Taking SQL to the PromQL

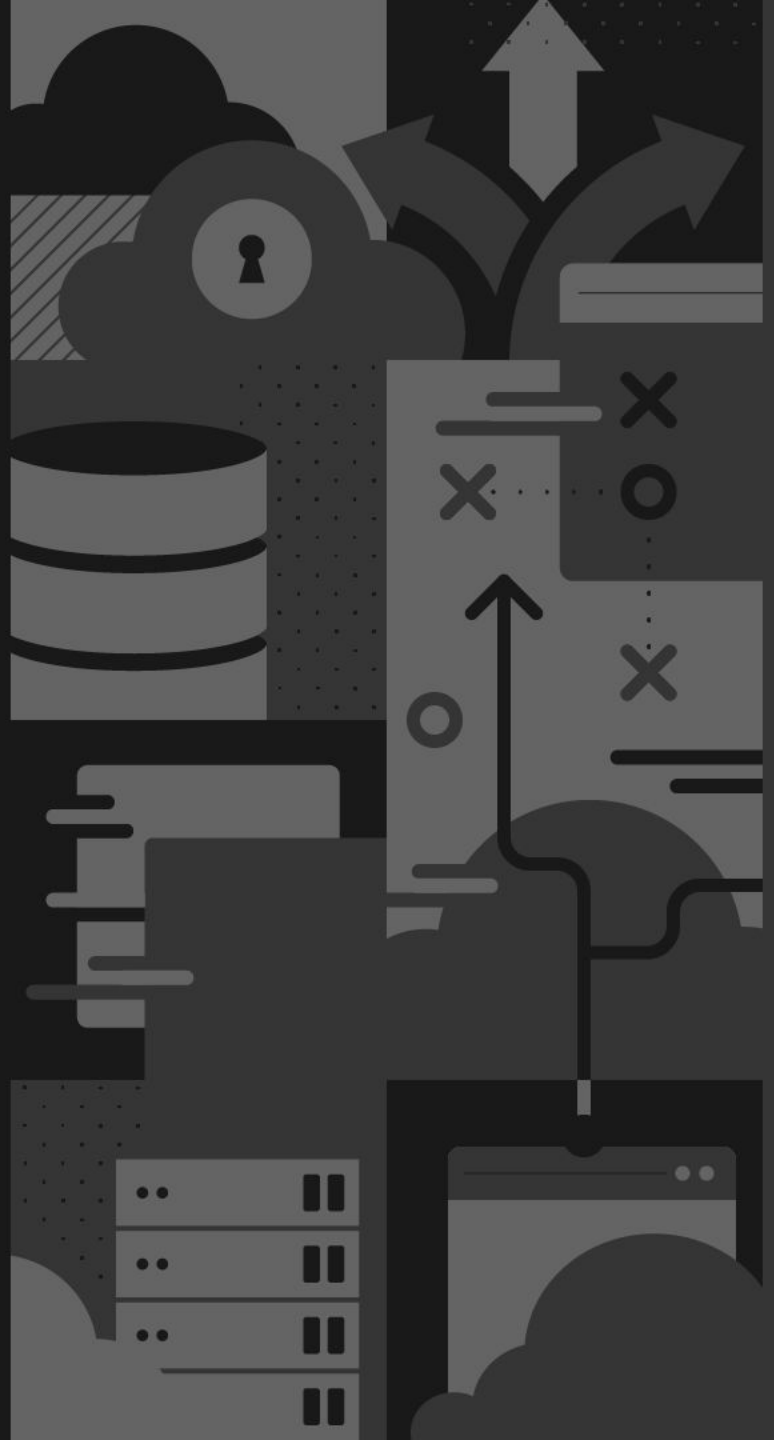
Kevin Howell
Principal Software Engineer

Lindsey Burnett
Senior Software Engineer

Red Hat

What we'll discuss today

- Why time series?
- Strings and Numbers
- Metrics
- Labels
- Instant Vectors
- Range Vectors
- Aggregations
- Grouping
- Recording Rules
- Analyzing PromQL



Why store data in time series?

Red Hat

Use the right tool for the job

Monitoring Use Cases
Prometheus was built for monitoring

Governance
Cloud Native Computing Foundation project

Industry Standard
"Prometheus has become the industry standard for monitoring applications and services" - Grafana

Source:
https://grafana.com/oss/prometheus/

Important Definitions

- SQL: Structured query language
- PromQL: Prometheus query language
- Instance Vector: data points from one or more time series at a single **instant** in time
- Range Vector: data points from one or more time series across a **range** of time.

http_requests_total{method="GET", path="/hello/world/1", code="200"} => 100

http_requests_total{method="GET", path="/hello/world/2, code="200"} => 150

http_requests_total{method="GET", path="/hello/world/3", code="200"} => 250

http_requests_total{method="POST", path="/hello/world", code="400"} => 75

Table Name: http_requests

Column Name	Data Type	Description
id	INT	Primary key, unique identifier for each request
method	VARCHAR(50)	HTTP request method (e.g., GET, POST)
path	VARCHAR(255)	Path of the HTTP request
code	INT	HTTP response code
timestamp	DATETIME	Timestamp of the request

id	method	path	code	timestamp
1	GET	/hello/world/1	200	2023-05-01 10:15:00
2	GET	/hello/world/2	200	2023-05-01 10:18:30
3	GET	/hello/world/1	200	2023-05-01 10:21:15
4	POST	/hello/world	400	2023-05-01 10:24:45
5	GET	/hello/world/3	200	2023-05-01 10:28:10

SQL

```
SELECT "timestamp", *
FROM http_requests
```

PromQL

```
http_requests_total
```

Result

- Retrieves data from the table/set of time series

http_requests_total{method="GET", path="/hello/world/1", code="200"} => 100

http_requests_total{method="GET", path="/hello/world/2, code="200"} => 150

http_requests_total{method="GET", path="/hello/world/3", code="200"} => 250

http_requests_total{method="POST", path="/hello/world", code="400"} => 75

SQL

```
SELECT "timestamp"
FROM http_requests
WHERE code='200'
```

PromQL

```
http_requests_total{code="200"}
```

Result

- Filters data based on conditions

http_requests_total{method="GET", path="/hello/world/1", code="200"} => 100

http_requests_total{method="GET", path="/hello/world/2", code="200"} => 150

http_requests_total{method="GET", path="/hello/world/3", code="200"} => 250

~~http_requests_total{method="POST", path="/hello/world", code="400"} => 75~~

SQL

```
SELECT count(*)
FROM http_requests
WHERE code='200'
```

PromQL

```
sum(http_requests_total{code="200"})
```

Result

- Aggregates matching rows or time series

http_requests_total{method="GET", path="/hello/world/1", code="200"} => 100

http_requests_total{method="GET", path="/hello/world/2", code="200"} => 150

http_requests_total{method="GET", path="/hello/world/3", code="200"} => 250

~~http_requests_total{method="POST", path="/hello/world", code="400"} => 75~~

SQL

```
SELECT count(*),code
FROM http_requests
GROUP BY code
```

PromQL

```
sum(http_requests_total)
by (code)
```

Result

- Groups data by an attribute's value

SQL

```
SELECT "timestamp",code
FROM http_requests
WHERE "timestamp" >
NOW() - INTERVAL '5
minutes'
```

PromQL

```
http_requests_total[5m]
```

Result

- Time-based range selection

SQL

- Use complicated window functions (e.g., LAG) to calculate the difference between consecutive rows over a specific time interval.

PromQL

- `rate(http_requests_total[5m])`

Result

- Computes the rate of change of a metric over time

Source:
https://www.postgresqltutorial.com/postgresql-window-function/postgresql-lag-function/
https://mode.com/sql-tutorial/sql-window-functions/

SQL

```
SELECT *
FROM http_requests
INNER JOIN http_errors
  ON http_requests.id =
http_errors.id
```

PromQL

```
rate(http_errors_total[5m]) /
on(request_id)
rate(http_requests_total[5m])
```

Result

- Combines data from two related data sets

SQL

```
SELECT *
FROM http_requests
INNER JOIN http_errors
  ON http_requests.id =
http_errors.id
```

PromQL

```
rate(http_errors_total[5m]) /
ignoring(code, path)
rate(http_requests_total[5m])
```

Result

- Combines data from two related data sets

SQL

```
SELECT *
FROM http_requests
INNER JOIN http_errors
  ON http_requests.id =
http_errors.id
```

PromQL

```
rate(http_errors_total[5m]) /
ignoring(code)
group_left
rate(http_requests_total[5m])
```

Result

- Combines data from two related data sets, group_left/group_right necessary when the data sets have different cardinality

HTTP Query Endpoint

- GET /api/v1/query
  - timestamp: find data close to this time (within 5 minutes prior to this timestamp)
- Depending on the query, can return:
  - Data points and labels for a **single** timestamp for one or more time series (Instance Vector)
  - Data points and labels for **multiple** timestamps for one or more time series (Range Vector)
  - A single data point (Scalar or String)

```
curl -XGET 'http://prometheus-server:9090/api/v1/query' \
 -d 'query=http_requests_total{path="/hello/world"}'
```


```
{
 "status": "success",
 "data": {
   "resultType": "vector",
   "result": [
     {
       "metric": {
         "__name__": "http_requests_total",
         "method": "GET",
         "path": "/hello/world",
         "instance": "example.com:9090",
         "code": 200
       },
       "value": [1623456000, "100"]
     },
     {
       "metric": {
         "__name__": "http_requests_total",
         "method": "POST",
         "path": "/hello/world",
         "instance": "example.com:9090",
         "code": 400
       },
       "value": [1623456100, "150"]
     }
   ] } }
```


Note

resultType: “vector” in the HTTP response indicates an Instance Vector

Red Hat

HTTP Query Range Endpoint

- GET /api/v1/query_range
  - start: Start timestamp, inclusive.
  - end: End timestamp, inclusive.
  - step: The amount of time between each datapoint
- Query repeated for each timestamp
  - Starting with **start**, increasing by **step** up to and including **end**
- Can only return data points and labels for **multiple** timestamps for one or more time series (Range Vector)

```
curl -XGET 'http://prometheus-server:9090/api/v1/query_range' \
 -d 'query=http_requests_total{method="GET", path="/hello/world"}' \
 -d 'start=1623456000' \
 -d 'end=1623456100' \
 -d 'step=10s'
```


```
{
 "status": "success",
 "data": {
   "resultType": "matrix",
   "result": [
     {
       "metric": {
         "__name__": "http_requests_total ",
         "method": "GET",
         "path": "/hello/world",
         "instance": "example.com:9090 "
       },
       "values": [
         [1623456000, "100"],
         [1623456010, "110"],
         [1623456020, "120"],
         [1623456030, "130"],
         [1623456040, "140"],
         [1623456050, "145"],
         [1623456060, "148"],
         [1623456070, "152"],
         [1623456080, "155"],
         [1623456090, "158"],
         [1623456100, "160"]
       ]
     }
   ]
 }
}
```


Note

resultType: “matrix” in the HTTP response indicates a Range Vector

Recording Rules

- Similar to a materialized view, you can have Prometheus synthesize the results of query as a time series.

```
groups:
 - name: example
   rules:
   - record: code:prometheus_http_requests_total:sum
     expr: sum by (code) (prometheus_http_requests_total)
```

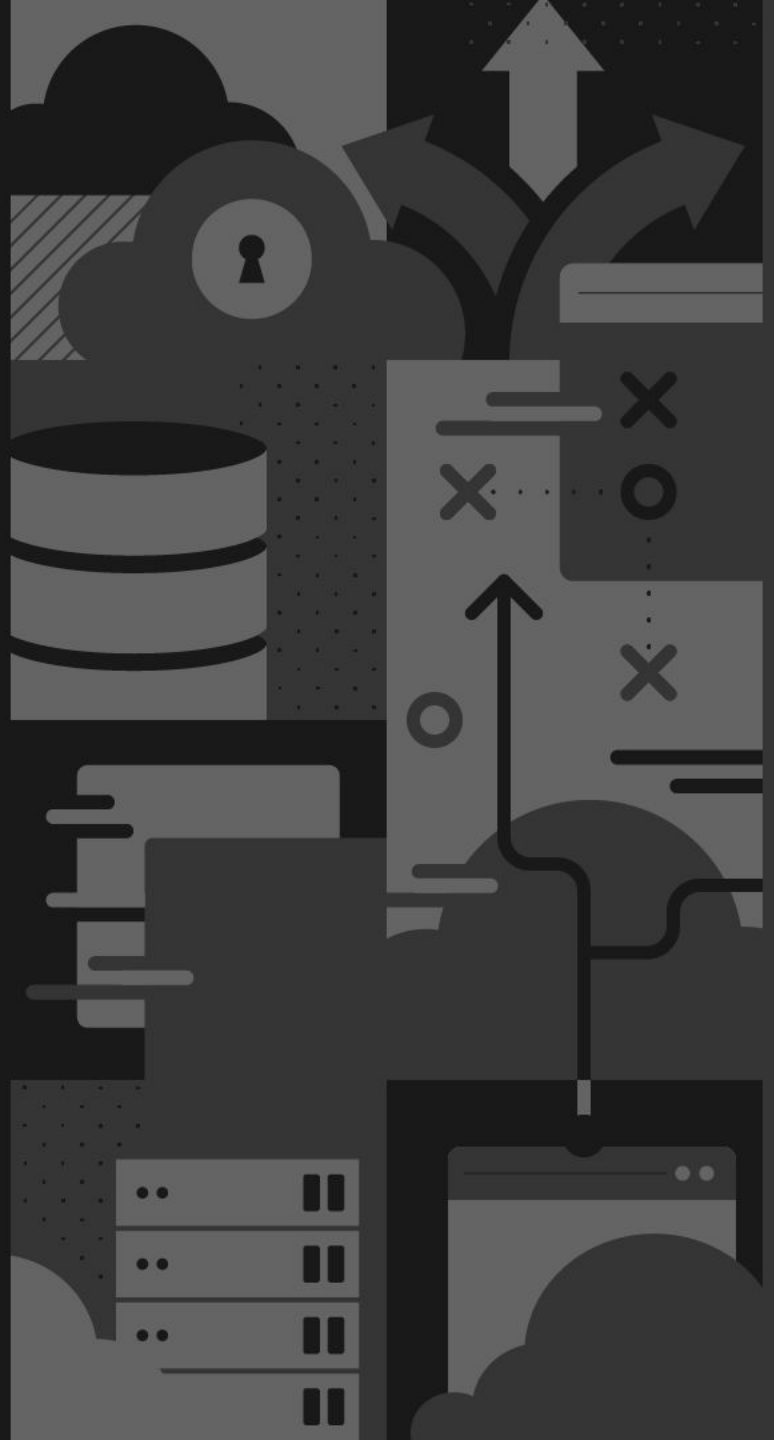
Advice for Analyzing PromQL

- Start at the innermost expressions and work outwards
- Use comments to keep notes; PromQL supports comments and multi-line queries

```
# calculate max velocity over the previous 10 minutes
max_over_time(
   # calculate its derivative (velocity)
   deriv(
       # calculate its rate over the previous 30 seconds, w/ 5 sec samples
       rate(
           # look at the previous 5 seconds of distance covered
           distance_covered_total[5s]
       )[30s:5s]
   )[10m:]
)
```

Resources

- Prometheus documentary: https://www.youtube.com/watch?v=rT4fJNbfe14
- Documentation: https://prometheus.io/docs/prometheus/latest/querying/basics/
- Book: Prometheus: Up & Running (O'Reilly)



Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat