Test Cases:

1.) **Valid Subscription Test** → If a user has an active subscription for an event, they should receive a notification.

2.) **System-Wide Notification** → If an event is marked as system-wide, all users should be notified.

3.) **Role-Based Notification** → If a role is subscribed, all users in that role should get the notification.

4. **Non-Existent Dataset** → If we trigger an event for a dataset that doesn't exist, the system should log an error instead of sending notifications.

5. **Invalid Event Type** → If an unknown event type is emitted, the system should reject it.

6. **No Subscribed Users** → If no users are subscribed to an event, the system should not attempt to send notifications.

7. **Expired Subscription** → If a user's subscription has expired, they should not receive notifications.

8. **Invalid JSON in RabbitMQ** → If a malformed message is placed in the queue, it should be discarded safely.

9. **Email Sending Failure** → If the SMTP server is down, the system should retry and log an error.

10. **RabbitMQ Server Failure** → If RabbitMQ is unavailable, the system should attempt to reconnect.

11. **High Volume Test** → We simulate **10,000 notification events** and check if all are processed successfully.

12. **Parallel Processing** → We run multiple instances of the processor script to ensure that events are **distributed evenly** without duplication.

Creating Issue in Github:

Common setup for test cases:

These steps must be completed before executing any of the test cases in this document:

1. Run Core Services:

RabbitMQ Server

- Ensure the RabbitMQ server is running and accessible at the expected host/port.
- The following queues should be declared:
  - `events`
  - `email_notifications`

2. Required Services Must Be Running

- **`notification_processor.js`**
  - This service must be running to consume from the `events` queue and push to notification queues.
- **`emailProcessor.py`**
  - This service should be running to consume from the `email_notifications` queue and send emails via SMTP.

3. Database Setup

- Ensure the following tables are populated as needed:
    - `user`: At least one valid user with an email address.
    - `event_type`: Must contain the tested event types (e.g., `DATASET_STAGED`).
    - `notification_subscription`: Correctly configured based on the test case (user/role/system-wide).
    - `notification_preference`: Users should have `email` delivery method enabled for tests requiring email.

4.)Event Triggering:

Use the `test.py` script to emit events.

```
from workers.services.events import eventBus

eventBus.emit('EVENT_NAME', resource_id='...', resource_type='...')
```

Replace the placeholders based on the specific test case scenario.

5.)Monitoring Tools:

RabbitMQ UI (`http://localhost:15672`)

Use this to observe queue states, messages, and message flow.

- **Logs**
    - Monitor logs from `notification_processor.js` and `emailProcessor.py` in real time.
    - Look for key log messages such as:
        - `Found X users subscribed to event: ...`
        - `Created X notification msgs...`
        - `Email sent successfully to ...`
        - `Error sending email: ...`

6. **Clean Test Environment:**

- Empty the queues (`events`, `email_notifications`) before starting each test.
- Clear or reset any temporary test data inserted during previous test runs.

1.) **Valid Subscription Test** – Verify that a user with an active subscription receives a notification when an event occurs.

Description:

This test case verifies that when a user has an active subscription to a specific event (e.g., `DATASET_STAGED`), the system correctly generates and sends a notification through the user's enabled delivery channels (e.g., email, app) upon triggering that event.

Preconditions:

- Common environment setup from the Shared Setup & Requirements section is complete.
- The user is registered and exists in the `user` table.
- The `event_type` table contains an entry for `DATASET_STAGED` (`event_type_id = 1`).
- The `notification_subscription` table contains an **active, non-expired, user-specific subscription** to `DATASET_STAGED` for a valid dataset.

Test Data:

User Table:

- `user_id`: `6`
- `email`: `kadugg@iu.edu`

Event Type Table:

- `event_type_id`: 1
- `name`: `DATASET_STAGED`

Notification Subscription Table:

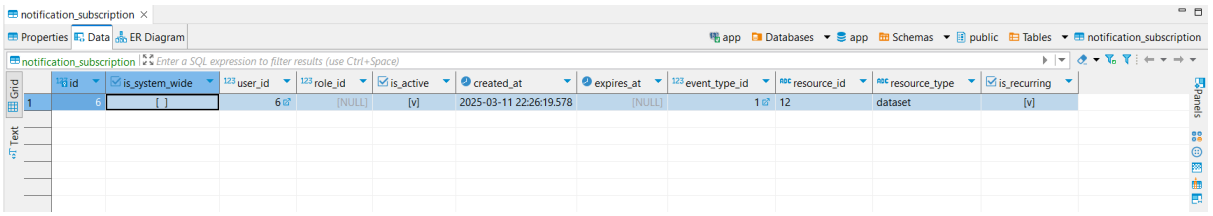
user_id	event_type_id	resource_id	resource_type	is_active	is_recurring
6	1	12	dataset	true	true

Trigger Event (`test.py`):

```
from workers.services.events import eventBus

eventBus.emit('DATASET_STAGED', resource_id='12',
resource_type='dataset')
```

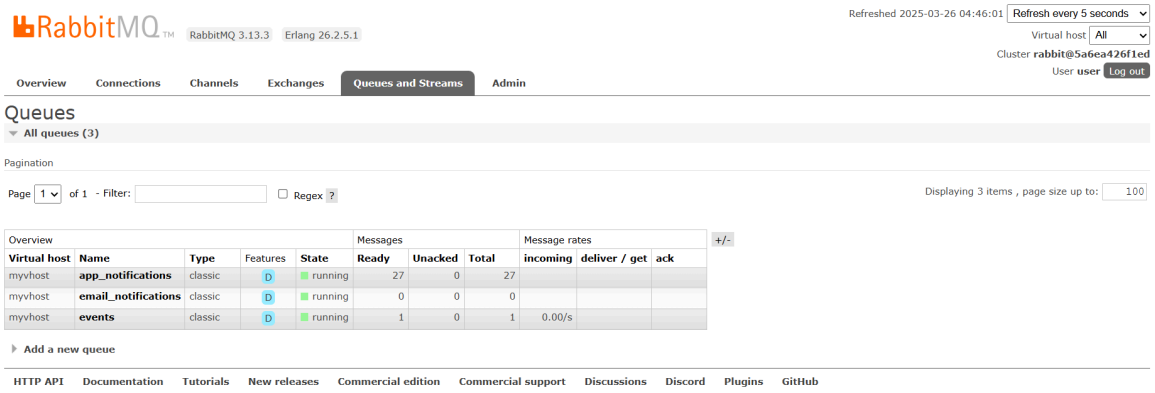
Test Steps:

1. Insert the user, event type, and subscription entries if they do not already exist.



2. Verify the user has an email delivery method enabled in notification_preference.

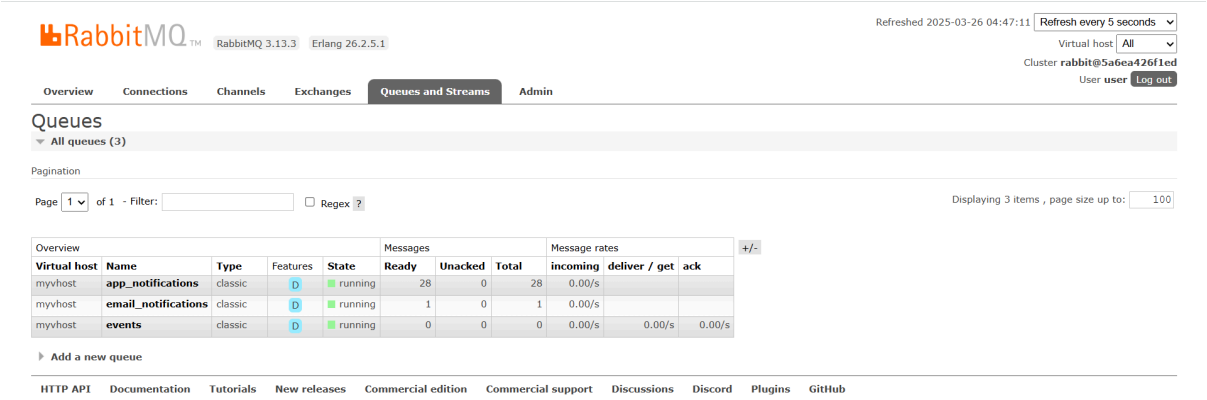
3. Run the test.py script to emit the DATASET_STAGED event.



4. Observe logs from notification_processor.js to confirm subscription match and notification creation.

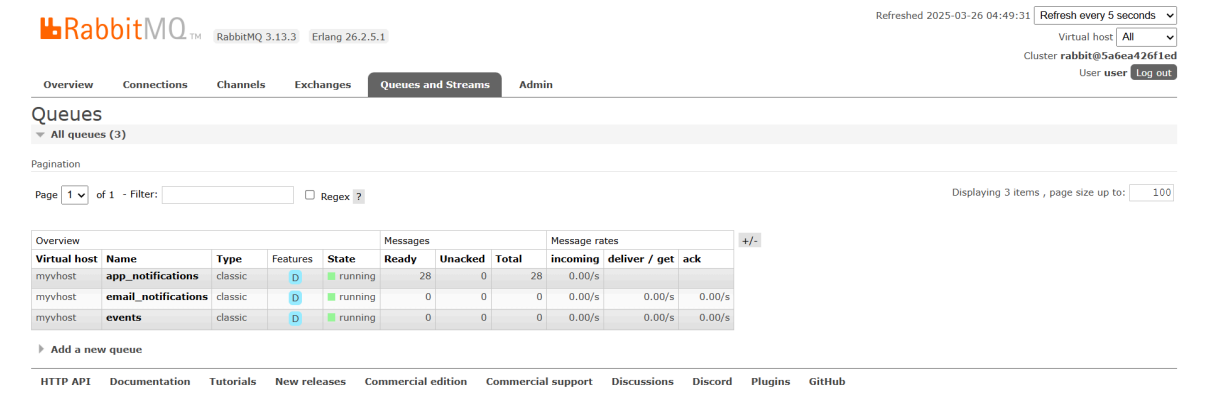
```
Karthiek Duggirala@LAPTOP-OF4FJBMF MINGW64 ~/Documents/sca/bioloop/api (notifications)
$ node src/scripts/notification_processor.js
WARNING: NODE_ENV value of 'default' is ambiguous.
WARNING: See https://github.com/node-config/node-config/wiki/Strict-Mode
2025-03-26 04:46:59 info: [*] Waiting for messages in events. To exit, press CTRL+C
2025-03-26 04:47:01 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 04:47:03 info: Created 2 notification msgs for event: DATASET_STAGED
2025-03-26 04:47:03 info: Sent 2/2 messages to notification queues
```

5. Confirm that a message is added to the email_notifications queue.



6. Monitor logs from emailProcessor.py to verify email delivery.

```
Karthiek Duggirala@LAPTOP-OF4FJBMF MINGW64 ~/Documents/sca/bioloop/workers (notifications)
$ python -m workers.scripts.emailProcessor
loading  conf
Waiting for messages in queue: email_notifications...
📩 Received email request: 📊 New dataset staged: PCM230412_new -> kadugg@iu.edu
```



Expected Result:

notification_processor.js logs:

Found 1 users subscribed to event: DATASET_STAGED

Created 2 notification msgs for event: DATASET_STAGED

```
Sent 2/2 messages to notification queues
```

emailProcessor.py logs:

```
Received email request: New dataset staged ->
kadugg@iu.edu

Email sent successfully to kadugg@iu.edu
```

- The user receives the notification via email (and app if applicable).
- No errors are thrown during processing.

2.) **System-Wide Notification** – Verify that all users receive a notification when a system-wide event occurs

Description:

This test case verifies that when an event (e.g., `DATASET_STAGED`) is marked as **system-wide**, the notification system sends notifications to **all active users** who have at least one enabled delivery method (such as email), regardless of whether they've explicitly subscribed to that event.

Preconditions:

- The `event_type` table contains the `DATASET_STAGED` event (`event_type_id = 1`).
- At least 3 active users exist in the `user` table with:
  - Valid email addresses
  - Enabled notification preferences for the email channel
- A **system-wide subscription** is defined in the `notification_subscription` table for `DATASET_STAGED`.

Test Data:

Notification Subscription Table Entry (System-Wide):

user_id	event_type_id	resource_id	resource_type	is_active	is_recurring
NULL	NULL	true	1	true	true

User Table (Example Users):

user_id	email
6	kadugg@iu.edu
7	user-7@iu.edu
8	user-8@iu.edu

Notification Preference Table (per user):

user_id	delivery_method	enabled
6	email	true
7	email	true

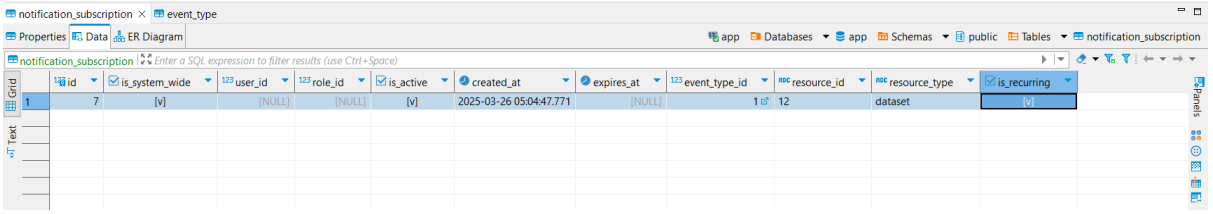
8	email	true

Trigger Event (`test.py`):

```
from workers.services.events import eventBus

eventBus.emit('DATASET_STAGED', resource_id='12',
resource_type='dataset')
```

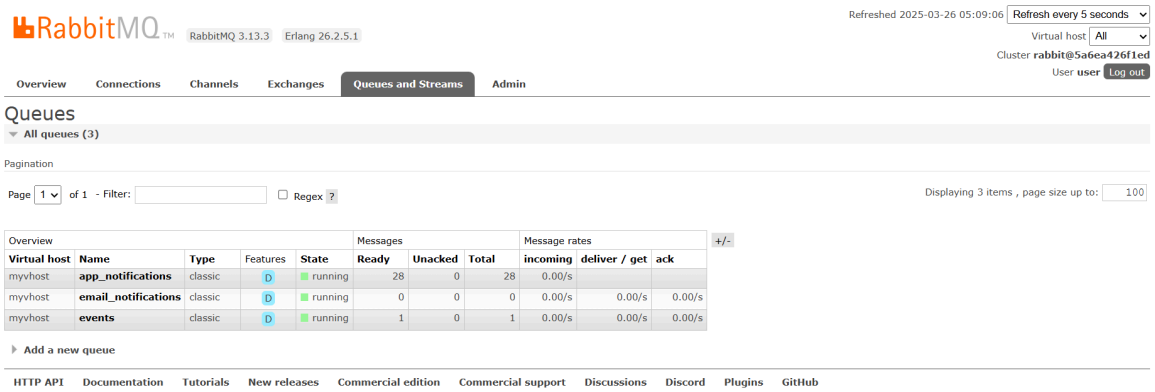
Test Steps

1. Insert or verify that the `event_type` and system-wide subscription entries exist for `DATASET_STAGED`.



2. Confirm that at least 3 users have email preferences enabled and are not marked as deleted.

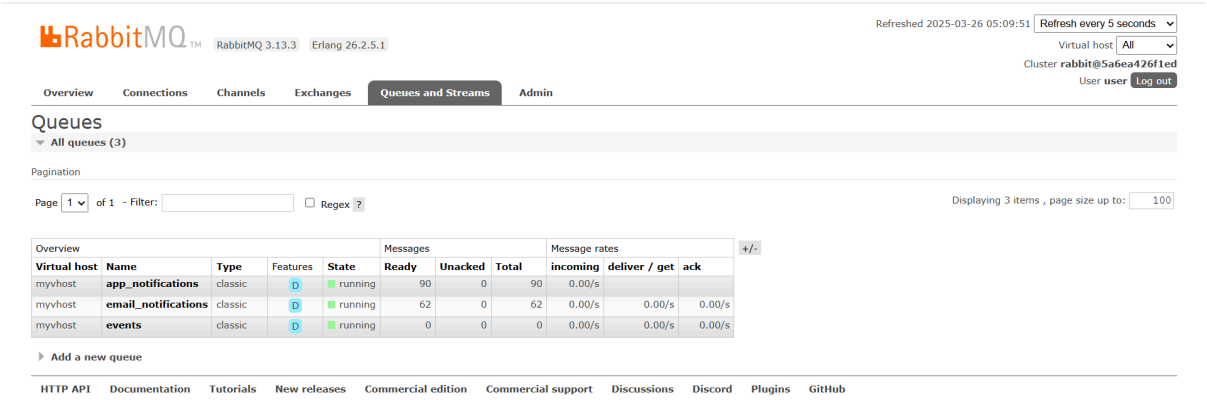
3. Run the `test.py` script to emit the system-wide event.



4. Monitor `notification_processor.js` logs to confirm that notifications were generated for all users.

```
Karthiek Duggirala@LAPTOP-OF4FJBMF MINGW64 ~/Documents/sca/bioloop/api (notifications)
$ node src/scripts/notification_processor.js
WARNING: NODE_ENV value of 'default' is ambiguous.
WARNING: See https://github.com/node-config/node-config/wiki/Strict-Mode
2025-03-26 05:09:34 info: [*] Waiting for messages in events. To exit, press CTRL+C
2025-03-26 05:09:37 info: Found 62 users subscribed to event: DATASET_STAGED
2025-03-26 05:09:39 info: Created 124 notification msgs for event: DATASET_STAGED
2025-03-26 05:09:39 info: Sent 124/124 messages to notification queues
```

5. Verify that all messages appear in the `email_notifications` queue.



6. Observe `emailProcessor.py` logs to confirm successful delivery of each email.

Expected Result

notification_processor.js logs:

```
Found 62 users subscribed to event: DATASET_STAGED

Created 124 notification msgs for event: DATASET_STAGED
```

```
Sent 124/124 messages to notification queues
```

(Assuming each user receives both email and app notifications)

emailProcessor.py logs:

```
📩 Received email request: New dataset staged -> kadugg@iu.edu

📩 Received email request: New dataset staged -> user-7@iu.edu

📩 Received email request: New dataset staged -> user-8@iu.edu
```

- All eligible users receive email notifications.
- No duplicate or missed messages are observed.

3.) Role-Based Notification – Verify that all users associated with a subscribed role receive a notification when an event occurs

Description:

This test case verifies that when a **role-based subscription** is configured for an event (e.g., `DATASET_STAGED`), all users assigned to that role receive notifications via their enabled delivery channels (e.g., email). This validates correct handling of **role-level notification logic**.

Preconditions:

- The `event_type` table includes `DATASET_STAGED` with `event_type_id = 1`.
- The `role` table contains a role ("Operators") with `role_id = 2`.
- Two or more users are associated with this role via the `user_role` table.
- Each user has at least one enabled delivery method (e.g., `email`) in the `notification_preference` table.
- A **role-based subscription** to `DATASET_STAGED` is present in the `notification_subscription` table.

Test Data:

Role Table Entry:

Field	Value
role_id	2
name	Operators

User Table Entries:

user_id	role_id	email

7	2	user7@iu.edu
8	2	user8@iu.edu

Notification Subscription Table Entry (Role-Based):

user_id	role_id	event_type_id	resource_id	resource_type	is_active	is_recurring
NULL	2	false	1	12	true	true

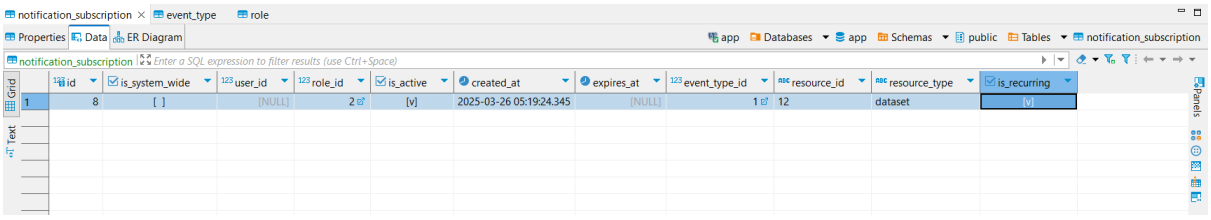
Trigger Event (`test.py`):

```
from workers.services.events import eventBus

eventBus.emit('DATASET_STAGED', resource_id='12',
resource_type='dataset')
```

Test Steps

1. Ensure the "Operators" role exists (`role_id = 2`) in the `role` table.

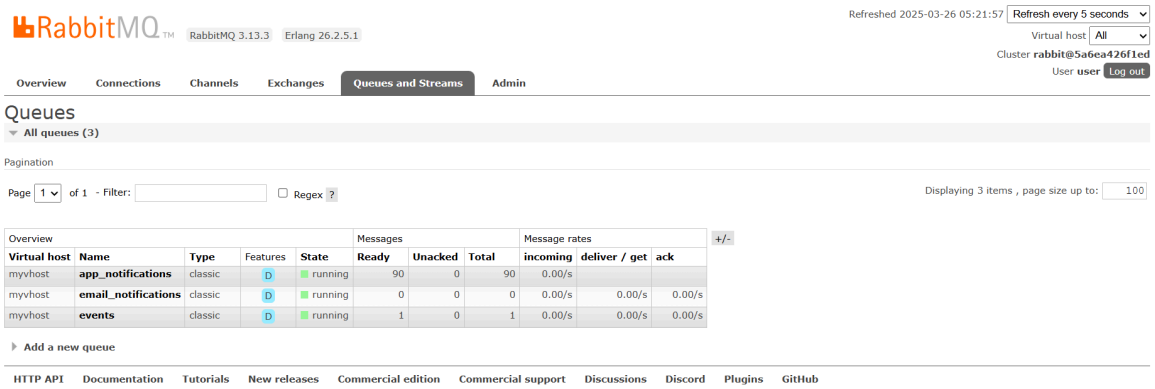
2. Confirm that `user_id = 7` and `user_id = 8` are linked to this role in the `user_role` table.



3. Insert a role-based subscription to `DATASET_STAGED` with `resource_id = 12` in the `notification_subscription` table.

4. Confirm both users have `email` delivery preferences enabled in `notification_preference`.

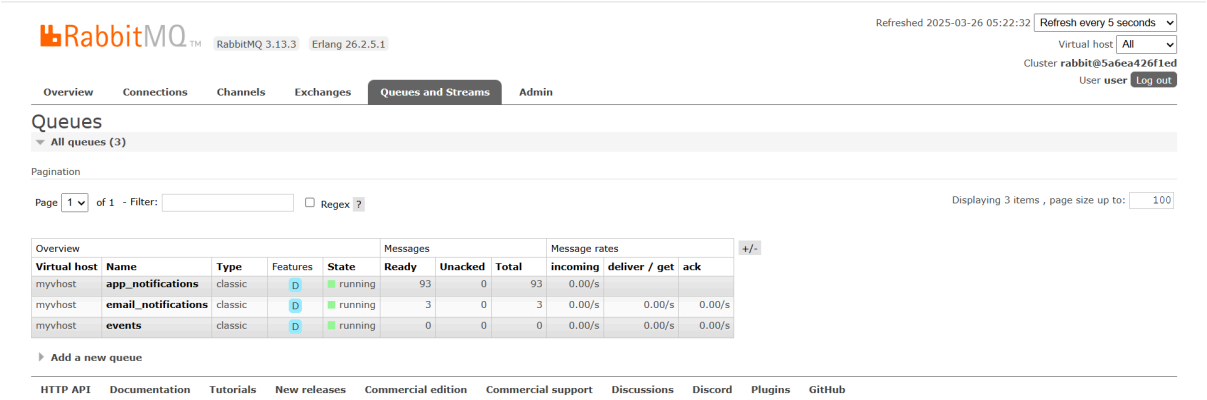
5. Trigger the event using the `test.py` script.



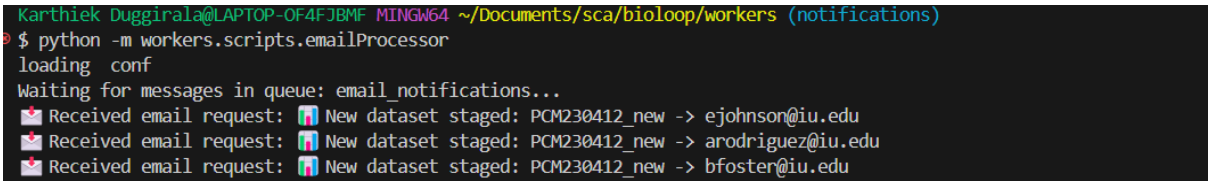
6. Monitor `notification_processor.js` logs to verify that both users are detected and notifications are generated.

```
Karthiek Duggirala@LAPTOP-OF4FJBMF MINGW64 ~/Documents/sca/bioloop/api (notifications)
$ node src/scripts/notification_processor.js
WARNING: NODE_ENV value of 'default' is ambiguous.
WARNING: See https://github.com/node-config/node-config/wiki/Strict-Mode
2025-03-26 05:22:21 info: [*] Waiting for messages in events. To exit, press CTRL+C
2025-03-26 05:22:23 info: Found 3 users subscribed to event: DATASET_STAGED
2025-03-26 05:22:25 info: Created 6 notification msgs for event: DATASET_STAGED
2025-03-26 05:22:25 info: Sent 6/6 messages to notification queues
```

7. Check that two messages appear in the `email_notifications` queue.



8. Confirm that `emailProcessor.py` sends emails to both users.



Expected Result

notification_processor.js logs:

`Found 3 users subscribed to event: DATASET_STAGED`

`Created 3 notification msgs for event: DATASET_STAGED`

`Sent 6/6 messages to notification queues`

RabbitMQ Queues:

- `email_notifications` queue contains **2 messages**, one for each user.

emailProcessor.py logs:

```
📩 Received email request: New dataset staged -> user7@iu.edu

📩 Received email request: New dataset staged -> user8@iu.edu

📩 Received email request: New dataset staged -> user9@iu.edu
```

- Both users receive the expected email notification for the staged dataset event.

4.) Non-Existent Dataset – Verify that triggering an event with an invalid dataset ID results in a logged error and no notification is sent

Description:

This test case ensures that when an event (e.g., `DATASET_STAGED`) is triggered for a **non-existent dataset**, the notification system does **not send any notifications** and instead logs an appropriate error. This prevents users from receiving messages related to invalid or deleted resources.

Preconditions:

- `DATASET_STAGED` exists in the `event_type` table with `event_type_id = 1`.
- The dataset with `resource_id = 99999` does **not exist** in the system.

- A user-based subscription is created for `resource_id = 99999`, even though the resource does not exist.
- The subscribed user has an enabled notification preference for email.

Test Data:

Notification Subscription Table Entry:

role_id	is_system_wide	event_type_id	resource_id	resource_type	is_active	is_recurring
NULL	false	1	99999	dataset	true	true

Note: There must be **no dataset** with ID `99999` in the database.

Trigger Event (`test.py`):

```
from workers.services.events import eventBus

eventBus.emit('DATASET_STAGED', resource_id='99999',
resource_type='dataset')
```

Test Steps:

1. Verify that dataset ID `99999` is not present in the database.
2. Insert a user-based subscription targeting `resource_id = 99999` into `notification_subscription`.

notification_subscription | event_type | role | dataset

Properties | Data | ER Diagram

app | Databases | app | Schemas | public | Tables | notification_subscription

notification_subscription | Enter a SQL expression to filter results (use Ctrl+Space)

	id	is_system_wide	user_id	role_id	is_active	created_at	expires_at	event_type_id	resource_id	resource_type	is_recurring
1	9	[ ]	6	[NULL]	[v]	2025-03-26 05:32:23.146	[NULL]	1	99999	dataset	[v]

3. Ensure the user (`user_id = 6`) has email enabled in `notification_preference`.

4. Emit the event using the `test.py` script.

RabbitMQ™ RabbitMQ 3.13.3 Erlang 26.2.5.1

Refreshed 2025-03-26 05:33:22 | Refresh every 5 seconds
Virtual host All
Cluster rabbit@5a6ea426f1ed
User user | Log out

Overview | Connections | Channels | Exchanges | Queues and Streams | Admin

Queues

All queues (3)

Pagination

Page 1 of 1 - Filter: Regex ?

Displaying 3 items , page size up to: 100

Overview					Messages			Message rates		
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
myvhost	app_notifications	classic	D	running	93	0	93	0.00/s		
myvhost	email_notifications	classic	D	running	0	0	0	0.00/s	0.00/s	0.00/s
myvhost	events	classic	D	running	1	0	1	0.00/s	0.00/s	0.00/s

Add a new queue

HTTP API | Documentation | Tutorials | New releases | Commercial edition | Commercial support | Discussions | Discord | Plugins | GitHub

5. Monitor `notification_processor.js` logs for errors related to missing dataset.

```
Karthiek Duggirala@LAPTOP-OF4FJBMF MINGW64 ~/Documents/sca/bioloop/api (notifications)
$ node src/scripts/notification_processor.js
WARNING: NODE_ENV value of 'default' is ambiguous.
WARNING: See https://github.com/node-config/node-config/wiki/Strict-Mode
2025-03-26 05:35:08 info: [*] Waiting for messages in events. To exit, press CTRL+C
2025-03-26 05:35:10 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 05:35:12 error: Error processing event No dataset found
C:\Users\Karthiek Duggirala\Documents\sca\bioloop\api\src\scripts\notification_processor.js:161
          const retryCount = msg.properties.headers['x-retry-count'] || 0;
                                                   ^

TypeError: Cannot read properties of undefined (reading 'x-retry-count')
    at readChannel.consume.noAck (C:\Users\Karthiek Duggirala\Documents\sca\bioloop\api\src\scripts\notification_processor.js:161
:52)

Node.js v19.9.0
(bioloop)
```

6. Verify that **no messages** are added to `email_notifications` or other queues.

7. Confirm that `emailProcessor.py` shows **no processing activity**.

Expected Result:

notification_processor.js logs:

```
Error processing event No dataset found
```

RabbitMQ Queues:

- `email_notifications` and `app_notifications` queues remain empty.

emailProcessor.py logs:

- No logs or delivery activity (since no notification is published).

5.) Invalid Event Type – Verify that emitting an undefined event type results in a logged warning and no notification is sent

Description:

This test case ensures that if an event with an **unrecognized or undefined name** (i.e., not present in the `event_type` table) is emitted, the system handles it gracefully by logging a warning and not attempting to generate or send any notifications.

Preconditions:

- The event name `UNKNOWN_EVENT_TYPE` does **not exist** in the `event_type` table.
- The `notification_subscription` table may contain entries for other events — they should not affect this test.

Test Data:

Event Type Table:

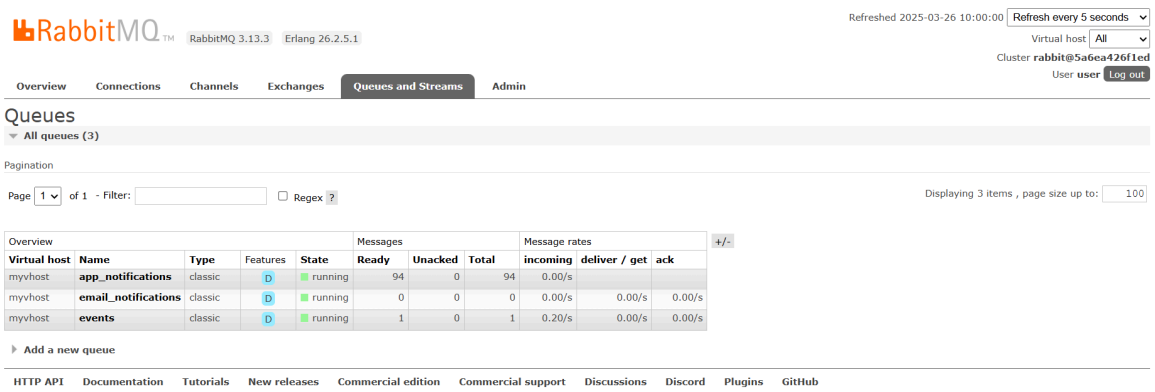
- Make sure, there is no entry for `UNKNOWN_EVENT_TYPE`.

Trigger Event (`test.py`):

```
from workers.services.events import eventBus

eventBus.emit('UNKNOWN_EVENT_TYPE', resource_id='12',
resource_type='dataset')
```

Test Steps:

1. Verify that the event_type table does **not contain** `UNKNOWN_EVENT_TYPE`.
2. Start `notification_processor.js`.
3. Run the `test.py` script to emit the unknown event.



4. Observe the logs from `notification_processor.js` to verify that:
    - The event is received.
    - A warning is logged for the missing template or event type.
    - No notification messages are generated.

```
Karthiek Duggirala@LAPTOP-OF4FJBMF MINGW64 ~/Documents/sca/bioloop/api (notifications)
$ node src/scripts/notification_processor.js
WARNING: NODE_ENV value of 'default' is ambiguous.
WARNING: See https://github.com/node-config/node-config/wiki/Strict-Mode
2025-03-26 10:00:39 info: [*] Waiting for messages in events. To exit, press CTRL+C
2025-03-26 10:00:42 info: Found 0 users subscribed to event: UNKNOWN_EVENT_TYPE
2025-03-26 10:00:42 info: Created 0 notification msgs for event: UNKNOWN_EVENT_TYPE
2025-03-26 10:00:42 info: Sent 0/0 messages to notification queues
```

5. Confirm that:
    - The `email_notifications` and `app_notifications` queues remain **empty**.
    - `emailProcessor.py` logs no activity.

Expected Result

notification_processor.js logs:

```
No template class found for the event:
UNKNOWN_EVENT_TYPE
```

or

```
Event type 'UNKNOWN_EVENT_TYPE' not found in the
database.
```

RabbitMQ Queues:

- `email_notifications` and `app_notifications` remain **unchanged**.

emailProcessor.py:

- Shows **no logs or processing** related to this event.

This test validates the system's ability to gracefully reject unknown events without affecting other operations or crashing the pipeline. **(Not as expected)**

6.) No Subscribed Users – Verify that when no users are subscribed to an event, no notifications are generated or sent

Description:

This test case ensures that when an event (e.g., `DATASET_STAGED`) is triggered, but there are **no user**, **role**, or **system-wide** subscriptions associated with it, the system silently acknowledges the event and does **not attempt to generate or send notifications**. It validates proper handling of unassigned event triggers.

Preconditions:

- All common setup steps from the Shared Setup & Requirements section are complete.
- `DATASET_STAGED` exists in the `event_type` table (`event_type_id = 1`).
- There are **no active notification subscriptions** for:
  - `event_type_id = 1`
  - Any combination of `resource_id` / `resource_type`
- Test users may exist in the system, but **none should be subscribed** to the event.

Test Data:

Trigger Event (`test.py`):

```
from workers.services.events import eventBus

eventBus.emit('DATASET_STAGED', resource_id='15',
resource_type='dataset')
```

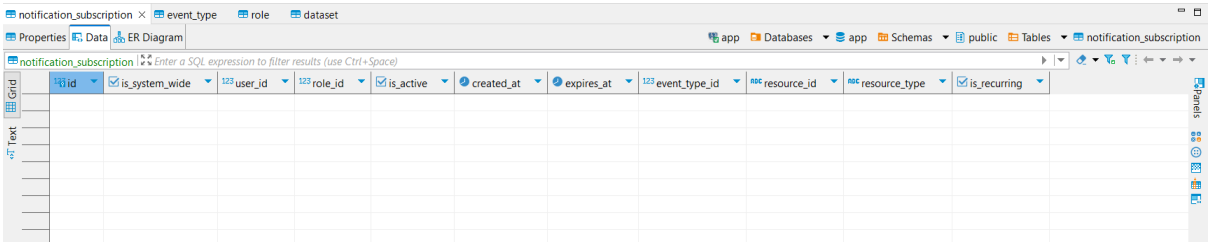
Note:
Ensure the `notification_subscription` table does **not** contain any entry with:

- `event_type_id = 1`

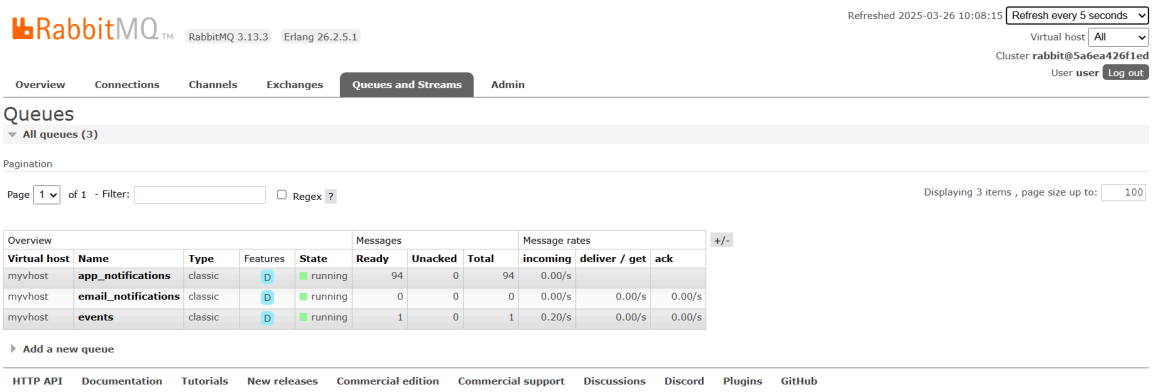
- `is_active = true`
- `resource_id = 15` or `NULL`
- `resource_type = dataset` or `NULL`

Test Steps

1. Verify that `DATASET_STAGED` exists in the `event_type` table.
2. Confirm that the `notification_subscription` table has **no active entries** for this event and dataset.



3. Start `notification_processor.js`.
4. Emit the event using the `test.py` script.

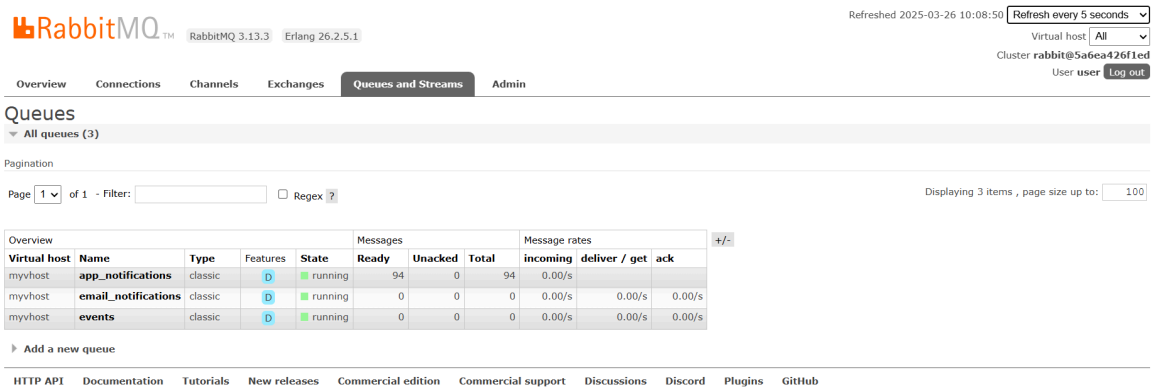


5. Observe the logs from `notification_processor.js` to verify that:
    - No users were matched to the event.
    - No messages were created or pushed to queues.

.

```
Karthiek Duggirala@LAPTOP-OF4FJBMF MINGW64 ~/Documents/sca/bioloop/api (notifications)
$ node src/scripts/notification_processor.js
WARNING: NODE_ENV value of 'default' is ambiguous.
WARNING: See https://github.com/node-config/node-config/wiki/Strict-Mode
2025-03-26 10:08:36 info: [*] Waiting for messages in events. To exit, press CTRL+C
2025-03-26 10:08:39 info: Found 0 users subscribed to event: DATASET_STAGED
2025-03-26 10:08:39 info: Created 0 notification msgs for event: DATASET_STAGED
2025-03-26 10:08:39 info: Sent 0/0 messages to notification queues
```

6. Check RabbitMQ to ensure that:
    - `email_notifications` and `app_notifications` queues remain unchanged.



7. Confirm that `emailProcessor.py` logs show **no activity**.

Expected Result

notification_processor.js logs:

```
Found 0 users subscribed to event: DATASET_STAGED

Created 0 notification msgs for event: DATASET_STAGED
```

RabbitMQ Queues:

- `email_notifications` and `app_notifications` show **no new messages**.

emailProcessor.py:

- Remains **idle** with no new processing.

This test confirms that the system behaves correctly when no one is supposed to be notified, avoiding unnecessary processing and ensuring system efficiency.

7.) Expired Subscription – Verify that users with expired subscriptions do not receive notifications when an event occurs

Description:

This test case ensures that when a user has a valid but **expired** subscription to an event (e.g., `DATASET_STAGED`), the system **does not generate or send notifications** to them. It confirms that the expiration logic is correctly enforced during event processing.

Preconditions:

- All common setup steps from the Shared Setup & Requirements section are completed.

- `DATASET_STAGED` exists in the `event_type` table (`event_type_id = 1`).

- A user (`user_id = 6`) has an **expired** subscription to this event in the `notification_subscription` table.
- No other users are subscribed to this event (for test isolation).
- The dataset `resource_id = 12` with `resource_type = dataset` exists and is valid.

Test Data

Notification Subscription Table Entry (Expired):

user_id	role_id	is_system_wide	event_type_id	resource_id	resource_type	is_active	expires_at	is_recurring
6	NULL	false	1	12	dataset	true		false

Trigger Event (`test.py`):

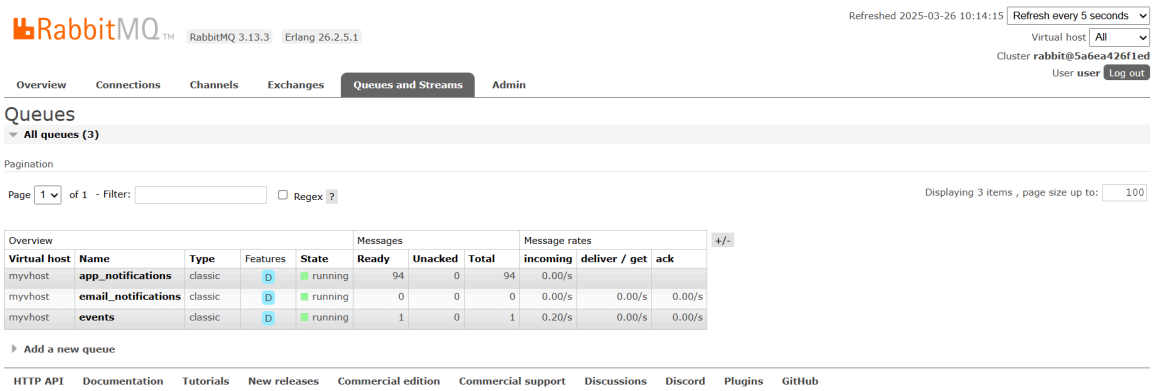
```
from workers.services.events import eventBus

eventBus.emit('DATASET_STAGED', resource_id='12',
resource_type='dataset')
```

Test Steps:

1. Insert or update the subscription for `user_id = 6` in `notification_subscription`:
    - `expires_at` is a past date (e.g., `2024-01-01T00:00:00Z`)

- `is_active = true`
- `is_recurring = false`

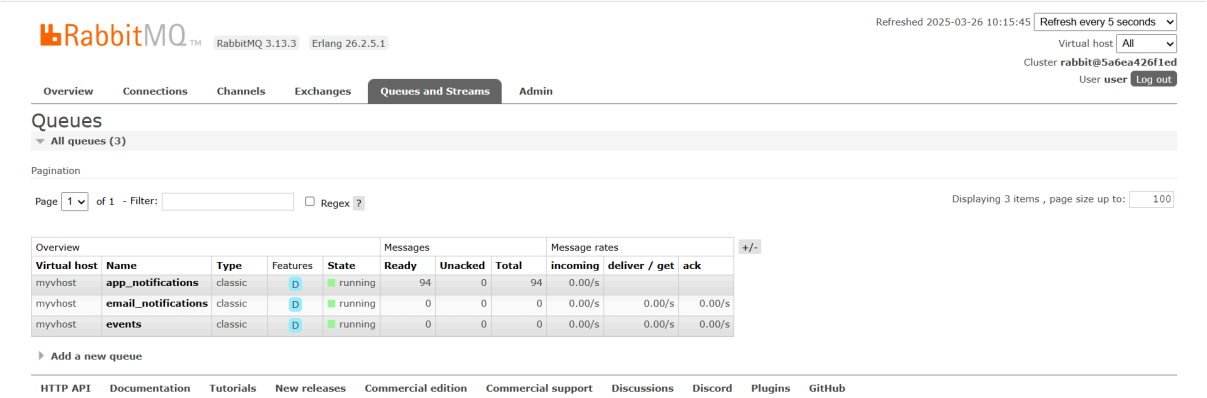
2. Ensure no other active subscriptions exist for `DATASET_STAGED`.
3. Verify that `resource_id = 12` exists and is valid in the dataset table.
4. Start `notification_processor.js`.
5. Emit the event using the `test.py` script.



6. Monitor the logs from `notification_processor.js` to confirm that no users are matched.

```
Karthiek Duggirala@LAPTOP-OF4FJBMF MINGW64 ~/Documents/sca/bioloop/api (notifications)
$ node src/scripts/notification_processor.js
WARNING: NODE_ENV value of 'default' is ambiguous.
WARNING: See https://github.com/node-config/node-config/wiki/Strict-Mode
2025-03-26 10:14:58 info: [*] Waiting for messages in events. To exit, press CTRL+C
2025-03-26 10:15:00 info: Found 0 users subscribed to event: DATASET_STAGED
2025-03-26 10:15:00 info: Created 0 notification msgs for event: DATASET_STAGED
2025-03-26 10:15:00 info: Sent 0/0 messages to notification queues
```

7. Check RabbitMQ queues to ensure no messages are published.



8. Confirm that `emailProcessor.py` remains idle (no delivery activity).

Expected Result

notification_processor.js logs:

```
Found 0 users subscribed to event: DATASET_STAGED

Created 0 notification msgs for event: DATASET_STAGED
```

RabbitMQ Queues:

- `email_notifications` and `app_notifications` queues remain **unchanged**.

emailProcessor.py:

- No logs or activity related to the event.

Database:

- The expired subscription remains intact in the database.

- It is **not deleted**, as it is recurring = false, but deletion logic only applies to one-time use after valid trigger.

This test confirms that the system **respects expiration logic** and avoids notifying users whose subscriptions have lapsed, maintaining data relevance and reducing noise.

8.) Invalid JSON in RabbitMQ – Verify that malformed event messages are safely discarded and logged without crashing the processor

Description:

This test case verifies that if an **invalid or malformed JSON message** is placed into the `events` queue, the `notification_processor.js` script will **log the error and skip the message**, rather than crashing or affecting the processing of future valid messages.

Preconditions:

- All prerequisites from the Shared Setup & Requirements section are complete.
- The `notification_processor.js` service is actively running and listening to the `events` queue.
- The system is **idle**—no valid events are being processed during this test.
- `SMTP/emailProcessor.py` may be running, but it is **not relevant** for this test case.

Test Data:

Malformed JSON Message to Manually Inject into RabbitMQ:

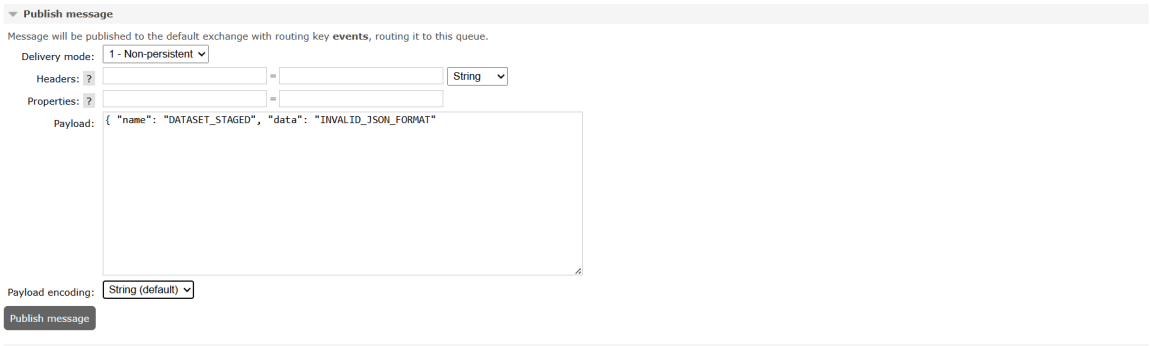
```
{ "name": "DATASET_STAGED", "data":
"INVALID_JSON_FORMAT"
```

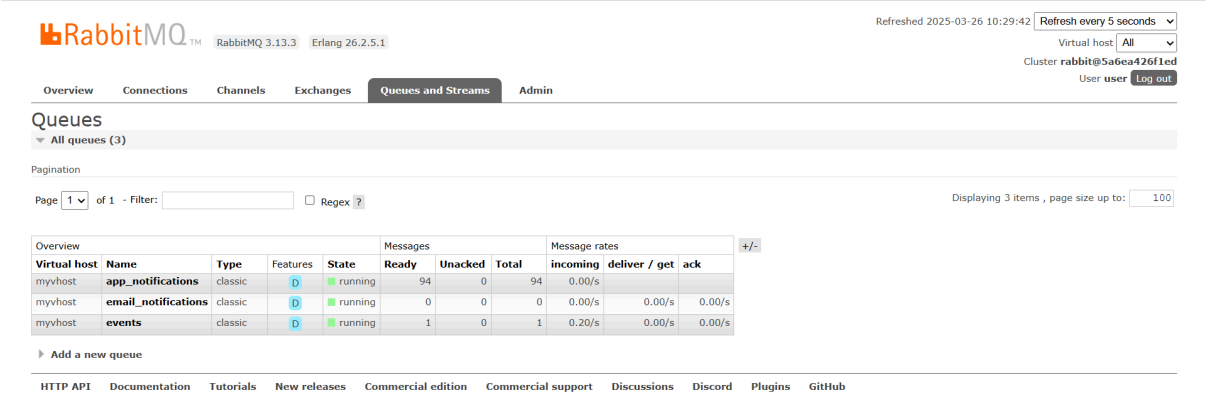
Note:
The above message is intentionally invalid (missing closing brace, improper structure).

Test Steps

1. Open the RabbitMQ Admin UI at http://localhost:15672 or use a CLI publishing tool.

2. Navigate to the `events` queue and manually **publish a malformed JSON** message using the “Publish message” form.





3. Ensure `notification_processor.js` is running and observing the queue.

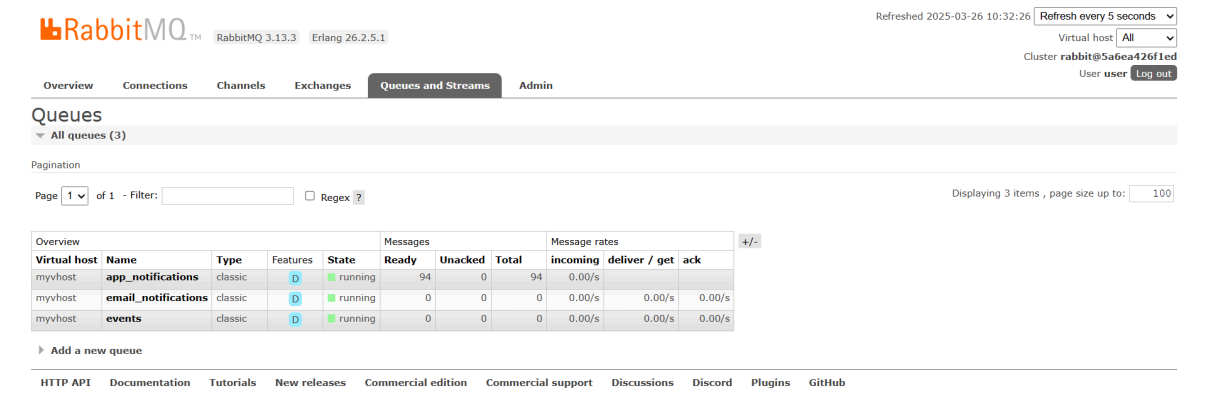
```
Karthiek Duggirala@LAPTOP-OF4FJBMF MINGW64 ~/Documents/sca/bioloop/api (notifications)
$ node src/scripts/notification_processor.js
WARNING: NODE_ENV value of 'default' is ambiguous.
WARNING: See https://github.com/node-config/node-config/wiki/Strict-Mode
2025-03-26 10:30:56 info: [*] Waiting for messages in events. To exit, press CTRL+C
```

4. Monitor the logs in the terminal where `notification_processor.js` is running.

5. Verify the following:

    - The processor logs an error about failed JSON parsing.

    - The malformed message is not reprocessed infinitely.



6. If retry headers are implemented:
    - The message should retry a maximum number of times (e.g., 5).
    - After that, it should be acknowledged and discarded.
7. Confirm that **no notification messages** are pushed to downstream queues (`email_notifications`, `app_notifications`, etc.).

Expected Result

notification_processor.js logs:

```
Failed to parse JSON message
```

If retry mechanism is enabled:

```
Message discarded after 5 retry attempts
```

RabbitMQ Queues:

- No new messages in `email_notifications` or other queues.

emailProcessor.py:

- Remains idle with no activity.

System Behavior:

- The processor **continues running normally**, handling other messages if added later.

- No crash or pipeline interruption occurs.

This test confirms the **fault tolerance and input validation** of the notification system. It ensures that invalid or malformed input from RabbitMQ is safely handled without risk to system stability or downstream services.

9.) Email Sending Failure – Verify that when email delivery fails (e.g., SMTP server is down), the system logs the error and handles the failure without crashing

Description:

This test case validates the system's behavior when the **SMTP server is unavailable**. It ensures that the `emailProcessor.py` service **does not crash**, properly **logs the failure**, and (if applicable) **retries or discards** the message based on configuration. This confirms the system's **resilience to external service outages**.

Preconditions:

- All common setup steps from the Shared Setup & Requirements section are completed.
- A user (e.g., `user_id = 6`) is actively subscribed to `DATASET_STAGED` with `email` delivery enabled.
- The `notification_processor.js` service is running and will enqueue a message in the `email_notifications` queue.
- The SMTP server is **intentionally misconfigured or stopped** to simulate a failure.

Test Data:

Notification Subscription Table Entry:

user_id	event_type_id	resource_id	resource_type	is_active	is_recurring
6	1	12	dataset	true	true

Trigger Event (`test.py`):

```
from workers.services.events import eventBus

eventBus.emit('DATASET_STAGED', resource_id='12',
resource_type='dataset')
```

Simulating SMTP Failure (Choose one):

- Shut down the SMTP server manually.

Misconfigure SMTP in `config.yaml`:

```
SMTP:

  host: localhost

  port: 2525  # Wrong port to simulate failure
```

- Block SMTP connection via firewall or network rules.

Test Steps:

1. Verify that the test user has an active subscription and email preference enabled.

notification_subscription × event_type role dataset

Properties Data ER Diagram app Databases app Schemas public Tables notification_subscription

notification_subscription Enter a SQL expression to filter results (use Ctrl+Space)

	id	is_system_wide	user_id	role_id	is_active	created_at	expires_at	event_type_id	resource_id	resource_type	is_recurring
1	12	[ ]	6	[NULL]	[v]	-03-26 10:37:51.683	[NULL]	1	12	dataset	[v]

2. Misconfigure or stop the SMTP server to **simulate failure**.

3. Start `notification_processor.js` and `emailProcessor.py`.

```
Karthiek Duggirala@LAPTOP-OF4FJBMF MINGW64 ~/Documents/sca/bioloop/api (notifications)
$ node src/scripts/notification_processor.js
WARNING: NODE_ENV value of 'default' is ambiguous.
WARNING: See https://github.com/node-config/node-config/wiki/Strict-Mode
2025-03-26 10:39:37 info: [*] Waiting for messages in events. To exit, press CTRL+C
2025-03-26 10:39:39 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 10:39:41 info: Created 2 notification msgs for event: DATASET_STAGED
2025-03-26 10:39:41 info: Sent 2/2 messages to notification queues
```

4. Emit the `DATASET_STAGED` event using `test.py`.

5. Monitor `emailProcessor.py` logs to observe how the failure is handled.

```
Karthiek Duggirala@LAPTOP-OF4FJBMF MINGW64 ~/Documents/sca/bioloop/workers (notifications)
$ python -m workers.scripts.emailProcessor
loading  conf
Waiting for messages in queue: email_notifications...
📩 Received email request: 📊 New dataset staged: PCM230412_new -> kadugg@iu.edu
Traceback (most recent call last):
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\runpy.py", line 196, in _run_module_as_main
    return _run_code(code, main_globals, None,
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\runpy.py", line 86, in _run_code
    exec(code, run_globals)
  File "C:\Users\Karthiek Duggirala\Documents\sca\bioloop\workers\workers\scripts\emailProcessor.py", line 101, in <module>
    asyncio.run(main())
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\asyncio\runners.py", line 44, in run
    return loop.run_until_complete(main)
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\asyncio\base_events.py", line 649, in run_until_complete
    return future.result()
  File "C:\Users\Karthiek Duggirala\Documents\sca\bioloop\workers\workers\scripts\emailProcessor.py", line 89, in main
    loop.add_signal_handler(sig, shutdown)
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\asyncio\events.py", line 553, in add_signal_handler
    raise NotImplementedError
NotImplementedError
(bioloop)
```

6. Confirm that the message does **not crash** the service.

7. Optionally, check if the message is retried or silently discarded.

8. Restart the SMTP server to ensure future messages are sent properly.

Expected Result

emailProcessor.py logs:

```
📩 Received email request: New dataset staged -> kadugg@iu.edu

Failed to send email: [Errno 111] Connection refused
```

RabbitMQ Behavior:

- Messages may remain in queue for retry (if retry logic is implemented).
- Or it may be discarded after a single failure (depending on `no_ack=True` or message handling strategy).

System Behavior:

- `emailProcessor.py` **does not crash** or exit unexpectedly.
- Other messages (if any) are still processed correctly.
- Once the SMTP server is restored, future messages are **delivered successfully**.

This test confirms the **fault-tolerance** of the notification pipeline and its ability to gracefully handle **external service outages** without interrupting the core workflow.

10.) RabbitMQ Server Failure – Verify that if RabbitMQ is down, the system logs the connection failure and retries or exits gracefully

Description:

This test case verifies the system's behavior when the **RabbitMQ server is unavailable**. It ensures that event-driven services such as `notification_processor.js`, `emailProcessor.py`, and the `test.py` event emitter **do not crash**, and instead handle the failure by **logging errors and exiting or retrying as configured**.

Preconditions:

- All other common setup steps from the Shared Setup & Requirements section are completed.
- RabbitMQ was previously running and correctly configured.
- For this test, RabbitMQ will be **intentionally stopped or made unreachable**.
- SMTP/email server is irrelevant to this test (can remain running or inactive).

Test Data:

Trigger Event (`test.py`) – Optional if RabbitMQ is already down:

```
from workers.services.events import eventBus

eventBus.emit('DATASET_STAGED', resource_id='12',
resource_type='dataset')
```

Simulate RabbitMQ Failure:

If using Docker:

```
docker stop rabbitmq
```

If installed locally:

```
sudo systemctl stop rabbitmq-server
```

Test Steps:

1. **Shut down RabbitMQ** using Docker or systemd.
2. Attempt to run the following services and observe behavior:
   - `test.py` script
   - `notification_processor.js`
   - `emailProcessor.py`
3. Monitor each of their logs for **connection errors**.

```
Karthiek Duggirala@LAPTOP-OF4FJBMF MINGW64 ~/Documents/sca/bioloop/workers (notifications)
$ python test.py
loading  conf
Traceback (most recent call last):
  File "C:\Users\Karthiek Duggirala\Documents\sca\bioloop\workers\test.py", line 1, in <module>
    from workers.services.events import eventBus
  File "C:\Users\Karthiek Duggirala\Documents\sca\bioloop\workers\workers\services\events.py", line 43, in <module>
    eventBus = EventBus(rabbitmq_url=broker_url, queue=config['notifications']['event_queue'])
  File "C:\Users\Karthiek Duggirala\Documents\sca\bioloop\workers\workers\services\events.py", line 14, in __init__
    self.connection = pika.BlockingConnection(pika.URLParameters(rabbitmq_url))
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\site-packages\pika\adapters\blocking_connection.py", line 360, in
__init__
    self._impl = self._create_connection(parameters, _impl_class)
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\site-packages\pika\adapters\blocking_connection.py", line 451, in
_create_connection
    raise self._reap_last_connection_workflow_error(error)
pika.exceptions.AMQPConnectionError
(bioloop)
```

```
Karthiek Duggirala@LAPTOP-OF4FJBMF MINGW64 ~/Documents/sca/bioloop/api (notifications)
$ node src/scripts/notification_processor.js
WARNING: NODE_ENV value of 'default' is ambiguous.
WARNING: See https://github.com/node-config/node-config/wiki/Strict-Mode
2025-03-26 10:44:12 error: Error in notification processor: connect ECONNREFUSED 127.0.0.1:5672
C:\Users\Karthiek Duggirala\Documents\sca\bioloop\api\src\scripts\notification_processor.js:95
    await Promise.allSettled(channels.map((channel) => channel.close()));
                                                                 ^

TypeError: Cannot read properties of undefined (reading 'close')
    at C:\Users\Karthiek Duggirala\Documents\sca\bioloop\api\src\scripts\notification_processor.js:95:64
    at Array.map (<anonymous>)
    at cleanup (C:\Users\Karthiek Duggirala\Documents\sca\bioloop\api\src\scripts\notification_processor.js:95:39)
    at run (C:\Users\Karthiek Duggirala\Documents\sca\bioloop\api\src\scripts\notification_processor.js:189:11)
    at process.processTicksAndRejections (node:internal/process/task_queues:95:5)

Node.js v19.9.0
(bioloop)
```

```
$ python -m workers.scripts.emailProcessor
loading  conf
Traceback (most recent call last):
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\site-packages\aiormq\connection.py", line 457, in connect
    reader, writer = await asyncio.open_connection(
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\asyncio\streams.py", line 48, in open_connection
    transport, _ = await loop.create_connection(
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\asyncio\base_events.py", line 1081, in create_connection
    raise exceptions[0]
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\asyncio\base_events.py", line 1060, in create_connection
    sock = await self._connect_sock(
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\asyncio\base_events.py", line 969, in _connect_sock
    await self.sock_connect(sock, address)
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\asyncio\proactor_events.py", line 709, in sock_connect
    return await self._proactor.connect(sock, address)
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\asyncio\windows_events.py", line 826, in _poll
    value = callback(transferred, key, ov)
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\asyncio\windows_events.py", line 613, in finish_connect
    ov.getresult()
ConnectionRefusedError: [WinError 1225] The remote computer refused the network connection

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\runpy.py", line 196, in _run_module_as_main
    return _run_code(code, main_globals, None,
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\runpy.py", line 86, in _run_code
    exec(code, run_globals)
  File "C:\Users\Karthiek Duggirala\Documents\sca\bioloop\workers\workers\scripts\emailProcessor.py", line 101, in <module>
    asyncio.run(main())
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\asyncio\runners.py", line 44, in run
    return loop.run_until_complete(main)
  File "C:\Users\Karthiek Duggirala\Documents\sca\bioloop\workers\workers\scripts\emailProcessor.py", line 67, in main
    connection = await aio_pika.connect_robust(RABBITMQ_URL)
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\site-packages\aio_pika\robust_connection.py", line 334, in connect_robust
    await connection.connect(timeout=timeout)
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\site-packages\aio_pika\robust_connection.py", line 180, in connect    await self.__fail_fast_future
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\site-packages\aio_pika\robust_connection.py", line 135, in __connection_factory
    await Connection.connect(self, self.__connect_timeout)
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\site-packages\aio_pika\connection.py", line 125, in connect
    self.transport = await UnderlayConnection.connect(
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\site-packages\aio_pika\abc.py", line 679, in connect
    connection = await cls.make_connection(
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\site-packages\aio_pika\abc.py", line 667, in make_connection
    connection: aiormq.abc.AbstractConnection = await asyncio.wait_for(
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\asyncio\tasks.py", line 408, in wait_for
    return await fut
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\site-packages\aiormq\connection.py", line 920, in connect
    await connection.connect(client_properties or {})
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\site-packages\aiormq\base.py", line 164, in wrap
    return await self.create_task(func(self, *args, **kwargs))
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\site-packages\aiormq\abc.py", line 44, in __inner
    return await self.task
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\site-packages\aiormq\connection.py", line 464, in connect
    raise AMQPConnectionError(*e.args) from e
aiormq.exceptions.AMQPConnectionError: [WinError 1225] The remote computer refused the network connection
(bioloop)
```

4. Confirm that the services:
    - Do **not crash unexpectedly**
    - **Log the failure**
    - Either **retry** or **exit cleanly**
5. Restart RabbitMQ.
6. Observe whether:

- The services **automatically reconnect** (if retry logic is implemented), **or**
- Require **manual restart** if they exited previously.

Expected Result

`test.py` **Output:**

```
pika.exceptions.AMQPConnectionError: [Errno 111]
Connection refused
```

- Script exits gracefully without hanging.

`notification_processor.js` **Logs:**

```
Error in notification processor: connect ECONNREFUSED
127.0.0.1:5672

Connection closed

Process exited
```

- Service logs the connection issue and **does not crash**.

`emailProcessor.py` **Logs:**

```
aio_pika.exceptions.AMQPConnectionError: Connection
refused
```

- The Python process logs the error and shuts down gracefully or retries (based on configuration).

After RabbitMQ is Restarted:

- If **retry logic is implemented**: services automatically **reconnect** and resume operations.
- If **no retry logic**: services need to be manually restarted to resume functionality.

This test confirms the **resilience** of the notification system during RabbitMQ outages, ensuring that dependent services **fail gracefully**, do not lose data silently, and are recoverable without unexpected crashes.

11.) High Volume Test – Verify that the notification system can process a large number of events (e.g., 10,000) efficiently without failure

Description:

This test case ensures the notification pipeline can handle **high event throughput** (e.g., 10,000 `DATASET_STAGED` events) without failures, memory leaks, message loss, or performance degradation. It validates the system's **scalability and robustness** under simulated production-like load.

Preconditions:

- All steps from the Shared Setup & Requirements section are completed.
- A test user (`user_id = 6`) is **actively subscribed** to `DATASET_STAGED` events with `email` notification preferences

enabled.

- The dataset with `resource_id = 12`, `resource_type = dataset` exists or is mocked and accessible.
- RabbitMQ queues (`events`, `email_notifications`) are empty before starting the test.

Test Data:

Notification Subscription Table Entry:

user_id	event_type_id	resource_id	resource_type	is_active	is_recurring
6	1	12	dataset	true	true

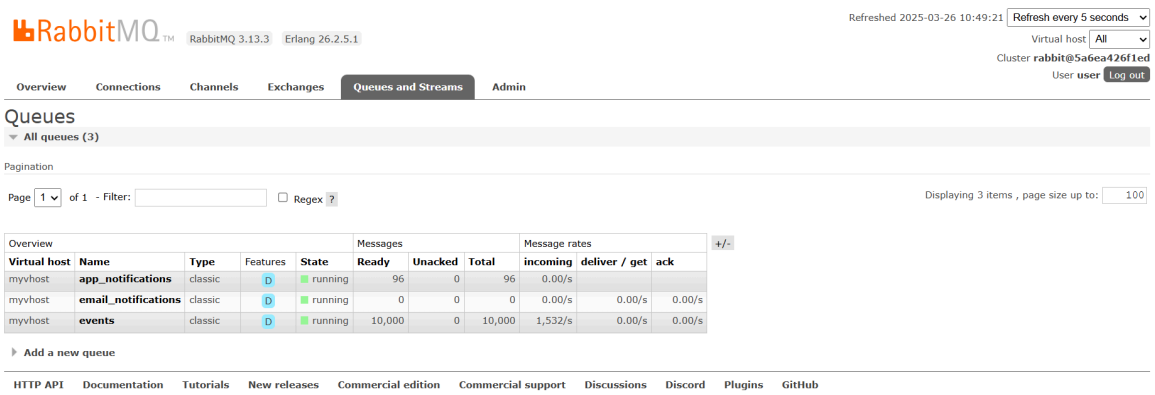
Trigger Events (`test.py`):

```
from workers.services.events import eventBus


# Emit 10,000 identical events for testing

for i in range(10000):

    eventBus.emit('DATASET_STAGED', resource_id='12',
resource_type='dataset')
```

Test Steps:

1. Ensure the subscription and email preference for `user_id = 6` are properly configured.

2. Start all required services:
    - `notification_processor.js`
    - `emailProcessor.py`
    - RabbitMQ
    - SMTP server
3. Run the above `test.py` script to emit **10,000 events**.
4. Use the RabbitMQ Admin UI to monitor:
    - The `events` queue for message ingestion.
    - The `email_notifications` queue for outbound messages.



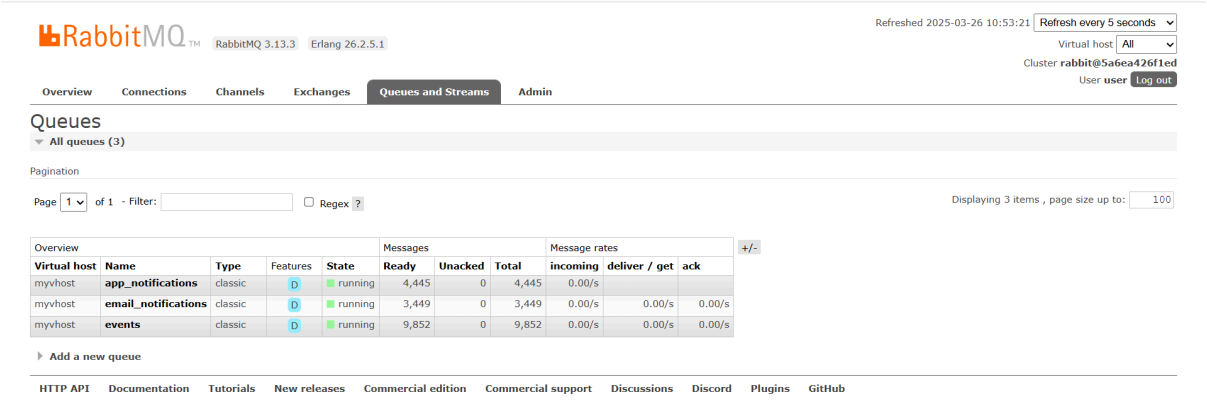
5. Monitor `notification_processor.js` logs for:
    - Message consumption
    - Notification creation
    - Queuing stats

```
2025-03-26 10:50:33 error: Error processing event
Invalid `prisma.dataset.findFirstOrThrow()` invocation in
C:\Users\Karthiek Duggirala\Documents\sca\bioloop\api\src\services\dataset.js:126:40

  123   },
  124 } : INCLUDE_WORKFLOWS;
  125
→ 126 const dataset = await prisma.dataset.findFirstOrThrow(
Timed out fetching a new connection from the connection pool. More info: http://pris.ly/d/connection-pool (Current connection poo
l timeout: 10, connection limit: 9)
C:\Users\Karthiek Duggirala\Documents\sca\bioloop\api\src\scripts\notification_processor.js:161
          const retryCount = msg.properties.headers['x-retry-count'] || 0;
                                                                        ^

TypeError: Cannot read properties of undefined (reading 'x-retry-count')
    at readChannel.consume.noAck (C:\Users\Karthiek Duggirala\Documents\sca\bioloop\api\src\scripts\notification_processor.js:161
:52)

Node.js v19.9.0
(bioloop)
```

6. Observe `emailProcessor.py`:

   - Email processing throughput
   - Logging of errors or retries (if any)

```
$ python -m workers.scripts.emailProcessor
loading  conf
Waiting for messages in queue: email_notifications...
📩 Received email request: 📊 New dataset staged: PCM230412_new -> kadugg@iu.edu
📩 Received email request: 📊 New dataset staged: PCM230412_new -> kadugg@iu.edu
📩 Received email request: 📊 New dataset staged: PCM230412_new -> kadugg@iu.edu
Traceback (most recent call last):
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\runpy.py", line 196, in _run_module_as_main
    return _run_code(code, main_globals, None,
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\runpy.py", line 86, in _run_code
    exec(code, run_globals)
  File "C:\Users\Karthiek Duggirala\Documents\sca\bioloop\workers\workers\scripts\emailProcessor.py", line 101, in <module>
    asyncio.run(main())
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\asyncio\runners.py", line 44, in run
    return loop.run_until_complete(main)
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\asyncio\base_events.py", line 649, in run_until_complete
    return future.result()
  File "C:\Users\Karthiek Duggirala\Documents\sca\bioloop\workers\workers\scripts\emailProcessor.py", line 89, in main
    loop.add_signal_handler(sig, shutdown)
  File "C:\Users\Karthiek Duggirala\anaconda3\envs\bioloop\lib\asyncio\events.py", line 553, in add_signal_handler
    raise NotImplementedError
NotImplementedError
(bioloop)
```



7. Optionally track system metrics (CPU, memory, processing time) for performance profiling.

Expected Result:

notification_processor.js logs:

```
Found 1 users subscribed to event: DATASET_STAGED

Created 10000 notification msgs for event: DATASET_STAGED

Sent 10000/10000 messages to notification queues
```

RabbitMQ Queues:

- All messages from the `events` queue are consumed.
- `email_notifications` queue receives **10,000** messages (1 per event).

emailProcessor.py logs:

```
📩 Received email request: New dataset staged -> kadugg@iu.edu
```

```
Email sent successfully to kadugg@iu.edu

...

(repeated thousands of times)
```

System Behavior:

- No crashes, unhandled exceptions, or out-of-memory errors occur.
- Messages are processed efficiently and in a timely manner (performance varies by hardware).
- If throttling or rate-limiting is configured, email delivery should follow defined behavior (e.g., queued, delayed, or retried).

This test validates the system's ability to handle **bulk notifications** triggered by real-world workflows such as research dataset staging, automated jobs, or multi-user requests. It ensures stability, scalability, and consistent user experience under high load.

12.) Parallel Processing – Verify that running multiple notification processor instances handles events concurrently without duplication or data loss

Description:

This test case validates that the notification system supports **horizontal scaling** by allowing multiple `notification_processor.js` instances to consume and process events **concurrently**. It ensures **no duplication, no data loss**, and **balanced distribution** of event handling between processor instances.

Preconditions:

- All common setup steps from the Shared Setup & Requirements section are completed.
- `user_id = 6` is actively subscribed to `DATASET_STAGED` with email delivery enabled.
- `resource_id = 12`, `resource_type = dataset` exists and is valid.
- RabbitMQ is running and the `events` queue is ready.
- SMTP/email server is running (to validate downstream delivery).
- Multiple terminal sessions are available to launch processors in parallel.

Test Data:

Notification Subscription Table Entry:

user_id	event_type_id	resource_id	resource_type	is_active	is_recurring
6	1	12	dataset	true	true

Trigger Events (`test.py`):

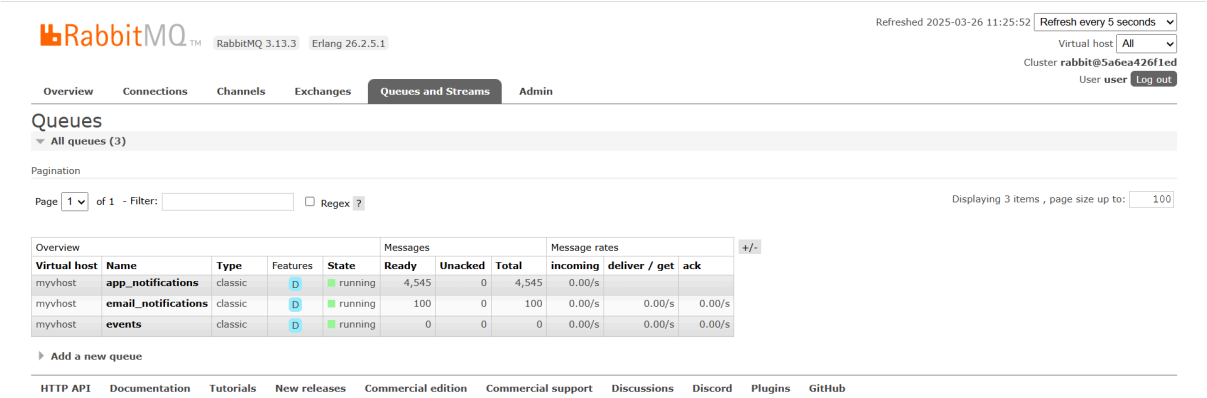
```
from workers.services.events import eventBus


# Emit 100 test events for concurrency simulation

for i in range(100):
```

```
eventBus.emit('DATASET_STAGED', resource_id='12', resource_type='dataset')
```

Test Steps

1. Ensure the user subscription and notification preference are properly configured.

2. Emit **100 events** into the `events` queue using the provided `test.py`.

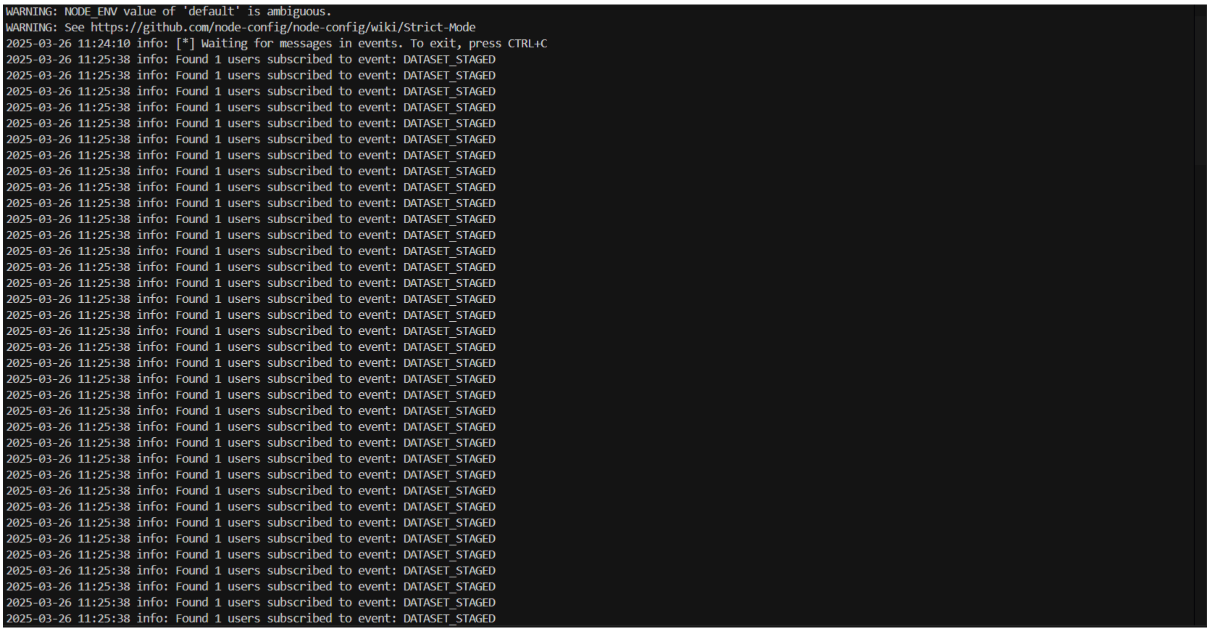


3. Open **two or more terminal windows** (or background processes).

4. In each terminal, start a separate instance of `notification_processor.js`:

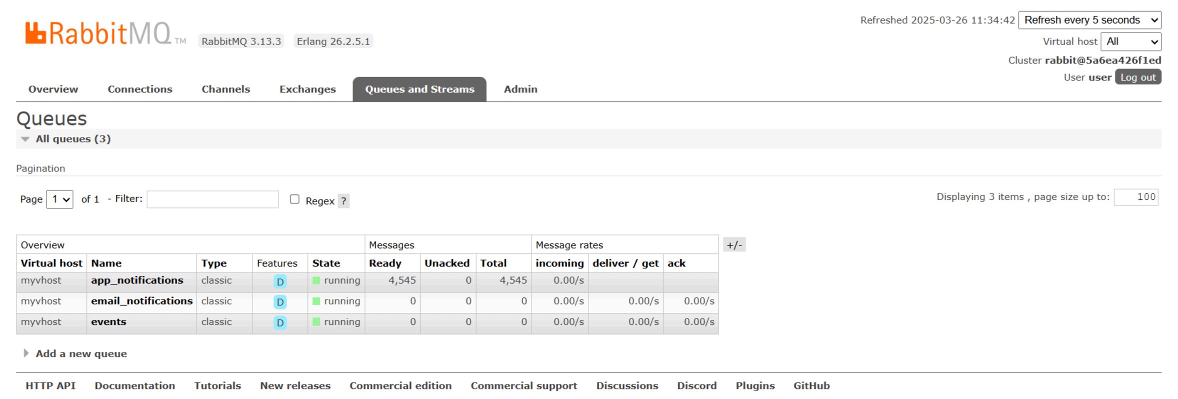
```
node src/scripts/notification_processor.js
```

5. Monitor the logs in each processor instance to track which events are processed.



```
WARNING: NODE_ENV value of 'default' is ambiguous.
WARNING: See https://github.com/node-config/node-config/wiki/Strict-Mode
2025-03-26 11:24:10 info: [*] Waiting for messages in events. To exit, press CTRL+C
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
2025-03-26 11:25:38 info: Found 1 users subscribed to event: DATASET_STAGED
```

6. Confirm:
   - **No event is processed more than once**
   - **Each event is handled exactly once by a single instance**
7. Check the `email_notifications` queue to verify messages are queued for delivery.



Refreshed 2025-03-26 11:34:42 Refresh every 5 seconds
Virtual host All
Cluster rabbit@5a6ea426f1ed
User user

Queues

All queues (3)

Pagination

Page 1 of 1 - Filter: ☐ Regex ?

Displaying 3 items , page size up to: 100

Overview					Messages			Message rates		
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack
myvhost	app_notifications	classic	D	running	4,545	0	4,545	0.00/s		
myvhost	email_notifications	classic	D	running	0	0	0	0.00/s	0.00/s	0.00/s
myvhost	events	classic	D	running	0	0	0	0.00/s	0.00/s	0.00/s

8. Monitor `emailProcessor.py` for message delivery confirmation.

Expected Result

Processor Logs:

- Logs show **distributed workload**, e.g.:

```
Instance 1: Created 53 notification msgs for event: DATASET_STAGED

Instance 2: Created 47 notification msgs for event: DATASET_STAGED
```

RabbitMQ Queues:

- `events` queue is **drained to 0** — all events consumed.
- `email_notifications` queue contains **100 messages**.

emailProcessor.py Logs:

```
📩 Received email request: New dataset staged -> kadugg@iu.edu

Email sent successfully to kadugg@iu.edu

... (100 entries)
```

System Behavior:

- No **duplicate processing** or message collision.
- No **message loss**.
- Services continue operating **smoothly under concurrent load**.

This test confirms that the notification system is designed for **concurrent processing**, enabling **horizontal scaling** in real-world deployments and ensuring reliability during high-volume or distributed workloads.