KubeCon | CloudNativeCon

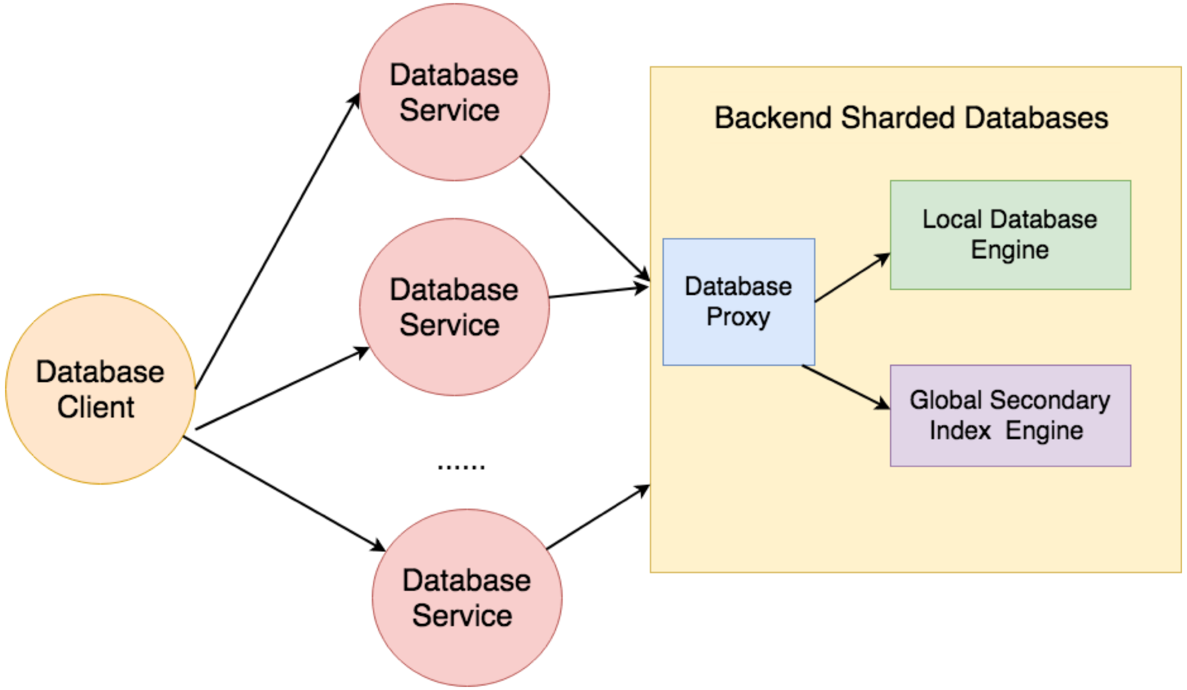
North America 2018

Sharded And Federated Prometheus Clusters to Monitor Distributed Databases

Jun Li and Viswa Vutharkar, eBay Inc.

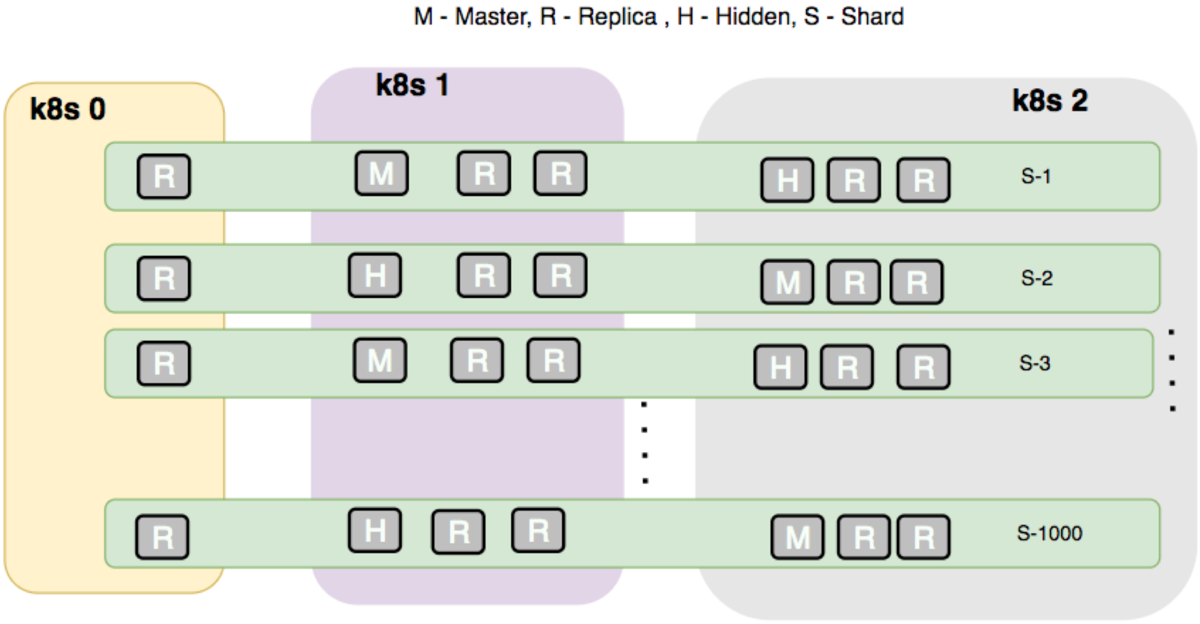
Monitoring a Geo-Distributed Database

- NuData: A geo-distributed database developed at eBay
- Deployment: Thousands of pods across datacenters in eBay internal Kubernetes based cloud infrastructure
- Metrics monitoring: Prometheus 2.3
- Real-time monitoring for:
  - System operation
  - System development



The Sharded Distributed Database

- The entire database framework supports many keyspaces
- A keyspace consists of multiple shards
- Each shard consists of multiple replicas (master, secondary, hidden)
- Replicas in each shard are provisioned across datacenters



Outline

- Metrics capturing and aggregation
- The Sharded and Federated Prometheus cluster
- Query routing & UI integration
- Monitoring experiences
- Conclusions

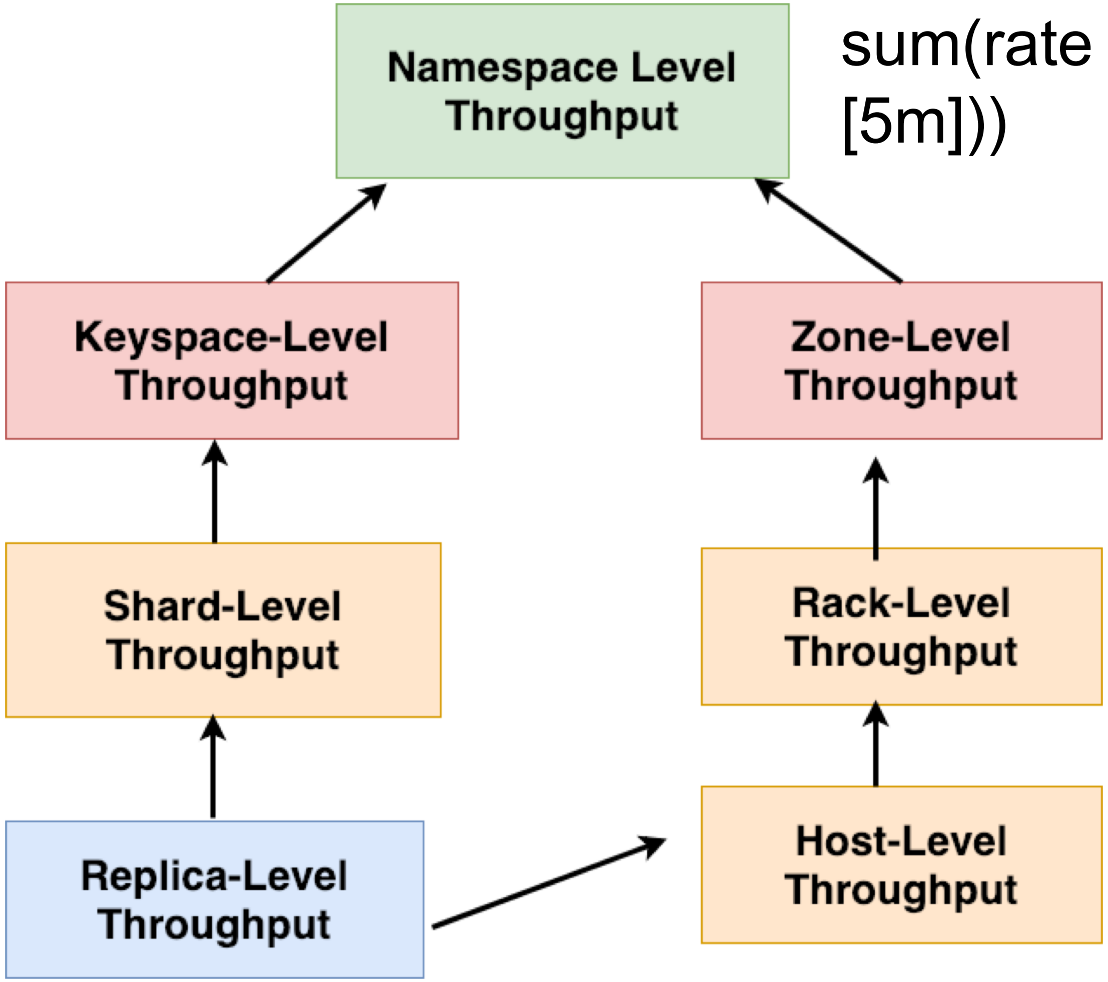
Metrics Being Monitored

- Metrics captured:
    - Throughputs
    - Latencies
    - Errors
    - Saturation (queuing)
    - State (master, replica, instance up/down)
- OS metrics (from Kubernetes Kubelet) and JVM metrics
- Custom metrics export: disk IO metrics from iostat
- Total metrics captured: **20M** metrics/scrape interval/per DC
    - Current scrape interval at 1 minute
    - Total storage size accumulated per day per DC: **195 GB**
    - Currently retain only **7** days of metrics data

Metric Labels & Labeling Hierarchy

- Prometheus on Kubernetes provides:
  - Dynamic discovery of target
  - Automatic label injection
  - Target filtering by auto-discovered labels
  - Re-labeling and label injection
- Metric labels:
  - Labels due to physical datacenter hierarchy:
    - pod, host, rack, datacenter
  - Labels due to logical database hierarchy:
    - replica, shard, keyspace
  - All of these labels are automatically injected by Prometheus from pod spec.

Hierarchical Multi-Label Metrics Aggregation



Namespace Level Throughput

sum(rate (db_processed_total [5m]))

Keyspace-Level Throughput

sum(rate (db_processed_total [5m])) by (keyspace)

Zone-Level Throughput

sum(rate (db_processed_total [5m])) by (keyspace, zone)

Shard-Level Throughput

sum(rate (db_processed_total [5m])) by (keyspace, shard)

Rack-Level Throughput

Host-Level Throughput

sum(rate (db_processed_total [5m])) by (keyspace, host)

Replica-Level Throughput

db_processed_total (zone=“dc1”, host=“…”, rack=“…”, pod=“…”, keyspace=“marketing”, shard=“78654”, replica=“21345”, type= “read”)

The Need for a Scalable Cluster

- To collect 20M metric samples/minute from a single Prometheus becomes prohibitive

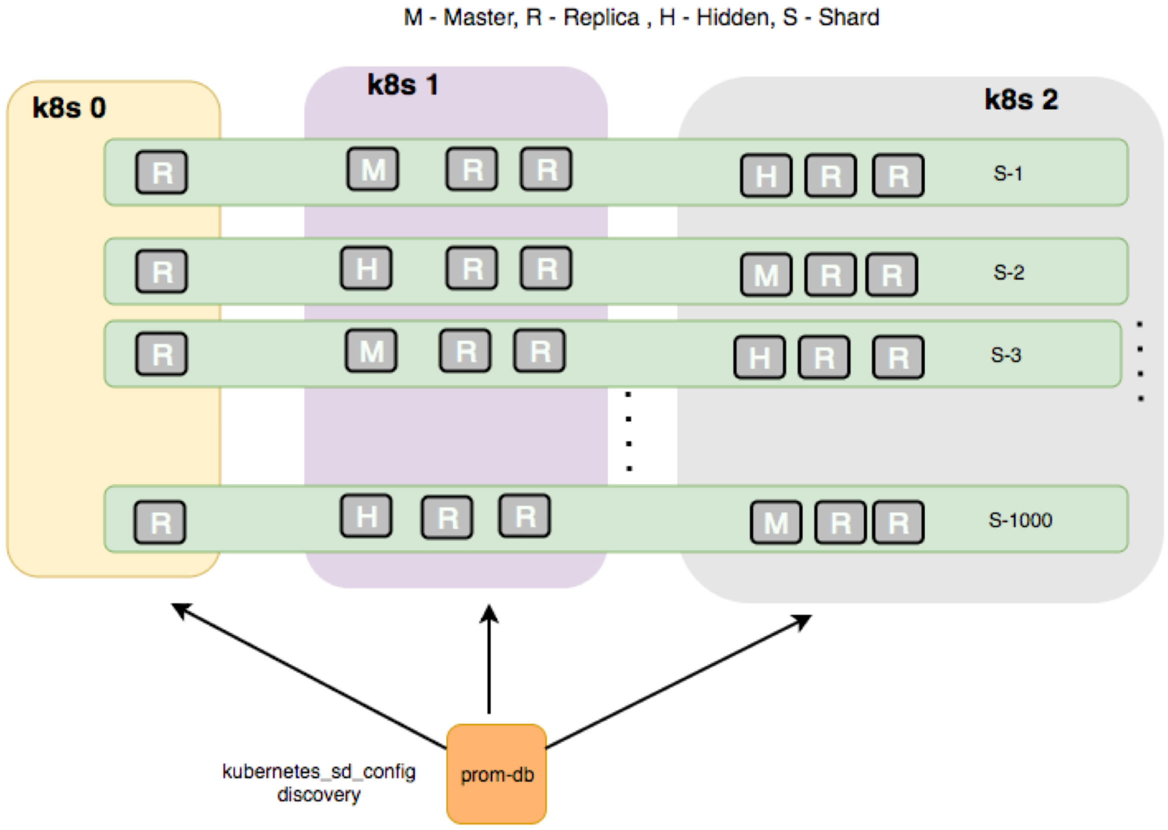
- In addition, >1200 recording rules in total to support real-time alerting and dashboards
  - One metric can be tied to multiple dashboards with different hierarchical aggregations

- The CPU consumption in Prometheus devoted to recording rules evaluation is much more significant compared to metrics scraping

Outline

- Metrics capturing and aggregation
- The Sharded and Federated Prometheus cluster
- Query routing & UI integration
- Monitoring experiences
- Conclusions

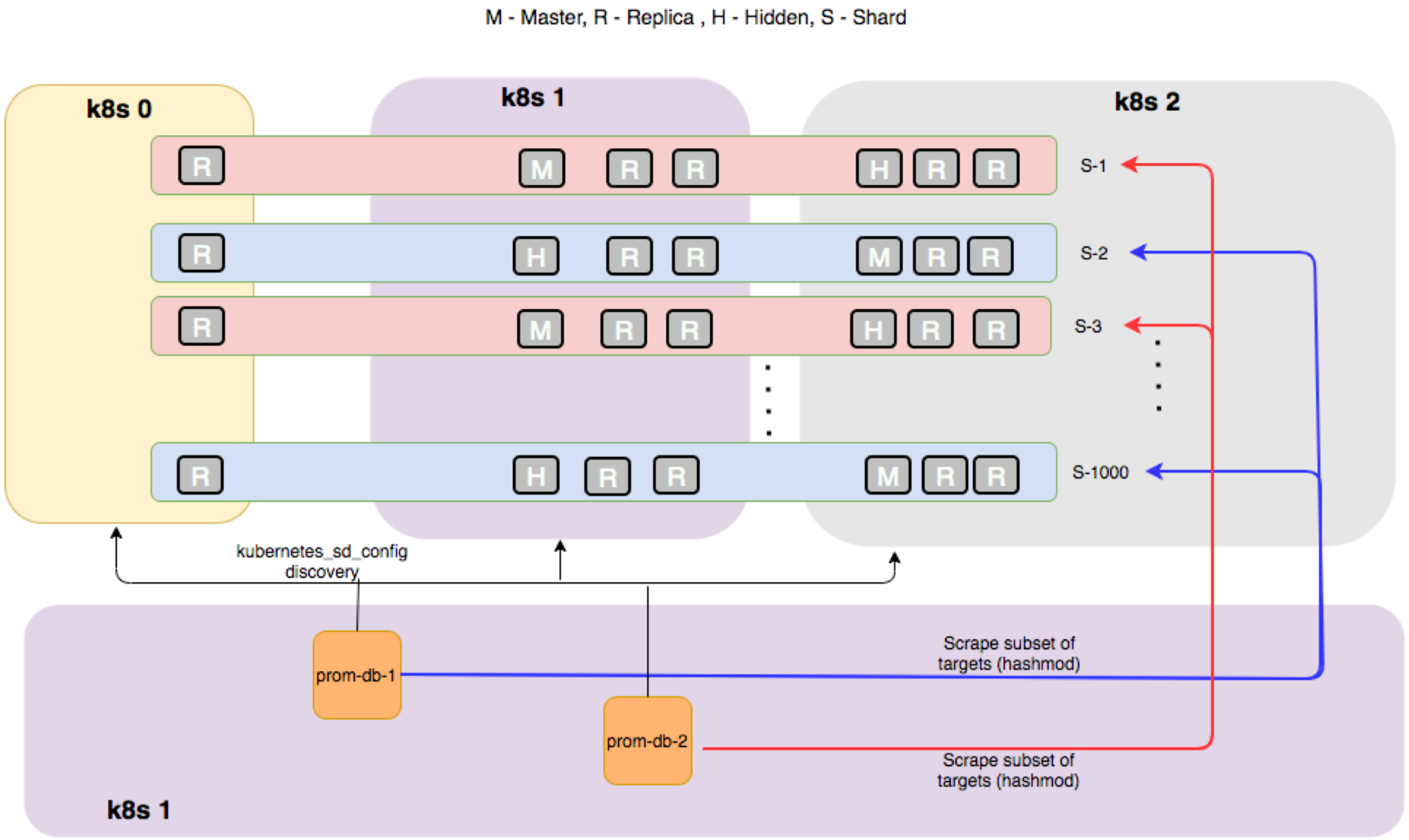
The Distributed Database Being Monitored



- Highly available distributed database across three DCs

- Need to have a Prometheus setup to linearly scale with the targets being scraped

Sharded Prometheus



- Sharded Prometheus setup
  - 2 Prometheus shards illustrated
  - “Even” numbered data shards scraped by prom-db-1
  - And “odd” numbered ones scraped by prom-db-2

- Generalization: hash and modulus using Prometheus hashmod relabel config
  - Hashing done on data shard ID (& keyspace)

Hashmod Relabel Config

```
…
-  source_labels: [__meta_kubernetes_pod_label_keyspace, __meta_kubernetes_pod_label_shard]
     action: hashmod
     modulus: __MODULUS__
     target_label:  hashmod
   - source_labels: [hashmod]
     regex: __SHARD__
     action: keep
...
```

prom-db-1.yaml

```
…
-  source_labels: [__meta_kubernetes_pod_label_keyspace, __meta_kubernetes_pod_label_shard]
     action: hashmod
     modulus: 2
     target_label:  hashmod
   - source_labels: [hashmod]
     regex: 0
     action: keep
...
```

prom-db-2.yaml

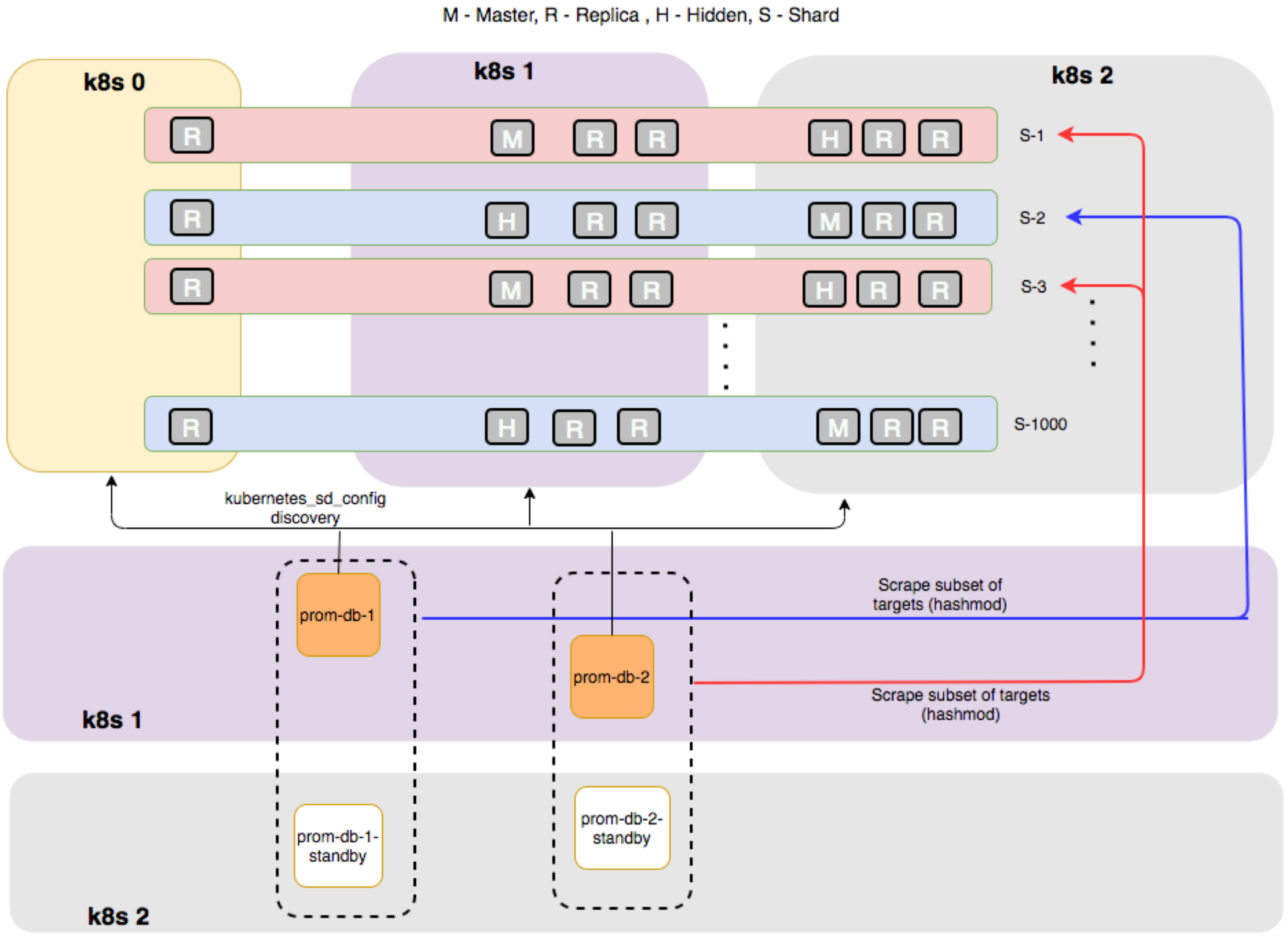
```
…
-  source_labels: [__meta_kubernetes_pod_label_keyspace, __meta_kubernetes_pod_label_shard]
     action: hashmod
     modulus: 2
     target_label:  hashmod
   - source_labels: [hashmod]
     regex: 1
     action: keep
...
```

- Our deployment scripts takes yaml template as input and generates prometheus yaml files

- “**Keyspace + shardID**” is the input to the hashmod function

- A nice side effect: **all replicas of a data shard** are scraped by the same Prometheus server

High Availability of Sharded Prometheus



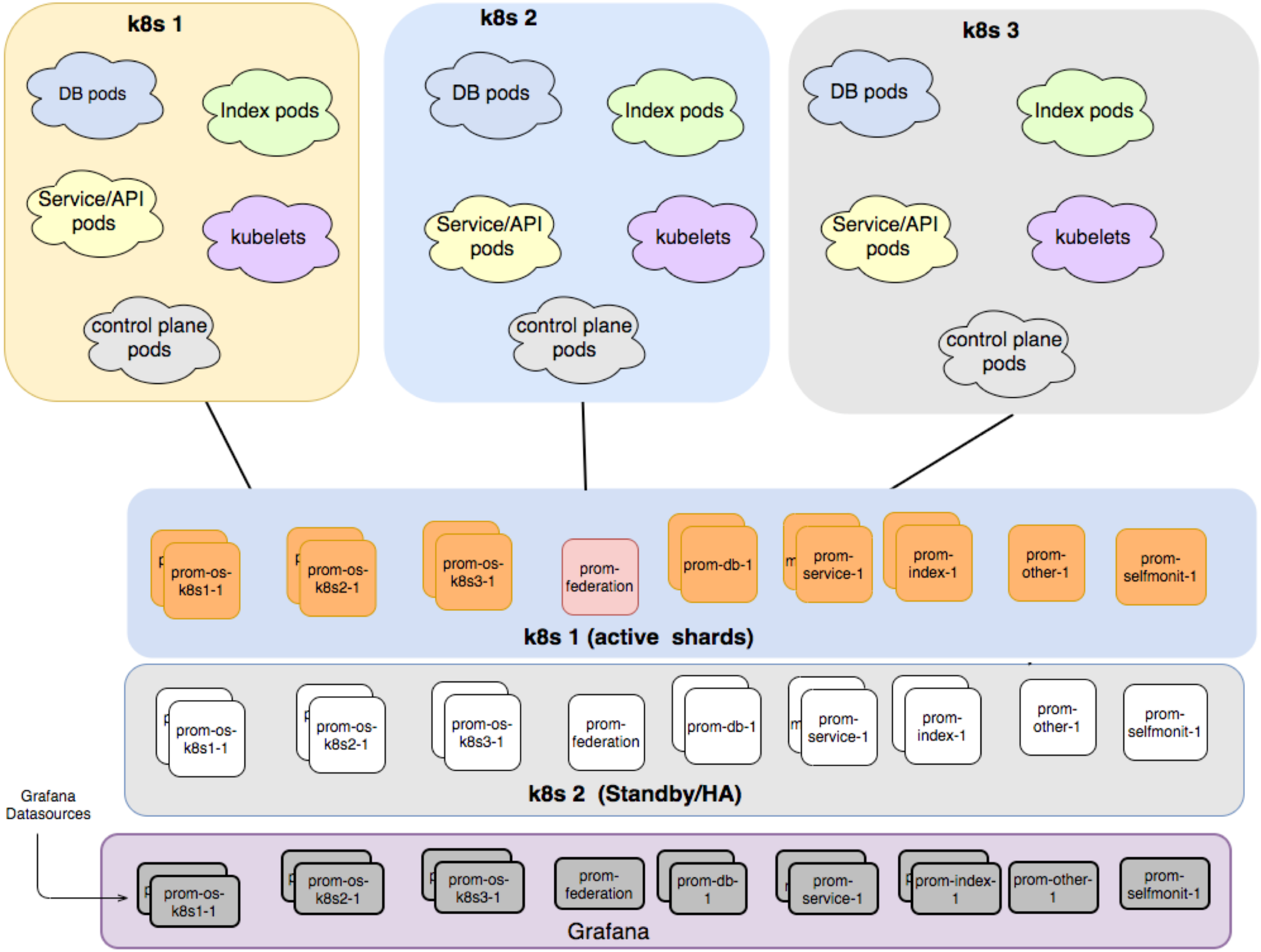
M - Master, R - Replica , H - Hidden, S - Shard
k8s 0
k8s 1
k8s 2
R
M
R
R
H
R
R
S-1
R
H
R
R
M
R
R
S-2
R
M
R
R
H
R
R
S-3
R
H
R
R
M
R
R
S-1000
kubernetes_sd_config discovery
prom-db-1
prom-db-2
Scrape subset of targets (hashmod)
Scrape subset of targets (hashmod)
k8s 1
prom-db-1-standby
prom-db-2-standby
k8s 2

- High availability: deploying the same set of Prometheus servers (mirrored config) in two clusters

- Active/standby configuration for each Prometheus server pair

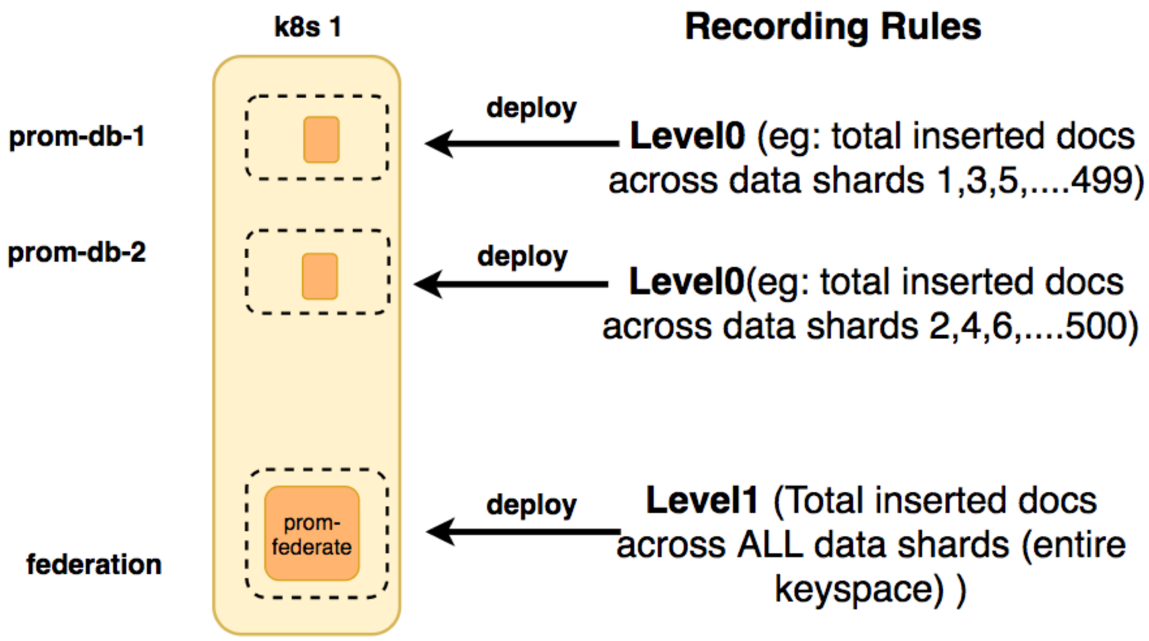
- The paired Prometheus servers share same config and scrape the same targets

Sharded Setup with Multiple Categories



- Multiple categories: DB Service, DB Proxy, DB Engine, Indexing Engine, OS metrics
- A Prometheus cluster dedicated to each category
- Each Prometheus cluster has multiple shards
- Each prometheus cluster is mirrored in a remote DC for HA

Federation: Level 0 & 1 Recording Rules



Recording Rules

Level0 (eg: total inserted docs across data shards 1,3,5,....499)

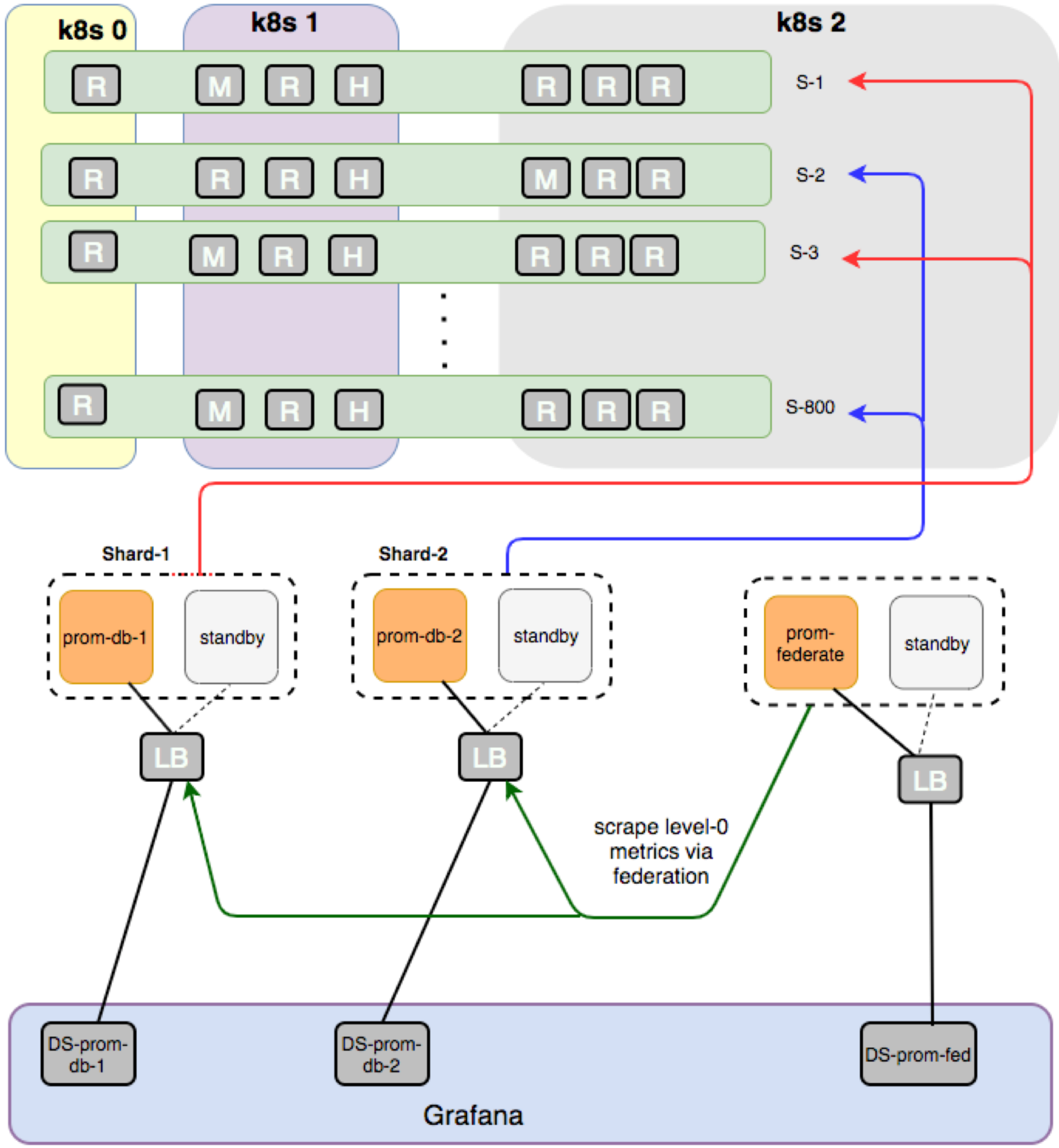
Level0(eg: total inserted docs across data shards 2,4,6,....500)

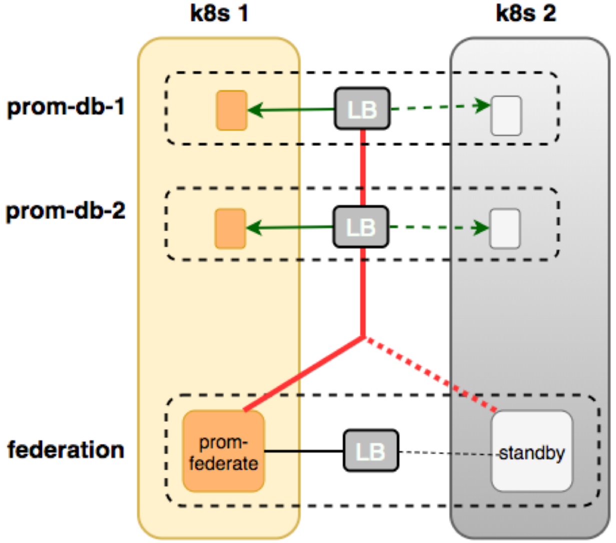
Level1 (Total inserted docs across ALL data shards (entire keyspace))

```
- record: level0:inserted_document_at_keyspace_level:rate5m
  expr: sum(rate(document_total{state=”inserted"}[5m])) BY(keyspace)
```

```
- record: level1:inserted_document_at_keyspace_level:rate5m
   expr: sum(level0:inserted_document_at_keyspace_level:rate5m) BY (keyspace)
```

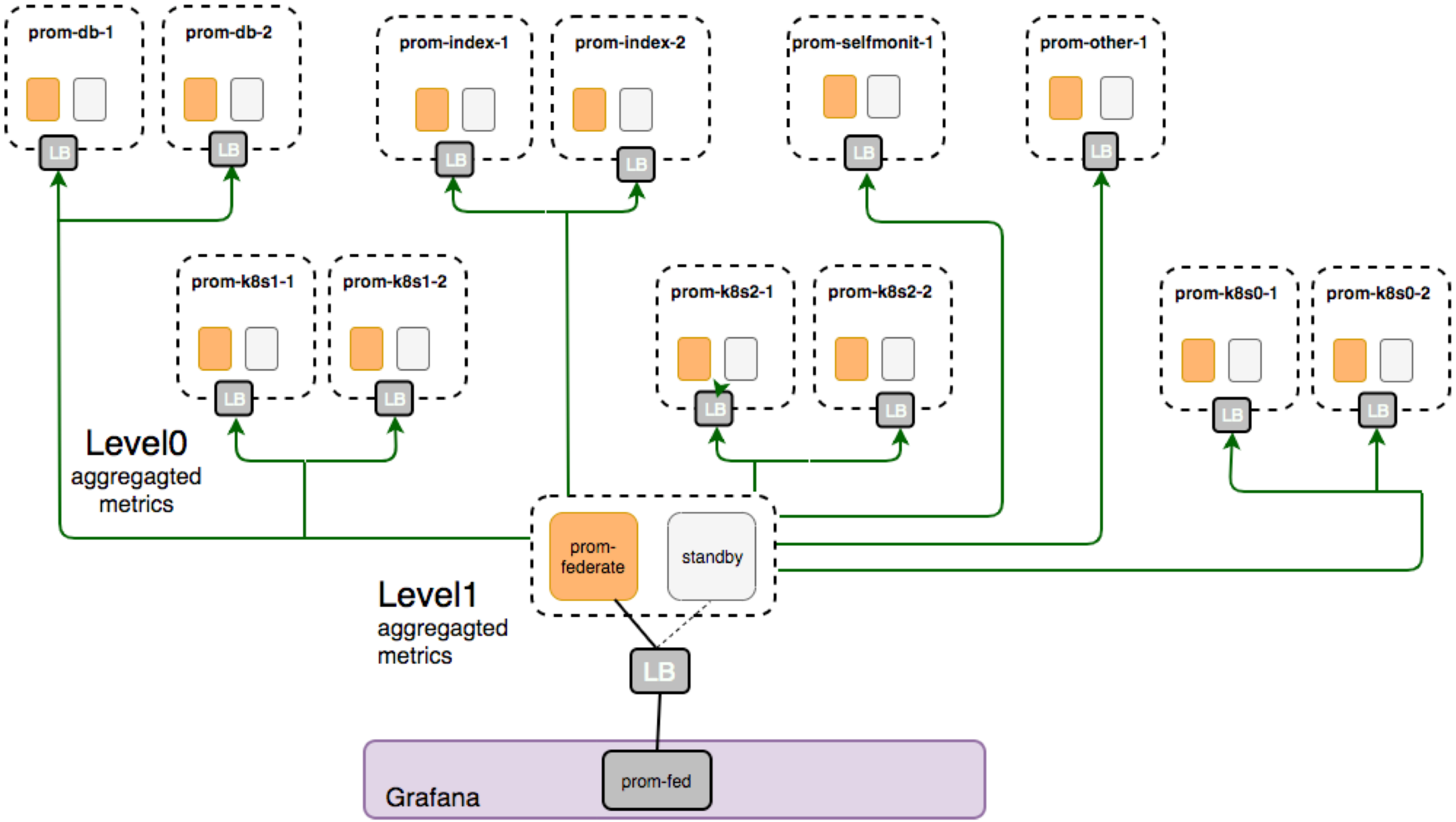
Highly Available Federated Setup





- Highly available federation server pair
- Highly available sharded Prometheus servers
- Each federation server scrapes the sharded Prometheus cluster via LB VIPs to provide HA transparently
- Grafana points to LB VIPs

Complete Picture: Federated and Sharded Setup



- Automation scripts developed to deploy the full setup illustrated above

Outline

- Metrics capturing and aggregation
- The Sharded and Federated Prometheus cluster
- ➡ Query routing & UI integration
- Monitoring experiences
- Conclusions

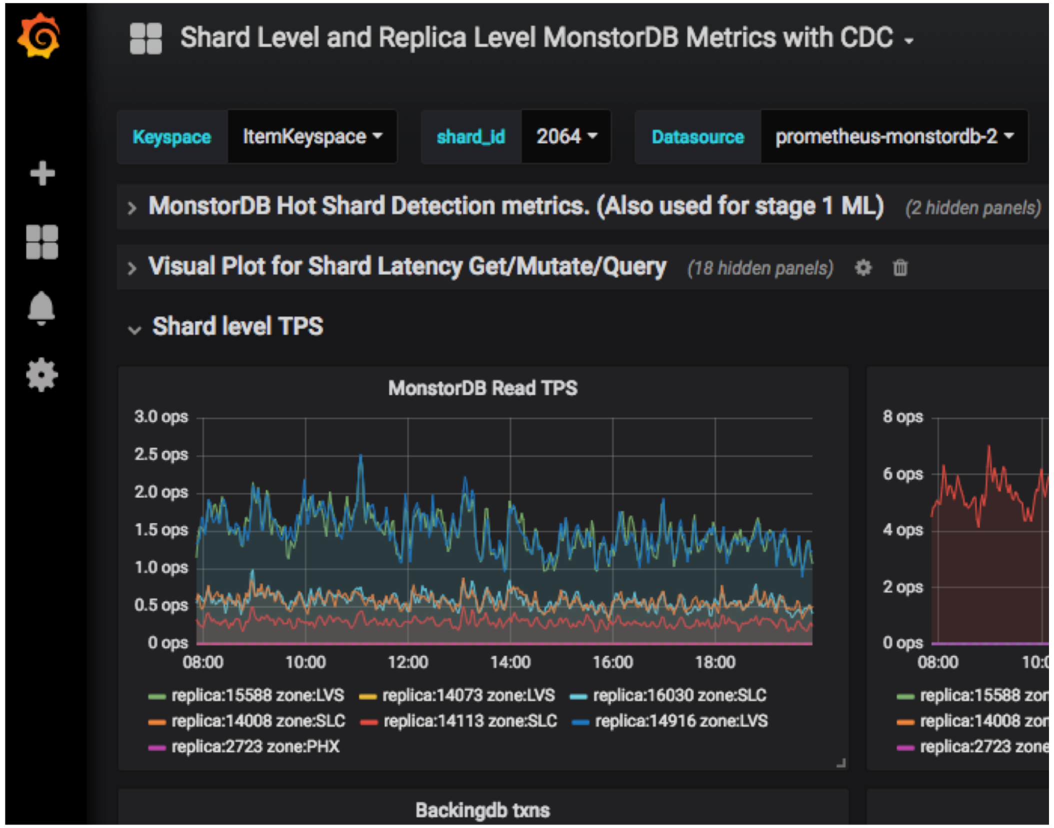
Query Routing

- The **hashmod** function determines which Prometheus sharded server should scrape (and store) metrics for a particular data shard

- A visualization framework (such as Grafana) requires auto-selection of the Prometheus data source to query metrics

- The solution:
  - A Federated Lookup/Routing table in our Prometheus cluster setup and
  - Templated variables and templated datasource in Grafana
  - **No changes needed to Prometheus and Grafana**

Auto-Populated Datasource in Grafana



Shard Level and Replica Level MonstorDB Metrics with CDC

Keyspace: ItemKeyspace
shard_id: 2064
Datasource: prometheus-monstordb-2

› MonstorDB Hot Shard Detection metrics. (Also used for stage 1 ML) *(2 hidden panels)*

› Visual Plot for Shard Latency Get/Mutate/Query *(18 hidden panels)*

⌄ Shard level TPS

MonstorDB Read TPS

Backingdb txns

- Example: View metrics at shard-level.
  - The metrics are labeled with {replica id, shard id, keyspace} hierarchy

- {keyspace, shard} are chosen from the first two drop-downs

- The Datasource (a templated data source) is automatically populated

Federated Lookup Routing Map

- A special recording rule (timeseries): **level0:routing_map_prom_keyspace_shard**
  - Based on the default ‘**up**’ for every scraped target
  - Deployed to each Prometheus shard
  - Scraped by the Federation server
- **A Time-based Global Lookup Table:** mapping of {scraped targets, Prometheus shard} over time.

```
record: level0:routing_map_prom_keyspace_shard
           expr: count(up{job=~"monstordb-.*"})
            BY (keyspace, shard, zone)
```

Instant Timeseries Vector on prom-shard-1

*level0:routing_map_prom_keyspace_shard {keyspace=”**KS1**",shard=”**1**", zone="PHX"}*

*level0:routing_map_prom_keyspace_shard {keyspace=”**KS2**",shard=”**3**", zone="PHX"}*

Instant Timeseries Vector on prom-shard-2

*level0:routing_map_prom_keyspace_shard {keyspace=”**KS1**",shard=”**2**", zone=”LVS"}*

*level0:routing_map_prom_keyspace_shard {keyspace=”**KS2**",shard=”**4**", zone=”LVS"}*

Instant Timeseries Vector on prom-federation (with injected external label ds_name)

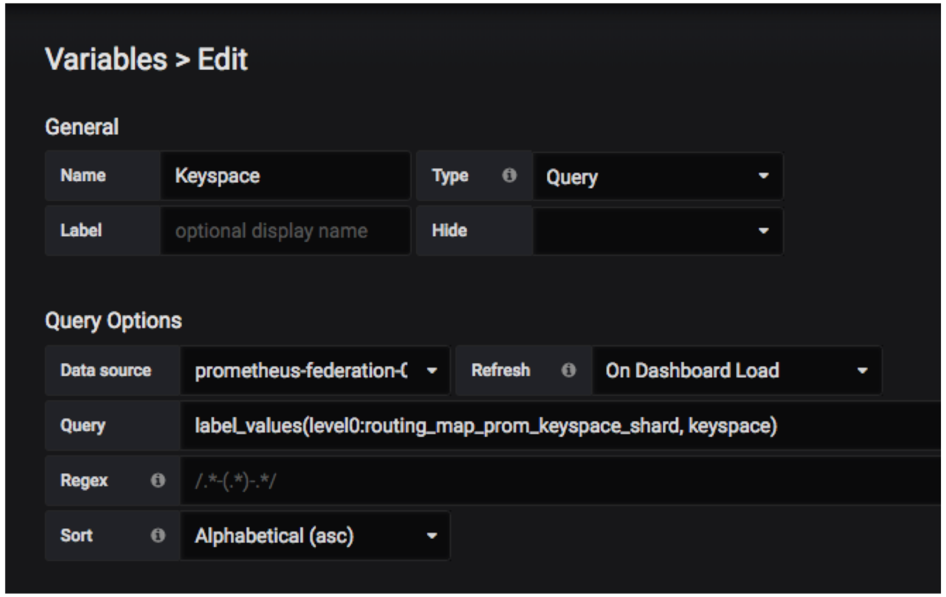
*level0:routing_map_prom_keyspace_shard {keyspace=”**KS1**",shard=”1", zone="PHX”, ds_name=“prom-shard-1}*

*level0:routing_map_prom_keyspace_shard {keyspace=”**KS1**",shard=”2", zone=”LVS”, ds_name=“prom-shard-2}*

*level0:routing_map_prom_keyspace_shard {keyspace=”**KS2**",shard=”3", zone="PHX”, ds_name=“prom-shard-1}*

*level0:routing_map_prom_keyspace_shard {keyspace=”**KS2**",shard=”4", zone=”LVS”, ds_name=“prom-shard-2}*

Template Variables based on Routing Map



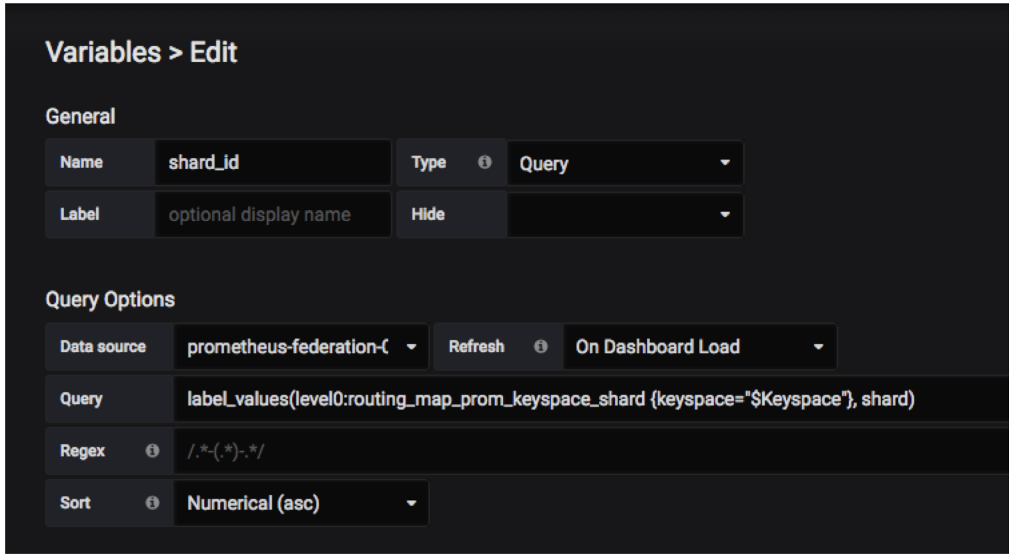
Variables > Edit

General

Name: Keyspace
Type: Query
Label: optional display name
Hide:

Query Options

Data source: prometheus-federation-(
Refresh: On Dashboard Load
Query: label_values(level0:routing_map_prom_keyspace_shard, keyspace)
Regex: /.*-(.*)-.*/
Sort: Alphabetical (asc)



Variables > Edit

General

Name: shard_id
Type: Query
Label: optional display name
Hide:

Query Options

Data source: prometheus-federation-(
Refresh: On Dashboard Load
Query: label_values(level0:routing_map_prom_keyspace_shard {keyspace="$Keyspace"}, shard)
Regex: /.*-(.*)-.*/
Sort: Numerical (asc)

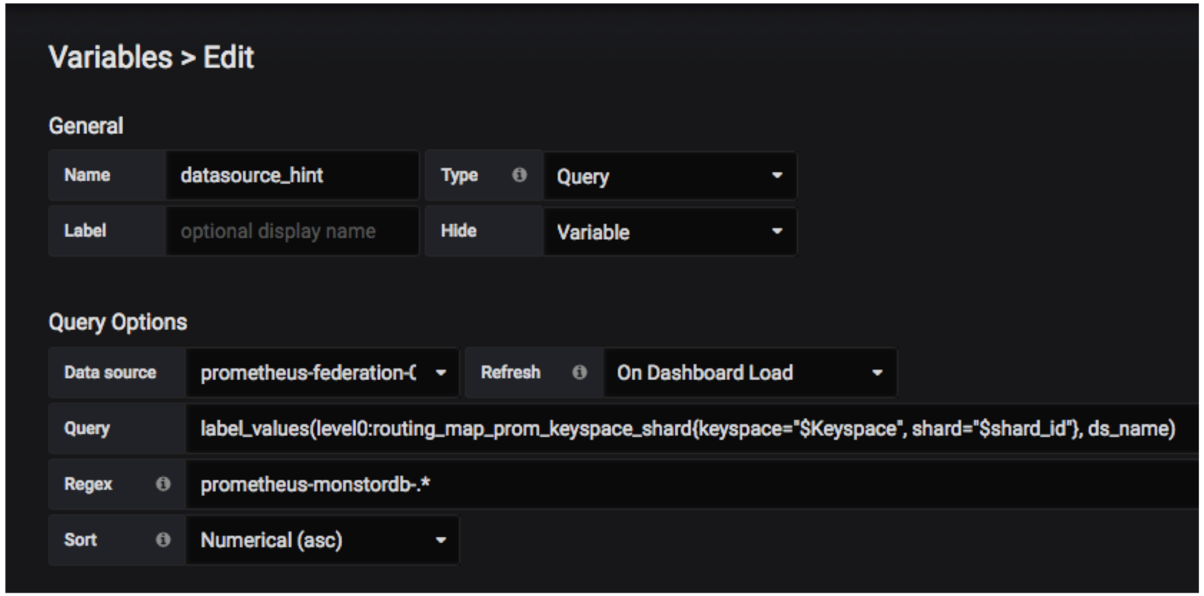
The label_values() function is applied to **level0:routing_map_prom_keyspace_shard** at the Federation server

List of **keyspaces** = label_values(level0:routing_map_prom_keyspace_shard , keyspace) = { KS1, KS2 }

List of **shards** of KS1 = label_values(level0:routing_map_prom_keyspace_shard {keyspace=KS1}, shard) = {1, 2 }

List of **shards** of KS2 = label_values(level0:routing_map_prom_keyspace_shard {keyspace=KS2}, shard) = {3, 4 }

Template Variables based on Routing Map



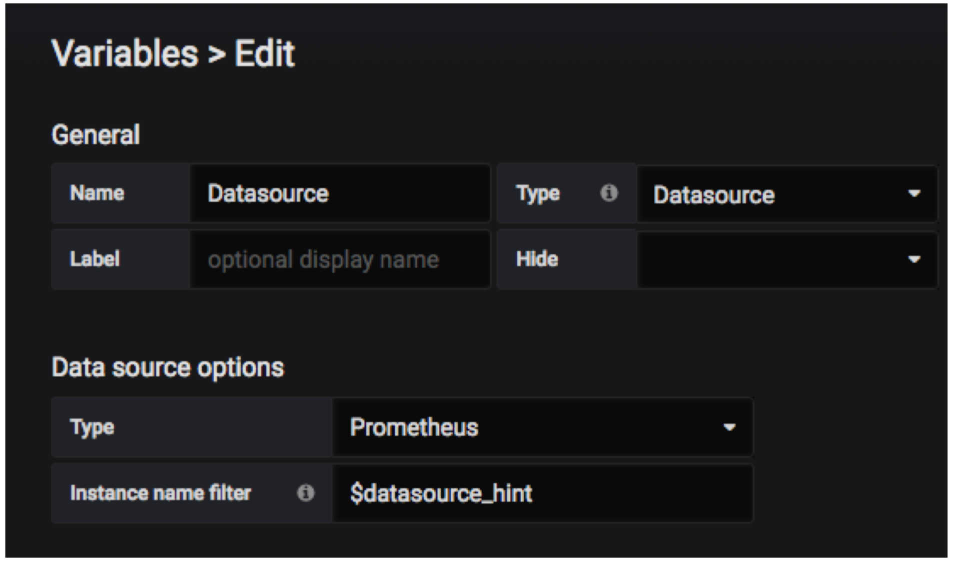
Variables > Edit

General

Name	datasource_hint	Type	Query
Label	optional display name	Hide	Variable

Query Options

Data source	prometheus-federation-(	Refresh	On Dashboard Load
Query	label_values(level0:routing_map_prom_keyspace_shard{keyspace="$Keyspace", shard="$shard_id"}, ds_name)		
Regex	prometheus-monstordb-.*		
Sort	Numerical (asc)		



Variables > Edit

General

Name	Datasource	Type	Datasource
Label	optional display name	Hide	

Data source options

Type	Prometheus
Instance name filter	$datasource_hint

- Upon a keyspace and shard selection, retrieve the Datasource name by making query to Federated Prometheus server:
  - **datasource_hint =** label_values(level0:routing_map_prom_keyspace_shard {keyspace=KS1, shard=1}, ds_name) = {prom-shard-1}
  - The label values of ds_name match the datasource names we define in Grafana

- Grafana `*datasource*` type template variable cannot be directly of 'query' value type

- Hence the **hidden** variable *datasource_hint* is introduced to hold query value in the transient/hidden variable

Outline

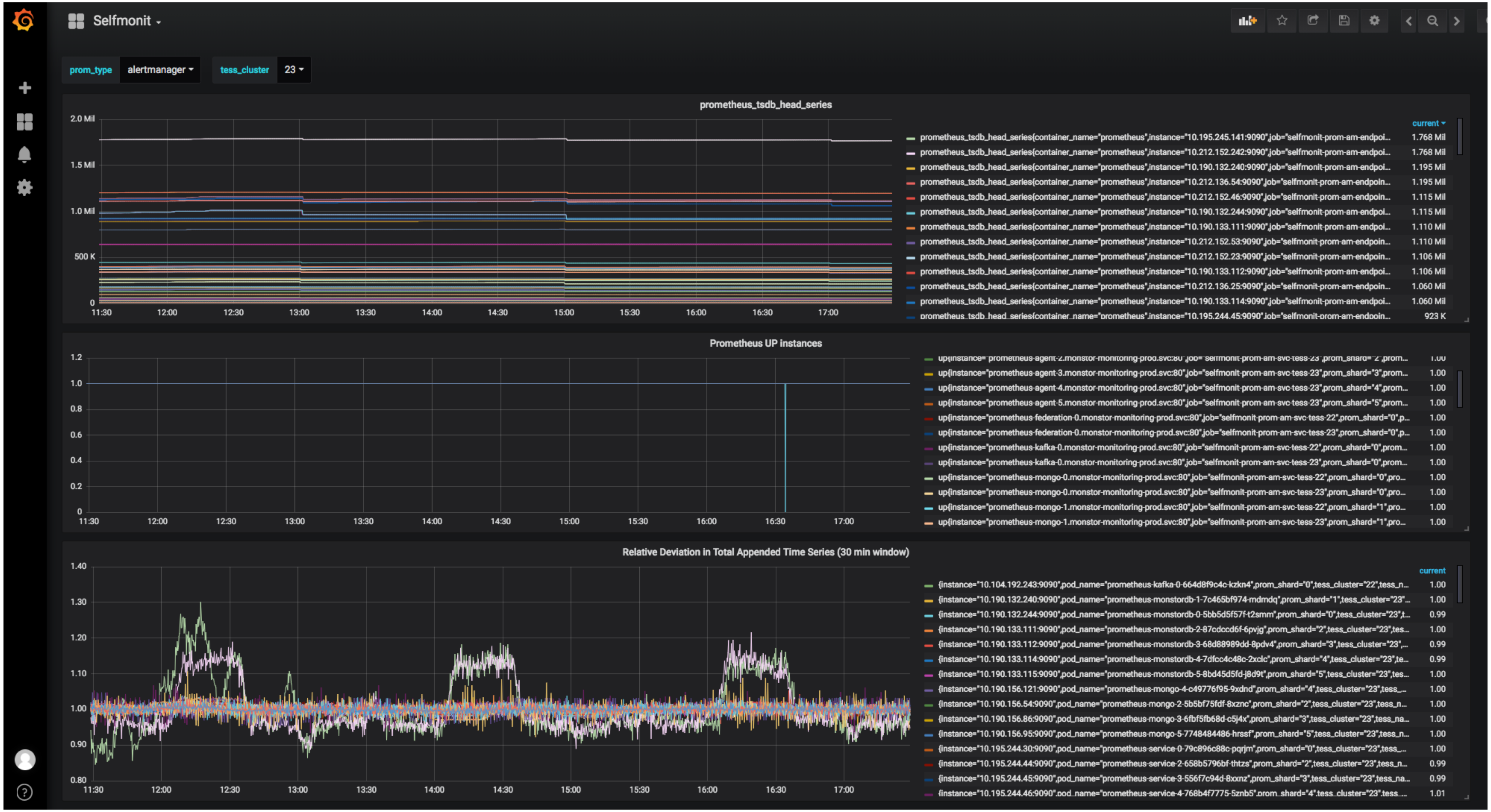
- Metrics capturing and aggregation
- The Sharded and Federated Prometheus cluster
- Query Routing & UI Integration
- Monitoring experiences
- Conclusions

Self Monitoring (Monitoring of Monitoring)

- All Prometheus instances are scraped by a **Selfmonit** Prometheus instance
  - The whole monitoring infrastructure metrics captured at one place
  - Easy comparison of metrics among Prometheus shards (e.g., load distribution is even or not)

- Self Monitoring instance is also HA

Self Monitoring (Monitoring of Monitoring)



Selfmonit

prom_type alertmanager
tess_cluster 23

prometheus_tsdb_head_series

```
                                                                                                              current
prometheus_tsdb_head_series{container_name="prometheus",instance="10.195.245.141:9090",job="selfmonit-prom-am-endpoi...   1.768 Mil
prometheus_tsdb_head_series{container_name="prometheus",instance="10.212.152.242:9090",job="selfmonit-prom-am-endpoi...   1.768 Mil
prometheus_tsdb_head_series{container_name="prometheus",instance="10.190.132.240:9090",job="selfmonit-prom-am-endpoi...   1.195 Mil
prometheus_tsdb_head_series{container_name="prometheus",instance="10.212.136.54:9090",job="selfmonit-prom-am-endpoin...   1.195 Mil
prometheus_tsdb_head_series{container_name="prometheus",instance="10.212.152.46:9090",job="selfmonit-prom-am-endpoin...   1.115 Mil
prometheus_tsdb_head_series{container_name="prometheus",instance="10.190.132.244:9090",job="selfmonit-prom-am-endpoi...   1.115 Mil
prometheus_tsdb_head_series{container_name="prometheus",instance="10.190.133.111:9090",job="selfmonit-prom-am-endpoi...   1.110 Mil
prometheus_tsdb_head_series{container_name="prometheus",instance="10.212.152.53:9090",job="selfmonit-prom-am-endpoin...   1.110 Mil
prometheus_tsdb_head_series{container_name="prometheus",instance="10.212.152.23:9090",job="selfmonit-prom-am-endpoin...   1.106 Mil
prometheus_tsdb_head_series{container_name="prometheus",instance="10.190.133.112:9090",job="selfmonit-prom-am-endpoi...   1.106 Mil
prometheus_tsdb_head_series{container_name="prometheus",instance="10.212.136.25:9090",job="selfmonit-prom-am-endpoin...   1.060 Mil
prometheus_tsdb_head_series{container_name="prometheus",instance="10.190.133.114:9090",job="selfmonit-prom-am-endpoi...   1.060 Mil
prometheus_tsdb_head_series{container_name="prometheus",instance="10.195.244.45:9090",job="selfmonit-prom-am-endpoin...     923 K
```

Prometheus UP instances

```
up{instance="prometheus-agent-2.monstor-monitoring-prod.svc:80",job="selfmonit-prom-am-svc-tess-23",prom_shard="2",prom...   1.00
up{instance="prometheus-agent-3.monstor-monitoring-prod.svc:80",job="selfmonit-prom-am-svc-tess-23",prom_shard="3",prom...   1.00
up{instance="prometheus-agent-4.monstor-monitoring-prod.svc:80",job="selfmonit-prom-am-svc-tess-23",prom_shard="4",prom...   1.00
up{instance="prometheus-agent-5.monstor-monitoring-prod.svc:80",job="selfmonit-prom-am-svc-tess-23",prom_shard="5",prom...   1.00
up{instance="prometheus-federation-0.monstor-monitoring-prod.svc:80",job="selfmonit-prom-am-svc-tess-22",prom_shard="0",p...   1.00
up{instance="prometheus-federation-0.monstor-monitoring-prod.svc:80",job="selfmonit-prom-am-svc-tess-23",prom_shard="0",p...   1.00
up{instance="prometheus-kafka-0.monstor-monitoring-prod.svc:80",job="selfmonit-prom-am-svc-tess-22",prom_shard="0",prom...   1.00
up{instance="prometheus-kafka-0.monstor-monitoring-prod.svc:80",job="selfmonit-prom-am-svc-tess-23",prom_shard="0",prom...   1.00
up{instance="prometheus-mongo-0.monstor-monitoring-prod.svc:80",job="selfmonit-prom-am-svc-tess-22",prom_shard="0",pro...   1.00
up{instance="prometheus-mongo-0.monstor-monitoring-prod.svc:80",job="selfmonit-prom-am-svc-tess-23",prom_shard="0",pro...   1.00
up{instance="prometheus-mongo-1.monstor-monitoring-prod.svc:80",job="selfmonit-prom-am-svc-tess-22",prom_shard="1",pro...   1.00
up{instance="prometheus-mongo-1.monstor-monitoring-prod.svc:80",job="selfmonit-prom-am-svc-tess-23",prom_shard="1",pro...   1.00
```

Relative Deviation in Total Appended Time Series (30 min window)

```
                                                                                                                      current
{instance="10.104.192.243:9090",pod_name="prometheus-kafka-0-664d8f9c4c-kzkn4",prom_shard="0",tess_cluster="22",tess_n...   1.00
{instance="10.190.132.240:9090",pod_name="prometheus-monstordb-1-7c465bf974-mdmdq",prom_shard="1",tess_cluster="23"...      1.00
{instance="10.190.132.244:9090",pod_name="prometheus-monstordb-0-5bb5d5f57f-t2smm",prom_shard="0",tess_cluster="23",t...    0.99
{instance="10.190.133.111:9090",pod_name="prometheus-monstordb-2-87cdccd6f-6pvjg",prom_shard="2",tess_cluster="23",tes...   1.00
{instance="10.190.133.112:9090",pod_name="prometheus-monstordb-3-68d88989dd-8pdv4",prom_shard="3",tess_cluster="23",...     0.99
{instance="10.190.133.114:9090",pod_name="prometheus-monstordb-4-7dfcc4c48c-2xclc",prom_shard="4",tess_cluster="23",te...   0.99
{instance="10.190.133.115:9090",pod_name="prometheus-monstordb-5-8bd45d5fd-j8d9t",prom_shard="5",tess_cluster="23",tes...   1.00
{instance="10.190.156.121:9090",pod_name="prometheus-mongo-4-c49776f95-9xdnd",prom_shard="4",tess_cluster="23",tess_...     1.00
{instance="10.190.156.54:9090",pod_name="prometheus-mongo-2-5b5bf75fdf-8xznc",prom_shard="2",tess_cluster="23",tess_n...    1.00
{instance="10.190.156.86:9090",pod_name="prometheus-mongo-3-6fbf5fb68d-c5j4x",prom_shard="3",tess_cluster="23",tess_na...   1.00
{instance="10.190.156.95:9090",pod_name="prometheus-mongo-5-7748484486-hrssf",prom_shard="5",tess_cluster="23",tess_n...    1.00
{instance="10.195.244.30:9090",pod_name="prometheus-service-0-79c896c88c-pqrjm",prom_shard="0",tess_cluster="23",tess_...   1.00
{instance="10.195.244.44:9090",pod_name="prometheus-service-2-658b5796bf-thtzs",prom_shard="2",tess_cluster="23",tess_n...   0.99
{instance="10.195.244.45:9090",pod_name="prometheus-service-3-556f7c94d-8xxnz",prom_shard="3",tess_cluster="23",tess_na...   0.99
{instance="10.195.244.46:9090",pod_name="prometheus-service-4-768b4f7775-5znb5",prom_shard="4",tess_cluster="23",tess_...   1.01
```

Self Monitoring

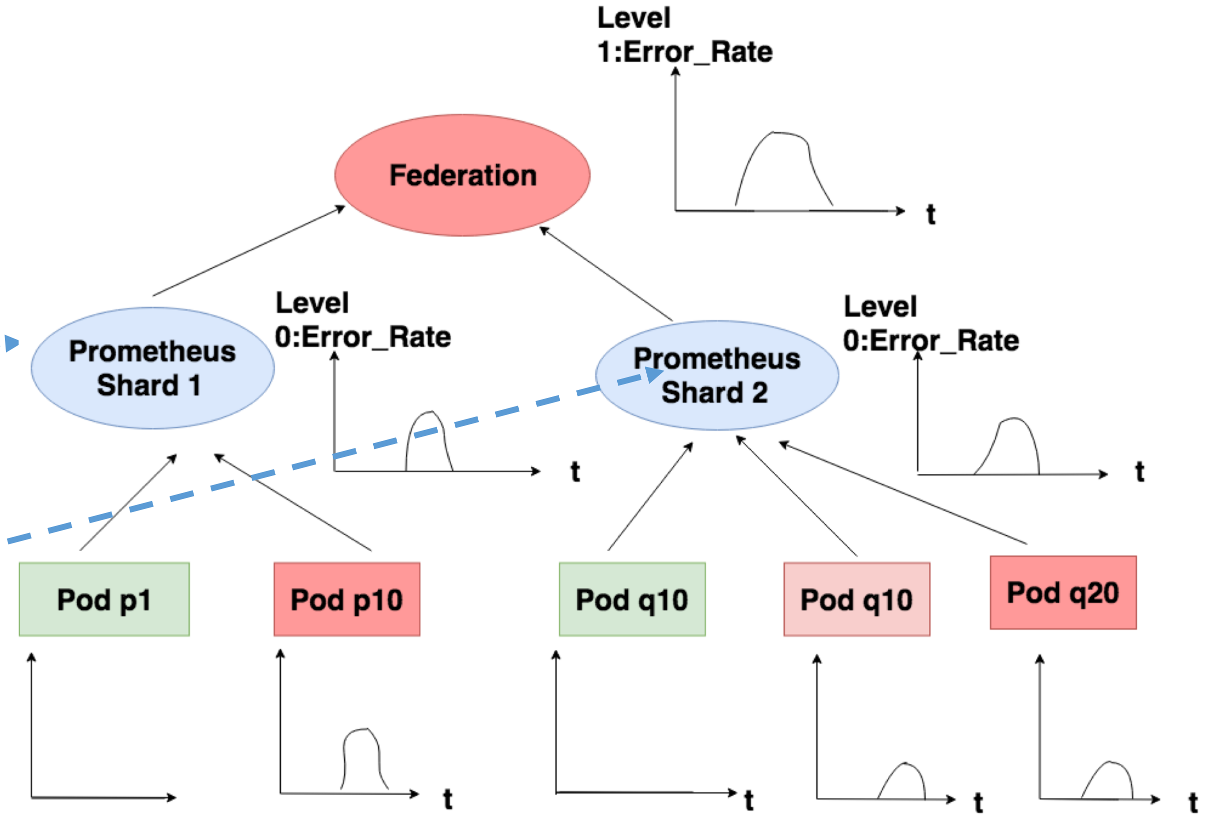
- Example queries:
  - *Relative deviation in time series appended:* sum (rate(prometheus_tsdb_head_samples_appended_total{container_name="prometheus"}[5m]) / rate(prometheus_tsdb_head_samples_appended_total{container_name="prometheus"}[5m] offset 30m)) by (instance, pod_name, prom_shard, prom_type, tess_cluster, tess_namespace)
  - *Rule evaluation duration percentile :* prometheus_rule_evaluation_duration_seconds{prom_type=~"prometheus.*", quantile="0.99"}

- Example of alert rules:
  - Down Prometheus instances
  - Abrupt drop in time series appended
  - Abrupt drop in targets discovered
  - VIP endpoint reachability

Pinpoint Troublesome Runtime Entities

- Example 1: Read error rate to a keyspace is now going up, what are the worst service pods that we need to investigate?

```
topk(5, sum(increase
(failed_responses_total{keyspace=”K1”,
method_name=”READ”}[30m])) by
(pod_name))
```

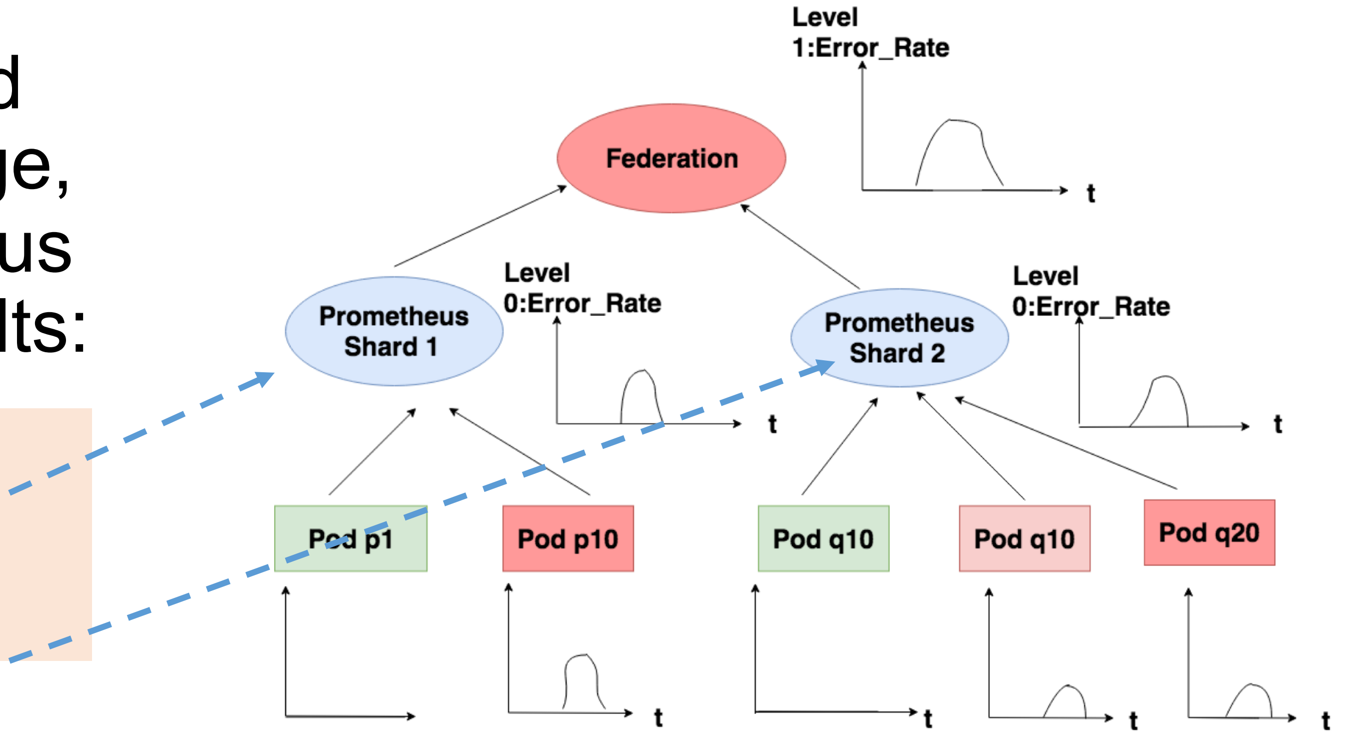


- Query over each sharded Prometheus server and combine the top-k results

Pinpoint Troublesome Runtime Entities (2)

- Alternatively, to have the following query to be plotted over the specified time range, on each sharded Prometheus server and inspect the results:

```
sum(rate
(failed_responses_total{keyspace=”K1”,
method_name=”READ”}[5m])) by
(pod_name))
```



- Plotting can be done in the Prometheus web console:
    - It can handle hundreds of time-series plots easily

Special OS Metrics Aggregation

- A database pod has pod spec. to track the logical hierarchy {keyspace, shard, replica}

- Kublet exposes OS pod level metrics, but without labels from the application’s pod spec attached

```
container_memory_rss (zone=“…”, host=“…”, rack=“…”, pod=pod_name) = 150000
```

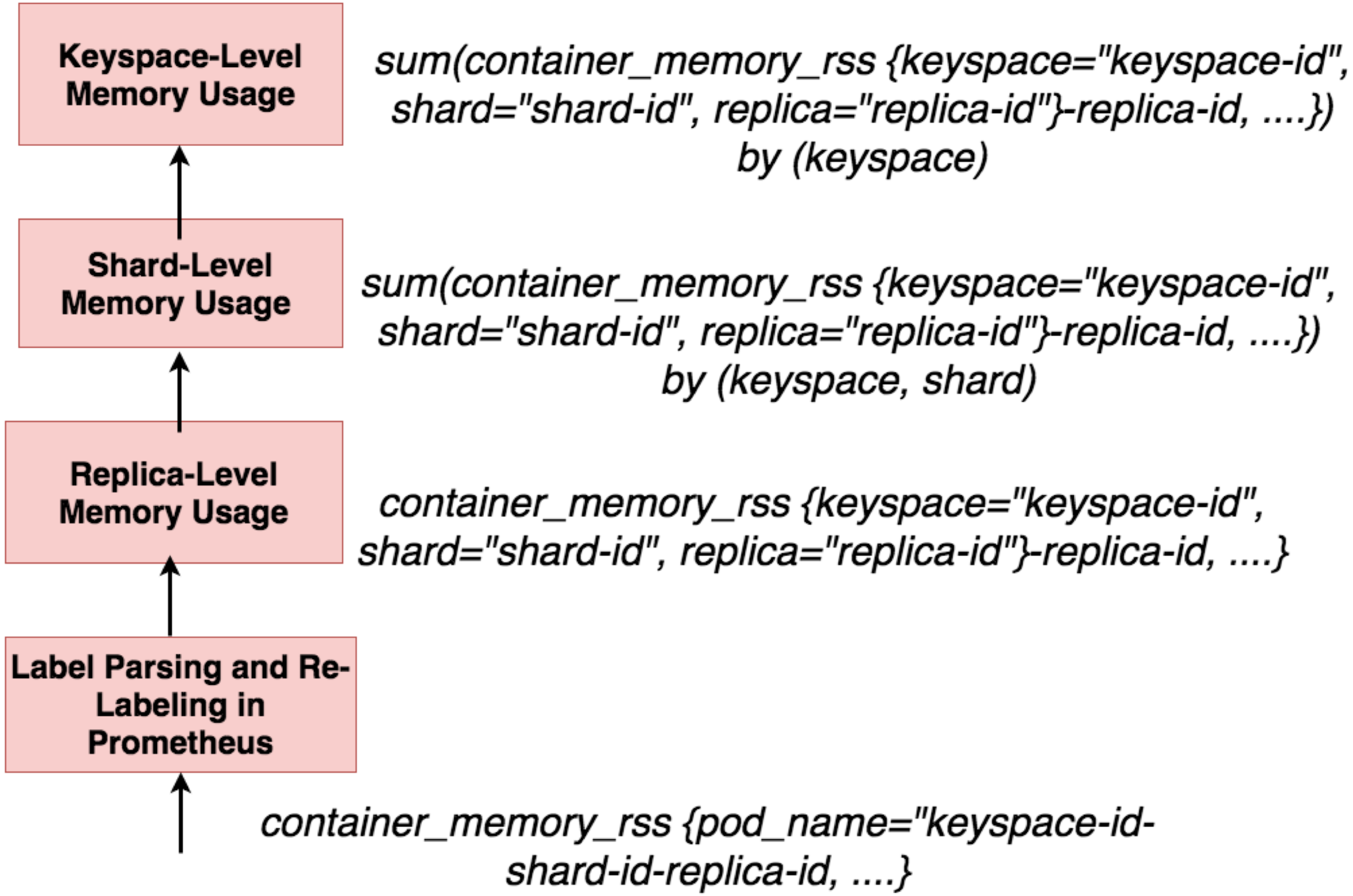
- Thus CPU/memory aggregation over logical hierarchy is not available

- Solution: label extraction and label injection, by leveraging the naming convention that we follow for database pods:

```
pod_name = keyspace-id + shard-id + replica-id + other information
```

Special OS Metrics Aggregation

- Label extraction: to extract keyspace id, shard id and replica id from pod name
- Label injection: to inject labels: {keyspace, shard, and replica} into the OS metrics
- Thus OS aggregation over logical aggregation is now available



Keyspace-Level Memory Usage

sum(container_memory_rss {keyspace="keyspace-id", shard="shard-id", replica="replica-id"}-replica-id, ...}) by (keyspace)

Shard-Level Memory Usage

sum(container_memory_rss {keyspace="keyspace-id", shard="shard-id", replica="replica-id"}-replica-id, ...}) by (keyspace, shard)

Replica-Level Memory Usage

container_memory_rss {keyspace="keyspace-id", shard="shard-id", replica="replica-id"}-replica-id, ...}

Label Parsing and Re-Labeling in Prometheus

container_memory_rss {pod_name="keyspace-id-shard-id-replica-id, ...}

Alert Summarization

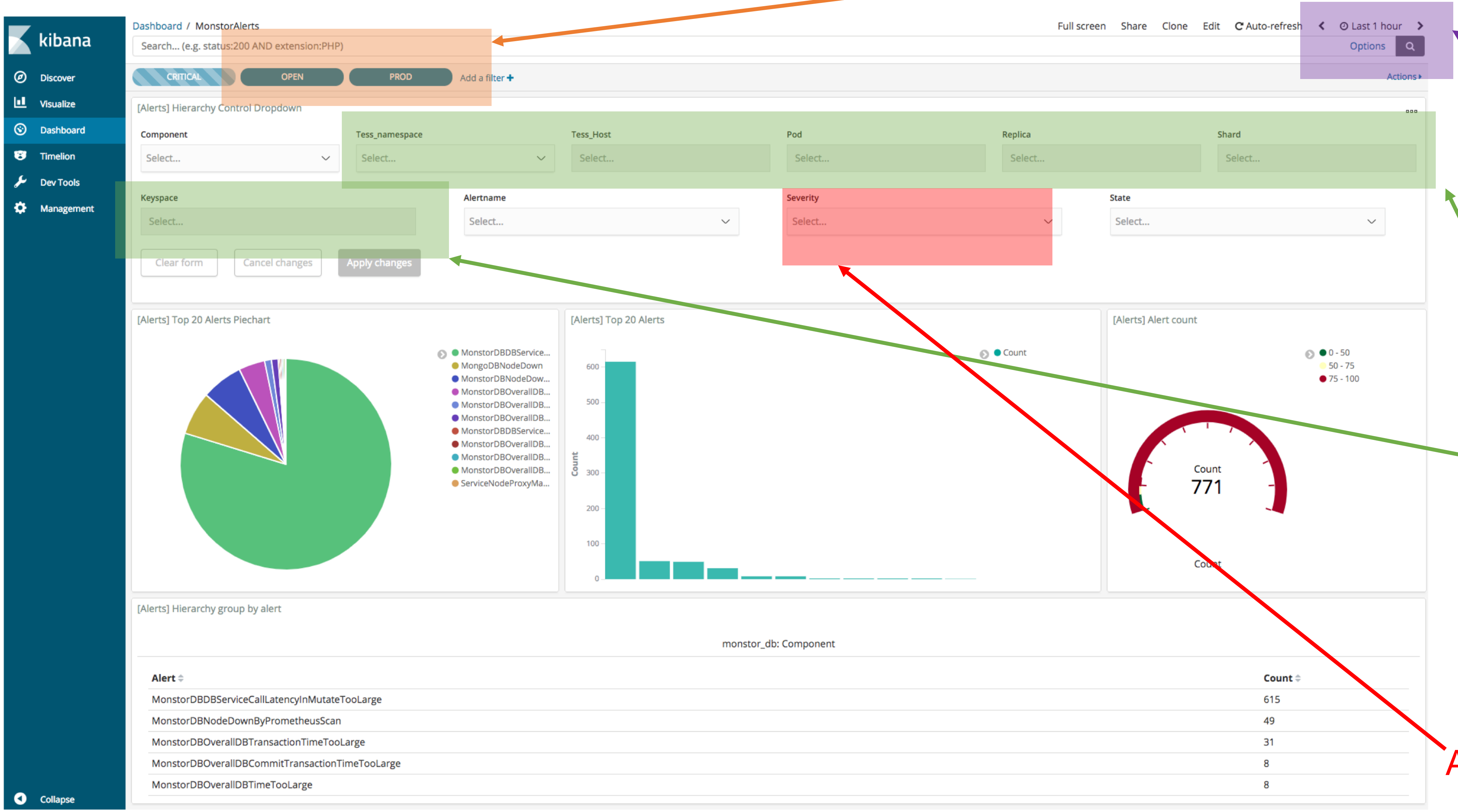
- A Prometheus alert has the labels from the recording rule evaluated

- Summarization on alerts:
    - Over severity {critical, high, warning}
    - Over logical hierarchy
    - Over physical hierarchy

- For both historical alerts and active firing (not resolved) alerts

- Solution: to store and index the received alerts into Elasticsearch

Alert Dashboard in Kibana



[Alerts] Hierarchy group by alert

monstor_db: Component

Alert	Count
MonstorDBDBServiceCallLatencyInMutateTooLarge	615
MonstorDBNodeDownByPrometheusScan	49
MonstorDBOverallDBTransactionTimeTooLarge	31
MonstorDBOverallDBCommitTransactionTimeTooLarge	8
MonstorDBOverallDBTimeTooLarge	8

Resolved/Active Firing

Historical or Real-time

Logical/Physical Hierarchies

Alert Severity

Outline

- Metrics capturing and aggregation
- The sharded and federated Prometheus cluster
- Query routing & UI integration
- Monitoring experiences
- Conclusions

Conclusions

- Prometheus itself is deployed as a standalone single process

- We have developed a horizontally scalable, sharded, and federated Prometheus monitoring cluster from Prometheus binary distribution with full automation scripts, without modifying its source code

- The scalable monitoring cluster allows us to have real-time dashboards and real-time alerts over the hierarchically aggregated metrics

Thank You !

Q & A

KubeCon | CloudNativeCon

North America 2018