# Akka

Akka is a set of open-source libraries for designing scalable, resilient systems that span multiple nodes. There is both an Actor framework and a Streams framework. I evaluated Akka as an Actor framework only. While the Streams functionality may be a closer fit for the scenario overall, a more common approach would be to use Spark for streaming.

## 1.1.0 Coverage

Akka does assist with a few hard problems, but not everything. This tables shows the areas that Akka Actors can address:

|  |  |  |
| --- | --- | --- |
|  | Work | AKKA? |
| **Router** | | |
| Determine partitions it is reading from. | If Kafka, this is handled by Kafka. If Cosmos, the app needs to divide them up. | No |
| Receive messages. | Messages are pushed or pulled from the appropriate partitions. | No |
| Checkpoint. | Record the offset of the last message read. | No |
| Dispatch messages to appropriate actor. | The appropriate messages are dispatched to the appropriate actors for processing. | Yes |
| **Actor (Aircraft, Airport, etc.)** | | |
| State is hydrated. |  |  |
| Receive messages. | Messages streamed to the actor need be received and buffered for the next step. | Yes |
| Put messages into a window. | The messages will be stored in a time-bound window (T-10 minutes or similar) whereby they are ordered and de-duped. | No |
| Process messages. | Messages are processed as appropriate. | No |
| State is updated. |  |  |
| Materialized views are updated. |  |  |

# 2.0.0 Delivery Guarantees

Akka supports at-most-once delivery (“tell” method). This means that the delivery of a message is not guaranteed but it will not be sent more than once.

If you add persistence, you can change the delivery to at-least-once (“deliver” method). This means that the delivery of a message is guaranteed, but it could be sent/received multiple times.

There are a couple of approaches that could be used to ensure that messages are received, if you didn’t implement persistence. Personally, I am not fond of implementing persistence for this use-case because the original messages are already persisted upstream so it doesn’t seem worth it to create another store.

## 2.1.0 Receipt

The Actor could send receipts back to the Router. If a Router didn’t get a receipt after a specified period of time, it could resend the message.

|  |  |
| --- | --- |
| https://cdn-images-1.medium.com/max/1600/1*B4wBCGf-A1gdmUbFlvjKAA.png | PROS:   * Very easy to implement. * Easily discard duplicates because of the unique ID.   CONS:   * Doubles the number of messages. * The messages must somehow be tagged with a unique ID. |

## 2.2.0 Checkpoint

Rather than sending receipts for every message, the Actor could send a checkpoint every so often saying what messages it had received. The Router could resend any messages that had not be received after a timeout.

|  |  |
| --- | --- |
|  | PROS:   * Relatively easy to implement. * Easily discard duplicates because of the unique ID.   CONS:   * The messages must somehow be tagged with a unique ID. |

## 2.3.0 Ask

There is an “ask” pattern for Akka that listens for a response.

|  |  |
| --- | --- |
| https://cdn-images-1.medium.com/max/1600/1*TbXTtwuIiXxgeWo5UkR_FA.png | PROS:   * Easy to implement.   CONS:   * Creates a “listener” actor for every single ask. |

This article is pretty good at covering “ask” and the associated patterns and anti-patterns:

<https://medium.com/@yuriigorbylov/akka-ask-antipattern-8361e9698b20>

# 3.0.0 Mailbox

Akka supports a number of different mailbox types. It uses UnboundedMailbox by default which will continue to accept messages until it gets an out-of-memory error, other options are available, but realize that there is always a limit to how many messages can be buffered.

# 4.0.0 Message Window

In Kafka and Cosmos (as with most other partitioned systems) the order of messages is only guaranteed within a partition, therefore whenever a partition is crossed the messages could be out-of-order. In addition, Akka only guarantees the order of messages sent from a specific actor to another specific actor, or in our terms from a specific Router to a specific Actor (Aircraft, Airport, etc.). All this is to say, the message order is not guaranteed to be accurate.

You will need to develop a “message window” capability that does the following:

* Buffers messages for a period of time (T-10 minutes or something).
* Orders the messages in the buffer.
* Removes duplicate messages.
* Messages received that are outside the window should be discarded – there is no longer a legitimate way to process them.
* Expires messages in order and passes them along to the Actor for processing.

The buffer window should be considered the lag time for the Actor to process messages. In other words, if you have a T-10 minutes window, the Actor will be 10 minutes behind.

Note that there are some capabilities in Akka Streams that can handle this functionality (albeit a bit differently than I have described here):

* Bounded Ordering: <https://squbs.readthedocs.io/en/latest/flow-ordering/>
* Deduplicate: <https://squbs.readthedocs.io/en/latest/deduplicate/>