Design of Lasair Infrastructure

LSST:UK Phase B WP2.3 Alert Handling Infrastructure

Project Acronym	LUSC-B
Project Title	UK Involvement in the Legacy Survey of Space and Time
Document Number	LUSC-B-31

Submission date	22/AUG/22
Version	1.0
Status	Final
Author(s) inc. institutional affiliation	Roy Williams and Gareth Francis, University of Edinburgh Ken Smith and Dave Young, Queens University Belfast
Reviewer(s)	Cosimo Inserra (Cardiff), George Beckett (UEDIN)
Dissemination level	Public

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	10 July 2022	First draft from Williams	
0.2	16 July 2022	Additions from Francis, Smith	
0.3	9 Aug 2022	Minor corrections and clarifications	G. Francis
0.4	31 Oct 2022	Added science requirements section	R. Williams
1.0	18/NOV/22	Formatting updates following Exec Group approval, Final	T. M. Sloan

Table of Contents

1 Executive Summary

This report describes the technical infrastructure of the Lasair Community Broker: the purpose, the architecture and design, the underlying technologies, and how the components work together.

2 Architecture

2.1 What is Lasair?

The Lasair Community Broker[1] is a platform for astronomers to work effectively with the LSST transient alert stream: it is designed to be fast, flexible, and capable. While intended for the LSST alert flow, that is still in the future. Therefore Lasair development work so far has used a prototype system, ZTF, that has been running since 2018, and which has been built to be a prototype of LSST.

Lasair gets alert data from the USA most nights – whenever it is clear at the site of the telescope. The public ZTF stream that we now ingest is between 100,000 and 600,000 alerts per night, each about 60 Kbytes. The alerts are sent by Kafka technology (see below), and LSST will also use Kafka. Having set up the infrastructure to receive Kafka, we have decided to use Kafka for communication within our system as well as for the data input.

A Kafka system [7] allows a set of compute nodes[1] to read data packets so that ***each packet is read by at least one node***, with the Kafka system itself keeping the "bookmark[2]" of what has been read up to now.

While other LSST community brokers are based on existing classification algorithms, Lasair is not. Therefore we instead made the 'annotator' system, so that a user can run a classifier on Lasair lightcurves and push the result back into the system. We have one such classifier now in operation, called FastFinder.

2.2 Pipeline of Clusters

Lasair ingests data with a ***pipeline of clusters***: each cluster does a different job, some more compute/data intensive than others, so it is difficult to know a priori how much resource should be allocated to each. Our design gives flexibility: each cluster can be grown or reduced according to need. Also, there are various persistent data stores, again, each is driven by a resilient cluster that can be grown or reduced according to need. Figure 1 shows the concept: data enters the Kafka system on the left and progresses to the right. The green cluster reads, processes, and puts different data into the Kafka bus; as soon as that starts the yellow cluster pulls and pushes; eventually the whole pipeline is working. The clusters may also be reading and writing the data stores.

We also include the web and annotator nodes in this picture (bottom and right), as well as the mining nodes, although they are not part of the data ingestion pipeline. The web server nodes support users by delivering web pages and responding to API requests. The annotator nodes may be far from the Lasair computing centre and not controlled by

[1] In this document we use the term "node" as an implementation agnostic way of referring to an element of the Lasair system. In the current deployment this generally means a virtual machine, but in principle it could mean a container, a physical server or some other instantiation.

[2] In Kafka terminology, each *message* has an associated *offset*. When a *consumer* has fully processed a message it *commits* that offset to tell the Kafka broker that the message is processed and need not be served again to members of the same *consumer group*.

us, but they are in this picture because just like the others, they push data into the data storage and may read from Kafka. For more details, see section 2.3.7.

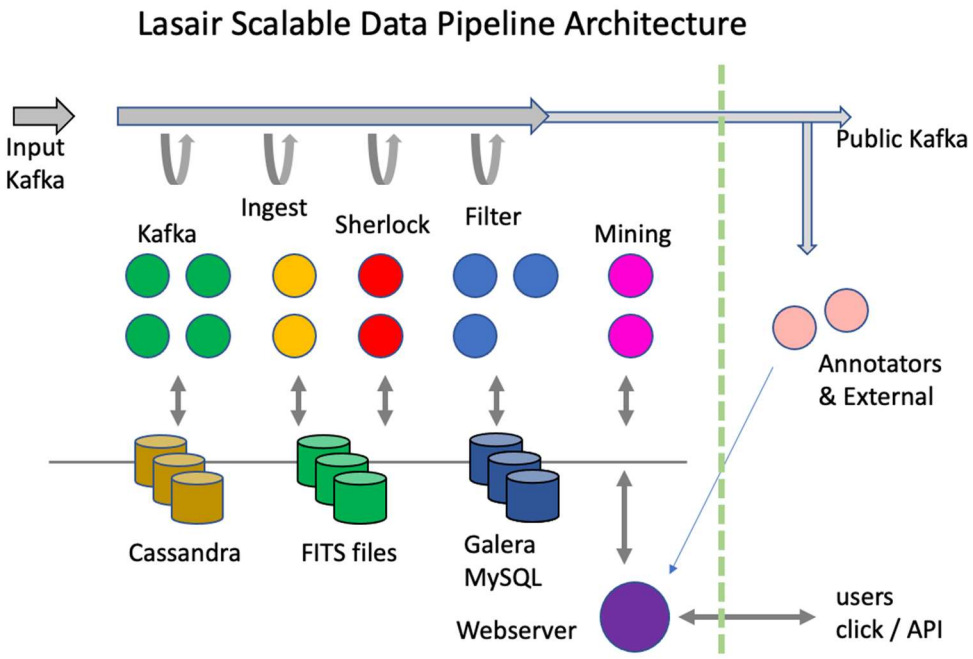

Figure 1. The Lasair processing pipeline. Each processing stage can have several compute nodes that are all executing a specified role in parallel. Nodes pull from and push to the Kafka system; nodes pull from and push to the persistent data stores.

2.3 Roles for nodes

2.3.1 Kafka

The Kafka system is represented by the green nodes in Figure 2 as well as the grey arrow at the top. It is responsible for reading and caching the alert packets from the USA, as well as sending it to the compute nodes and receiving their resulting packets.

The Kafka infrastructure is shown in Figure 2. There are actually two separate Kafka systems: a private system for the Lasair ingestion, designed for high throughput and large cache (dark grey arrows); the other a public system, that is for pushing streams of requested alerts to users.

2.3.2 Ingest

- Input: Avro-formatted alert packet
- Output:
    - Cutout images to shared filesystem, CephFS,
    - Lightcurve to NoSQL database,
    - JSON-formatted alert, with no cutouts, back into Kafka

An Ingest node reads the original alerts alerts from the Kafka system, and puts the cutout images in the shared filesystem, the recent lightcurve to NoSQL (Cassandra) database, then reformats the alert as JSON – since there is no binary content – then pushes that into the Kafka system. The ZTF alert includes 30 days of recent detections – LSST alerts will have a year of detections. The Cassandra lightcurve store, however, has full length lightcurves – ZTF has been running for four years now.

2.3.3 Sherlock

- Input: JSON formatted alert
- Output: The same alert with the addition of Sherlock data

Each Sherlock node has a SQL database of 5 Tbytes of astronomical sources from ~40 catalogues. The sky position of the input alert is used to intelligently decide on the most likely associated source from the catalogues, finding out, for example, if the alert is associated with a known galaxy, or if the alert is a flare from a known CV (cataclysmic variable). The speed of the Sherlock evaluation can be multiplied by adding further replicated nodes, each with its own 5 Tbyte database. We have also prototyped a cache system, so that when the same sky position is presented again, the result can be produced from cache.

2.3.4 Filter

- Input: Alerts with Sherlock addition
- Output: User-created streams on public Kafka, Records for the SQL database

Each filter node computes *features* of the 30-day light curve that comes with the alert (a year with LSST), as well as matching the alert against user-made watchlists and areas. Records are writen to a local SQL database onboard the node for the object and features, the Sherlock data, the watchlist and area tags. Other tables have already been copied into the local database from the main SQL database (see Background Services below). After a batch of perhaps 10,000 alerts are ingested to the local database, it can now execute the user-made queries and push out results via the public Kafka system – or via email if the user has chosen this option. The tables in the local database are then pushed to the main SQL database and replace any earlier information where and object is already known. Once a batch is finished, the local database tables are truncated and a new batch started.

2.3.5 Mining

- Input: Set of *objectIds*, code to run on light curves of those objects
- Output: Results of that code

These nodes can run massively-parallel computations on the light curves or cutout images. A controller farms out subsets of *objectIds* to worker nodes, which can fetch the whole light-curve of each object from the NoSQL Cassandra system. This system has been used to recompute the light-curve features that are in the relational database, and are used in SQL queries. While the technical infrastructure is in place to offer this service to users, we acknowledge that the mining system runs inside the compute cluster and has open access to the databases. Therefore giving users this capability still requires work to protect the Lasair systems from damage. So as of Oct 2022, the mining system is only available to Lasair staff. For more information see section 3.3.3.

2.3.6 Web

- Input: User clicks and API requests; SQL and NoSQL databases
- Output: Saved queries, watchlists, areas

The Lasair webserver and API server allow users to control their interactions with the alert database and the stream. They can create a watchlist of their interesting sources, and Lasair will report crossmatches with members of the watchlist. They can define regions of the sky, and Lasair will report when alerts fall inside a region. They can define and save SQL queries that can run in real time as filters on the alert stream.

2.3.7 Annotator

The Lasair API supports *annotation*: a structured external packet of extra information about a given object,that is stored in the annotations table in the SQL database. This could be, the result of running a machine-learning algorithm on the lightcurve, the classification created by another broker, or data from a follow-up observation on the object, for example a link to a spectrum. Users that put annotations into the Lasair database are vetted, and administrators then make it possible. That user will run a method in the Lasair API that pushes the annotation: all this can be automated, meaning the annotation may arrive within minutes of the observation that triggers it.

2.3.8 Openstack

Openstack [9] is an open-source platform to controls pools of compute, storage, and networking resources, all managed through APIs or a dashboard. Additional components provide orchestration and fault management. Lasair is deployed on an Openstack cloud, allowing easy instantiation of compute nodes that are connected to virtual storage.

2.3.9 Lasair-Deploy

The Lasair team has a deployment system, enabling easy instantiation of a fully-capable system as in Figure 1, installing all the necessary software and configuration. The Lasair model of a pipeline of clusters relies on Ansible [10] to provision each new node in a given role. The Lasair-Deploy system orchestrates deployment of an entire cluster, in a way that is quick, repeatable, reproducible, flexible, and self-documenting. Deployments can adapt to dev/testing/production as required.

It uses OpenStack Heat, a service to orchestrate composite cloud applications using a declarative template format through an OpenStack-native REST API. It creates disparate resources in one go, e.g. instances, volumes, floating IPs – known as a stack -- and defines relationships between them.

3 Infrastructure Components

3.1 Data Transport

In production deployments of Lasair, the Kafka cluster is deployed using at least three instances and a replication factor of two (see Figure 2), thus it can continue to function with the loss of any one such instance and recover when the instance is restored.

Kafka clients can be configured to operate with either at-least-once or at-most-once delivery semantics. Since duplicate alerts are generally preferable to missing ones we have tried to use the former. Lasair components are therefore designed to be, as far as possible, idempotent; that is, the ultimate system state should be independent of how many times and in what order the alert messages are processed.

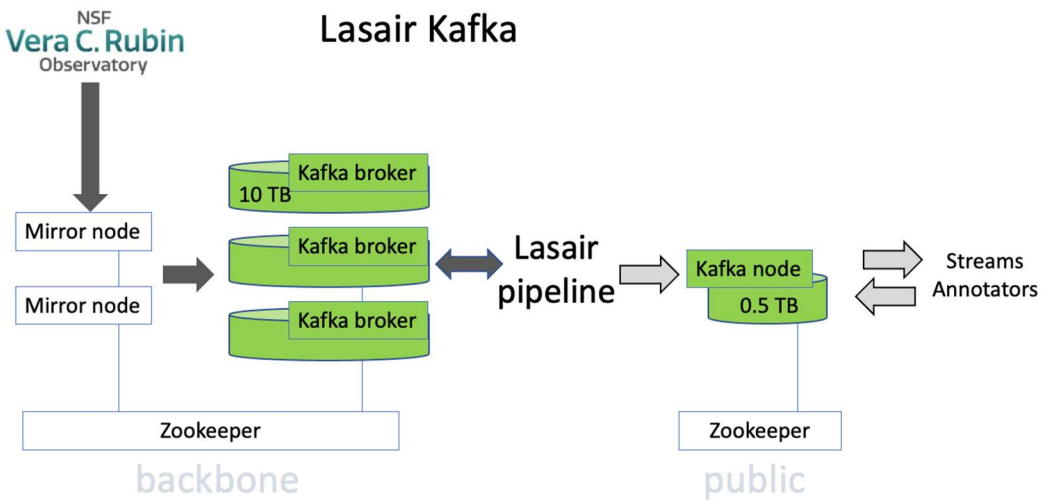

Figure 2. The Lasair Kafka system.

3.2 Data storage

While each node of the Openstack system has its own storage volume, there are three shared data systems – see the bottom of Figure 1. Given that users interact with Lasair through SQL queries, there is an SQL database to execute these; there are also local versions of this database on the filter nodes that are used to execute queries on each batch of alerts as they are received. Early prototypes of Lasair used the SQL database to also store the detections of objects, but we found that the database became very large very quickly. Therefore the lightcurves (sets of detections) are kept in a separate NoSQL database keyed by *objectId*. In this architecture, users query on object features (eg variance of magnitude) but not on individual detections. Once suitable objects are selected, their lightcurves can be retrieved from the NoSQL database. The third shared data system is a shared filesystem – CephFS – that all notes can mount.

3.2.1 SQL database

The first Lasair prototype used the MySQL/MariaDB database that runs on a single node, but we have now upgraded to Galera[11], a multi-master cluster based on synchronous replication. Galera provided both scalability and resilience over the single-node original[3].

The SQL database drives the webserver and API through Django, and also executes user queries on the object table, joined with other tables. Of course great care must be taken if a user can create SQL and have it run on the central database: we must be careful of 'SQL injection attacks'. Care has been taken that users cannot damage the database or the operation of the Lasair system through malicious or badly-made SQL clauses. Lasair does not accept whole SQL statements from users, but rather clauses that are sewn together. In doing this, there are some precautions and modifications applied to the SELECT and WHERE clauses that the user provides:

- User-written queries run on a read-only account on the database
- An execution time limit is enforced
- A limit on output rows is enforced

[3] Lasair retains the option of deploying using a single MariaDB instance where replication is not required, e.g. for development deployments.

- Some words are prohibited: some examples are `create`, `select`, `from`, `where`, `join`, `inner`, `outer`, `with`, `union`, `exists`.
- All queries are run first with 'LIMIT 0' appended as a syntax check
- There is a careful parsing of parentheses and brackets.

3.2.2 NoSQL database

For the lightcurve store, Lasair uses Apache Cassandra [12], an open source NoSQL distributed data store that is resilient, scalable, and elastic. Data replication is built-in and in the current deployment there are three copies of all data in the five node cluster. The vast majority of queries to Cassandra are to request detections pertinent to specified objects (lightcurves) but the data can also be reorganised to query by sky position if necessary. (This is tested but not yet fully integrated.) Data is organised by default by object, timestamp and detection id. Addition of load & storage capacity is managed by simply adding more nodes. The Cassandra Query Language (CQL) has strong similarities to SQL, and makes query implementation (bearing in mind the limitations imposed by the distributed by-object data indexing) for both reading and writing very straightforward.

3.2.3 CephFS

The Ceph File System, or CephFS, is a POSIX-compliant file system built on top of Ceph's distributed object store, RADOS. CephFS endeavours to provide a state-of-the-art, multi-use, highly available, and performant file store for a variety of applications, including traditional use-cases like shared home directories, HPC scratch space, and distributed workflow shared storage. Lasair uses CephFS to store the cutout images that come with the alerts, and also as a way for compute nodes to share information.

3.3 Other infrastructure

The major components of Lasair have been described above: processing pipeline, databases, and user-facing aspects. This section covers the remaining components.

3.3.1 Monitoring

We use Prometheus[13] and Grafana[14] to provide a display of the Lasair system. Each major component has a red/green panel to show if it working or not. The Kafka flows in the pipeline are shown, as alerts come in from the USA, then their progress through the *ingest* cluster, through the *Sherlock* cluster, and through the *filter* cluster.

There is also summary page on the Lasair website, with numbers of alerts that have passed through the stages, how many are of different categories (eg. Solar System), how much data sent to the NoSQL and to the SQL databases, as well as a record of the background services.

The Lasair software has a way to call for human attention through a dedicated Slack workspace. Throughout the code, many exceptions will cause a Slack message, for example unable to connect to another node, or insufficient memory, or no disk space.

3.3.2 Background services

There are a number of "cron" jobs that run regularly, some every few hours, some at midnight UTC before an observing night. The Transient Name Server (TNS) is updated by pulling all ~80,000 reported transients and making a local copy that Lasair can use; this is copied to all the filter nodes before an observing run because user filters may use this table. Similarly, the annotations table is copied to all the filter nodes before each observing run so they can service queries with this local copy. The emphasis here is to achieve scalability by making each filter node as independent as possible, without continual calls to the central SQL database.

Lasair has collaborated with other LSST community brokers, that are also using ZTF as a prototype stream. The Fink and Alerce projects produce Kafka streams of their classifications, which are pushed into the Lasair database as they appear, through an hourly cron process.

3.3.3 Massive computing

A new Lasair system is in prototype, to allow large data-mining or other jobs to run on multiple nodes of the Openstack cloud. The pattern is illustrated in Figure 3, where a head node runs a query on the SQL database to find objects, and this list is distributed among many worker nodes. Each worker then fetches the lightcurve from the NoSQL database, and computes, the result being a classification, a discrimination, or other text; these outputs are then collected by the head node.

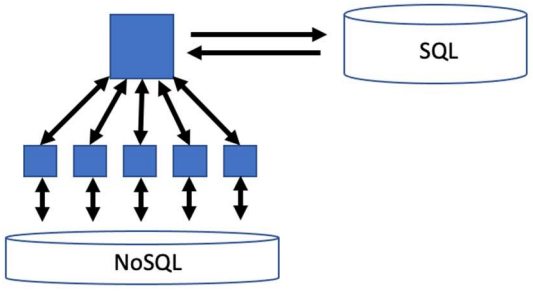

Figure 3. The design pattern for massive computing.

We have implemented this pattern using SSH, where a head node distributes tasks to worker nodes.

4 Risk and Sustainability

Lasair backs up its SQL and noSQL databases on a regular schedule, on a physically-distinct machine, although it is a machine in the same rack. The most likely failure scenario is a disk failure in the Openstack cloud, and that system is already redundant and can recover and continue until that disk is replaced. Lasair is built to the same safety standards as the rest of the LSST:UK Data Access Centre.

Lasair is well documented to minimise disruption case of the loss of key staff. There is a Principles of Operation on the LSST:UK wiki, which is updated with relevant information every time procedures are done, whenever a staff member says to themselves “I didn’t know that”. All code is open-source and kept in public off-site Github repositories[15], with README files to explain how components work. Code changes are seen by the whole team through code reviews, associated with pull requests. There is a comprehensive testing system as part of Lasair – unit tests, system tests, and integration tests. Creating a new deployment is automated with the ‘lasair-deploy’ system.

5 Science Requirements

This report needs to report on how the infrastructure as built answers the science requirements [16] as repeated here:

5.1 Lasair shall provide

- R2.01 A searchable database containing all the LSST alerts : with time latency to match the detailed science requirements specified in the document "LSST Transients and Variables Science requirements; Lawrence et al." -- ***YES***
- R2.02 Light-curves: assimilate all *diaSource* alerts in *diaObjects*: providing interactive webpages (linked to database), plots, ability to select ranges, submit user added points. – ***YES, currently for ZTF, with LSST in development***
- R2.03 Postage stamps: all LSST detections and most recent non-detections. Plus multi-colour images from LSST, near infra-red (VISTA/UKIDSS), H-alpha (VPHAS) and EUCLID, or HST/JWST if space based imaging is available. Size of postage stamps should be selectable (number of different, fixed sizes). –

YES if this survey is available as a HiPS survey [17] that can be used by AladinLite

- R2.04 Massive catalogue cross-match: with star, galaxy, AGN, x-ray, radio catalogues, galaxy cluster catalogues, strong lens catalogues, and provide classification through boosted decision trees through our already working code "*Sherlock*" (Young et al. 2018). ***-- YES***
- R2.05 Cross match to all *previously* known transients: supernovae, transients, gamma ray-bursts, x-ray and radio burst sources (e.g. searching for currently unknown physical links over time) – ***YES if the transient is registered with the IAU Transient Naming Service [18]***
- R2.06 A database query platform and user-owned storage : for users to query the database and return their own objects and selections in various useful formats. This should be both a SQL query form and Jupyter platform. Users should have access to their own storage where they can store lists of their own objects (more details in the detailed user requirements R2.13) – ***YES via watchlists, watchmaps, and stored queries***
- R2.07 In real-time, cross-match to all other wavelength time-domain surveys: gamma-ray, x-ray and radio (e.g. MEERKat/Thunderkat through 4pisky.org , Swift, SVOM, eRosita) – ***NO, not *all* time-domain surveys, but YES for the Transient Naming Service [18]***
- R2.08 Spectroscopic and/or photometric redshift : locate catalogued redshifts of the likely host galaxy and hence absolute mag (we will link to WPs 3.2 and 3.4 for redshifts) ***– YES through Sherlock crossmatch***
- R2.09 Combine all of the above information: including the first 24hr-48hr lightcurve trend (e.g. rapid rise/decline) to probabilistically classify all transients as : supernova – kilonova – GRB – Tidal Disruption Event – AGN – XRB – CV – eruption star – microlens – orphan – ***NO but we have the annotation mechanism so that astronomers can contribute their own classifiers. Also we ingest classifications from other LSST brokers.***
- R2.10 Multi-messenger cross-matching: *GW coincidence tag* based on their 4 dimensional position in space and time compared to LIGO-Virgo gravitational wave events (sky position, distance, and time). All transients will also be *Neutrino coincidence tagged* based on their 3D space time location (sky position and time) with IceCube high energy neutrinos. ***– YES we have GW coincidence infrastructure***
- R2.11 Provide a stream of transients to 4MOST and SOXS spectroscopic programmes and ingest the classifications and data from those facilities in return (linked to WP 3.3) ***– YES***
- R2.12 Provide users with a means to upload a "Watchlist" : up to 10^6 objects and provide means to triggering on magnitude variations, and allow an adjustable search radius. ***-- YES***
- R2.13 Collect a detailed list of user requests and implement them : for additions, enhancements, suggestions, alterations, we will engage with the UK community to maintain a wish list of enhancements and additions and work with the Transient and Variable star PoCs to prioritise this list ***-- YES***

5.2 Lasair should provide

- R2.13 Previous history from Pan-STARRS, DES, Skymapper, ATLAS, CRTS, PTF/ZTF – ***Not all of these***
- R2.14 Test, and if successful, Implement machine learning ***– YES can be implemented through annotations***
- R2.15 Machine learning algorithms for real-bogus classification: as a final check on real-bogus objects, we will run our own trained ML code to weed out spurious objects (Wright et al. 2016, 2017, Smartt et al. 2016). ***-- NO***

6 References

[1] Lasair Community Broker: https://lasair-ztf.lsst.ac.uk

[2] Zwicky Transient Facility: https://ztf.caltech.edu/

[3] LSST alert stream: https://www.lsst.org/scientists/alert-brokers

[4] Transient Naming System: https://www.wis-tns.org/

[5] Multi-Order Coverage Maps:

https://www.ivoa.net/documents/MOC/ and https://pypi.org/project/mocpy/

[6] Sherlock system: https://arxiv.org/abs/2003.09052

[7] Apache Kafka: https://kafka.apache.org/

[8] Django web framework: https://www.djangoproject.com/

[9] OpenStack cloud software: https://www.openstack.org/

[10] Ansible Automation Platform: https://www.ansible.com/

[11] Galera Cluster for MySQL: https://galeracluster.com/

[12] Apache Cassandra: https://cassandra.apache.org

[13] Prometheus: https://prometheus.io/

[14] Grafana: https://grafana.com/

[15] https://github.com/lsst-uk/lasair4 and https://github.com/lsst-uk/lasair-deploy

[16] https://lsst-uk.atlassian.net/wiki/spaces/LUSCSWG/pages/614465537/LSST+UK+Science+Requirements+Document, section R2

[17] Hips: Hierarchical Progressive Surveys, http://aladin.u-strasbg.fr/hips/

[18] IAU Transient Naming Service https://www.wis-tns.org/