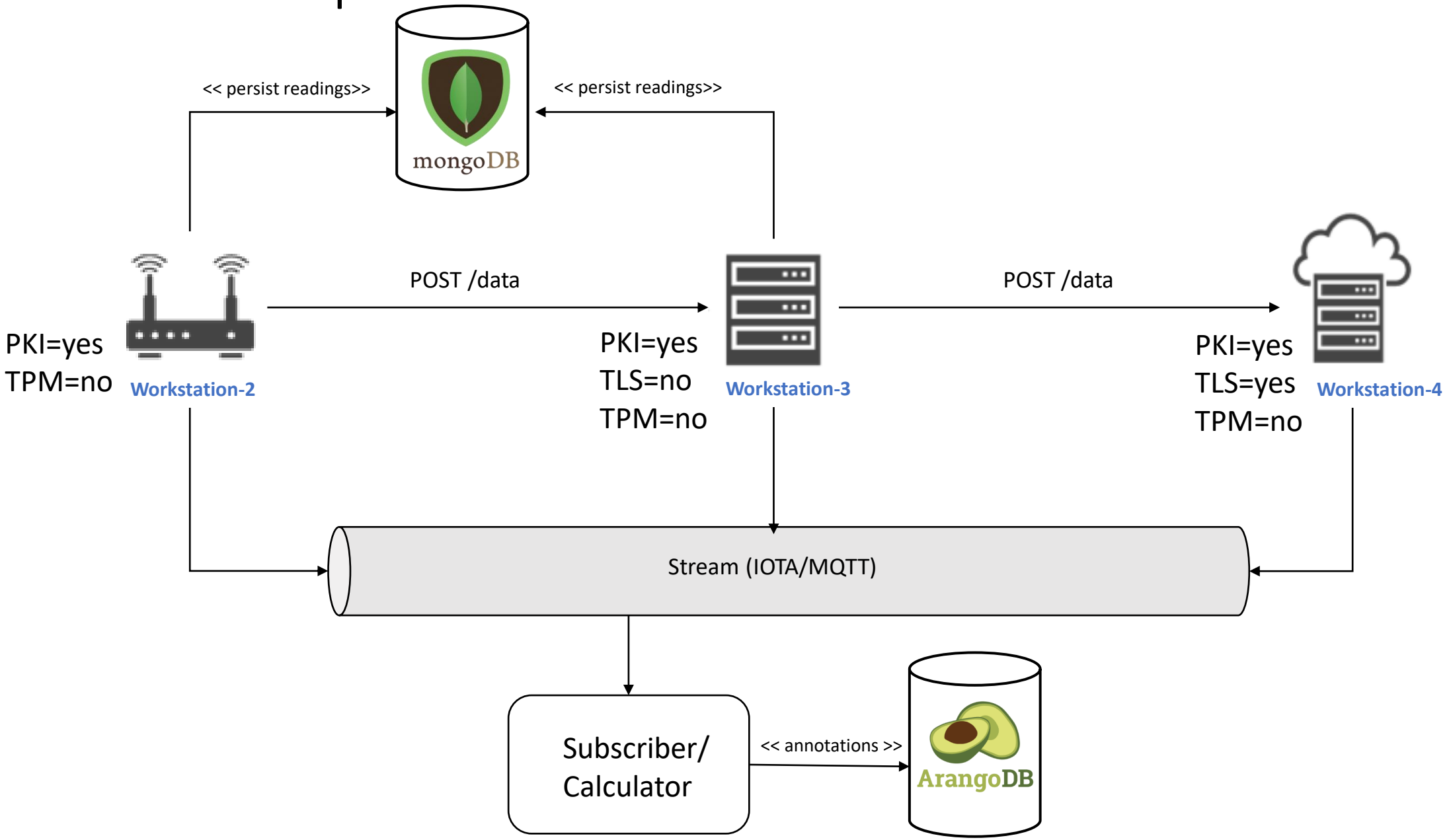
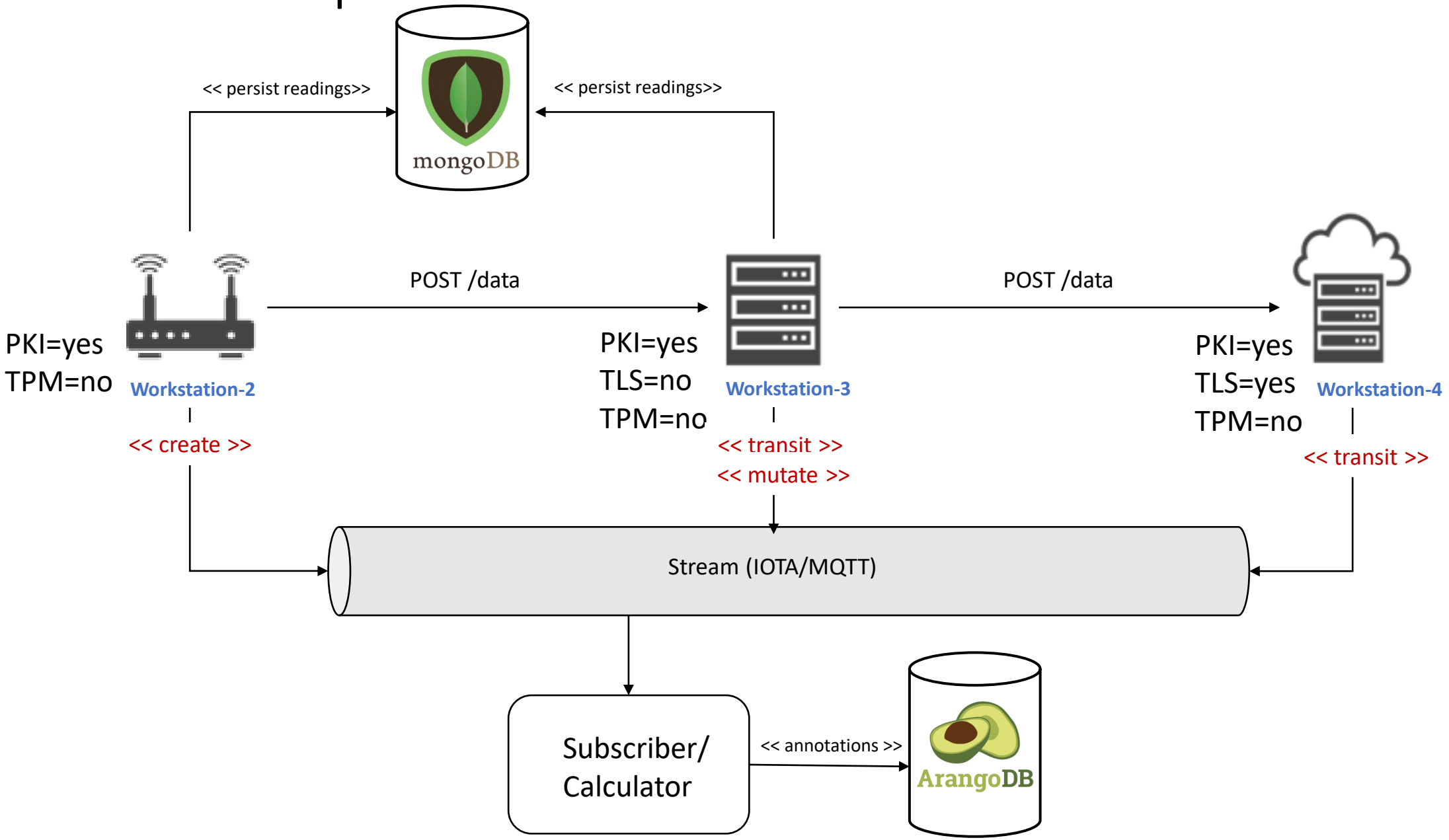
Data Confidence Fabric

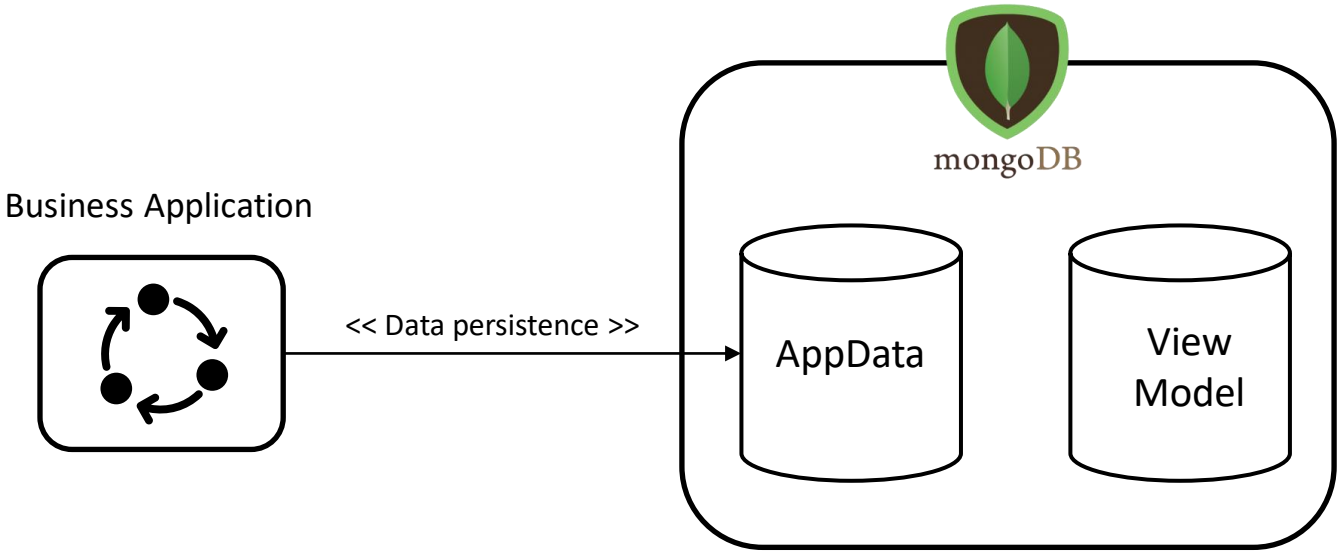
Scoring Methodology

Demo Setup

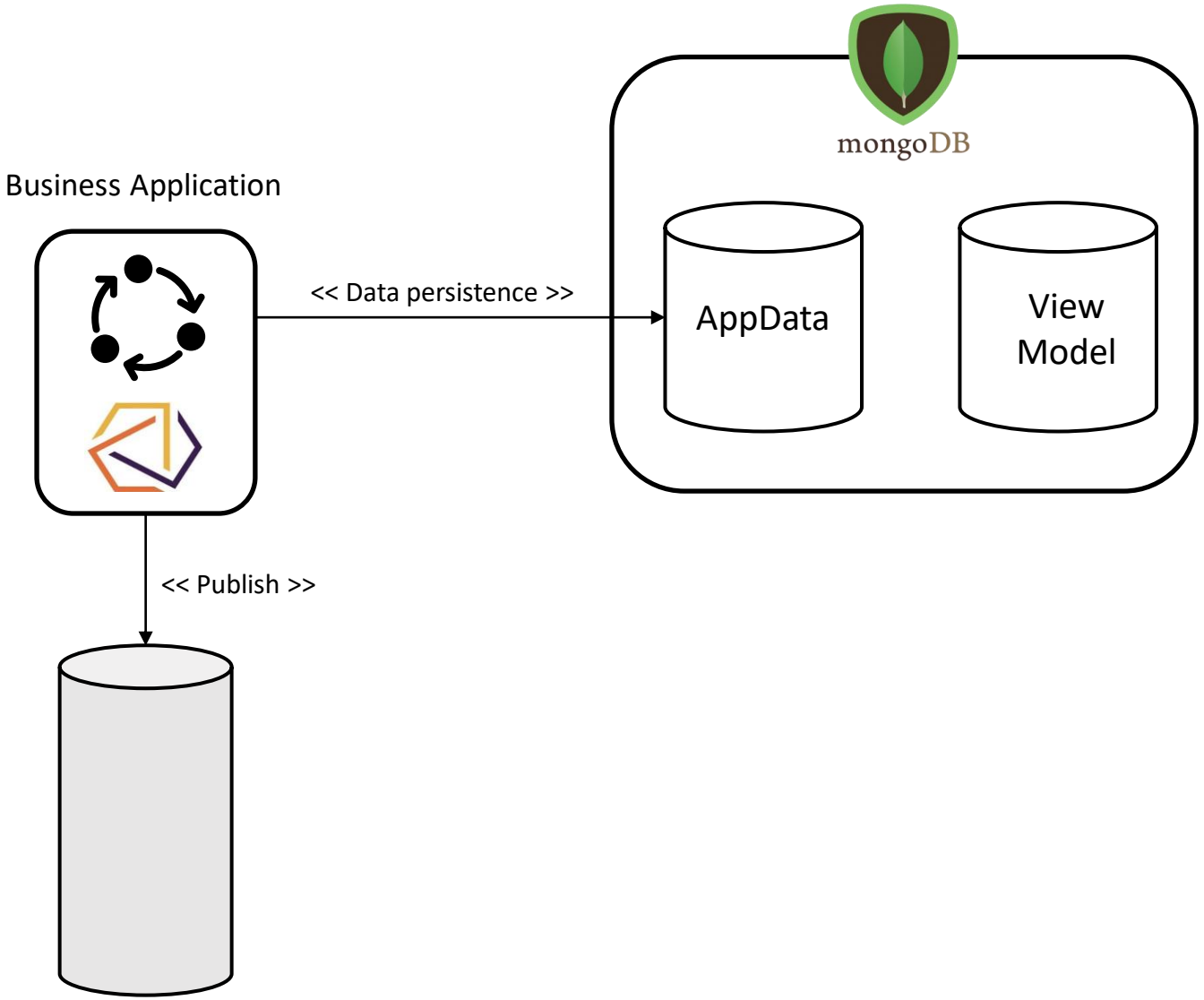

<< persist readings>>
<< persist readings>>
mongoDB
POST /data
POST /data
PKI=yes
TPM=no
Workstation-2
PKI=yes
TLS=no
TPM=no
Workstation-3
PKI=yes
TLS=yes
TPM=no
Workstation-4
Stream (IOTA/MQTT)
Subscriber/
Calculator
<< annotations >>
ArangoDB

Demo Setup

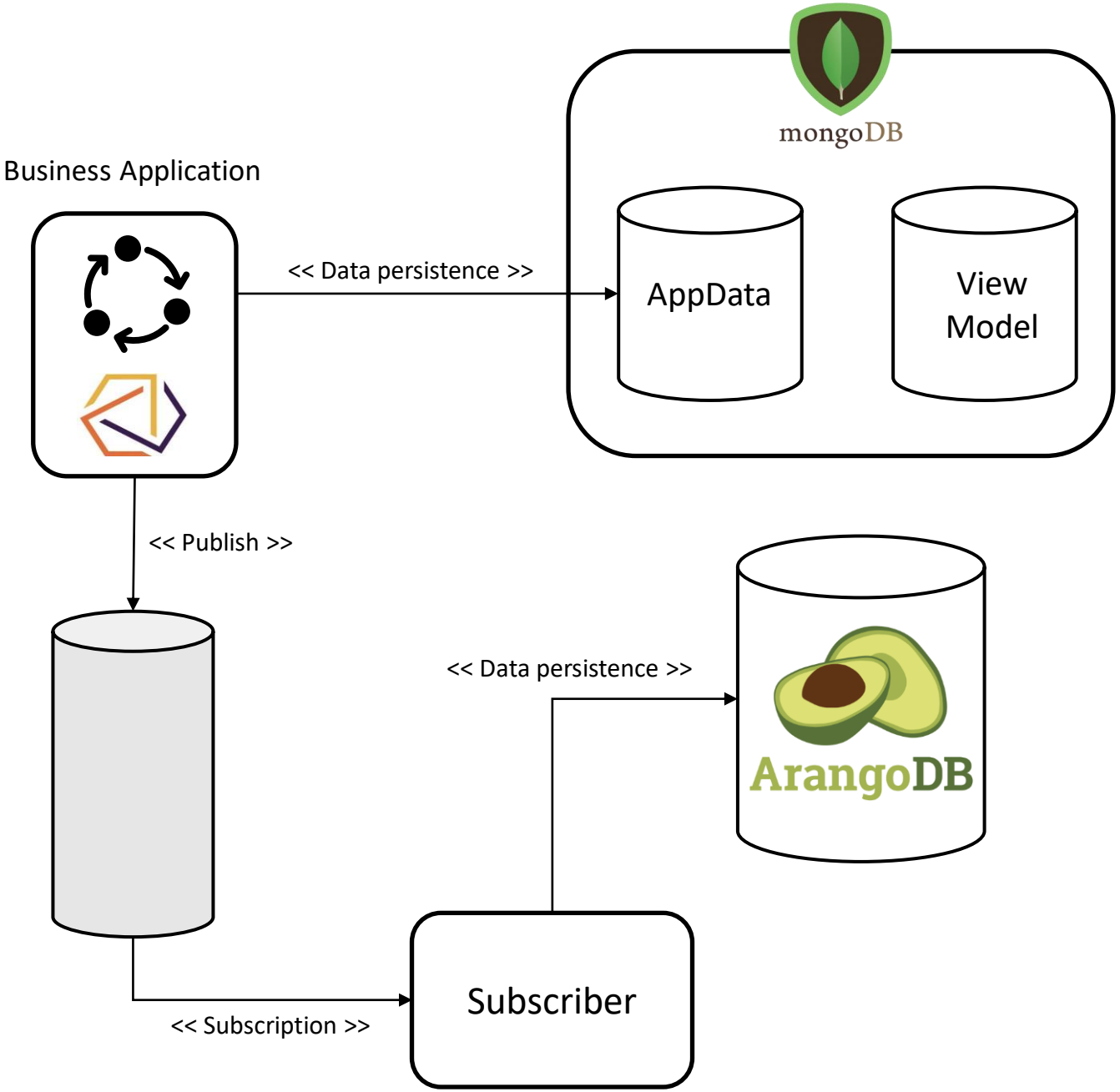

<< persist readings>>
<< persist readings>>
mongoDB
PKI=yes
TPM=no
Workstation-2
POST /data
PKI=yes
TLS=no
TPM=no
Workstation-3
POST /data
PKI=yes
TLS=yes
TPM=no
Workstation-4
<< create >>
<< transit >>
<< mutate >>
<< transit >>
Stream (IOTA/MQTT)
Subscriber/
Calculator
<< annotations >>
ArangoDB



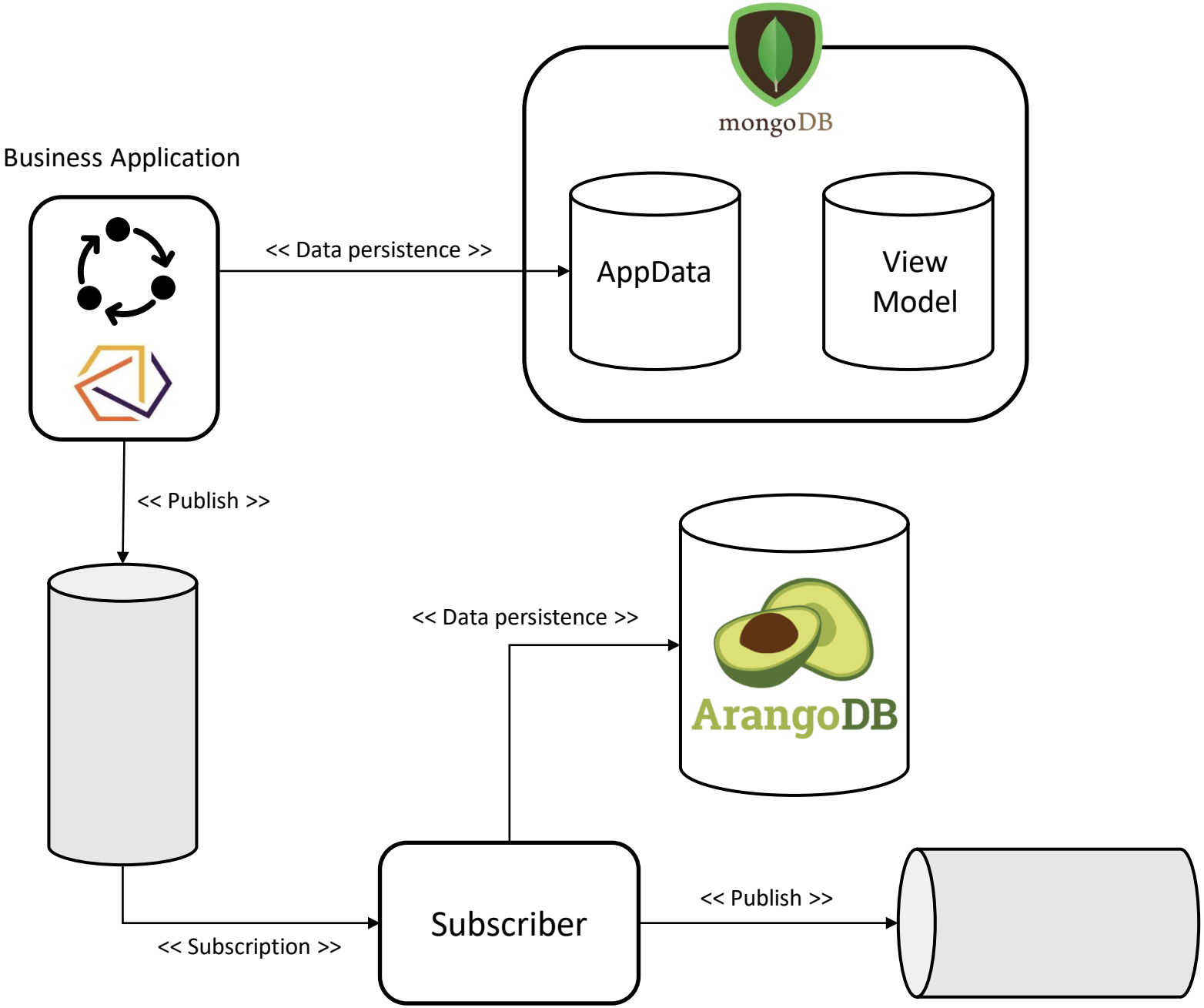
The business application creates and/or handles data via the application data path, and persists it in a database. This data is separate from anything related to confidence scoring.



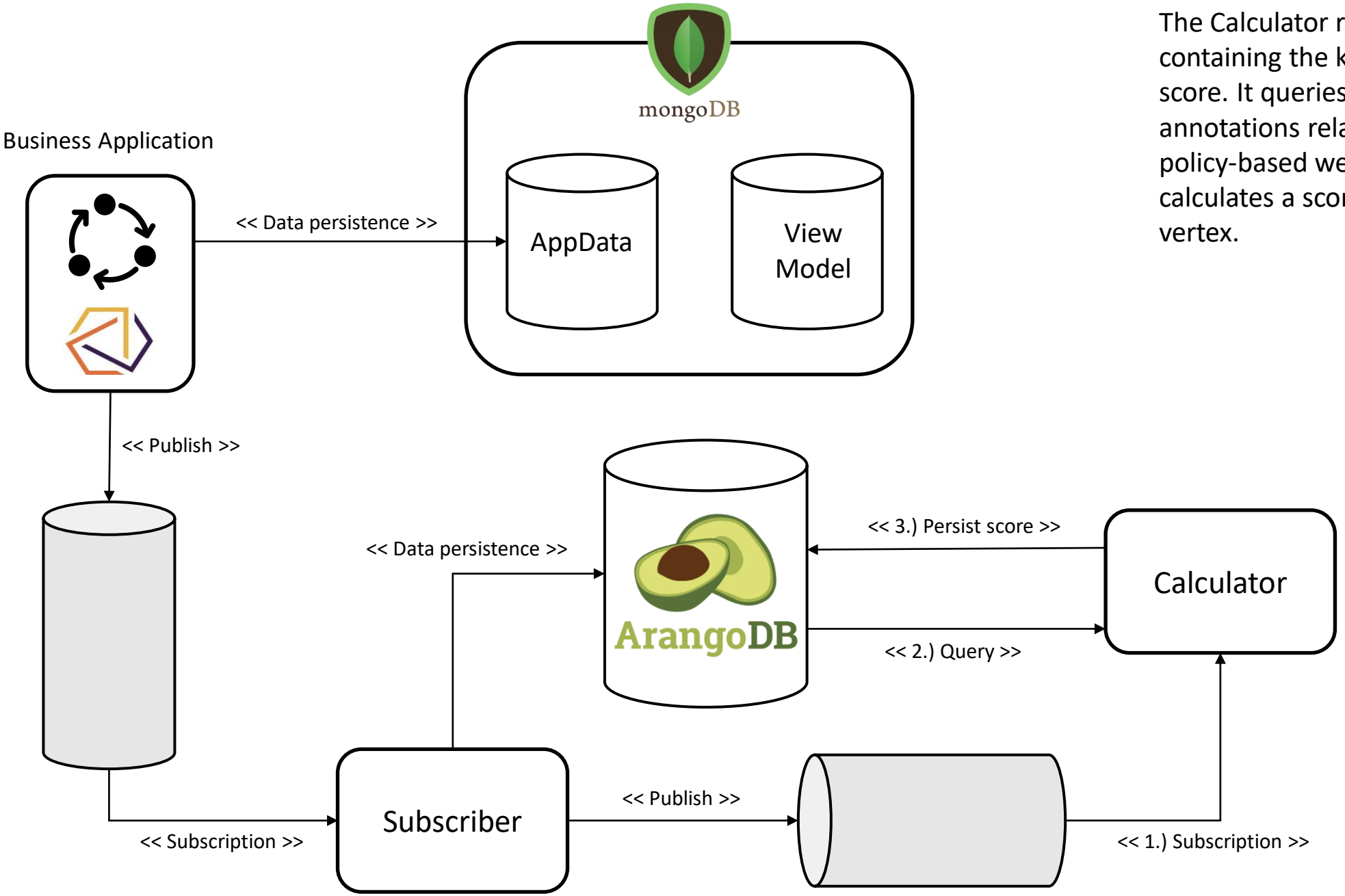
DCF-enabled applications can annotate data from the application's context and publish the annotation metadata to a stream.



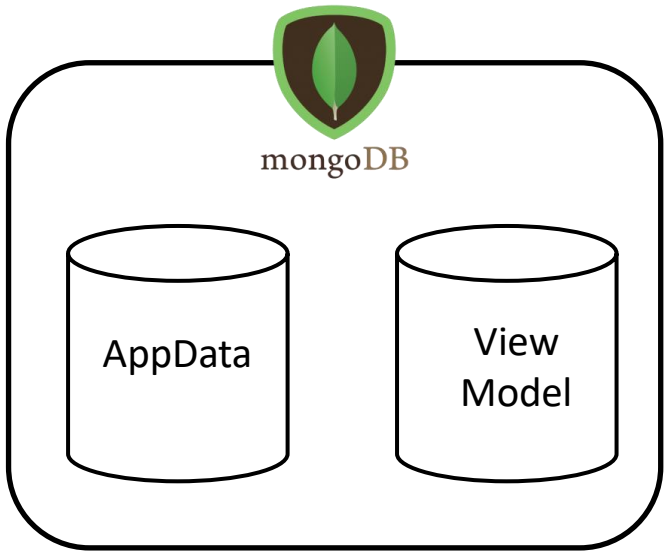
A subscriber receives the annotation metadata from the stream and constructs the relationships supporting data lineage and scoring. These are persisted in a graph database.



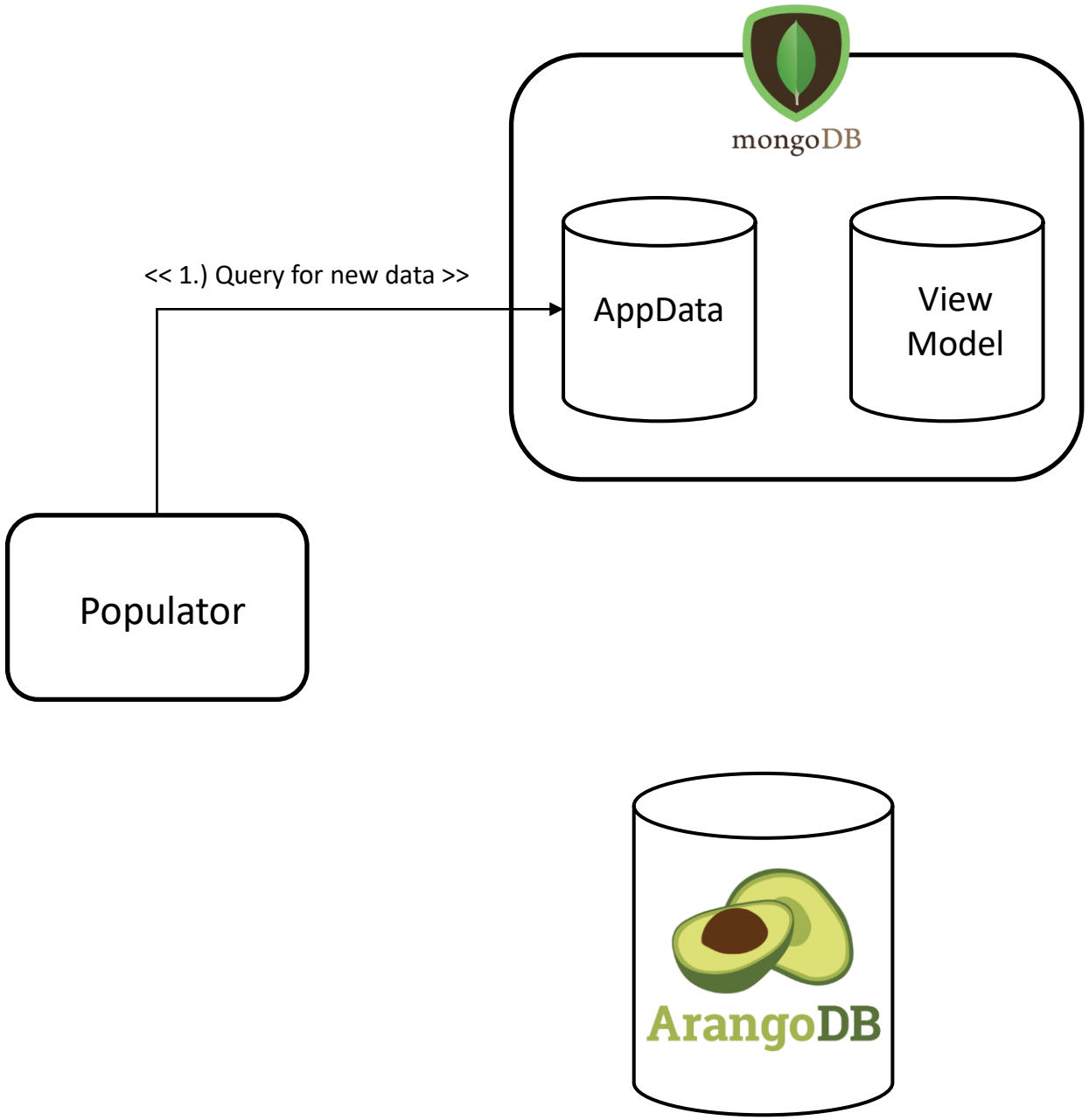
Having persisted the annotation relationships as a graph, the Subscriber itself publishes an event containing the data key. This is the hash that identifies a specific data element and indicates the data is ready for scoring.



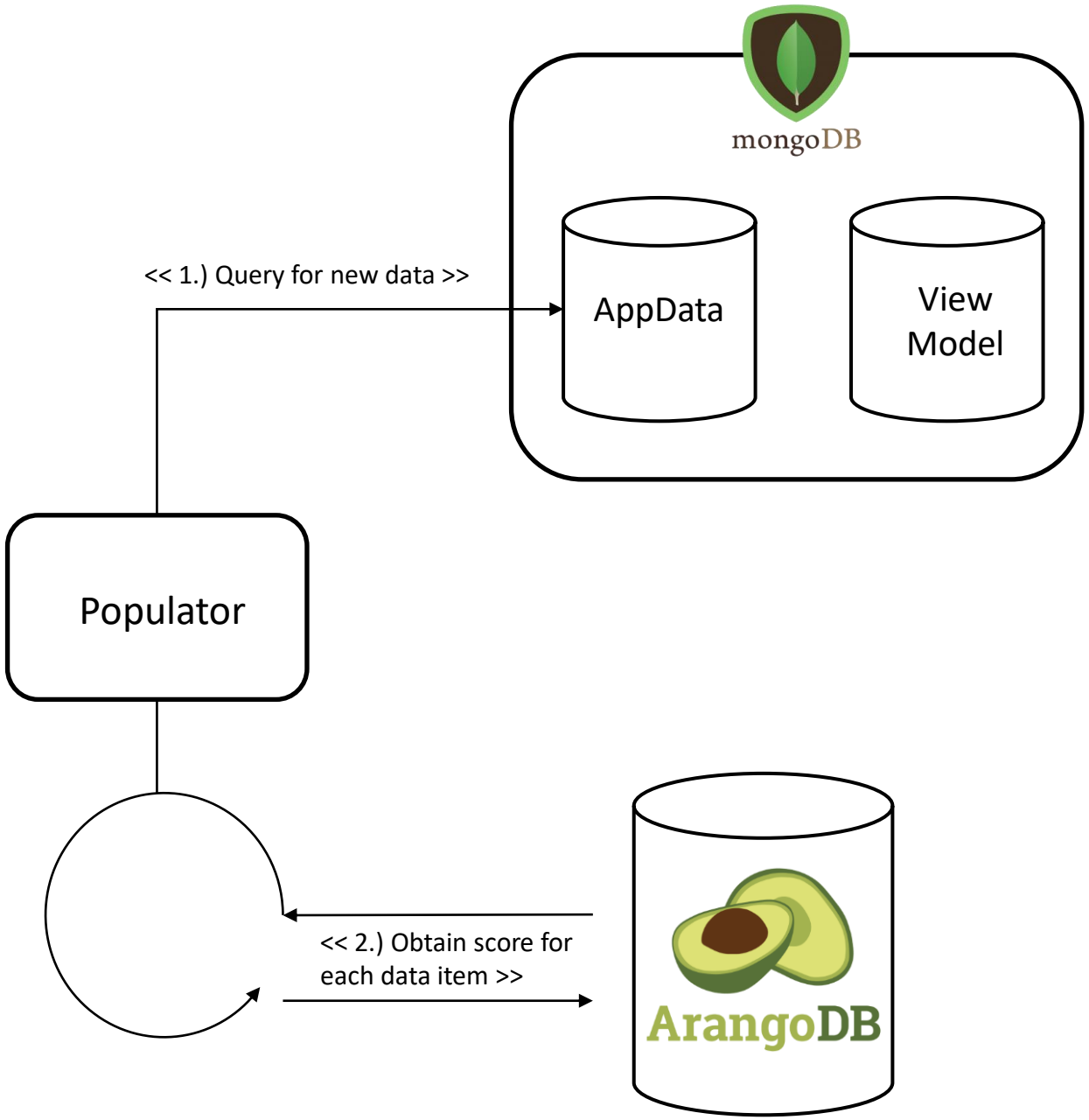
The Calculator receives the event containing the key of the data we need to score. It queries the graph for all annotations related to that key, applies policy-based weighting to the annotations, calculates a score and persists it as a vertex.



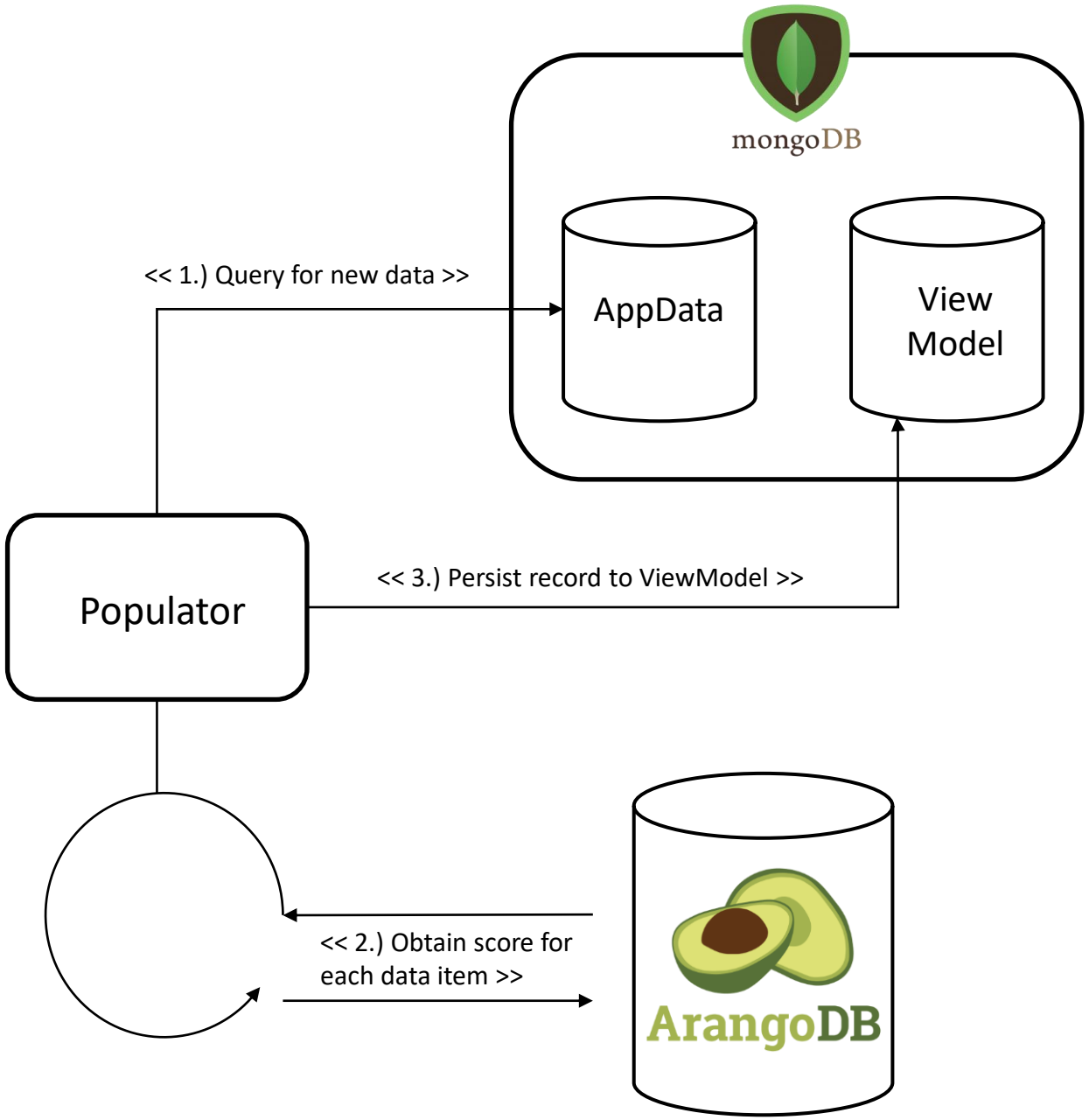
Having calculated the score, we have to now provide a unified view that brings together the application data and the score. This “View Model” facilitates ease of querying without having to go to multiple services in real-time while the user is waiting in order to join all of the data together.



The Populator runs a loop that does three things. First, it looks up any new data in the application database that needs scoring.



The next step is that for each new data record, the Populator will query the graph to find the calculated score. Operation should be such that if no score is found, the Populator should retry on its next iteration cycle. It can take a few seconds for the score to calculate as annotations are collected from the network.



Finally, assuming a score is found, the Populator is able to persist a ViewModel of the data that includes the score and will facilitate ease of querying. This does assume some knowledge of how the ViewModel is to be used (dashboard, report, etc…)

Once all of that is done, an API can host endpoints for retrieving data from the ViewModel collection and providing it to the appropriate UI.