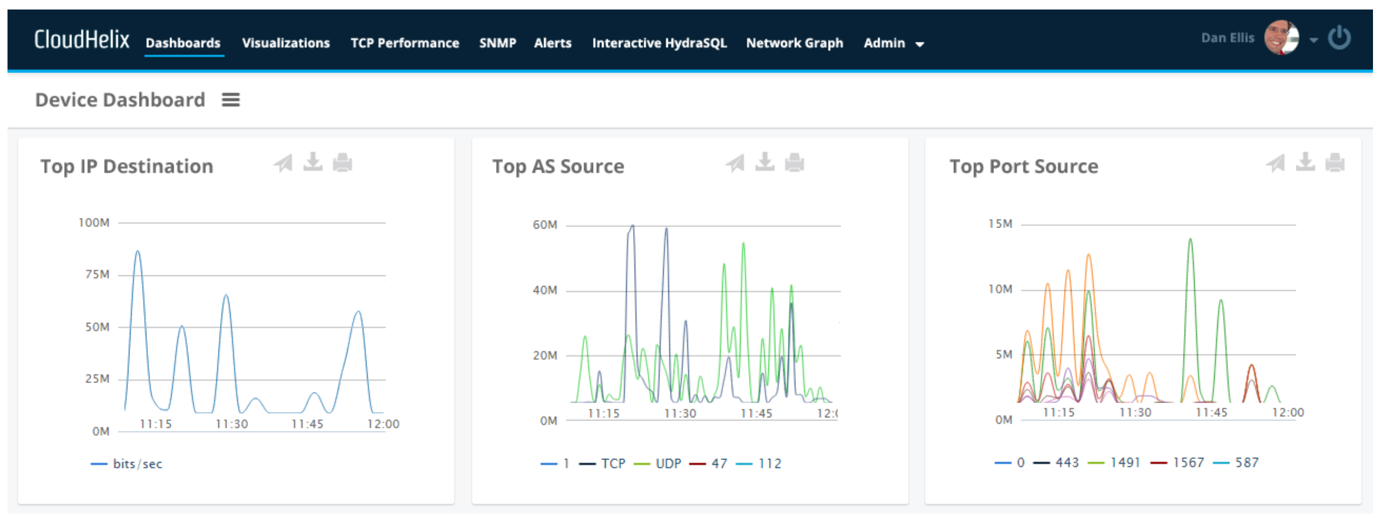
CloudHelix



Motivation

CloudHelix was founded to provide a cloud or on-premises offering that collects infrastructure data and allows flexible reporting, anomaly detection, alerting, and prediction. The carriers we work with want to be able to quickly answer questions around availability, security, attack detection, and performance. To fully support their requirements we need to be able to provide:

- Unsummarized retention of flow data covering tens to hundreds of terabits of traffic;
- Monitoring of flow data in real time;
- SQL-based querying of historic data.

After a year of exhaustive research, it was determined that no existing backend could provide continual data ingestion at the scale necessary to support fast querying and multi-tenant access. So we decided that we needed to build our own backend.

Usage Requirements

In defining our backend requirements, we determined a need for:

- Scale-out architecture
- Supporting multi-tenant and multi-user access
- Sub-second querying of years of data (for regular queries)
- The ability to do multiple parallel forensic queries across all of the data without disrupting the speed of the system for other users
- An API to access all data from individual flows to aggregates

Architectural Requirements

Our usage requirements led to the following architectural requirements:

- Support for constant high speed ingest, to trillions of records/day per cluster.
- Replication on ingest and resync on disk/node failure, for durability and HA.
- Data indexing for fast subselection without linear scan.
- Compression support for efficient storage (and I/O read speed) of data.
- Subquery caching/memorization for reuse without re-querying.
- Subquery rate-limiting to protect the cluster from large unbounded forensic queries.
- Support for in-memory storage of recent data to enable continual low-latency real-time DDoS scanning.

Alternatives Considered

We were unable to find any open source flow system with the required cluster support and multi-tenancy features (subquery caching and rate-limiting). We considered Hadoop and other OSS Big Data solutions, all of which have issues with ingest rate and/or query times. We were also unable to find an enterprise solution (e.g. Teradata, Vertica, etc.) offering what we considered to be a viable cost/capacity relationship (our first deployment is multiple petabytes). Architecturally we were aiming for something like Google's Dremel, but Impala and Drill are not yet able to support the levels of ingest and multi-tenancy required by our capacity and query requirements.

HydraSQL Architecture

To meet the requirements outlined above, CloudHelix built HydraSQL, a backend with the following architecture:

- An ingest layer receives flow and BGP data, unifies, aggregates, and indexes it in parallel to a set of N on-disk machines. Data is stored in columnar format sliced by time for months to years, and to a set of M in-RAM machines for more recent data.
- Postgres is used as a frontend/API. Queries are parsed by Postgres and passed to the HydraSQL system, which breaks up the query by time and target, checks a persistent subquery cache, doles out subqueries to the backend nodes that have the data, and combines the results.
- A scale-out metadata layer is used to track data across the nodes from index time to optional purge.
- Compression support for file storage to provide storage and I/O read efficiency.

Roadmap

- Resync is currently done only if nodes fail; we use RAID underneath the file system, but will be implementing JBOD support with resync on disk failure.
- Replication is supported as if all nodes are in the same (potentially distributed) cluster. We can already support geo-distributed storage of flow, but configuration is overly manual. Eventually we want make it easier to configure both topologies: hub and spoke as well as multi-hub.
- Sub-query caching will be further optimized for subsets of already-asked queries.
- Rate-limiting of queries will be upgraded to integrate feedback from the current cluster capacity/IO usage.
- Storage efficiency of older data will be improved by integrating erasure coding and automated migration to erasure-coded pools of storage.
- The SQL parser may be modified at some point to make queries more compact.

System Experience

The HydraSQL backend is in production today for the CloudHelix monitoring service, running on a cluster of machines in Ashburn, VA and supporting 40+ users with hundreds of devices, billions of flow records/day, and peaks of millions of flows per second of ingest. The portal front end retrieves all data by executing SQL queries, and can continually load pages driven by tens of queries in a few seconds or less. We've also completed several private deployments of HydraSQL on privately owned clusters for use on-premises by networks and enterprises.

As reflected in the roadmap, there is still work to be done, primarily in the following areas:

- Easier installation and configuration.
- More efficient operation in disk storage (optional erasure coding support).
- IO efficiency (removing per-node RAID).

Example Queries — All devices exporting flow have logical tables, and SQL queries can be run via DBI, psql client, or API. Increasingly complex queries may be created by selecting by multiple criteria and aggregating in different ways. A few examples:

Flow export, 100 flows in last 30 seconds for tx1_readnews_com:

```
SELECT * FROM tx1_readnews_com WHERE ctimestamp >
30 LIMIT 100;
```

Top 10 traffic generators by /24 over last 3 hours for all devices:

```
SELECT ipv4_src_addr AS f_cidr24_ipv4_src_addr,
round(sum(IN_pkts)/(3600*3)) AS f_sum_IN_pkts$pps,
round(sum(IN_bytes)/(3600*3)*8) AS
f_sum_IN_bytes$mbps FROM ALL_devices WHERE
ctimestamp > (3600*3) GROUP BY
f_cidr24_ipv4_src_addr
ORDER BY f_sum_IN_bytes$mbps DESC
LIMIT 10;
```

Retransmits > .1% by ASN at prime-time for ASNs with > 10k pkts:

```
SELECT i_start_time, src_AS, dst_AS,
sum(tcp_retransmit) AS f_sum_tcp_retransmit,
sum(out_pkts) AS f_sum_out_pkts,
round((sum(tcp_retransmit)/sum(out_pkts))*1000)/10
AS Perc_retransmits FROM mrn01_readnews_com WHERE
i_start_time >= '2015-01-09 22:00:00' AND
i_start_time < '2015-01-10 06:00:0' GROUP BY
src_AS, dst_AS, i_start_time HAVING sum(out_pkts) >
10000 AND (sum(tcp_retransmit)/sum(out_pkts))*100 >
0.1 ORDER BY Perc_retransmits DESC;
```

“Top 5” traffic generators by ASN over a period for all devices:

```
SELECT i_start_time, src_AS, sum(f_sum_IN_bytes) AS
sum_IN_, sum(f_sum_out_bytes) AS sum_out_ FROM (
SELECT i_start_time, src_AS, sum(IN_bytes) AS
f_sum_IN_bytes, sum(out_bytes) AS f_sum_out_bytes
FROM ALL_devices WHERE i_start_time >= '2015-01-09
17:53:00' AND i_END_time < '2015-01-09 18:53:00'
GROUP BY i_start_time, src_AS ORDER BY
f_sum_IN_bytes DESC ) a WHERE src_AS IN ( SELECT
src_AS FROM (SELECT src_AS, sum(IN_bytes) AS
f_sum_sum_bytes FROM ALL_devices WHERE i_start_time
>= '2015-01-09 17:53:00' AND i_END_time < '2015-01-
09 18:53:00' GROUP BY src_AS ORDER BY
f_sum_sum_bytes DESC LIMIT 10 ) aa ) GROUP BY
i_start_time, src_AS ORDER BY i_start_time ASC;
```