New Designs in Data Platform Reference Architectures

Ramesh K Balasubramanian
Principal Engineer, Qualcomm

“Architectures evolve with scars,
not just strategies”

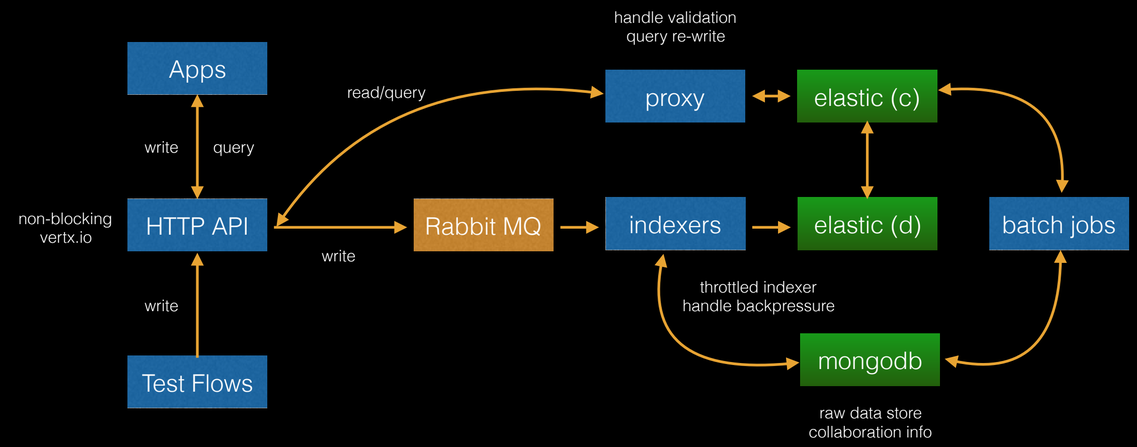
Use case[s]

- Hardware/Software simulations and tests generate huge datasets
- Needs to be ingested at scale, managed and analyzed
- Functionally quite similar to (not in terms of data)
  - Manufacturing data
  - IT Operations
  - ..and other things

The journey begins

Goals

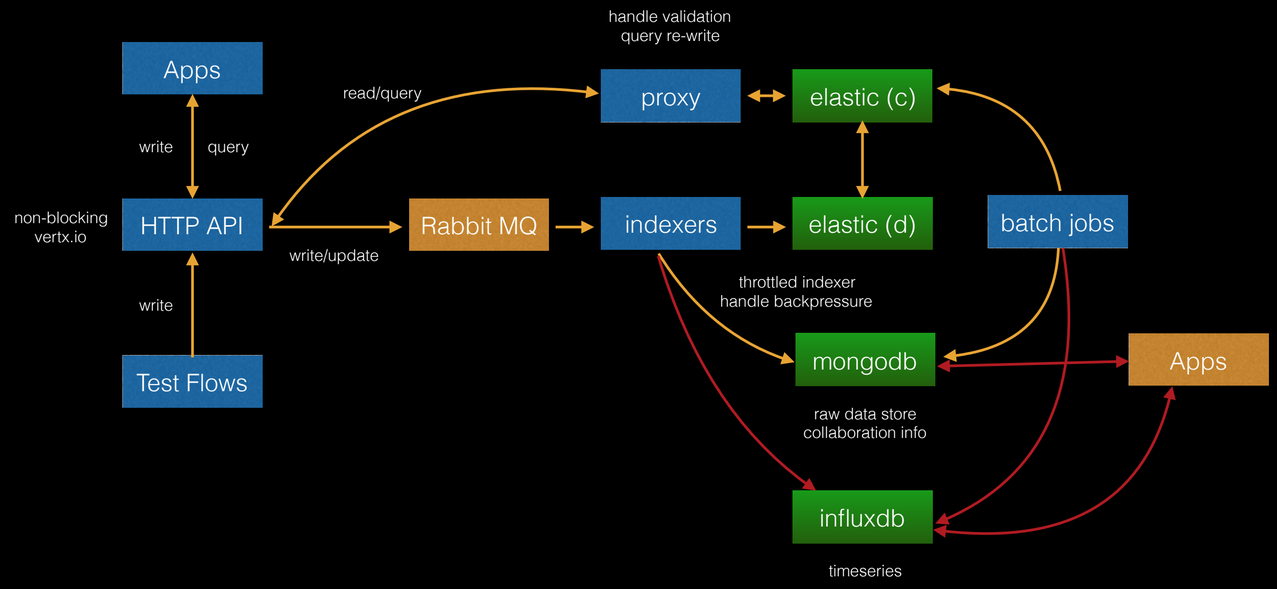
- Language agnostic
- Support flexible data models
- Handle multi-tenancy
- API to expose all data
- Support ad-hoc exploration
- Horizontal Scalability



The first wave

Goals

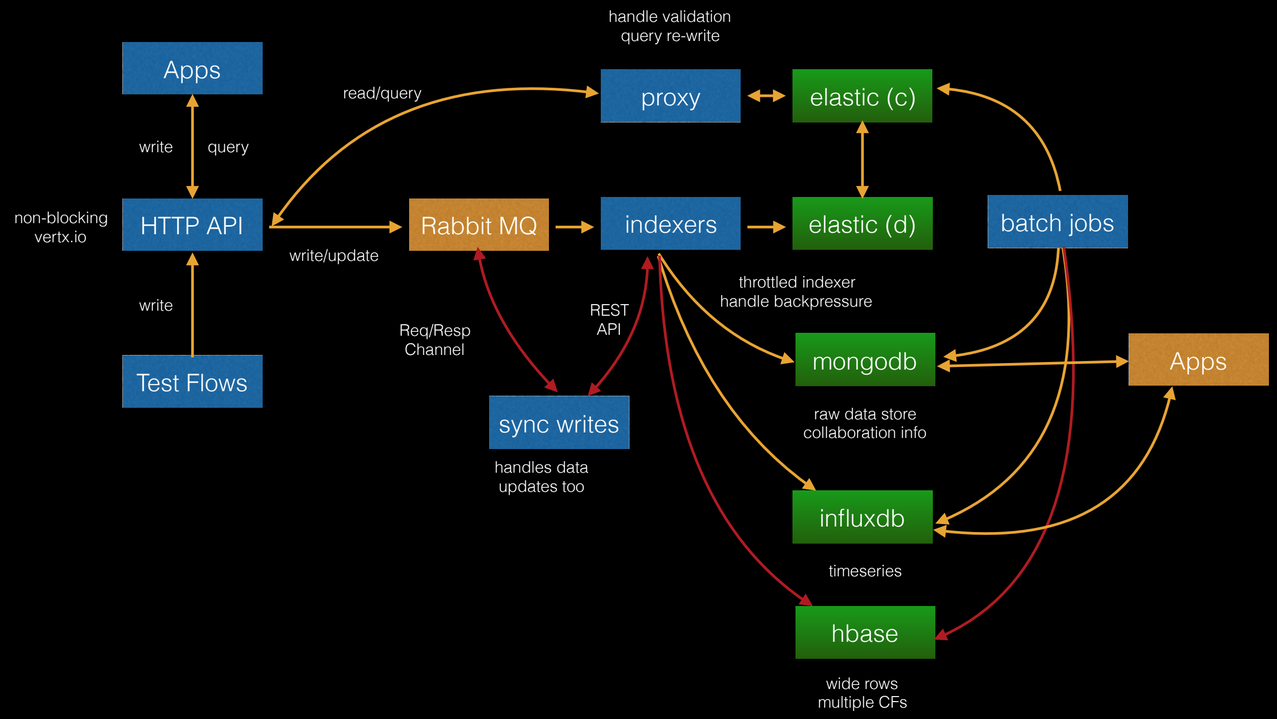
- Support time-series use cases
- Handle updates to data
- Direct access to data stores
- Additional convenience APIs
- Elastic, Mongo upgrades



The next wave

Goals

- Support synchronous writes
  - Maintain integrity across stores
- Handle docs >16MB
  - 50K + attributes (dynamic, sparse)
- Advanced query time intelligence
- InfluxDB upgrade



From where we stood

The Good Parts

- Completely functional architecture
- Evolved over time with additive improvements
- Had all the needed flexibility
- Has been running for ~1.5yrs

The Not So Good Parts

- Lots and lots of glue layer code
- 150+ monitors
- Lots of ansible code to manage deployments
- Disconnected data stores
- No compute infrastructure for machine learning
- Highly complicated integrations
- Couldn’t support ODBC across datastore

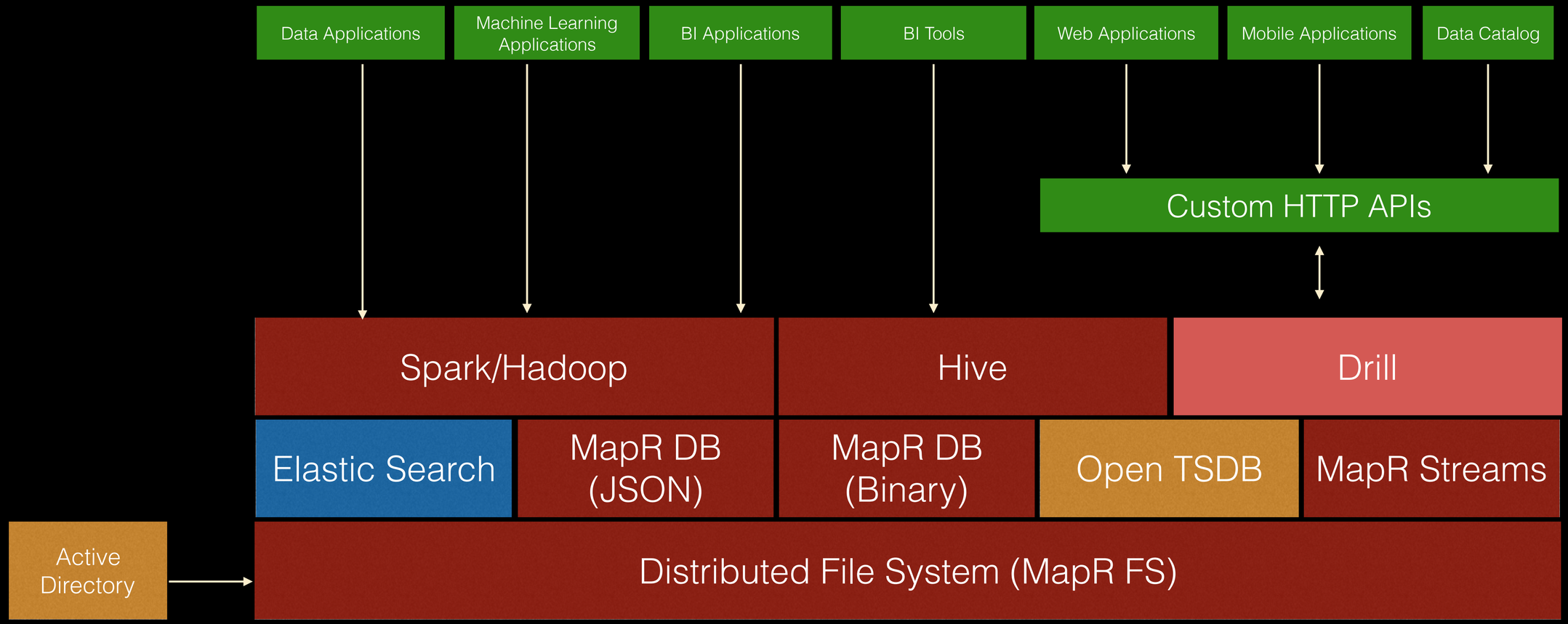
Data Platform - Foundation

- Single platform to store all data
- Multi-tenancy and security
- Tiered data storage
- Geographic scalability
- Cloud ready, yet vendor agnostic
- In-platform on-demand transformations
- Catalog all available data
- SQL queries and deep analytics
- REST APIs for data access, queries
- Usable and functional in parts

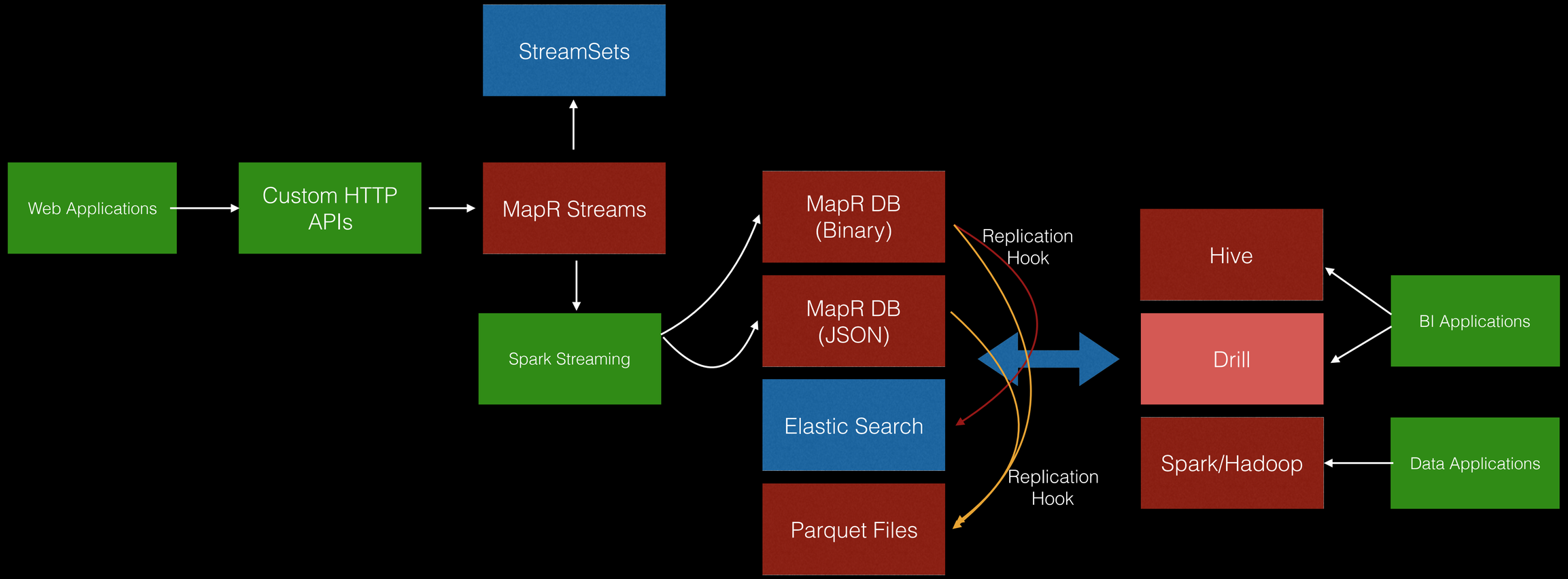
Reference Architecture - Components

- A generic high performant REST API
- Flexible storage platform
- Multi modal data stores on platform
- Streaming/Messaging components on platform
- Search Index
- Compute capacity on platform
- SQL (JDBC/ODBC) support for data access

Reference Architecture



An implementation



Recommended parts

- Develop a defined HTTP abstraction layer
- Leverage MapRFS to store raw files
  - When ever possible, write out data on disk as Parquet
- Leverage Volume mirroring to gain geographic distribution
- MapRDB (binary) for wide column scenarios
  - Secondary indices in Elasticsearch
    - Achieve through gateway at replication
  - Choose column families and partition keys wisely
  - Load data off from MapRDB to Parquet files as needed

Recommended parts

- MapRDB JSON for storing JSON documents
  - Secondary (mulitple) indices in Elasticsearch
  - Achieve through gateway at replication
  - *Need more API hooks here (not exposed currently)*
- Leverage Streams for any streaming/messaging situations
  - Spark streaming based processing to ingest data
  - Take into account that this is not per record, but a micro batch
  - Keep an eye on data flowing through the stream

Recommended parts

- Store elastic search data on MapR FS
  - Leverage shadow replicas (yes it is experimental as of now)
  - If not, use local volumes and use elastic search replication
  - Can be leveraged for small time-series (with elastic 5.0)
  - Expose necessary elastic search data through Hive
- OpenTSDB for large scale time-series scenario
  - Grafana to visualize time-series
  - We’d built our own version of Spyglass earlier

Suggested parts

- Drill or Presto
  - Functionally, too close to call this point (for our needs). YMMV
  - We suggested drill (will continue to watch Presto)
- BI on BigData
  - Too many tools and a turbid market - no clear winner
  - We suggested to leverage current tool set (through ODBC) / Custom apps
- Stream Visualization
  - We used Streamsets to visualize the flow.

Thank you!