# Query, measure & alert

## on BGP state in real-time via GraphQL

Vasileios Kotronis | CTO, Code BGP | Dr. sc. ETH Zurich

vkotronis@codebgp.com


21 June 2023 | Bern

# About me



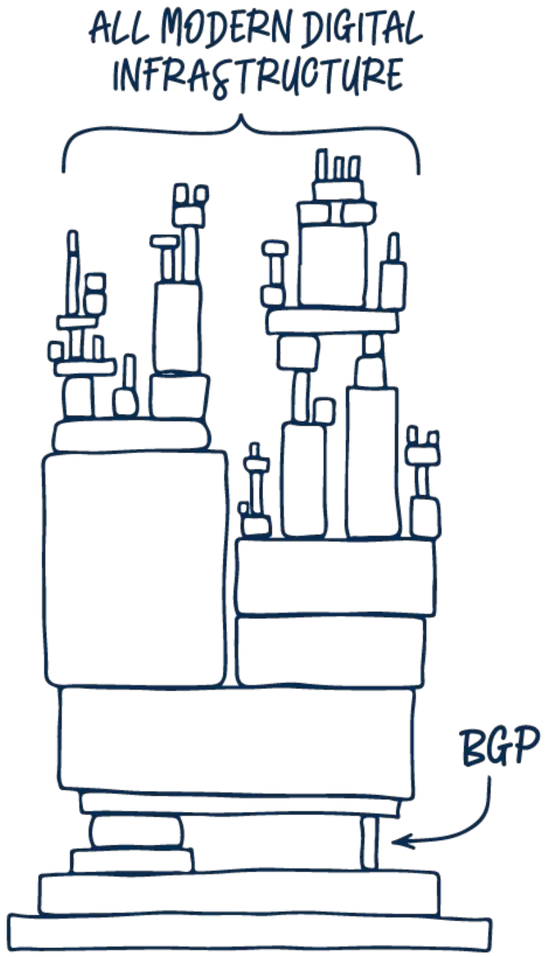
## Vasileios Kotronis

CTO & co-founder | Code BGP

vkotronis@codebgp.com

https://www.linkedin.com/in/vasileios-kotronis/

# BGP hijacks, leaks & route changes affect our networks

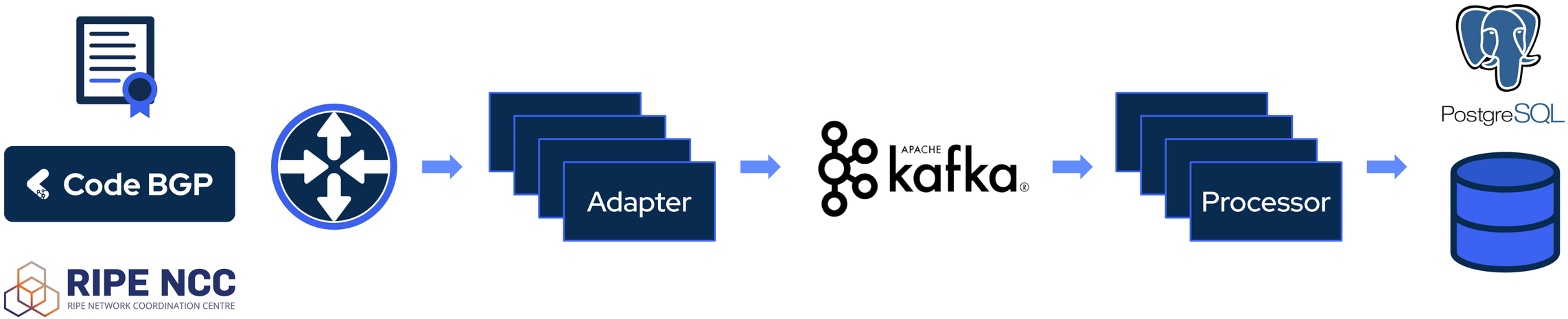



- Network teams are **blind** to what is happening with their Internet addresses and routes
  - Only the **tip of the iceberg** gets known
  - Routing events can critically affect:
    - **reliability**
    - **security**
    - **performance**

- Rapid action is **critical** when dealing with BGP outages
  - Detect events **in seconds**
  - Track the **current state** of the network (global routing system)
  - Analyze **on-going** events
  - **Automation**: immediate **programmatic** access to BGP data (**streaming APIs**)

# What we do: ingestion, processing, storage

- We collect/ingest BGP data (state) from real-time (streaming) data sources
  - From: Code BGP monitors, RIS Live, BGP/BMP sessions (your own routers), RPKI
  - Via: BGP, BMP, websockets, REST, etc.
- We process and store this state in real-time using a distributed event-driven mservice architecture



# What we do: looking glass

- We expose this state to the user (and other frameworks) in real-time via GraphQL (UI/API)

# What we do: looking glass

- We expose this state to the user (and other frameworks) in real-time via GraphQL (UI/API)

# GraphQL basics


https://graphql.org/

- **What it is**
  - Query language for APIs
  - Runtime for fulfilling queries with existing data

- **Features**
  - Ask exactly the data you need
  - Get many resources in single request
  - Single endpoint + type system: organized in terms of types and fields, not endpoints
  - No-version API evolution
  - Integration with own data + code

# GraphQL pros and cons


https://graphql.org/

- **Pros**
  - Speed + no over-fetching/under-fetching (ask and get exactly what you need)
  - Suitable for complex microservice-based systems (unified API)
  - Hierarchical structure
  - Data “shaping”
  - Strong typing
  - No “latest” version (Facebook use case)

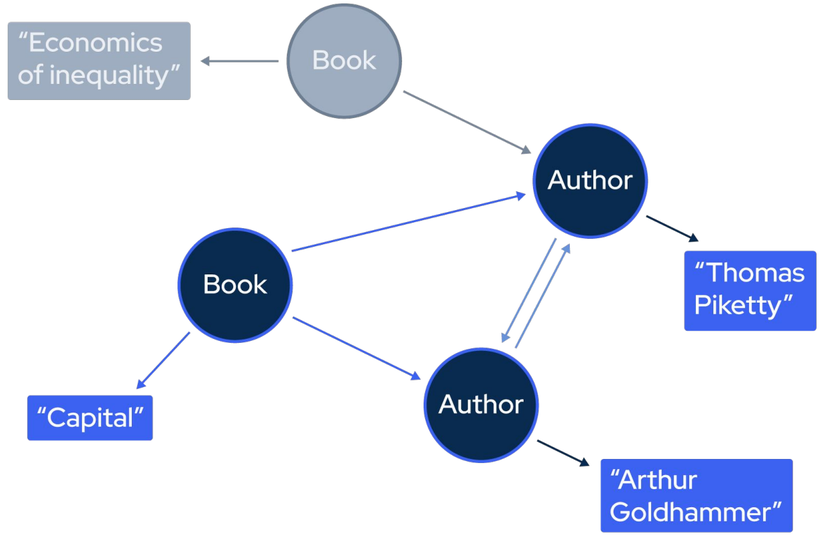
- **Cons**
  - Query complexity can be high → system load (query depth, recursion, etc.)
  - Complex caching (queries can be unpredictable, dynamic)
  - Complex rate-limiting

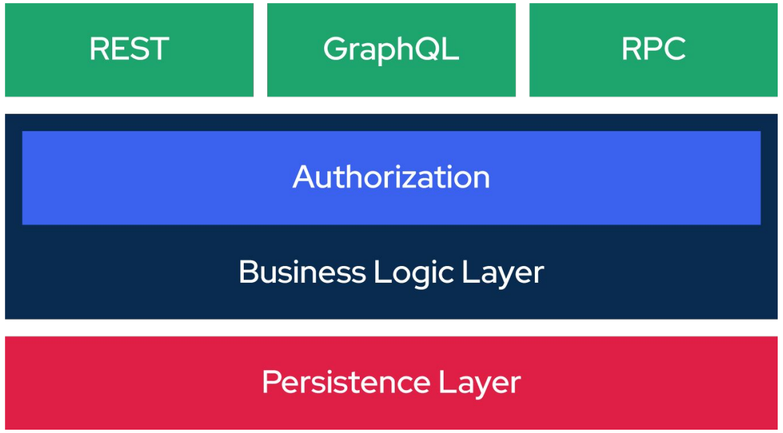
# GraphQL: thinking in graphs

GraphQL

https://graphql.org/

- Model your business domain as a “graph” by defining a schema
  - Within the schema, define:
    - different types of nodes
    - how they connect/relate to each other
  - Types may reference other types
    - e.g, a BGP route may reference a prefix or AS path
- Use GQL over your current business logic
  (do not implement it in GQL!)
- Treat your API as an expressive shared language
  - Express “how” clients consume the data
    (not “what” data)
  - Enable working with legacy data
- Expand/iterate the GQL schema gradually and frequently



# GraphQL concepts


https://graphql.org/

- Queries on objects fields, using optional (variable) arguments
- Directives for forming dynamic composite queries
- Mutations to modify server-side data
- Type system
  - queries/mutations
  - scalars
  - enums
  - interfaces
  - unions
- Type language: agnostic (use your favourite!)
- Queries/mutations validated and executed at run-time by GQL resolvers
- Introspection capabilities by design

# GraphQL type system


https://graphql.org/

```
type Prefixes {
  data_source_count: Int
  id: uuid
  ip_version: Int
  routes(
    distinct_on: [routes_select_column!]
    limit: Int
    offset: Int
    order_by: [routes_order_by!]
    where: routes_bool_exp
  ): [routes!]!
}
```

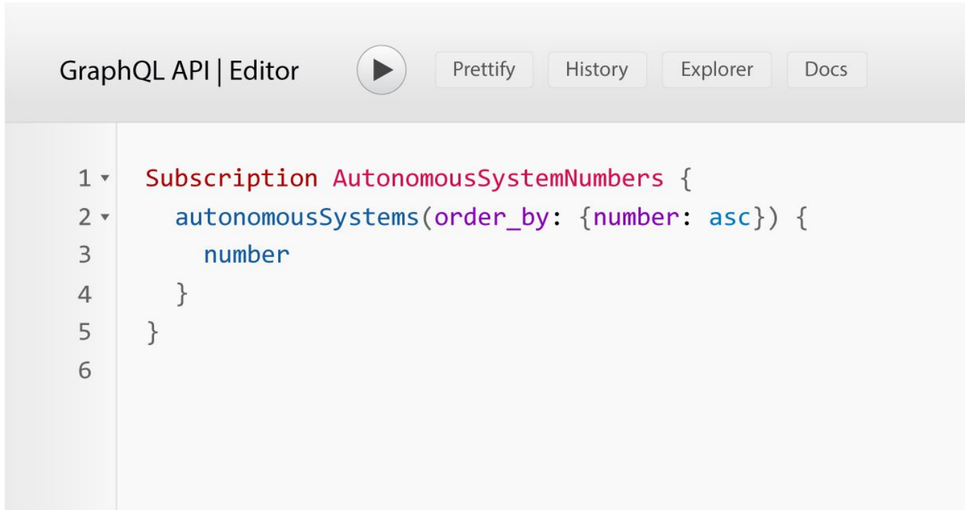
```
interface Identifiable {
  id: String!
}

type AutSystem implements Identifiable {
  id: String!
  number: Int!
}
```

# GraphQL subscriptions


https://graphql.org/

- GQL feature that allows a server to send data to its clients when a specific event happens
- Implemented with WebSockets
- Server maintains a steady connection to its subscribed client
- Breaks the “Request-Response-Cycle”
  - Client initially opens up a **long-lived connection** to the server
  - Sends a subscription query that specifies which event it is interested in
  - Every time this particular event happens, the server uses the connection to push the event data to the subscribed client(s).



# GraphQL best practices (I)


https://graphql.org/

- Serve over HTTP(S) via single endpoint
  - GET: `https://myapi/graphql?query={object{field}}`
  - POST:

```
{
  "query": "...",
  "operationName": "...",
  "variables": { "myVariable": "someValue", ... }
}
```

  - Response:

```
{
   "data": { ... },
   "errors": [ ... ]
}
```

- JSON syntax in responses (note that spec does not require it!)

# GraphQL best practices (II)


https://graphql.org/

- Versioning
  - Continuous evolution
  - Add/deprecate objects and fields
- Nullable/non-nullable types should be explicitly defined
- Authorization
  - Delegate to business logic layer (not the GQL layer!)
  - Frameworks like Hasura offer appropriate support for this
- Pagination: up to API designer (typically cursor-based)
- Batching & Caching: expose globally unique IDs for clients to use/cache on
- In general: most things besides the query contexts are kept out of the spec on purpose
  - Developer/operator freedom to implement own business logic!

# BGPQL: A GQL API for BGP data

GraphQL

https://graphql.org/

## Graph

- **Sample primitives**
  - dataSources
  - prefixes
  - autonomousSystems
  - peerings
  - routes
- **Sample associations/relationships**
  - dataSources → all
  - autonomousSystems → routes.Origin, routes.Neighbor, peerings.Left, peering.Right
  - prefixes → routes.prefix

## Query/Response

```
query MyV6Prefixes {
  prefixes(
    distinct_on: network
    where: {
     routes: {originAutonomousSystem: {number: {_eq: "50414"}},
     data_source_count: {_gte: 10}},
     ip_version: {_eq: 6}
    } order_by: {network: asc}
  ) {
    network
  }
}
```

```
{
  "data": {
    "prefixes": [
        {"network": "2a12:bc0::/48"},
        {"network": "2a12:bc0:1::/48"},
        {"network": "2a12:bc0:2::/48"}
    ]
  }
}
```

# An enabler: Hasura GraphQL engine


https://hasura.io/

- Objective: make data access fast, secure and reliable
- Automatically generates your GraphQL schema and resolvers based on tables/views in your database
  - auto-generate queries and mutations
  - accompany schema with actions, metadata, etc.
  - augment fields with DB-side functions (computed fields)
- You don't need to write a GraphQL schema or resolvers
- Supports PostgreSQL, MySQL, SQL Server and more
- Written in Haskell

# Hasura subscriptions


https://hasura.io/

- “Live” queries
- Client receives the complete updated state when value of any (queried) field changes upstream
- The result is the full answer to the query, as it is at the time of the change
- Example: “What are the visible AS paths originated by ASes $asns and related to prefix $prefix now?”

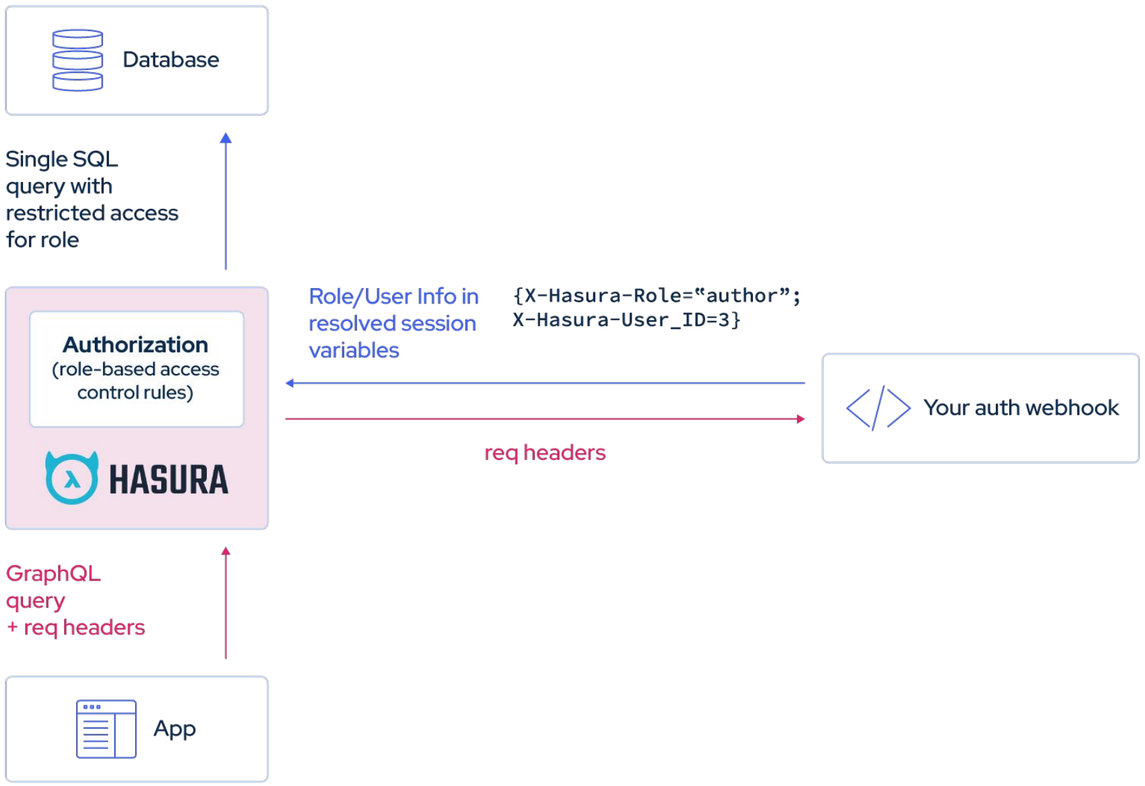
```
subscription PathsRelatedToPrefix($asns: [bigint!] = [], $prefix: cidr!) {
  routes(
    where: {prefix: {network: {_eq: $prefix}}, originAutonomousSystem: {number: {_in: $asns}}}
    distinct_on: as_path
    order_by: {as_path: asc_nulls_last}
  ) {
    as_path
  }
}
```

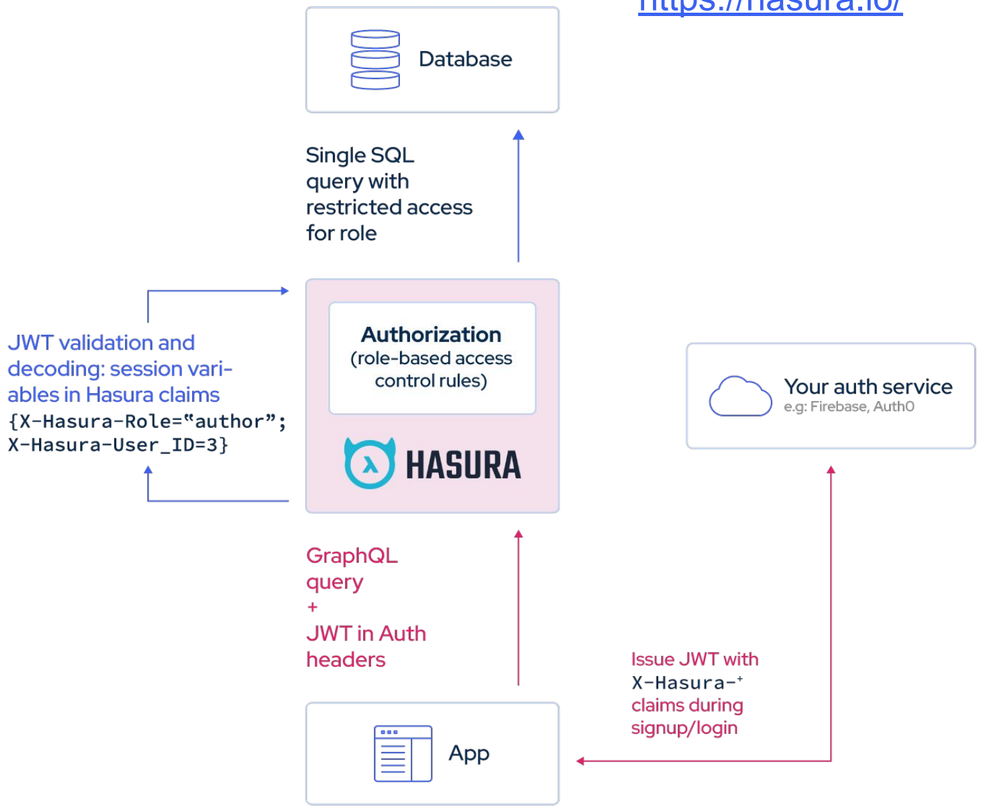
- Note: Hasura as of recently supports also streaming subscriptions
  - Streams the response according to the cursor provided by the user while making the subscription
  - Can be used to subscribe only to the data which has been newly added to the result set
  - Not covered in this presentation! (object identification implications)

# Hasura authentication & authorization

HASURA

https://hasura.io/



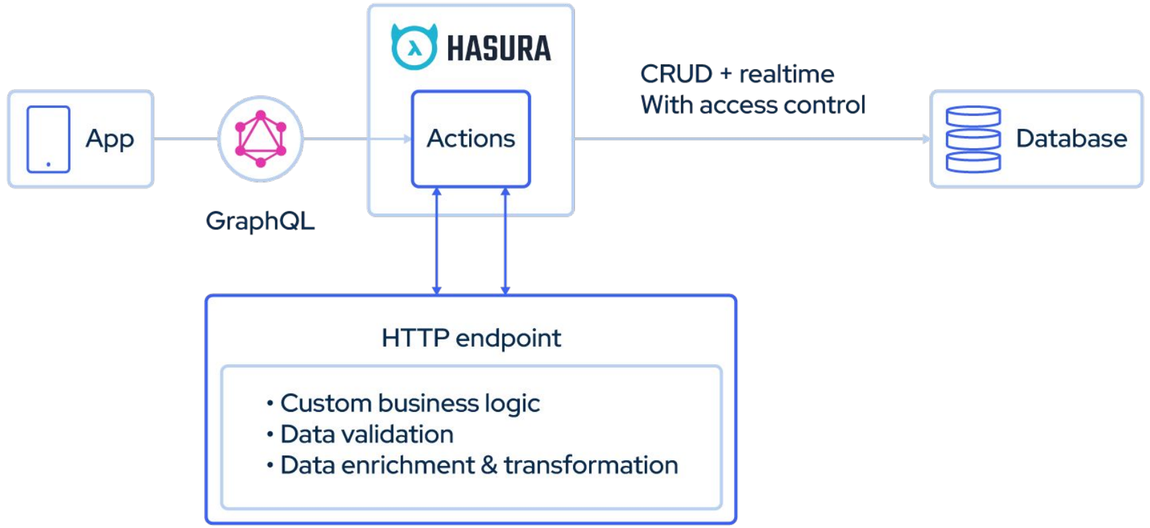



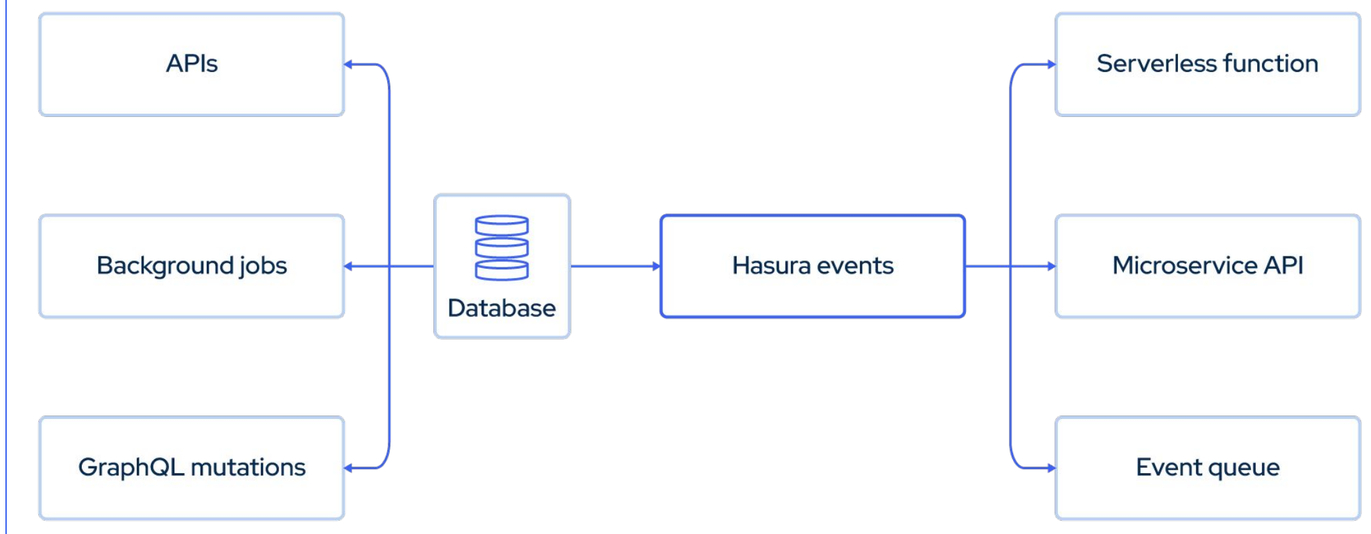

- RBAC supported via rules for select/insert/update/delete operations, using session variables in claims
- Role information is inferred from the `X-Hasura-Role` and `X-Hasura-Allowed-Roles` session variables
- Other session variables can be passed by your auth service as per your requirements

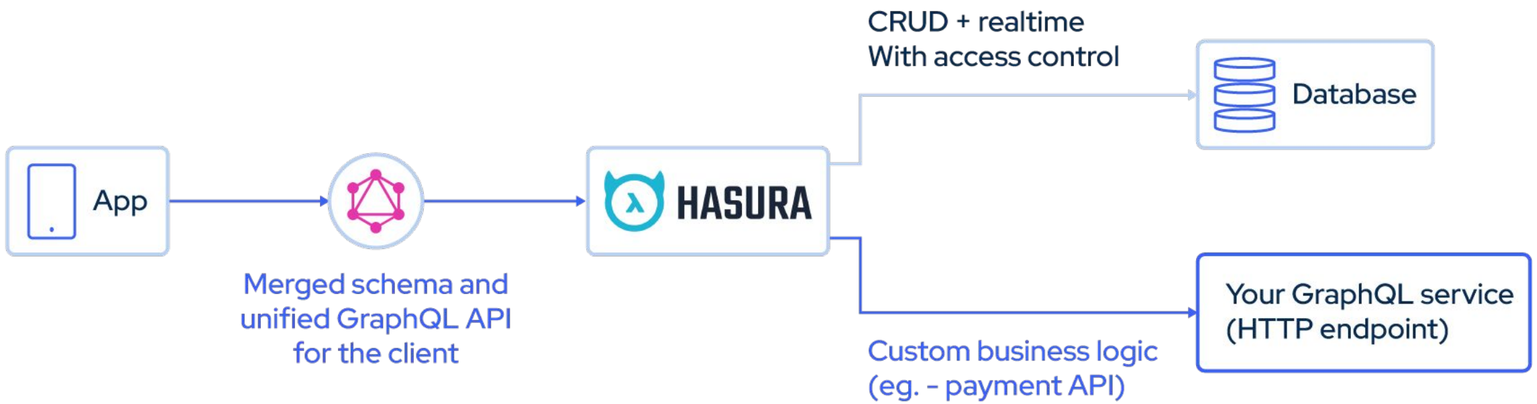
# Hasura actions, event triggers, remote schemas

HASURA

https://hasura.io/



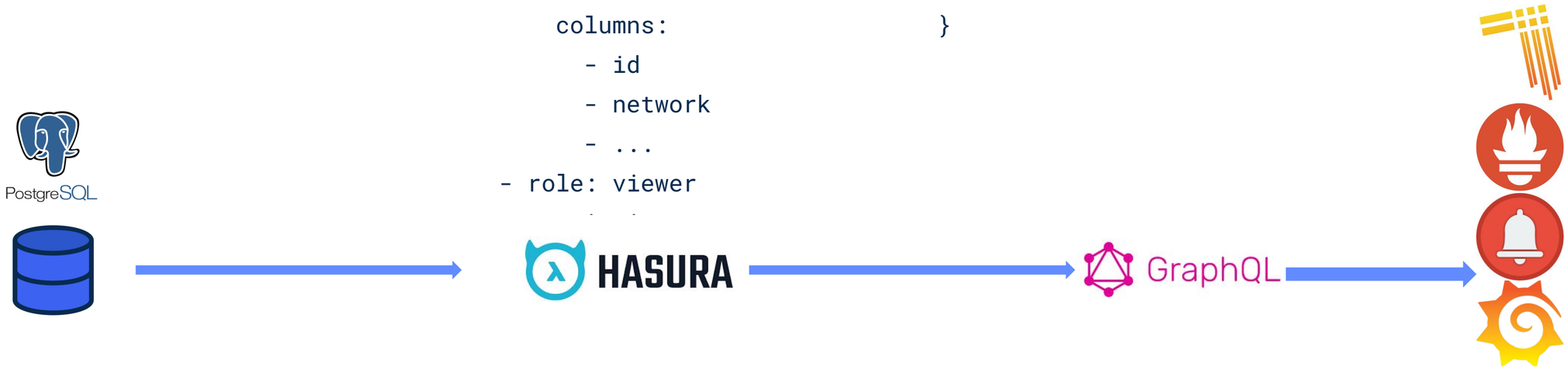



# Our use case: PostgreSQL → Hasura → GraphQL → applications

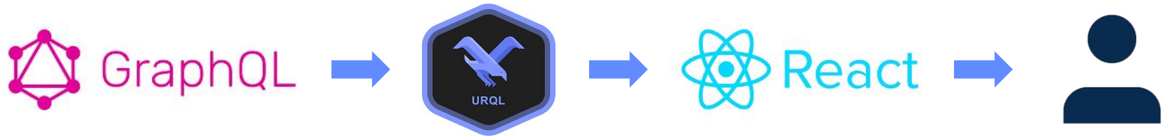
```
CREATE TABLE prefix (
  id uuid DEFAULT ext.uuid_generate_v4 (),
  network cidr NOT NULL,
  ip_version integer GENERATED ALWAYS AS
(family(network)) STORED,
  mask_length integer GENERATED ALWAYS AS
(masklen(network)) STORED,
  time_inserted timestamptz
);
...
```

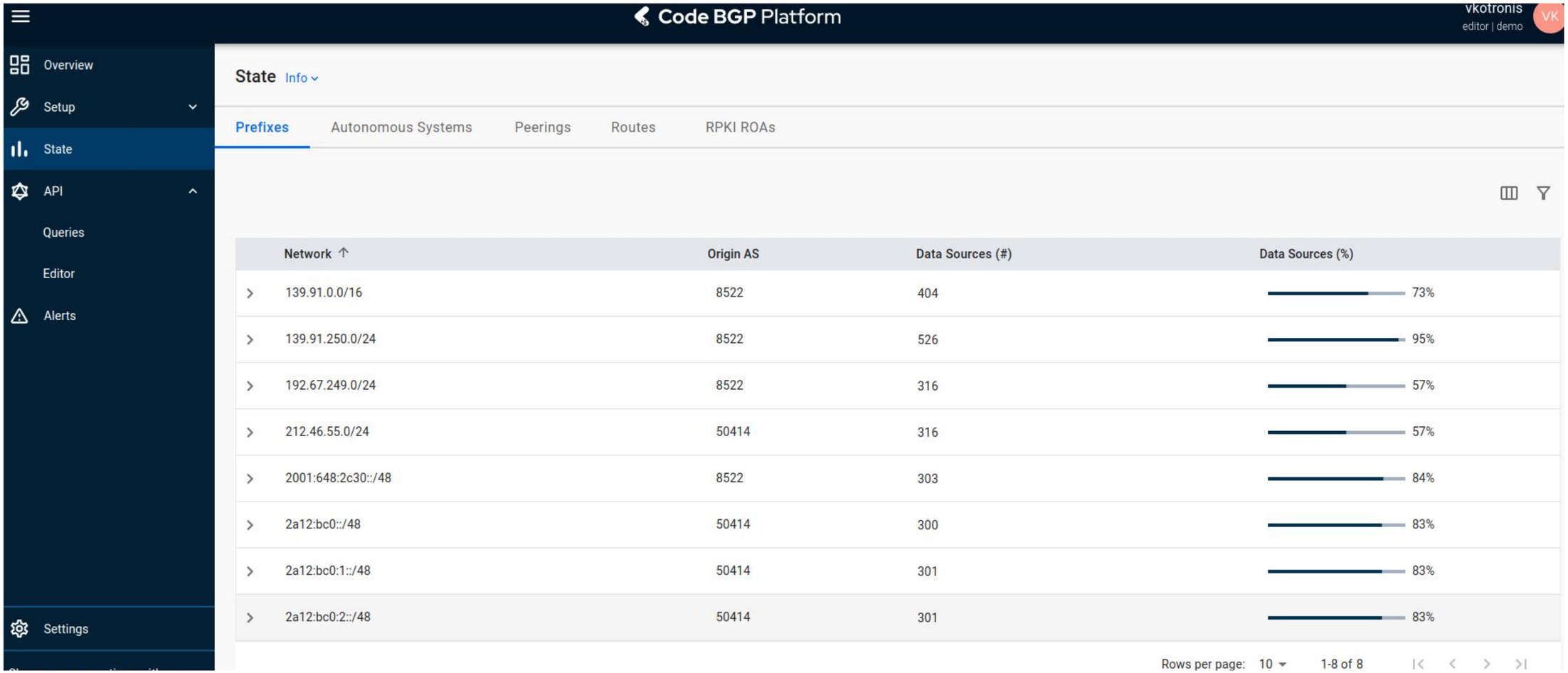
```
table:
  schema: main
  name: view_prefix
configuration:
  custom_name: prefixes
object_relationships: [...]
array_relationships: [...]
...
select_permissions:
  - role: editor
    permission:
      columns:
        - id
        - network
        - ...
  - role: viewer
```

```
query MyV6Prefixes {
  prefixes(
    distinct_on: network
    where: {
routes: {originAutonomousSystem: {number: {_eq:
"50414"}}, data_source_count: {_gte: 10}},
ip_version: {_eq: 6}
    }
  ) {
    network
  }
}
```

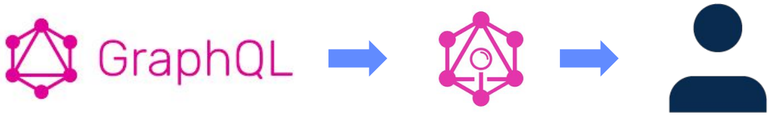


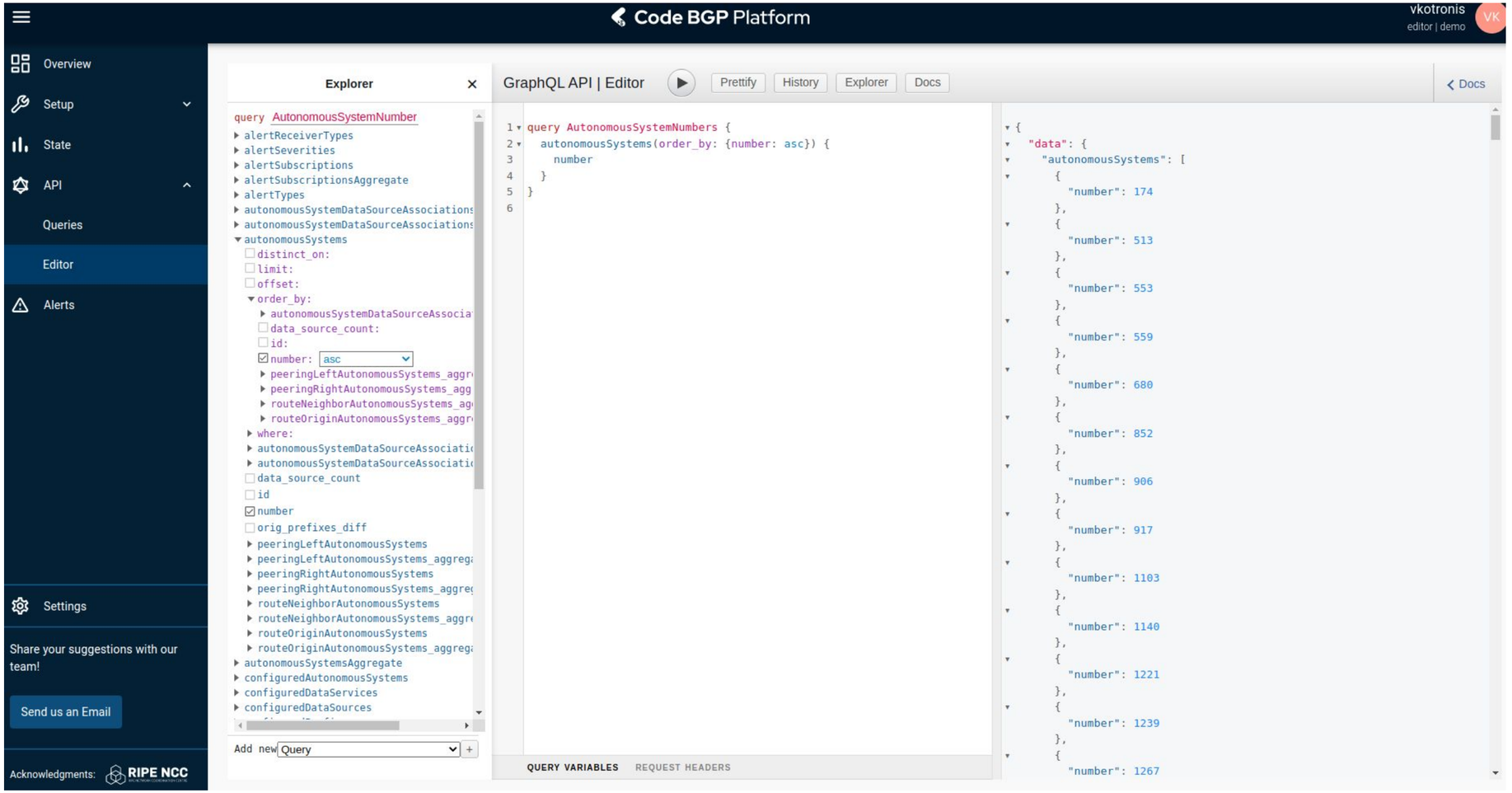
# Applications over GQL: UI



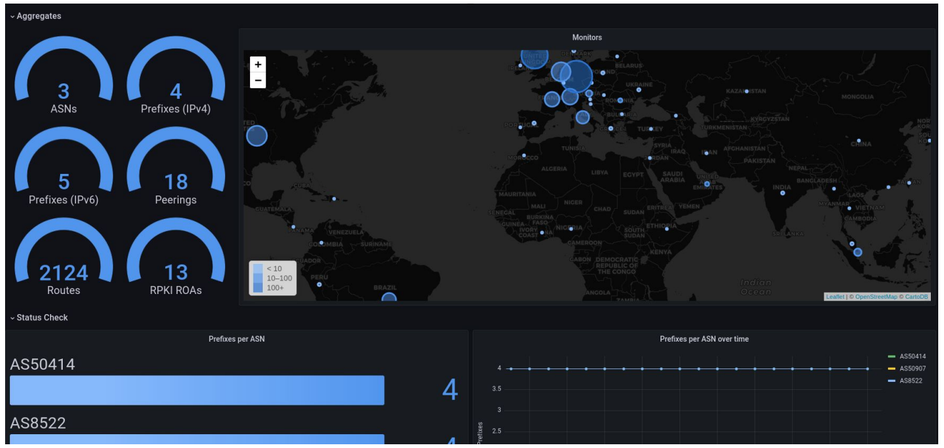
# Applications over GQL: API

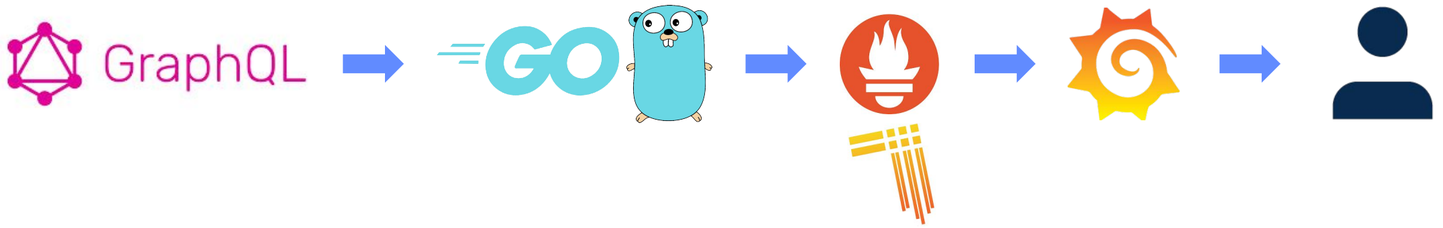


# Applications over GQL: Metrics

```
const prefixMetricQuery graphql.Query = `
subscription metricsproviderFilteredPrefixesOriginASFilteringQuery(
    $conf_prefixes: [String!], $conf_asns: [bigint!]
) {
    prefixes(where: {
        _or: [{configured_prefix_best_match: {_in: $conf_prefixes}},
              {routes: {originAutonomousSystem: {number: {_in: $conf_asns}}}}]
    }) {
        network
        ip_version
    }
}`
gaugeVec := promauto.NewGaugeVec(
    prometheus.GaugeOpts{
        Name: "filtered_prefixes_per_asn_total",
        Help: "The total number of prefixes per ASN, for the configured ASNs and prefixes",
    },
    []string{
        promKeyIPversion,
        promKeyAS,
    },
)
```



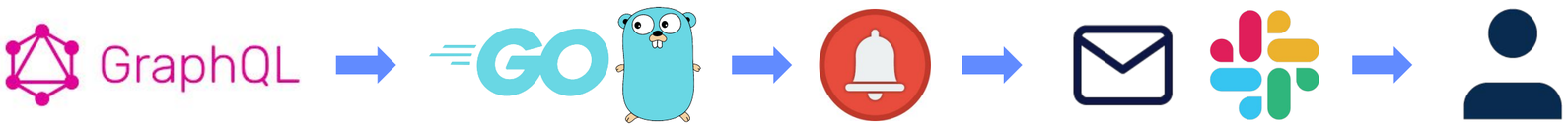
# Applications over GQL: Alerts

- Subscribe to **alertable subscriptions** on Go mservice(s)

```
type AlertSubscription struct {
    ID                string                 `json:"id"`
    Name              string                 `json:"name"`
    Query             string                 `json:"query"`
    Vars              map[string]interface{} `json:"vars"`
    FireAlertRegex    string                 `json:"fire_alert_regex"`
    AlertType         AlertType              `json:"alertType"`
    AlertSeverity     AlertSeverity          `json:"alertSeverity"`
    Description       string                 `json:"description"`
    ReceiverType      ReceiverType           `json:"alertReceiverType"`
    ReceiverEndpoint string                 `json:"receiver_endpoint"`
}
```

- If response data is actionable, e.g., matching a certain regex, **post to alertmanager API** `/api/v2/alerts`

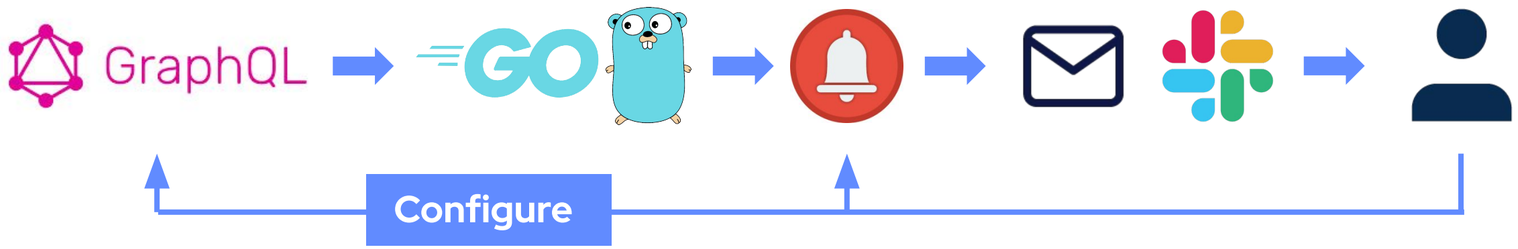
```
type Alert struct {
    StartsAt     string      `json:"startsAt,omitempty"`
    EndsAt       string      `json:"endsAt,omitempty"`
    Annotations  Annotations `json:"annotations"`
    Labels       Labels      `json:"labels"`
}
```



# Applications over GQL: Alerts

- Alertmanager **features**
  - handles alerts sent by client applications such as the Prometheus server
  - deduplication
  - grouping
  - routing to correct receiver integration
    - email, Slack, PagerDuty, OpsGenie, ...
  - silencing
  - inhibition
  - HA
- **Configuration**
  - routes
  - receivers
  - matchers
  - time intervals
  - inhibit/silence rules

```
route:
    groupWait: 10s
    groupInterval: 300s
    repeatInterval: 3600s
        groupBy:
            - alertname
            - severity
            - type
        matchers:
            - name: 'type'
                  matchType: '=~'
                  value: '(Route Leak)|(Exact Prefix Hijack)'
        receiver: 'email'
receivers:
        - name: 'email'
          emailConfigs:
            - to: '{{ .CommonLabels.receiver_endpoint }}'
                  ...
              headers:
                    - key: 'subject'
                      value: ...
                html: ...
```



# How we use GraphQL subscriptions for Alert Rules

- **Example** of a subscription query (which is entered to the system as a mutation) to detect exact prefix hijacks for prefixes belonging to Code BGP (AS 50414).
- No additional code needed, all the info is in the mutation!

```
mutation MutationExactPrefixHijack {
    insertAlertSubscription(object: {name: "Exact Prefix Hijack", query: "subscription IllegalOriginsFromWhichExactPrefixesAreAnnounced($asns:
[bigint!] = [], $prefixes: [cidr!] = []) { routes(where: {originAutonomousSystem: {number: {_nin: $asns}}, prefix: {network: {_in: $prefixes}}}
order_by:
{as_path: asc, prefix: {network: asc}, originAutonomousSystem: {number: asc}}) { originAutonomousSystem { number } prefix { network } as_path
}}", vars: {asns:[50414],
prefixes:["212.46.55.0/24","2a12:bc0::/48","2a12:bc0:1::/48","2a12:bc0:2::/48","2a12:bc0:3::/48","2a12:bc0:4::/48","2a12:bc0:5::/48"]},
fire_alert_regex: "^.*routes.*as_path.*$", type: "as_origin_violation_exact", severity: "critical", description: "Illegal origin ASes that
announce configured prefixes."}) {
      id
      name
      query
      vars
      fire_alert_regex
      type
      severity
      description
    }
}
```

# GQL alert rule | Example 01: Route Leak

- Query: `subscription LeakedPrefixesMyASNOriginates(`

```
    $asn: bigint!,
    $prefixes: [cidr!] = [],
    $ds_thres: Int!) {
    prefixes(where: {
        routes: {originAutonomousSystem: {number: {_eq: $asn}}},
        network: {_nin: $prefixes},
        data_source_count: {_gte: $ds_thres}
    } order_by: {network: asc}) {
        network
    }
}
```

- Variables: {

```
    asn: <asn>,
    prefixes: [<prefix_1>,...,<prefix_N>],
    ds_thres: <data_source_num_threshold>}
```

- Regex: `"^.*prefixes.*network.*$"`
- Description: Unexpected prefixes in the list of prefixes that are announced by configured ASes.

---ALERT START---
**Status**
Firing
**Started**
14:39:39 UTC 2023-02-14
**Ended**
No
**Severity**
Critical
**Name**
My Leak
**Type**
Route Leak
**Description**
Unexpected prefixes in the list of prefixes that are announced by configured ASes.
**Event**
Leaked prefixes: <leaked_prefix>
**Configured Resources**
AS<as> is configured to originate prefixes: <configured_prefix>, seen by at least <X> data sources.
---ALERT END---

# GQL alert rule | Example 02: Exact Prefix Hijack

- Query:
```
subscription IllegalOriginsFromWhichExactPrefixesAreAnnounced(
    $asns: [bigint!] = [], $prefixes: [cidr!] = []) {
        routes(where:
            {originAutonomousSystem: {number: {_nin: $asns}},
            prefix: {network: {_in: $prefixes}}
            } order_by: {
                prefix: {network: asc},
                originAutonomousSystem: {number: asc}
            }) {
            originAutonomousSystem {
                Number
            }
            prefix {
                Network
            }
        }
    }
```
- Variables: `{asns: [<asn_1>,...,<asn_K>], prefixes: [<prefix_1>,...,<prefix_N>]}`
- Regex: `"^.*routes.*originAutonomousSystem.*$"`
- Description: Illegal origin ASes that announce configured prefixes.

---ALERT START---
**Status**
Firing
**Started**
14:39:39 UTC 2023-02-14
**Ended**
No
**Severity**
Critical
**Name**
My Hijack
**Type**
Exact Prefix Hijack
**Description**
Illegal origin ASes that announce configured prefixes.
**Event**
AS<ash> has hijacked prefixes: <prefix>.
**Configured Resources**
AS<asv> are configured to originate prefixes: <prefix>.
---ALERT END---

# And many more can be expressed/supported!

| Supported Alert Types | Description |
|---|---|
| Exact Prefix Hijack | Illegal origin ASes that announce configured prefixes. |
| Sub-Prefix Hijack | Illegal origin ASes that announce subprefixes of configured prefixes. |
| Route Leak | Unexpected prefixes in the list of prefixes that are announced by configured ASes. |
| New Neighbor | New neighbors that appear to peer with configured ASes. Possible AS path manipulation. |
| Neighbor Leak/Hijack | New neighbors that not only appear to peer with configured ASes, but also propagate their prefixes. |
| Squatting | Illegal origin ASes announcing prefixes that are not currently announced by configured ASes. |
| Presence in AS Path | Presence of ASes in paths towards configured prefixes. |
| Invalid AS Path Pattern | Violation of valid pattern by AS paths towards configured prefixes. |
| Long AS Path | Paths towards configured prefixes exceed a specified length threshold. |
| Prefix Visibility Loss | Visibility of prefix falls below a configured data source count threshold. |
| Peering Visibility Loss | Visibility of peering falls below a configured data source count threshold. |

| Supported Alert Types | Description |
|---|---|
| RPKI-Invalid Detection | RPKI-Invalid announcements of configured prefixes by other ASes. |
| RPKI-Invalid Announcement | RPKI-Invalid announcements by configured ASes. |
| RPKI-Invalid Propagation | RPKI-Invalid routes propagated by configured ASes. |
| RPKI-NotFound Propagation | RPKI-NotFound routes propagated by configured ASes. |
| Bogon (Exact-)Prefix | Announcements of bogon prefixes by configured ASes. |
| Bogon (Sub-)Prefix | Announcements of bogon subprefixes by configured ASes. |
| Bogon AS | In-path presence of bogon ASes, in routes towards configured prefixes. |
| AS Path Comparison | Discrepancies in AS paths towards the same prefix, comparing between different Data Services, up to a terminating (end) AS. |
| Prefix Comparison | Discrepancies in prefixes announced by configured ASes, comparing between different Data Services. |
| Custom | User-defined |

# Summary

- **Ingest, process, store and query** streaming control-plane data in real-time
  - Expose stored data via GQL and subscribe to state changes (live queries or streams)
  - State changes are propagated in real-time to GQL subscription clients

- GQL offers **powerful primitives** to assist in the complex field of BGP and inter-domain routing
  - Strict type system to express data
  - Queries/Subscriptions/Mutations to access data
  - Data shaping and hierarchies
  - Unified API + single endpoint
  - Use case: BGPQL

# Summary

- **Distributed event-driven mservice streaming architectures** + **GQL**:
  - Programmatically ask operational questions
  - Drive network automation with a modern API
  - View real-time state updates in inter-domain routing
  - Generate useful metrics, like BGP update rates, aggregates, visibility artifacts, etc.
  - Be alerted and act on illegal changes (leaks, hijacks, etc.) even before BGP propagation ends!

# Thank you!

vkotronis@codebgp.com
codebgp.com

# Questions ❓

# Underlying software

BACKUP

Stack

















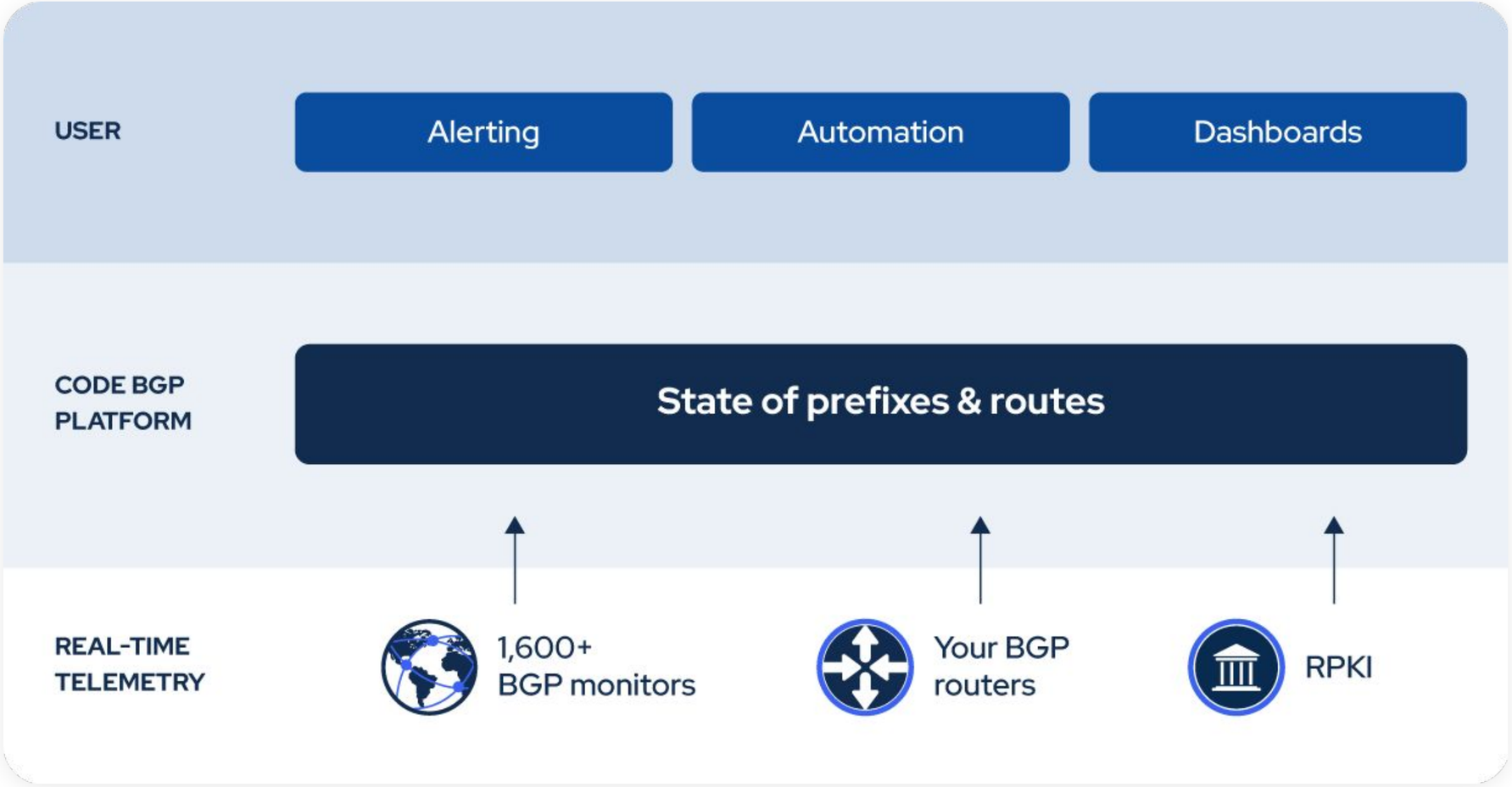





BACKUP

# What we have built: Code BGP Platform

Monitor • Detect • Protect



# About me

BACKUP


## Vasileios Kotronis

CTO & co-founder | Code BGP

vkotronis@codebgp.com

https://www.linkedin.com/in/vasileios-kotronis/