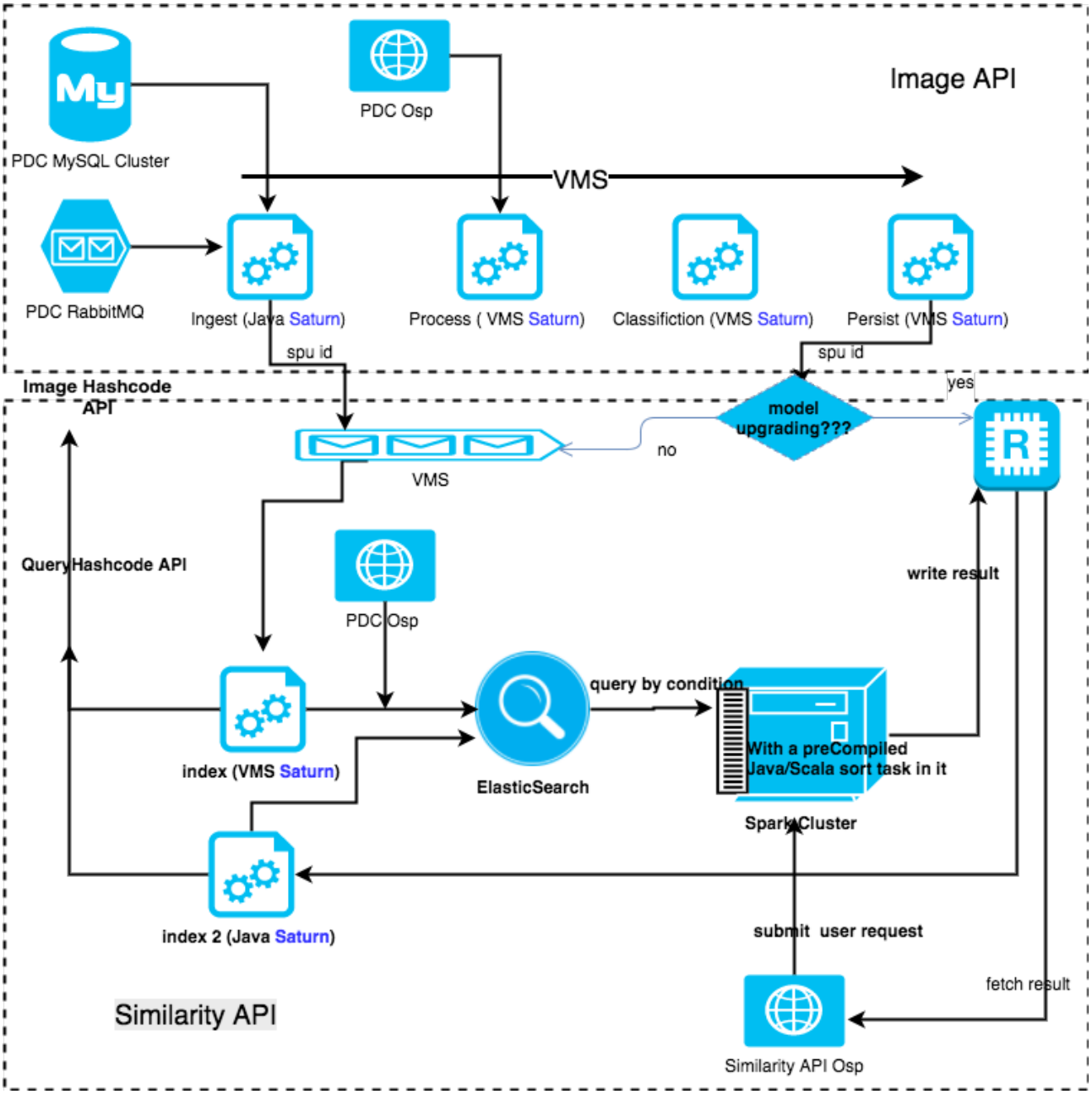

上图说明：
V2主要包括了一个VMS Saturn job（上图的index）、一个Java Saturn job（上图的index 2）、一个ElasticSearch cluster、一个Spark cluster以及沿用V1中的Osp和Restful服务

完整的一次相似度搜索流程如下

async：

1. 用户提交查询到osp（restful）服务 ==》osp将生成request Id返回给用户
2. osp将request Id同request一起提交给spark的scala sort job
3. job在spark中运行 最后将结果写入到redis中
4. 用户根据第一步的request id 提交fetch请求到osp osp检查redis中是否已经有结果

sync（开发完后根据一次搜索的耗时，是否需要待定？）

细节方面

index job

监听vms，此vms接收image api ingest和persist模块传来的消息，如果信息来自ingest，则获得spuid或者skuid，查询pdc osp接口，获取product信息，目前拟纳入索引的维度为商品ID、供应商ID、货号、最后版本号、商品标题、一句话卖点、seo标题、seo关键字、品牌ID、类目ID、销售价、吊牌价、标准货币类型、新旧程度、商品类型、商品状态、重量、长度、宽度、高度、创建时间、最后修改时间、产地、税率、品牌中文名称、品牌英文名称；如果信息来自persist，然后查询image api的queryHashcode接口获取hashcode，queryImageProperty接口获取imageapi的图片属性结果？，将上述属性提交到elasticsearch进行索引

（注：因为在imageapi模型更新期间，hashcode逻辑较为复杂 需要人工通过配置中心配置对外服务的model，这里使用queryHashcode接口，将逻辑留在imageapi中，另外如果imageapi persist判断目前处于模型更新期间，会将spuid skuid投入到redis set中）

同时还有一部分商品信息更新后 由于图片未更新所以 imageapi走到最后的模块会丢掉这部分spuid 所以要在ingest模块中构建商品信息索引，在persist模块中更新hashcode

index 2 job

主要处理模型更新期间，hashcode可能不稳定的那批spuid和skuid

首先判断是否处于模型更新下，如果是直接返回，如果不是，则拉取redis set中剩余元素 执行与index 1相同的操作

v1中的优化可以拿到v2中的

1. ~~map-reduce的threshold根据线上机器的配置实际调整，通常CPU频率越高，阈值可以适当增大~~
2. 设置hamming distance的threshold，实际上如果distance过大，相似度本身已经很低，在进行排序 差中选优 已经没有实际意义了 所以这里有必要设置一个阈值 可以很大程度上降低待排序数据的规模

similarity api的signature 增加一个定义
searchByQuery(Long baseId, String query, limit=l, threshold=t)
searchByQuery(List<Long> baseIds, String query, limit=l, threshold=t)
其中query是一个符合elasticsearch的query dsl的字符串

疑问待确定?

要处理sku和spu的关系，因为目前image api对于一个spu的hashcode记录是通过其款图识别得来的，假如两个spu的主款图颜色不同但是款式相近，那hashcode距离可能较远，通过这种方式比较就会有些问题

给一个image query+一个用户输入的“相似重点“（比如”颜色“”领型“）+一个search space（11-17万商品），对search space图片进行排序，并生成一个分数【先综合排序 再单通道排序】？

给一个batch of images, 建立一个kNN graph，以Json file返回；naive search complexity N^2; need to be optimized by some advanced hierarchical/indexing structures【这一类是无base查询，聚类？？？】