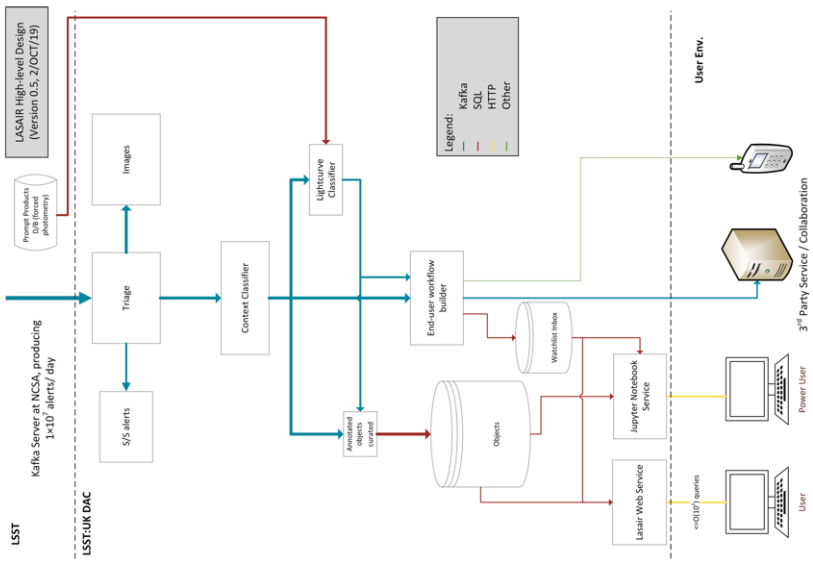
Parallel Ingestion and Workflow

Created by Roy Williams
Last updated 2 minutes ago • Analytics

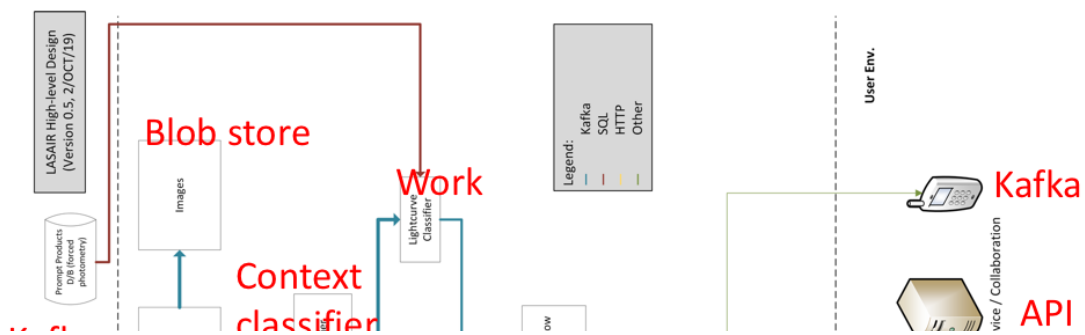
One of the crucial technologies for handling a high bandwidth stream will be parallelism: multiple nodes sharing the work. The following describes a prototype built on the STFC cloud that consists of N+2 nodes

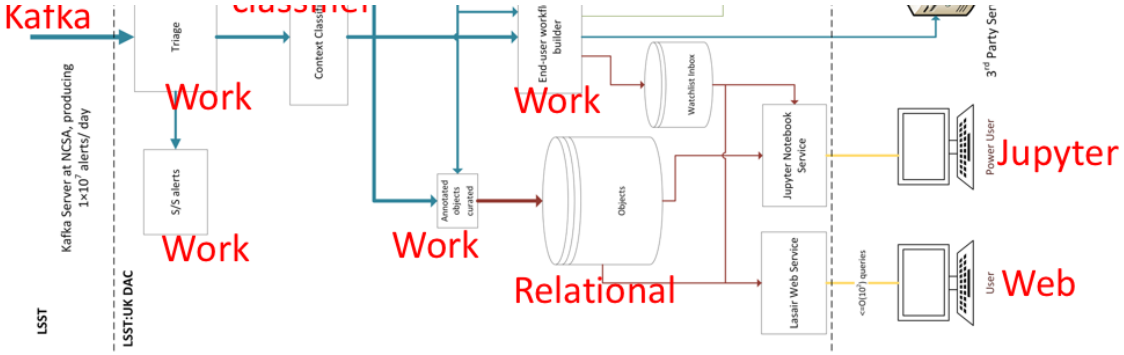
- N ingestion nodes that read a kafka stream of alerts using a shared groupID, meaning that each alert is ingested by exactly one of the nodes.
- One archive node that collects the data generated by the ingestion nodes
- One head node that orchestrates the rest of the cluster.

The code used is here, and there are ansible playbooks to provision the nodes.

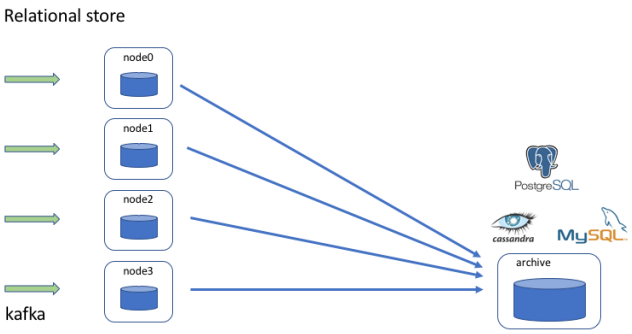


The thumbnail sketch of the Lasair-LSST architecture.

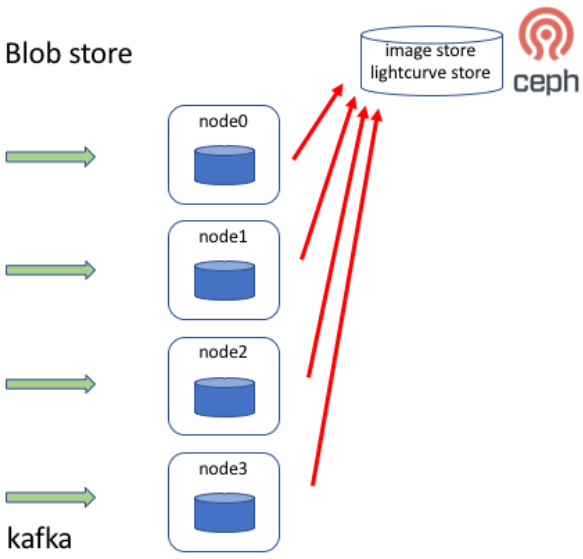




This is what it looks like as functional components. Everything labelled “work” can be handled by a farm of worker nodes, that use the services from nodes not labelled “work”.

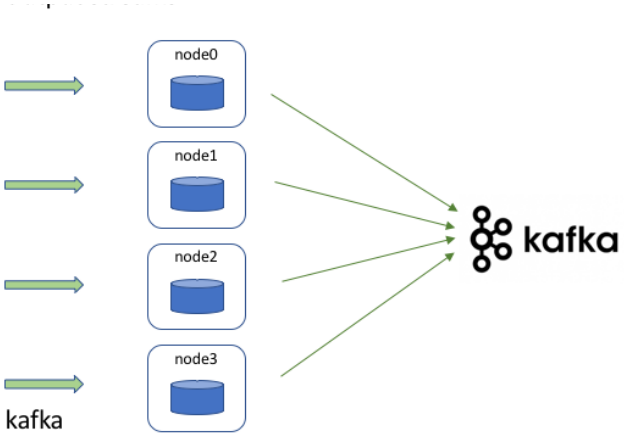


The ingestion nodes and archive node are shown here. The nodes share the kafka stream of alerts, coming in from the left, and they build a database record corresponding to each alert, which they write to a cache, a node-local database. Thus all the nodes work in parallel with no serial bottleneck. Every so often – 10000 alerts or nothing more to read – the contents of the local databases are transferred to the archive node.

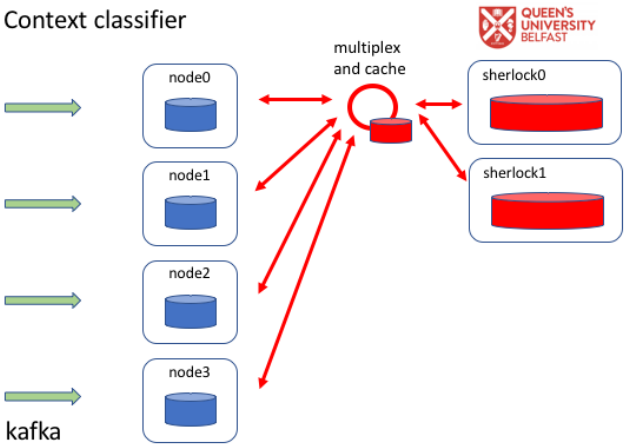


The next stage of the prototype is to treat binary data (images, perhaps also lightcurves) in the same way as the database records. The cache will be the local file storage, then when the data flow stops, transferring these to the blob store.

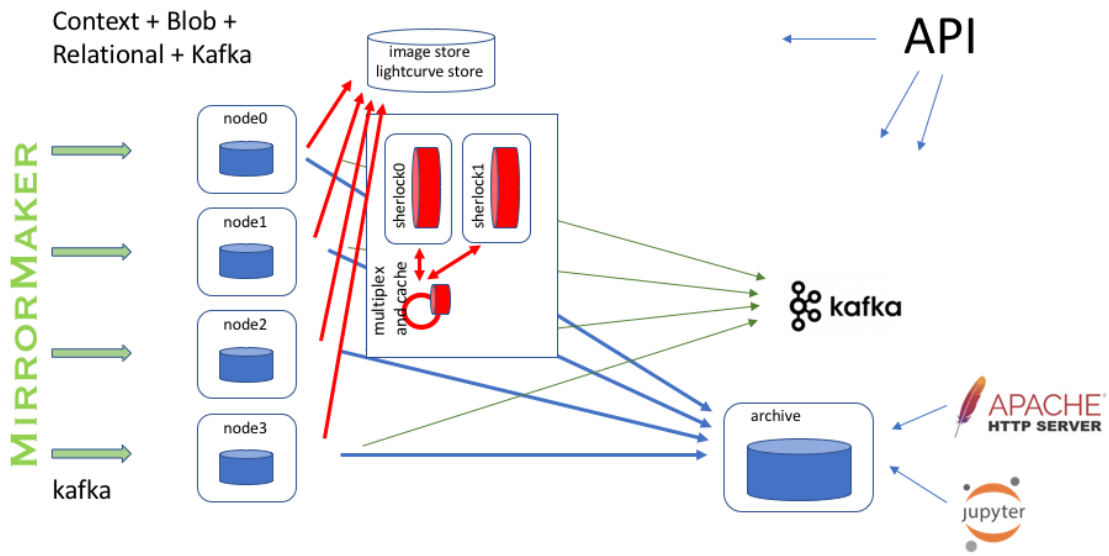
Output streams



A kafka producer can be added to each node. All the user-built queries will be available to each node, and all of them run against the alerts that the node has, and when the query is successful, the output matrixed together by a kafka producer node.



The context classifier (aka Sherlock) uses multi-TB databases to classify each alert by what is nearby in the sky. There will be a cache system, so that if the same requests is made again, then the saved information is sent immediately.



Here are all the moving parts of the system, an architecture that implements the required functionality. Services are set up and waiting, then each node starts consuming Kafka from the MIrrormaker cache. While 4 nodes are shown, it could be any number to achieve sufficient speed. Similarly with the sherlock cluster. Consumers can use the website to build complex filters, they can also use the Jupyter interface, consume a Kafka stream, or code against the Lasair API.