(A) The first service is responsible for receiving analytic data through a web interface (REST, WebSockets, Comet), encoding the payload using an efficient serialization format (Avro, Thrift, Protobuffers). It's second responsibility is to publish the incoming analytic data (with minimal manipulation of the data) to the data feed (**F**)

(B) This receiver is responsible for warehousing all analytic data coming in to the system, it does this by subscribing and ingesting from the data feed (**F**) and updating the database (**G**)

(C) This emitter is responsible for re-emitting historical analytic data to the data feed (**F**) by first reading it from the database (**G**). The emitted data goes to a separate low priority topic. This process would be triggered manually.

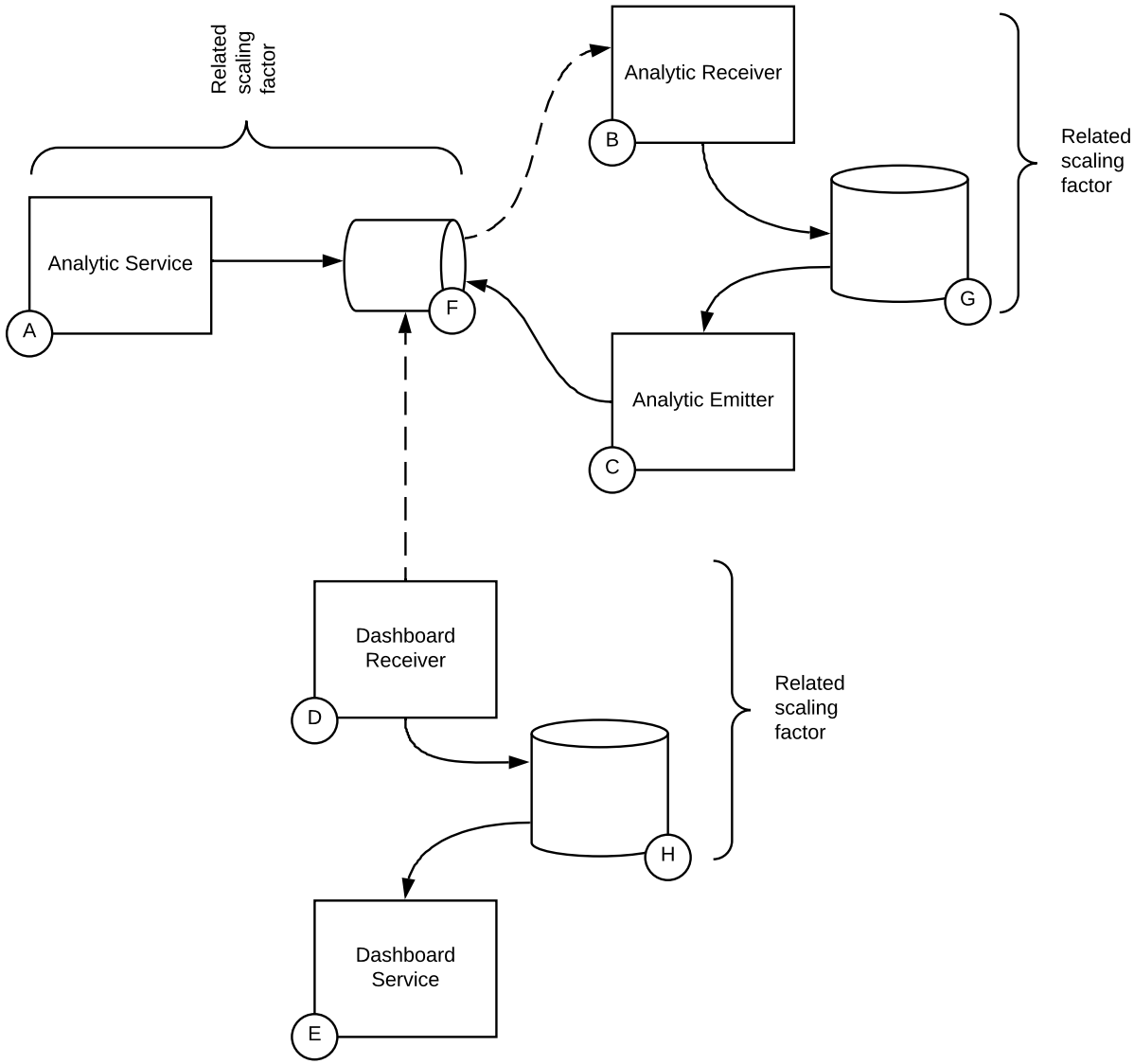
(D) This receiver is responsible for formulating time series view of the analytic data by subscribing and ingesting from the data feed (**F**) and storing it in the database (**H**). It is also able to re-create time series projection based on historical data by subscribing and ingesting data from the low priority historical analytic data topic mentioned in C.

(E) This service serves the time series analytic data view by reading it from the database (**E**) to the user using a web interface (REST, WebSockets, Comet)

(F) Data feed that is responsible partitioning and queuing data for processing, Kafka is a good choice for it's capabilities to be highly scalable but also because of the convenience of consumer groups. Alternatives like PubSub and Kinesis have a very similar offering to Kafka, but it means vendor lock-in.

(G) Database optimized for warehousing, this means highly scalable and focused on write speed. Cassandra fits the bill for highly scalable and boasts sufficiently good write speeds.

(H) Database optimized for serving time series data, this means highly scalable and efficient query speeds. Elasticsearch provides excellent query capabilities and provides horizontal scaling capabilites. Alternatives like Cloud Spanner or Aerospike could be a more performant solution, but it requires more heavy lifting in normalizing the data prior to read.



This design offers indpendently adjustable knobs on the primary pieces of the system to indenpently scale bottlenecks so that we use just the right amount of physical resources.

We can independently scale:
- Analytic data ingestion from users/businesses
- Analytic data warehousing
- Analytic time-series data formulation
- Analytic time-series data serving

In theory this system can be scaled to handle the billions of writes and millions of reads, it depends how much resources are available to the infrastructure. The design pattern and technolgies proposed are meant to adhere to DDD and reactive architecture to meet the tenets of the reactive manifesto.

***Related scaling factor**: Generally would not scale one without the other

***Assumptions**:
- Authentication of data in transit is handled using self signed tokens
- All events are designed to be idempotent and will not cause duplicate processing errors
- A CDN could be introduced in front of Dashboard Service for increased read performance