# Architecture

Below is the list of KDDs (Key design decision)

## Data Storage

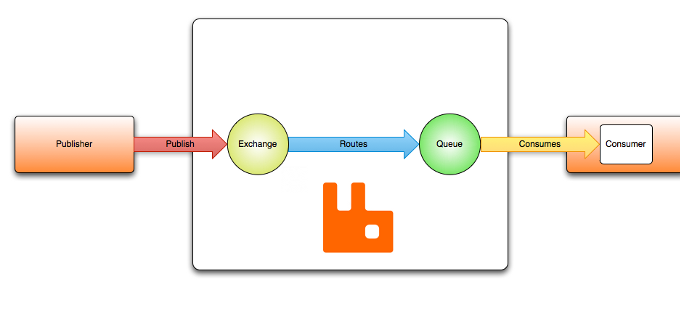
* Must be sufficient resource for database
  + CPU
  + Memory
  + Max connections
* Indexing
  + indexing
* CQRS
  + Recommend to have read and written to separate databases.
    - The requirement of reading and write audit data requirement won’t be the same
    - Demoralised view for readers to increase performance
* Jason
  + Store JSON for object payload so that that audit service can be generated. The application can deserialise back.
* Decuple from service
  + Using either actor or message model to decouple database from service so that that service won’t be heavily I/O bound

## Middle layer

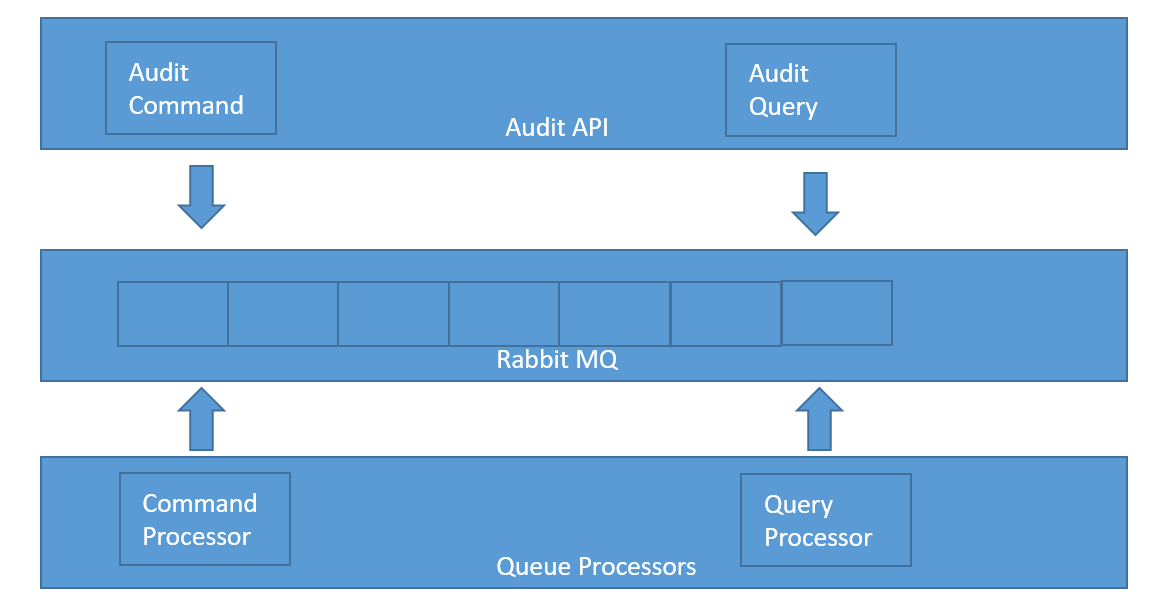
* Use either a message queue or actor model design to fulfil to handle required load.
  + Because audit service will be more close to pure CRUD based, so decided to use message base RabbitMq instead of Akka.net(actor model).
  + Use different queues and control different policy between reading and writing.

## Data Ingestion

* Requests will be published to a message queue, and there will be a subscriber to process them separately



## Overall architecture



# Data Model

|  |  |
| --- | --- |
| Audit Event |  |
| Id | Primary key |
| event\_name | Name of the event |
| event\_unique\_id | Unique id for event |
| event\_source | Which service or which client it is coming from |
| event\_time | Time event happened |
| request\_origin | Which uri is coming from |
| request\_action | POST, GET, etc |
| request\_headers | Headers in a string |
| user\_id | User id |
| Created | Datetime row is created |
| Update | Update timestamp |

|  |  |
| --- | --- |
| Audit Event Payload |  |
| Id | Primary key |
| request\_payload\_type | Payload type name |
| request\_payload\_content | Json string for request payload |
| event\_id | FKfor audit event |
| Created | Datetime row is created |
| Update | Update timestamp |