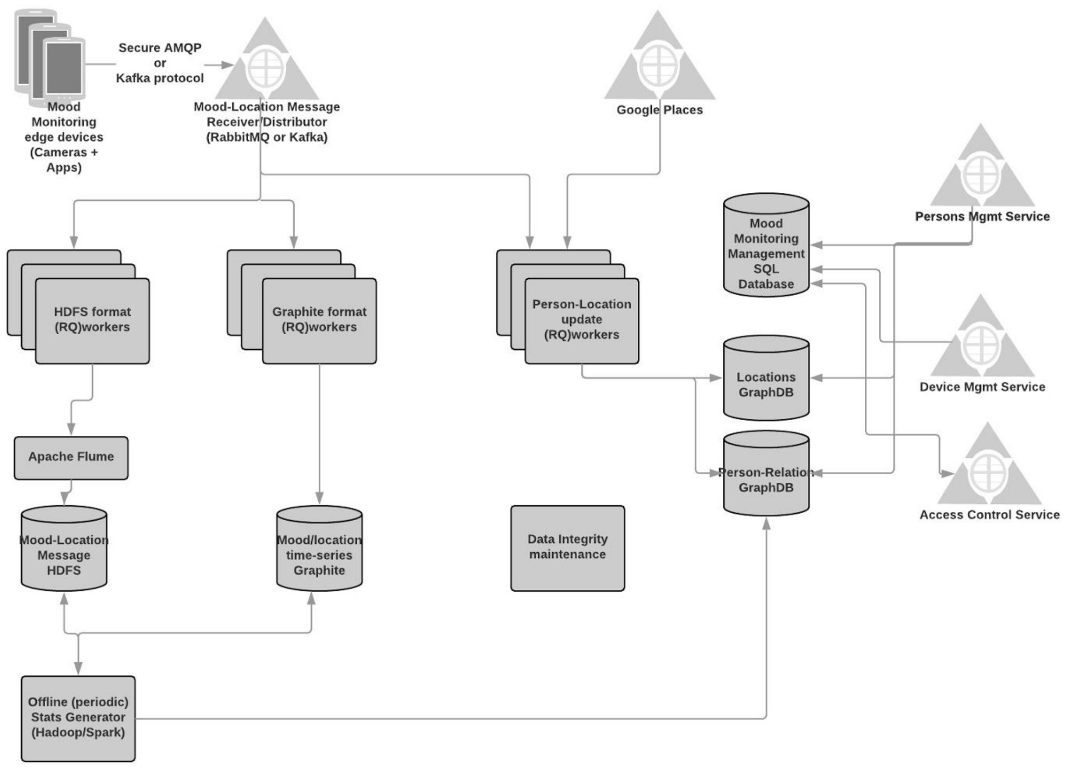
Secure AMQP or Kafka protocol
Mood Monitoring edge devices (Cameras + Apps)
Mood-Location Message Receiver/Distributor (RabbitMQ or Kafka)
Google Places
Persons Mgmt Service
Mood Monitoring Management SQL Database
HDFS format (RQ)workers
Graphite format (RQ)workers
Person-Location update (RQ)workers
Locations GraphDB
Device Mgmt Service
Apache Flume
Person-Relation GraphDB
Access Control Service
Mood-Location Message HDFS
Mood/location time-series Graphite
Data Integrity maintenance
Offline (periodic) Stats Generator (Hadoop/Spark)

Overview:

Apps on smartphones asynchronously photograph the device owners and assess the owners mood. The app generates a message {person_id, mood, location} and publishes it (secure AMQP or Kafka) to the enterprise. The Enterprise RabbitMQ or Kafka server publishes the message into at least three channels: HDFS, Graphite, and Person-Location.

RQ (Redis Queue) workers pull messages out of the channels and massage them for the various databases.

Various datastore technologies are used for differing purpose:

HDFS is used as a "raw" message storage. "Long period" Statistics are calculated by the Offline Stats Generator using HDFS/Pig/Hadoop or Spark. Some of the statistics generated are then updated to the individual person records in the SQL database and/or Person/Relation graph database. Other statistics can be rearchived into HDFS and/or added as aggregate time series to the "Short Term Mood" Graphite database.

The Graphite system consists of Carbon, a data collector, Whisper, a round-robin database, and Django webapp for graphical display. The Graphite format workers, format the mood/location messages to append in the appropriate time series. As a round-robin database, old data is dropped after a given amount of time. As a first guess I would choose 30 days. Webservices can access the Graphite system via the Django webapp.

The Person-Location (PL) update workers update the Person-Relation and Locations Graph databases. One hour or 24 hour mood and location data is attached to the Person-Relation node. The PL workers also query the Google places for location/distance information to “Public” locations which haven’t been queried before. This data is added to the Locations Graph DB.

I think it is obvious why I would choose to implement locations on a Graph DB because I want to minimize the number queries to Google places going forward. Geographical locations are not going to change from day-to-day and the types of public locations nearby will also change slowly. However, it is less obvious why persons (app users) should be in a graph db. I decided to do this because I believe the next version of the mood location system would want to consider distance from relations (parents, children, co-workers,...) . As a human being, I think those are actually more likely to affect a person’s mood than say, distance to the grocery store.

The Mood Monitoring Management SQL Database is for managing meta-information about the persons, edge devices and edge device apps. The User facing webservices heavily depend upon this SQL database.

The Persons Mgmt Service provides webservices to manage person information in the system. This includes app users, their relations, and all human entered locations.
The Device Mgmt Service provides similarly for the user device and apps. The edge devices will need to periodically reauthorize themselves, upgrade the app, etc. Also, as new devices come onto the market, then the app should be tested and information added through this webservice.

The Access Control Service should be used by any process or service on the edge or when requested through a human interface for authentication or authorization.

Commentary:

App and Edge Device:

1. I’m assuming that the edge device and RabbitMQ/Kafka can execute some sort of HTTPS or SSL like protocol to set up a secure connection. I need to research this.
2. If it is a safe assumption that the edge devices will not be sending data too frequently (since it is collected by the camera app), then it may be safe to replace the direct secure channel with a HTTPS web service. The edge device could supply a userid/secret password which was installed during the first manual login, or the user could log into the app at every use.

3. App Self Test and App Back Haul test.

Mood-Location Message Receiver/Distributor (RabbitMQ or Kafka):

1. Kafka has the advantage of a round-robin database. Message remain available until some date when they are deleted. RabbitMQ require subscriber management.
2. RabbitMQ has clients with rich functionality. I'm not as certain about Kafka.

RQ workers:

1. Self-scaling: If a daemon is added to monitor channel depth and workers can be spun up or allowed to terminate as needed.
2. I would endeavor to share a single docker or vm image for deploy of all RQ workers. This will decrease the maintenance for RQ workers. Also it would enable creating a pool "hosts" where any RQ worker could be spun up.

HDFS Data:
By formatting the message stream into HDFS, there are benefits:

1. Parallelized generation of statistics offline.
2. Generation of sanitized test datasets.
3. Disaster recovery replay of messages

Time-series database, Graphite:

1. The Graphite Webapp provides time series graphs api.

Graph Databases:

1. Locations are proximate to locations so it makes sense to model locations in a graph database.
2. Mood is probably more connected to the combination of personal relationship and location, so modeling person in a graph database is probably better in the future. I've created the schema for the node with this in mind.
3. There are privacy implications which affect the number of nodes: suppose a person always meets their friend/relative at a particular coffee shop. The friend might not want the person to know they have the app. The person could enter the friend as having a relationship and the coffee shop the location. This should cause a new person node to be created. In order to remove the duplicate node we would need some sort of mutually acknowledged "friendship" and privacy settings.

Data Integrity Maintenance:

1. In time, the number and type of nearby relevant locations changes. These changes need to be incorporated periodically.
2. Once in a while data will not be back-hauled to the enterprise for whatever reason. In this case a mechanism needs to be created to get the edge device to resend it's data. The manner in which this is accomplished has to be such that "data storms" are not created as in the case a significant number of edge devices were "offline"

simultaneously. I don't have a solution here, but I envision that this process would play a role in it.

Persons Mgmt Service/Device Mgmt Service API:

1. User Account: Login, Password change, New user registration
2. App Download, App Setup
3. Privacy settings.

Project:

Each block component pictured is a project. As they are numerous I recommend the following directory hierarchy:

<project>

- <VM Image>
    - Configuration file or compressed image file
    - Deployment script
- Src
    - Directory hierarchy per the project
- Test
    - Unit tests
    - Integration tests