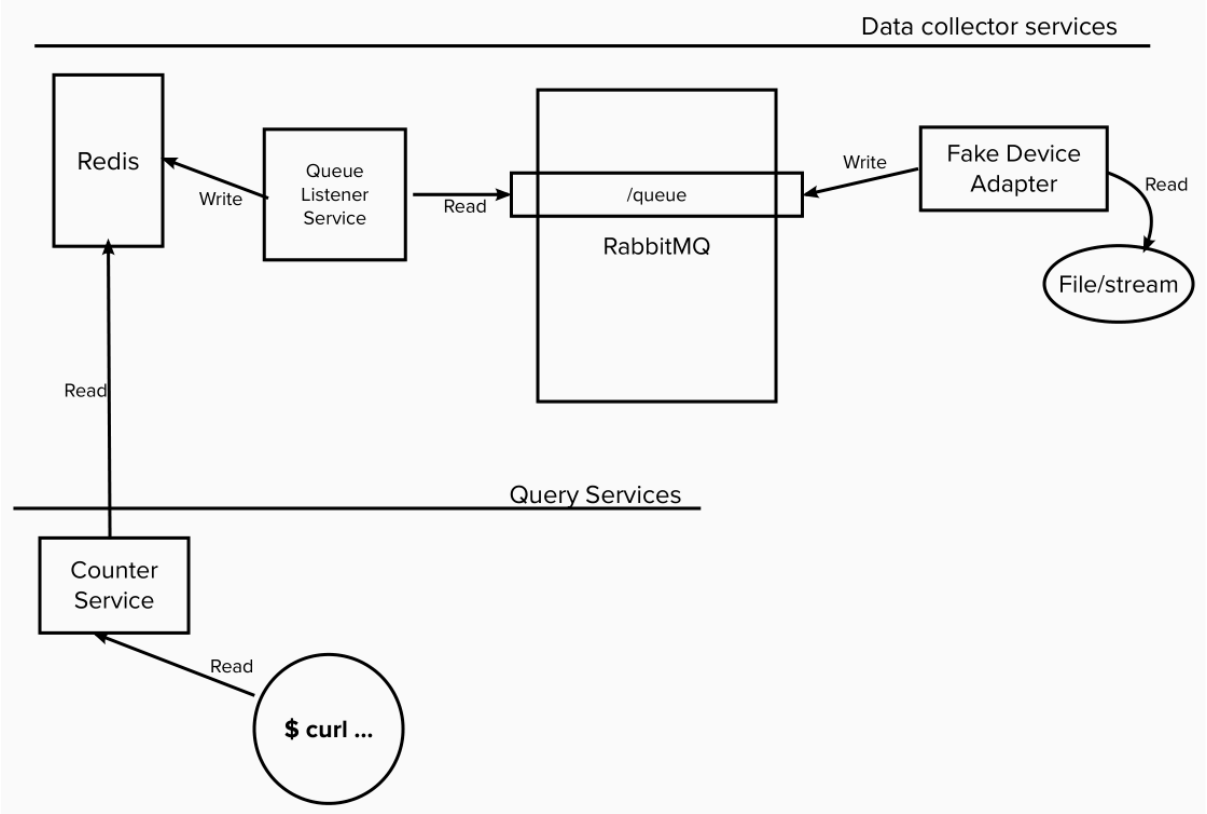
DDC - Proof Of Concept

1. Definition

In this document the implementation of PoC will be defined. Opposed to the architecture document which gives a conceptual overview, here, protocols, data formats and concrete products will be defined.

2. Concrete Architecture



2.1 Products to Use

2.1.1 RabbitMQ

RabbitMQ will be used as the message broker in DDC. It's a stable product which has been used extensively in the industry and implements AMQP protocol. It's proven and robust server is accompanied by many client libraries as well as a wealth of utilities. RabbitMQ could be configured for different brokerage usages, we need its queue machinery.

2.1.2 Redis

Redis will be used as the scalable data store in DDC.It's a robust product which has proven its maturity in the industry. Although it lacks strong consistency and durability guarantees, it is very fast for our storage and query needs. As the measurement counts is our business need, it's decided to be no mission critical and no need to be stored in a relational DB. Redis

shines when a simple key value is needed in load balanced fashion while synchronous durability is not vital.

2.1.3 Curl Command Line

As PoC is not targeting a UI implementation, curl will serve our needs to query our pipeline. Curl is an ubiquitous utility for command line HTTP tasks. Here we'll hit the Counter's endpoint to get the result in raw JSON hence test the whole line.

2.2 Services to Implement

2.2.1 Fake Device Adapter

As mentioned in the architecture document we need adapter services to feed/adapt measurements from devices to DDC. To get rid of a concrete device dependency and ease testing of the full pipeline, a fake device adapter will be implemented. This service adapter daemon is fed arbitrary data from file or data stream which could be utilized for programmable inputs. A script could be used to feed the Fake Adapter Service with programmed measurements, so the solution is easily composable with different data density and types.

2.2.2 Queue Listener Service

Queue Listener Service reads the queue at RabbitMQ and writes devices' count data to Redis. As the only query service we support is providing total measurements received per device, we save the data in a supporting format. For this purpose we define a common key prefix which will prefix all keys concatenated by device ID, and value as simple integers which increase per measurement of that device.

2.2.3 Counter Service

Counter Service will provide a REST endpoint (http://<host>/measurements) to provide each devices' total measurements. The JSON will be a simple dictionary where keys are device IDs while values are integers which correspond to total measurements received from that device. As Queue Listener incremented/saved into key names with common prefix, Counter will first list all keys with that prefix and get those values with those keys.