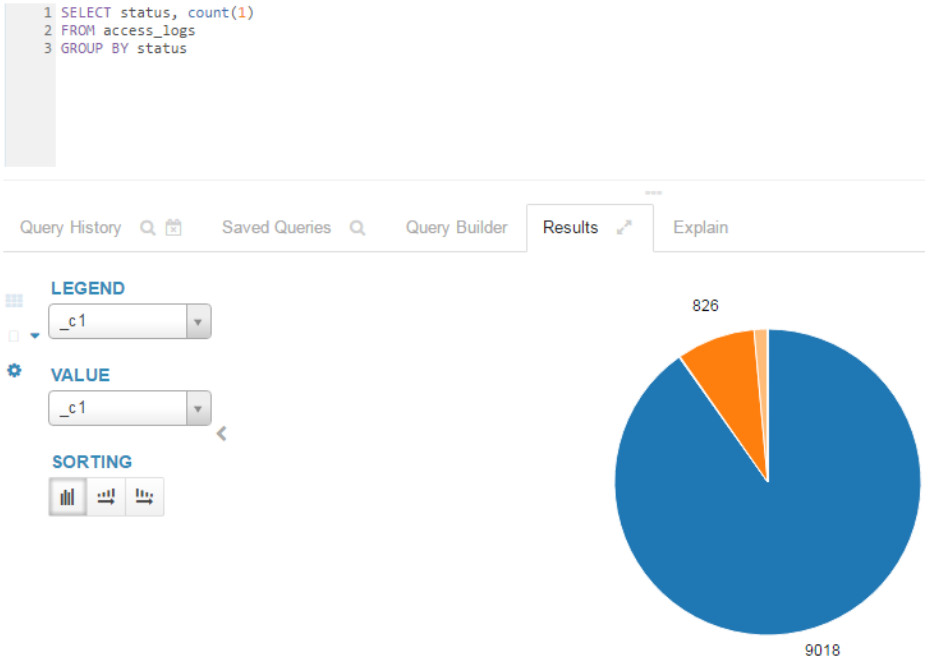
查询

定了表之后，便可以进行查询了。

- 若统计网页请求的结果，可使用以下语句：

```
 SELECT status, count(1)
FROM access_logs
GROUP BY status
```

查询结果可切换到图表页，还可以以饼图的形式可视化数据，如下图所示：

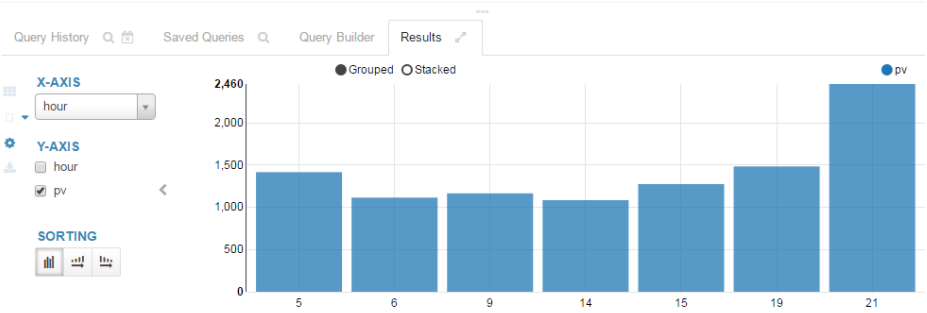


- 若想了解那个时段网页访问量最大，可使用下面的语句：

```
 SELECT hour(from_unixtime(unix_timestamp(time_local, 'dd/MMMM/yyyy:HH:mm:ss Z'))) as hour, count(1) as pv
FROM access_logs
GROUP BY hour(from_unixtime(unix_timestamp(time_local, 'dd/MMMM/yyyy:HH:mm:ss Z')))
```

查询结果可切换到图表页，或以柱状图来更直观的查看结果：

```
SELECT hour(from_unixtime(unix_timestamp(time_local, 'dd/MMMM/yyyy:HH:mm:ss Z'))) as hour, count(1) as pv
FROM access_logs
GROUP BY hour(from_unixtime(unix_timestamp(time_local, 'dd/MMMM/yyyy:HH:mm:ss Z')))
```



分析结果

网页访问量最大的时间点是晚上九点。

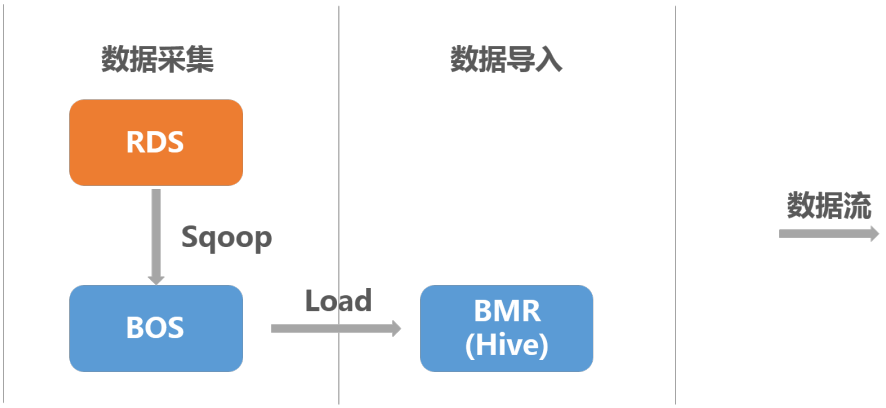
6 相关产品

BMR（MapReduce）、弹性公网IP（EIP）

Sqoop应用文档

场景描述

通过BMR Sqoop可以将RDS上的数据导入BMR Hive:



在本示例中，hive数据表的location为BOS路径，hive数据表的partition为dt（string），根据dt指定日期，区分每一天的导入数据。

说明

由于hive数据表的location为BOS，无法直接通过sqoop将RDS的数据导入hive，因为hive在加载数据时，会先将数据写入本地hdfs，然后将数据所在目录移动到hive表的location上。由于本地hdfs和BOS数据两个不同的文件系统，直接进行移动操作会抛出异常。因此，本场景需要“数据导入BOS”和“数据导入hive”两个步骤。

准备阶段

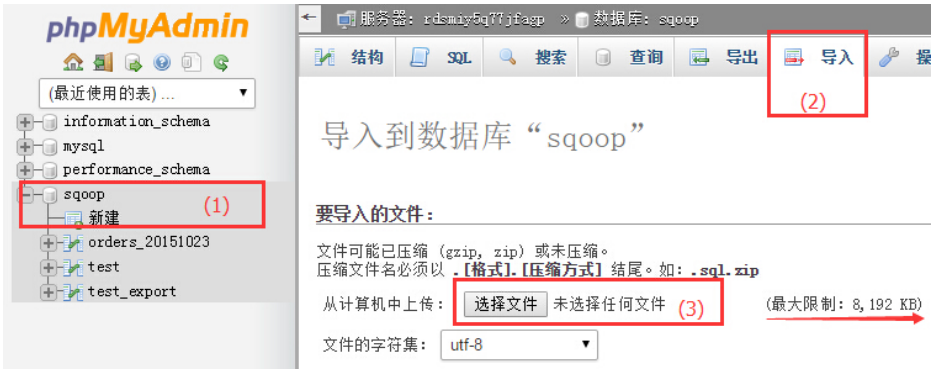
本示例所用数据为BMR Samples公共数据集。

第一步 准备数据

本示例的建表以及数据可以通过sql文件进行，下载地址。

第二步 建表并导入数据

- 关于如何登录RDS数据库，参考文档。
- 登录RDS后，选择一个数据库导入下载的sql文件，构建bmr_public_data_logs数据表，导入公共数据集。



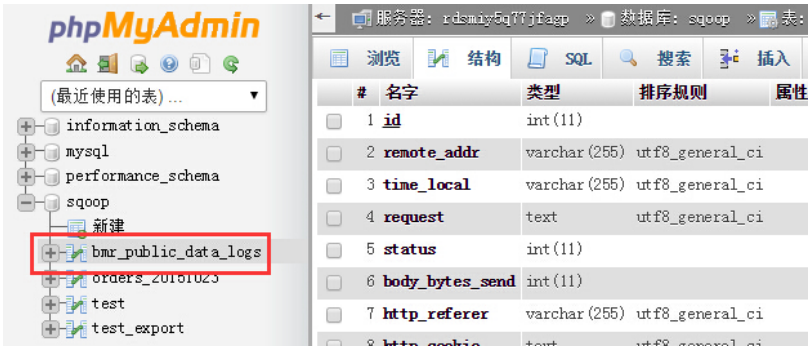
步骤如下：

1. 选中数据库sqoop；

2. 点击导入；

3. 选择下载的sql文件，注意如果是自己的sql文件，不能超过8M，超过8M可以先尝试压缩成zip文件，注意压缩文件结尾为.sql.zip。

结果如下图所示：



第三步 创建Hive数据表

登录BMR集群，切换到hdfs用户，创建hive数据表。

```
 # eip可以在BMR Console集群详情页的实例列表获取
ssh root@eip

# 切换到hdfs用户
su hdfs
cd

# 启动hive shell
hive

# 输入以下建表语句
CREATE EXTERNAL TABLE `bmr_public_data_logs`(
  `id` int,
  `remote_addr` string,
  `time_local` string,
  `request` string,
  `status` int,
  `body_bytes_send` int,
  `http_referer` string,
  `http_cookie` string,
  `remote_user` string,
  `http_user_agent` string,
  `request_time` double,
  `host` string,
  `msec` double)
PARTITIONED BY (
  `dt` string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS INPUTFORMAT
   'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
   'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
   'bos://test-sqoop-example/bmr-public-data-logs/';
```

注意

- 由于sqoop import默认的字段分隔符是','，所以在建表的时候将hive表的字段分隔设为','，即建表语句中：ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
- 将location的bos路径修改为自己的bos路径

数据导入BOS

使用Sqoop将RDS上的数据导入BOS，关于Sqoop的更多用法，参考文档。

```
 # eip可以在BMR Console集群详情页的实例列表获取
ssh root@eip

# 切换到hdfs用户
su hdfs
cd

# 使用sqoop命令将数据导入bos
sqoop import --connect jdbc//rds_mysql_hostname:port/sqoop --username test --password test --table  bmr_public_data_logs --split-by id --target-dir
bos://test-sqoop-example/bmr-log-test
```

备注：

Sqoop命令中需要替换的地方，如下图所示：

（1）RDS域名；（2）RDS数据库；（3）RDS数据库用户名；（4）RDS数据库密码；（5）BOS上的目标路径（不能在sqoop导入前存在）；

数据导入Hive

使用hive进行加载data

```
 # eip可以在BMR Console集群详情页的实例列表获取
ssh root@eip

# 切换到hdfs用户
su hdfs
cd

# 启动hive shell
hive

# 导入数据
load data inpath 'bos://test-sqoop-example/bmr-log-test' into table bmr_public_data_logs partition (dt='2015-11-02');
```

说明：

1. bos://test-sqoop-example/bmr-log-test是sqoop导入数据到bos上的目标路径。
2. 加载数据过程中，hive将sqoop导入bos上的数据移动到hive表所指定的location中，在location（bos://test-sqoop-example/bmr-public-data-logs/）中新建目录“dt=2015-11-02”，将数据移动到这个目录下，而sqoop导入到bos上的原目录已经成为空目录，可以进行删除。
3. 在加载数据过程中，可以带有overwrite，例如“load data inpath 'bos://test-sqoop-example/bmr-log-test' overwrite into table bmr_public_data_logs partition (dt='2015-11-02');”覆盖原有partition，如果不覆盖，只会将重名文件重命名，这样如果数据文件中有重复数据，则hive表中就会存在重复数据。

流式应用场景

概览

实现云上流式场景下数据流打通，方便用户在百度智能云上使用各个产品实现流式需求，实现流式数据处理全流程。

需求场景

事件流

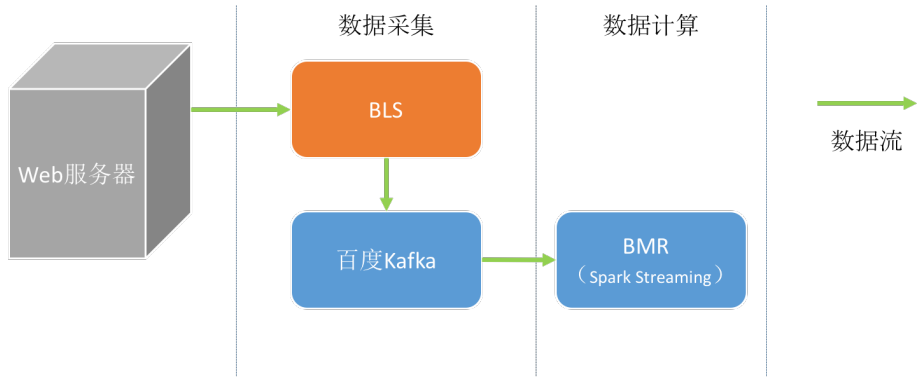
事件流具能够持续产生大量的数据，这类数据最早出现与传统的银行和股票交易领域，也在互联网监控、无线通信网等领域出现、需要以近实时的方式对更新数据流进行复杂分析如趋势分析、预测、监控等。简单来说，事件流采用的是查询保持静态，语句是固定的，数据不断变化的方式。

持续计算

比如对于大型网站的流式数据：网站的访问PV/UV、用户访问了什么内容、搜索了什么内容等，实时的数据计算和分析可以动态实时地刷新用户访问数据，展示网站实时流量的变化情况，分析每天各小时的流量和用户分布情况；比如金融行业，毫秒级延迟的需求至关重要。一些需要实时处理数据的场景也可以应用Flink/Kafka，比如根据用户行为产生的日志文件进行实时分析，对用户进行商品的实时推荐等。

方案概述

本场景应用于数据流式处理，使用到BLS（百度Log Service）、BMS（百度消息服务）以及BMR（MapReduce）三个产品。



整个流程分为数据采集和数据计算两部分。

数据采集

数据采集过程通过BLS以及百度消息服务BMS实现。

创建消息服务BMS Topic

参考文档：创建BMS主题。

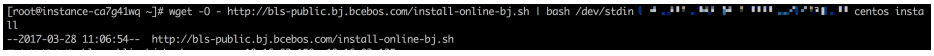
目前百度消息服务BMS支持“华北-北京”、“华南-广州”以及“香港2区”三个地区，创建主题前可以根据具体需求选择不同的区域。

安装BLS收集器

参考文档：安装收集器

1. 选择“收集器安装”，选择相应的操作系统后，点击“复制”；

2. 登录需要传输日志的主机，在root权限下执行所“复制”的安装命令。

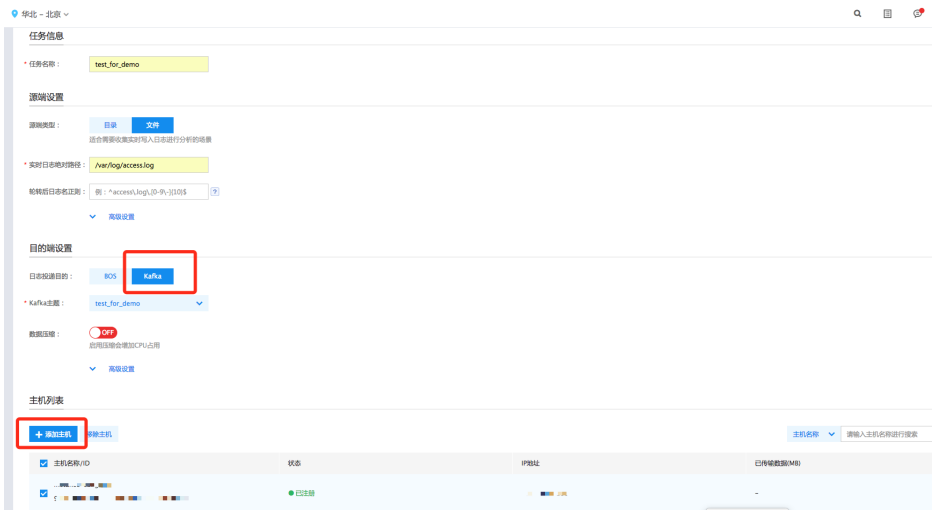


创建BLS传输任务

具体步骤为： 1. 在传输任务列表页面，点击“创建传输任务”，进入创建传输任务页面； 2. 在“任务信息”区，输入任务名称； 3. 在“源端设置”区，根据源数据类型，选择不通的源端类型以及进行相应的配置； 4. 在“目的端设置”区，选择“Kafka”作为日志投递目的； 5. 在“主机列表”区，点击“添加主机”，选择安装好“收集器”的主机； 6. 在“主机列表”区，选择需部署该传输任务的主机，点击“创建”；

详细操作步骤请参考文档：创建传输任务。

目前百度消息服务支持“华北-北京”、“华南-广州”两个地区，创建topic前可以根据具体需求选择不同的区域。



数据计算（Python）

数据计算过程通过BMR的Spark Streaming连接百度消息服务。本文以使用PySpark为例，Spark版本1.6，线上Kafka版本0.10。具体步骤如下：

创建BMR Spark集群

参考文档：创建集群

注意：在“集群配置”区，选择“Spark”内置模板，并将Spark选上。

下载Spark Kafka Streaming依赖

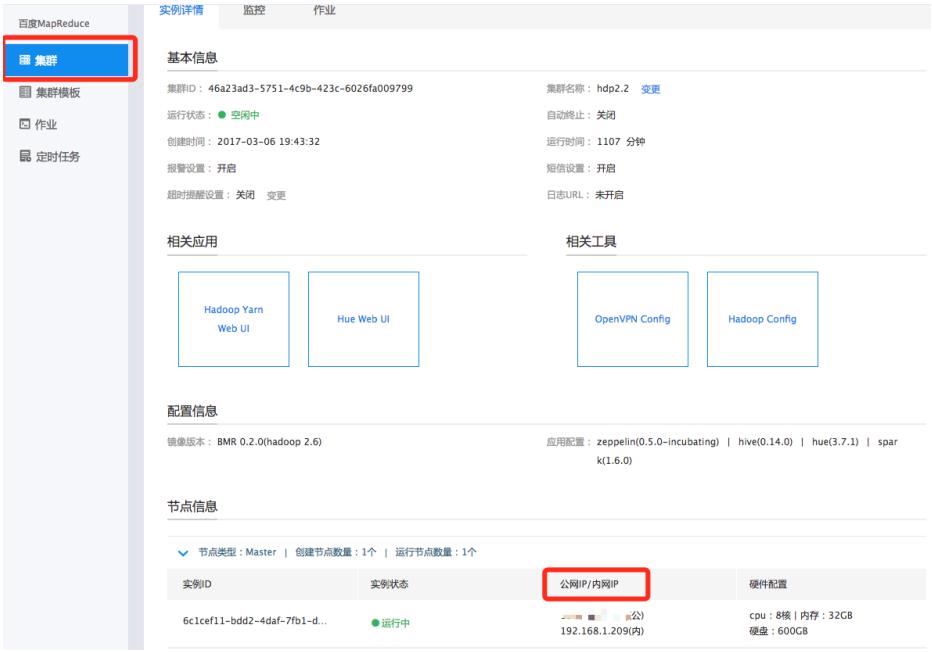
```
 # eip可以在BMR Console集群详情页的实例列表获取
ssh root@eip

# 切换到hdfs用户
su hdfs
cd

# 下载依赖
wget http://bmr-public-bj.bj.bcebos.com/sample/spark-streaming-kafka-0-10-assembly_2.10-1.6.0.jar
```

备注

获取集群登录公网IP



编写Spark Streaming程序

以Kafka_wordcount为例，使用前请删除文中注释：

```
 from __future__ import print_function

import sys
import ConfigParser

from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka010 import KafkaUtils
from pyspark.streaming.kafka010 import PreferConsistent
from pyspark.streaming.kafka010 import Subscribe

# 读取配置文件
def read_config(file_name):
    cf = ConfigParser.ConfigParser()
    # read config file
    cf.read(file_name)
    # read kafka config
    section = "kafka"
    opts = cf.options(section)
    config = {}
    for opt in opts:
        # topics should be a list
        if opt == "topics":
            config[opt] = str.split(cf.get(section, opt), ",")
        else:
            config[opt] = cf.get(section, opt)
    return config

if __name__ == "__main__":
    """
    if len(sys.argv) != 3:
        print("Usage: kafka_wordcount.py <bootstrap-server> <topic>", file=sys.stderr)
        exit(-1)
    """
    # 建立SparkContext和StreamingContext，demo处理间隔为20s
    sc = SparkContext(appName="PythonStreamingKafkaWordCount")
    ssc = StreamingContext(sc, 20)

    # 读取配置文件test.conf，获取连接百度Kafka参数
    config_file = "test.conf"
    kafkaParams = read_config(config_file)

    # 建立kafka输入流
    topics = kafkaParams["topics"]
    kvs = KafkaUtils.createDirectStream(ssc, PreferConsistent(), Subscribe(topics, kafkaParams))
    lines = kvs.map(lambda x: x[1])
    counts = lines.flatMap(lambda line: line.split(" ")) \
        .map(lambda word: (word, 1)) \
        .reduceByKey(lambda a, b: a+b)
    counts.pprint()

    # 启动StreamingContext
    ssc.start()
    ssc.awaitTermination()
```

下载百度消息服务的证书

下载证书：

创建连接Kafka配置文件

以“test.conf"为例：

```
vi test.conf

# 写入如下内容
[kafka]
bootstrap.servers = kafka.bj.baidubce.com:9091
topics = test_for_demo
group.id = test
# 以下为SSL配置，根据client.properties中的内容进行更换
security.protocol = SSL
ssl.truststore.password = test_truststore_password
ssl.truststore.location = client.truststore.jks
ssl.keystore.location = client.keystore.jks
ssl.keystore.password = test_keystore_password
```

参数说明：

```
bootstrap.servers: kafka服务地址
topics: 需要消费的topic，如需消费多个topic，以逗号分割，如topic1，topic2，topic3
group.id: consumer group的id，请不要随意设置，以免与其他用户冲突（kafka服务未来将支持groupId隔离）
```

更多配置见：http://kafka.apache.org/0100/documentation.html#newconsumerconfigs

提交Streaming作业

按照如上四步，当前目录（hdfs home目录）有五个文件：test.conf、client.truststore.jks、client.keystore.jks、kafka_wordcount.py、spark-streaming-kafka-0-10-assembly_2.10-1.6.0.jar：

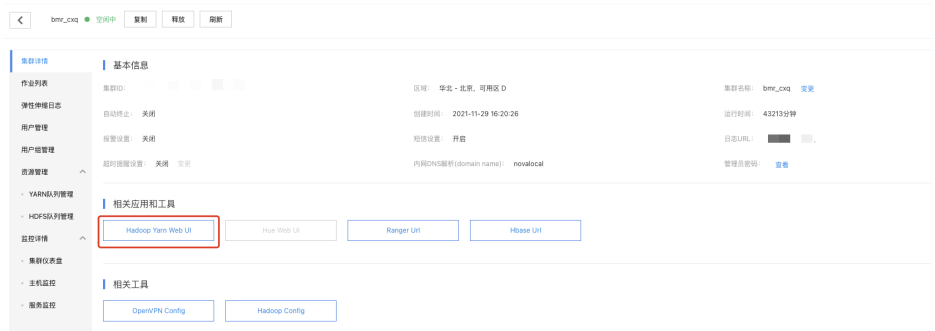
```
bash-4.1$ ls
client.keystore.jks    kafka_wordcount.py                                test.conf
client.truststore.jks  spark-streaming-kafka-0-10-assembly_2.10-1.6.0.jar
bash-4.1$
```

使用spark-submit提交streaming作业：

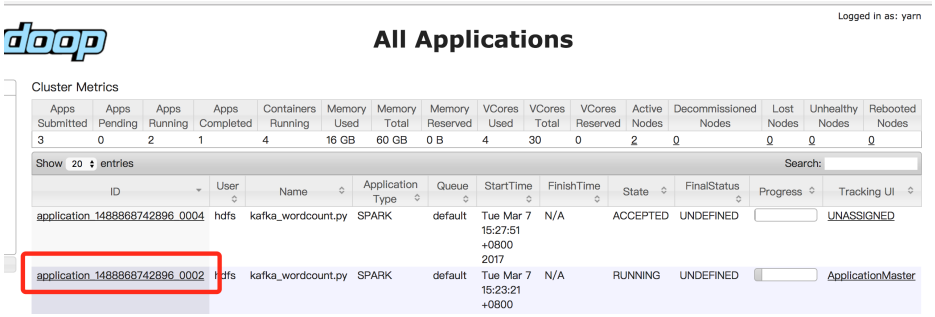
```
/usr/bin/spark-submit --master yarn --deploy-mode cluster  --files test.conf,client.keystore.jks,client.truststore.jks --jars spark-streaming-kafka-0-10-assembly_2.10-1.6.0.jar kafka_wordcount.py
```

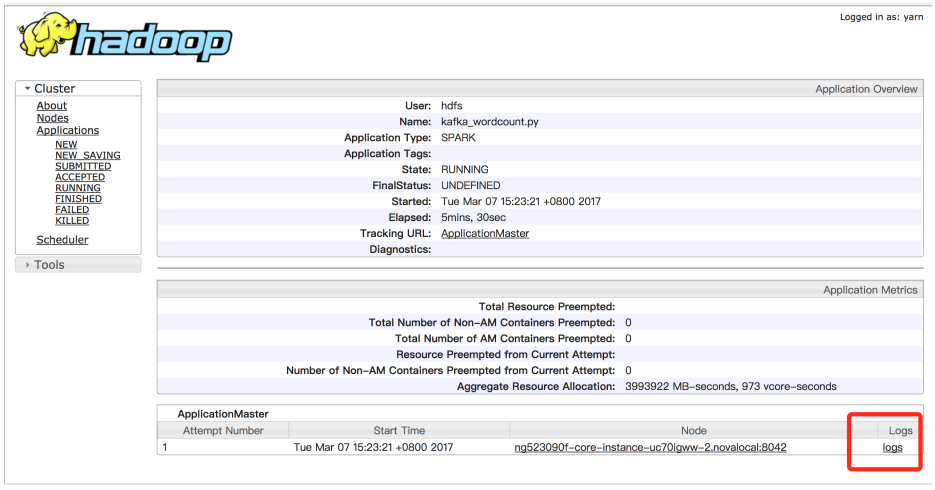
查看作业输出

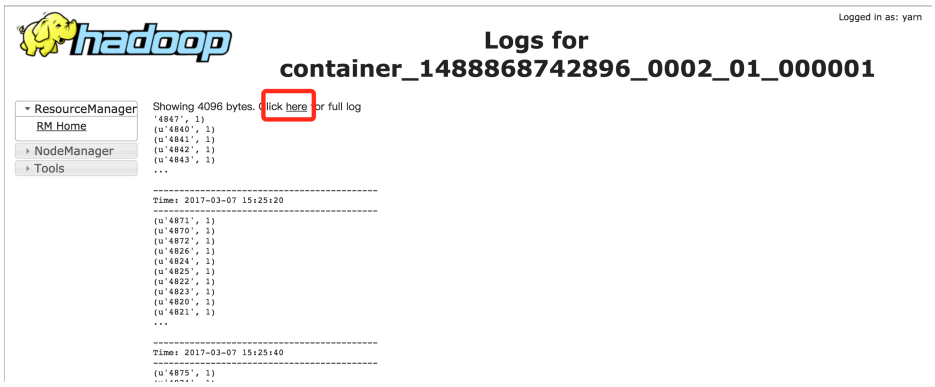
在“集群详情页”点开“Hadoop Yarn Web UI”，即可打开yarn console：



在yarn console可以查看对应application的日志，用以查看程序的输出，以kafka_wordcount为例，输出在stdout中：







注意事项

1. 如果需要停掉某个作业，可以使用“yarn application -kill applicationId”命令，例如：

```
yarn application -kill application_1488868742896_0002
```

2. 同时跑多个作业，请注意修改test.conf中的group.id配置

数据计算（Scala）

数据计算过程通过BMR的Spark Streaming连接百度消息服务BMS。本文以使用Scala为例，Spark版本2.1,线上Kafka版本0.10。具体步骤如下：

创建BMR Spark集群

参考文档：创建集群

注意：在“集群配置”区，选择“Spark2”内置模板，并将Spark选上。

下载Spark Kafka Streaming依赖

```
 # eip可以在BMR Console集群详情页的实例列表获取
ssh root@eip

# 切换到hdfs用户
su hdfs
cd

# 下载依赖
wget https://bmr-public-bj.bj.bcebos.com/sample/original-kafke-read-streaming-1.0-SNAPSHOT.jar
```

下载百度消息服务的证书

下载证书：

创建连接Kafka配置文件

以“test.conf”为例：

```
 vi test.conf

# 写入如下内容
[kafka]
bootstrap.servers = kafka.bj.baidubce.com:9091
topics = test_for_demo
group.id = test
# 以下为SSL配置，根据client.properties中的内容进行更换
security.protocol = SSL
ssl.truststore.password = test_truststore_password
ssl.truststore.location = client.truststore.jks
ssl.keystore.location = client.keystore.jks
ssl.keystore.password = test_keystore_password
```

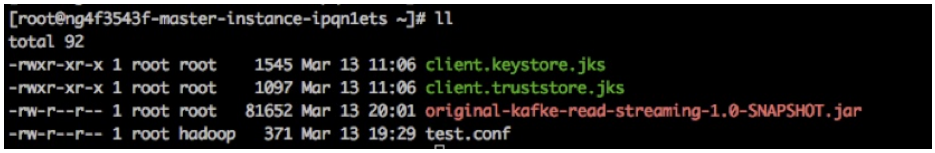
参数说明：

```
 bootstrap.servers: kafka服务地址
topics: 需要消费的topic，如需消费多个topic，以逗号分割，如topic1，topic2，topic3
group.id: consumer group的id，请不要随意设置，以免与其他用户冲突（kafka服务未来将支持groupId隔离）
```

更多配置见：http://kafka.apache.org/0100/documentation.html#newconsumerconfigs

提交Streaming作业

按照如上四步，当前目录有四个文件：test.conf、client.truststore.jks、client.keystore.jks、



使用spark-submit提交streaming作业：

```
spark-submit --class com.baidu.inf.spark.WordCount --master yarn --deploy-mode cluster  --files test.conf,client.keystore.jks,client.truststore.jks ./original-kafke-read-streaming-1.0-SNAPSHOT.jar "$topics" "$bootstrap.servers" "$group.id"  "$ssl.truststore.password" "$ssl.keystore.password"
```

(注意：这里请替换掉命令中最后面5个参数值为实际的值，即文件test.conf中描述的字段。)

例子：


```
spark-submit --class com.baidu.inf.spark.WordCount --master yarn --deploy-mode cluster --files test.conf,client.keystore.jks,client.truststore.jks ./original-kafke-read-streaming-1.0-SNAPSHOT.jar "868313b92dbe474b80ee4ef0904df26d__test" "kafka.bj.baidubce.com:9091" "test" "kafka" "k7ynher0"
```

附上WordCount 示例代码：

```
 package com.baidu.inf.spark

import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.SparkConf
import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe
import org.apache.spark.streaming.kafka010.KafkaUtils
import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent
import org.apache.spark.streaming.{Seconds, StreamingContext}

object WordCount {
  def className = this.getClass.getName.stripSuffix("$")

  def main(args: Array[String]): Unit = {
if (args.length < 4) {
  System.err.println(
    s"Usage:Input Params: "
      + " <topic> "
      + " <bootstrap.servers> "
      + " <group.id> "
      + " <ssl.truststore.password>"
      + " <ssl.keystore.password>"
  )
  sys.exit(1)
}
val Array(topic, bootstrap, group,
truststore, keystore, _*) = args

val conf = new SparkConf().setAppName(className).setIfMissing("spark.master", "local[2]")
conf.set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")

val ssc = new StreamingContext(conf, Seconds(5))

val kafkaParams = Map[String, Object](
  "bootstrap.servers" -> bootstrap,
  "key.deserializer" -> classOf[StringDeserializer].getName,
  "value.deserializer" -> classOf[StringDeserializer].getName,
  "group.id" -> group,
  "auto.offset.reset" -> "latest",
  "serializer.class" -> "kafka.serializer.StringEncoder",
  "ssl.truststore.location" -> "client.truststore.jks",
  "ssl.keystore.location" -> "client.keystore.jks",
  "security.protocol" -> "SSL",
  "ssl.truststore.password" -> truststore,
  "ssl.keystore.password" -> keystore,
  "enable.auto.commit" -> (true: java.lang.Boolean)
)
ssc.sparkContext.setLogLevel("WARN")

val topics = Array(topic)
// 消费kafka数据
val stream = KafkaUtils.createDirectStream[String, String](
  ssc,
  PreferConsistent,~~~~
  Subscribe[String, String](topics, kafkaParams)
).map(record => record.value())

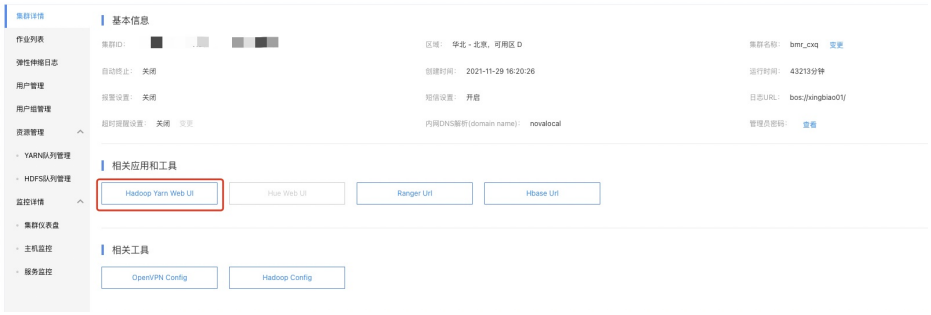
val counts = stream.flatMap(_.split(" "))
  .map(word => (word, 1))
  .reduceByKey(_ + _)

  counts.print()

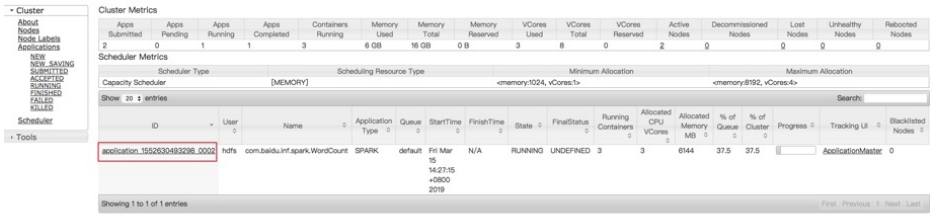
  ssc.start()
  ssc.awaitTermination()
  ssc.stop(true, true)
  }
 }
```

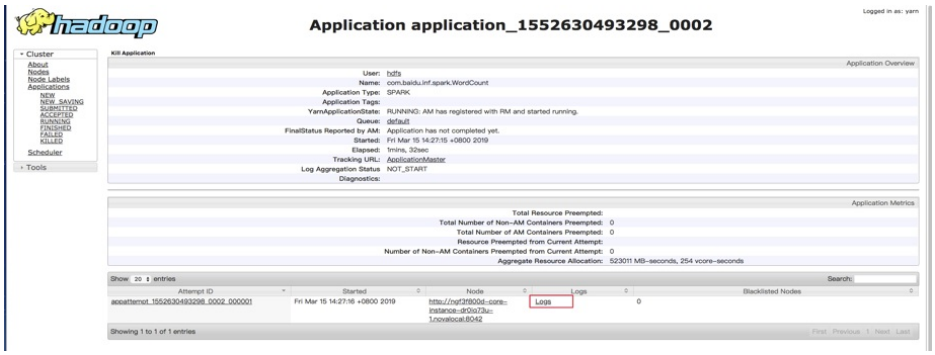
查看作业输出

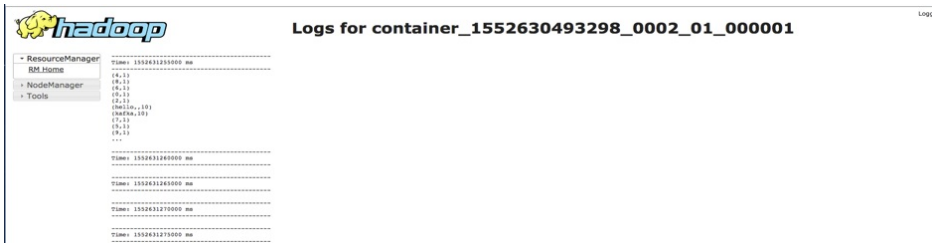
在“集群详情页”点开“Hadoop Yarn Web UI”，即可打开yarn console：



在yarn console可以查看对应application的日志，用以查看程序的输出，以kafka_wordcount为例，输出在stdout中：







注意事项

1. 如果需要停掉某个作业，可以使用“yarn application -kill applicationId”命令，例如：

```
yarn application -kill application_1488868742896_0002
```

2. 同时跑多个作业，请注意修改test.conf中的group.id配置

相关产品

BLS（百度Log Service）、BMS（百度消息服务）以及BMR（MapReduce）、弹性公网IP（EIP）。

离线应用场景

概览

离线数据分析适用于数据规模大、处理实时性要求不高的场景，例如用户行为分析、用户留存分析、报表统计等等。基于百度智能云大数据平台，用户可以便捷地实现离线数据分析，包括数据的采集、数据清洗、数据仓库以及商业智能展现。

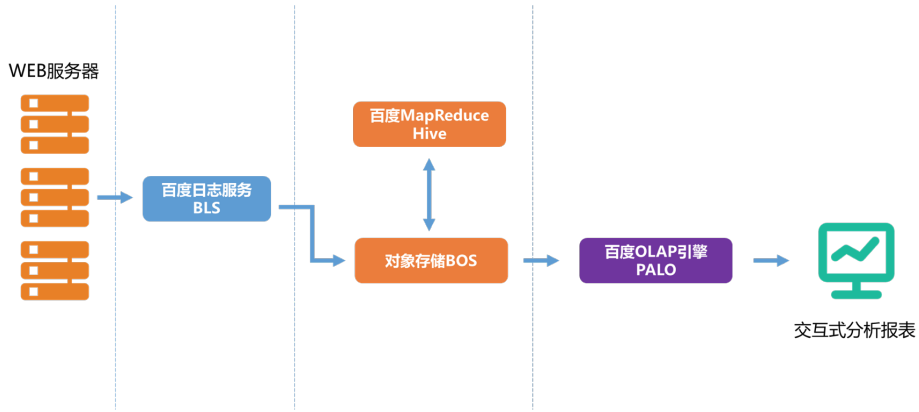
需求场景

大数据离线分析场景

通常是指对海量数据进分析和处理，形成结果数据，供下一步数据应用使用。离线处理对处理时间要求不高，但是所处理数据量较大，占用计算存储资源较多，通常通过MR或者Spark作业或者SQL作业实现。离线分析系统架构中以HDFS分布式存储软件为数据底座，计算引擎以基于MapReduce的Hive和基于Spark的SparkSQL为主。

方案概述

具体而言，可以通过使用BLS（百度LogService）、BOS（百度对象存储）、BMR（MapReduce）、Palo（百度OLAP引擎）这些产品实现上述场景。我们以常见的用户访问日志分析场景作为示例，离线处理架构图如下图所示：



首先，用户访问日志保存在WEB服务器的文件系统，通过在BLS服务创建传输任务，把相关服务器上的日志收集到BOS进行存储；然后使用BMR集群运行Hive作业对日志数据进行清洗和处理，输出的目标数据仍保存在BOS；最后，把目标数据从BOS导入到OLAP引擎Palo中，即可进行多维分析。百度智能云Palo还支持对接兼容JDBC接口的可视化分析应用，更加直观、便捷地展示数据分析结果。

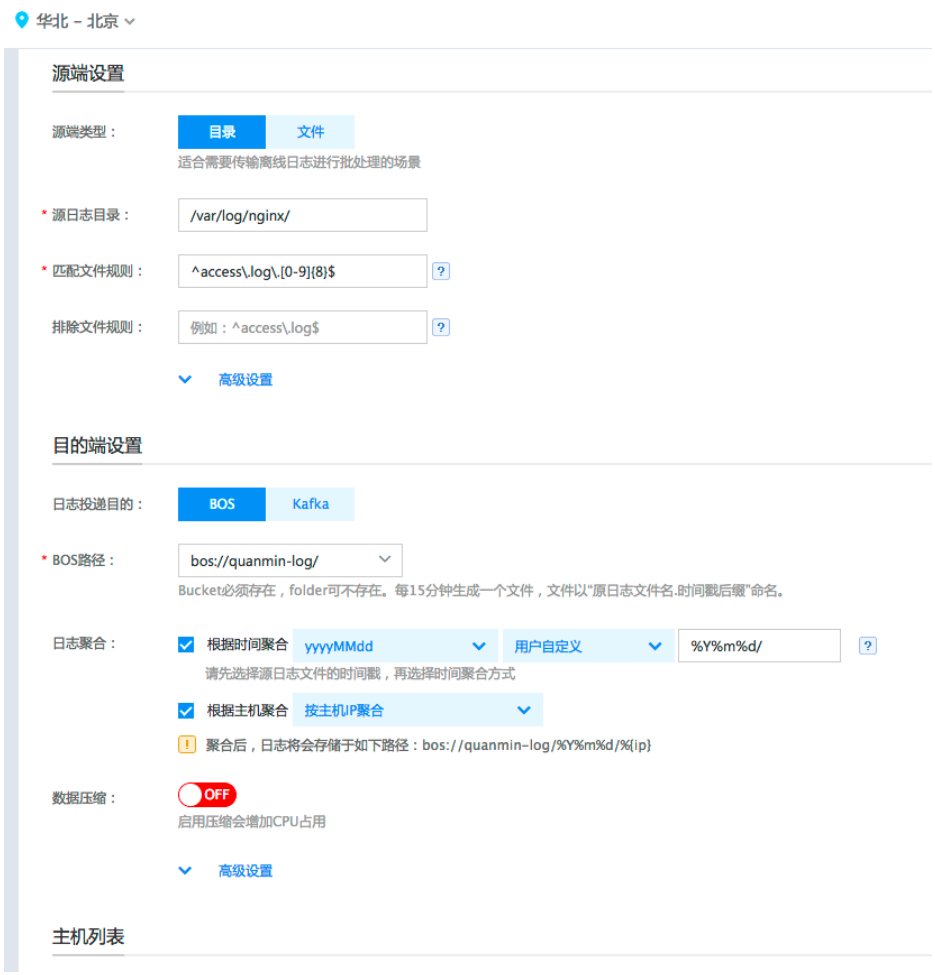
数据采集

通过BLS服务采集日志

参考文档

使用BLS服务首先需要在目标机器上安装收集器，收集器负责从BLS服务接收传输任务，并把日志数据从本地磁盘上传到BOS指定目录。

在本案例中，我们的Web服务器上运行了nginx进程，并且把nginx日志生成路径配置在/var/log/nginx/下，日志文件的格式为access.log.yyyyMMdd，根据日期进行轮转，每天生成一个新的日志文件，比如2017年3月20号的日志将都保存在文件access.log.20170320中。为了将这些日志文件收集到BOS保存，我们在BLS服务创建一个新的传输任务，具体配置信息如下图所示。源端类型为“目录”，将“源日志目录”配置为nginx日志所在目录：/var/log/nginx，“匹配文件规则”配置为日志文件的正则表达式：^access.log.[0-9]{8}$。



目的端设置中，选择“日志投递目的”为“BOS”，并且通过下拉选框选择BOS上目的路径为“bos://quanmin-log/”。另外，我们可以根据日志聚合的需要，在BOS端重新组织日志数据的目录结构。在本案例中，我们勾选“根据时间聚合”，注意选择源日志文件的时间戳为yyyyMMdd，这是跟我们服务器上日志文件名中的时间戳正则匹配的，BLS收集器将根据这个正则去匹配日志文件名，从而得到改日志文件对应的时间数据。然后，我们选择“用户自定义”，这里是配置具体的聚合路径，我们填入配置“%Y%m%d”，也就是把BOS目的端路径配置为bos://nginx-logs/%Y%m%d/。由于Web服务器有多台，我们需要把日志来源的服务器也在日志保存路径中体现出来，因此勾选“根据主机聚合”，并选择按主机ip聚合。最终，在BOS上保存的日志数据的路径将符合bos://quanmin-log/%Y%m%d/%{ip}的格式。

数据清洗

创建BMR集群，运行定时任务

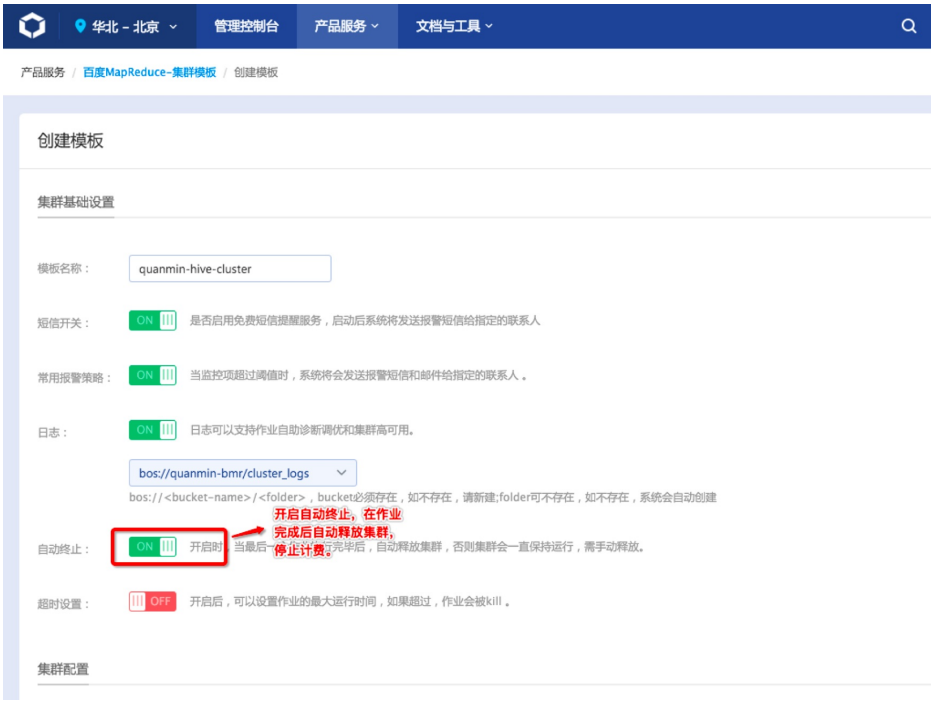
配置好BLS传输任务后，预期每天的日志都将传输到指定的BOS目录。nginx日志包含字段较多，我们需要进行选择并转换字段来得到符合需求的数据。这一步我们可以使用BMR定时任务功能，每天定期运行Hive作业，对导入BOS的日志数据进行用户活跃留存统计。

在本案例中，具体操作步骤如下：

1. 创建集群模板，操作入口是BMR产品控制台左侧导航栏“集群模板”-页面中部主体“创建模板”按钮：

创建模块的参数填写需要注意两点：

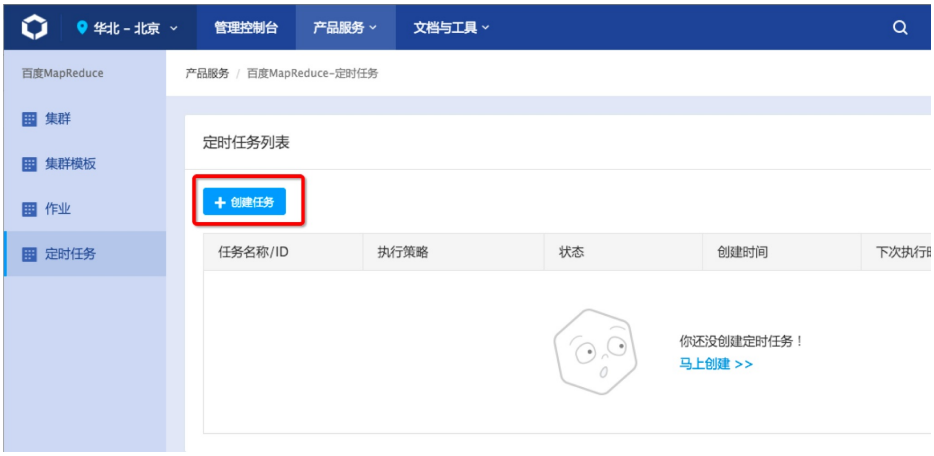
- 启动“自动终止”，该选项说明在定时任务完成后自动释放集群，停止计费。如果不启动，则定时任务执行过程创建出的集群将一直保持活跃状态，计入产品费用：



- 注意添加hive应用，否则集群将不能运行hive作业：

集群节点配置是根据具体作业任务的规模来配置的，如果不了解实际规模可以先保持默认节点配置，后续根据作业运行耗时情况来重新配置。

2. 创建定时任务，操作入口是BMR产品控制台左侧导航栏“定时任务”-页面中部主体“创建任务”按钮：



配置定时任务主要是选择集群模板、执行策略以及作业。集群模板需要选择刚才创建的集群模板。执行策略可以根据实际业务需要进行选择。这里以每天运行数据分析举例，配置执行频率是每1天。任务开始时间为立即开始（如果需要第二天一早看到数据结果，可以把开始时间定为凌晨某个时间点）。

3. 添加Hive作业到定时任务中。我们编写的Hive作业将对原始日志数据进行如下处理：

   1）对空值附以对业务无意义的数值。

   由于Palo不支持空值字段，需要对日志中的空值字段做一定处理，目前均处理为0。

   2）将IP信息转化为地址位置信息。

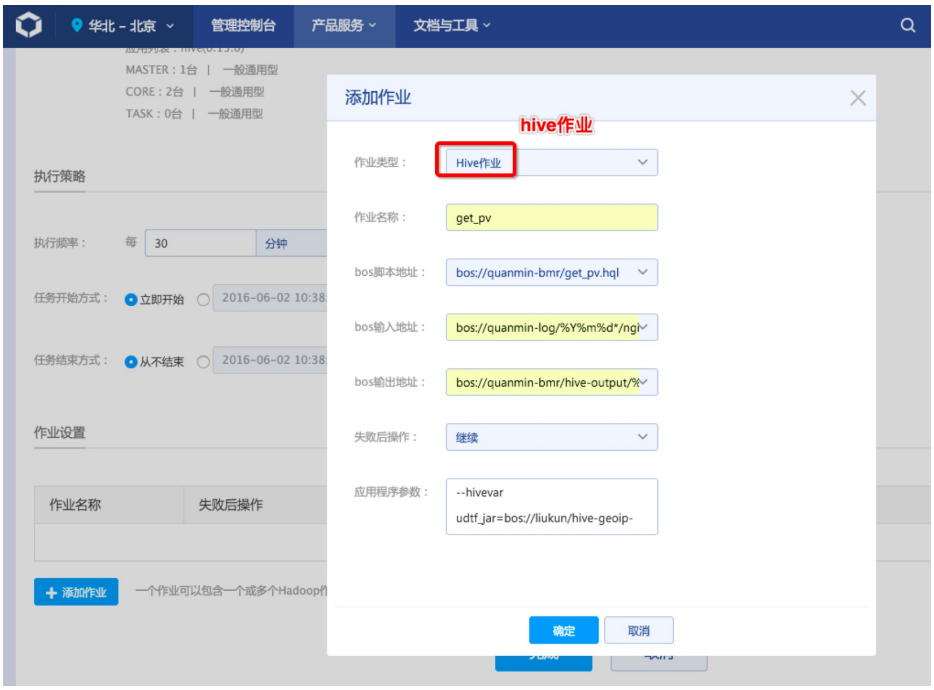
   通过IP-GEO库与UDF把IP信息转化为地理位置信息，精确到市一级，同时提供经纬度信息便于在可视化工具中做地图报表。

   3）将时间字段打散为多个字段。

   为方便下载查询，将日志中的时间字段打散为年、月、日、时、分等五个字段。同时为了在Palo中声明为date类型从而加快查询速度，且不丢失字段本身的含义，年、月字段需满足yyyy-mm-dd格式的合法值，故年字段统一为当年第一天，月字段统一为当月第一天，例如2016年表示为2016-01-01，2016年7月表示为2016-07-01。

   4）统计用户次日留存情况。

   次日留存定义为当天的所有访问IP中在前一天也访问过的IP子集，也就是一个客户端IP在某天及其前一天都出现在nginx访问日志中，则可认为该IP是一个次日留存客户。



点击“确定”。样例定时任务就只有一个Hive作业，因此可以点击“完成”提交定时任务。

标准创建定时任务的步骤请参考文档；

标准的Hive作业样例请参考文档。

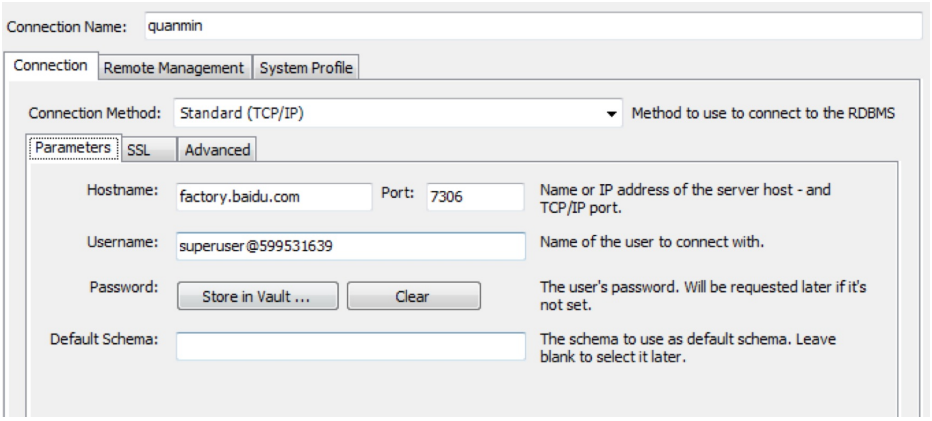
报表展现

Palo建表与数据导入

BMR运行hive作业处理过后的数据保存在BOS上，本案例中，上面定时任务运行hive作业，配置的BOS输出地址是 bos://quanmin-bmr/hive-output/%Y%m%d/access_pv/，其中"%Y%m%d"将根据任务实际执行日期替换为数字，比如20160601。为了把BMR清洗好的数据导入Palo，我们首先需要在Palo预先建立好数据库表。首先，在Palo服务创建一个新集群。

集群创建好以后， Palo提供兼容MySQL的接口，可以直接使用MySQL的相关库或者工具进行连接Palo集群（目前Palo只支持MySQL 5.0以上的客户端，在连接之前请确认您的客户端版本）。以MySQL Workbench为例：

1. 连接Palo集群。使用的配置信息均可以在Palo集群的详情页面查看到：
   - hostname：factory.baidu.com
   - port：7036
   - username：superuser@cluster_id，实例中cluster_id为599531639



2. 建立数据库。

```
CREATE DATABASE QUANMIN;
```

3. 建立数据表。目前需要建两张表，一张是包含所有数据和字段的明细表（detail table），一张是次日留存表（retain_day table，当前页包含了所有字段）。这里我们给出datail表的建表语句：

```
 CREATE TABLE detail
(
platform int,
action varchar(100),
v1 varchar(100),
v2 varchar(100),
year date,
month date,
day date,
hour datetime,
minute datetime,
time datetime,
user_id int,
device varchar(50),
ip varchar(20),
server_ip varchar(20),
province_name varchar(50),
city_name varchar(50),
longitude varchar(20),
latitude varchar(20),
pv int sum
)
engine = olap
partition by range(time) (
PARTITION p1 VALUES LESS THAN ("2016-08-01 00:00:00"),
PARTITION p2 VALUES LESS THAN ("2016-10-01 00:00:00")
)
distributed by hash(platform)
```

4. 最后进行BOS数据导入，在MySQL Workbench继续执行语句：

```
 load label detail1 (
data infile("bos://quanmin-bmr/hive-output/2016-07-30/*") into table
`detail` columns terminated by "," (platform, action, v1, v2, user_id, device, time, year, month, day, hour,minute, ip, server_ip, province_name,
city_name,longitude, latitude, pv)
)
PROPERTIES(
"bos_accesskey" ="your_ak",
"bos_secret_accesskey" = "your_sk",
"bos_endpoint"= "http://bj.bcebos.com"
);
```

5. 使用主流的BI/可视化工具进行数据分析与展现。Palo支持JDBC接口连接访问，因此对于兼容JDBC接口的BI/可视化工具都能连接Palo集群，对我们导入的数据进行可视化分析。

注意：

- 每次成功导入后，下次需要使用不同的导入label，导入语句中的your_ak、your_sk需要替换为实际的ak、sk。百度智能云使用ak、sk验证用户身份，请妥善保管。
- 导入过程中，可以通过 “SHOW LOAD;” 命令查看load过程。
- 导入成功后，即可通过MySQL客户端和BI/可视化工具进行查询分析。

相关产品

BLS（百度LogService）、BOS（百度对象存储）、BMR（MapReduce）、Palo（百度OLAP引擎）

HIVE

Hive 操作 Hbase 外部表

本示例使用 BMR-2.3.0(Hadoop: 3.1.1, Hive:3.1.0, HBASE: 2.2.7)来演示 Hive 操作 Hbase 外部表。其他版本操作应该类似。

1. 创建 Hbase 表

```
 hbase shell
```

创建 hbase 表 hbase_hive_table。

```
 hbase(main):004:0> create 'hbase_hive_table', 'cf'
Created table hbase_hive_table
Took 1.3137 seconds
```

2. Hive 操作

进入 hive 环境

```
 hive
```

- 设置引擎为 mr MapReduce 引擎 和本集群的 Hbase 环境已经调好，如果使用其他集群的 Hbase，可以用 add file hbase-site.xml 添加其他集群的配置文件（需要服务器打通）。TEZ 引擎需要额外的 Hbase 相关的配置，需使用 add jar 的方式把 hbase 的相关 jar 包放到执行环境里。

```
 set hive.execution.engine=mr;
```

- 创建外部表

```
 CREATE EXTERNAL TABLE hbase_hive_table (key int, value string)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf:val")
TBLPROPERTIES ("hbase.table.name" = "hbase_hive_table", "hbase.mapred.output.outputtable" = "hbase_hive_table");
```

- 插入数据

```
 INSERT OVERWRITE TABLE HBASE_HIVE_TABLE values(98, 'abc');
```

- 检索数据

```
 select * from HBASE_HIVE_TABLE;
OK
98 abc
Time taken: 0.246 seconds, Fetched: 1 row(s)
```

2.1 Join 测试

- 创建表

```
 create table t1(t1_c1 int, t1_c2 string);
insert into t1 values(1,'t1_key1'),(2,'t1_key2'),(3,'t1_key3');

create table t2(t2_c1 int, t2_c2 string);
insert into t2 values(2,'t2_key2'),(3,'t2_key3'),(4,'t2_key4');
```

- 插入数据

```
 INSERT OVERWRITE TABLE HBASE_HIVE_TABLE
select t1_c1, concat(t1_c2, t2_c2)
from t1 join t2 on t1_c1 = t2_c1;
```

- 检索结果

```
 select * from HBASE_HIVE_TABLE;
OK
2 t1_key2t2_key2
3 t1_key3t2_key3
98 abc
Time taken: 0.141
```

不同集群的 Hive 迁移方案

不同集群的 Hive 迁移方案

本文档描述了怎样把 Hive 数据库从一个 Hadoop 集群迁移到另一个 Hadoop 集群。

本文档假定新集群的 Hive 元数据库的内容可以清空。

1. 停止老集群 Hive 集群 的 hive-metastore 和 hive-server2

停止方法：使用命令`systemctl stop hive-metastore`或者通过 ambari 操作。

```
 systemctl stop hive-metastore
systemctl stop hive-server2
```

停止 hive metastore 后，集群所有的访问 hive 的任务都会报错。

2. 新集群 Hive 停止所有的 hive-metastore 和 hive-server2

```
 systemctl stop hive-metastore
systemctl stop hive-server2
```

3. 备份 MySQL 数据库

```
 mysqldump -h ${MYSQL_HOST} -uhive '-p${MYSQL_PASSWD}' hive > hive-metastore.bak
```

4. 恢复 MySQL数据库到新的地址

登录新的数据库

```
 mysql -h ${NEW_MYSQL_HOST} -uhive '-p${MYSQL_PASSWD}' hive
```

如果目标数据库已经存在则先删除

```
 drop database hive;
```

创建目标数据库

```
create database hive;
```

导入数据到新的数据库。

```
mysql -h ${NEW_MYSQL_HOST} -uhive '-p${MYSQL_PASSWD}' hive < hive-metastore.bak
```

${NEW_MYSQL_HOST} 是数据库新的地址。

5. 新 Hive 集群修改 MetaStore 服务器所在的 hive-site.xml

按需要修改以下配置，如果 MYSQL 数据库信息没有改变，则不需要修改。

```
  <property>
   <name>javax.jdo.option.ConnectionDriverName</name>
   <value>com.mysql.jdbc.Driver</value>
  </property>

  <property>
   <name>javax.jdo.option.ConnectionPassword</name>
   <value>${MYSQL_PASSWD}</value>
  </property>

  <property>
   <name>javax.jdo.option.ConnectionURL</name>
   <value>jdbc:mysql://${NEW_MYSQL_HOST}/hive?createDatabaseIfNotExist=true&amp;characterEncoding=UTF-8</value>
  </property>
  <property>
   <name>javax.jdo.option.ConnectionUserName</name>
   <value>hive</value>
  </property>
```

6. 升级 MetaStore 到新集群的版本（选做）

如果新集群的 Hive 版本比老集群的 Hive 版本新，执行此步骤。如果一致，则跳过此步骤。新集群的 Hive 版本不得小于老集群的 Hive 版本。 在新集群 metastore 所在服务器，使用 hive 用户下执行以下操作, ${NEW_MYSQL_HOST}是新 mysql 数据库所在的服务器 IP 地址。 如下是把 hive 从 1.2 升级到 3.1。

```
 cd /opt/bmr/hive/scripts/metastore/upgrade/mysql
mysql -uhive '-p${MYSQL_PASSWD}' -h ${NEW_MYSQL_HOST} hive
```

进入 MySQL 之后，执行以下 SQL 命令。

```
 source upgrade-1.2.0-to-2.0.0.mysql.sql;
source upgrade-2.0.0-to-2.1.0.mysql.sql;
source upgrade-2.1.0-to-2.2.0.mysql.sql;
source upgrade-2.2.0-to-2.3.0.mysql.sql;
source upgrade-2.3.0-to-3.0.0.mysql.sql;
source upgrade-3.0.0-to-3.1.0.mysql.sql;
```

7. 复制老集群的数据到新集群

7.1 复制老集群管理表数据到新集群 此步骤要求所有管理表都在老集群参数`hive.metastore.warehouse.dir`设置的目录下。

```
 <property>
  <name>hive.metastore.warehouse.dir</name>
  <value>/warehouse/tablespace/managed/hive</value>
 </property>
```

执行命令：

```
 hadoop fs -rm -r hdfs://${new-cluster}/${hive.metastore.warehouse.dir}
hadoop distcp hdfs://${old-cluster}/${hive.metastore.warehouse.dir} hdfs://${new-cluster}/${hive.metastore.warehouse.dir
```

注意：目标路径 hdfs://${new-cluster}/${hive.metastore.warehouse.dir} 必须不存在，否则会拷贝到它的子目录。

一般在新集群中执行 distcp 命令。如果在新集群执行 distcp 命令，需要在新集群设置老集群的配置，并且所有新集群的服务器需要识别老集群的服务器，反之亦然。

如果不能用 hdfs 协议访问老集群，可以使用 hftp 或者 webhdfs 协议。

7.2 复制老集群外部表数据到新集群 此步骤要求所有管理表都在老集群参数`hive.metastore.warehouse.external.dir`设置的目录下，或者统一的目录。

```
<property>
<name>hive.metastore.warehouse.external.dir</name>
<value>/warehouse/tablespace/external/hive</value>
</property>
```

执行命令：

```
hadoop fs -rm -r hdfs://${new-cluster}/${hive.metastore.warehouse.external.dir}
hadoop distcp hdfs://${old-cluster}/${hive.metastore.warehouse.external.dir} hdfs://${new-cluster}/${hive.metastore.warehouse.external.dir}
```

一般在新集群中执行 distcp 命令。如果在新集群执行 distcp 命令，需要在新集群设置老集群的配置，并且所有新集群的服务器需要识别老集群的服务器，反之亦然。

如果不能用 hdfs 协议访问老集群，可以使用 hftp 或者 webhdfs 协议。

8. 修改元数据库数据的位置。

mysql -h 172.18.0.148 -uhive '-pxxx' hive

```
-- managed db location
update DBS SET DB_LOCATION_URI=replace(DB_LOCATION_URI, 'hdfs://${old-cluster}/${hive.metastore.warehouse.dir}','hdfs://${new-cluster}/${hive.metastore.warehouse.dir}');
-- external db location
update DBS SET DB_LOCATION_URI=replace(DB_LOCATION_URI, 'hdfs://${old-cluster}/${hive.metastore.warehouse.external.dir}', 'hdfs://${new-cluster}/${hive.metastore.warehouse.external.dir}' );
-- managed table and partition location
UPDATE SDS SET LOCATION = REPLACE(LOCATION, 'hdfs://${old-cluster}/${hive.metastore.warehouse.dir}','hdfs://${new-cluster}/${hive.metastore.warehouse.dir}');
-- external table partition location
update SDS SET LOCATION=replace(LOCATION, 'hdfs://${old-cluster}/${hive.metastore.warehouse.external.dir}', 'hdfs://${new-cluster}/${hive.metastore.warehouse.external.dir}');
```

9. 重启新集群的 metastore 和 hiveserver

进入新集群的 metastore 和 hiveserver 对应的服务器，以 root 身份执行以下命令。

```
systemctl start hive-metastore;
systemctl start hive-server2;
```

10. 进行测试

查询老集群的表，看数据是否正确。

迁移示例

老集群 hdfs 地址为：hdfs://master-9331fa9:8020，hive 版本为 hive 1.2.0，mysql 元数据库地址为 master-9331fa9 新集群 hdfs 地址为：hdfs://master-b882990:8020，hive 版本为 hive 3.1.0，mysql 元数据库地址为 master-b882990（IP：172.18.0.148）

在老集群创建相应的库和表

```
create database managed;
use managed;
create table t(c1 string) stored as textfile;
load data local inpath '/etc/profile' overwrite into table t;
create table tp(c1 string) partitioned by (pt string)stored as textfile;
load data local inpath '/etc/profile' overwrite into table tp partition(pt='profile');

create database e_db location '/warehouse/tablespace/external/hive/e_db.db';
use e_db;
create table e(c1 string) stored as textfile;
load data local inpath '/etc/hosts' overwrite into table e;
create table ep(c1 string) partitioned by (pt string)stored as textfile;
load data local inpath '/etc/hosts' overwrite into table ep partition(pt='hosts');
```

1. 停止老集群 hive 集群 的 hive-metastore 和 hive-server2

在老集群的 master-9331fa9 的 root 账号执行以下命令：

```
systemctl stop hive-metastore
systemctl stop hive-server2
```

2. 新集群 hive 停止所有的 hive-metastore 和 hive-server2

在老集群的 master-b882990 的 root 账号执行以下命令：

```
systemctl stop hive-metastore
systemctl stop hive-server2
```

3. 备份 mysql 数据库

```
mysqldump -h master-9331fa9 -uhive '-pxxx' hive > hive-metastore.bak
```

4. 恢复 mysql数据库到新的地址

登录新的数据库

```
mysql -h 172.18.0.148 -uhive '-pxxx' hive
```

目标数据库已经存在，先删除

```
drop database hive;
```

创建目标数据库

```
create database hive;
```

恢复 MYSQL:

```
mysql -h 172.18.0.148 -uhive '-pxxx' hive < hive-metastore.bak
```

5. 新集群修改 metastore 服务器所在的 hive-site.xml

新集群连接 MYSQL 数据库的地址没有改变，跳过此步。

6. 升级 metastore 到 新集群的版本

如下是把 hive 从 1.2 升级到 3.1。

```
cd /opt/bmr/hive/scripts/metastore/upgrade/mysql
mysql -h 172.18.0.148 -uhive '-pxxx' hive
```

进入 MYSQL 之后，执行以下 SQL 命令。

```
source upgrade-1.2.0-to-2.0.0.mysql.sql;
source upgrade-2.0.0-to-2.1.0.mysql.sql;
source upgrade-2.1.0-to-2.2.0.mysql.sql;
source upgrade-2.2.0-to-2.3.0.mysql.sql;
source upgrade-2.3.0-to-3.0.0.mysql.sql;
source upgrade-3.0.0-to-3.1.0.mysql.sql;
quit;
```

7. 复制老集群的数据到新集群

把老集群的 /etc/hosts 的内容负责到新集群所有服务器的 /etc/hosts 中。

7.1 复制老集群管理表数据到新集群

以 hive 账号执行以下命令

```
hadoop fs -rm -r hdfs://master-b882990/warehouse/tablespace/managed/hive
hadoop distcp  hdfs://master-9331fa9:8020/apps/hive/warehouse hdfs://master-b882990:8020/warehouse/tablespace/managed/
```

7.2 复制老集群外部表数据到新集群 以 hive 账号执行以下命令:

```
hadoop fs -rm -r /warehouse/tablespace/external/hive
hadoop distcp  hdfs://master-9331fa9:8020/warehouse/tablespace/external/hive hdfs://master-b882990:8020/warehouse/tablespace/external/hive
```

8. 修改元数据库数据的位置

```
mysql -h 172.18.0.148 -uhive '-pxxx' hive
```

进入 MYSQL 环境执行以下命令：

```
 -- managed db location
update DBS SET DB_LOCATION_URI=replace(DB_LOCATION_URI, 'hdfs://master-9331fa9:8020/apps/hive/warehouse','hdfs://master-b882990:8020//warehouse/tablespace/managed/hive');
-- external db location
update DBS SET DB_LOCATION_URI=replace(DB_LOCATION_URI, 'hdfs://master-9331fa9:8020/warehouse/tablespace/external/hive','hdfs://master-b882990:8020/warehouse/tablespace/external/hive');
-- managed table and partition location
UPDATE SDS SET LOCATION = REPLACE(LOCATION, 'hdfs://master-9331fa9:8020/apps/hive/warehouse','hdfs://master-b882990:8020/warehouse/tablespace/managed/hive');
-- external table partition location
update SDS SET LOCATION=replace(LOCATION, 'hdfs://master-9331fa9:8020/warehouse/tablespace/external/hive','hdfs://master-b882990:8020/warehouse/tablespace/external/hive');
```

9. 重启新集群的 metastore 和 hiveserver

进入新集群的 metastore 和 hiveserver 对应的服务器，以 root 身份执行以下命令。

```
 systemctl start hive-metastore;
systemctl start hive-server2;
```

10. 验证

验证正常。

API参考

API简介

概述

MapReduce（BMR）是全托管的Hadoop/Spark集群，您可以按需部署并弹性扩展集群，只需专注于大数据处理、分析、报告，拥有多年大规模分布式计算技术积累的百度运维团队全权负责集群运维。

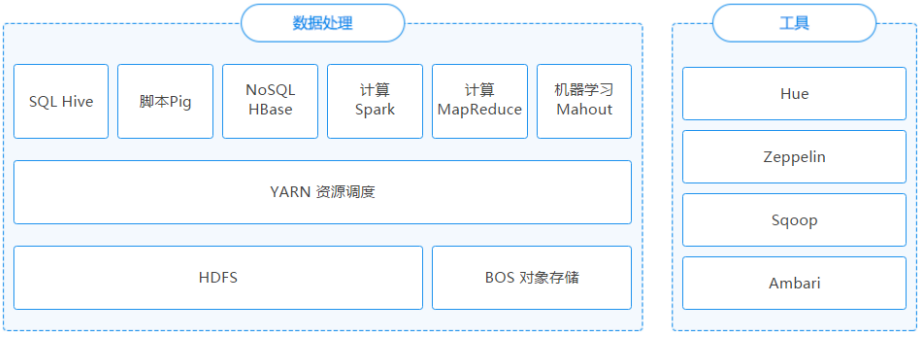
MapReduce支持完整的Hadoop生态：

Hadoop：提供可靠存储HDFS以及MapReduce编程范式以便大规模并行处理数据。 Spark：提供基于分布式内存的大规模并行处理框架，从而大大提高大数据分析性能。Spark提供了SQL查询接口、流数据处理以及机器学习。 HBase：大规模分布式NoSQL数据库，提供随机存取大量的非结构化和半结构化的海量数据。

与自己搭建Hadoop集群相比，MapReduce有以下优势：

方便：几分钟便可创建集群，无需为节点分配、部署、优化投入时间。 弹性：创建任意大小的集群并动态调整集群规模，高峰期加大集群规模以提高计算能力，低峰期可对应缩减集群规模降低花费。 开放：完全兼容开源Hadoop/Spark社区，零成本业务迁移。 实惠：支持按需付费以及包年包月，计价简单而透明。 安全：专属私有网络，独占系统环境，确保数据安全。

MapReduce组件



如果您是初次调用百度智能云产品的API，可以观看API入门视频指南，快速掌握调用API的方法。

接口概览

本节汇总了BMR集群可调用 API，具体接口信息请点击链接查看详细内容。

接口	描述
集群操作接口	集群列表查询、集群信息查询、创建集群、释放集群。
实例组操作接口	查询实例组列表、修改实例组配置
实例操作接口	查询实例列表

产品限制

系统限制

- 活跃集群总数的限制 同时活跃的集群总数不得超过5个。
- 单集群作业总数的限制 单一集群中提交的作业总数不得超过256个。
- 单集群节点个数的限制 默认情况下，集群中Master节点实例组的实例个数应为1个，HA模式下为3个；Core实例组的实例个数不得小于2个且不得大于20个，Task实例组的实例个数不得大于20个。

通用说明

实名认证

百度智能云提供个人认证、企业认证两种认证方式，您可以选择任意一种方式进行认证。

认证机制

所有API的安全认证一律采用Access Key与请求签名机制。 Access Key由Access Key ID和Secret Access Key组成，均为字符串。 对于每个HTTP请求，使用下面所描述的算法生成一个认证字符串。提交认证字符串放在Authorization头域里。服务端根据生成算法验证认证字符串的正确性。 认证字符串的格式为`bce-auth-v{version}/{accessKeyId}/{timestamp}/{expirationPeriodInSeconds}/{signedHeaders}/{signature}`。

- version是正整数。
- timestamp是生成签名时的UTC时间。
- expirationPeriodInSeconds表示签名有效期限。
- signedHeaders是签名算法中涉及到的头域列表。头域名之间用分号（;）分隔，如host;x-bce-date。列表按照字典序排列。（本API签名仅使用host和x-bce-date两个header）
- signature是256位签名的十六进制表示，由64个小写字母组成。

当百度智能云接收到用户的请求后，系统将使用相同的SK和同样的认证机制生成认证字符串，并与用户请求中包含的认证字符串进行比对。如果认证字符串相同，系统认为用户拥有指定的操作权限，并执行相关操作；如果认证字符串不同，系统将忽略该操作并返回错误码。鉴权认证机制的详细内容请参见鉴权认证机制。

通信协议

支持HTTP和HTTPS两种调用方式。为了提升数据的安全性，建议通过HTTPS调用。

请求结构说明

数据交换格式为JSON，所有request/response body内容均采用UTF-8编码。

请求参数包括如下4种：

参数类型	说明
URI	用于指明操作实体，如:PUT /v1/cluster/{clusterUuid}
Query参数	URL中携带的请求参数
HEADER	通过HTTP头域传入，如:x-bce-date
RequestBody	通过JSON格式组织的请求数据体

响应结构说明

响应值分为两种形式：

响应内容	说明
HTTP STATUS CODE	如200,400,403,404等
ResponseBody	JSON格式组织的响应数据体

版本号

参数	类型	参数位置	描述	是否必须
version	String	URI参数	API版本号	必须

幂等性

当调用创建接口时如果遇到了请求超时或服务器内部错误，用户可能会尝试重发请求，这时用户通过clientToken参数避免创建出比预期要多的资源，即保证请求的幂等性。

幂等性基于clientToken，clientToken是一个长度不超过64位的ASCII字符串，通常放在query string里，如http://bcc.bj.baidubce.com/v1/instance?clientToken=be31b98c-5e41-4838-9830-9be700de5a20。

如果用户使用同一个clientToken值调用创建接口，则服务端会返回相同的请求结果。因此用户在遇到错误进行重试的时候，可以通过提供相同的clientToken值，来确保只创建一个资源；如果用户提供了一个已经使用过的clientToken，但其他请求参数（包括queryString和requestBody）不同甚至url Path不同，则会返回IdempotentParameterMismatch的错误代码。

clientToken的有效期为24小时，以服务端最后一次收到该clientToken为准。也就是说，如果客户端不断发送同一个clientToken，那么该clientToken将长期有效。

日期与时间

日期与时间的表示有多种方式。为统一起见，除非是约定俗成或者有相应规范的，凡是HTTP标准中规定的表示日期和时间字段用GMT，其他日期时间表示的地方一律采用UTC时间，遵循ISO 8601，并做以下约束：

- 表示日期一律采用YYYY-MM-DD方式，例如2014-06-01表示2014年6月1日。
- 表示时间一律采用hh:mm:ss方式，并在最后加一个大写字母Z表示UTC时间。例如23:00:10Z表示UTC时间23点0分10秒。
- 凡涉及日期和时间合并表示时，在两者中间加大写字母T，例如2014-06-01T23:00:10Z表示UTC时间2014年6月1日23点0分10秒。

规范化字符串

通常一个字符串中可以包含任何Unicode字符。在编程中这种灵活性会带来不少困扰。因此引入“规范字符串”的概念。一个规范字符串只包含百分号编码字符以及URI（Uniform Resource Identifier）非保留字符（Unreserved Characters）。 RFC 3986规定URI非保留字符包括以下字符：字母（A-Z，a-z）、数字（0-9）、连字号（-）、点号（.）、下划线（_）、波浪线（~）。 将任意一个字符串转换为规范字符串的方式是:

- 将字符串转换成UTF-8编码的字节流。
- 保留所有URI非保留字符原样不变。
- 对其余字节做一次RFC 3986中规定的百分号编码（Percent-Encoding），即一个%后面跟着两个表示该字节值的十六进制字母。字母一律采用大写形式。

示例： 原字符串：this is an example for 测试，对应的规范字符串：this%20is%20an%20example%20for%20%E6%B5%8B%E8%AF%95。

BMR产品编码要求

- 可解析内容，所有request/response body内容目前均使用UTF-8编码，后续会支持更多encoding类型。
- 在请求时，需要对以下做UrlEncode：
  - Objectname，其中，Resource做UrlEncode的时候需要忽略“/”。
  - Querystring的Value。
  - x-bce-copy-source（忽略“/”）。
  - 自定义Meta:Meta Value只支持可见的ASCII字符，如果需要其它的字符，推荐使用UrlEncode处理。

BMR基本概念

Cluster：集群，由多个实例组组成的计算集群。 Step：作业，同一集群中的多个作业按照提交顺序依次执行。 BOS：百度对象存储。 Bucket：对象存储系统中的文件桶。 BOS Key：简称Key，百度对象存储在用户所属的Bucket级别内，唯一标示一个文件资源标示符。 Access Key ID / Secret Access Key：BMR的API服务采用Access Key与请求签名机制。Access Key由Access Key ID和Secret Access Key组成，均为字符串。

服务域名

区域	服务端点Endpoint	协议
北京	bmr.bj.baidubce.com	HTTP and HTTPS
广州	bmr.gz.baidubce.com	HTTP and HTTPS
苏州	bmr.su.baidubce.com	HTTP and HTTPS
上海	bmr.fsh.baidubce.com	HTTP and HTTPS
度小满	bmr.hb-fsg.baidubce.com	HTTP and HTTPS
保定	bmr.bd.baidubce.com	HTTP and HTTPS

> 注意：
>
> Region(区域)代表着一个独立的地域，是百度智能云中的重要概念，请参考区域选择说明。百度智能云中的服务除了极少数如账号服务全局有效之外，绝大部分服务都是区域间隔离的，每个区域的服务独立部署互不影响，服务间共享数据需要通过显式拷贝完成，在API中引用区域必须使用其ID。 为了保证整体服务的高效，BMR的服务区域会和百度对象存储BOS的服务区域保持一致。北京区域的BMR服务可以处理北京区域BOS上存储的数据，广州区域的BMR服务可以处理广州区域BOS上存储的数据，以此类推。

公共头

公共请求头

头域	说明	是否必须
host	服务域名	必须
Authorization	鉴权信息，详细请参考鉴权认证	必须
x-bce-date	该请求创建的时间，表示日期一律采用YYYY-MM-DD方式，例如2014-06-01表示2014年6月1日。如果用户使用了标准的Date域，该头域可以不填。当两者同时存在时，以x-bce-date为准。	必须
content-type	application/json	可选
x-bce-content-sha256	表示内容部分的SHA256签名的十六进制字符串，其中内容指HTTP Request Payload Body，即Content部分在被HTTP encode之前的原始数据。	可选
x-bce-if-match	同If-Match语义。	可选
x-bce-if-none-match	同If-None-Match语义。	可选

公共响应头

头域	说明
x-bce-request-id	对应请求的requestId。

错误码

错误码格式

当用户访问API出现错误时，会返回给用户相应的错误码和错误信息，便于定位问题，并做出适当的处理。请求发生错误时通过Response Body返回详细错误信息，遵循如下格式：

参数名	类型	说明
code	String	表示具体错误类型。
message	String	有关该错误的详细说明。
requestId	String	导致该错误的requestId。

例如：

```
 {
   "code":"IllegalRequestUrl",
   "message":"The requested url belongs to domain which is not under acceleration",
   "requestId":" 81d0b05f-5ad4-1f22-8068-d5c9de60a1d7"
}
```

公共错误码

错误码	错误消息	HTTP状态码	描述
AccessDenied	Access denied.	403 Forbidden	无权限访问对应的资源。
InappropriateJSON	The JSON you provided was well-formed and valid, but not appropriate for this operation.	400 Bad Request	请求中的JSON格式正确，但语义上不符合要求。如缺少某个必需项，或者值类型不匹配等。出于兼容性考虑，对于所有无法识别的项应直接忽略，不应该返回这个错误。
InternalError	We encountered an internal error. Please try again.	500Internal Server Error	所有未定义的其他错误。在有明确对应的其他类型的错误时（包括通用的和服务自定义的）不应该使用。
InvalidAccessKeyId	The Access Key ID you provided does not exist in our records.	403 Forbidden	Access Key ID不存在。
InvalidHTTPAuthHeader	The HTTP authorization header is invalid. Consult the service documentation for details.	400 Bad Request	Authorization头域格式错误。
InvalidHTTPRequest	There was an error in the body of your HTTP request.	400 Bad Request	HTTP body格式错误。例如不符合指定的Encoding等。
InvalidURI	Could not parse the specified URI.	400 Bad Request	URI形式不正确。例如一些服务定义的关键词不匹配等。对于ID不匹配等问题，应定义更加具体的错误码，例如NoSuchKey。
MalformedJSON	The JSON you provided was not well-formed.	400 Bad Request	JSON格式不合法。
InvalidVersion	The API version specified was invalid.	404 Not Found	URI的版本号不合法。
OptInRequired	A subscription for the service is required.	403 Forbidden	没有开通对应的服务。
PreconditionFailed	The specified If-Match header doesn't match the ETag header.	412 Precondition Failed	详见ETag。
RequestExpired	Request has expired. Timestamp date is XXX.	400 Bad Request	请求超时。XXX要改成x-bce-date的值。如果请求中只有Date，则需要将Date转换为datetime。
IdempotentParameterMismatch	The request uses the same client token as a previous, but non-identical request.	403 Forbidden	clientToken对应的API参数不一样。
SignatureDoesNotMatch	The request signature we calculated does not match the signature you provided. Check your Secret Access Key and signing method. Consult the service documentation for details.	400 Bad Request	Authorization头域中附带的签名和服务端验证不一致。

BMR错误码

错误码	错误信息	HTTP状态码	描述
ClusterNotFound	The requested cluster does not exist. Please double confirm your clusterId or the existence of the cluster.	404	所访问的集群不存在，请重新确认指定的集群ID是否正确或指定的集群是否存在。
ClusterOverLimit	The maximum number of your Running or Waiting clusters is 5.	403	单一用户最多可以保持5个活跃（运行中或空闲中）的集群。
InappropriateJSON	The JSON provided is well-formed and valid, but not appropriate for this operation.	400	请求中的JSON格式正确，但语义上不符合要求。如缺少某个必须参数，或者参数类型不匹配等。对于所有无法识别的参数将直接忽略，不会返回这个错误。
InsufficientBalance	Insufficient balance, please refill your account.	403	余额不足，请充值。
InternalError	We encountered an internal error. Please try again or contact us.	500	所有未定义的其他错误。
InvalidArguments	The arguments of step properties can not be parsed correctly.	400	解析作业参数时遇到错误，请重新确认作业参数是否填写正确。
InvalidBosPath	The BOS path provided may be malformed or incorrect.	400	BOS路径不合法，请重新确认请求中依赖的BOS文件存在并填写正确的路径。
InvalidInstanceGroupType	Instance group type should be Master or Core.	400	实例组类型可以选择Master、Core或者Task。
InvalidInstanceType	The instance type specified is invalid.	400	无法识别实例类型，请参考实例规格中的实例类型填写。
InvalidStepType	The step type specified is invalid.	400	无法识别作业类型，请参考作业类型的编码进行填写。
InvalidVersion	The API version specified is invalid.	404	URI的版本号不合法。当前取值为1。
MalformedJSON	The JSON provided is not well-formed.	400	JSON格式不合法。
MissingProperties	One or more required fields are missing in step properties.	400	作业属性中缺少必要的信息，请根据作业类型指定必要的字段。
OperationDenied	The request could not be fullfilled.	403	执行请求的条件未被满足。
StepNotFound	The requested step does not exist. Please double confirm your stepId or the existence of the step.	404	所访问的作业不存在，请重新确认指定的作业ID是否正确或指定的作业是否存在。
StepOverLimit	The maximum number of steps in one cluster is 256.	403	单一集群中提交的作业总数不得超过256个，尝试通过交互式访问运行更多作业。
ValidationError	The request fails to satisfy the constraints specified by BMR service.	400	请求不符合BMR服务的规范，请检查请求路径，请求头域，各项参数以及消息体格式等是否符合要求。
Invalid_RootDisk_Type	The specified type system disk is not exist.	400	该类型的系统盘不存在，请检查您的配置！请参考云磁盘类型的编码进行填写
Invalid_CdsDataDisk_Type	The specified type data disk is not exist.	400	该类型的数据盘不存在，请检查您的配置！请参考云磁盘类型的编码进行填写

集群操作接口

创建集群

接口描述

向BMR服务请求创建一个新的集群。

权限说明

请求发起人需要具有合法的AccessKeyID和SecretAccessKey才能发起请求，请参考 鉴权认证。

注意事项

如果请求中没有用户验证信息（即匿名访问），返回`403 Forbidden`，错误信息：`AccessDenied`。

请求结构

```
 POST /v{version}/cluster?clientToken={clientToken} HTTP/1.1
accept-encoding: gzip, deflate
x-bce-date: {utc-date-string}
host: bmr.bj.baidubce.com
connection: keep-alive
accept: */*
content-type: application/json
authorization: {bce-authorization-string}
{
    "applications": [
        {
            "name": appilicationName,
            "version": applicationVersion
        },
        {
            "name": appilicationName,
            "version": applicationVersion
        }
    ],
    "securityGroup": securityGroupId,
    "vpcId": vpcShortId,
    "subnetId": subnetShortId,
    "autoTerminate": false,
    "imageType": imageType,
    "imageVersion": imageVersion,
    "serviceHaEnabled":serviceHaEnabled,
    "safeModeEnabled":safeModeEnabled,
    "instanceGroups": [
        {
            "instanceCount": instanceCount,
            "instanceType": instanceType,
            "type": instanceGroupType,
            "rootDiskSizeInGB": rootDiskSizeInGB,
            "rootDiskMediumType": rootDiskMediumType,
            "cds": [
                {
                    "sizeInGB": sizeInGB,
                    "mediumType": mediumType
                },
                {
                    "sizeInGB": sizeInGB,
                    "mediumType": mediumType
                }
            ]
        }
    ],
    "logUri": logUri,
    "name": clusterName,
    "adminPassword": adminPassword,
    "steps": [
        {
            "actionOnFailure": actionOnFailture,
            "name": stepName,
            "properties": {
                "arguments": arguments,
                "mapper": mapper,
                "reducer": reducer,
                "input": input,
                "output": output
            },
            "type": stepType,
            "additionalFiles":[]
        }
    ]
}
```

请求头域

除公共头域外，无其它特殊头域。

售卖套餐请参考https://cloud.baidu.com/doc/BMR/s/6jwvxw85z

请求参数

名称	类型	是否必须	参数位置	参数描述
version	String	是	URL参数	API版本号，当前取值1
imageType	String	是	RequestBody参数	集群类型，比如 hadoop，参考BMR集群
imageVersion	String	是	RequestBody参数	集群版本，比如 2.2.2，参考BMR集群
instanceGroups	List<InstanceGroupConfig>	是	RequestBody参数	实例组配置
clientToken	String	否	Query参数	幂等性Token，是一个长度不超过64位的ASCII字符串
applications	List<ApplicationConfig>	否	RequestBody参数	需要安装的组件信息（如hive，pig，hbase）
autoTerminate	Boolean	否	RequestBody参数	为true则在运行完所有作业后自动释放集群，默认为true
logUri	String	否	RequestBody参数	用于上传运行日志的BOS路径，不提供则不上传日志
name	String	否	RequestBody参数	集群名称，默认为my-cluster
steps	List<StepConfig>	否	RequestBody参数	需要运行的作业信息，参考添加作业协议
vpcId	String	是	RequestBody参数	设置vpc短ID信息
subnetId	String	是	RequestBody参数	设置子网短ID信息
securityGroup	String	是	RequestBody参数	设置安全组短ID
adminPassword	String	是	RequestBody参数	设置集群密码，8-16位字符，英文，数字和符号必须同时存在，符号仅限!@#$%^*()，密码需要加密传输

响应头域

除公共头域外，无其它特殊头域。

响应参数

名称	类型	描述
clusterId	String	系统生成的集群唯一标识

请求示例

```
 POST /v1/cluster?clientToken=be31b98c-5e41-4838-9830-9be700de5a20 HTTP/1.1
accept-encoding: gzip, deflate
x-bce-date: 2022-01-26T13:02:00Z
host: bmr.bj.baidubce.com
connection: keep-alive
accept: */*
content-type: application/json
authorization: bce-auth-v1/46bd9968a6194b4bbdf0341f2286ccce/2022-01-26T13:02:00Z/1800/host;x-bce-
date/994014d96b0eb26578e039fa053a4f9003425da4bfedf33f4790882fb4c54903

{
    "applications": [
        {
            "name": "hive",
            "version": "3.1.0"
        },
        {
            "name": "tez",
            "version": "0.9.1"
        },
        {
            "name": "spark2",
            "version": "2.4.4"
        }
    ],
```

```
    "securityGroup": "g-ai3jzpiadv1x",
    "vpcId": "d5465344-e783-41b1-9765-ec3ee4bec332",
    "subnetId": "b35c3b60-18a0-42a6-87e7-e857dadd9ec3",
    "autoTerminate": false,
    "imageType": "hadoop",
    "imageVersion": "2.2.2",
    "serviceHaEnabled":true,
    "safeModeEnabled":false,
    "instanceGroups": [
      {
        "instanceCount": 1,
        "instanceType": "bmr.g1.xlarge",
        "type": "Master",
        "rootDiskSizeInGB": 40,
        "rootDiskMediumType": "ssd",
        "cds": [
          {
            "sizeInGB": 50,
            "mediumType": "ssd"
          }
        ]
      },
      {
        "instanceCount": 3,
        "instanceType": "bmr.g1.xlarge",
        "type": "Core",
        "rootDiskSizeInGB": 40,
        "rootDiskMediumType": "ssd",
        "cds": [
          {
            "sizeInGB": 50,
            "mediumType": "ssd"
          }
        ]
      }
    ],
    "logUri": "bos://data-wh-sdk-bigdata/clusterlog",
    "name": "my-cluster",
    "adminPassword": "7c658fefaaae7d4516d9305b27770a55",
    "steps": [
      {
        "actionOnFailure": "Continue",
        "name": "my-mapreduce-streaming",
        "properties": {
          "arguments": "-D mapreduce.job.reduces=20",
          "mapper": "cat",
          "reducer": "cat",
          "input": "bos://mybucket/inputs/txt-1m",
          "output": "bos://mybucket/outputs/streaming"
        },
        "type": "Streaming",
         "additionalFiles":[
         {
           "remote":"bos://mybucket/i",
           "local":""
         },
         {
           "remote":"",
           "local":""
         }
         ]
      }
    ]
}
```

响应示例

```
HTTP/1.1 201 CREATED
Transfer-Encoding: chunked
x-bce-request-id: 73c4e74c-3101-4a00-bf44-fe246959c05e
Cache-Control: no-cache
Server: BWS
Date: Wed, 26 Jan 2022 13:02:01 GMT
Content-Type: application/json;charset=UTF-8
{
    "clusterId": "0181b994-576a-4a00-b04e-f6fa9d5eee48"
}
```

查询集群列表

接口描述

查询所有集群的信息，支持分页查询。

权限说明

请求发起人需要具有合法的AccessKeyID和SecretAccessKey才能发起请求，请参考 鉴权认证。

注意事项

如果请求中没有用户验证信息（即匿名访问），返回`403 Forbidden`，错误信息：`AccessDenied`。

请求结构

```
GET /v{version}/cluster?marker={marker}&maxKeys={maxKeys} HTTP/1.1
accept-encoding: gzip, deflate
x-bce-date: {utc-date-string}
host: bmr.bj.baidubce.com
connection: keep-alive
accept: */*
content-type: application/json
authorization: {bce-authorization-string}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名	参数类型	是否必须	参数位置	参数描述
version	String	是	URL参数	API版本号，当前取值1
marker	String	否	Query参数	批量获取列表的查询的起始位置，是一个由系统生成的字符串
maxKeys	Int	否	Query参数	每页包含的最大数量，最大数量不能超过1000。大于1000的会被修正为1000。缺省值为1000

响应头域

除公共头域外，无其它特殊头域。

响应参数

参数名	参数类型	参数描述
marker	String	标记查询的起始位置
isTruncated	Boolean	true表示后面还有数据，false表示已经是最后一页
clusters	List<Cluster>	返回的集群列表

请求示例

```
GET /v1/cluster?marker=9e0c8cf1-690c-444a-8727-04241f3beaa5&maxKeys=30 HTTP/1.1
accept-encoding: gzip, deflate
x-bce-date: 2022-01-26T13:02:00Z
host: bmr.bj.baidubce.com
connection: keep-alive
accept: */*
content-type: application/json
authorization: bce-auth-v1/46bd9968a6194b4bbdf0341f2286ccce/2022-01-26T13:02:00Z/1800/host;x-bce-
date/994014d96b0eb26578e039fa053a4f9003425da4bfedf33f4790882fb4c54903
```

响应示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
x-bce-request-id: 73c4e74c-3101-4a00-bf44-fe246959c05e
Cache-Control: no-cache
Server: BWS
Date: Wed, 26 Jan 2022 13:02:01 GMT
Content-Type: application/json;charset=UTF-8
{
    "clusters": [
        {
            "id": "8fa594c8-665b-49cc-b1ec-743bf1bf5ca2",
            "name": "test_name",
            "payType": "postpay",
            "status": {
                "state": "Running",
                "code": "",
                "message": "",
                "orderStatus": "",
                "creationDateTime": "2022-01-25T06:03:42Z",
                "endDateTime": "2022-01-26T11:14:20.26516422+08:00",
                "readyDateTime": "2022-01-25T06:06:05Z",
                "expireDateTime": "2022-01-25T06:03:41Z",
                "expireDates": 0
            },
            "imageType": "hadoop",
            "imageVersion": "2.3.0",
            "enableAutoScale": true,
            "createTime": "2022-01-25T06:03:42Z",
            "tags": null,
            "applications": [
                {
                    "name": "hdfs",
                    "version": "3.1.1",
                    "properties": {}
                },
                {
                    "name": "yarn",
                    "version": "3.1.1",
                    "properties": {}
                },
                {
                    "name": "mapreduce2",
                    "version": "3.1.1",
                    "properties": {}
                },
                {
                    "name": "zookeeper",
                    "version": "3.4.6",
                    "properties": {}
                },
                {
                    "name": "hadoop-manager",
                    "version": "1.0.0",
                    "properties": {}
                },
                {
                    "name": "ldap",
                    "version": "2.4.48",
                    "properties": {}
                }
            ]
        },
          {
            "id": "06a30746-ec2e-4f11-9bd1-643fd06d28d2",
            "name": "hadoop_120_test",
            "payType": "postpay",
            "status": {
                "state": "Starting",
                "code": "",
                "message": "",
                "orderStatus": "",
                "creationDateTime": "2022-01-26T03:14:52Z",
                "endDateTime": "2022-01-26T11:17:21.453684987+08:00",
                "readyDateTime": null,
                "expireDateTime": "2022-01-26T03:14:51Z",
                "expireDates": 0
            },
```

```
            "imageType": "hadoop",
            "imageVersion": "1.2.0",
            "enableAutoScale": true,
            "createTime": "2022-01-26T03:14:52Z",
            "tags": null,
            "applications": [
                {
                    "name": "hdfs",
                    "version": "2.7.1"
                },
                {
                    "name": "zookeeper",
                    "version": "3.4.6"
                },
                {
                    "name": "yarn",
                    "version": "2.7.1"
                },
                {
                    "name": "mapreduce2",
                    "version": "2.7.1"
                },
                {
                    "name": "hadoop-manager",
                    "version": "1.0.0"
                },
                {
                    "name": "spark2",
                    "version": "2.1.0"
                },
                {
                    "name": "hive",
                    "version": "1.2.0",
                    "properties": {}
                },
                {
                    "name": "pig",
                    "version": "0.15.1"
                },
                {
                    "name": "hue",
                    "version": "3.10.0"
                },
                {
                    "name": "hbase",
                    "version": "1.1.2"
                },
                {
                    "name": "ranger",
                    "version": "0.5.0"
                },
                {
                    "name": "tez",
                    "version": "0.7.0"
                }
            ]
        }
    ],
    "isTruncated": false,
    "marker": "9e0c8cf1-690c-444a-8727-04241f3beaa5"
}
```


查询集群信息

接口描述

查询一个指定集群的信息。

权限说明

请求发起人需要具有合法的AccessKeyID和SecretAccessKey才能发起请求，请参考 鉴权认证。

注意事项

如果请求中没有用户验证信息（即匿名访问），返回`403 Forbidden`，错误信息：`AccessDenied`。

请求结构

```
GET /v{version}/cluster/{clusterId} HTTP/1.1
accept-encoding: gzip, deflate
x-bce-date: {utc-date-string}
host: bmr.bj.baidubce.com
connection: keep-alive
accept: */*
content-type: application/json
authorization: {bce-authorization-string}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名	参数类型	是否必须	参数位置	参数描述
version	String	是	URL参数	API版本号，当前取值1
clusterId	String	是	URL参数	待查询的集群ID

响应头域

除公共头域外，无其它特殊头域。

响应参数

参数名	参数类型	参数描述
applications	List<Application>	需要安装的组件信息（如hive，pig，hbase）
orderId	String	订单号
id	String	集群ID，是一个定长字符串，只包含大小写字母、数字、连字号（-）和下划线（_）
name	String	集群名称
status	ClusterStatus	集群状态信息
imageType	String	集群类型，参考BMR集群
imageVersion	String	集群版本，参考BMR集群
payType	String	付费方式
logUri	String	用于上传运行日志的BOS路径
availabilityZone	String	集群所选可用区
vpcId	String	集群所选VPC
subnetId	String	集群所选子网ID
securityGroupId	String	集群所选安全组ID
serviceHaEnabled	Boolean	是否开启高可用模式
safeModeEnabled	Boolean	是否开启安全模式
enableAutoScale	Boolean	是否支持弹性伸缩
instanceGroup	InstanceGroup	实例组

请求示例

```
GET /v1/cluster/d647cda9-b308-47e4-a2ab-0a4908e17643 HTTP/1.1
accept-encoding: gzip, deflate
x-bce-date: 2022-01-26T13:02:00Z
host: bmr.bj.baidubce.com
connection: keep-alive
accept: */*
content-type: application/json
authorization: bce-auth-v1/46bd9968a6194b4bbdf0341f2286ccce/2022-01-26T13:02:00Z/1800/host;x-bce-
date/994014d96b0eb26578e039fa053a4f9003425da4bfedf33f4790882fb4c54903
```

响应示例

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
x-bce-request-id: 73c4e74c-3101-4a00-bf44-fe246959c05e
Cache-Control: no-cache
Server: BWS
Date: Wed, 26 Jan 2022 13:02:01 GMT
```

```
Content-Type: application/json;charset=UTF-8
{
    "id": "cf866824-f35b-4e2c-b5f5-6ee067806d7c",
    "name": "test_hadoop_230_zzf",
    "availabilityZone": "zoneA",
    "payType": "postpay",
    "status": {
        "state": "TerminatedWithError",
        "code": "",
        "message": "openApi request[BatchCreateServer] get unexpected status code, code: 400, body: {\"requestId\":\"a3211ab4-672c-492c-5c24-
3eafc8bb0b0c\",\"code\":\"Instance.BccExceedCapacity\",\"message\":\"Bcc capacity is not enough\"}",
        "orderStatus": "",
        "creationDateTime": "2022-01-24T02:47:12Z",
        "endDateTime": "2022-01-26T11:19:10.434269608+08:00",
        "readyDateTime": null,
        "expireDateTime": "2022-01-24T02:47:12Z",
        "expireDates": 0
    },
    "orderId": "",
    "imageType": "hadoop",
    "imageVersion": "2.3.0",
    "applications": [
        {
            "name": "hdfs",
            "version": "3.1.1",
            "properties": {}
        },
        {
            "name": "yarn",
            "version": "3.1.1",
            "properties": {}
        },
        {
            "name": "mapreduce2",
            "version": "3.1.1",
            "properties": {}
        },
        {
            "name": "zookeeper",
            "version": "3.4.6",
            "properties": {}
        },
        {
            "name": "hadoop-manager",
            "version": "1.0.0",
            "properties": {}
        },
        {
            "name": "ldap",
            "version": "2.4.48",
            "properties": {}
        },
        {
            "name": "spark2",
            "version": "2.4.4",
            "properties": {}
        },
        {
            "name": "hive",
            "version": "3.1.0",
            "properties": {
                "metastore": "default"
            }
        },
        {
            "name": "tez",
            "version": "0.9.1",
            "properties": {}
        },
        {
            "name": "azkaban",
            "version": "3.58.0",
            "properties": {}
        },
        {
            "name": "flink",
            "version": "1.12.4",
            "properties": {}
```

```
            "properties": {}
        },
        {
            "name": "presto",
            "version": "0.219",
            "properties": {}
        },
        {
            "name": "pig",
            "version": "0.17.0",
            "properties": {}
        },
        {
            "name": "zeppelin",
            "version": "0.8.0",
            "properties": {}
        },
        {
            "name": "hbase",
            "version": "2.2.7",
            "properties": {
                "backupEnabled": false,
                "restoreEnabled": false
            }
        },
        {
            "name": "oozie",
            "version": "5.1.0",
            "properties": {}
        },
        {
            "name": "hue",
            "version": "4.8.0",
            "properties": {}
        }
    ],
    "vpcId": "d5465344-e783-41b1-9765-ec3ee4bec331",
    "subnetId": "b35c3b60-18a0-42a6-87e7-e857dadd9eb2",
    "securityGroupId": "g-ai3jzpiadv1q",
    "serviceHaEnabled": true,
    "safeModeEnabled": true,
    "enableAutoScale": true，
    "instanceGroups":[
      {
        id:"b35c3b60-18a0-42a6-87e7-e8355",
        instanceType:"Master",
        name:"",
        type:"",
        instanceCount:12,
        rootDiskSizeInGB:1024,
        rootDiskMediumType:"hdd",
        cds:[
        {
          "sizeInGB":1024,
          "mediumType":"hdd"
        }
        ]
      }
    ]
}
```

释放集群

接口描述

释放集群，仅限单个集群释放。

权限说明

请求发起人需要具有合法的AccessKeyID和SecretAccessKey才能发起请求，请参考 鉴权认证。

注意事项

如果请求中没有用户验证信息（即匿名访问），返回403 Forbidden，错误信息：AccessDenied。

请求结构

```
DELETE /v{version}/cluster/{clusterId} HTTP/1.1
accept-encoding: gzip, deflate
x-bce-date: {utc-date-string}
host: bmr.bj.baidubce.com
connection: keep-alive
accept: */*
content-type: application/json
authorization: {bce-authorization-string}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名	参数类型	是否必须	参数位置	参数描述
version	String	是	URL参数	API版本号，当前取值1
clusterId	String	是	URL参数	待释放的集群ID

响应头域

除公共头域外，无其它特殊头域。

响应参数

无参数

请求示例

```
DELETE /v1/cluster/0ce4f730-4af2-4f37-8fa2-b14f2f44e50e HTTP/1.1
accept-encoding: gzip, deflate
x-bce-date: 2022-01-26T13:02:00Z
host: bmr.bj.baidubce.com
connection: keep-alive
accept: */*
content-type: application/json
authorization: bce-auth-v1/46bd9968a6194b4bbdf0341f2286ccce/2022-01-26T13:02:00Z/1800/host;x-bce-
date/994014d96b0eb26578e039fa053a4f9003425da4bfedf33f4790882fb4c54903
```

响应示例

```
HTTP/1.1 204 NO_CONTENT
Transfer-Encoding: chunked
x-bce-request-id: 73c4e74c-3101-4a00-bf44-fe246959c05e
Cache-Control: no-cache
Server: BWS
Date: Wed, 26 Jan 2022 13:02:01 GMT
Content-Type: application/json;charset=UTF-8
```

实例组操作接口

查询实例组列表

接口描述

查询指定集群的实例组信息

权限说明

请求发起人需要具有合法的AccessKeyID和SecretAccessKey才能发起请求，请参考 鉴权认证。

注意事项

如果请求中没有用户验证信息（即匿名访问），返回`403 Forbidden`，错误信息：`AccessDenied`。

请求结构

```
 GET /v{version}/cluster/{clusterId}/instanceGroup HTTP/1.1
accept-encoding: gzip, deflate
x-bce-date: {utc-date-string}
host: bmr.bj.baidubce.com
connection: keep-alive
accept: */*
content-type: application/json
authorization: {bce-authorization-string}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名	参数类型	是否必须	参数位置	参数描述
version	String	是	URL参数	API版本号，当前取值1
clusterId	String	是	URL参数	指定的集群ID

响应头域

除公共头域外，无其它特殊头域。

响应参数

参数名	参数类型	参数描述
instanceGroups	List<InstanceGroup>	返回的实例组列表

请求示例

```
 GET /v1/cluster/0ce4f730-4af2-4f37-8fa2-b14f2f44e50e/instanceGroup HTTP/1.1
accept-encoding: gzip, deflate
x-bce-date: 2022-01-26T13:02:00Z
host: bmr.bj.baidubce.com
connection: keep-alive
accept: */*
content-type: application/json
authorization: bce-auth-v1/46bd9968a6194b4bbdf0341f2286ccce/2022-01-26T13:02:00Z/1800/host;x-bce-
date/994014d96b0eb26578e039fa053a4f9003425da4bfedf33f4790882fb4c54903
```

响应示例

```
 HTTP/1.1 200 OK
Transfer-Encoding: chunked
x-bce-request-id: 73c4e74c-3101-4a00-bf44-fe246959c05e
Cache-Control: no-cache
Server: BWS
Date: Wed, 26 Jan 2022 13:02:01 GMT
Content-Type: application/json;charset=UTF-8
{
    "instanceGroups": [
        {
            "id": "04ef4f35-2d16-4e5b-9944-bd4003bd8139",
            "name": "ngca0712f-master",
            "type": "Master",
            "spec": "bcc.g3.c4m16",
            "cpu": 4,
            "memory": 16,
            "isSpot": false,
            "bidModel": "",
            "bidPrice": "",
            "cds": [
                {
                    "sizeInGB": 50,
                    "mediumType": "ssd"
                }
            ],
            "diskType": "",
            "localDiskSize": 0,
            "rootDiskSizeInGB": 40,
            "rootDiskMediumType": "ssd",
            "requestedInstanceCount": 0,
            "totalInstanceCount": 0,
            "runningInstanceCount": 1,
            "maxCount": 2,
            "minCount": 1,
            "canExpand": 0,
            "canShrink": 0,
            "code": "",
            "message": ""
        },
        {
            "id": "9fd3f4fe-85cc-459d-8548-214beffaa7f3",
            "name": "ng138e063-core",
            "type": "Core",
            "spec": "bcc.g3.c4m16",
            "cpu": 4,
            "memory": 16,
            "isSpot": false,
            "bidModel": "",
            "bidPrice": "",
            "cds": [
                {
                    "sizeInGB": 50,
                    "mediumType": "ssd"
                }
            ],
            "diskType": "",
            "localDiskSize": 0,
            "rootDiskSizeInGB": 40,
            "rootDiskMediumType": "ssd",
            "requestedInstanceCount": 0,
            "totalInstanceCount": 0,
            "runningInstanceCount": 3,
            "maxCount": 200,
            "minCount": 2,
            "canExpand": 1,
            "canShrink": 0,
            "state": "Initializing",
            "code": "",
            "message": ""
        }
    ]
}
```

修改实例组配置

接口描述

修改指定集群的实例组配置

权限说明

请求发起人需要具有合法的AccessKeyID和SecretAccessKey才能发起请求，请参考 鉴权认证。

注意事项

如果请求中没有用户验证信息（即匿名访问），返回`403 Forbidden`，错误信息：`AccessDenied`。

请求结构

```
 PUT /v{version}/cluster/{clusterId}/instanceGroup?clientToken={clientToken} HTTP/1.1
accept-encoding: gzip, deflate
x-bce-date: {utc-date-string}
host: bmr.bj.baidubce.com
connection: keep-alive
accept: */*
content-type: application/json
authorization: {bce-authorization-string}
{
    "instanceGroups": [
        {
            "id": "cefb0a10-6028-467b-8dd5-d71d3cf74fcb",
            "instanceCount": 4
        },
        {
            "id": "0575111f-a702-434c-b463-8991b5fb4552",
            "instanceCount": 2
        }
    ]
}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名	参数类型	是否必须	参数位置	参数描述
version	String	是	URL参数	API版本号，当前取值1
clusterId	String	是	URL参数	待修改的集群ID，只有状态为RUNNING或WAITING的集群才能进行该操作
clientToken	String	否	Query参数	幂等性Token，是一个长度不超过64位的ASCII字符
instanceGroups	List<ModifyInstanceGroupConfig>	是	RequestBody参数	待修改实例组配置，可省略无须修改的实例组

响应头域

除公共头域外，无其它特殊头域。

响应参数

无参数

请求示例

```
PUT /v1/cluster/f5fa683d-4cf4-46b6-b5ab-bb2c0340c970/instanceGroup?clientToken=bf31b98c-5e41-4838-9830-9be700de5a20 HTTP/1.1
accept-encoding: gzip, deflate
x-bce-date: 2022-01-26T13:02:00Z
host: bmr.bj.baidubce.com
connection: keep-alive
accept: */*
content-type: application/json
authorization: bce-auth-v1/46bd9968a6194b4bbdf0341f2286ccce/2022-01-26T13:02:00Z/1800/host;x-bce-
date/994014d96b0eb26578e039fa053a4f9003425da4bfedf33f4790882fb4c54903
{
    "instanceGroups": [
        {
            "id": "cefb0a10-6028-467b-8dd5-d71d3cf74fcb",
            "instanceCount": 4
        },
        {
            "id": "0575111f-a702-434c-b463-8991b5fb4552",
            "instanceCount": 2
        }
    ]
}
```

响应示例

```
HTTP/1.1 204 NO_CONTENT
Transfer-Encoding: chunked
x-bce-request-id: 73c4e74c-3101-4a00-bf44-fe246959c05e
Cache-Control: no-cache
Server: BWS
Date: Wed, 26 Jan 2022 13:02:01 GMT
Content-Type: application/json;charset=UTF-8
```

实例操作接口

查询实例列表

接口描述

查询指定集群和实例组的实例

权限说明

请求发起人需要具有合法的AccessKeyID和SecretAccessKey才能发起请求，请参考 鉴权认证。

注意事项

如果请求中没有用户验证信息（即匿名访问），返回`403 Forbidden`，错误信息：`AccessDenied`。

请求结构

```
GET /v{version}/cluster/{clusterId}/instanceGroup/{instanceGroupId}/instance HTTP/1.1
accept-encoding: gzip, deflate
x-bce-date: {utc-date-string}
host: bmr.bj.baidubce.com
connection: keep-alive
accept: */*
content-type: application/json
authorization: {bce-authorization-string}
```

请求头域

除公共头域外，无其它特殊头域。

请求参数

参数名	参数类型	是否必须	参数位置	参数描述
version	String	是	URL参数	API版本号，当前取值1
clusterId	String	是	URL参数	指定的集群ID
instanceGroupId	String	是	URL参数	指定的实例组ID

响应头域

除公共头域外，无其它特殊头域。

响应参数

参数名	参数类型	参数描述
instances	List<Instance>	返回的实例列表

请求示例

```
GET /v1/cluster/0ce4f730-4af2-4f37-8fa2-b14f2f44e50e/instanceGroup/dabc21c6-41f8-4b23-98f2-3415e5886a5f/instance HTTP/1.1
accept-encoding: gzip, deflate
x-bce-date: 2022-01-26T13:02:00Z
host: bmr.bj.baidubce.com
connection: keep-alive
accept: */*
content-type: application/json
authorization: bce-auth-v1/46bd9968a6194b4bbdf0341f2286
```

响应示例

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
x-bce-request-id: 73c4e74c-3101-4a00-bf44-fe246959c05e
Cache-Control: no-cache
Server: BWS
Date: Wed, 26 Jan 2022 13:02:01 GMT
Content-Type: application/json;charset=UTF-8
{
 "instances": [
     {
         "id": "0413bbc7-55c5-4567-46ca-5eaa84e1700d",
         "bccInstanceId": "i-SrbDLXLH",
         "instanceName": "bmr-core-111ca89-1",
         "status": {
             "state": "Stopped",
             "code": "",
             "message": "",
             "creationDateTime": "2022-01-24T11:20:52Z",
             "endDateTime": "0001-01-01T00:00:00Z"
         },
         "privateDnsName": "",
         "privateIpAddress": "172.18.0.15",
         "publicDnsName": "",
         "publicIpAddress": "",
         "sshPort": 22
     },
     {
         "id": "8402adb7-3f96-49ab-5573-a8a8fc21f8e7",
         "bccInstanceId": "i-kmeHJ4tX",
         "instanceName": "bmr-core-111ca89-2",
         "status": {
             "state": "Stopped",
             "code": "",
             "message": "",
             "creationDateTime": "2022-01-24T11:20:52Z",
             "endDateTime": "0001-01-01T00:00:00Z"
         },
         "privateDnsName": "",
         "privateIpAddress": "172.18.0.14",
         "publicDnsName": "",
         "publicIpAddress": "",
         "sshPort": 22
     }
   ]
}
```

数据类型

Model对象定义

Cluster

参数名	参数类型	参数描述
applications	List<Application>	需要安装的组件信息（如hive，pig，hbase）
id	String	集群ID，是一个定长字符串，只包含大小写字母、数字、连字号（-）和下划线（_）
imageType	String	集群类型，参考BMR集群
imageVersion	String	集群版本，参考BMR集群
name	String	集群名称
payType	String	支付方式
enableAutoScale	String	是否支持自动扩缩容
status	ClusterStatus	集群状态信息
vpcId	String	集群所选 vpc 信息
subnetId	String	集群所选子网信息
securityGroupId	String	集群所选安全组
createTime	Date	创建时间
tags	List	集群标签

Application

参数名	参数类型	参数描述
name	String	组件名称（如hive，spark，hbase，hue），参考BMR集群
properties	Object	组件属性，具体内容由不同组件决定
version	String	组件版本，参考BMR集群

组件属性

hive组件的properties定义如下：

参数名	参数类型	参数描述
metastore	String	hive metastore启用方式。default表示使用集群内部的metastore服务，mysql表示使用用户自行提供的mysql服务。
host	String	mysql服务器地址，当metastore为default时，该域不出现
port	String	mysql服务器端口，当metastore为default时，该域不出现
database	String	metastore使用的数据库名称，当metastore为default时，该域不出现
userName	String	用于连接mysql服务器的用户名，当metastore为default时，该域不出现hbase组件、pig组件、hue组件没有properties域

ClusterStatus

参数名	参数类型	参数描述
creationDateTime	String	集群创建的时间，符合日期时间格式约束
endDateTime	String	集群停止的时间，符合日期时间格式约束
readyDateTime	String	集群完成部署的时间，符合日期时间格式约束
state	String	集群状态
message	String	集群错误信息
code	String	集群错误码
orderStatus	String	订单状态
expireDateTime	Date	过期时间
expireDates	Date	过期日期

Step

参数名	参数类型	参数描述
actionOnFailure	String	作业失败策略
id	String	作业ID，是一个定长字符串，只包含大小写字母、数字、连字号（-）和下划线（_）
name	String	作业名称
properties	Object	作业描述，具体内容由作业类型决定
status	StepStatus	作业状态信息
type	String	作业类型
logUri	String	作业日志路径
stderr	String	作业error信息
stdout	String	作业输出
syslog	String	作业log输出
clusterId	String	集群id

StepStatus

参数名	参数类型	参数描述
creationDateTime	String	作业提交的时间，符合日期时间格式约束
endDateTime	String	作业结束的时间，符合日期时间格式约束
startDateTime	String	作业开始执行的时间，符合日期时间格式约束
state	String	作业状态
code	String	作业错误码
message	String	作业错误信息

枚举类型定义

实例组类型

编码	描述
Master	主实例组
Core	核心实例组，适合存储
Task	任务实例组，适合扩展
Client	客户端实例组，适合扩展
AutoScaling	弹性伸缩实例组，适合扩展

集群状态

编码	描述
Starting	启动中
Running	运行中
Resizing	调整中
Terminating	释放中
Terminated	已释放
TerminatedWithError	已释放但有错误
Suspending	停服中
Suspended	已停服
Resuming	恢复中

实例状态

编码	描述
Configuring	配置中
Running	运行中
Suspending	挂起中
Suspended	已挂起
Resuming	恢复中
Deleting	释放中
Terminated	已释放

作业类型

编码	描述
Streaming	MapReduce Streaming程序
Hive	HQL程序
Pig	pig作业
Java	用户自定义Java程序
Spark	Spark作业

作业失败策略

编码	描述
Continue	继续执行其他作业
TerminateCluster	作业失败后释放集群
CancelAndWait	作业失败后取消其他尚未执行的作业并将集群置为空闲状态

作业状态

编码	描述
Pending	等待中
Running	运行中
Completed	已完成
Cancelled	已取消
Failed	已失败

BMR集群

集群类型	集群版本	支持的组件
BMR	1.0.0	hadoop 2.7 spark 1.6.0 hive 1.2.0 pig 0.15.1 hue 3.10.0 hbase 1.1.2
BMR	1.1.0	hadoop 2.7 hive 1.2.0 pig 0.15.1 hue 3.10.0 spark 2.1.0 hbase 1.1.2
BMR	1.2.0	hadoop 2.7 hive 1.2.0 pig 0.15.1 hue 3.10.0 spark 2.1.0 hbase 1.1.2 ranger 0.5.0
BMR	2.0.0	hadoop 3.1 hive 3.1.0 spark 2.3.2 pig 0.17.0 hue 4.4.0 presto 0.219 hbase 2.0.2 azkaban 3.58.0 zeppelin 0.8.0 flink 1.8.2 druid 0.12.1 impala 3.2.0
BMR	2.1.0	hadoop 3.1 hive 3.1.0 spark 2.4.2 pig 0.17.0 hue 4.4.0 presto 0.219 hbase 2.0.2 azkaban 3.58.0 zeppelin 0.8.0 flink 1.8.2 druid 0.12.1 impala 3.2.0 kafka 2.0.1

实例规格

实例规格请参见实例规格

云磁盘类型

编码	类型	描述
ssd	高性能云磁盘	系统盘和数据盘支持选择该类型
premium_ssd	通用型SSD盘	系统盘和数据盘支持该类型
enhanced_ssd_pl1	增强型SSD盘	系统盘和数据盘支持该类型

InstanceGroupConfig数据结构

参数名	参数类型	是否必须	参数描述
instanceCount	Int	是	实例组中的虚拟机数量
instanceType	String	是	实例组中的虚拟机类型，参考实例规格
type	String	是	Master、Core或Task，参考实例组类型
name	String	否	实例组名称，默认为空
rootDiskSizeInGB	int	是	系统盘磁盘容量大小
rootDiskMediumType	String	是	系统盘磁盘介质类型
cds	List<CdsItem>	否	数据盘云磁盘列表

> **注意**：instanceGroups域应包括三个实例组，type分别为Master、Core和Task，Master实例组instanceCount为1，Core实例组instanceCount不小于2且不大于20，Task实例组instanceCount不大于20。

ApplicationConfig数据结构

参数名	参数类型	是否必须	参数描述
name	String	是	组件名称（如hive，pig，hbase, hue），参考BMR集群
version	String	是	组件版本，参考BMR集群
properties	Object	否	组件属性，具体内容由不同组件决定

组件属性

hive组件的properties定义如下：

参数名	参数类型	是否必须	参数描述
metastore	String	否	hive metastore启用方式。若使用集群内部的metastore服务则应设为default，若用户自行提供mysql服务则应设为mysql。默认为default
host	String	否	mysql服务器地址，若metastore选用mysql，则必须提供
port	String	否	mysql服务器端口，若metastore选用mysql，则必须提供
database	String	否	metastore使用的数据库名称，若metastore选用mysql，则必须提供
userName	String	否	用于连接mysql服务器的用户名，若metastore选用mysql，则必须提供
password	String	否	用于连接mysql服务器的密码，若metastore选用mysql，则必须提供

StepConfig数据结构

参数名	参数类型	是否必须	参数描述
actionOnFailure	String	是	作业失败策略
properties	Object	是	作业描述，具体内容由作业类型决定
type	String	是	作业类型
name	String	否	作业名称，默认为my-step
additionalFiles	List<AdditionalFile>	否	额外的文件

AdditionalFile数据结构

参数名	参数类型	是否必须	参数描述
remote	String	否	远程文件
local	String	否	本地文件

作业描述

Streaming作业的properties定义如下：

参数名	参数类型	是否必须	参数描述
input	String	是	输入路径
mapper	String	是	mapper程序
output	String	是	输出路径
arguments	String	否	hadoop streaming执行参数
reducer	String	否	reducer程序

Hive作业的properties定义如下：

参数名	参数类型	是否必须	参数描述
script	String	是	hql脚本在BOS上的存储路径
arguments	String	否	hql脚本执行参数
input	String	否	预定义的输入路径，可在hql脚本中通过${INPUT}来引用
output	String	否	预定义的输出路径，可在hql脚本中通过${OUTPUT}来引用

Pig作业的properties定义如下：

参数名	参数类型	是否必须	参数描述
script	String	是	pig脚本在BOS上的存储路径
arguments	String	否	pig脚本执行参数
input	String	否	预定义的输入路径，可在pig脚本中通过${INPUT}来引用
output	String	否	预定义的输出路径，可在pig脚本中通过${OUTPUT}来引用

Java作业的properties定义如下：

参数名	参数类型	是否必须	参数描述
jar	String	是	自定义java程序在BOS上的存储路径
mainClass	String	是	自定义java程序的主入口
arguments	String	否	自定义java程序的执行参数

Spark作业的properties定义如下：

参数名	参数类型	是否必须	参数描述
jar	String	是	自定义程序在BOS上的存储路径
submitOptions	String	是	spark-submit脚本的参数
arguments	String	否	自定义程序的执行参数

ModifyInstanceGroupConfig数据结构

参数名	参数类型	是否必须	参数描述
id	String	是	实例组id
instanceCount	int	是	修改后实例组内实例数

InstanceGroup数据结构

参数名	参数类型	参数描述
id	String	实例组ID
name	String	实例组名称
type	String	Master、Core、Task、Client 或 AutoScaling，参考实例组类型
spec	String	实例规格
cpu	Int	cpu 核数
Memory	Int	内存大小
isSpot	Boolean	是否为抢占实例
bidModel	String	竞价类型
bidPrice	String	竞价实例价格
cds	CDS	云磁盘
diskType	String	磁盘类型
localDiskSize	Int	本地盘大小
rootDiskSizeInGB	Int	系统盘大小
rootDiskMediumType	String	系统盘介质类型
totalInstanceCount	Int	总实例数量
requestedInstanceCount	Int	请求实例数量
runningInstanceCount	Int	运行中的实例数量
maxCount	Int	最大允许的实例数量
minCount	Int	最小允许的实例数量
canExpand	Int	是否支持扩容，1表示支持，0表示不支持
canShrink	Int	是否支持缩容，1表示支持，0表示不支持

Instance数据结构

参数名	参数类型	参数描述
id	String	BMR实例ID
bccInstanceId	String	BCC实例ID
instanceName	String	B实例名称
status	InstanceStatus	实例状态
privateIpAddress	String	实例内网ip
publicIpAddress	String	实例公网ip

InstanceStatus

参数名	参数类型	参数描述
creationDateTime	String	实例创建时间，符合日期时间格式约束
endDateTime	String	实例停止时间，符合日期时间格式约束
state	String	实例状态

CdsItem数据结构

参数名	参数类型	参数描述
sizeInGB	Int	磁盘大小
mediumType	String	磁盘介质类型

版本更新记录

2019-04-11

- 将旧版本的api 迁移到新版

Java-SDK

概述

本文档主要介绍BMR Java SDK的安装和使用。在使用本文档之前，您需要先了解BMR的一些基本知识，并已经开通了BMR服务。若您还不了解BMR，可以参考产

品描述和操作指南。

快速入门

1. 初始化一个BmrClient。 BmrClient是与BMR服务交互的客户端，所有BMR操作都是通过BmrClient完成的。用户可以参考 新建BmrClient，完成初始化客户端的操作。
2. 新建一个BMR(集群)。

   用户创建一个BMR集群，用于提交并运行指定的作业。在提交作业之前，必须先创建出一个集群。创建集群的请求成功后，将返回新集群的ID，可用于后续对集群进行相关操作。

   用户在创建集群时，需要为集群指定虚拟机实例采用的镜像类型、镜像版本号和虚拟机实例组配置。另外，可选的配置项包括集群的名字、需要安装的组件信息(如Hive，Pig，HBase)、上传运行日志的BOS路径以及需要运行的作业信息。用户可以参考新建集群，进行具体的集群配置操作。
3. 查看集群的状态信息。 用户通过集群ID可以查看集群的状态信息。创建集群需要等待一定时间后集群成为可用状态，此时提交至集群的作业开始被调度执行。
4. 查看集群实例组信息。 用户通过集群ID可以查看集群的实例组信息，包括实例组类型、实例组实例数等。
5. 修改实例组配置。 用户通过集群ID和实例组ID修改CORE或TASK实例组配置。
6. 查看实例信息。 用户通过集群ID和实例组ID查看实例信息。
7. 添加一个或多个Step(作业)。 向指定集群添加一个或多个作业任务。集群创建成功后，用户可以根据集群已安装的应用来添加不同类型的作业，包括Custom Jar、Streaming、Hive、Pig类型的作业。添加作业请求成功后，将返回一个包含所有新作业ID的数组，作业ID可用于后续对作业进行相关操作。
8. 查看作业的运行状态信息。 用户通过作业ID可以查看作业的运行状态信息。
9. 终止集群。 在结束作业的运行后，用户可以通过集群ID发送终止集群的请求。终止集群将释放集群的虚拟机实例，结束计费。

安装SDK工具包

运行环境

Java SDK工具包可在jdk6、jdk7、jdk8环境下运行。

安装步骤

1. 在Java SDK下载Java SDK压缩工具包。
2. 将下载的`bce-java-sdk-version.zip`解压后，复制到工程文件夹中。
3. 在Eclipse右键“工程 -> Properties -> Java Build Path -> Add JARs”。
4. 添加SDK工具包`lib/bce-java-sdk-version.jar`和第三方依赖工具包`third-party/*.jar`。

   其中，`version`为版本号。

SDK目录结构

```
com.baidubce
    ├─── auth                          //BCE签名相关类
    ├─── http                          //BCE的Http通信相关类
    ├─── internal                      //SDK内部类
    ├─── model                         //BCE公用model类
    ├─── services
    │      └─── bmr                    //BMR服务相关类
    │          ├─── model              //BMR内部model，如Request或Response
    │          └─── BmrClient.class    //BMR客户端入口类
    ├─── util                          //BCE公用工具类
    ├─── BceClientConfiguration.class  //对BCE的HttpClient的配置
    ├─── BceClientException.class      //BCE客户端的异常类
    ├─── BceServiceException.class     //与BCE服务端交互后的异常类
    ├─── ErrorCode.class               //BCE通用的错误码
    └─── Region.class                  //BCE提供服务的区域
```

Instance（实例）

查询实例列表

如下代码可查询指定集群和实例组的实例：

```
public void listInstances(BmrClient client, String clusterId, String instanceGroupId) {
    try {
        ListInstancesResponse response = client.listInstances(clusterId, instanceGroupId);
        for (Instance instance : response.getIntances()) {
            System.out.println(instance.getId() + "," + instance.getPrivateIpAddress()
                    + "," + instance.getPublicIpAddress());
        }
    } catch (BceServiceException e) {
        System.out.println("List instance failed: " + e.getErrorMessage());
    }
}
```

请求返回的ListInstancesResponse对象包含了相关的实例对象数组List， 获取实例对象数组的方法为response.getInstances()。实例对象Instance的属性包括了内网IP、公网IP等配置信息，每个属性均有对应的getter访问器方法。

```
public class Instance {
    private String id;
    private String bccInstanceId;
    private String instanceName;
    private String privateIpAddress;
    private String publicIpAddress;
    private Status status;
    private String privateDnsName;
    private String publicDnsName;
    private int sshPort;
}
```

InstanceGroup（实例组）

查询实例组列表

如下代码可根据集群ID获取其实例组信息：

```
public void listInstanceGroups(BmrClient client, String clusterId) {
    try {
        // 方法 1. 罗列指定集群ID相关的实例组
        ListInstanceGroupsResponse response1 = client.listInstanceGroups(clusterId);
        // 方法 2. 自定义ListInstanceGroupsRequest对象的查询请求
        ListInstanceGroupsResponse response2 = client.listInstanceGroups(new ListInstanceGroupsRequest().withClusterId(clusterId));
        for (InstanceGroup instanceGroup: response2.getInstanceGroups()) {
            System.out.println(instanceGroup.getId() + "," +  instanceGroup.getType()
                    + "," + instanceGroup.getType() + "," + instanceGroup.getRequestedInstanceCount()
                    + "," + instanceGroup.getRootDiskSizeInGB() + "," + instanceGroup.getRootDiskMediumType());
            for (CdsItem cdsItem: instanceGroup.getCds()){
                System.out.println("cdsItem " + cdsItem.getSizeInGB() + "," + cdsItem.getMediumType());
            }
        }
    } catch (BceServiceException e){
        System.out.println("List instance group failed: " + e.getErrorMessage());
    }
}
```

请求返回的ListInstanceGroupsResponse对象包含了相关的实例组对象数组List， 获取实例组对象数组的方法为response.getInstanceGroups()。实例组对象InstanceGroup的属性包括了实例组类型、实例数、系统盘容量大小等配置信息，每个属性均有对应的getter访问器方法。其中系统盘存储介质类型支持ssd(SSD云磁盘)、premium_ssd(高性能云磁盘)两种。

```
public class InstanceGroup {
    private String id;                          //实例组ID
    private String instanceType;                    //实例组套餐类型
    private String name;                        //实例组名称
    private String type;                        //实例组类型
    private int requestedInstanceCount;             //请求的实例个数
    private int rootDiskSizeInGB;                   //系统盘容量大小
    private String rootDiskMediumType;              //系统盘存储介质类型
    private List<CdsItem> cds;                  //云磁盘信息列表
}
```

云盘对象CdsItem的属性包括云盘容量大小和云盘存储介质，每个属性均有对应的getter访问器方法，其中云存储介质支持premium_ssd（SSD云磁盘）,ssd（高性能云磁盘）两种类型。

```
public class CdsItem {
  private int sizeInGB;                    //云磁盘容量大小
  private String mediumType;               //云磁盘存储介质
}
```

修改实例组配置

BMR支持对CORE、TASK节点进行配置变更，示例代码如下：

```
public void modifyInstanceGroups(BmrClient bmrClient, String clusterId, String coreGroupId, String taskGroupId) {
  // 可省略无须修改的实例组
  List<ModifyInstanceGroupConfig> instanceGroups = new ArrayList<ModifyInstanceGroupConfig>();
  // core group
  ModifyInstanceGroupConfig coreGroup = new ModifyInstanceGroupConfig();
  coreGroup.setId(coreGroupId);
  coreGroup.setInstanceCount(4);
  instanceGroups.add(coreGroup);
  // task group
  ModifyInstanceGroupConfig taskGroup = new ModifyInstanceGroupConfig();
  taskGroup.setId(taskGroupId);
  taskGroup.setInstanceCount(2);
  instanceGroups.add(taskGroup);

  String clientToken = UUID.randomUUID().toString();
  try {
     bmrClient.modifyInstanceGroups(
           new ModifyInstanceGroupsRequest()
                 .withClusterId(clusterId)
                 .withInstanceGroups(instanceGroups)
                 .withClientToken(clientToken)
     );
  } catch (BceServiceException e) {
     System.out.println("Modify instance groups failed: " + e.getErrorMessage());
  }
}
```

Cluster（集群）

新建cluster

如下代码可以新建一个集群，集群包含1个master节点和2个core节点，且安装了Hive、Pig、HBase应用。请注意：参考下面样例代码时，需要修改相关的BOS路径为您的账户可用的BOS路径，包括withLogUri函数参数、HBase应用配置的withBackupLocation函数参数。

新建集群可以通过配置CreateClusterRequest对象的clientToken属性来保证创建请求的幂等性。clientToken是一个长度不超过64位的ASCII字符串，配置CreateClusterRequest对象的clientToken方法是：`createClusterRequest.withClientToken(clientToken)`。

请求返回的CreateClusterResponse对象包含了新创建出集群的集群ID，获取方法为`response.getClusterId()`。

```
public void createCluster(BmrClient bmrClient) {
  // 发送创建集群请求
  String clusterId = null;
  try {
     CreateClusterResponse response = bmrClient.createCluster(
           new CreateClusterRequest()
                 .withName("java-sdk-cluster")
                 .withImageType("hadoop")
                 .withImageVersion("0.1.0")
                 .withAutoTerminate(false)
                 .withLogUri("bos://path/to/logUri/")
                 .withServiceHaEnabled(false)
                 .withSafeModeEnabled(false)
                 .withInstanceGroup(new InstanceGroupConfig()
                       .withName("ig-master")
                       .withType("Master")
                       .withInstanceType("bmr.g1.2xlarge")
                       .withInstanceCount(1)
                       .withRootDiskSizeInGB(50)
                       .withRootDiskMediumType("ssd")
                       .withCds(new CdsItem()
                          .withSizeInGB(100)
                          .withMediumType("ssd")))
                 .withInstanceGroup(new InstanceGroupConfig()
                       .withName("ig-core")
                       .withType("Core")
                       .withInstanceType("bmr.gh1.large")
                       .withInstanceCount(2)
                       .withRootDiskSizeInGB(50)
                       .withRootDiskMediumType("ssd"))
                 .withApplication(new PigApplicationConfig().withVersion("0.11.0"))
                 .withApplication(new HiveApplicationConfig().withVersion("0.13.0").withMetastore("default"))
                 .withApplication(new HBaseApplicationConfig()
                       .withVersion("0.98.0")
                       .withBackupEnabled(true)
                       .withBackupLocation("bos://tester01/hbase_backup")
                       .withBackupIntervalInMinutes(300)
                       .withBackupStartDatetime("2015-08-18T23:00:00Z"))
                 .withStep(new JavaStepConfig()
                       .withName("init-step")
                       .withActionOnFailure("Continue")
                       .withJar("bos://bmr/samples/mapreduce/libs/hadoop-mapreduce-examples.jar")
                       .withMainClass("org.apache.hadoop.examples.WordCount")
                       .withArguments("bos://bmr/samples/mapreduce/wordcount/hamlet.txt bos://tester01/out"))
     );
     // 获取新创建集群的集群ID。
     clusterId = response.getClusterId();
  } catch (BceServiceException e) {
     System.out.println("Create cluster failed: " + e.getErrorMessage());
  } catch (BceClientException e) {
     System.out.println(e.getMessage());
  }
}
```

售卖的套餐类型请参考https://cloud.baidu.com/doc/BMR/s/6jwvxw85z

列出全部cluster

如下代码可以罗列出属于请求调用者的所有集群，用户可以通过配置查询参数maxKeys来限制每次请求返回的集群数目，并且通过配置有效的查询参数marker来指定查询记录的起点。marker参数值是由BMR系统生成并返回的，因而初次查询请求中不需要配置该参数，它是在多次循环查询请求的后继请求中进行使用的。

```
 public void listClusters(BmrClient bmrClient) {
int maxKeys = 10;
  try {
    // 方法 1. 默认的查询请求
    ListClustersResponse response1 = bmrClient.listClusters();

    // 方法 2. 配置maxKeys的单次查询请求
    ListClustersResponse response2 = bmrClient.listClusters(maxKeys);
     // 方法 3. 配置maxKeys和marker参数的循环查询请求
    boolean isTruncated = true;
    int page = 0;
    String marker = null;
    while (isTruncated) {
      ListClustersResponse response3 = bmrClient.listClusters(marker, maxKeys);
      page++;
      System.out.format("Page %d: cluster count: %d\n", page, response3.getClusters().size());
      isTruncated = response3.isTruncated();
      marker = response3.getNextMarker();
    }

    // 方法 4. 自定义ListClustersResquest对象的查询请求
    ListClustersResponse response4 = bmrClient.listClusters(
          new ListClustersRequest().withMaxKeys(maxKeys)
    );

    // 输出各个集群的ID
    for (Cluster cluster : response4.getClusters()) {
       System.out.format("cluster id: %s\n", cluster.getId());
    }
  } catch (BceServiceException e) {
    System.out.println("List clusters failed: " + e.getErrorMessage());
  }
}
```

请求返回的ListClustersResponse对象包含了相关的集群对象数组List<Cluster>， 获取集群对象数组的方法为response.getClusters()。集群对象Cluster的属性包括了集群相关的配置信息，每个属性均有对应的getter访问器方法。

```
 public class Cluster {
  private String payType;                    //集群支付方式
  private boolean enableAutoScale;                 //集群是否支持自动扩缩容
  private Date createTime;                    //集群创建时间
  private List<Tag> tags;                    // 标签
  private List<Application> applications;              // 集群已安装的应用信息
  private String id;                        // 集群ID
  private String name;                      // 集群名称
  private String imageType;                     // 集群虚拟机实例的镜像类型
  private String imageVersion;                    // 集群虚拟机实例的镜像版本
  private ClusterStatus status;                   // 集群当前的状态信息
}

public class ClusterStatus {
private String code;                        // 集群创建失败的错误码
private String message;                       // 集群创建失败的错误信息
private String orderStatus;                     // 订单状态
private Date expireDateTime;                     // 过期时间
private int expireDates;                    // 过期日期

  private String state;                     // 集群状态字段
  private Date creationDateTime;                  // 集群创建时间
  private Date endDateTime;                   // 集群终止时间
  private Date readyDateTime;                   // 集群完成部署时间
}
```

查询指定的cluster

获取了集群ID后，可用如下代码查询指定集群的信息。

请求返回的GetClusterResponse对象包含了获取集群属性的getter访问器方法，可以直接调用response的访问器方法来获得目标集群的属性信息。

```
public void getCluster(BmrClient bmrClient, String clusterId) {
  try {
     // 方法 1. 查询对应ID的集群信息
     GetClusterResponse response1 = bmrClient.getCluster(clusterId);

     // 方法 2. 自定义GetClusterRequest对象的查询请求
     GetClusterResponse response2 = bmrClient.getCluster(
           new GetClusterRequest().withClusterId(clusterId)
     );
     // 输出集群的镜像类型
     System.out.println(response1.getImageType());
     // 输出集群当前的状态
     System.out.println(response2.getStatus().getState());
  } catch (BceServiceException e) {
     System.out.println("Describe cluster failed: " +  e.getErrorMessage());
  }
}
```

终止指定的cluster

如下代码可以终止指定的集群：

```
public void terminateCluster(BmrClient bmrClient, String clusterId) {
  try {
   // 方法 1. 终止对应ID的集群
     bmrClient.terminateCluster(clusterId);

     /*
     方法 2. 自定义TerminateClusterRequest对象的终止请求
     bmrClient.terminateCluster(
           new TerminateClusterRequest().withClusterId(clusterId)
     );
     */
  } catch (BceServiceException e) {
     System.out.println("Terminate cluster failed: " + e.getErrorMessage());
  }
}
```

Step（作业）

概述

作业是和集群相关联的资源，对作业的操作需要指定相关集群的ID。

添加steps

BMR支持多种类型的作业，不同类型的作业有不同的配置项。如下代码可向指定的hadoop类型的集群添加Custom Jar、Streaming、Hive、Pig作业。

添加作业可以通过配置AddStepsRequest对象的clientToken属性来保证创建请求的幂等性。clientToken是一个长度不超过64位的ASCII字符串，配置AddStepsRequest对象的clientToken方法是：`addStepsRequest.withClientToken(clientToken)`。

请求返回的AddStepsResponse对象包含了新创建作业ID的数组`List<String>`，获取方法为`response.getStepIds()`。

```
public void addSteps(BmrClient bmrClient, String clusterId) {
   List<StepConfig> steps = new ArrayList<StepConfig>();
  // Custom Jar作业
  steps.add(
        new JavaStepConfig()
              .withName("java-step")
              .withActionOnFailure("Continue")
              .withJar("bos://benchmark/hadoop/hadoop-mapreduce-examples.jar")
              .withMainClass("org.apache.hadoop.examples.WordCount")
              .withArguments("bos://path/to/input bos://path/to/java_output")
  );

  // Streaming作业
  steps.add(
        new StreamingStepConfig()
              .withName("streaming-step")
              .withActionOnFailure("Continue")
              .withInput("bos://path/to/input_streaming")
              .withMapper("cat")
              .withOutput("bos://path/to/output_streaming")
  );
```

```
    List<AdditionalFile> additionalFiles = new ArrayList<AdditionalFile>();
    additionalFiles.add(new AdditionalFile().withRemote("bos://path/to/testA.jar").withLocal("testB.jar"));

    // 使用附加文件的Streaming作业
    steps.add(
            new StreamingStepConfig()
                    .withName("streaming-step2")
                    .withActionOnFailure("Continue")
                    .withInput("bos://path/to/input_streaming2")
                    .withMapper("cat")
                    .withReducer("cat")
                    .withOutput("bos://path/to/output_streaming2")
                    .withAdditionalFiles(additionalFiles)
    );

    // Hive作业
    steps.add(
            new HiveStepConfig()
                    .withName("hive-step")
                    .withActionOnFailure("Continue")
                    .withScript("bos://path/to/hive/hql/hive_src.hql")
                    .withInput("bos://path/to/hive/data/hive_src.data")
                    .withOutput("bos://path/to/output_hive")
                    .withArguments("--hivevar LOCAT=bos://chy3/hive/tables/src")
    );

    // Pig作业
    steps.add(
            new PigStepConfig()
                    .withName("pig-step")
                    .withActionOnFailure("Continue")
                    .withScript("bos://path/to/pig/script/pig_grep.pig")
                    .withInput("bos://path/to/pig/data/pig_grep.data")
                    .withOutput("bos://path/to/output_pig")
    );

    //Spark作业
    steps.add(
            new SparkStepConfig()
                    .withName("spark-step")
                    .withActionOnFailure("Continue")
                    .withJar("bos://bmr-public-bj/sample/spark-1.0-SNAPSHOT.jar")
                    .withSubmitOptions("--class com.baidu.cloud.bmr.spark.AccessLogAnalyzer")
                    .withArguments("bos://bmr-public-bj/data/log/accesslog-1k.log bos://tester01/sdk/output/out")
    );


try {
    AddStepsResponse response = bmrClient.addSteps(
            new AddStepsRequest().withClusterId(clusterId)
                    .withSteps(steps)
    );
    // 输出各个添加的作业ID
    for (String stepId : response.getStepIds()) {
       System.out.println(stepId);
    }
} catch (BceServiceException e) {
    System.out.println("Add steps failed: " + e.getErrorMessage());
} catch (BceClientException e) {
    System.out.println(e.getMessage());
}
}
```

列出全部steps

如下代码可以罗列出指定集群上的全部作业，用户可以通过配置查询参数pageNo和pageSize来限制每次请求返回的作业数目和查询记录的起点。

```
public void listSteps(BmrClient bmrClient, String clusterId) {
  int maxKeys = 10;
  try {
       // 罗列指定集群ID相关的作业
    ListStepsRequest request = new ListStepsRequest().withClusterId(clusterId);
    ListStepsResponse response = bmrClient.listSteps(request);

    // 输出各个作业的状态
    for (Step step : response.getSteps()) {
      System.out.println(step.getStatus().getState());
    }
  } catch (BceServiceException e) {
    System.out.println("List steps failed: " + e.getErrorMessage());
  }
}
```

请求返回的ListStepsResponse对象包含了相关的集群对象数组List<Step>， 获取集群对象数组的方法为response.getSteps()。作业对象Step的属性包括了作业相关的配置信息，每个属性均有对应的getter访问器方法。

```
public class Step {
  private String id;                          // 作业ID
  private String actionOnFailure;               // 作业失败策略
  private String type;                        // 作业类型
  private Map<String, String> properties;         // 作业描述
  private String name;                        // 作业名称
  private StepStatus status;                    // 作业状态
}

public class StepStatus {
  private String createDateTime;                // 作业提交时间
  private String endDateTime;                   // 作业结束时间
  private String startDateTime;                 // 作业开始执行的时间
  private String state;                       // 作业状态字段
}
```

查询指定的step

如下代码可以查看指定作业的信息：

请求返回的GetStepResponse对象包含了获取作业属性的getter访问器方法，可以直接调用response的访问器方法来获得目标作业的属性信息。

```
public void getStep(BmrClient bmrClient, String clusterId, String stepId) {
  try {
    // 方法 1. 查询指定集群ID、作业ID对应作业的信息
    GetStepResponse response1 = bmrClient.getStep(clusterId, stepId);

    // 方法 2. 自定义GetStepRequest对象的查询请求
    GetStepResponse response2 = bmrClient.getStep(
        new GetStepRequest().withClusterId(clusterId).withStepId(stepId)
    );
    // 输出作业的状态信息
    System.out.println(response1.getStatus().getState());
  } catch (BceServiceException e) {
    System.out.println("Describe steps failed: " + e.getErrorMessage());
  }
}
```

日志

概述

BMR Java SDK发布版本中增加了logback作为slf4j的实现，如果用户没有自己的实现可以直接用，如果工程中有其他的如log4j则可以替代。

默认日志

如果用户使用默认的logback，则需要配置logback.xml到classpath中。如果没有这个配置文件，日志级别默认为DEBUG。

```
<configuration>
  <property name="LOG_HOME" value="./log/"/>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <!-- encoders are assigned the type
         ch.qos.logback.classic.encoder.PatternLayoutEncoder by default -->
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>

  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <FileNamePattern>${LOG_HOME}/BosUnitTest.%d{yyyy-MM-dd}.log</FileNamePattern>
      <MaxHistory>30</MaxHistory>
    </rollingPolicy>
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
      <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{50} - %msg%n</pattern>
    </encoder>
    <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
      <MaxFileSize>10MB</MaxFileSize>
    </triggeringPolicy>
  </appender>

  <root level="info">
    <appender-ref ref="STDOUT"/>
    <appender-ref ref="FILE"/>
  </root>
</configuration>
```

自有日志模块

若用户使用自己的日志实现模块，例如项目依赖于Maven，则可以类似下面的配置到pom.xml中来去除logback。

```
<?xml version="1.0" encoding="utf-8"?>

<dependency>
  <groupId>com.baidubce</groupId>
  <artifactId>bce-java-sdk</artifactId>
  <version>${bce.sdk.version}</version>
  <exclusions>
    <exclusion>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-classic</artifactId>
    </exclusion>
    <exclusion>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-core</artifactId>
    </exclusion>
    <exclusion>
      <groupId>org.slf4j</groupId>
      <artifactId>jcl-over-slf4j</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

BmrClient

新建BmrClient

BmrClient是BMR服务的Java客户端，为调用者与BMR服务进行交互提供一系列的方法。

用户可以参考如下代码新建一个BmrClient:

```
public class Sample {
  public static void main(String[] args) {
      String ACCESS_KEY_ID = "your-access-key-id";                // 用户的Access Key ID
      String SECRET_ACCESS_KEY = "your-secret-access-key";        // 用户的Secret Access Key

      // 初始化一个BmrClient
      BceClientConfiguration config = new BceClientConfiguration();
      config.setCredentials(new DefaultBceCredentials(ACCESS_KEY_ID, SECRET_ACCESS_KEY));
      BmrClient client = new BmrClient(config);
  }
}
```

在上面代码中，变量ACCESS_KEY_ID与SECRET_ACCESS_KEY是由系统分配给用户的，均为 字符串，用于标识用户，为访问BMR做签名验证。其中ACCESS_KEY_ID对应控制台中的“Ac cess Key ID”，SECRET_ACCESS_KEY对应控制台中的“Access Key Secret”，获取方式 请参考《管理ACCESSKEY》。

上面的方式使用默认域名作为BMR的服务地址，可能出现不能访问的情况。目前BMR产品使用独占服务地址，用户可以通过传入endpoint参数来指定。

```
 String ACCESS_KEY_ID = "your-access-key-id";                // 用户的Access Key ID
String SECRET_ACCESS_KEY = "your-secret-access-key";         // 用户的Secret Access Key
String ENDPOINT = "http://bmr.bj.baidubce.com";              // 用户指定BMR的服务域名

BceClientConfiguration config = new BceClientConfiguration();
config.setCredentials(new DefaultBceCredentials(ACCESS_KEY_ID,SECRET_ACCESS_KEY));
config.setEndpoint(ENDPOINT);
BmrClient client = new BmrClient(config);
```

> **注意：**ENDPOINT参数只能用指定的包含Region的域名来进行定义，eg：北京Region的endpoint的域名是http://bmr.bj.baidubce.com，广州Region的endpoint的域名是http://bmr.gz.baidubce.com，更多服务域名描述请参考服务域名。

配置BmrClient

如果用户需要配置BmrClient的一些细节的参数，可以在构造BmrClient的时候传入BceClientConfiguration对象。 BceClientConfiguration是BMR服务的配置类，可以为客户端配置代理，最大连接数等参数。

使用代理

下面一段代码可以让客户端使用代理访问BMR服务：

```
 String ACCESS_KEY_ID = "your-access-key-id";                // 用户的Access Key ID
String SECRET_ACCESS_KEY = "your-secret-access-key";         // 用户的Secret Access Key
String ENDPOINT = "domain-name";                             // 用户自己指定的域名

// 创建BceClientConfiguration实例
BceClientConfiguration config = new BceClientConfiguration();

// 配置代理为本地8080端口
config.setProxyHost("127.0.0.1");
config.setProxyPort(8080);

// 配置认证秘钥和服务器信息
config.setCredentials(new DefaultBceCredentials(ACCESS_KEY_ID,SECRET_ACCESS_KEY));
config.setEndpoint(ENDPOINT);

// 创建BMR客户端
BmrClient client = new BmrClient(config);
```

使用上面的代码段，客户端的所有操作都会通过127.0.0.1地址的8080端口做代理执行。

对于有用户验证的代理，可以通过下面的代码段配置用户名和密码：

```
 // 创建BceClientConfiguration实例
BceClientConfiguration config = new BceClientConfiguration();

// 配置代理为本地8080端口
config.setProxyHost("127.0.0.1");
config.setProxyPort(8080);

//设置用户名和密码
config.setProxyUsername("username");
config.setProxyPassword("password");
```

设置网络参数

用户可以用BceClientConfiguration对基本网络参数进行设置：

```
 BceClientConfiguration config = new BceClientConfiguration();

// 设置HTTP最大连接数为10
config.setMaxConnections(10);

// 设置TCP连接超时为5000毫秒
config.setConnectionTimeout(5000);

// 设置Socket传输数据超时的时间为2000毫秒
config.setSocketTimeout(2000);
```

BceClientConfiguration参数说明

通过BceClientConfiguration能指定的所有参数如下表所示：

参数	说明
UserAgent	用户代理，指HTTP的User-Agent头
Protocol	连接协议类型
ProxyDomain	访问NTLM验证的代理服务器的Windows域名
ProxyHost	代理服务器主机地址
ProxyPort	代理服务器端口
ProxyUsername	代理服务器验证的用户名
ProxyPassword	代理服务器验证的密码
ProxyPreemptiveAuthenticationEnabled	是否设置用户代理认证
ProxyWorkstation	NTLM代理服务器的Windows工作站名称
LocalAddress	本地地址
ConnectionTimeoutInMillis	建立连接的超时时间（单位：毫秒）
SocketTimeoutInMillis	通过打开的连接传输数据的超时时间（单位：毫秒）
MaxConnections	允许打开的最大HTTP连接数
RetryPolicy	连接重试策略
SocketBufferSizeInBytes	Socket缓冲区大小

异常

异常提示

BMR异常提示有如下两种方式：

异常方法	说明
BceClientException	客户端异常
BceServerException	服务器异常

获取事件异常

用户可以使用try获取某个事件所产生的异常，例如：

```
try {
    CreateClusterResponse response = bmrClient.createCluster(
            new CreateClusterRequest().withName("java-sdk-test")
                                      .withImageType("hadoop")
                                      .withImageVersion("0.1.0")
                                      .withAutoTerminate(false)
                                      .withLogUri("bos://tester01/sdk/")
                                      .withInstanceGroups(instanceGroups)
                                      .withApplications(applications)
    );
} catch (BceServiceException e) {
    System.out.println(e.getMessage());
} catch (BceClientException e) {
    System.out.println(e.getMessage());
}
```

版本更新记录

v 0.9.4

首次发布：

- 支持创建、罗列、查看、终止 Cluster
- 支持添加、罗列、查看 Step

文档更新记录

2019-05-23

- SDK支持含有云磁盘的新套餐

2018-12-13

- 添加集群高可用和安全模式支持
- 修正提交job设置附加文件失败bug

2018-01-04

- 新增Java SDK附加文件作业示例

Python-SDK

简介

本文档主要介绍BMR Python SDK的安装和使用。在使用本文档之前，您需要先了解BMR的一些基本知识，并已经开通了BMR服务。若您还不了解BMR，可以参考产品描述和操作指南。

快速入门

1. 初始化一个BmrClient。 BmrClient是与BMR服务交互的客户端，所有BMR操作都是通过BmrClient完成的。用户可以参考[新建BmrClient](BMR/Python SDK/BmrClient.md#新建BmrClient)，完成初始化客户端的操作。
2. 新建一个BMR Cluster(集群)。 用户创建一个BMR集群，用于提交并运行指定的作业。在提交作业之前，必须先创建出一个集群。创建集群的请求成功后，将返回新集群的ID，可用于后续对集群进行相关操作。

   用户在创建集群时，需要为集群指定集群中虚拟机实例采用的镜像类型、镜像版本号和虚拟机实例组配置。另外，可选的配置项包括集群的名字、需要安装的组件信息(如Hive，Pig，HBase)、上传运行日志的BOS路径以及需要运行的作业信息。用户可以参考[新建集群](BMR/Python SDK/Cluster（集群）.md#新建cluster)，进行具体的集群配置操作。
3. 查看集群的状态信息。 用户通过集群ID可以查看集群的状态信息。创建集群需要等待一定时间后集群成为可用状态，此时提交至集群的作业开始被调度执行。
4. 查看集群实例组信息。 用户通过集群ID可以查看集群的实例组信息，包括实例组类型、实例组实例数等。
5. 修改实例组配置。 用户通过集群ID和实例组ID修改CORE或TASK实例组配置。
6. 查看实例信息。 用户通过集群ID和实例组ID查看实例信息。
7. 添加一个或多个Step(作业)。 向指定集群添加一个或多个作业任务。集群创建成功后，用户可以根据集群已安装的应用来添加不同类型的作业，包括Custom Jar、Streaming、Hive、Pig。添加作业请求成功后，将返回包含所有新作业ID的数组，作业ID可用于后续对作业进行相关操作。
8. 查看作业的运行状态信息。 用户通过作业ID可以查看作业的运行状态信息。
9. 终止集群。 在结束作业的运行后，用户可以通过集群ID发送终止集群的请求。终止集群将释放集群的虚拟机实例，结束计费。

安装SDK工具包

环境准备

1. 运行环境

   Python SDK工具包支持在Python 2.7 以上环境运行。
2. 安装pycrypto依赖

   安装SDK之前，需要先执行命令`pip install pycrypto`安装pycrypto依赖。
   如果安装失败，请执行`pip install pycryptodome`

下载和安装

方式一：通过pip安装

您可以通过pip安装的方式将百度智能云Python SDK安装到您的环境中。 联网状态下，在命令行中执行如下命令：

```
pip install bce-python-sdk
```

即可将Python SDK安装到本地。 **方式二：将源码包下载到本地后进行安装**

1. 在开发者资源中心下载Python SDK压缩工具包。
2. 命令行移动到压缩包所在路径，执行如下命令（version替换为包名称中的版本号）：

```
pip install bce-python-sdk-version.zip
```

即可将Python SDK安装到本地。

您也可以解压压缩包后执行如下命令（version替换为包名称中的版本号）

```
cd bce-python-sdk-version

python setup.py install
```

SDK目录结构

```
baidubce
├─── auth              //公共权限目录
├─── services          //服务公共目录
│    └─── bmr          //BMR目录
└─── http              //Http请求模块
```

Instance（实例）

查询实例列表

如下代码可查询指定集群和实例组的实例：

```
try:
    response = bmr_client.list_instances(cluster_id, instance_group_id)
    for instance in response.instances:
        LOG.debug('list instances %s: %s' % (instance.id, instance))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        LOG.error('list_instances failed. Response %s, code: %s, msg: %s'
                  % (e.last_error.status_code, e.last_error.code, e.last_error.args))
    else:
        LOG.error('list_instances failed. Unknown exception: %s' % e)
```

InstanceGroup（实例组）

查询实例组列表

如下代码可根据集群ID获取其实例组信息：

```
try:
    response = bmr_client.list_instance_groups(cluster_id)
    for instance_group in response.instance_groups:
        LOG.debug('list instance groups %s: %s' % (instance_group.id, instance_group))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        LOG.error('list_instance_groups failed. Response %s, code: %s, msg: %s'
                  % (e.last_error.status_code, e.last_error.code, e.last_error.args))
    else:
        LOG.error('list_instance_groups failed. Unknown exception: %s' % e)
```

修改实例组配置

BMR支持对CORE、TASK节点进行配置变更，示例代码如下：

```
  try:
    instance_group_config_core = instancegroup_config(core_instance_id, core_instance_count + 1)
    instance_group_config_task = instancegroup_config(task_instance_id, task_instance_count + 1)
    instance_group_config_client = instancegroup_config(client_instance_id, client_instance_count + 1)
    bmr_client.scale_cluster(cluster_id, instance_group_config=[instance_group_config_core, instance_group_config_task, instance_group_config_client])
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        LOG.error('scale_cluster failed. Response %s, code: %s, msg: %s'
                  % (e.last_error.status_code, e.last_error.code, e.last_error.args))
    else:
        LOG.error('scale_cluster failed. Unknown exception: %s' % e)
```

Cluster（集群）

新建cluster

如下代码可以新建一个集群，集群包含1个master节点和2个core节点，且安装了Hive、Pig、HBase应用。请注意：参考下面样例代码时，需要修改log_uri参数指定BOS路径为您的账户可用的BOS路径。

```
  instance_groups = [
    BmrClient.instance_group(
        'Master',
        'bmr.g1.xlarge ',
        1,
        'ig-master'
    ),
    BmrClient.instance_group(
        'Core',
        'bmr.g1.xlarge ',
        3,
        'ig-core'
    )
] # 构造请求体
request_body = {
    'applications': [
        {
            "name": "hive",
            "version": "3.1.0"
        },
        {
            "name": "pig",
            "version": "0.17.0"
        },
        {
            "name": "hbase",
            "version": "2.2.7"
        }
    ],
    'auto_terminate': False,
    'admin_pass': '1234rewq!',
    'name': 'sdk-cluster01',
    'service_ha_enabled': False,
    'safe_mode_enabled': False
}

try:
    response = bmr_client.create_cluster(
        'hadoop',
        '2.2.1',
        instance_groups,
        **request_body)
    cluster_id = response.cluster_id
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        LOG.error('create_cluster failed. Response %s, code: %s, msg: %s'
                  % (e.last_error.status_code, e.last_error.code, e.last_error.args))
    else:
        LOG.error('create_cluster failed. Unknown exception: %s' % e)
```

售卖的套餐类型请参考https://cloud.baidu.com/doc/BMR/s/6jwvxw85z

列出全部cluster

如下代码可以罗列出属于请求调用者的所有集群，用户可以通过配置查询参数max_keys来限制每次请求返回的集群数目：

```
try:
    response = bmr_client.list_clusters(max_keys=5)
    LOG.debug('list total %s clusters' % len(response.clusters))
    # 输出各个集群的信息
    for cluster in response.clusters:
        LOG.debug('cluster: %s' % cluster)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        LOG.error('list_clusters failed. Response %s, code: %s, msg: %s'
                  % (e.last_error.status_code, e.last_error.code, e.last_error.args))
    else:
        LOG.error('list_clusters failed. Unknown exception: %s' % e)
```

另外，也可以使用分页查询的方式来获取所有集群，通过配置max_keys来限制每页查询返回的集群数目：

```
try:
    page = 1
    next_marker = None
    max_keys = 5
    is_truncated = True
    while is_truncated:
        response = bmr_client.list_clusters(marker=next_marker, max_keys=max_keys)
        LOG.debug('list %s clusters on Page %s' % (len(response.clusters), page))
        page += 1
        is_truncated = response.is_truncated
        next_marker = response.next_marker
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        LOG.error('list_clusters failed. Response %s, code: %s, msg: %s'
                  % (e.last_error.status_code, e.last_error.code, e.last_error.args))
    else:
        LOG.error('list_clusters failed. Unknown exception: %s' % e)
```

查询指定的cluster

获取了集群ID后，可用如下代码查询指定集群的信息：

```
try:
    response = bmr_client.get_cluster(cluster_id)
    # 输出集群的状态信息
    status = response.status.state
    LOG.debug('check cluster %s: status %s' % (cluster_id, status))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        LOG.error('get_cluster failed. Response %s, code: %s, msg: %s'
                  % (e.last_error.status_code, e.last_error.code, e.last_error.args))
    else:
        LOG.error('describe_cluster failed. Unknown exception: %s' % e)
```

终止指定的cluster

如下代码可以终止指定的集群：

```
try:
    response = bmr_client.terminate_cluster(cluster_id)
    LOG.debug('terminate cluster %s: status %s' % (cluster_id, response.status))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        LOG.error('terminate_cluster failed. Response %s, code: %s, msg: %s'
                  % (e.last_error.status_code, e.last_error.code, e.last_error.args))
    else:
        LOG.error('terminate_cluster failed. Unknown exception: %s' % e)
```

Step（作业）

添加steps

作业是和集群相关联的资源，对作业的操作需要指定集群ID。

BMR支持多种类型的作业，不同类型的作业有不同的配置项。如下代码可向指定的hadoop类型的集群添加Custom Jar、Streaming、Hive、Pig作业。请注意：参考下面样例代码时，需要修改作业参数指定的BOS路径为您的账户可用的BOS路径。

```
steps = [
    BmrClient.step(
            'Java',
            'Continue',
            BmrClient.java_step_properties(
                'bos://benchmark/hadoop/hadoop-mapreduce-examples.jar',
                'org.apache.hadoop.examples.WordCount',
                'bos://helloworld/input/install.log bos://tester01/sdk/output_java/out1'
                ),
            'sdk-job-01'
            ),
    BmrClient.step(
            'Streaming',
            'Continue',
            BmrClient.streaming_step_properties(
                'bos://helloworld/input/install.log',
                'bos://tester01/sdk/output_streaming/out1',
                'cat'),
            'sdk-job-02'
            ),
    BmrClient.step(
            'Hive',
            'Continue',
            BmrClient.hive_step_properties(
                'bos://chy3/hive/hql/hive_src.hql',
                '--hivevar LOCAT=bos://chy3/hive/tables/src',
                'bos://chy3/hive/data/hive_src.data',
                'bos://tester01/sdk/output_hive/out1'
                ),
            'sdk-job-03'
            ),
    BmrClient.step(
            'Pig',
            'Continue',
            BmrClient.pig_step_properties(
                'bos://chy3/pig/script/pig_grep.pig',
                input='bos://chy3/pig/data/pig_grep.data',
                output='bos://tester01/sdk/output_pig/out1'
                ),
            'sdk-job-04'
            ),
    BmrClient.step(
            'Spark',
            'Continue',
            BmrClient.spark_step_properties(
                'bos://bmr-public-bj/sample/spark-1.0-SNAPSHOT.jar',
                '--class com.baidu.cloud.bmr.spark.AccessLogAnalyzer',
                'bos://bmr-public-bj/data/log/accesslog-1k.log bos://tester01/sdk/output/out'
                ),
            'sdk-job-05'
            ),
    BmrClient.step(
            'Streaming',
            'Continue',
            BmrClient.streaming_step_properties(
                'bos://helloworld/input/install.log',
                'bos://tester01/sdk/output_streaming/out1',
                'cat',
                'cat',
                '-libjars testB.jar'
                ),
            'sdk-job-06',
            [BmrClient.additional_file("bos://path/to/testA.jar", "testB.jar")]
            )
]

try:
    response = bmr_client.add_steps(cluster_id, steps)
    LOG.debug('add steps response: %s' % response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        LOG.error('add_steps failed. Response %s, code: %s, msg: %s'
                % (e.last_error.status_code, e.last_error.code, e.last_error.args))
    else:
        LOG.error('add_steps failed. Unknown exception: %s' % e)
```

列出全部steps

如下代码可以罗列出指定集群上的全部作业，用户可以通过指定pageNo和pageSize来限制一次请求返回的最大作业数目：

```
try:
    response = bmr_client.list_steps(cluster_id, pageNo=1, pageSize=50)
    for step in response.steps:
        LOG.debug('list step %s: %s' % (step.id, step))
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        LOG.error('list_steps failed. Response %s, code: %s, msg: %s'
                  % (e.last_error.status_code, e.last_error.code, e.last_error.args))
    else:
        LOG.error('list_steps failed. Unknown exception: %s' % e)
```

查询指定的step

如下代码可以查看指定作业的信息：

```
try:
    response = bmr_client.get_step(cluster_id, step_id)
    LOG.debug('describe step response: %s' % response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        LOG.error('get_step failed. Response %s, code: %s, msg: %s'
                  % (e.last_error.status_code, e.last_error.code, e.last_error.args))
    else:
        LOG.error('get_step failed. Unknown exception: %s' % e)
```

BmrClient

配置BmrClient

BmrClient是BMR服务的Python客户端，为调用者与BMR服务进行交互提供一系列的方法。

在新建BmrClient之前，需要先创建配置文件对BmrClient进行配置，以下将此配置文件命名为`bmr_client_conf.py`，具体配置信息如下所示：

```
#!/usr/bin/env python
#coding=utf-8

#导入Python标准日志模块
import logging

#从Python SDK导入BMR配置管理模块以及安全认证模块
from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials

#设置BmrClient的Host，Access Key ID和Secret Access Key
host = "bmr.bj.baidubce.com"
access_key_id = "your-access-key-id"
secret_access_key = "your-secret-access-key"

#设置日志文件的句柄和日志级别
logger = logging.getLogger('baidubce.services.bmr.bmrclient')
fh = logging.FileHandler("sample.log")
fh.setLevel(logging.DEBUG)

#设置日志文件输出的顺序、结构和内容
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
fh.setFormatter(formatter)
logger.setLevel(logging.DEBUG)
logger.addHandler(fh)

#创建BceClientConfiguration
config = BceClientConfiguration(credentials=BceCredentials(access_key_id, secret_access_key),
                                endpoint=host)
```

注意：

1. 在上面的代码中，变量access_key_id与secret_access_key是系统分配给用户的，用于标识用户，为访问Media做签名验证。其中access_key_id对应控制台中的“Access Key ID”，secret_access_key对应控制台中的“Access Key Secret”，获取方式请参考《管理ACCESSKEY》。
2. BceClientConfiguration构造函数的endpoint参数只能用指定的包含Region的域名来进行定义，目前开放了BMR SDK服务的区域是“华北-北京”和“华南-广州”，

> 北京Region的endpoint的域名是`http://bmr.bj.baidubce.com`，广州Region的endpoint的域名是`http://bmr.gz.baidubce.com`。

新建BmrClient

在完成上述配置后，用户可以参考如下代码新建一个BmrClient:

```
import logging
import bmr_client_conf
from baidubce.services.bmr.bmr_client import BmrClient

logging.basicConfig(level=logging.DEBUG)
LOG = logging.getLogger(__name__)
CONF = bmr_client_conf

bmr_client = BmrClient(CONF.config)
```

参数说明

Python SDK在`baidubce/bce_client_configuration.py`中默认设置了一些基本参数，若用户想要对参数的值进行修改，可以参考此文件创建自身的参数配置函数，并在构造BmrClient的时候传入，传入代码参考如下：

```
from baidubce.retry_policy import BackOffRetryPolicy
from baidubce.bce_client_configuration import BceClientConfiguration
from baidubce.auth.bce_credentials import BceCredentials
from baidubce.protocol import HTTP
from baidubce.region import BEIJING

my_policy = BackOffRetryPolicy(max_error_retry=3,
                               max_delay_in_millis=20 * 1000,
                               base_interval_in_millis=300)

my_config = BceClientConfiguration(
    credentials=BceCredentials('your-access-key-id', 'your-secret-access-key'),
    endpoint='bmr_service_host',
    protocol=baidubce.protocol.HTTP,
    region=baidubce.region.BEIJING,
    connection_timeout_in_mills=50 * 1000,
    send_buf_size=1024 * 1024,
    recv_buf_size=10 * 1024 * 1024,
    retry_policy=my_policy)

#### create BmrClient with my config
my_client = BmrClient(my_config)
```

参数说明如下：

参数	说明	默认值
PROTOCOL	协议	baidubce.protocol.HTTP
REGION	区域	baidubce.region.BEIJING（目前只支持北京地区）
CONNECTION_TIMEOUT_IN_MILLIS	请求超时时间（单位：毫秒）	50 * 1000
SEND_BUF_SIZE	发送缓冲区大小	1024 * 1024
RECV_BUF_SIZE	接收缓冲区大小	10 * 1024 * 1024
retry_policy	重试逻辑	最大重试次数3次，超时时间为20 * 1000毫秒，重试间隔300毫秒

异常处理

BMR异常提示有如下几种方法：

异常方法	说明
BceHttpClientError	重试时抛出的异常
last_error	最后一次重试时抛出的异常
BceClientError	BMR客户端产生的异常
BceInvalidArgumentError	传递参数产生的异常
BceServerError	BMR服务器产生的异常

用户可以使用try获取某个事件所产生的异常：

```
 from baidubce.exception import BceHttpClientError
from baidubce.exception import BceServerError

try:
    response = bmr_client.get_step(new_cluster_id, step_id)
    LOG.debug('describe steps response: %s' % response)
except BceHttpClientError as e:
    if isinstance(e.last_error, BceServerError):
        LOG.error('get_step failed. Response %s, code: %s, msg: %s'
                  % (e.last_error.status_code, e.last_error.code, e.last_error.message))
    else:
        LOG.error('get_step failed. Unknown exception: %s' % e)
```

版本变更记录

- Python SDK开发包[2016-01-05]版本号0.8.8

首次发布：

- 支持创建、罗列、查看、终止 Cluster
- 支持添加、罗列、查看 Step

文档更新记录

2018-01-04

- 新增Python SDK附加文件作业示例

服务等级协议SLA

BMR服务等级协议SLA

协议生效时间：2019年05月01日

本服务等级协议（Service Level Agreement，以下简称 "SLA"）规定了百度智能云向客户提供的MapReduce BMR的服务可用性等级指标及赔偿方案。

1. 定义

服务周期： 一个服务周期为一个自然月。 失败请求：用户直接调用Step API，返回的HTTP状态码为5XX的请求。

每5分钟错误率：

$$每5分钟错误率 = \left(\frac{每5分钟失败请求数}{每5分钟有效请求数}\right)$$

（注：如果用户在此时间段内未产生请求，则错误率视为0）

月度服务费： 根据地域分别计算，用户在一个服务周期（即自然月）中就MapReduce BMR所支付的服务费用总额。

2. 服务可用性

2.1 服务可用性计算方式

$$服务可用性 = \left(1 - \frac{服务周期内\sum每5分钟错误率}{服务周期内5分钟个数}\right) \times 100\%$$

（注：服务周期内5分钟总个数=12*24*该服务周期的天数）

2.2 服务可用性承诺

MapReduce BMR承诺在一个服务周期内，每一服务区域内服务可用性不低于99.90%。

如MapReduce BMR未达到上述服务可用性承诺，客户可以根据本协议第三条约定获得赔偿。赔偿范围不包括以下原因所导致的不可用：

(1) 百度智能云预先通知用户后进行系统维护所引起的，包括割接、维修、升级和模拟故障演练；

(2) 任何百度智能云所属设备以外的网络、设备故障或配置调整引起的；

(3) 用户的应用程序或数据信息受到黑客攻击而引起的；

(4) 用户维护不当或保密不当致使数据、口令、密码等丢失或泄漏所引起的；

(5) 用户的疏忽或由用户授权的操作所引起的；

(6) 不可抗力以及意外事件引起的；

(7) MapReduce BMR只赔付MapReduce BMR自身，不赔付MapReduce BMR所关联的其他云服务；

(8) 其他非百度智能云原因所造成的MapReduce BMR无法正常使用；

(9) 用户未遵循百度智能云产品使用文档或使用建议引起的。

3. 服务赔偿条款

3.1 赔偿标准

根据客户某一百度智能云账号下某一地域的MapReduce BMR的月度服务可用性，按照下表中的标准计算赔偿金额，赔偿方式仅限于用于购买MapReduce BMR的代金券，且赔偿总额不超过未达到服务可用性承诺的当月客户就该账号下该地域支付的月度服务费的25%。

服务可用性	赔偿代金券金额
低于99.90%但等于或高于99.00%	月度服务费用的10%
低于99.00%	月度服务费用的25%

3.2 赔偿申请时限

客户可以在每月第五（5）个工作日后对上个月没有达到可用性的服务提出赔偿申请。赔偿申请必须限于在消息服务没有达到可用性的相关月份结束后两（2）个月内提出。超出申请时限的赔偿申请将不被受理。

4. 其他说明

(1) 在法律法规允许的范围内，百度智能云负责对本协议进行解释说明。

(2) 本协议一经公布立即生效，百度智能云有权对本SLA条款作出修改。如本SLA条款有任何修改，百度智能云将提前30天以网站公示或发送邮件的方式通知您。如您不同意百度智能云对SLA所做的修改，您有权停止使用消息服务，如您继续使用消息服务，则视为您接受修改后的SLA。

(3) 本协议项下百度智能云对于用户所有的通知均可通过网页公告、站内信、电子邮件、手机短信或其他形式等方式进行；该等通知于发送之日视为已送达收件人。因用户未及时获知百度智能云的服务变更或终止条款遭受损失的，百度智能云不承担任何责任。

(4) 协议的订立、执行和解释及争议的解决均应适用中国法律并受中国法院管辖。如双方就本协议内容或其执行发生任何争议，双方应尽量友好协商解决；协商不成时，任何一方均可向北京市海淀区人民法院提起诉讼。

(5) 本协议构成双方对本协议之约定事项及其他有关事宜的完整协议，除本协议规定的之外，未赋予本协议各方其他权利。

(6) 如本协议中的任何协议无论因何种原因完全或部分无效或不具有执行力，本协议的其余协议仍有效并且有约束力。

(7) 关于用户约束条款，详见《用户服务协议》中的"用户的权利与义务"相关条款内容。

(8) 关于服务商免责条款，详见《用户服务协议》中的"免责声明"相关条款内容。

视频专区

产品介绍

BMR产品介绍

- 介绍BMR的优势及产品特性。

操作指南

BMR作业操作演示

- 演示BMR如何在控制台编辑作业。

BMR集群操作演示

- 演示BMR如何在控制台编辑集群。

常见问题

常见问题总览

故障类问题

- 作业运行失败怎么办？
- 作业为什么会运行失败？
- 作业为什么会提交失败？
- 集群为什么会自动终止？

配置类问题

- 如何配置集群，使得作业运行完毕后，集群自动终止？
- BMR无法手动停止 job，必须等待它运行完吗？
- 集群是否支持外网登录？
- Core节点与Task节点的使用区别在哪里，如何选择？

计费类问题

- BMR如何进行收费？
- 费用不足，用户该如何充值？
- 充值时，一次性充多少钱可以满足用户处理数据的需求？

性能类问题

- BMR是否可估算处理作业的运行时间？
- BMR能运行一个超级大的作业吗？
- 每个账户可以同时运行多少个集群？
- 集群在不停止服务的情况下是否支持规模扩展？

安全性问题

- 用户的数据安全性如何？
- 在集群运行过程中，用户的数据保密性如何？

故障类问题

作业运行失败怎么办？

可以在集群的作业列表中，找到运行失败作业的日志，分别为syslog、stderr、stdout三个日志。其中Syslog日志记录了作业运行的信息，stderr记录了作业运行失

败的原因，stdout记录了作业运行的过程中输出的信息。通过查看stderr日志，找到作业运行失败的原因并进行修复，再次运行作业。

作业为什么会运行失败？

1：作业在BOS中的输入目录不存在或者输出目录已存在，导致reduce task无法读取或写入数据，从而造成作业失败。

解决方案：请确保BOS中输入目录存在且输出目录不存在。

2：作业本身存在错误。

解决方案：若您提交的是Custom JAR、Spark、Pig类型作业，有可能是您自定义的参数不符合规范。可查阅task日志，找出对应错误，对作业进行修复。

3：输入参数有问题。

解决方案：可能是输入的参数关键字拼写有问题。通过查阅task日志，找出对应错误，对作业进行修复。

作业为什么会提交失败？

1：参数设置不符合标准的格式规范。

解决方案：根据页面返回的错误信息，参阅新手指南的新建各类型作业的参数配置，进行相应的修改，。

2：作业数目超过256个。

解决方案：按照提示，新建集群，在新建的集群中添加作业。

集群为什么会自动终止？

说明集群在处理作业时，遇到了一些意外，比如输入的bos地址不存在，或者用户对这个地址没有相应的处理权限等。且用户在对作业进行设置时，在【失败后的操作】选择了销毁集群，从而当作业运行失败后，集群会自动终止。这种情况大多数是由于用户对作业的设置不符合规定从而导致的，可检查errlog日志，重新对作业进行设置。

配置类问题

如何配置集群，使得作业运行完毕后，集群自动终止？

在创建集群时进行设置，在【自动终止】选项中，选择开启，则在作业最后一步完成时，集群会自动终止。

BMR无法手动停止 job，必须等待它运行完吗？

不是的，作业流中的作业支持在控制台作业列表中进行人工停止作业。

集群是否支持外网登录？

支持。您可在集群运行期间，使用安全外壳协议（SSH），通过Master节点的公网IP、登录用户名和密码、以及SSH端口号，连接到Master主节点并实现与集群交互。

- Linux环境下：通过 ssh [username]@[eip] -p [端口号]命令连接Master节点。
- Windows环境下：通过SSH客户端（Putty、SecureCRT及Xshell等）使用SSH协议登录。

Core节点与Task节点的使用区别在哪里，如何选择？

1. Core节点与Task节点的区别：和Core节点相比，Task节点没有部署hdfs，在增加和减小规模的时候没有丢失hdfs数据副本的风险，因此更加适合用来动态调整集群的计算能力（Core节点存储数据，可以运行DataNode和NodeManager，Task节点只运行NodeManager）。
2. 如何选择：举例说明，例如处理10G数据，按照128MB切分，一共80个map task，一个计算节点（比如内存最优型）预期需要8个task的话，想尽快完成就需要10个计算节点，如果core配置5个节点，那么task也配5个就可以了。关于core节点数目，如果使用集群内的hdfs，需要根据hdfs中的数据大小规划core节点数量（一般常驻集群是这样的）；如果不使用集群内的hdfs，而是使用bos作为数据源，可以使用最少的core节点数量（一般按需启动的集群是这样的）。以上只是一个例子，仅供参考。

VPC未增加域名解析导致集群创建失败，如何处理？

1. 进入 VPC 产品页面，点击实例名称，查看 VPC 详情；
2. 查看“域名”是否为 "-"，若域名已有具体值，如 "xxx.baidu.internal" ，则已经增加域名解析，返回 BMR 产品继续创建集群即可；
3. 如果“域名”为 "-"，请您使用主账户，点击“变更”按钮，按照 region 变更域名，参照以下表格，

region	域名
武汉	fwh.baidu.internal
北京	bj.baidu.internal
保定	bdbl.baidu.internal
广州	gz.baidu.internal
苏州	su.baidu.internal
香港	hkg.baidu.internal
上海	fsh.baidu.internal

4. 配置完成后，返回产品继续创建集群即可；

计费类问题

BMR如何进行收费？

BMR的计费规则请参见产品定价。

费用不足，用户该如何充值？

用户可以到【财务中心】下，购买需要的服务。

充值时，一次性充多少钱可以满足用户处理数据的需求？

在增加各种类型的作业，到确认订单页面后，会显示预计所需要的花费。可以依此为参照，到【财务中心】菜单下，购买BMR服务。

性能类问题

BMR是否可估算处理作业的运行时间？

处理作业的运行时间依赖若干种因素，包括作业的规模和复杂度、设置实例的数目以及套餐类型等多种情况，因此BMR没有提供估算作业运行时间的服务。

BMR能运行一个超级大的作业吗？

BMR对处理作业的大小没有任何限制。若作业大于5G，BOS 提供了MultiUpload通道上传文件。但需要注意，每个集群中的作业数量不能超过256个。

每个账户可以同时运行多少个集群？

目前BMR单用户付费集群最多同时运行5个。

集群在不停止服务的情况下是否支持规模扩展？

支持，集群是可扩展的。点击“集群列表>节点配置变更”，进入“变更配置”页面，设置增加CORE节点数量，点击“确认配置调整”即完成扩展。

安全性问题

用户的数据安全性如何？

BOS提供多种访问控制机制，如签名验证机制、访问控制列表等，确保用户存储数据的安全性以防止未被授权进行访问。其中签名验证机制是通过对URL进行签名来识别访问者的身份，从而实现用户身份验证。其中访问控制列表是对Bucket的管理和校验。

在集群运行过程中，用户的数据保密性如何？

首先BMR有账户和网络系统作为保证，除非登录到master节点或者通过console查看，否则读不到用户数据。除此之外，BMR底层会形成一个封闭的虚拟网络，不同用户之间的机器是不可见的，严格保证了数据不会外泄。