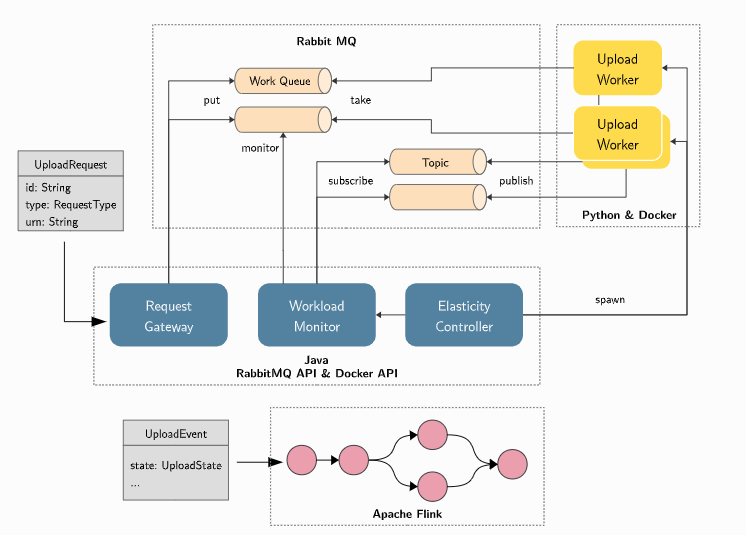
**RabbitMQ Assignment**

RabbitMQ is a robust and popular message broker, for routing messages asynchronously between systems and apps. There is a hypothetical business called OBB Corp.

**Contextual Use Case:** Content Management App

Course administrators want to upload different materials such as quizzes, documents (e.g., PDFs), or videos. Each material type requires a different type of processing for upload and storage. For example, quizzes are first transformed into JSON documents and then inserted in the Document DB, documents like PDFs are scanned using OCR software to prepare them for full-text indexing, and videos may have to be transcoded in a different format. This part of the system is faced with serious scalability challenges in the face of a growing user base and varying workloads at different weekdays and times. Furthermore, system administrators want to be alerted about problems in the pipeline.



Uploading a material is represented as an UploadRequest, a simple message object that contains the material type and the URN of the material. We will not worry about how specifically the material data gets into the system - there are many ways of solving this, instead we focus on the process and the pipeline. For each type of material, there can be zero or more workers that handle this type. Note that this is a classic producer/consumer problem, where synchronization is commonly solved via queues. For each worker type there is one queue that holds the upload requests of the respective type. The request gateway receives the upload requests and routes them to the correct queue. The workers take items from the queue, process them, and then report back to the system by publishing data to a topic associated with the material type. A monitoring component monitors the amount of requests in a queue and also uses the published data by the workers to calculate the average time it took to process requests. This data is used by the elasticity controller that dynamically spawns or removes upload workers based on current demand. The components emit system events that are fed into the stream processing platform that creates alerts when it detects problems in the pipeline.

Note that we will not to actually integrate the system end-to-end (although if you implement all components, it’s possible), but instead we will focus on the technologies to implement the individual components.

**Todo’s:**

1. Download and install RabbitMQ from <https://www.rabbitmq.com/download.html>
2. RabbitMQ is built on Erlang runtime environment so before installing RabbitMQ, you may need to download and install Erlang from <http://www.erlang.org/downloads>
3. After installing RabbitMQ, open RabbitMQ Command Prompt and enable Web Management Plugin
4. For OBB Corp's content management app, you will use the RabbitMQ Java API.
5. Create all necessary queues and exchanges required by the system. Implement the IQueueManager and put your implementation in the impl package. The QueueManager is used in the test cases to create and tear down your queues and exchanges. The easiest way to implement it is to run your QueueManager code and verify the effects in the RabbitMQ admin interface.