Scaling for Real

PunchPlatform team

Agenda

Problem Statement

What you expect from big streaming data platform is simple :

- scale to hundreds of thousands of events per sec
- do not loose any
- be cheap yet resilient

In this presentation we highlight the PunchPlatform way to solve this equation with a focus on writing the events (typically logs) to disk.

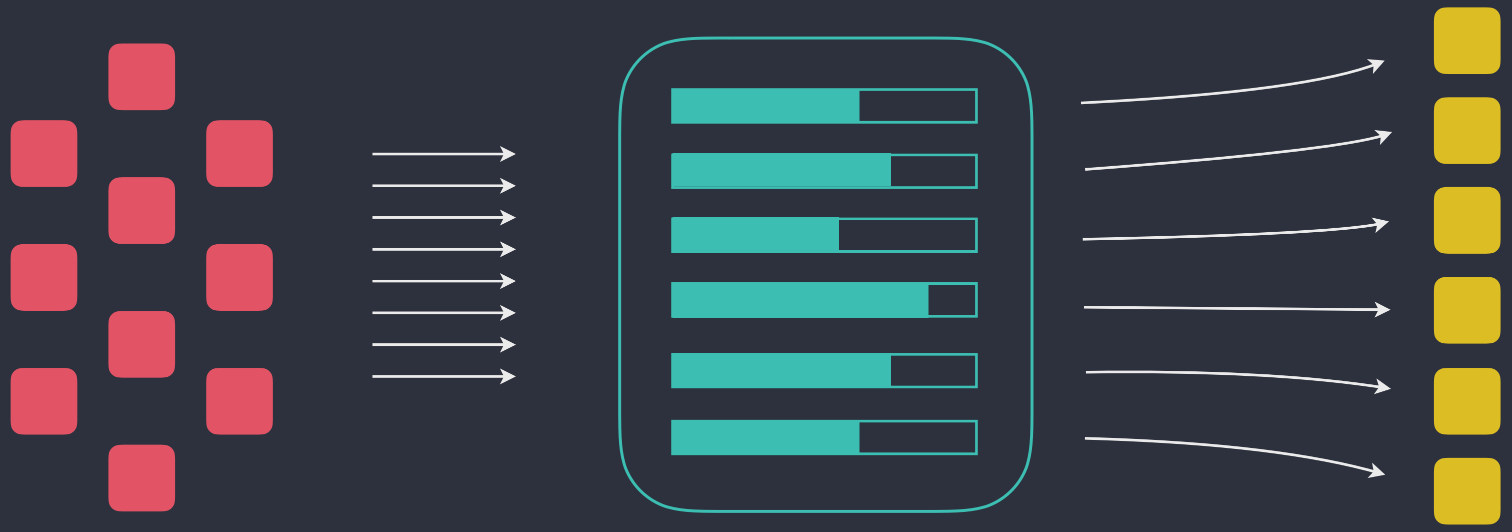
What is explained hereafter actually applies to writing to Kafka, ElasticSearch, Cassandra, a shared FileSystems, a CEPH Cluster, HDFS.

As a leading example we will refer to the CEPH cluster.

Reading the Data

The PunchPlatform is built on top of Kafka. That is where we read the logs from. Kafka can scale up to very large clusters (checkout the Linkedin numbers), takes care of replicating the data, and let several consumers consume the data in real time.

In order to scale, producer and consumers write (read) the data in ***multi partitioned topics***. Think of a *topic* as a giant fifo structure, divided in several sub-queues called *partitions*. Several **producers** write logs. Several **consumers** read them in real time. The key factor to scale is the number of partitions. You have one reader for each partition. Each must read fast enough to continuously consume its partition.

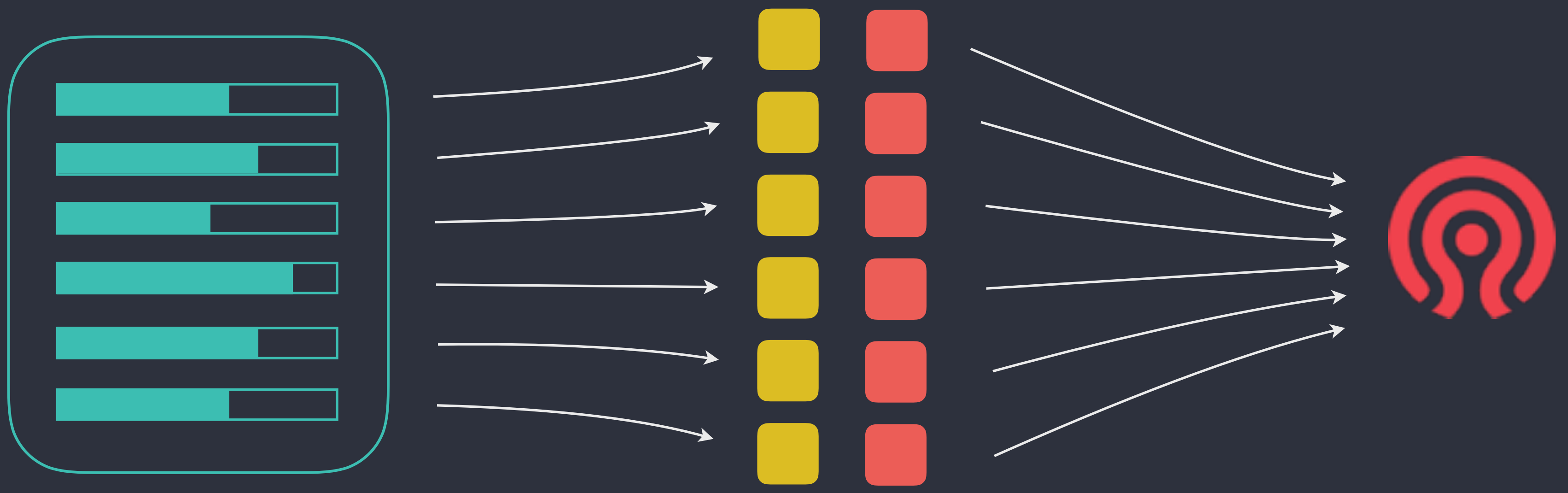


Writing the Data

The problem now is for the consumers to write the data downstream.

The consumers are actually **Storm Kafka spouts** : Kafka consumer running inside a Storm job. PunchPlatform storm topologies can be configured easily to have the required number of consumers, each reading at high speed one partition.

You need to couple these with (CEPH or HDFS or ...) **Storm File bolts**. These write the logs to files. How many you have depend on your need. Typically you have as many or more bolts than spouts.



Data Streaming Terminology

reader == consumer == spout

writer == producer == bolt

Easy ?

Doing what we just explained looks easy. It actually is touchy because:

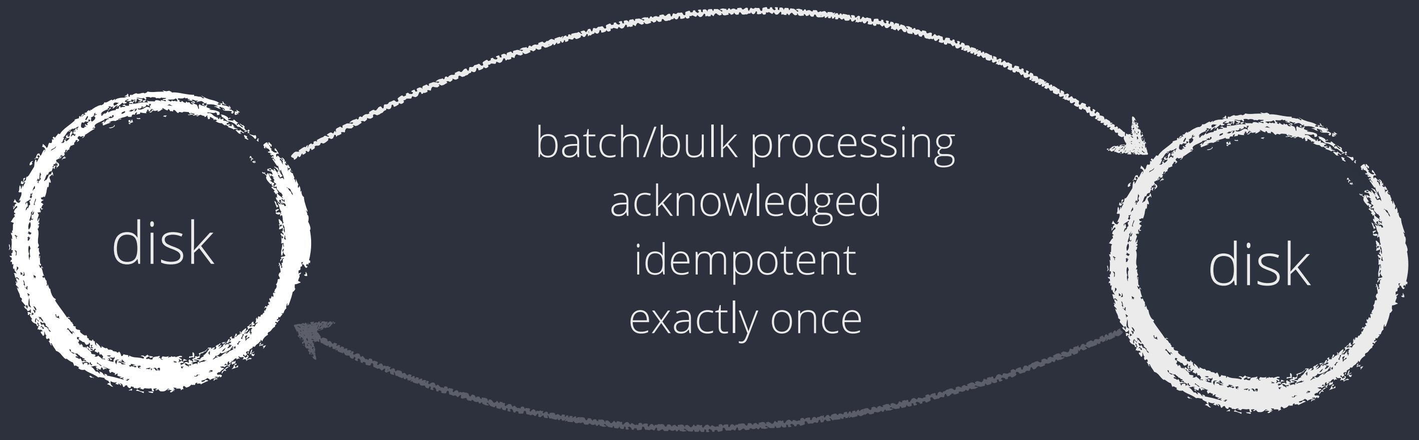
- to run this at high rate you need a bulk strategy. I.e. you must write files of several Mb instead of just individual logs of 1K.
- processing is *continuous*. You can have failures and/or restarts. You want these to have no effect on the files you end up writing. ***No loss of logs, no duplicates.***

The PunchPlatform provide ***repeatable and idempotent*** processing. It achieves eventual consistency towards *one or several* storage backend. All these are extremely important features. It allows users to have their data saved in several backend, and make sure they have *the same data* in all of them, despite failures.

These are hard issues. How are they solved ?

Strategy

In a nutshell : as follows



if failure then replay

This requires : partition identifiers, timestamping, unique identifiers, batch identifiers, smart kafka offset handling, idempotent bulk file writing, on the fly efficient zero-copy compression, on the fly ciphering ... and of course real time supervision

... in a way manageable by the user. That is what the PunchPlatform provides.

Thanks !