ICAT-PACER

A high-throughput, asynchronous data ingestion engine for ICAT

Rodrigo Cabezas Quirós
Marc Armenter Hierro

Table of contents

- Context and motivation.
- Project architecture.
- Current status.
- Next steps.

Context

- Data catalogue.
  - Working with ICAT since 2023.
  - Using the ESRF's dataset ingestion system (Apache's ActiveMQ and Camel framework).
- Data synchronization between systems.
  - Several inefficient user and investigation sync processed to keep data up to date with ICAT and VISA.
  - Ongoing migration of User Office portal.
- Error tracing.
  - ICAT's operation is divided across different teams at ALBA; difficult to provide single easy error tracing point.
  - Keeping full historical failure log records is not possible.
  - No active error alert mechanisms.

Motivation

- Improve our ingestion system.
  - Offload scheduled data synchronization processes into our ingester.
  - Add better message and error tracing through the whole data cycle process.

- Contribute to the ICAT collaboration.
  - Build the PACER.

Python

Asynchronous

Catalogue

Entry

Router

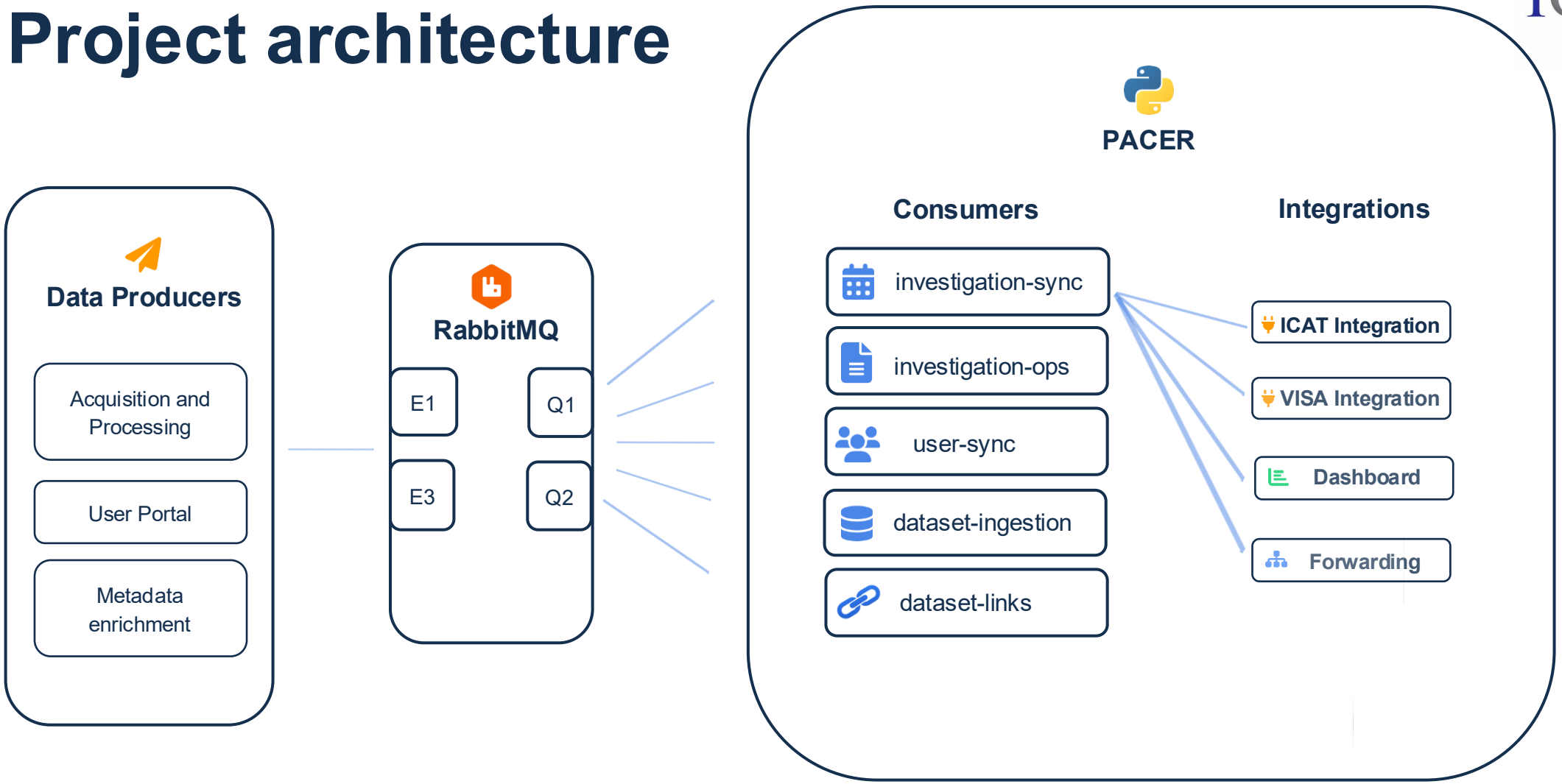
Project architecture

- Message broker components.
  - Exchanges: Entry point – decides how messages are routed. There are Direct, Topic, Fanout, Headers types.
  - Routing key: Label attached to message that exchanges use to determine where the message should go.
  - Bindings: Rules that connect Exchanges to Queues. They tell queues if they are interested in the message.
  - Queue: Stores messages until consumed.

Message example

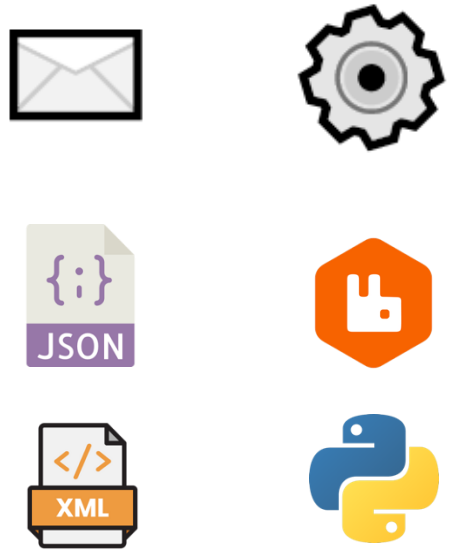
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dataset
  xmlns="http://www.cells.es/icat">
  <investigation>20250330096</investigation>
  <instrument>bl06</instrument>
  <name>ref-cq_3_diffraction_thumbnail_run1</name>
  <location>Sample-cq/ref-cq_3_diffraction_thumbnail_run1/all_files</location>
  <startDate>2025-07-15_15:53:33</startDate>
  <endDate>2025-07-15_15:53:35</endDate>
  <parameter>
    <name>scanType</name>
    <value>diffraction_thumbnail</value>
  </parameter>
  <parameter>
    <name>input_datasets</name>
    <value>Sample-cq/ref-cq_3</value>
  </parameter>
  <parameter>
    <name>ResourcesGalleryFilePaths</name>
    <value>
    /icat/gallery/0_diffraction_thumbnail.png,
    /icat/gallery/1_diffraction_thumbnail.png,
    /icat/gallery/2_diffraction_thumbnail.png,
    /icat/gallery/3_diffraction_thumbnail.png
    </value>
  </parameter>
  <sample>
    <name>20250703_Sample-cq</name>
    <type>SAMPLE</type>
  </sample>
  <datafile>
    <location>0_diffraction_thumbnail.png</location>
  </datafile>
  <datafile>
    <location>1_diffraction_thumbnail.png</location>
  </datafile>
  <datafile>
    <location>2_diffraction_thumbnail.png</location>
  </datafile>
  <datafile>
    <location>3_diffraction_thumbnail.png</location>
  </datafile>
</dataset>
```

Project architecture



Project architecture

- Project stack and features
  - RabbitMQ as message broker.
  - Built entirely on Python.
  - Kombu – message handling framework.
  - Support for JSON and XML messages.
  - Integration with Elastic Search.
  - External system for tracing all messages.
  - Configuration validation.



Project architecture

- PACER configuration:
  - Configured through a YAML file.
  - Integrations for ICAT, VISA, DataCite, PaNOSC and dashboard – can be individually enabled / disabled per consumer.
  - Consumers are independent from each other and can be scaled up separately as well.
- Open by design:
  - Easy to integrate any other feature, integration or consumer module just through the configuration.

```yaml
exchanges:
  - name: "dashboard-logging-exchange"
    type: "direct"
  - name: "uos-sync-exchange"
    type: "direct"

queues:
  - name: "dashboard-logging"
    exchange: "dashboard-logging-exchange"
    routingKey: "message.logging"

consumers:
  - className: "UsersConsumer"
    module: "consumers.users"
    enabled: true
    queues:
      - "uos-sync-users"
    workers: 1
    integrations:
      - "dashboard"
      - "messageForwarding"
      - "icat"
      - "visa"

integrations:
  icat:
    enabled: true
    server:
      url: "<url>"
      authPlugin: "db"
      username: "<user>"
      password: "<psswrd>"
```

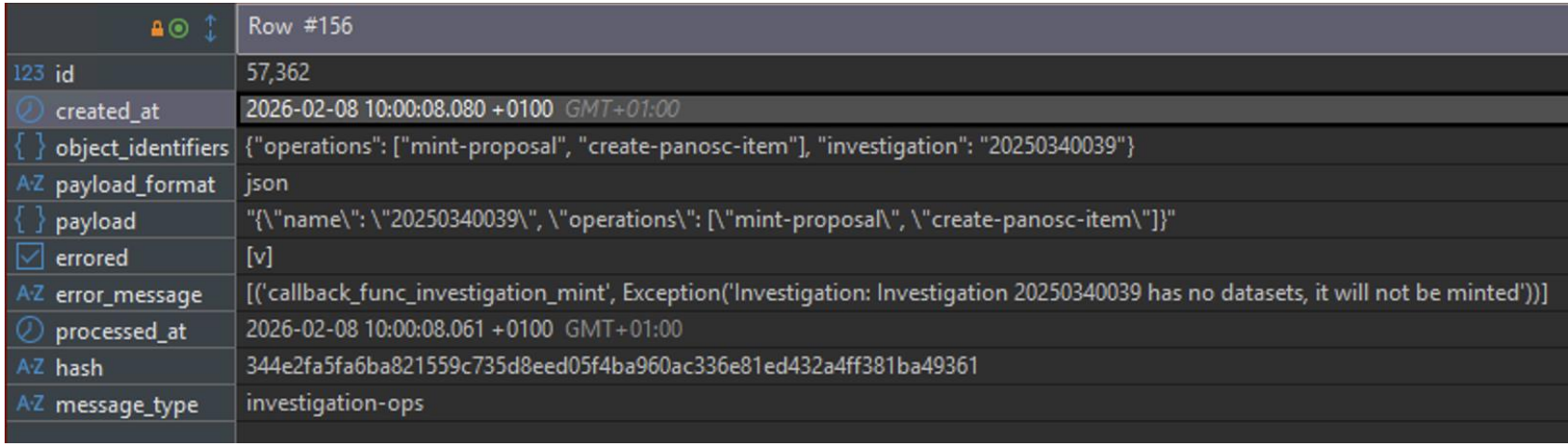
Project architecture

- Data synchronization integrations.
  - A single message is sent, and it is propagated to many ends (using tasks).
  - User Office portal pushes user and investigation updates to ICAT (and VISA).
  - Message forwarding is implemented by the PACER – e.g.: investigation updates are done in prod, then pushed to test environments.

- Investigation operations and checks.
  - Investigations are minted at the end of the experiment (30 day max window).
  - Some checks are run on each investigation: DOI, PaNOSC Search API item creation.
  - Planned other future checks to be implemented (e.g.: data deletion from disk).

Project architecture

- Dashboard logging.
  - Additional (also optional) integration for the PACER.
  - Logs every message into a partitioned Postgres database.
  - Enables better message for traceability, processing metrics and easy messge reingestion.
  - Whenever an error occurs, it is also logged with the message.
  - There's also plans for building a user-friendly UI for scientists and other ICAT-involved teams.



	Row #156
123 id	57,362
created_at	2026-02-08 10:00:08.080 +0100 GMT+01:00
object_identifiers	{"operations": ["mint-proposal", "create-panosc-item"], "investigation": "20250340039"}
payload_format	json
payload	"{\"name\": \"20250340039\", \"operations\": [\"mint-proposal\", \"create-panosc-item\"]}"
errored	[v]
error_message	[('callback_func_investigation_mint', Exception('Investigation: Investigation 20250340039 has no datasets, it will not be minted'))]
processed_at	2026-02-08 10:00:08.061 +0100 GMT+01:00
hash	344e2fa5fa6ba821559c735d8eed05f4ba960ac336e81ed432a4ff381ba49361
message_type	investigation-ops

Project architecture

- python-icat library extended:
  - RollbackContext: Using a context manager we track modifications made to tracked entities, allowing rollbacks to initial or intermediate states.
  - Lazy loading: Enabled on-the-fly access for object's non-included foreign keys and many-to-many relations.
  - Django-like querying: Overridden default search functions for enabling querying with new operators – no need to build complex queries (e.g.: '__in', '__eq', '__gte', '__lte', …").

```
input_dataset_locations = input_dataset_param.stringValue.split(",")

raw_datasets = icat_client.search( entity: "Dataset",
                                   conditions=
                                   {"location__in": input_dataset_locations},
                                   flatten_single=False)
if not raw_datasets:
    self.logger.warning(f"No raw datasets found for dataset {dataset_id}")
    return
```

Current status

- Currently in prod.
  - User and investigation synchronization from new and old User Office.
  - Investigation operations (DOI, PaNOSC).
- In testing.
  - Dataset ingestion, raw-processed dataset linkage, gallery upload and statistics.
  - Aiming at 100% coverage testing.
- Pending.
  - Transtition to one investigation per experiment instead of a single ICAT investigation.

Next steps

- PACER start-up and publication.
  - Aiming at getting all features running in production by beginning of Q2.
  - Collaborate with dataset ingestion features and other projects.

Any questions?

cells.es

Cerdanyola del Vallès (Barcelona)

Spain

Tel: (+34) 93 592 43 00