FBC Unified SMO OpenSchema Design Proposal - Draft

Background and OpenSchema Overview

Heterogeneity and coexistence of multiple radio access wireless networks are key to the current and future generation networks. This makes multiple wireless networks available for the User Equipment (UE) to choose from for coverage, capacity, quality, and cost optimization, or the UE can aggregate multiple wireless networks, to provide an even better overall quality of experience (QoE).

To leverage the advantages or Heterogeneous Networks (HetNet) environments, key metrics from various network elements are needed to enable Intelligent Network Selection. However, independent and non-standardized capture and storage methods of network and device related metrics data increases HetNet utilization complexity and slows optimization and other possible connectivity innovations.

OpenSchema is a unified, plug and play model to identify, collect, store, and retrieve metrics and statistics from user equipment (UEs), CPEs, and different elements comprising Access Networks in a HetNet environment. Adopting OpenSchema as a common data store for network and device related metrics can provide faster and easier integration for eco-system innovation and collaboration for improved network asset planning and augmentation as well as enable new services for monetization and user experience enhancements.

The OpenSchema architecture, as a Magma Service Management and Orchestration (SMO) Plug-In, facilitates separation of the scalability discussion from other SMO components and can provide a quick and easy stepping stone for fully Magma deployment by capture key data driven insights for network planning, deployment, and operations. Additionally, a unified OpenSchema data lake facilitates ingestion of metrics from other HetNet Access sources such as APs, WACs, GWs and etc. With this architecture, building and deploying data intensive applications leveraging HetNet QoE will be much easier, faster, and more flexible to enable network cost and QoE optimization.

HetNet OpenSchema Use Cases

The main use cases of HetNet OpenSchema comprise:

- HetNet Performance Measurement: Provides a 360 degree view of network and resource consumption, user QoE, application performance, and the UE's perspective on vertical and horizontal handovers.
- Seamless Connectivity and Improved UX: Provides a basis for uninterrupted connectivity regardless of the wireless technology used.
- HetNet Aggregation: Provides a capability for aggregating all available spectrum to increase connection speed, reliability and security.

- Mobile Data Offload and Network Augmentation: Facilitates better offload/network augmentation decisions based on QoE and facilitates an Always Best Connected UE on any accessible network.
- Hybrid Access or Fixed Mobile Convergence: Provides real-time WAN connectivity metrics and user experience measurement on converged gateways.
- Training Data for Other Service: Enables use of access layer metrics to train ML or for other control functions (e.g., useful for Maveric as near-RT RIC).
- Other Use Cases: Enables policy management, time of day analytics, network slicing, etc.

OpenSchema Main Functionalities

OpenSchema, as part of the Unified SMO, must provide the following functionalities:

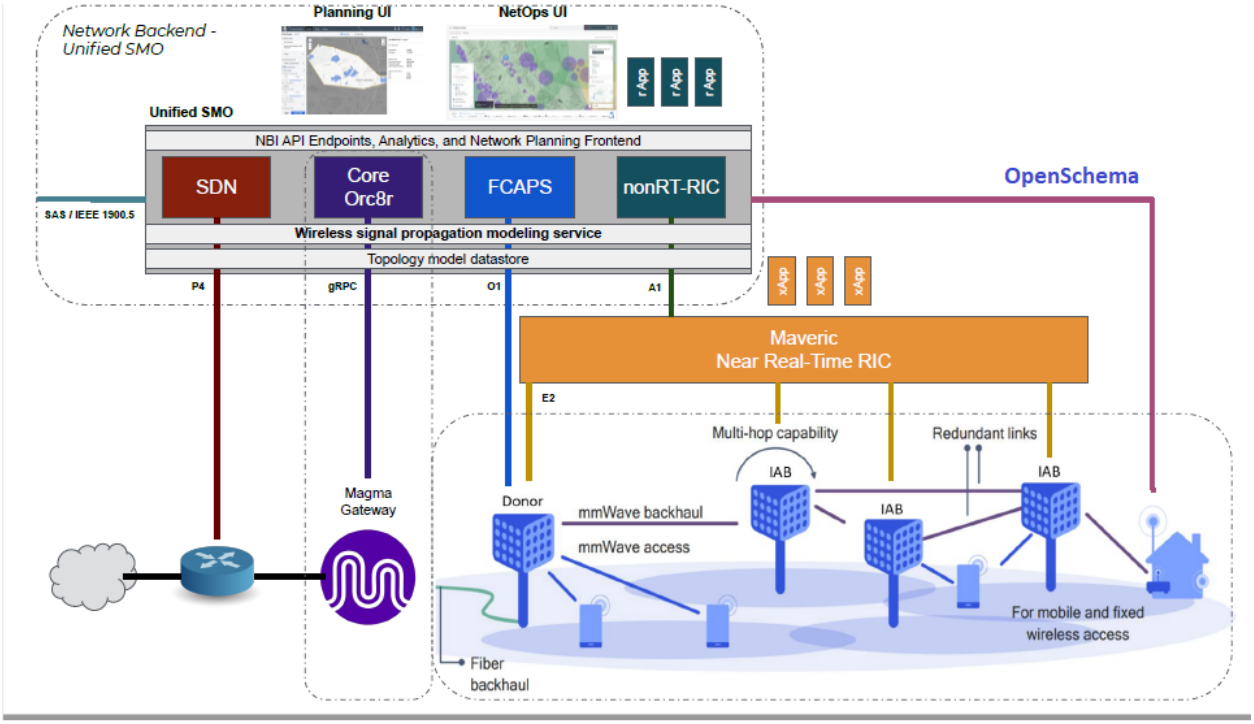
- A secure and trusted way for access devices to authenticate themselves for metrics collection and telemetry push and policy pull operations.
- A secure and efficient method to collect metrics and telemetry on UEs, APs, GWs, CPEs, and etc.
- A secure southbound APIs for HetNet policy and connection management.
- Secure APIs and Interfaces for Orchestration, Visualization, Dashboards, and Metrics Streaming.
- A scalable and cost effective datastore and analytics platform.

OpenSchema Proposed Scalability Requirements

- Average 200KB transmitted per device
- 30 million active devices per day
- Collection & Storage
    - Connections per Hour: 7.5M (Average) / 15M (Peak)
    - Average data stored per day: 6 TB
    - Ability to horizontally scale (elasticity on collection and storage)
- Data Validation
    - Format – match SDK Version
    - Size – limit to theoretical max per device / per day

FBC Unified SMO Proposal

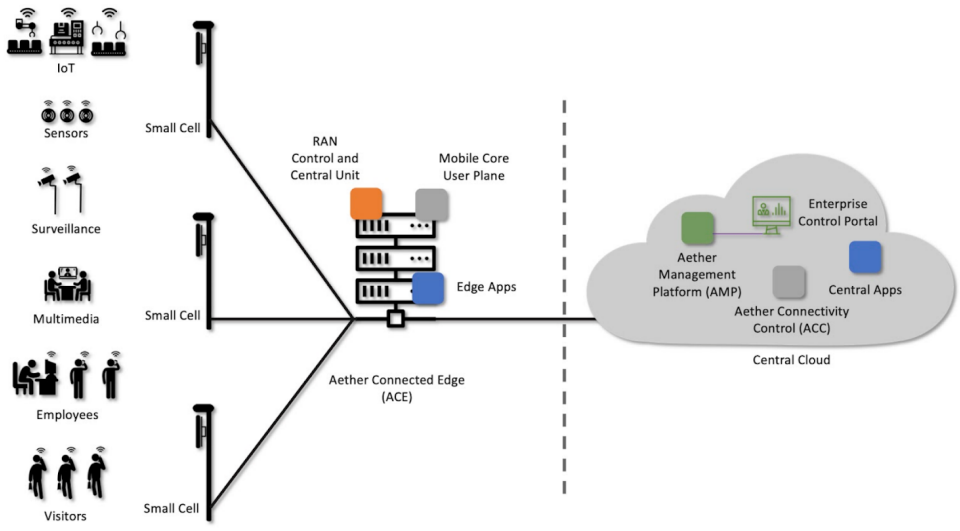
Currently, FBC is proposing to build a Unified Cloud-Native Service Management and Orchestration (SMO) by using ONF's Aether Runtime Operation Control (ROC) framework as the basis of the work. Pictured below is the initial vision for FBC/MetaC Unified SMO:



Aether Based OpenSchema Design Considerations

Aether Components

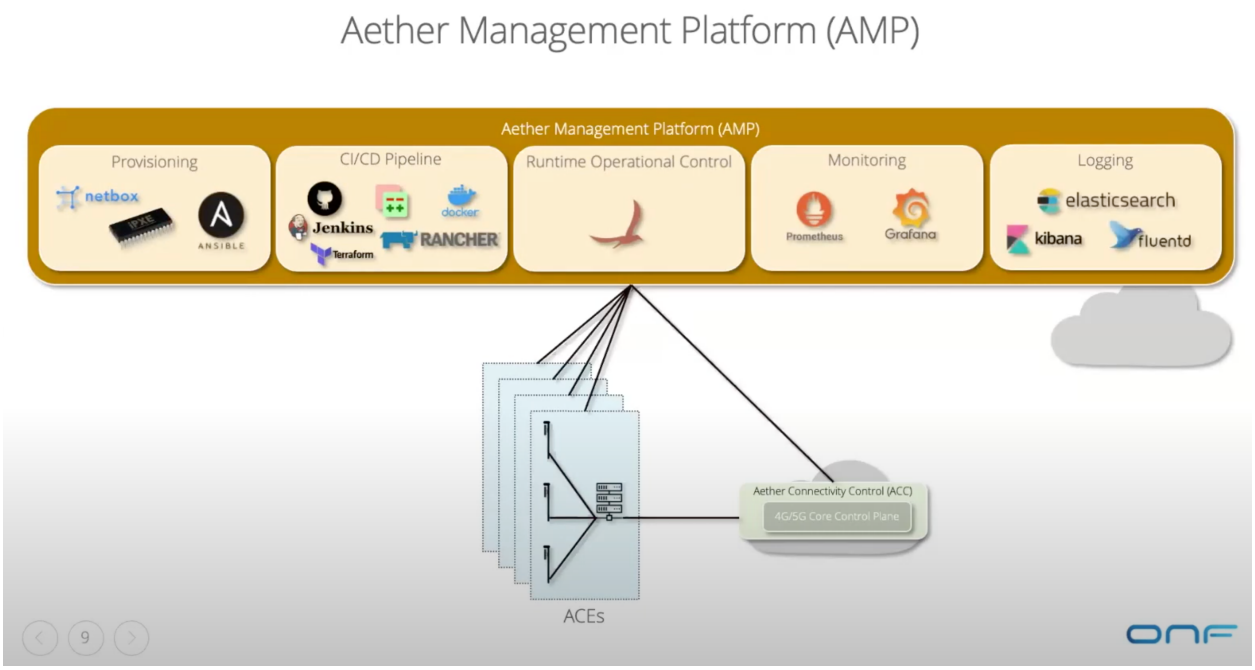
A typical Enterprise Private Cellular Aether based deployment is shown below.



Main components of Aether are the: Connected Edge (ACE), Connectivity Service (ACS), Connectivity Control (ACC), and Aether Management Platform (AMP). ACC is essentially the cellular (4G/5G) core and is deployed in the Central Cloud. Apart from ACC and AMP, most of the components are in the edge.

Aether Management Platform (AMP)

Aether Runtime Operation Control (ROC) is a subsystem of the Aether Management Platform (AMP) which is a highly scalable cloud service used to control, configure, monitor and lifecycle-manage multiple connected edges and control planes. AMP defines Aether-specific workflows for adding and controlling subscribers, small cells, mobile core user planes, and 3rd-party edge services. It provides a global infrastructure for: collecting, monitoring, and logging data about Aether sites and services, providing operator alerts and diagnostic tools for analyzing the collected data, and a CI/CD toolchain for managing the lifecycle of the deployed systems. AMP itself is cloud agnostic and can run on commodity clouds (e.g., Google Cloud, AWS, Azure), operator-own Telco clouds (e.g., AT&T's AIC) or an enterprise's own data center.

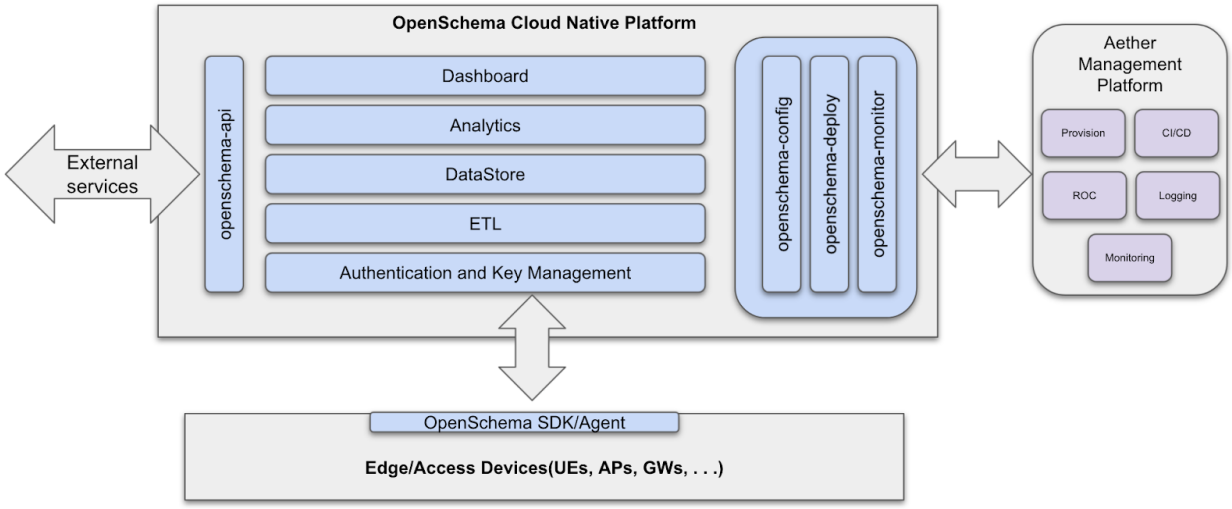


ROC Analytics Engine

Inside AMP, ROC provides an open source based collection of raw metrics and logs from edge services. Collection is accomplished using Prometheus and LogStash. There is also a Metrics API for dashboard and visualization on the Enterprise Portal.

OpenSchema Components

The Figure below represents a holistic view of the OpenSchema components and how they would communicate with external elements.



Edge/Access Devices: Collects metrics and telemetry data and pushes them using the platform specific openschema-sdk. For UEs, the openschema-sdk will be implemented and for other platforms generic golang lib.

Authentication and Key Management: Facilitates scalable authentication for Edge/Access Devices as well as other services or roles that require access to OpenSchema resources. Aether uses OpenID and JWT type tokens for authentication, but they don't have any specific method for user space applications to authenticate to the cloud. There are also some 3GPP based authentication schemes but these are not useful for OpenSchema purpose.

ETL: Provides a scalable service to receive and load metric and telemetry data in DataStore.

DataStore: A scalable, NoSql database to store complex metric structure.

Analytics: Provides post processing on raw metrics.

Dashboard: Provides visualization and monitoring of metrics and telemetry.

openschema-api: Provides an interface for external services to query and stream metrics, run jobs, and etc.

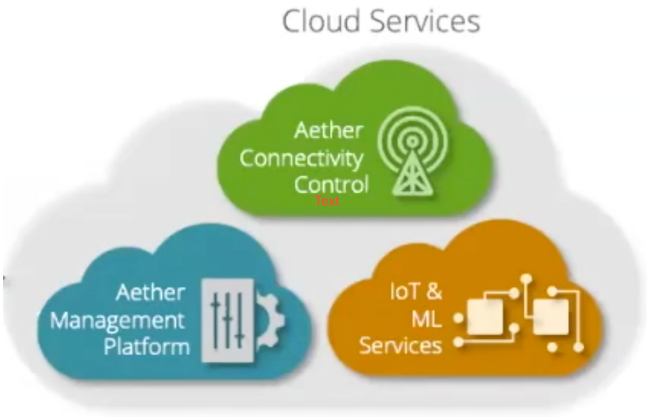
openschema-config/deploy/monitor: Simplifies configuration, deployment, and monitoring of OpenSchema by leveraging existing AMP components.

Edge Cloud Or Central Cloud Deployment Options

The current architecture of OpenSchema is based on deploying and running all the non Device/UE workloads in a centralized cloud. However depending over network architecture and deployment some components can be deployed closer to edge.

Aether 4G/5G deployment:

Aether is essentially a framework for Private 4G/5G deployments with several edge sites and a Central Cloud. Per Aether's architecture and use cases, Services that consume data can be deployed in the central cloud with some of the workload running on the edge. For OpenSchema, device SDK/lib will always run on the device edge. Other components should be containerized and be able to any edge or central cloud.



OpenSchema must also support large-scale public networks such as of MNOs and MVNOs. The initial OpenSchema design focus is up to 30 millions subs.

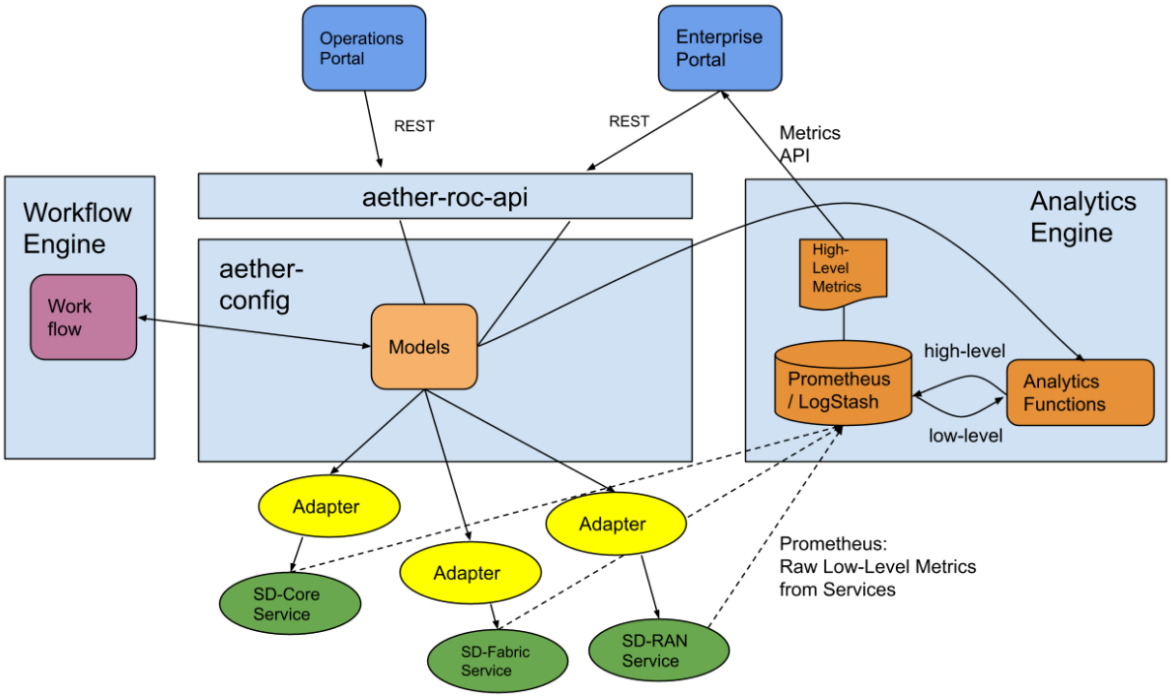
Below are some criteria needed to consider for OpenSchema:

- Cost and infrastructure availability: Is there sufficient cloud infrastructure on the edge to handle OpenSchema work load? What is the cost of central public cloud vs private cloud? What are the data transfer costs?
- Use Cases: If multiple edge locations are deployed, are raw metrics needed from each location in a central cloud? Can the metric be streams for multiple locations to generate reports, train ML models, etc.?
- Technology: For some of the backend components, should fully managed technologies from Public Cloud Vendors (e.g., AWS, Azure, GCP) be leveraged since it might not be easy or cost effective to move these technologies to the Edge?

A central cloud deployment of all the backend components is typically less complex and easier with a single point of communication for the UEs/Devices. However since all the backend components are containerized and deployed on Kubernetes, these elements can be moved closer to the edge if needed due to design or policy requirements.

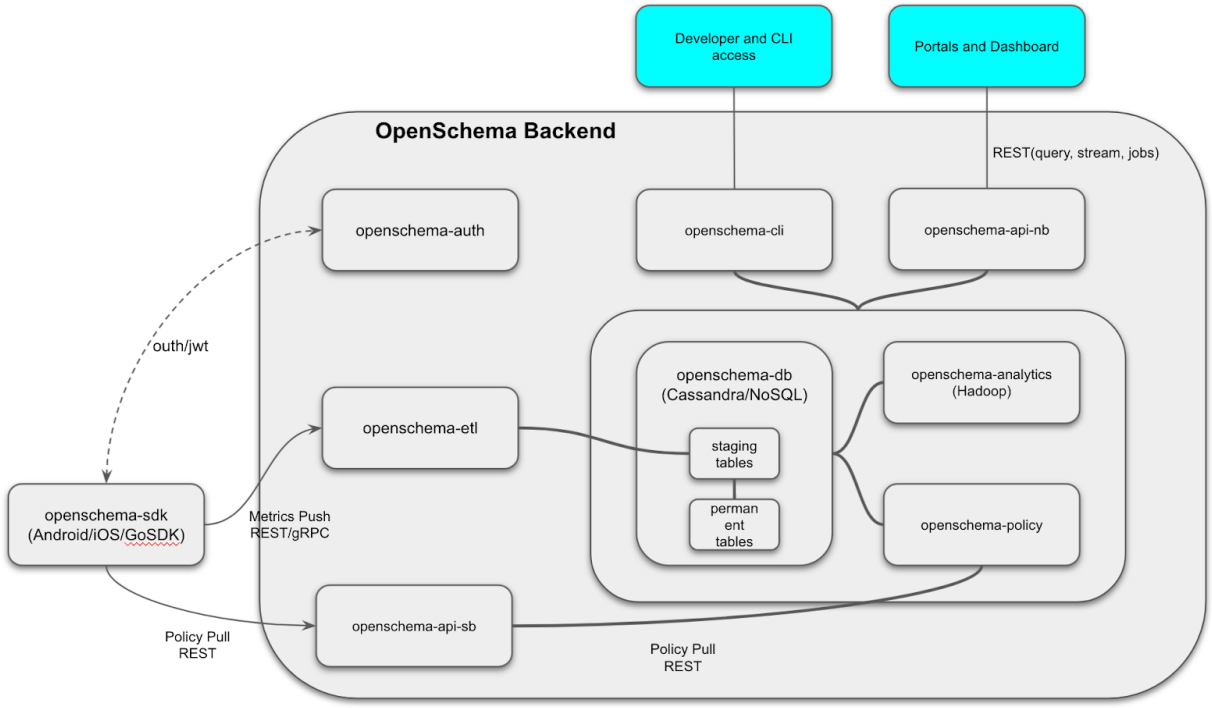
AMP, ROC, or Stand-Alone Service Considerations

Figure below is the architecture of ROC, as a subsystem of AMP. ROC has an analytics engine similar to Magma Orc8r metrics collection and is based on Prometheus/ElasticSearch and Grafana. Previously a Magma Orc8r hosted OpenSchema data lake was implemented and proved to be hard to scale due to specific OpenSchema Scalability Requirements[1].



As a result, OpenSchema data lake was proposed as a fully separate and stand-alone component that can be a subsystem of AMP or deployed side-by-side AMP (similar to ACC) in the central cloud. This way OpenSchema can also leverage existing monitoring, logging, and other operational services available in AMP. Furthermore, by having OpenSchema as a fully stand-alone component, the architecture is more flexible in case OpenSchema components are divided between edge and central cloud.

The Diagram below shows the summary and flow of OpenSchema Components. In the following section, the main components of the OpenSchema technology (DataStore(DB), Analytics, and ETL) are presented.



Proposed Database Technology for OpenSchema DataStore *(i.e., Rationale for Apache Casandra)*

Exploration of Databases

Two widely use NoSql Databases were evaluated against Cassandra. MongoDB was evaluated in the OpenSchema proof of concept and as part of performance evaluation against Magma Prometheus based metric collection. HBase as a potential database was also reviewed for the data lake. While both MongoDB and HBase are a much better fit than Prometheus for OpenSchema metrics collection[1], there are still shortcomings when compared to the proposed use of Cassandra and these are described below.

Data Model:

- OpenSchema uses a fixed schema, and hence there's no need for MongoDB json-style document store.
- MongoDB supports an arbitrary schema, allowing deeply nested and extremely flexible structures. OpenSchema's schema can always fit into key value pairs, therefore MongoDB's secondary index support is not needed. Querying deeply nested data also comes with a significant performance hit, which is unnecessary for OpenSchema use cases.

- HBase is a column-based database, which may limit the type and efficiency of data collected and stored.

Availability:

- MongoDB has a single master node, which means it's not 100% available. When the master node is down, there can be up to 1 minute where UEs cannot post data. This would require executing multiple data posting retries on the UE and impact system efficiency.
- HBase functions as a master-slave as well, which means there's still a single point of failure. In addition, it only handles data management, and relies on other technologies like HDFS for storage.

Scalability:

- OpenSchema is write heavy, so it requires multiple write nodes.
- MongoDB has a single master write node, which is not acceptable and limits system scaling.

Query Language:

- SQL remains the most popular query language for data scientists[1], which is crucial for complex queries that OpenSchema is designed for.
- MongoDB does json style queries instead.
- HBase has no query language support.

Dependencies:

- HBase requires Zookeeper to route clients to the nodes to write to.
- HBase also requires HDFS for storage.

Data Base Proposal

The only database that meets all of the OpenSchema goals and requirements thus far is Cassandra. The following outlines Cassandra's suitability:

Data Model:

- Cassandra uses wide-column store that's flexible and sufficient for OpenSchema.

Consistency:

- Cassandra has eventual consistency instead of immediate consistency. However, this does not matter as much for OpenSchema as most of the analytics are processed offline after reaching consistency.

Availability:

- Cassandra is master-less, which means no down time. This comes at the cost of potentially much longer read latency if it needs to search through all nodes. However, for OpenSchema use cases, read latency is not as critical since there's no current need for true real-time analytics.

Scalability:

- Cassandra supports multiple write nodes for high scalability.

Query Language:

- Cassandra supports CQL that's similar to SQL.

Aggregation:

- Cassandra is widely integrated with Hadoop or Spark for data aggregation.

Pricing:

- Cassandra is open source and completely free to use.

Deployment

Cassandra's deployment is very well documented, and in most cases only requires a couple of configuration files and scripts which make the operation and DevOps activities simple and easy to automate. Additionally many public cloud vendors offer "Managed Cassandra" or Cassandra compatible services that can further speed up and simplify deployment and reduce operational cost.

Query APIs

Cassandra CQL is very similar to SQL with some caveats.

```
SELECT * | select_expression | DISTINCT partition
FROM [keyspace_name.] table_name
[WHERE partition_value
   [AND clustering_filters
   [AND static_filters]]]
[ORDER BY PK_column_name ASC|DESC]
[LIMIT N]
[ALLOW FILTERING]
```

Integration with Grafana

Cassandra cluster can be monitored on Grafana using DataStax metrics collector.

While Cassandra is designed to be more than a Time Series Database, a good portion of the OpenSchema metrics are actually time series data. As such, it will be useful to add this Cassandra Datasource for Grafana to visualize the time series data.

Analytics Technology for OpenSchema: Hadoop is Proposed

For OpenSchema, an analytics engine is needed that is able to query the data in a complex way and provide highly insightful business intelligence.

For example, it would be very useful to know the potential data offload opportunity for the number of users that were nearby an accessible Wi-Fi SSID but didn't connect but other users have had a good QoE on that SSID. This could provide a "lost offload opportunity" metric and help prioritize areas of further investigation.

Another example, where network insight could be ascertained, would be understanding how many users connect to a particular Wi-Fi SSID, have a bad connection (e.g. packet loss > 2% or slow speeds), and then disconnect from it.

It is also possible, through machine learning, to learn if a particular usage pattern indicates poor user experience. For example, a user who typically consumes 100MB within an hour whenever s/he is active, only consumes 10MB and spent much less time in a particular location, possibly indicating a poor user experience that should be investigated further.

These analytics can provide tremendous value to businesses, but are very complex and may not be feasible or very challenging to do with PromQL.

Analytics Goals

A highly scalable highly available analytics engine that fits the following criteria is proposed:

- Able to process large amount of data.
- Provide good read performance for batch processing and support for near real time analytics.
- Provide the facility for complex queries.
- Is cost effective.
- Is an open-source with a strong community.

Exploration of Other Analytics

As part of the investigation, Hadoop, Spark, and Kafka analytics capabilities were explored for comparison. While all of these technologies are highly performant, they fall short when compared to the proposed use of Hadoop. The shortcomings of the other options are broken down below into several categories:

Architecture:

- Spark compute is all in-memory, so it's much faster, but more costly to deploy. OpenSchema use cases do not warrant additional cost since processing speed is not as critical.
- For Spark, workload size is also limited by amount of memory. OpenSchema requires processing large amount of data that can quickly become very costly to store in RAM.

Scalability:

- Due to the cost of RAM, Spark scalability can be hindered.

Ease of use:

- Spark may be easier to use, however it hasn't been around very long (2014), which means less people are familiar with it.

Stream processing:

- Spark has Spark Streaming framework for stream processing, for low latency real time analytics.
- Kafka offers true real time processing, and is more flexible than Spark.
- OpenSchema use cases do not require true real time processing, hence the advantages provided by Spark Streaming and Kafka do not matter. In addition, Cassandra already provides near real time analytics through its built-in CQL.

Analytics Proposal

The only analytics engine that meets all of the OpenSchema goals and requirements best at this time is Hadoop. The following outlines Hadoop's suitability:

Architecture:

- Hadoop can be deployed on top of Cassandra, avoiding the needs to move data to HDFS.

Scalability:

- Hadoop is easy and cost effective to scale.
- Hadoop can process very large datasets that exceed available memory.

Machine learning:

- Hadoop comes with the ML library Mahout.

Ease of use:

- Hadoop has been around since 2006, and is a well known framework among developers.

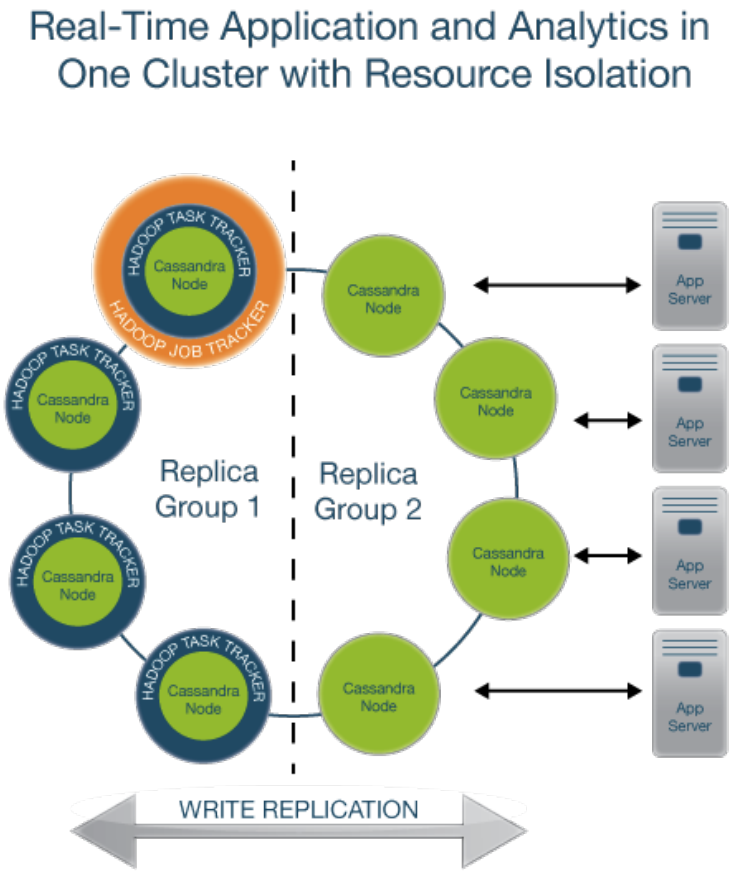
Batch processing:

- Hadoop is good for batch processing, good for many of OpenSchema use cases where immediate results are not required and time is less of a limiting factor.

Hadoop Deployment with Cassandra

With the Cassandra Hadoop integration, both real time analytics (through Cassandra-only nodes) and batch processing (through Hadoop nodes) can be supported by the same cluster.

The following block diagram shows how Cassandra and Hadoop workloads can be isolated to provide the best of both worlds.



OpenSchema ETL Technology

Go is recommended as the main language to be used for OpenSchema Backend since it's designed for high performance multiprocessing, managing the large amount of incoming requests, and redirecting them to write into Cassandra makes it a good fit.

The Magma team also has experience managing code in Go, which is an added bonus to the learning curve for this project.

gocql

GoCQL is a driver written in Go to easily interface with Apache Cassandra. This is the largest Go project mentioned in the official Cassandra drivers documentation.

https://github.com/gocql/gocql

Go Backend

Using Go, a server can be established using the standard libraries available. Although a web server framework is optional, it may accelerate development but following Magma's Go dependency considerations (https://github.com/magma/magma/tree/master/src/go, the added weight should be evaluated.

Some other options can be gorilla/mux: https://github.com/gorilla/mux or gin: (https://github.com/gin-gonic/gin).

Endpoint Communication

Endpoints are exposed following REST API principles. REST APIs is one of the most commonly used standards for a web-based API. Also, due to the nature of mobile clients and not necessarily updating all in a timely manner, interface portability through API versioning will be important.

Client Authentication

Since mobile clients will only be pushing data to the cloud, there won't be multiple roles or login/logout which would require an authorization protocol like OAuth2.0. JSON Web Tokens (JWT) (https://jwt.io/) will be used to authenticate that mobile clients are valid through the HTTP 'Authorization' header. JWTs have the added benefit of not needing the additional storage space neither in the server nor in the client. OpenID (also used by Aether) can be utilized to facilitate JWT.

Metrics Data Model

UE and others: reference to https://github.com/magma/openschema-docs

Summary and Suggested Next Steps

A new high level design for OpenSchema data lake is proposed as a standalone and plug and plug module for FBC/MetaC Unified SMO. This design can support private 4G/5G deployment, as is the purpose of the ONF Aether framework, as well as larger MVNO or MNO deployments

and a variety of different use case. For consistency and ease of future integration, OpenSchema data lake leverages existing tools and APIs from Aether framework. also consistent with Magma and Aether, Go is the preferred language for all the ETL, Authentication and DataStore drives.

For private networks and long tail of small operators, standalone OpenSchema deployment is an easy and fast way to start measuring and quantifying network performance and pave the ways for larger Magma or Aether deployments witch require more time and investment.

Next Steps

- Develop a PoC based on this design
- PoC Filed Trials
- Validate design and make necessary modification
- Prepare a detail roadmap and schedule for stand alone OpenSchema

References

1. Magma Data Lake Performance Evaluation:https://docs.google.com/presentation/d/1SkdlnwcDoq1IrLO1o_WRPD172K_nOi7b/edit?usp=sharing&ouid=101410299033145733032&rtpof=true&sd=true
2. AWS Keyspaces: https://aws.amazon.com/keyspaces/

List of Acronyms

ACC	Aether Connectivity Control
ACE	Aether Connectivity Edge
AMP	Aether Management Platform
AP	Access Point
API	Application Programming Interface
CI/CD	Continuous Integration and Continuous Deployment
CPE	Customer Premise Equipement
ETL	Extract Transform Load
HDFS	Hadoop distributed FileSystem
HetNet	Heterogeneous Networks
JWT	JSON Web Token
ONF	Open Networking foundation
QoE	Quality of Experience

REST	Representational state transfer
ROC	Runtime Operation Control
SDK	Software Development Kit
SMO	Service management and orchestration
UE	User Equipment