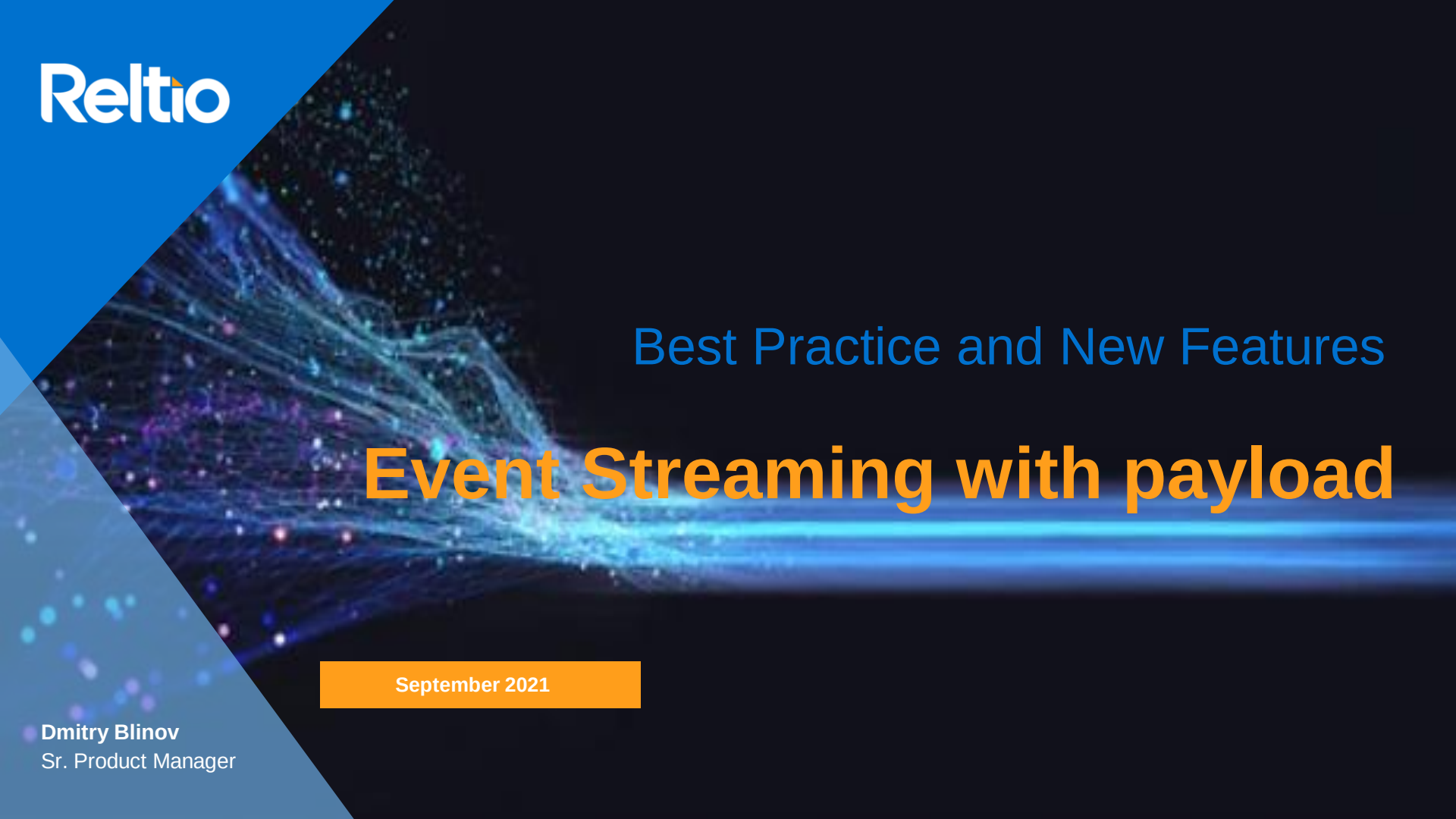


Best Practice and New Features

Event Streaming with payload

September 2021

Dmitry Blinov
Sr. Product Manager

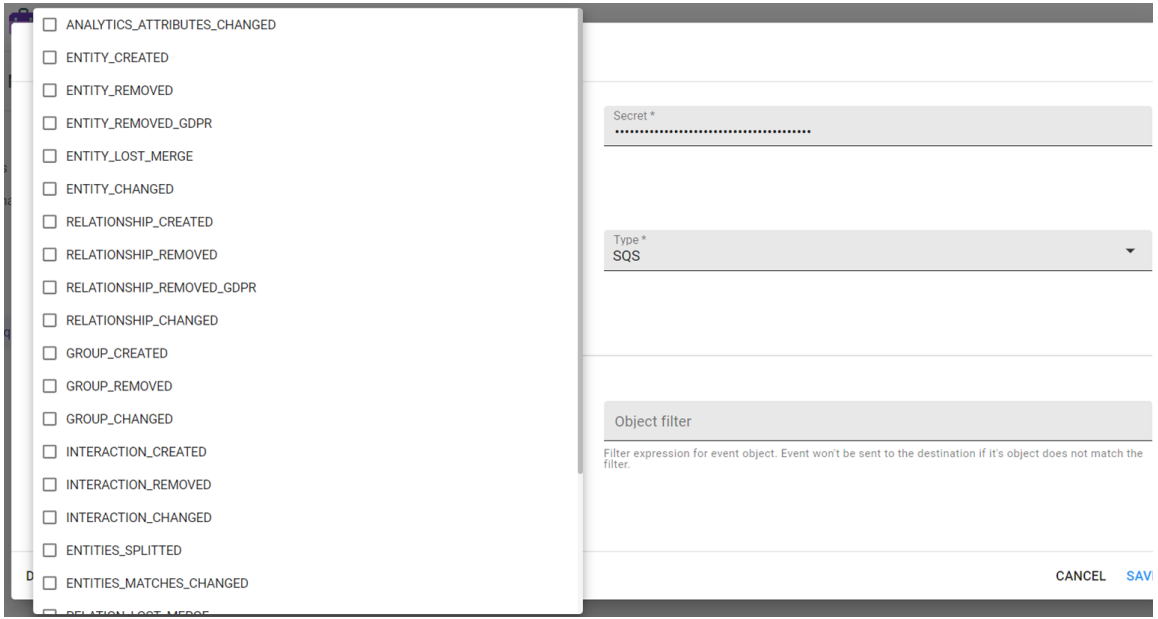
Agenda

- ★ What is Reltio Event Streaming ?
- ★ How Event Streaming Works?
- ★ Supported types of the queues and known limitations
- ★ How to deal with the message size limit
- ★ Event Streaming Architecture - why no FIFO ?
- ★ How to order events ?
- ★ How do I filter event stream ?
- ★ A deeper dive into the event filtering
- ★ Payload configuration today
- ★ Delta event streaming - coming in 21.3 preview
- ★ Getting the best value out of event streaming and future plans

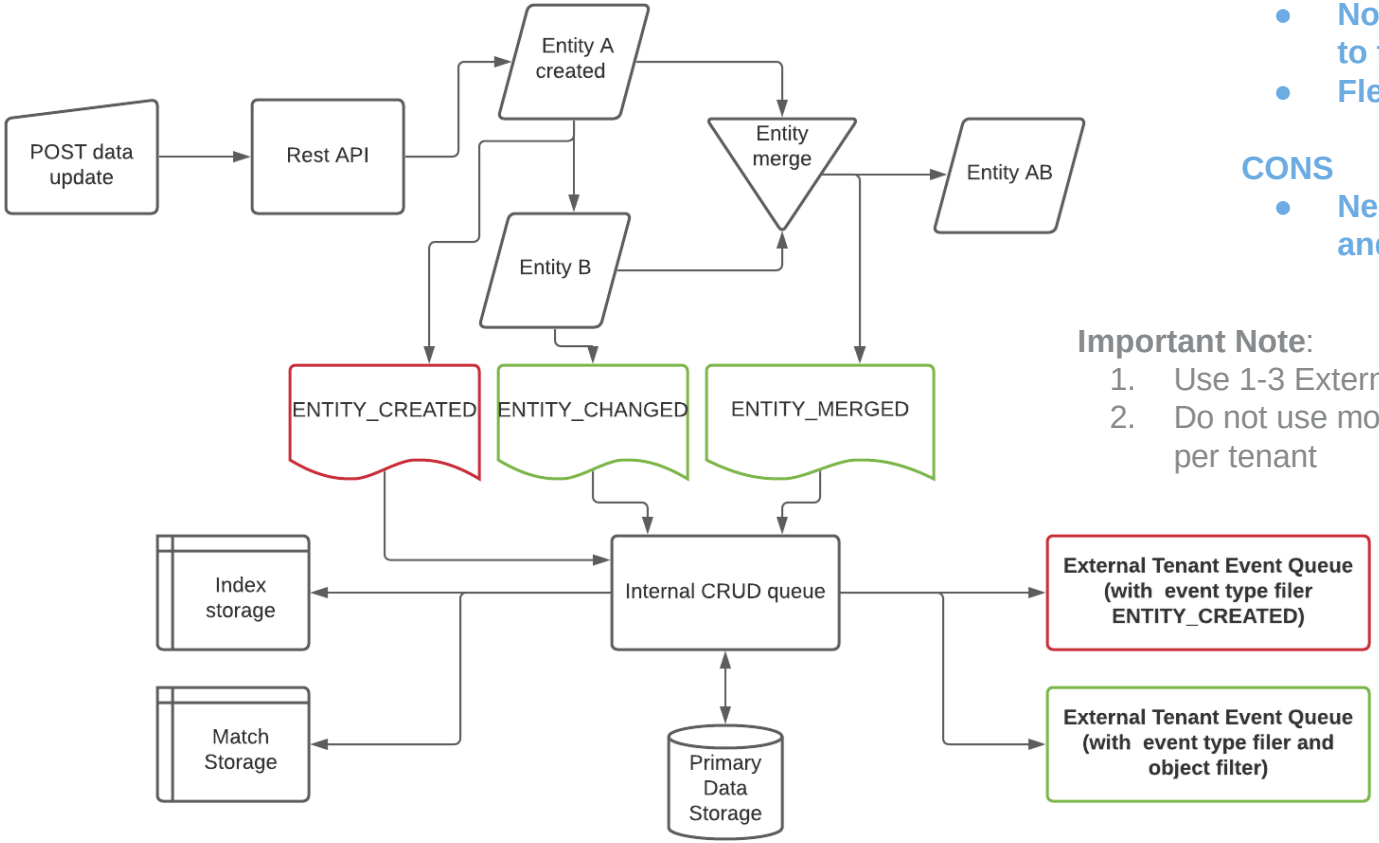
What is Reltio Event Streaming ?

Event Streaming or Message Streaming is Reltio Platform Service that enables the Reltio platform to process events from an internal queue into an external queue or topic in JSON format.

- Event Streaming supports filter by event type
- It supports object filters for events - same format as for Search API
- Events with payload are supported. Today payload may be configured with limited granularity (ex. Include all attributes or include all crosswalks)
- NEW - payload may be configured per queue
- NEW - streaming of delta payload supported
- COMING in 21.3 - payload configuration in Tenant Management UI
- Multiple destinations per queue



How Event Streaming Works?



PROS

- Active notification of changes
- No need to do additional request to the primary data storage
- Flexible filtering

CONS

- Need to take care of event size and ordering

Important Note:

1. Use 1-3 External queues per tenant
2. Do not use more than 5 external queues per tenant

Supported types of the queues and known limitations

Queue type by cloud	Maximum event size allowed
AWS SQS	256 kb
GCP PubSub	~10Mb
MS Azure	1Mb

*Projected future: support direct streaming to Kafka

Reltio Platform stream events into all of the above types of the queues

- Using the same JSON format
- The format that is used is not-decorated JSON schema. By default such JSON has no additional characters inserted - **no** spaces, returns etc. is added to the JSON by default

At the same time, topics may have their own schemas, including customized ones. As result, JSON message consumed from this topic may be formatted differently depending on the topic configuration.

IMPORTANT NOTE: **Always parse messages with JSON parser and never as a String object**

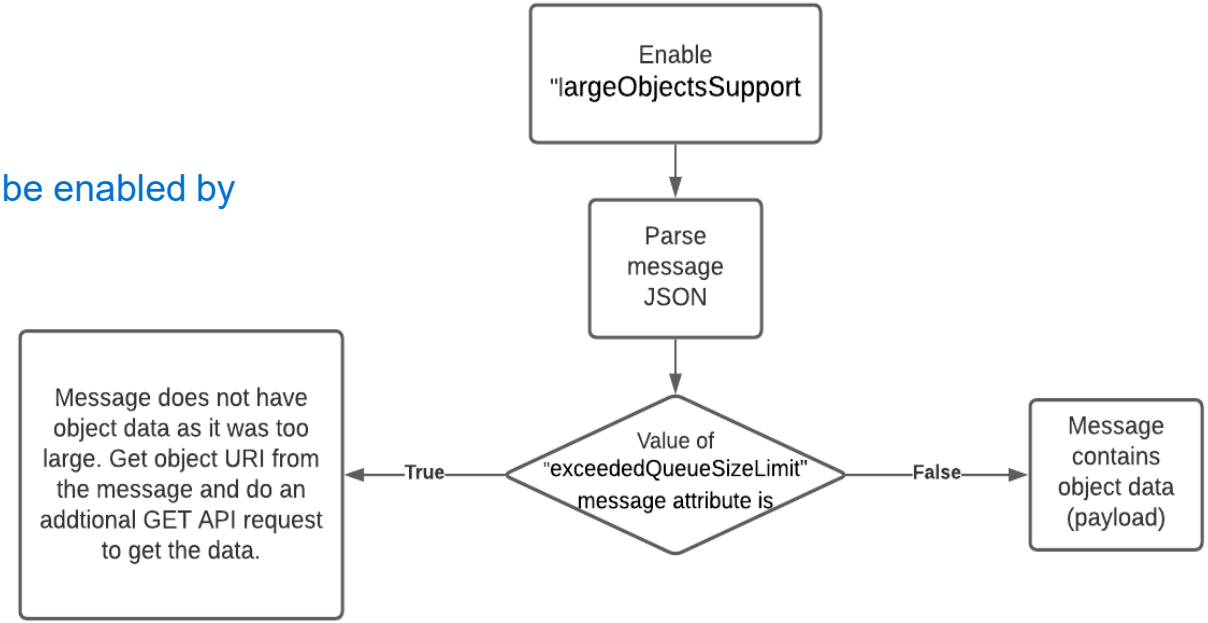
How to deal with the message size limit

Today

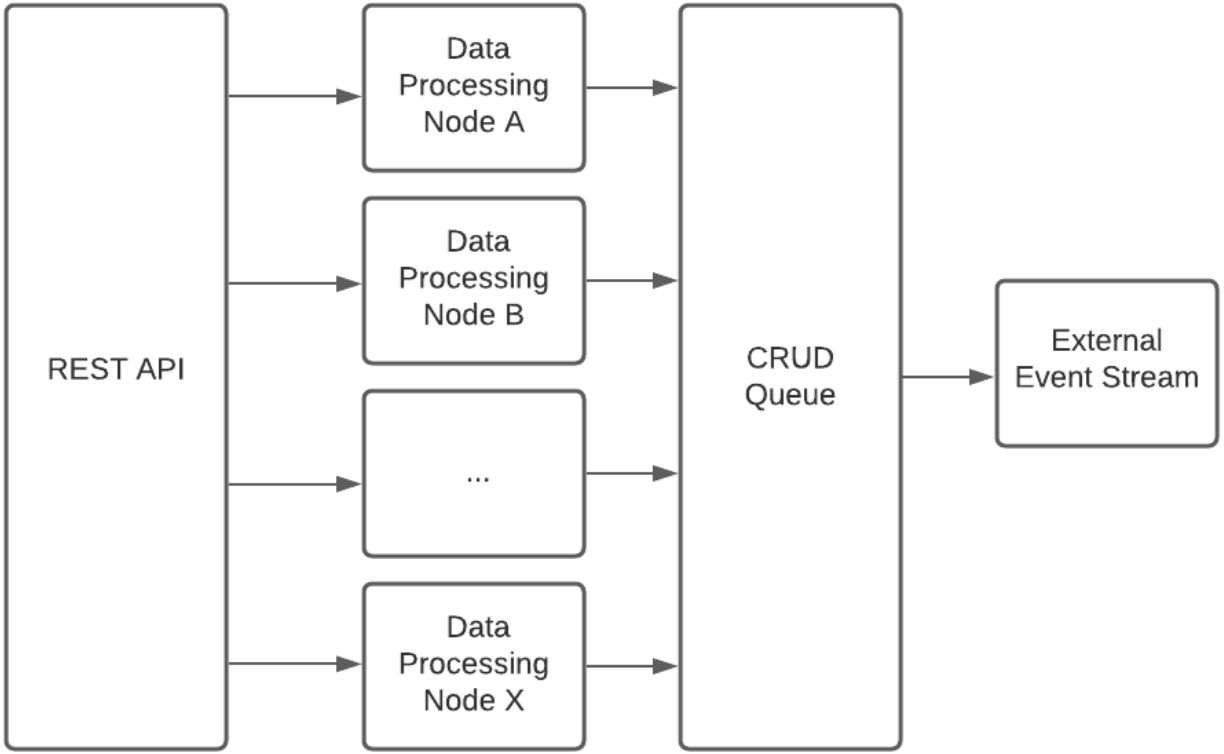
- Make sure you have “largeObjectsSupport“ tenant configuration enabled
- Use "exceededQueueSizeLimit" attribute of the message to build the consumer logic to work with large messages

21.3 preview

- “largeObjectsSupport” will be enabled by default on all tenants



Event Streaming Architecture - why no FIFO ?

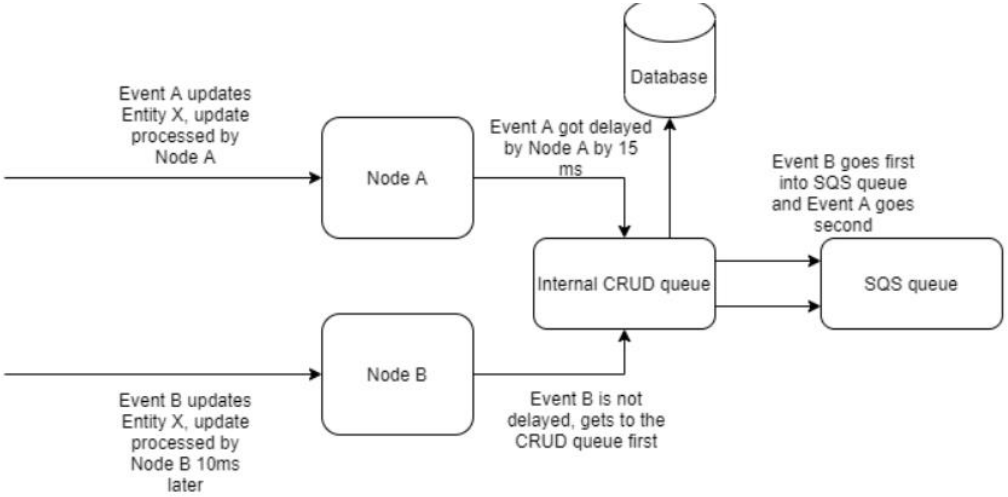


In the Reltio Platform multiple requests are processed in parallel by multiple nodes. This allows fast and efficient horizontal scaling. But this also allows a potential scenario where the request that came a few milliseconds later, will be processed faster and get into the queue first.
In this architecture use of FIFO is not relevant and instead, other techniques should be used to order events by timestamp.

How to order events

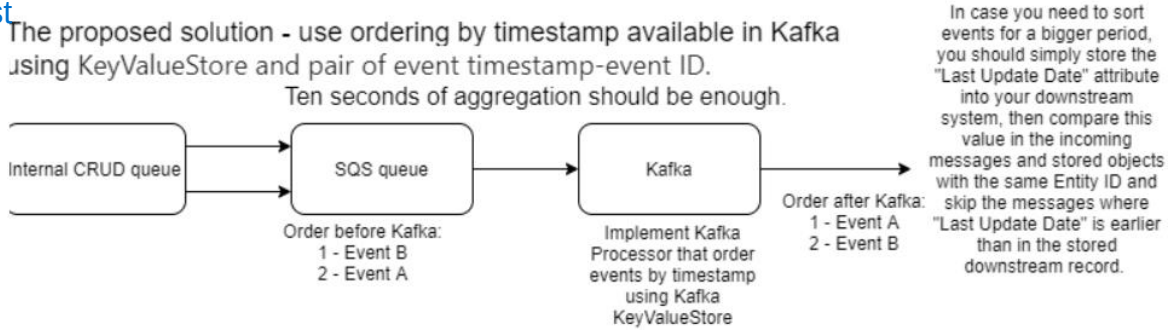
Today

- Include objectVersion field
- (*Optional, secondary alternative to objectVersion*) Include **commitTime timestamp** into the message payload fields
- Whenever you aggregate or persist streamed object - store and compare the version or timestamp
- Do not persist or propagate the consumed message into the downstream if you already have latest timestamp stored



Possible solution

- Re-stream messages to Kafka to aggregate and order messages

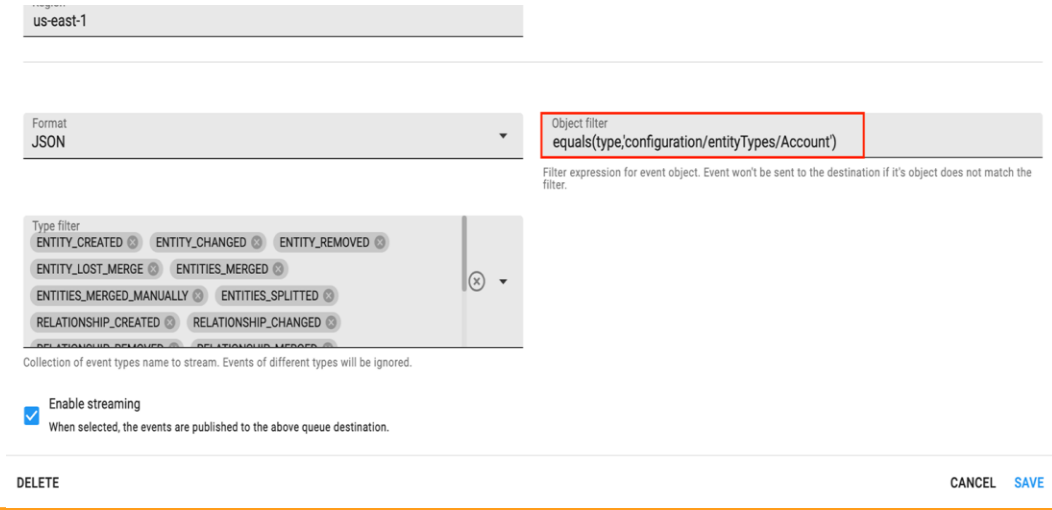


How do I filter event stream ?

- Reltio Event Streaming supports standard Reltio Object filter queries - the same format as Search and Export.
- Format for the filter query parameter: **filter=({Condition Type}[AND/OR {Condition Type}]*)**
- Today filter configuration is supported on the destination level, but Tenant Management UI allows only one destination per queue

Some examples

- **Filter through all attribute values:** `filter=equals(attributes, 'Nick')`
- **Fuzzy search** `fuzzy(attributes.Name,'Mic')`
- **Exact search with equals and equalsCaseSensitive, containsWordStartingWith, startsWith, contains, etc.**
- **regexp** `filter=regexp(attributes.Name, 'Mat.*')`
- **gt, lt, gte,lte, range** `filter=range(attributes.Age, 20, 40)`

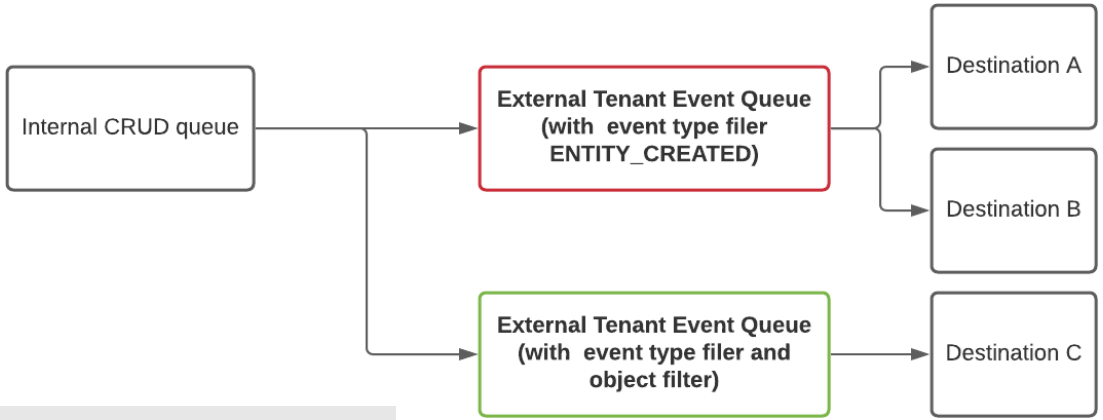


A deeper dive into the event filtering

- Filter can be configured per destination
- **Multiple destinations can be configured per queue**

Example use case - *“I’d like to receive messages only if there was a change in a particular attribute”*

You can configure 2 queues or 1 queue with 2 destinations, filter by event type and use “changes” filter in one of the 2 destinations



example:

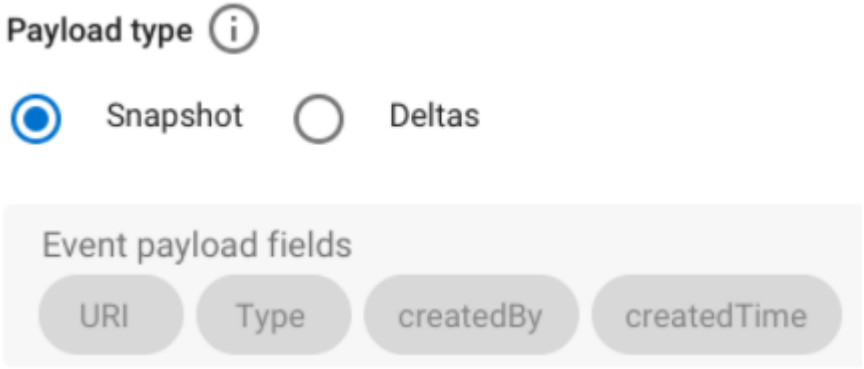
```
filter= changes(configuration/entityTypes/HCP/attributes/FirstName)
```

Payload configuration

Today

- Payload may be configured per individual queue
- Payload configuration is controlled by "`JMSEventsFilteringFields`"
  - `URI`
  - `Type`
  - `createdBy`
  - `createdTime`
  - `updatedBy`
  - `updatedTime`
  - `startDate`
  - `endDate`
  - `Attributes`
  - `Crosswalks`
  - `Tags`
- **Empty "JMSEventsFilteringFields" means everything will be streamed**

starting 21.3



Future

- **Configure destinations per queue in UI**
- **More granular filtering of payload**

Delta event streaming - coming in 21.3 preview

Sample configuration

```
"messaging": {
      "destinations":[
        {
          "payloadType":"DELTAS",
          "type":"queue",
          "provider":"default",
          "name":"env_tenant",
          "dtssQueue":false,
          "enabled":true,
          "format":"JSON"
        },
```

< Sample delta message

```
{
   "type":"ENTITY_CHANGED",
   "uri":"entities/00009ab",
   "deltas":{
      "ovChanged":true,
      "delta":[
          {
            "type":"ATTRIBUTE_CHANGED",
            "attributeType":"configuration/entityTypes/Location/attributes/City",
            "newValue":{
               "value":"ChangedCity",
               "ov":true,
               "id":"8",
               "sources":[
                  "Reltio"
               ]
            },
            "oldValue":{
               "value":"InitialCity",
               "ov":true,
               "id":"8",
               "sources":[
                  "Reltio"
               ]
            }
         }
   }
```