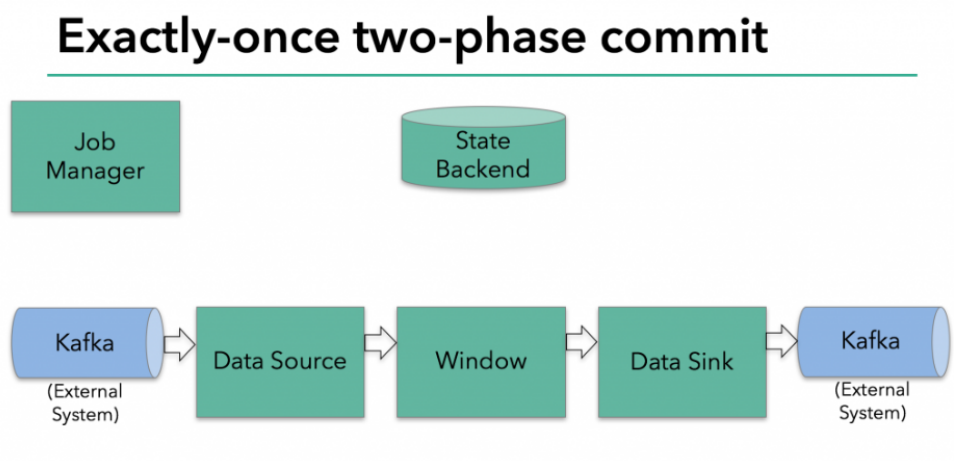
- 需求

接下来将介绍两阶段提交协议，以及它如何在一个读写Kafka的Flink程序中实现端到端的Exactly-Once语义。Kafka经常与Flink一起使用，且Kafka在最近的0.11版本中添加了对事务的支持。这意味着现在通过Flink读写Kafaka，并提供端到端的Exactly-Once语义有了必要的支持。



在上图中，我们有：

- 从Kafka读取的数据源（Flink内置的KafkaConsumer）
- 窗口聚合
- 将数据写回Kafka的数据输出端（Flink内置的KafkaProducer）

要使数据输出端提供Exactly-Once保证，它必须将所有数据通过一个事务提交给Kafka。提交捆绑了两个checkpoint之间的所有要写入的数据。这可确保在发生故障时能回滚写入的数据。

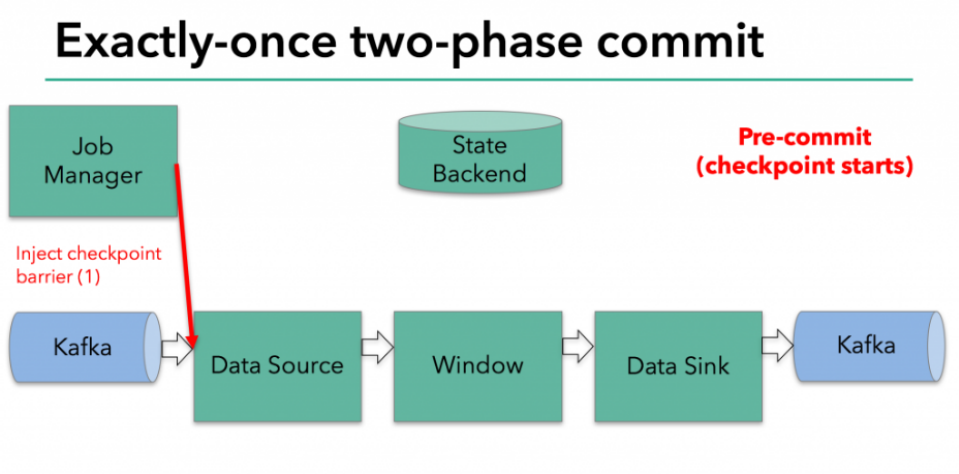
但是在分布式系统中，通常会有多个并发运行的写入任务的，简单的提交或回滚是不够的，因为所有组件必须在提交或回滚时"一致"才能确保一致的结果。

Flink使用两阶段提交协议及预提交阶段来解决这个问题。

- 预提交-内部状态

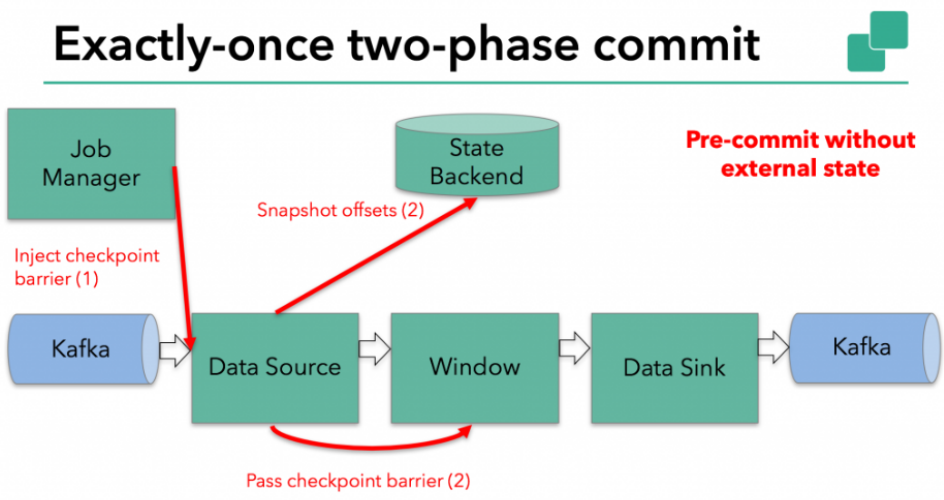
在checkpoint开始的时候，即两阶段提交协议的"预提交"阶段。当checkpoint开始时，Flink的JobManager会将checkpoint barrier（将数据流中的记录分为进入当前checkpoint与进入下一个checkpoint）注入数据流。

brarrier在operator之间传递。对于每一个operator，它触发operator的状态快照写入到state backend。



数据源保存了消费Kafka的偏移量(offset)，之后将checkpoint barrier传递给下一个operator。

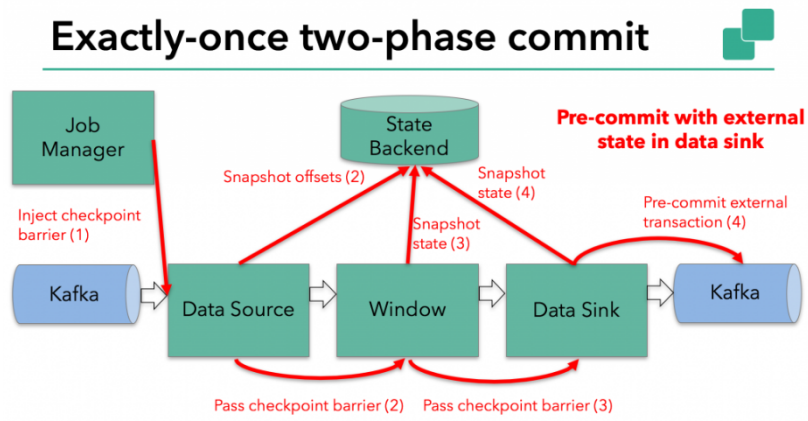
这种方式仅适用于operator具有『内部』状态。所谓内部状态，是指Flink state backend保存和管理的 -例如，第二个operator中window聚合算出来的sum值。当一个进程有它的内部状态的时候，除了在checkpoint之前需要将数据变更写入到state backend，不需要在预提交阶段执行任何其他操作。Flink负责在checkpoint成功的情况下正确提交这些写入，或者在出现故障时中止这些写入。



- 预提交-外部状态

但是，当进程具有『外部』状态时，需要作些额外的处理。外部状态通常以写入外部系统（如Kafka）的形式出现。在这种情况下，为了提供Exactly-Once保证，外部系统必须支持事务，这样才能和两阶段提交协议集成。

在该示例中的数据需要写入Kafka，因此数据输出端（Data Sink）有外部状态。在这种情况下，在预提交阶段，除了将其状态写入state backend之外，数据输出端还必须预先提交其外部事务。

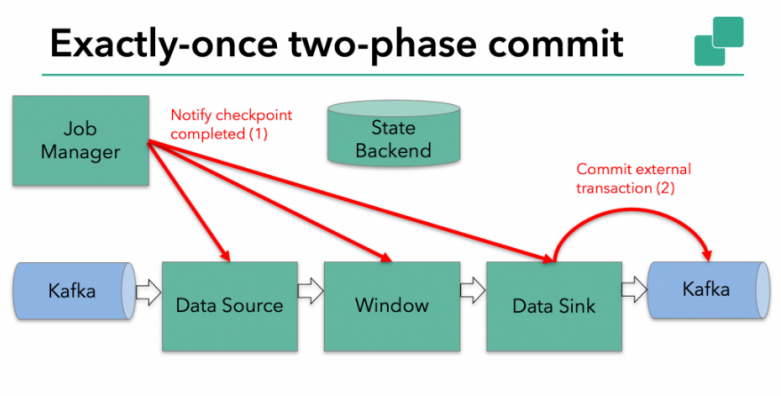


当checkpoint barrier在所有operator都传递了一遍，并且触发的checkpoint回调成功完成时，预提交阶段就结束了。所有触发的状态快照都被视为该checkpoint的一部分。checkpoint是整个应用程序状态的快照，包括预先提交的外部状态。如果发生故障，我们可以回滚到上次成功完成快照的时间点。

- 提交阶段

下一步是通知所有operator，checkpoint已经成功了。这是两阶段提交协议的提交阶段，JobManager为应用程序中的每个operator发出checkpoint已完成的回调。

数据源和widnow operator没有外部状态，因此在提交阶段，这些operator不必执行任何操作。但是，数据输出端（Data Sink）拥有外部状态，此时应该提交外部事务。



- 总结

我们对上述知识点总结下：

1.一旦所有operator完成预提交，就提交一个commit。

2.如果只要有一个预提交失败，则所有其他提交都将中止，我们将回滚到上一个成功完成的checkpoint。

3.在预提交成功之后，提交的commit需要保证最终成功 – operator和外部系统都需要保障这点。如果commit失败（例如，由于间歇性网络问题），整个Flink应用程序将失败，应用程序将根据用户的重启策略重新启动，还会尝试再提交。这个过程至关重要，因为如果commit最终没有成功，将会导致数据丢失。

4.完整的实现两阶段提交协议可能有点复杂，这就是为什么Flink将它的通用逻辑提取到抽象类TwoPhaseCommitSinkFunction中的原因。