free

Akvorado: a Flow Collector and Visualizer Backed by ClickHouse

Vincent Bernat

2022-07-27 — San Francisco Bay Area ClickHouse meetup

free

About Free

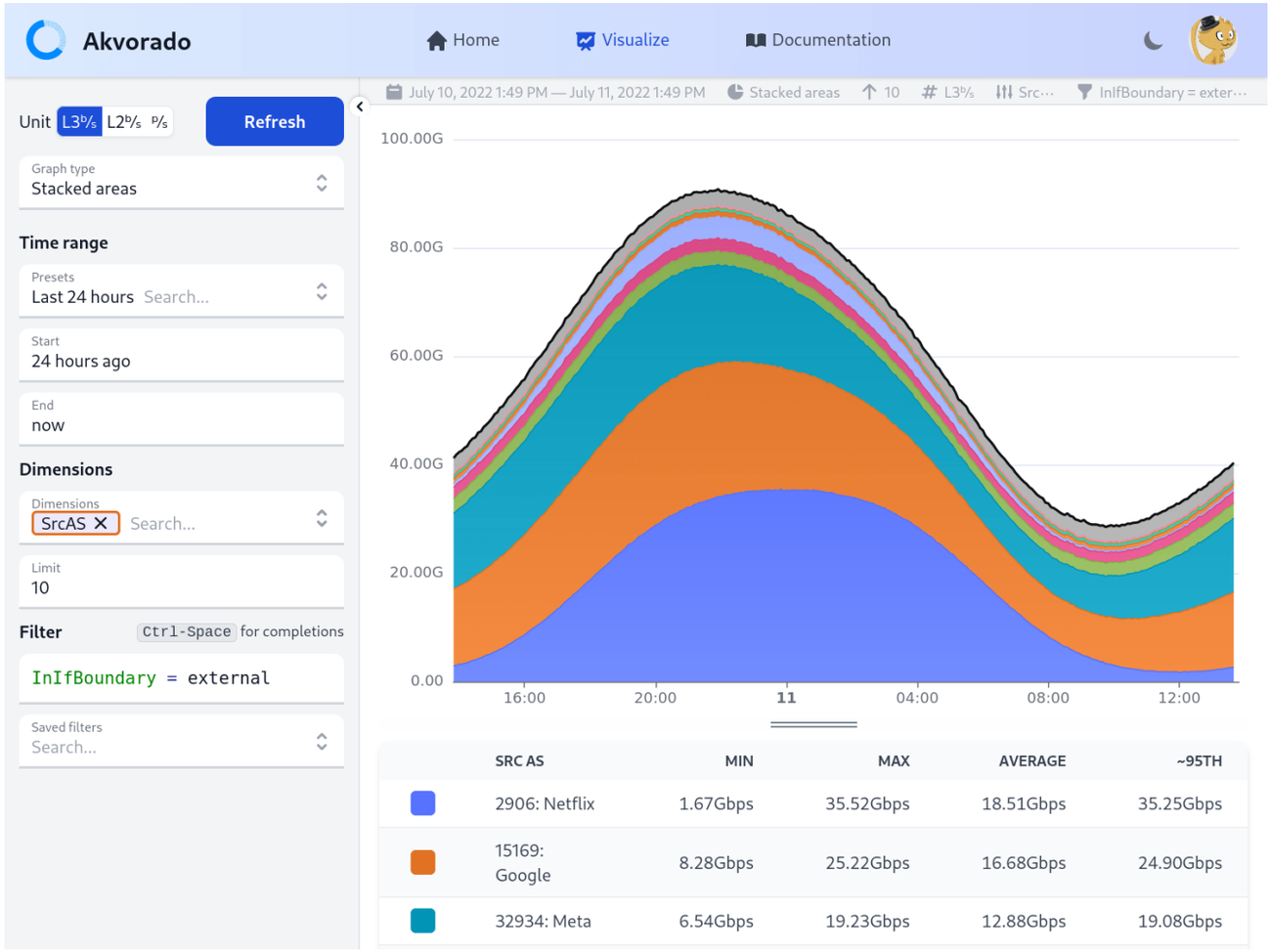
- French ISP
- 1999: free **dial-up internet** access
- 2002: ADSL access
- 2004: “triple play” with the Freebox
- 2007: FTTH access
- 2008: IPv6
- 2012: mobile offers (3G/4G)

About Akvorado

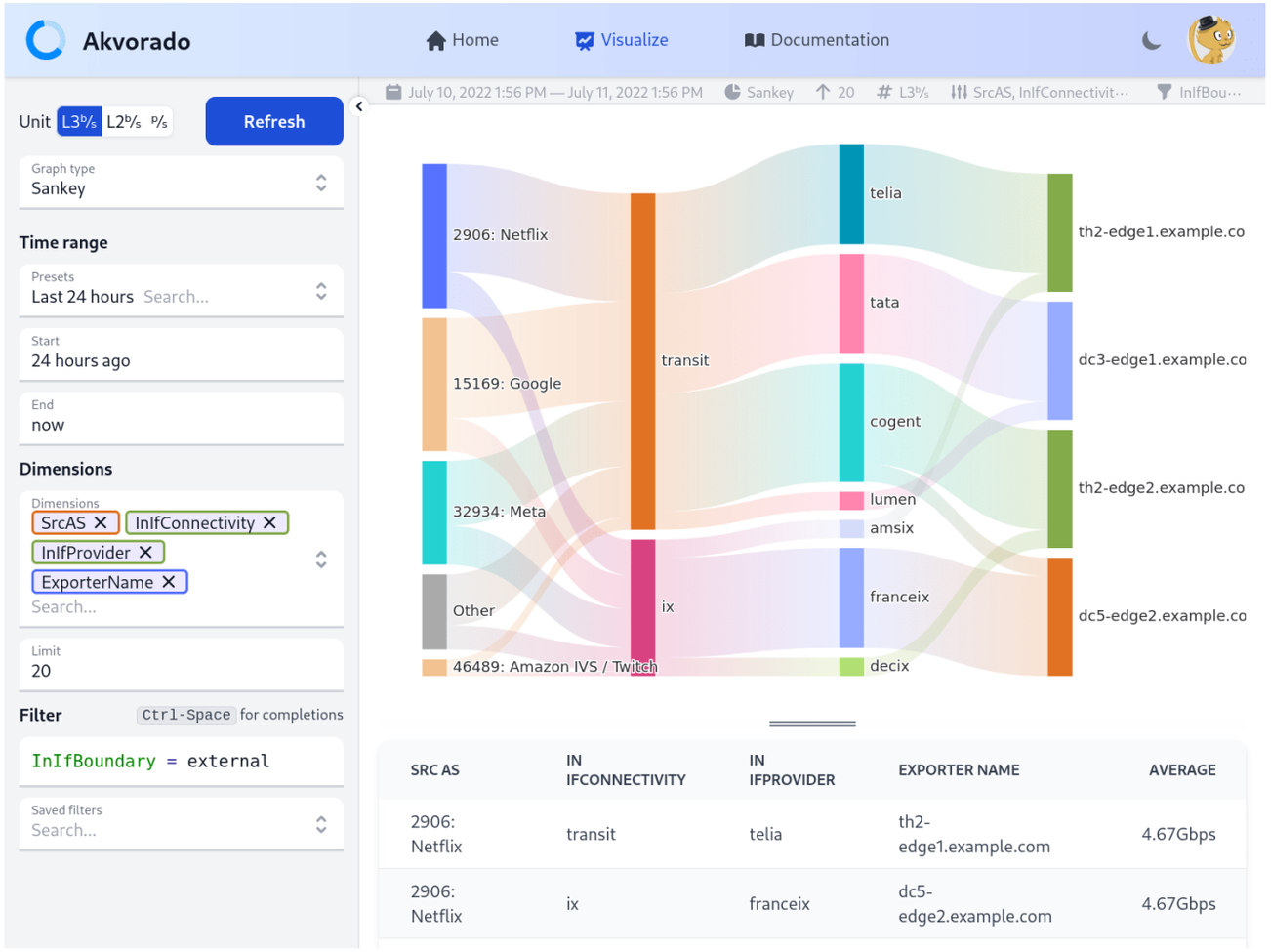
- NetFlow/IPFIX/sFlow collector
- Enrich data (GeoIP, interface names, classification)
- Serialize to Protobuf and send to Kafka
- Opensource: https://github.com/vincentbernat/akvorado
- Web frontend to query data

Screenshots (1/3)

Screenshots (2/3)



Screenshots (3/3)



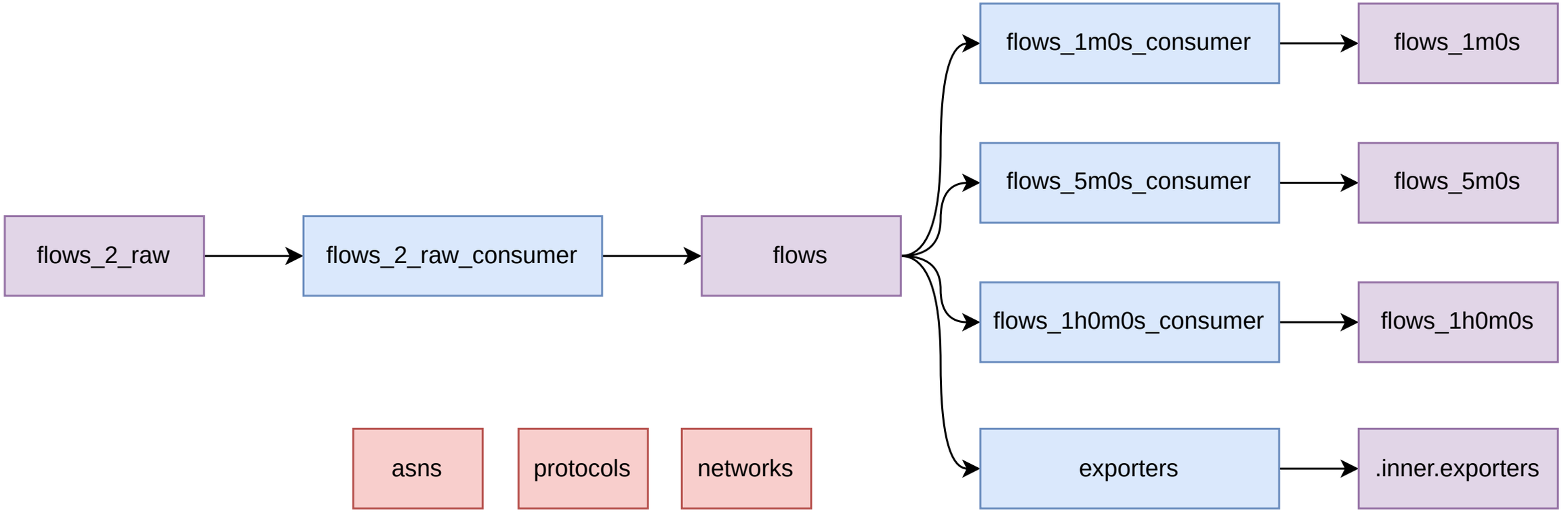
SRC AS	IN IFCONNECTIVITY	IN IFPROVIDER	EXPORTER NAME	AVERAGE
2906: Netflix	transit	telia	th2-edge1.example.com	4.67Gbps
2906: Netflix	ix	franceix	dc5-edge2.example.com	4.67Gbps

free

Live demo

https://demo.akvorado.net

ClickHouse usage



Ingestion

- Kafka engine
- Data encoded with ProtoBuf
- Versioned schemas: `FlowMessagev1`, `FlowMessagev2`, …
- Versioned topics: `flows-v1`, `flows-v2`, …
- Versioned tables: `flows_1_raw`, `flows_2_raw`, …
- No registry

Flows table

```
CREATE TABLE default.flows
(
    `TimeReceived` DateTime CODEC(DoubleDelta, LZ4),
    `SamplingRate` UInt64,
    `ExporterAddress` LowCardinality(IPv6),
    `ExporterName` LowCardinality(String),
    `SrcAddr` IPv6,
    `SrcAS` UInt32,
    `SrcNetName` LowCardinality(String),
    `SrcCountry` FixedString(2),
    `InIfName` LowCardinality(String),
    `InIfDescription` String,
    `InIfSpeed` UInt32,
    `InIfConnectivity` LowCardinality(String),
    `InIfProvider` LowCardinality(String),
    `InIfBoundary` Enum8('undefined' = 0, 'external' = 1, 'internal' = 2),
    `EType` UInt32, `Proto` UInt32,
    `SrcPort` UInt32,
    `Bytes` UInt64, `Packets` UInt64
)
ENGINE = MergeTree
PARTITION BY toYYYYMMDDhhmmss(toStartOfInterval(TimeReceived, toIntervalH
ORDER BY (TimeReceived, ExporterAddress, InIfName, OutIfName)
TTL TimeReceived + toIntervalSecond(1296000)
```

free

Aggregating timeseries

- The `flows` table keeps 15 days of data (500 GB)
- Slow to query over a large period of time
- Too big to keep for 5 years

RRD-like aggregation

- RRD-like aggregation to keep data longer:
  - Summing merge tree on `Bytes` and `Packets`
  - Drop IP addresses
  - Drop TCP/UDP ports
- 1-minute aggregate, keep for 7 days (3 GB)
- 5-minute aggregate, keep for 90 days (15 GB)
- 1-hour aggregate, keep for 5 years (estimate 200 GB)
- Unlike RRD, max/min values are lost
- Akvorado chooses the best table to query

Materialized view for aggregated table

```
CREATE MATERIALIZED VIEW flows_1h0m0s_consumer TO flows_1h0m0s
SELECT * EXCEPT (SrcAddr, DstAddr, SrcPort, DstPort)
         REPLACE toStartOfInterval(TimeReceived, toIntervalSecond(3600))
           AS TimeReceived
FROM flows
```

Exporters table

- Goal: get a list of exporters and interfaces for completion

```
CREATE MATERIALIZED VIEW exporters
(
    `TimeReceived` DateTime,
    `ExporterAddress` LowCardinality(IPv6),
    `ExporterName` LowCardinality(String),
    `IfName` String, `IfDescription` String,
)
ENGINE = ReplacingMergeTree(TimeReceived)
ORDER BY (ExporterAddress, IfName)
SELECT DISTINCT
    TimeReceived, ExporterAddress, ExporterName,
    [InIfName, OutIfName][num] AS IfName,
    [InIfDescription, OutIfDescription][num] AS IfDescription,
FROM flows ARRAY JOIN arrayEnumerate([1, 2]) AS num
```

Dictionaries

free

- AS numbers to names (used during queries)
- protocol numbers to names (UDP, TCP, ...) (used during queries)
- networks to name, role, region, tenant (used during ingestion)

```
SELECT * FROM asns WHERE asn = 12322
```

asn	name
12322	Free SAS

```
SELECT * FROM protocols WHERE proto = 17
```

proto	name	description
17	UDP	User Datagram Protocol

```
SELECT * FROM networks WHERE isIPAddressInRange('::ffff:88.120.156.11', network)
```

network	name	role	site	region	tenant
::ffff:88.120.0.0/109		customers		france	ftth

Network classification during ingestion

```
CREATE MATERIALIZED VIEW flows_2_raw_consumer TO flows
SELECT
    *,
    dictGetOrDefault('networks', 'name', SrcAddr, '') AS SrcNetName,
    dictGetOrDefault('networks', 'name', DstAddr, '') AS DstNetName,
    dictGetOrDefault('networks', 'role', SrcAddr, '') AS SrcNetRole,
    dictGetOrDefault('networks', 'role', DstAddr, '') AS DstNetRole,
    dictGetOrDefault('networks', 'site', SrcAddr, '') AS SrcNetSite,
    dictGetOrDefault('networks', 'site', DstAddr, '') AS DstNetSite,
    dictGetOrDefault('networks', 'region', SrcAddr, '') AS SrcNetRegion,
    dictGetOrDefault('networks', 'region', DstAddr, '') AS DstNetRegion,
    dictGetOrDefault('networks', 'tenant', SrcAddr, '') AS SrcNetTenant,
    dictGetOrDefault('networks', 'tenant', DstAddr, '') AS DstNetTenant
FROM flows_2_raw
```

free

User queries

The web interface allows a user to specify:

- time range
- columns (dimensions)
- filter expression

Filter expression

- Looks like the WHERE part of a SQL query
- Translated to ClickHouse SQL using a PEG parser

```
ExporterAddress=203.0.113.1
AND SrcAS NOTIN (AS12322, AS29447)
AND EType = ipv4
AND InIfProvider = "cogent"
AND DstAddr << 203.0.113.0/24
```

```
WHERE ExporterAddress = toIPv6('203.0.113.1')
AND SrcAS NOT IN (12322, 29447)
AND EType = 0x800
AND InIfProvider = 'cogent'
AND DstAddr BETWEEN toIPv6('203.0.113.0') AND toIPv6('203.0.113.255')
```

User query to ClickHouse query

- ClickHouse helps a lot to return directly exploitable data

```
WITH rows AS (SELECT SrcAS
    FROM flows_5m0s WHERE (timefilter) AND (userfilter)
  GROUP BY SrcAS ORDER BY SUM(Bytes) DESC LIMIT 10
) SELECT
  toStartOfInterval(TimeReceived, INTERVAL 600 second) AS t,
  SUM(Bytes*SamplingRate*8/600) AS xps,
  if(SrcAS IN rows, [concat(toString(SrcAS), ': ',
    dictGetOrDefault('asns', 'name', SrcAS, '???'))], ['Other']) AS dimensions
FROM flows_5m0s
WHERE (timefilter) AND (userfilter)
GROUP BY t, dimensions
ORDER BY t WITH FILL
  FROM toStartOfInterval((timefilter.start), INTERVAL 600 second)
  TO (timefilter.end) STEP 600
```

free

Go bindings

- Use github.com/ClickHouse/clickhouse-go/v2
- Native client-server protocol
- Unit tests by generating a mock with gomock

```
mockConn.EXPECT().
  Select(gomock.Any(), gomock.Any(), `
SELECT DISTINCT name AS attribute
FROM networks
WHERE positionCaseInsensitive(name, $1) >= 1
ORDER BY name
LIMIT 20`, "c").
  SetArg(1, []struct {
    Attribute string `ch:"attribute"`
  }{{"customer-1"}, {"customer-2"}, {"customer-3"}}).
  Return(nil)
```

free

Migrations

- We don't expect users to know how to operate ClickHouse
- A component manages the ClickHouse tables
- Schema migrations are done with Go code
- Each migration step has a description, a test and a function
- No state: each migration step is executed
- Forward migration only

free

Steps

- create protocols dictionary
- create asns dictionary
- create networks dictionary
- create flows table with resolution X
- add more columns to flows table with resolution X
- create flows table consumer with resolution X
- configure TTL for flows table with resolution X
- create exporters view
- create raw flows table
- create raw flows consumer view

Migration for dictionaries and views

- We don't need to keep data.
- Check if the table is in its final state, otherwise destroy and create.

```
func queryTableHash(hash uint64, more string) string {
        return fmt.Sprintf(`
SELECT bitAnd(v1, v2) FROM (
 SELECT 1 AS v1
 FROM system.tables
 WHERE name = $1 AND database = currentDatabase() %s
) t1, (
 SELECT groupBitXor(cityHash64(name,type,position)) == %d AS v2
 FROM system.columns
 WHERE table = $1 AND database = currentDatabase()
) t2`, more, hash)
}
```

Migration for data tables

- We have to keep existing data, so we use mutations.

```
migrationStep{
  CheckQuery: `
SELECT 1 FROM system.columns
WHERE table = $1 AND database = currentDatabase() AND name = $2`,
  Args: []interface{}{tableName, "DstNetName"},
  Do: func() error {
    modifications := []string{
      `ADD COLUMN SrcNetName LowCardinality(String) AFTER DstAS`,
      `ADD COLUMN DstNetName LowCardinality(String) AFTER SrcNetName`,
    return conn.Exec(ctx, fmt.Sprintf(`ALTER TABLE %s %s`,
       tableName, strings.Join(modifications, ", ")))
}
```

free

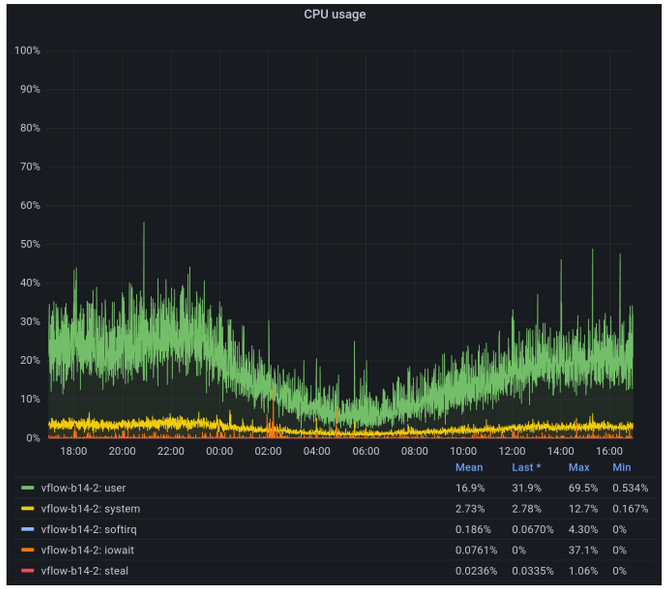
Testing migrations

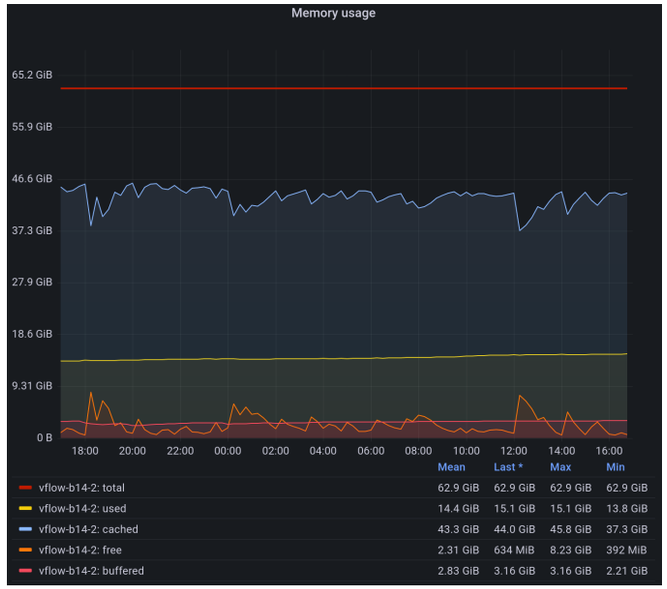
- Migrations are functionally tested from various states, including with an empty database
- Each test must get the same final state
- Each time a new migration step is added, the final state is recorded to be used for future tests

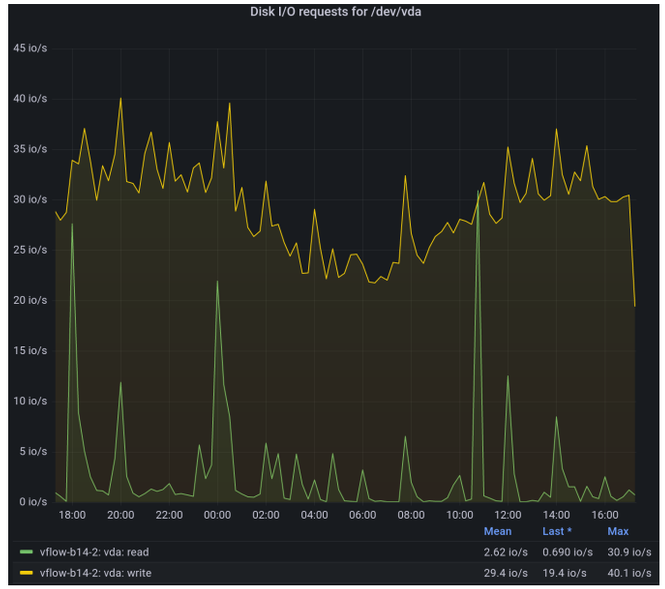
```
SELECT table, create_table_query
FROM system.tables
WHERE database=currentDatabase() AND table NOT LIKE '.%'
FORMAT CSV
```

Single node setup

- 1 single VM (also running Kafka and Akvorado)
- Docker Compose
- 1 TB of disk
- 64 GB of RAM
- 30k flows/second (target is 100k flows/second)







free

Opinions about ClickHouse

- Great out-of-the-box experience
- Great documentation
- Many builtin functions available
- Feel like magic: fast without much effort
- Aggregating merge trees take some time to understand how they work

free

Questions?