尚硅谷大数据项目之电商实时数仓（电商

数据仓库系统）

（作者：尚硅谷研究院）

版本：V5.0

1. 部署前准备工作

在/etc/profile.d/my\_env.sh中配置

export HADOOP\_CLASSPATH=`hadoop classpath`

修改yarn-site.xml配置 container最小和最大值以及nm分配的内存

<property>

<name>yarn.scheduler.minimum-allocation-mb</name>

<value>1024</value>

</property>

<property>

<name>yarn.scheduler.maximum-allocation-mb</name>

<value>2048</value>

</property>

<property>

<name>yarn.nodemanager.resource.memory-mb</name>

<value>4096</value>

</property>

修改hadoop下的capacity-scheduler.xml配置

这里意思是集群中可用于运行应用程序主机的最大资源百分比 - 控制并发活动应用程序的数量。每个队列的限制与其队列容量和用户限制成正比。官方文档默认是每个队列最大使用10%的资源，把这里的0.1按需更改最大使用的资源数就行了

<property>

<name>yarn.scheduler.capacity.maximum-am-resource-percent</name>

<value>1</value>

</property>

修改flink客户端配置

jobmanager.memory.process.size: 1024m

taskmanager.memory.process.size: 1024m

taskmanager.numberOfTaskSlots: 2

1. 手动部署
2. 将gmall-realtime父模块install到本地仓库
3. 将realtime-common模块install到本地仓库
4. 打包dwd下单和dws省份下单模块,上传到flink目录下
5. 启动zk、kafka、maxwell、hdfs、hbase、doris、redis、yarn环境
6. 启动历史服务mapred --daemon start historyserver
7. 提交DwdTradeOrderDetail、DwsTradeProvinceOrderWindow应用

-独立分窗口启动

bin/flink run-application \

-t yarn-application \

-d \

-Djobmanager.memory.process.size=1024mb \

-Dtaskmanager.memory.process.size=1024mb \

-Dtaskmanager.numberOfTaskSlots=2 \

-c com.atguigu.gmall.realtime.app.dwd.db.DwdTradeOrderDetail \

/opt/module/flink-1.17.0/gmall2023-realtime-1.0-SNAPSHOT.jar

bin/flink run-application \

-t yarn-application \

-d \

-Djobmanager.memory.process.size=1024mb \

-Dtaskmanager.memory.process.size=1024mb \

-Dtaskmanager.numberOfTaskSlots=2 \

-c com.atguigu.gmall.realtime.app.dws.DwsTradeProvinceOrderWindow \

/opt/module/flink-1.17.0/gmall2023-realtime-1.0-SNAPSHOT.jar

10.打包publisher并上传运行

11.修改内网穿透natapp映射的地址

改为hadoop102:8070

12.sugar修改空间映射

13.运行模拟生成日志的jar包，查看效果

**Doris内存大小限制修改**

show VARIABLES like '%mem\_limit%'

SET GLOBAL exec\_mem\_limit = 4096934592;

**Yarn查看错误日志**

yarn logs –applicationId xxxx

1. 项目部署及上线
   1. StreamPark部署

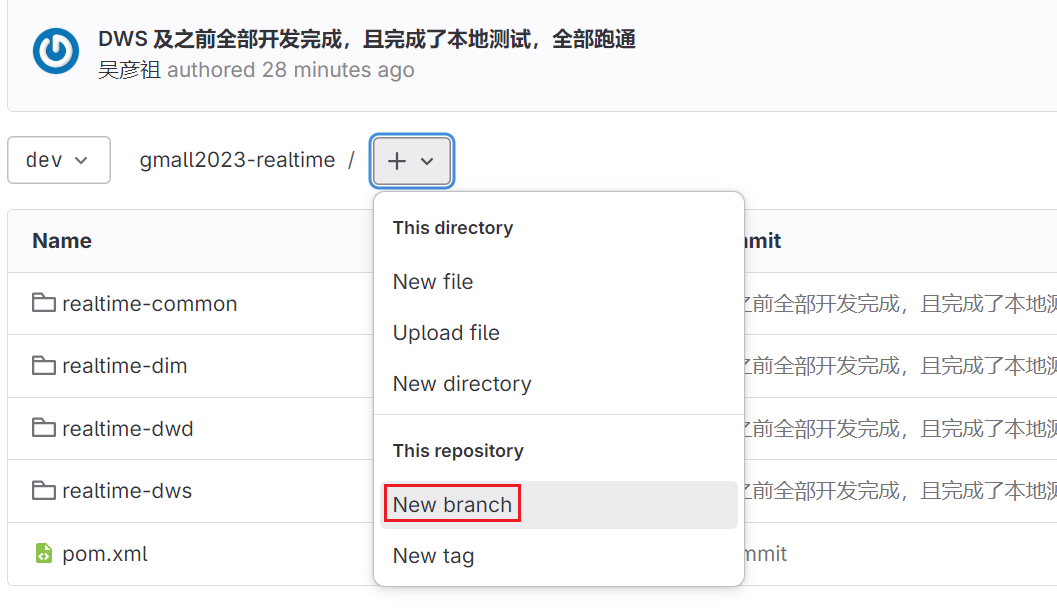


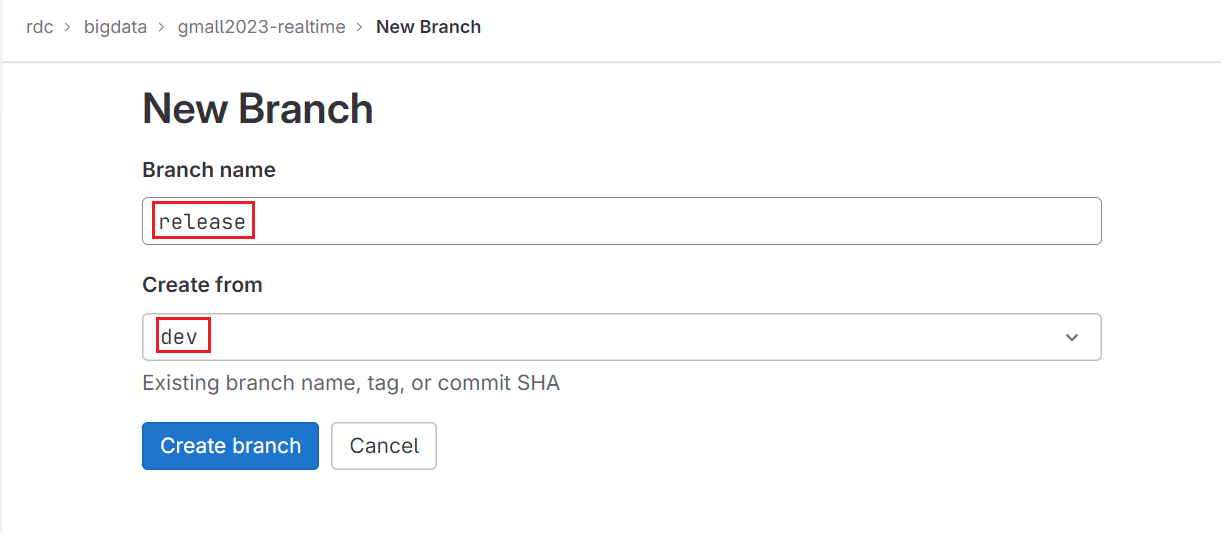
* 1. 业务代码部署测试

本项目按照GitFlow工作流开发，为简化流程，开发全部在dev分支完成，没有新建feature分支。此时，所有业务代码已开发完成，在发布上线之前，我们要创建release分支，在该分支进行测试，测试通过则正式发布。

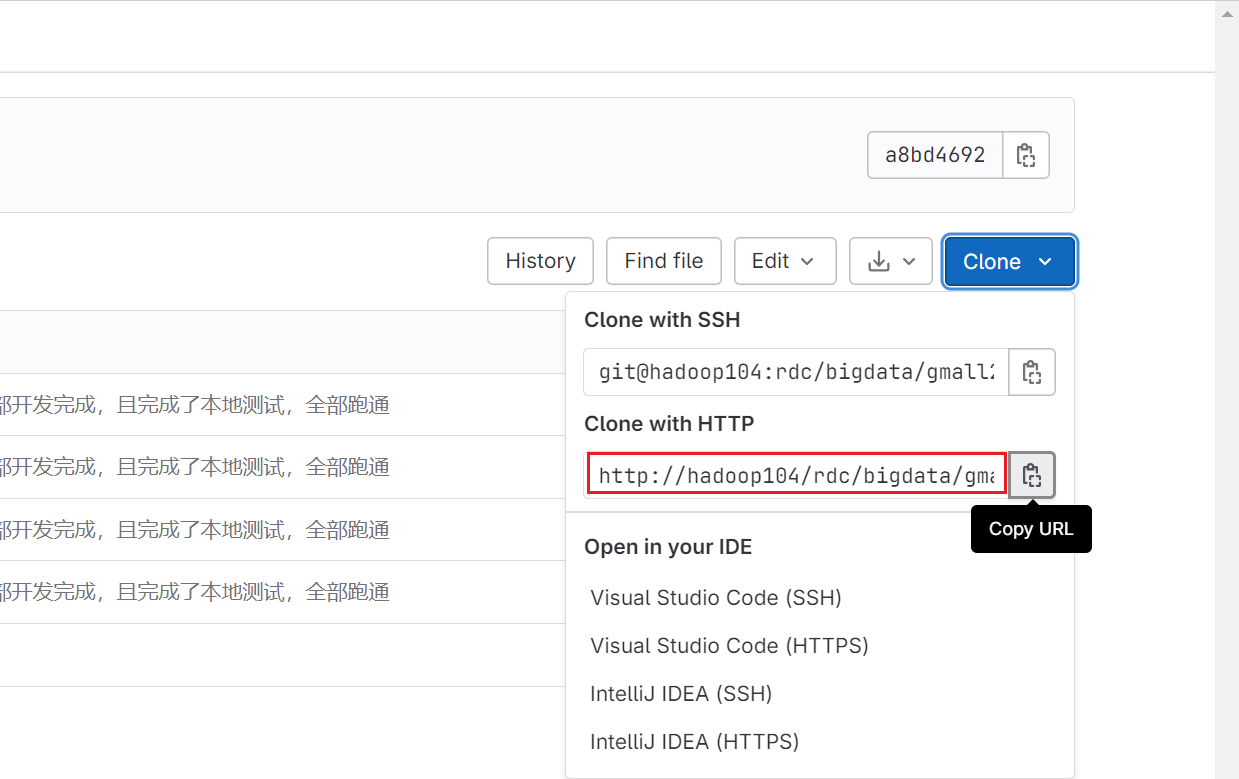
* + 1. 新建release分支

假设预发布分支的创建和后续的测试工作均由开发者atguigu完成，登录该账户，创建release分支，如下。

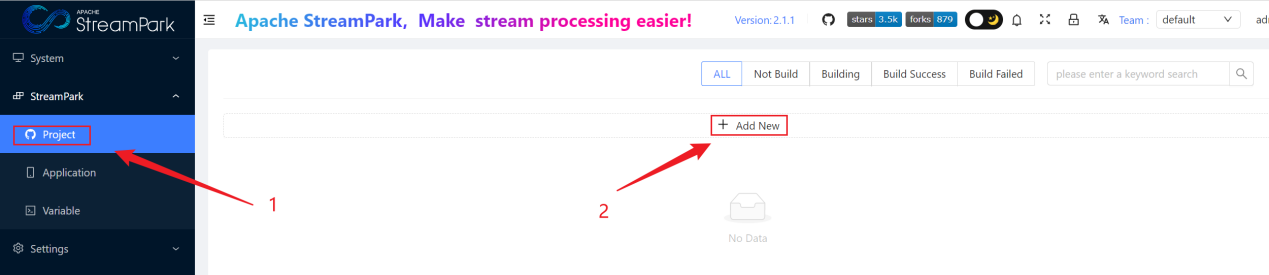




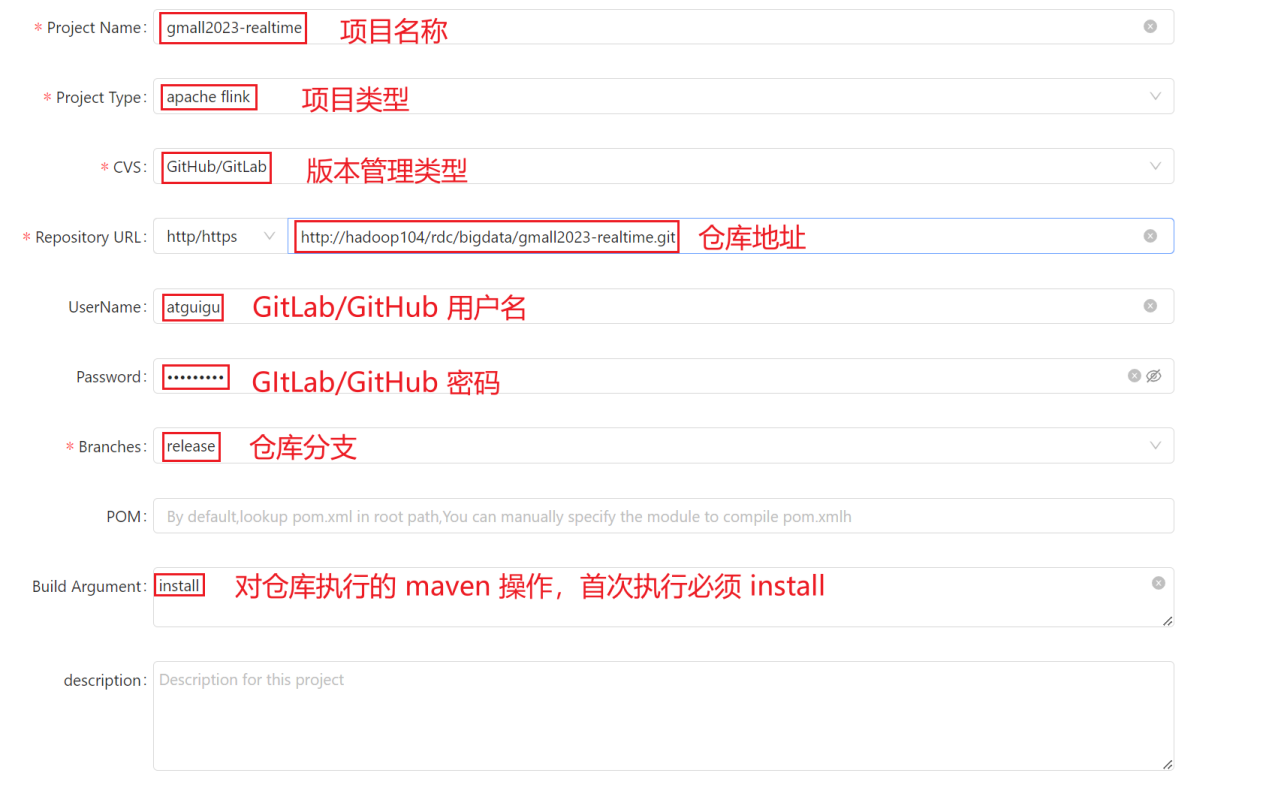
* + 1. 在StreamPark中新建项目
       - 1. 查看GitLab仓库地址



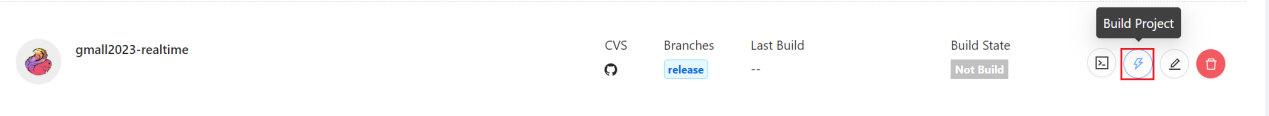
* + - * 1. 新建项目并配置



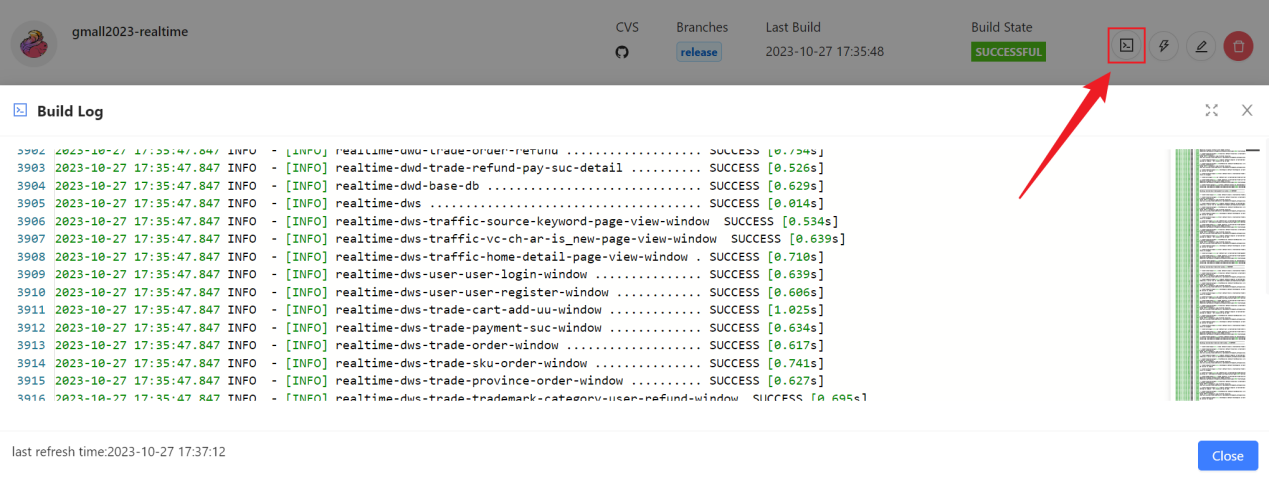
* + - * 1. 添加项目信息



* + 1. 编译项目
       - 1. 首次编译整个工程



* + - * 1. 查看日志



* + - * 1. 查看项目状态



编译成功。

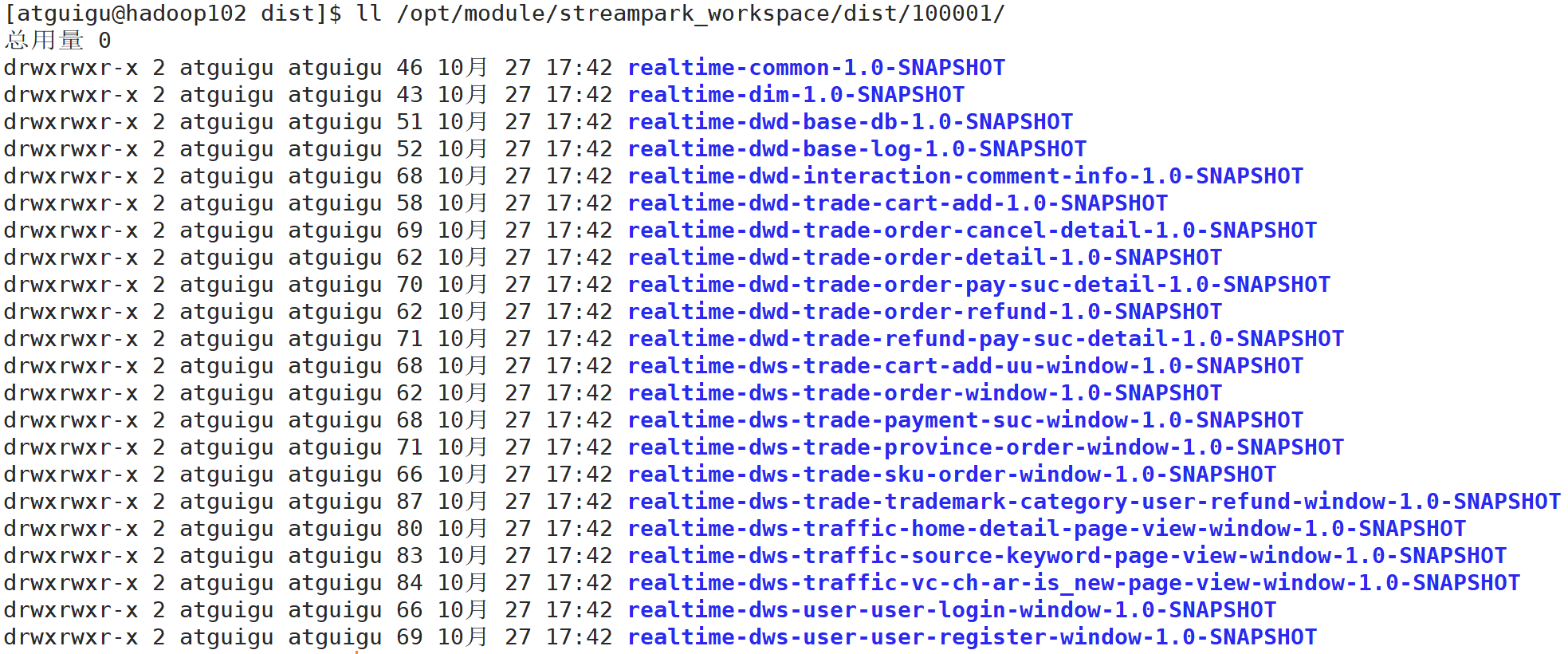
* + 1. 拷贝common包
       - 1. 查看StreamPark目录

本地编译目录

StreamPark会把代码从GitLab拉取到服务器上，打好的包放在ditst目录中。

[atguigu@hadoop102 dist]$ ll /opt/**module**/**streampark\_workspace**/**dist**/100001/

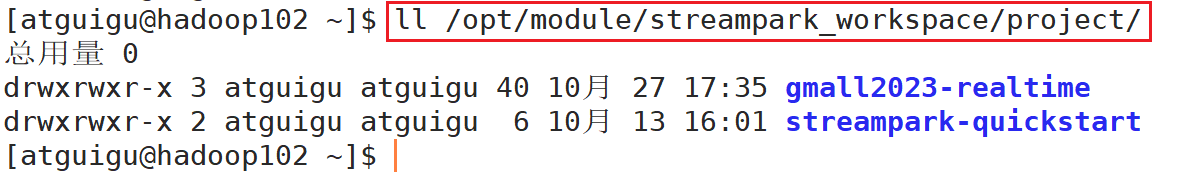
结果如下



本地项目源码目录

[atguigu@hadoop102 ~]$ ll /opt/**module**/**streampark\_workspace**/**project**/

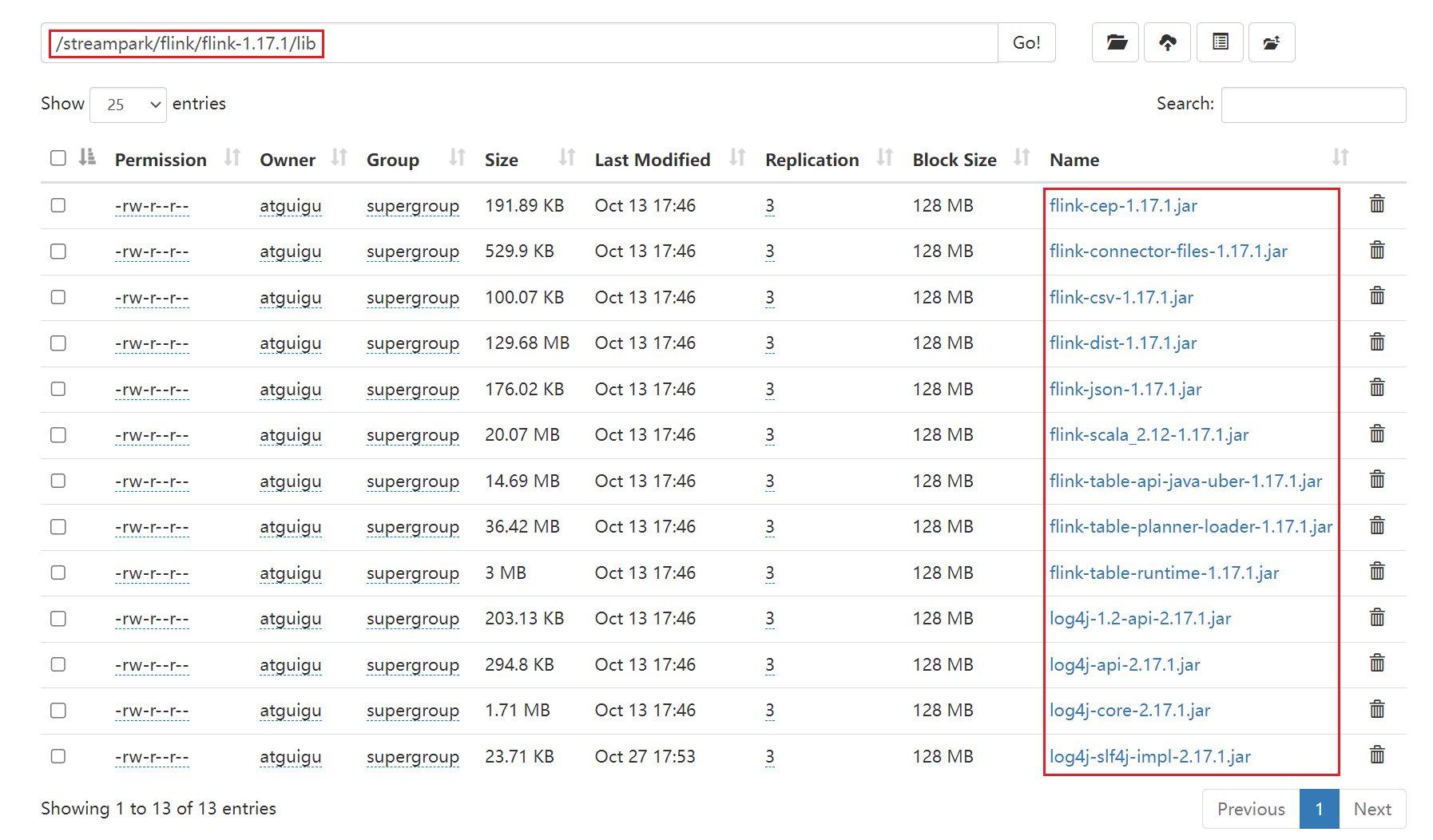
结果如下



将本地的Flink-1.17.1目录上传到HDFS的/streampark/flink/目录

hadoop fs -mkdir -p /streampark/flink/

hadoop fs -put /opt/module/flink-1.17.1 /streampark/flink/



* + - * 1. 将编译好的common包拷贝至Flink的lib目录下

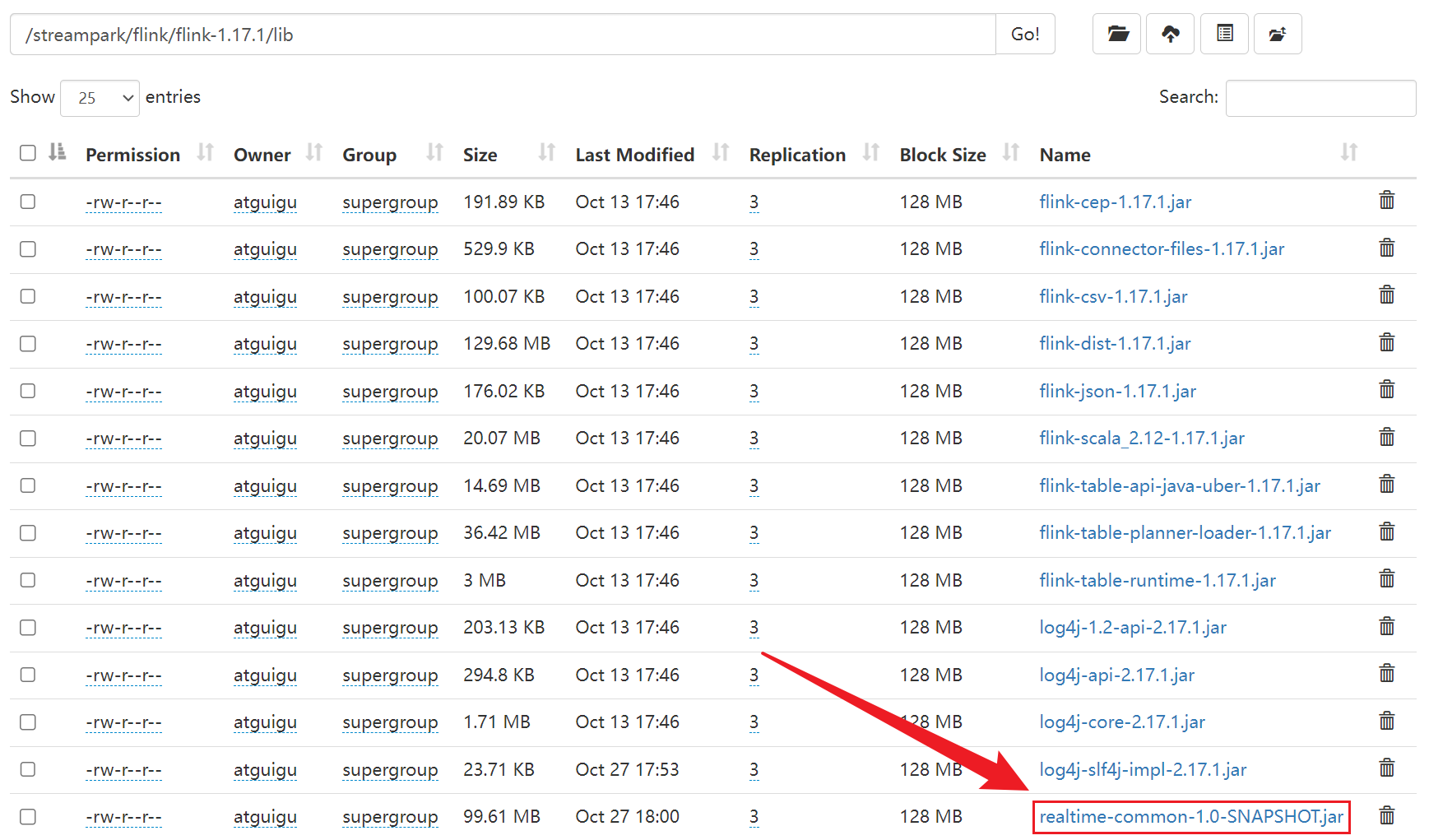
拷贝至本地目录

[atguigu@hadoop102 ~]$ cp /opt/module/streampark\_workspace/dist/100001/realtime-common-1.0-SNAPSHOT/realtime-common-1.0-SNAPSHOT.jar /opt/module/flink-1.17.1/lib/

上传至HDFS

[atguigu@hadoop102 ~]$ hadoop fs -put /opt/module/streampark\_workspace/dist/100001/realtime-common-1.0-SNAPSHOT/realtime-common-1.0-SNAPSHOT.jar /streampark/flink/flink-1.17.1/lib

查看HDFS目标路径



* + 1. 编译单个模块

如果修改了某个模块的代码，可以单独编译该模块，假设已完成代码修改并提交至GitLab。

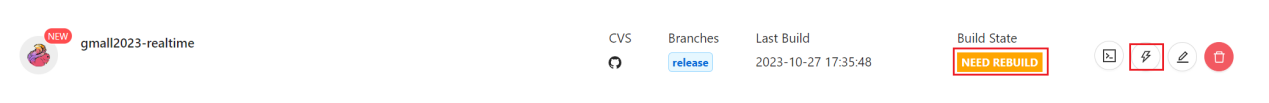
* + - * 1. 修改项目配置

将POM依赖配置为需要编译的子模块POM文件

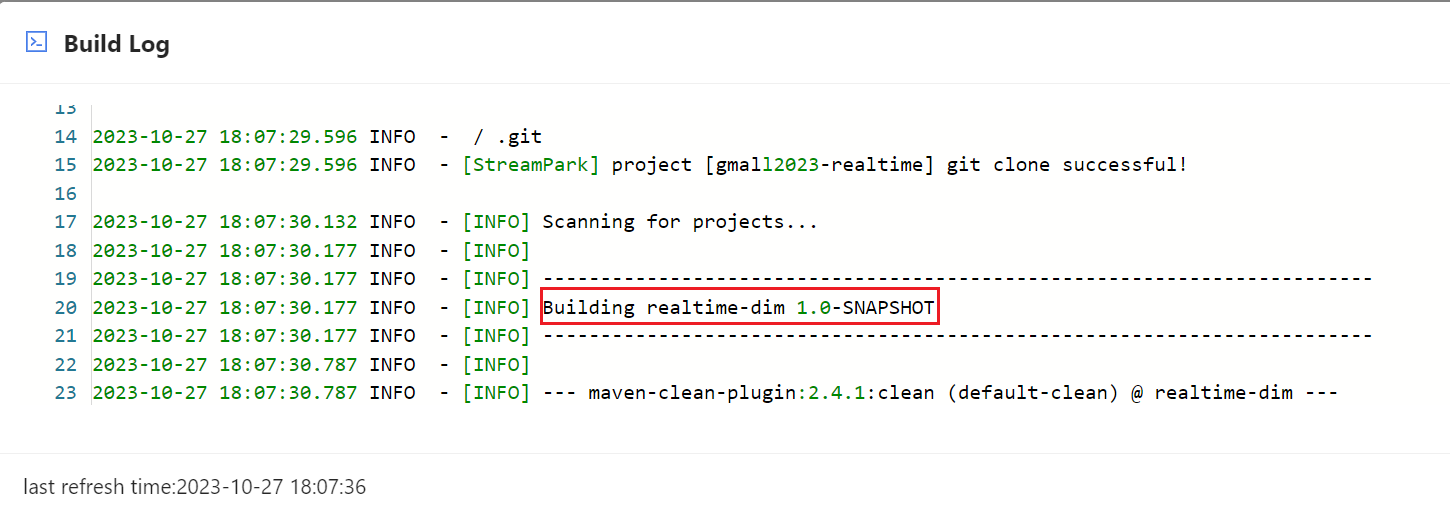


此处的路径为以父工程为基准的相对路径。

* + - * 1. 重新编译



* + - * 1. 查看日志



可以看到，此处只编译了目标模块。

* + 1. 配置作业
       - 1. 注释代码中的环境配置

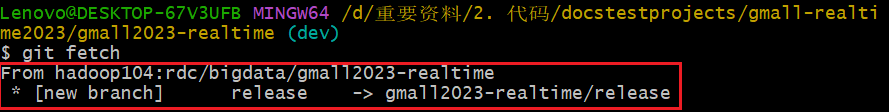
Flink作业的并行度等配置信息可以代码中指定，也可以在提交时指定，前者的优先级更高，部署前我们应该注释BaseApp和BaseSQLApp中的相关配置，提交至GitLab，重新编译，更新本地和HDFS的common包。这样，更改配置信息只需要修改提交参数，不必重新执行上述流程。

追踪release分支

打开git命令行终端

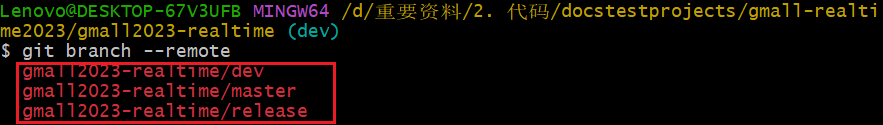
① 获取远端分支信息

$ git fetch



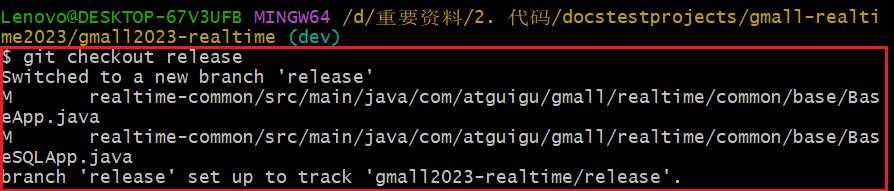
② 查看远端分支

$ git branch --remote



③ 追踪远端分支

$ git checkout release



修改BaseApp的start方法

public void start(int port, int parallelism, String ckAndGroupId, String topic) {

// 1. 环境准备

// 1.1 设置操作 Hadoop 的用户名为 Hadoop 超级用户 atguigu

System.setProperty("HADOOP\_USER\_NAME", "atguigu");

// 1.2 获取流处理环境，并指定本地测试时启动 WebUI 所绑定的端口

Configuration conf = **new** Configuration();

conf.setInteger("rest.port", port);

StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment(conf);

// 1.3 设置并行度

/\*

env.setParallelism(parallelism);

// 1.4 状态后端及检查点相关配置

// 1.4.1 设置状态后端

env.setStateBackend(new HashMapStateBackend());

// 1.4.2 开启 checkpoint

env.enableCheckpointing(5000);

// 1.4.3 设置 checkpoint 模式: 精准一次

env.getCheckpointConfig().setCheckpointingMode(CheckpointingMode.EXACTLY\_ONCE);

// 1.4.4 checkpoint 存储

env.getCheckpointConfig().setCheckpointStorage("hdfs://hadoop102:8020/gmall2023/stream/" + ckAndGroupId);

// 1.4.5 checkpoint 并发数

env.getCheckpointConfig().setMaxConcurrentCheckpoints(1);

// 1.4.6 checkpoint 之间的最小间隔

env.getCheckpointConfig().setMinPauseBetweenCheckpoints(500);

// 1.4.7 checkpoint 的超时时间

env.getCheckpointConfig().setCheckpointTimeout(10000);

// 1.4.8 job 取消时 checkpoint 保留策略

env.getCheckpointConfig().setExternalizedCheckpointCleanup(RETAIN\_ON\_CANCELLATION);

\*/

// 1.5 从 Kafka 目标主题读取数据，封装为流

KafkaSource<String> source = FlinkSourceUtil.getKafkaSource(ckAndGroupId, topic);

DataStreamSource<String> stream = env.fromSource(source, WatermarkStrategy.noWatermarks(), "kafka\_source");

// 2. 执行具体的处理逻辑

handle(env, stream);

// 3. 执行 Job

**try** {

env.execute();

} catch (Exception e) {

e.printStackTrace();

}

}

修改BaseSQLApp的start方法

public void start(int port,

int parallelism,

String ck) {

System.setProperty("HADOOP\_USER\_NAME", "atguigu");

Configuration conf = **new** Configuration();

conf.setInteger("rest.port", port);

StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment(conf);

/\*

env.setParallelism(parallelism);

// 1. 设置状态后端

env.setStateBackend(new HashMapStateBackend());

// 2. 开启 checkpoint

env.enableCheckpointing(5000);

// 3. 设置 checkpoint 模式: 精准一次

env.getCheckpointConfig().setCheckpointingMode(CheckpointingMode.EXACTLY\_ONCE);

// 4. checkpoint 存储

env.getCheckpointConfig().setCheckpointStorage("hdfs://hadoop102:8020/gmall2023/sql/" + ck);

// 5. checkpoint 并发数

env.getCheckpointConfig().setMaxConcurrentCheckpoints(1);

// 6. checkpoint 之间的最小间隔

env.getCheckpointConfig().setMinPauseBetweenCheckpoints(500);

// 7. checkpoint 的超时时间

env.getCheckpointConfig().setCheckpointTimeout(10000);

// 8. job 取消的时候的, checkpoint 保留策略

env.getCheckpointConfig().setExternalizedCheckpointCleanup(RETAIN\_ON\_CANCELLATION);

\*/

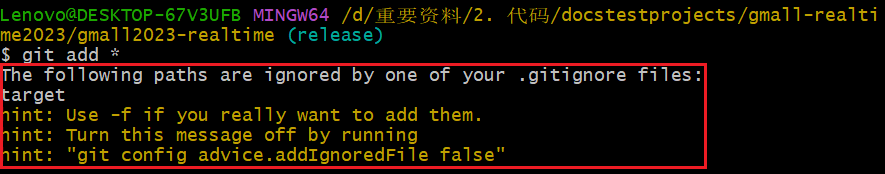
StreamTableEnvironment tEnv = StreamTableEnvironment.create(env);

handle(env, tEnv);

}

提交代码至本地仓库

$ git **add** \*

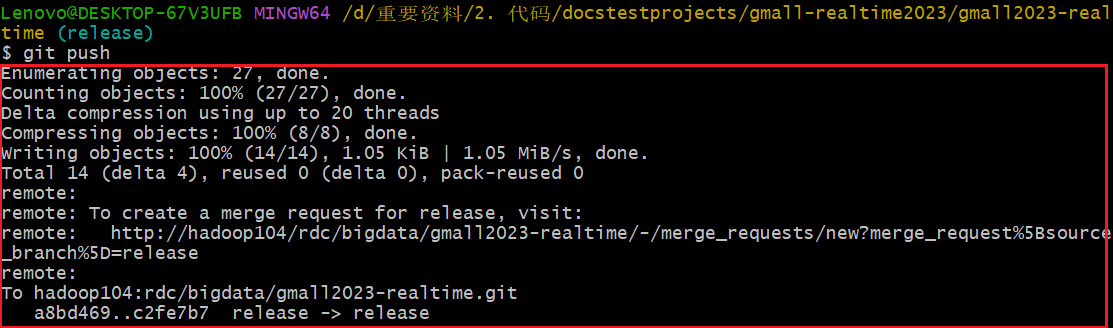


$ git commit -m "部署前注释代码中的配置操作"



提交代码至远端

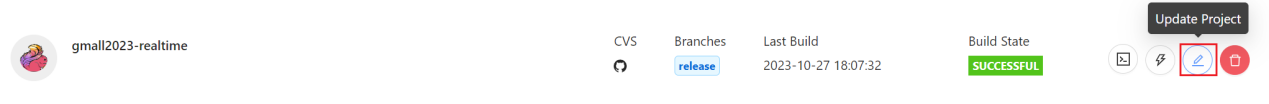
$ git **push**

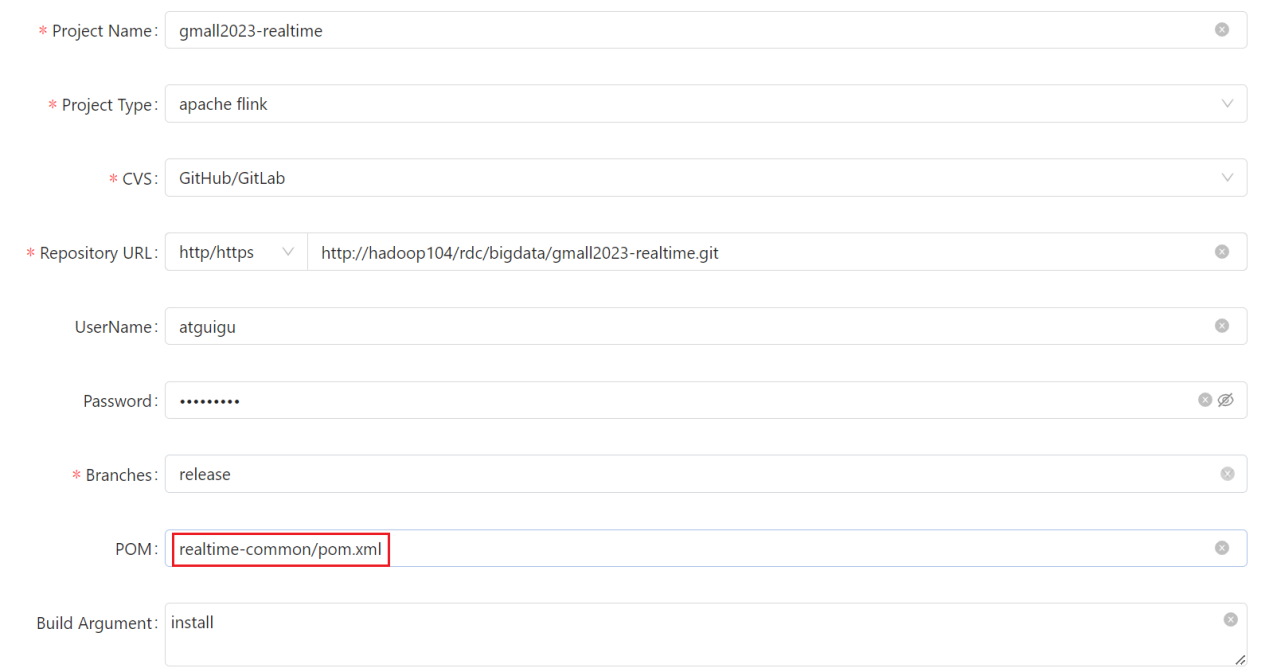


* + - * 1. 重新编译common模块

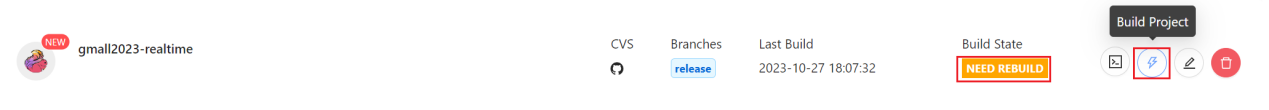
修改了common模块代码，需要重新编译生效

修改POM路径

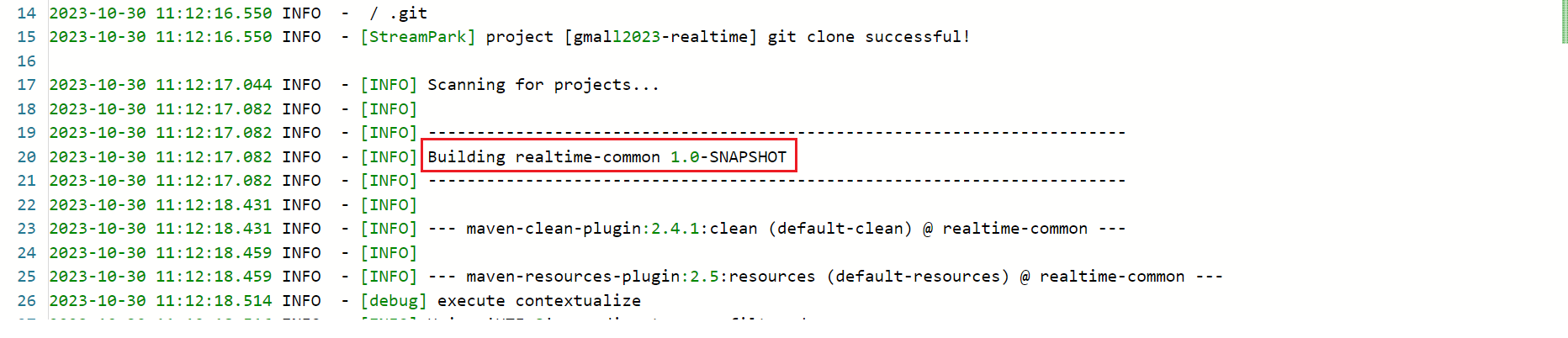


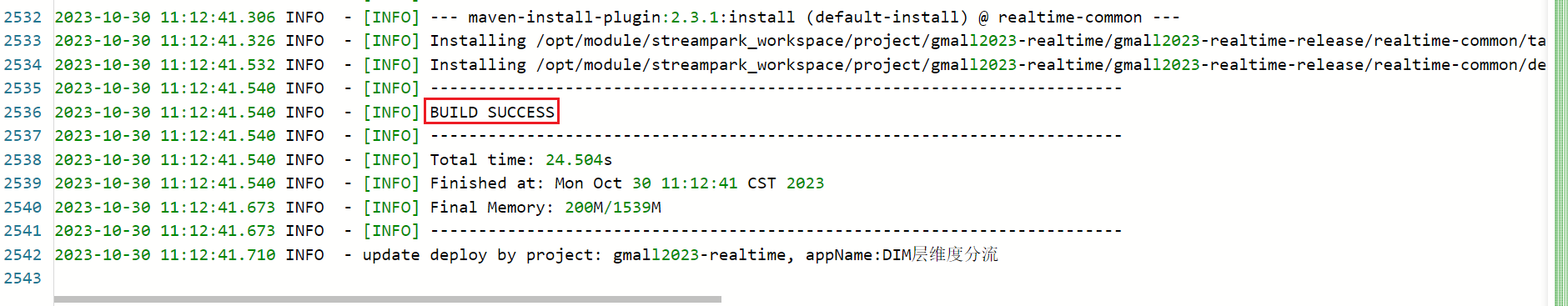


重新编译



完成





* + - * 1. 重新上传realtime-common-1.0-SNAPSHOT.jar

替换HDFS /streampark/flink/flink-1.17.1/lib的realtime-common-1.0-SNAPSHOT.jar

略。

替换本地/opt/module/flink-1.17.1/lib/下的realtime-common-1.0-SNAPSHOT.jar

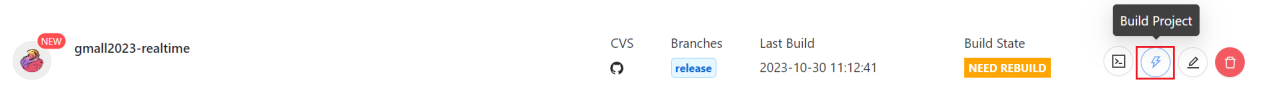
略。

* + - * 1. 修改POM路径

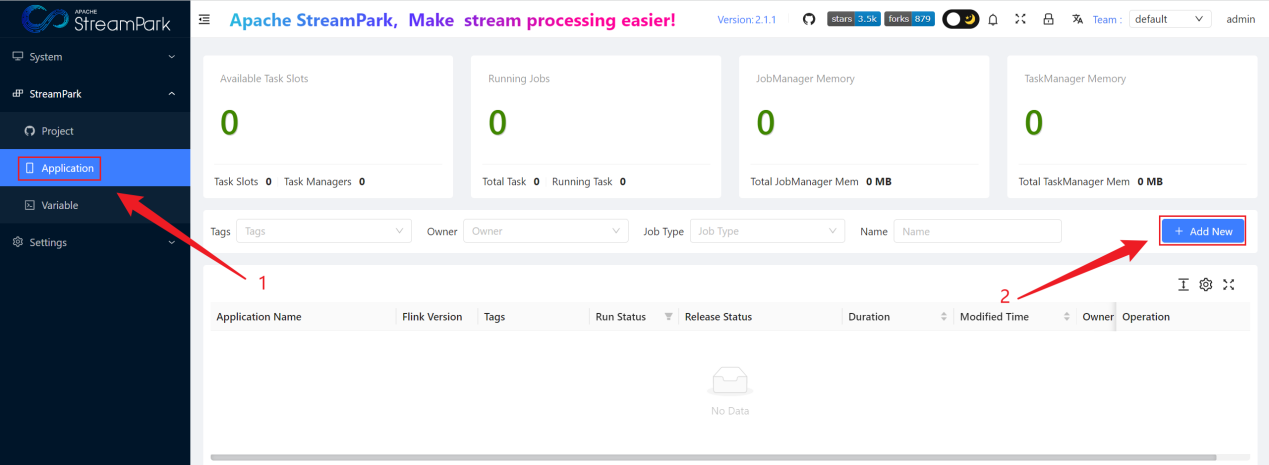
将POM路径修改为业务module POM路径。



重新编译



* + - * 1. 新建作业



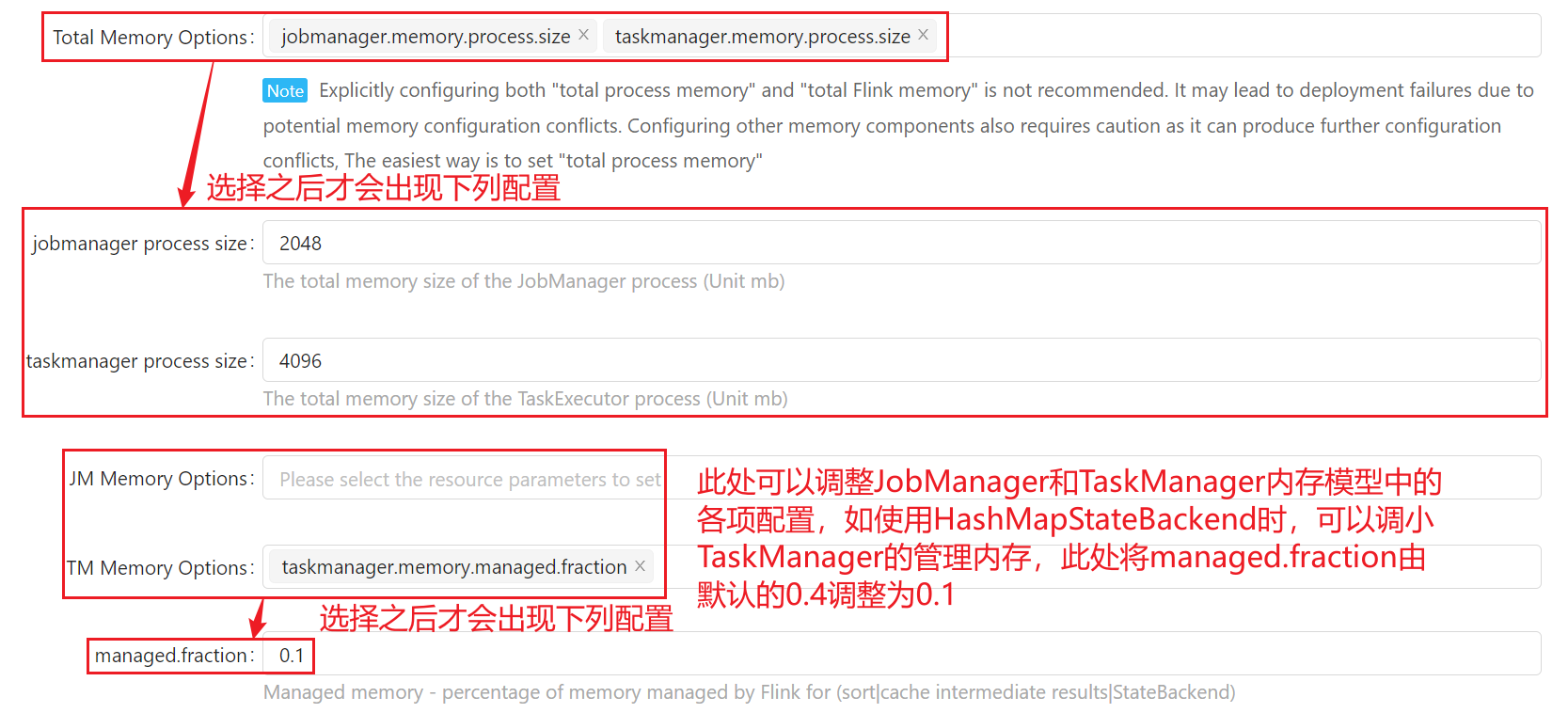
* + - * 1. 配置作业

基本配置





内存配置

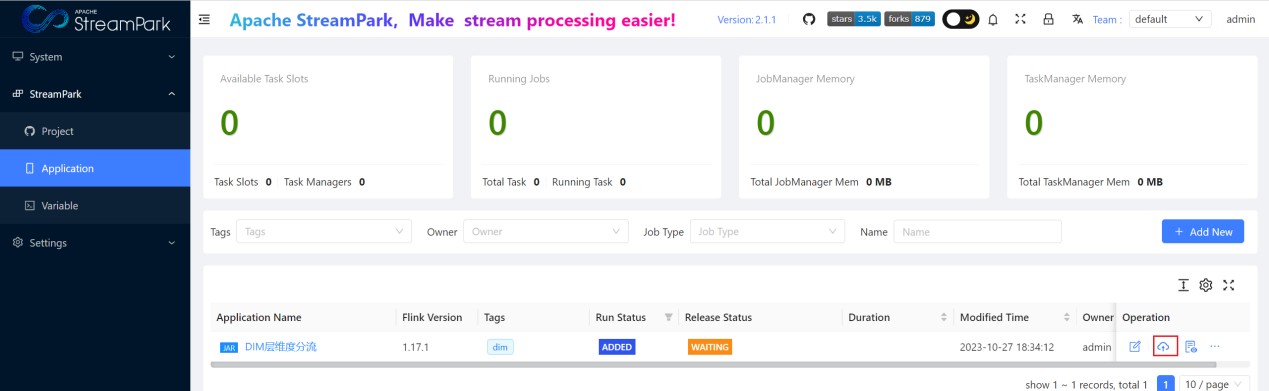


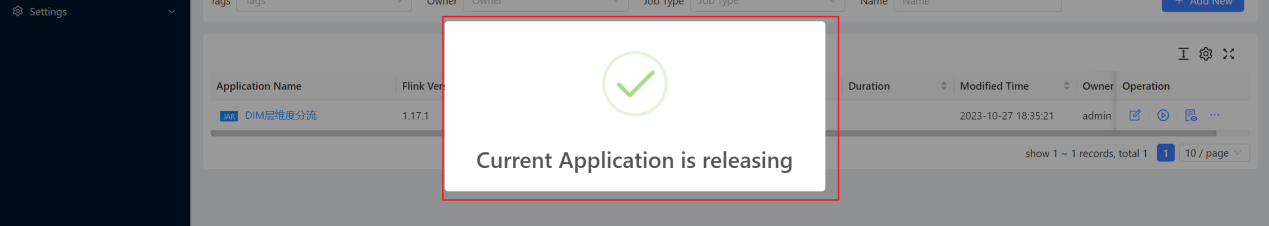
其它配置

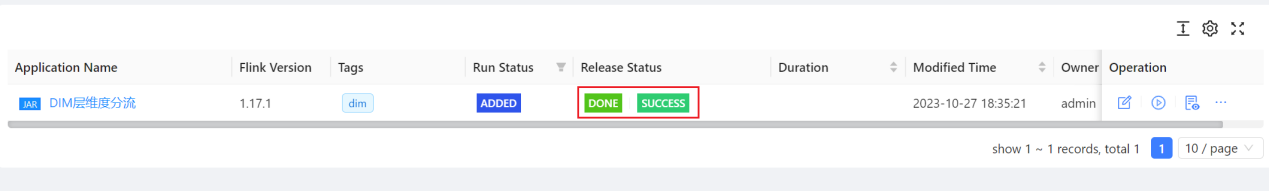




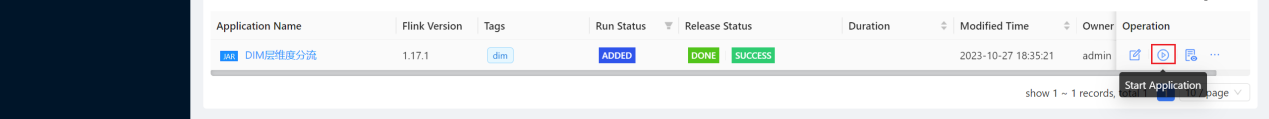
* + 1. 发布和启动作业
       - 1. 发布作业





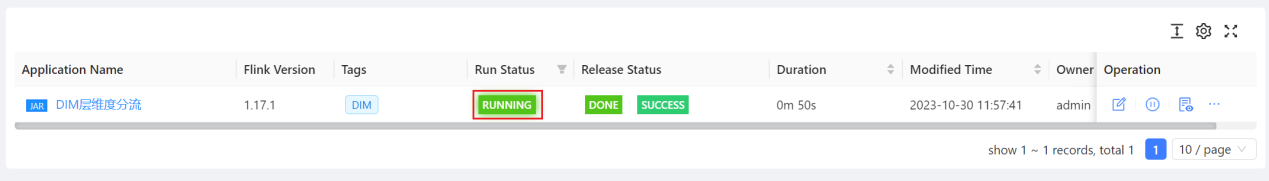


* + - * 1. 启动作业





* + - * 1. 查看作业状态

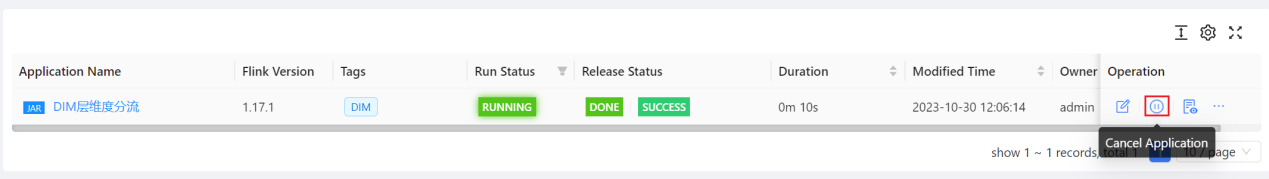


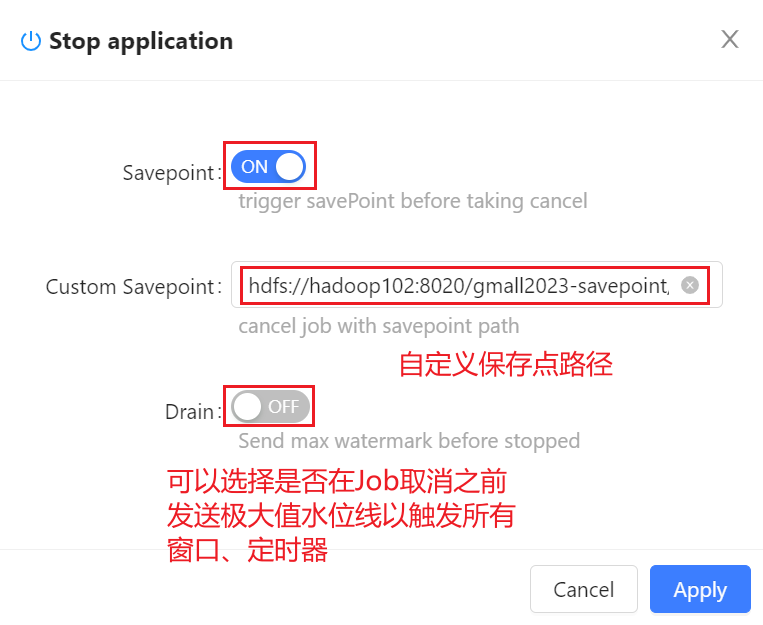


作业正在运行，部署完成。

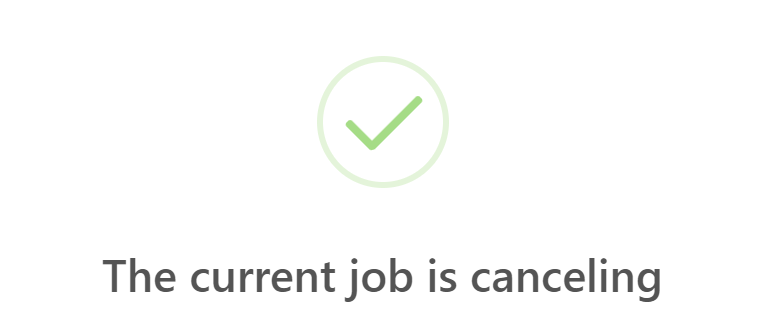
* + - * 1. 取消Job并触发保存点

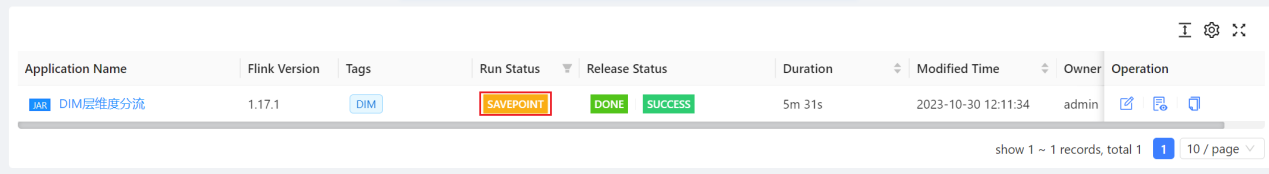
取消Job



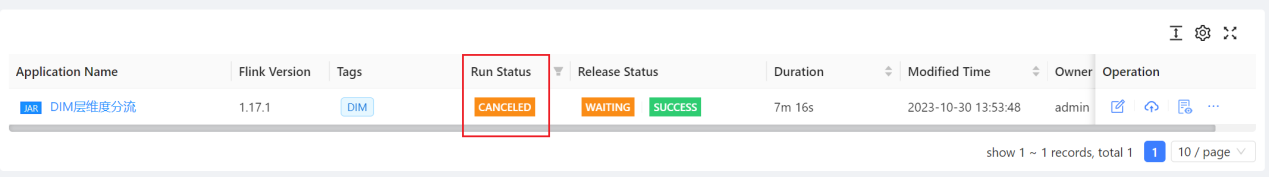


正在取消Job并触发保存点

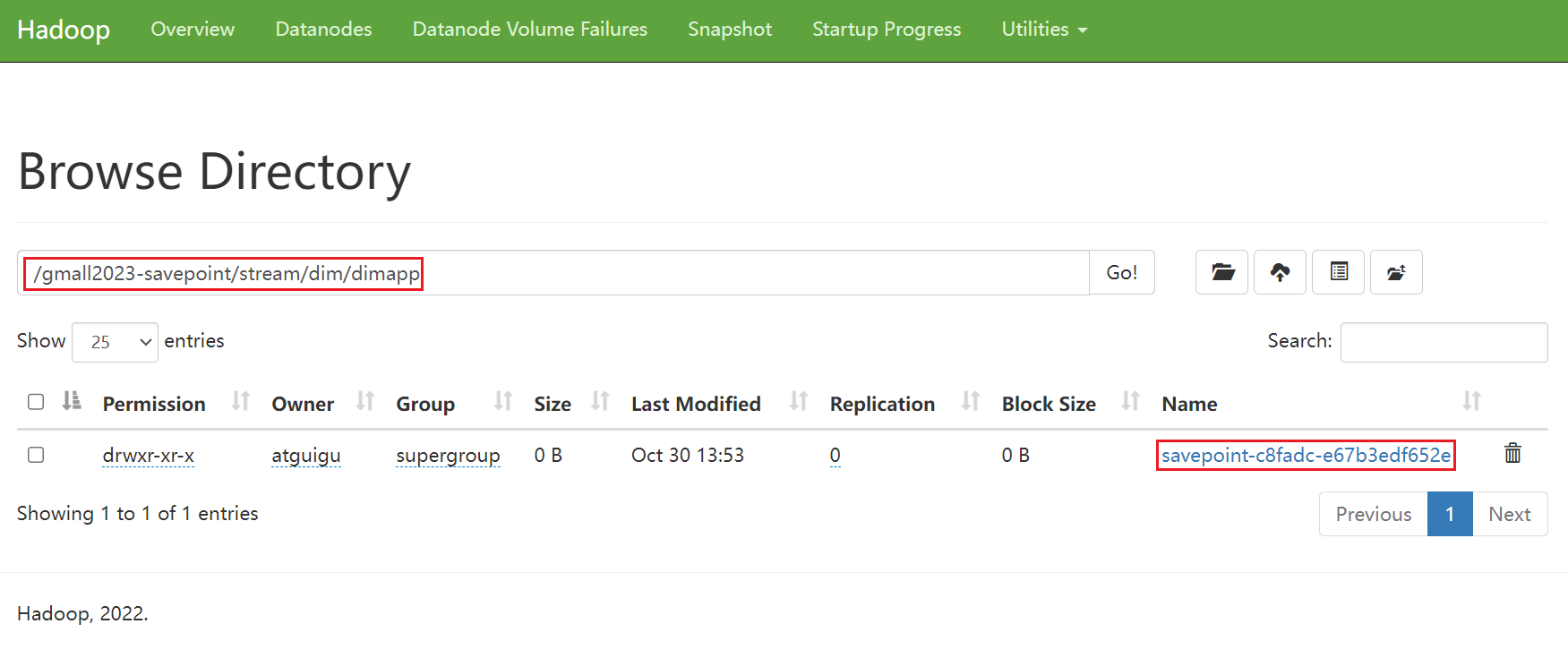




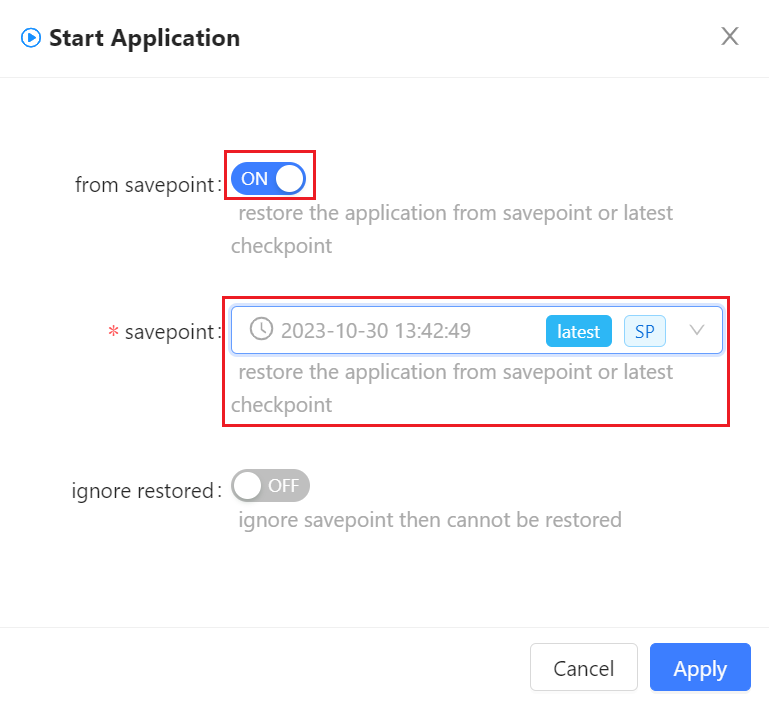
Job已取消



查看保存点路径

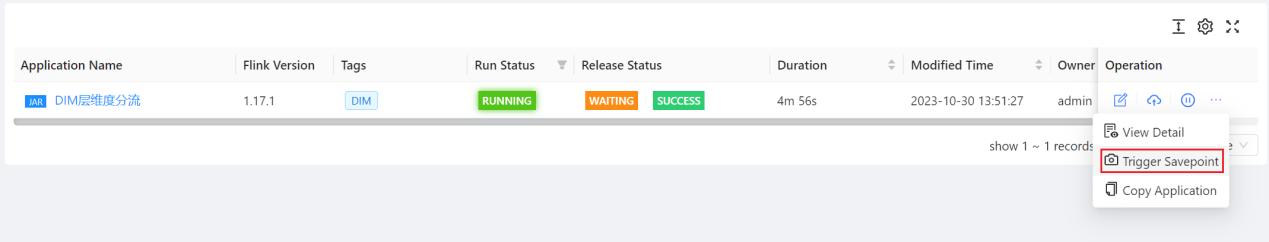


触发保存点后，再次启动，可以从保存点启动



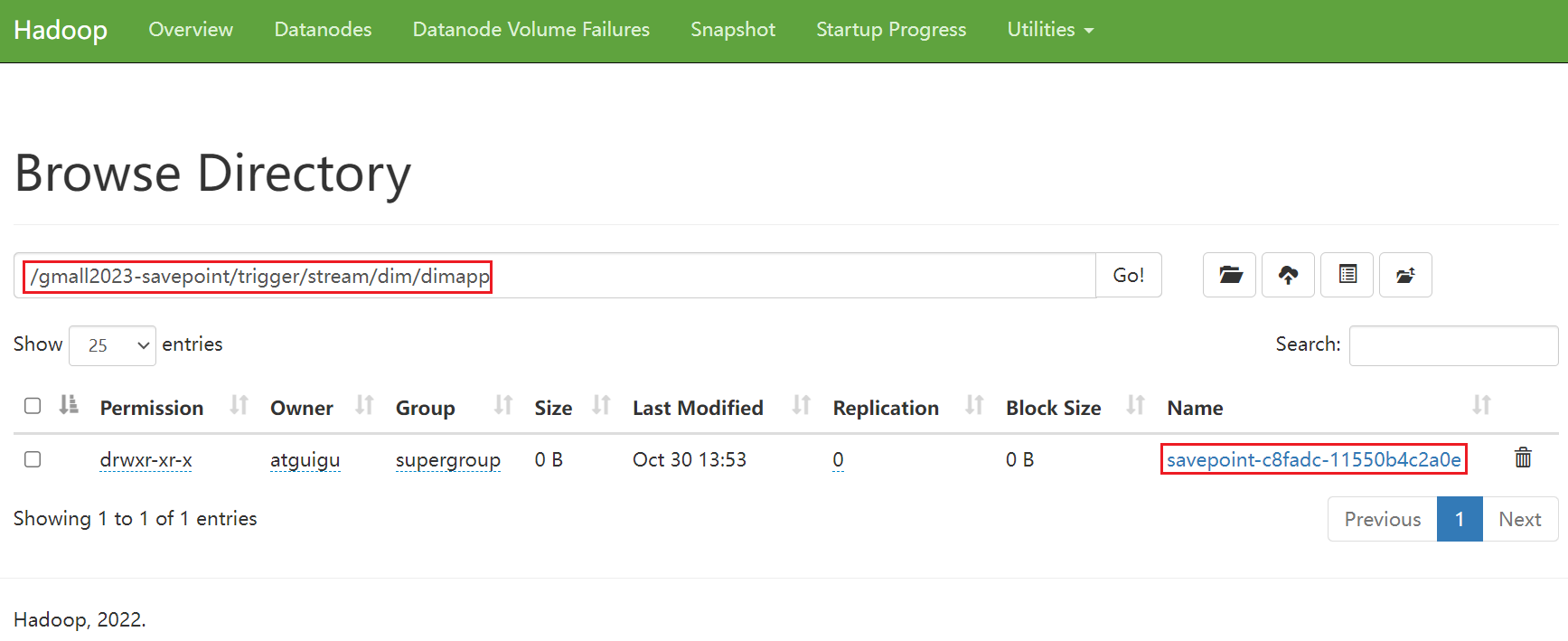
* + - * 1. 触发保存点但不停止Job

触发保存点





查看目标路径



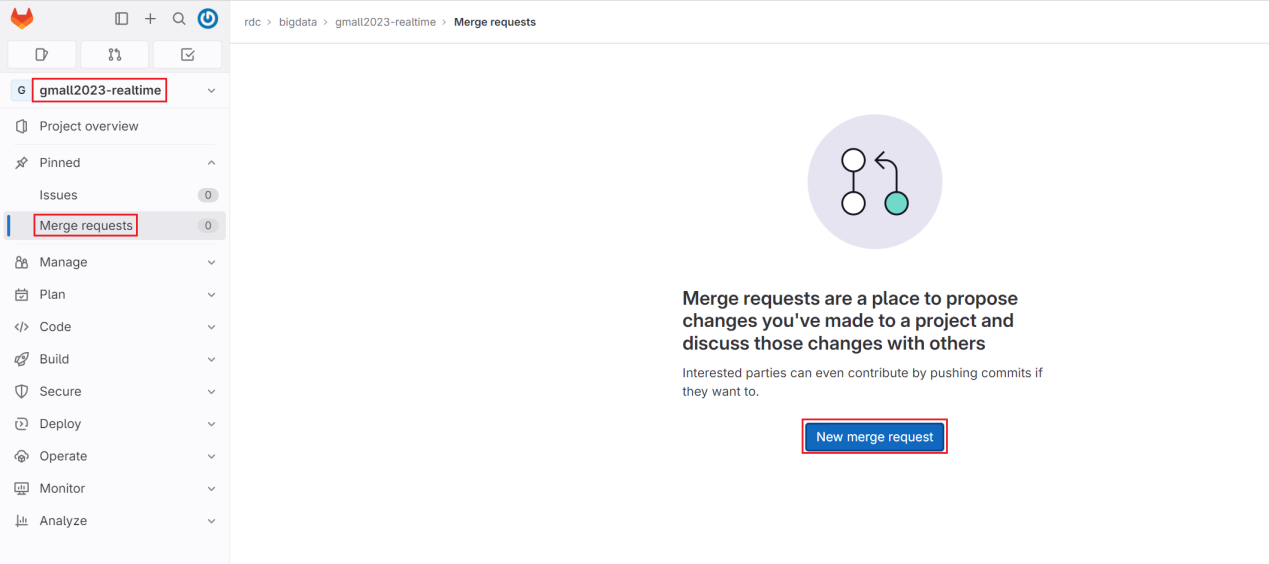
* + 1. 合并至master和dev分支

按照上述流程依次部署测试每个Job，此处不再一一演示。

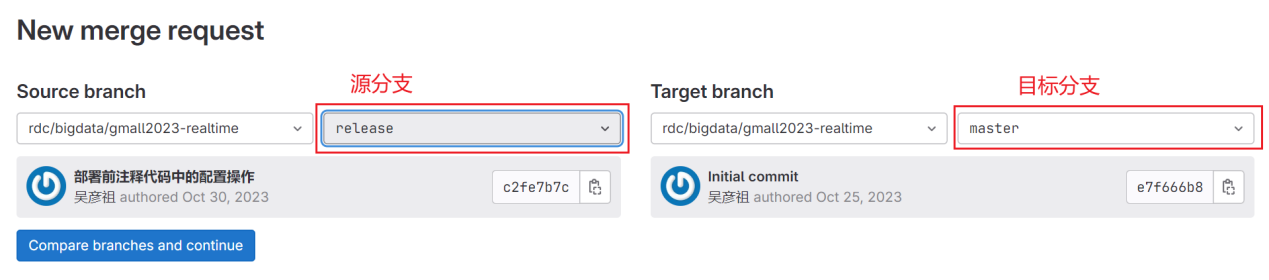
假设我们已经完成了所有Job的测试，接下来准备正式发布1.0版本。

* + - * 1. 将release分支合并至master分支

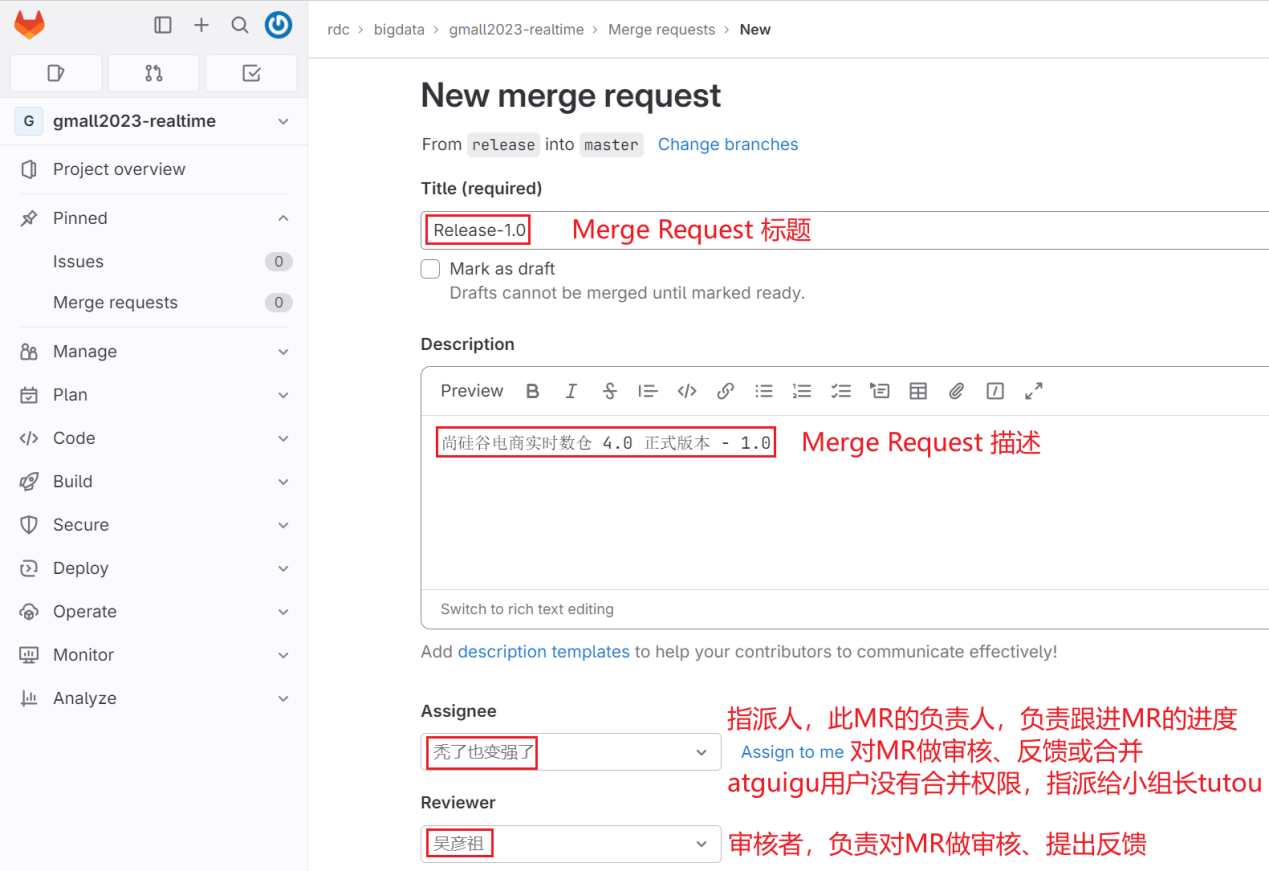
新建Merge Request



选择源分支和目标分支



配置Merge Request



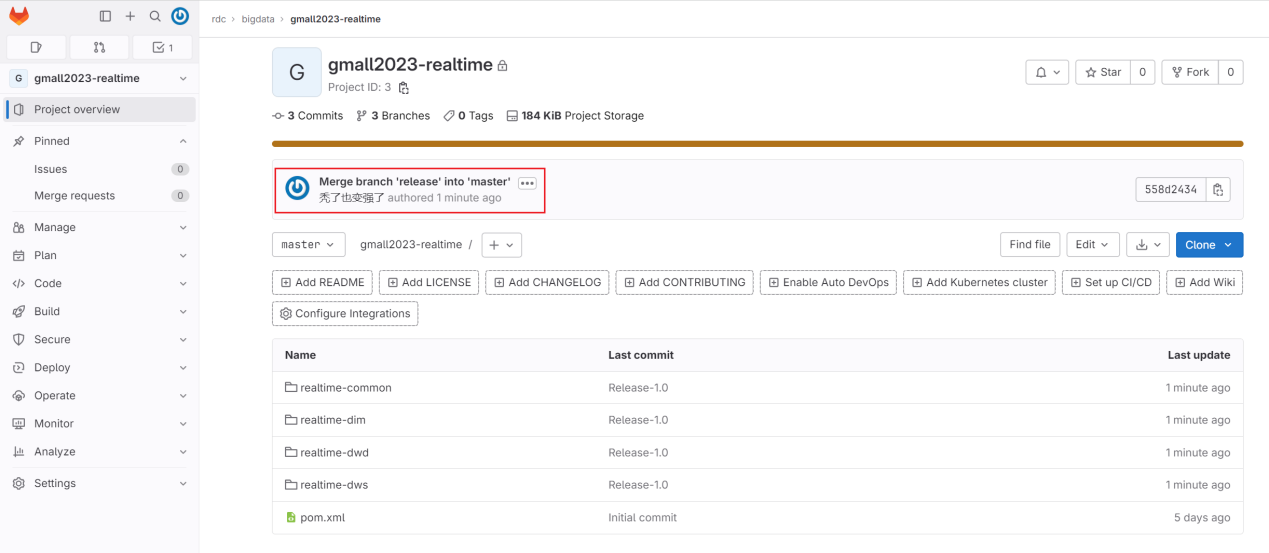


批准Merge Request，完成合并

切换到tutou用户，批准Merge Request。



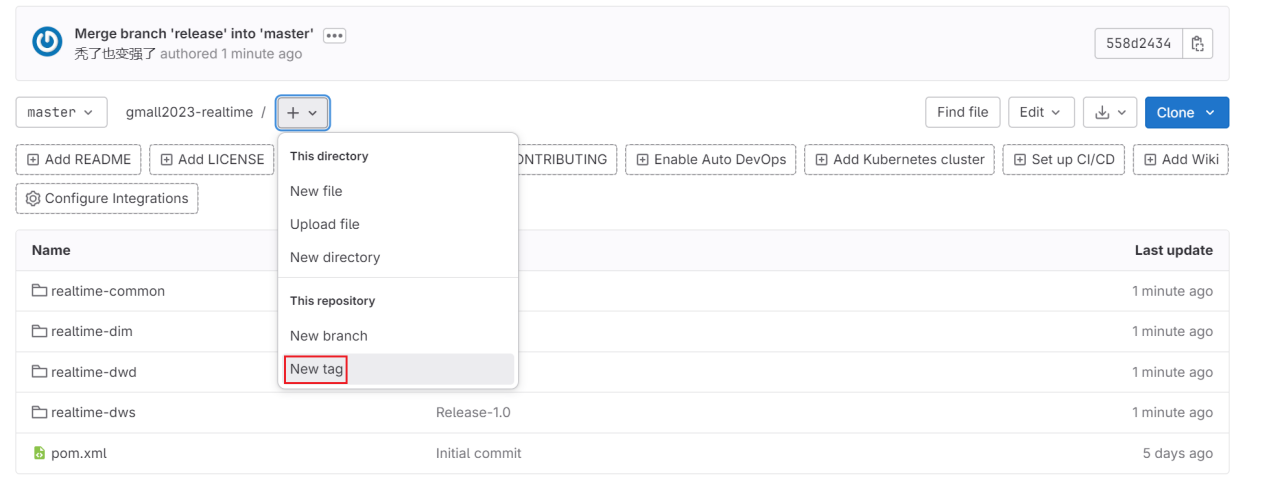
查看master分支



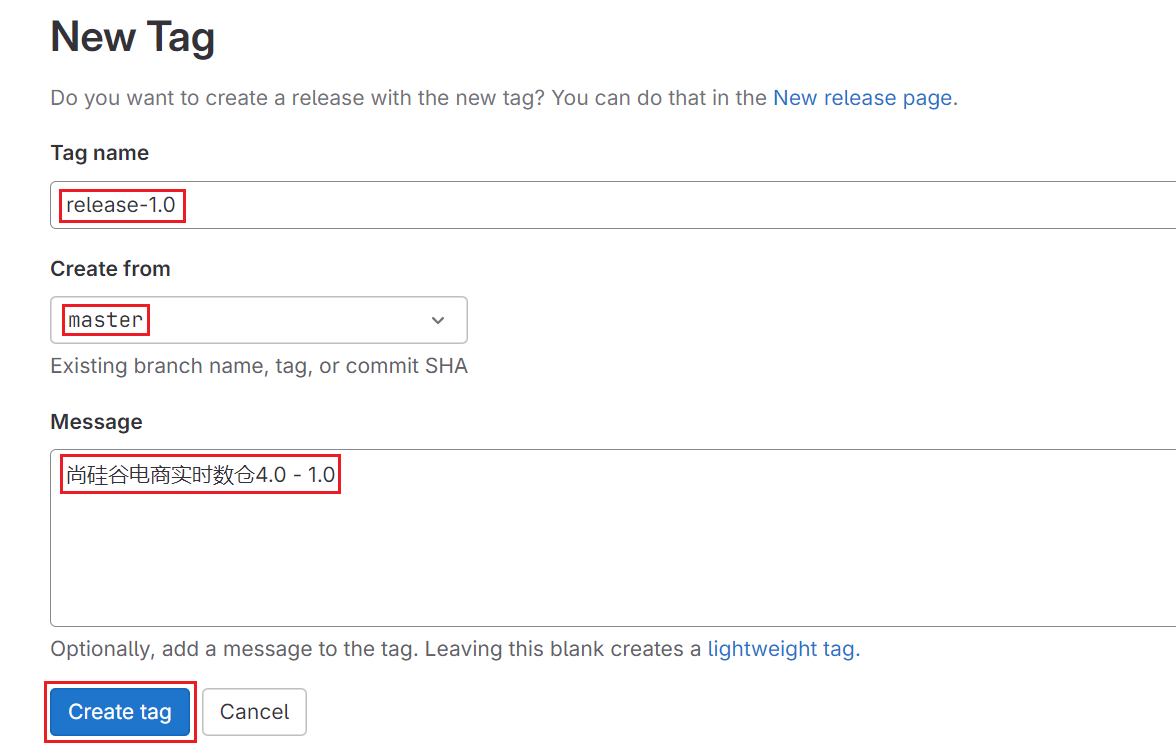
已将release分支合并至master分支

打标签

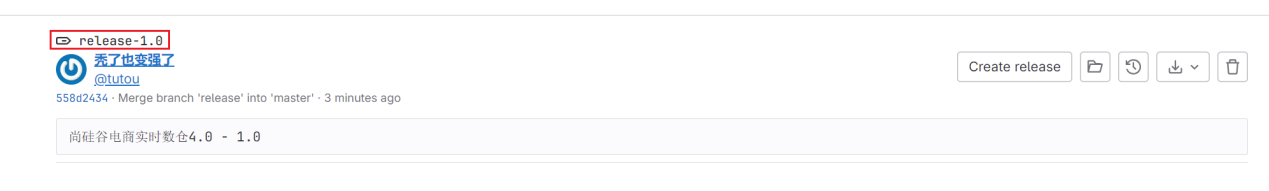
① 新建标签



② 配置标签



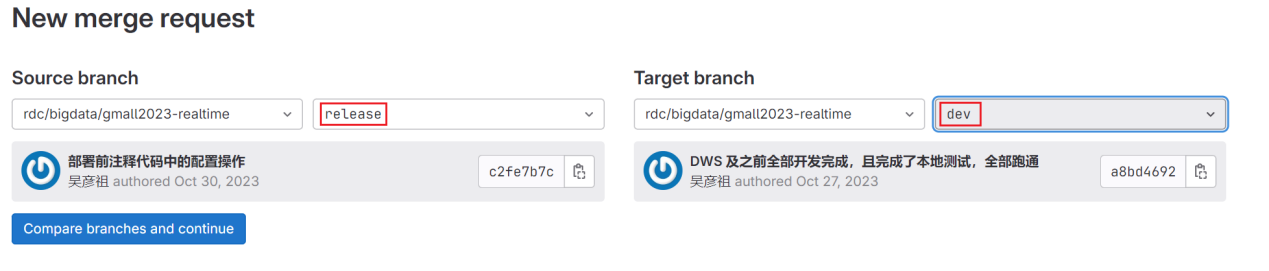
③ 查看已完成的标签



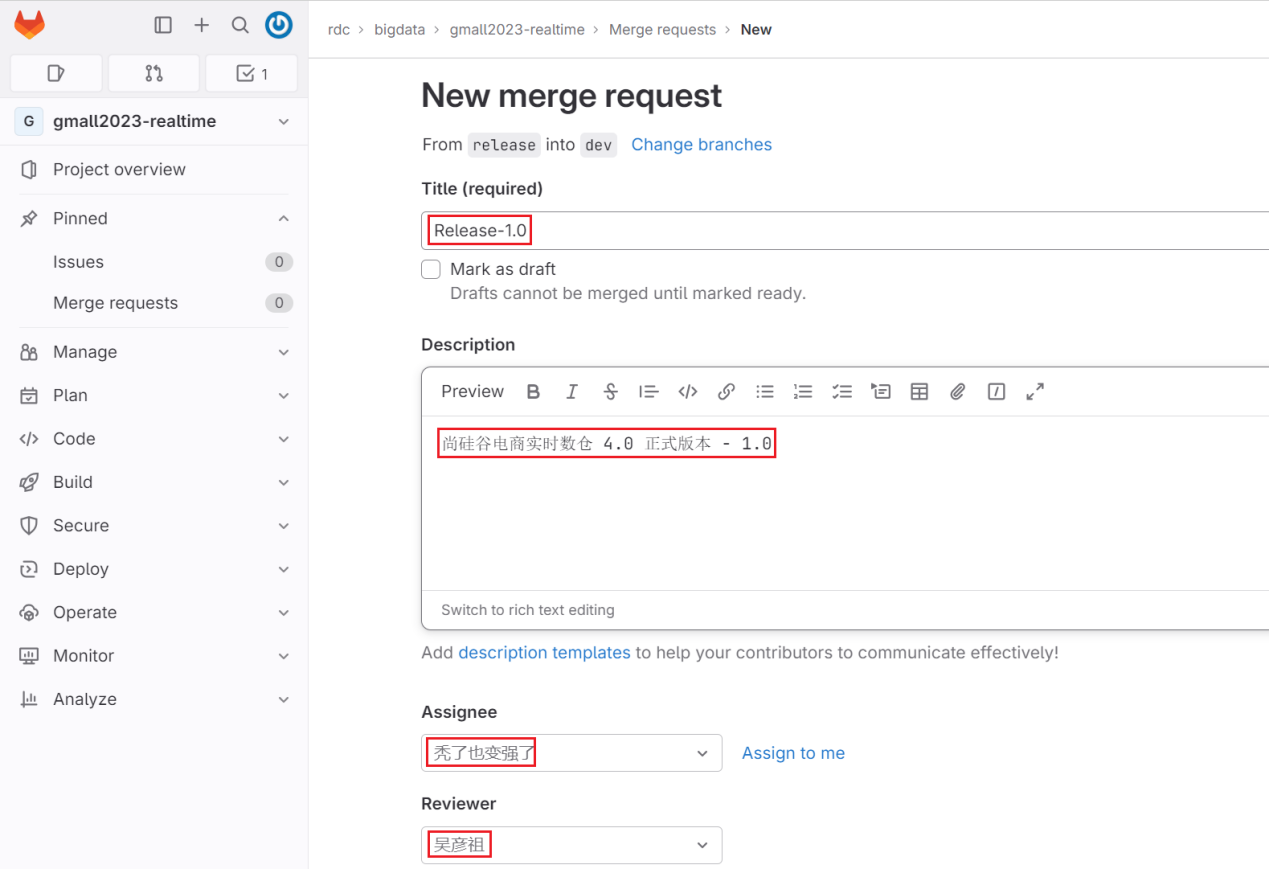
* + - * 1. 将release分支合并至dev分支

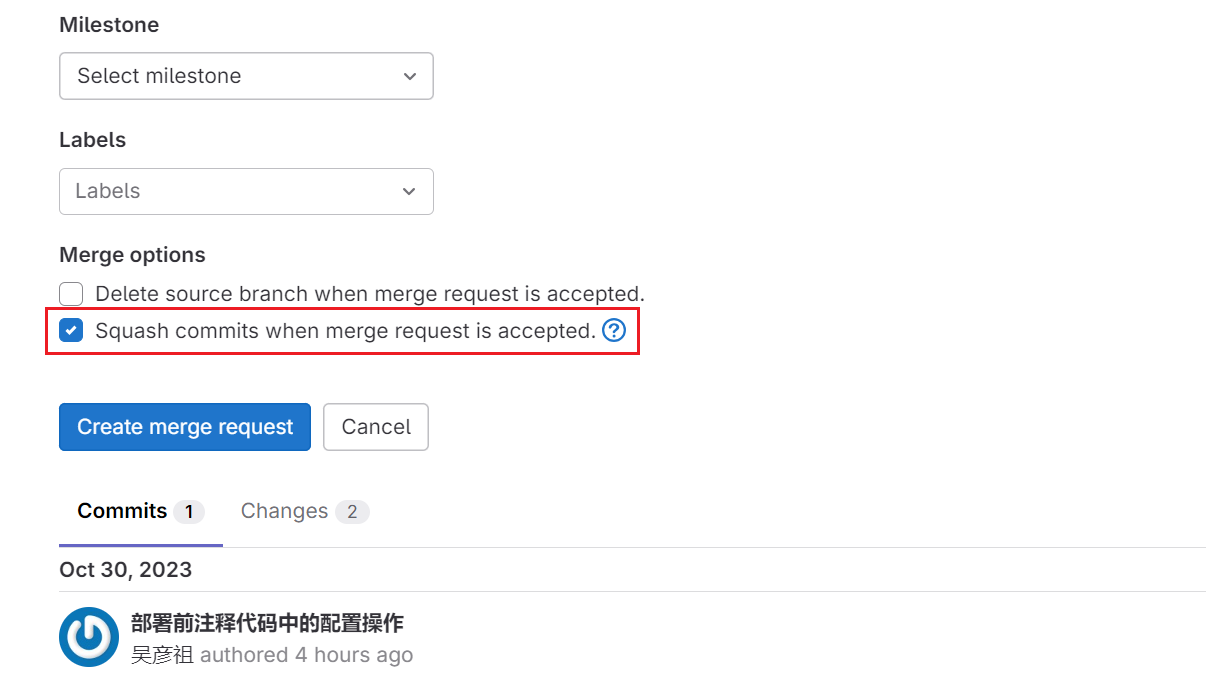
回到atguigu用户

新建MR



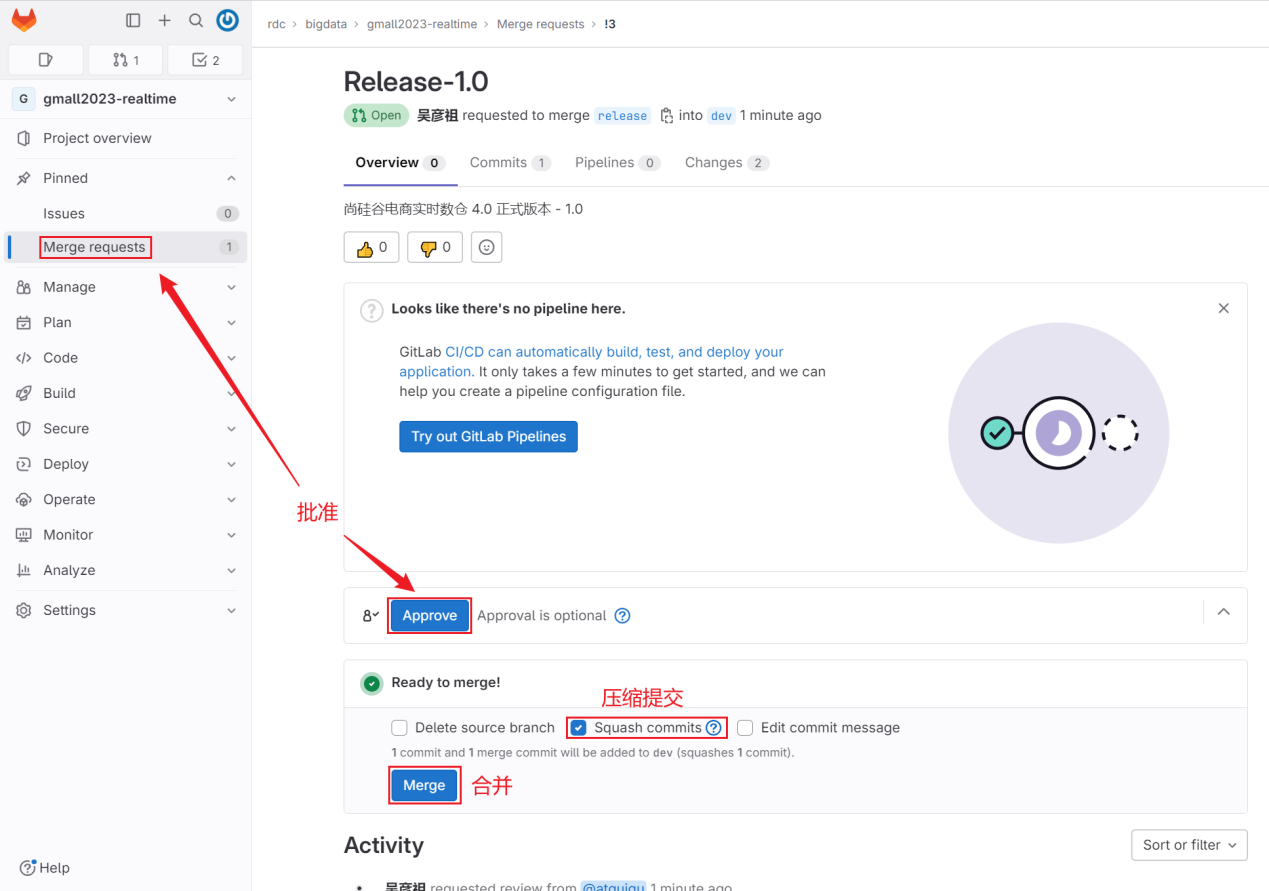
配置MR



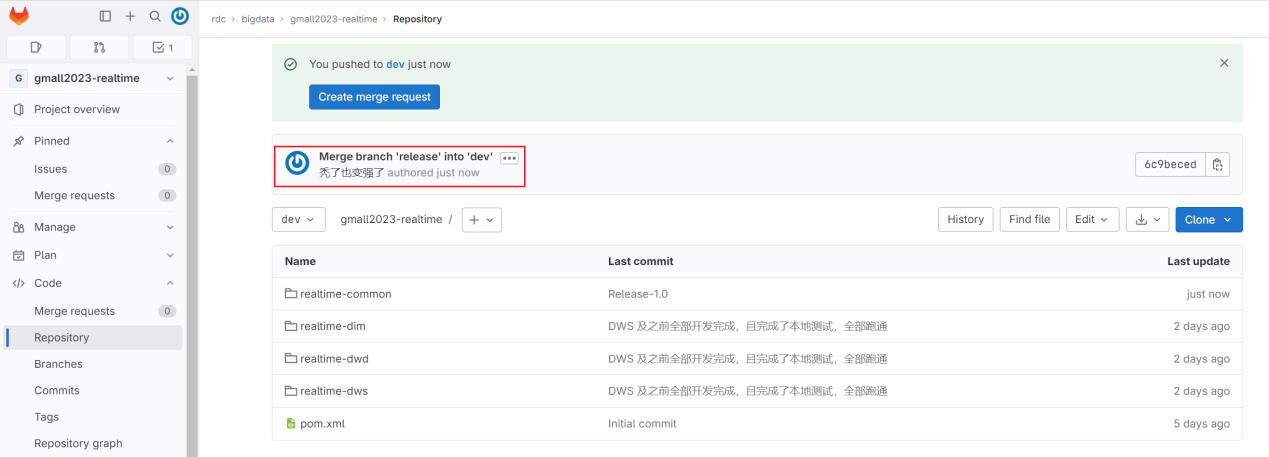


批准MR，完成合并

切换到tutou用户



查看dev分支



已完成合并