

- Data Passing and State Management I need to make several decisions about aggregation and correlation:
 - How should I track jobs through the system in such a way as to make aggregation possible.
 - Should I track jobs and trigger correlation events at the end, or should I simply trigger such events on a regular interval if the system has changed?
 - Do I need to track anything other than ancestor information?
 - Should I trigger correlation events off based on queue behavior, or data lake updates?

Each decision I make will have some tradeoffs that I'll need to consider within the context of local and cloud, small organization and large enterprise. Can these use cases be balanced?

- Reusable Architecture and Appropriate Abstraction Transitioning a pipeline from local to cloud should be (almost) transparent. Architecting in such a way as to make that happen will mean being very intentional about how components are isolated and what the plugin architecture looks like.
- Scaling, Retry Logic, and Rate Limiting Workers are pulling from DockerHub, which implements rate limiting. Since rate limiting is a common challenge, I won't try to mitigate it by setting up a docker proxy. Rather, I'll need to figure out a way to manage this. Rate limiting will limit the value of multiple pulling threads. If I choose to auto-scale, I will need to use a different metric than Queue depth or CPU.

I'll need to figure out how best, if at all, to communicate rate limits to workers.

- Idempotency While less important for local execution, networking can significantly increase the likelihood of duplicate queue entries. Unlike Queues, Pub/Sub has no build-in mechanism to determine that one job is taken by one worker. In both the case of the buggy Queue and the case of Pub/Sub, I'll need to make sure actions are only carried out once.

This becomes most important for vulnerability enrichment, where data can be gathered in parallel and need not block other enrichment actions.

- Missed work items Workers should be expected to crash occasionally, for any reason. The system must be left in a good state after a worker crash. A missed work item could be a missed security signal. I will probably need to manage this with a DLQ in the cloud situation. Locally crashes will lead to thread locking, meaning that my framework will need to implement some kind of retry logic within a try/except/finally block.
- Threat Model I will need to threat model the application during the design phase to understand the security implications of each decision.

- **Data Models and Developer Experience** A developer implementing a worker will need to a way to communicate the pattern (Queue, Pub/Sub, Agg) and the processing mode (serial or batch). For patterns, they'll need to communicate what types of messages their worker can consume, what it outputs, and how it runs.

I need to determine if I can use a similar data model for the data lake as for messages, or if I'll need a different system.

- **Resource Contention** The data MVP data lake will be flat files on local systems, and probably an s3 compatibility layer (like garage) in the cloud. I will need to implement a resource abstraction layer that manages resource contention in both contexts.