# CMS Monitoring R&D: Real-time monitoring and Alerts

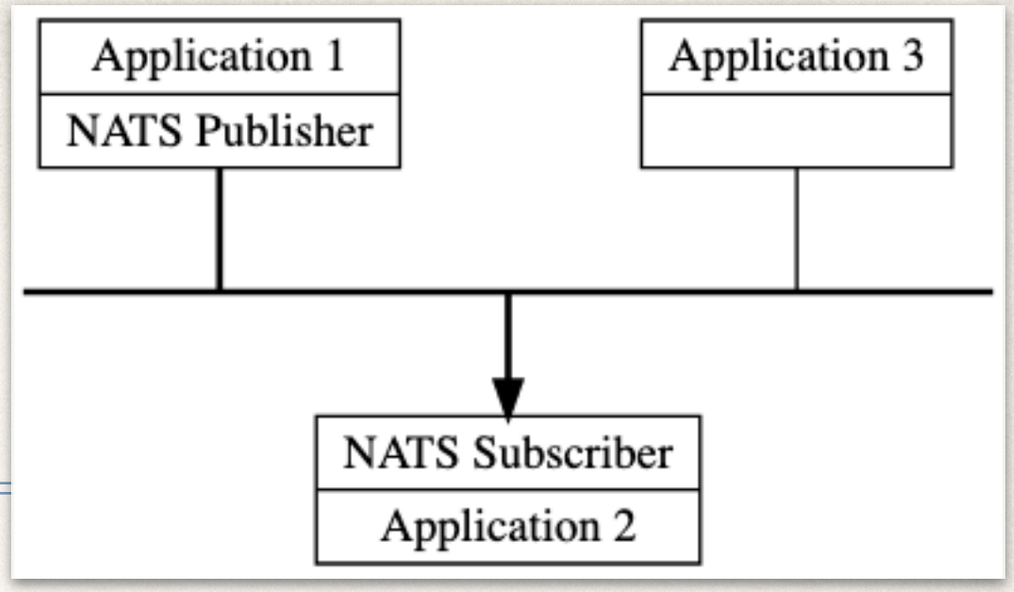
Valentin Kuznetsov, Cornell University

*Operational Intelligent meeting*

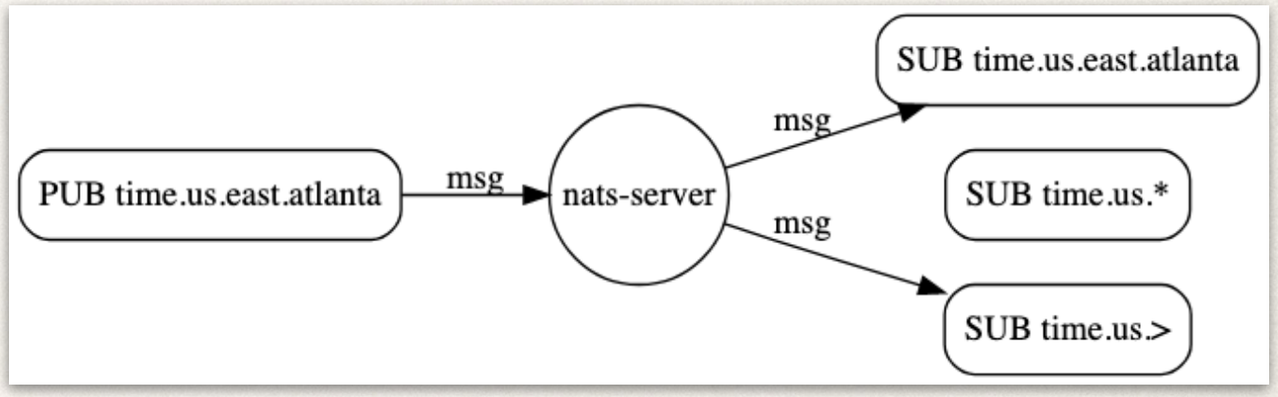
# Use cases

- Real-time monitoring may be useful for several use-cases
  - workflow failures, e.g. at a certain site
  - file/dataset access at a site and/or popularity
  - production, campaign monitoring
- Quite often various teams use data bookkeeping system (DBS in CMS) for monitoring needs where they place constant queries to see progress of dataset creation and workflow statuses
  - common use-cases require minimum information from our services in real-time therefore we can use messaging system for this purposes

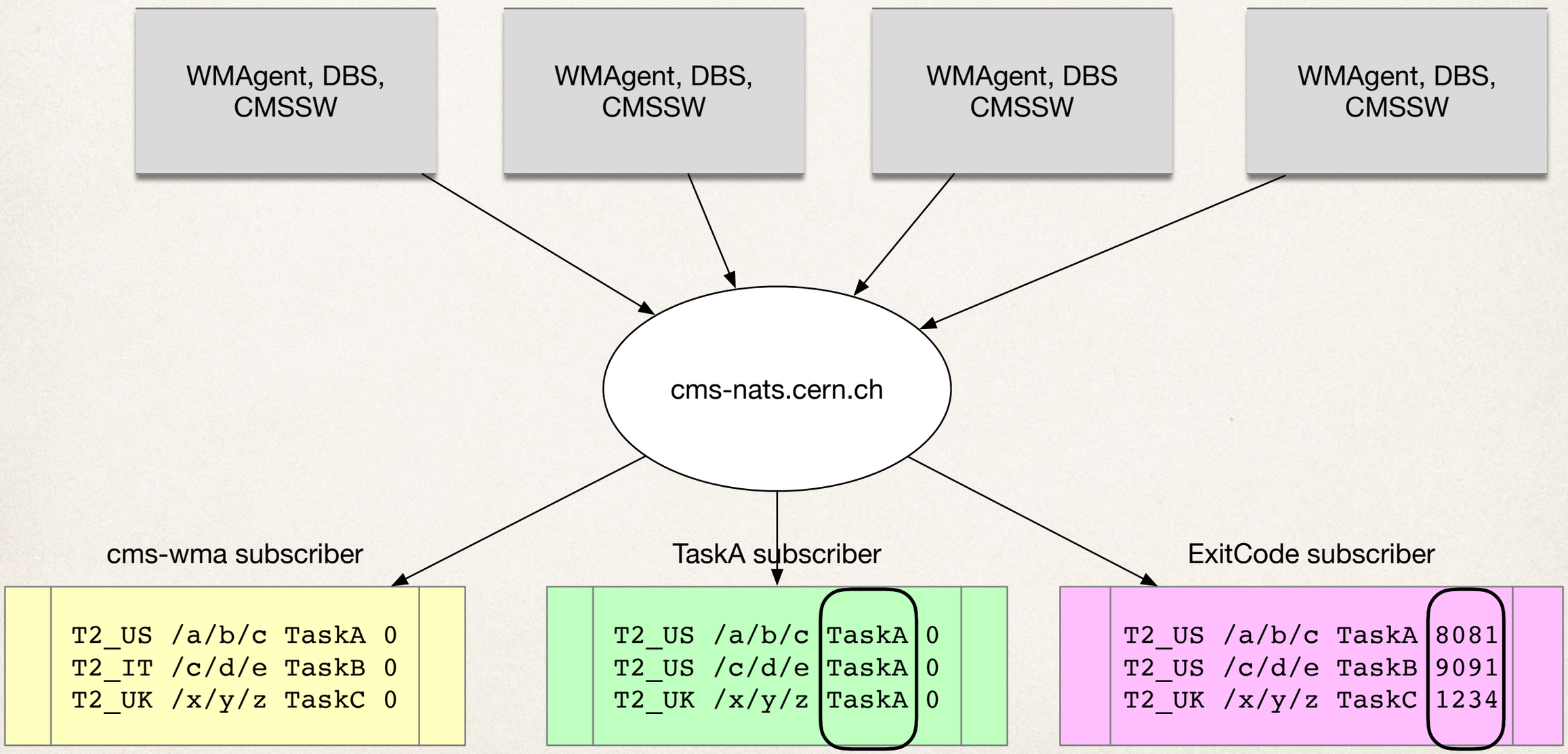
# Introducing NATS




- NATS (Neural Autonomic Transport System) is a simple, secure and high performance open source messaging system for cloud native applications
- It is lightweight system for sending/receiving messages
- It does not have overhead of Kafka, can be easily deployed (Go static executable)
- Wide range of clients: C/C#, Python, Ruby, Go, Java, etc (30 programming languages)
- It supports publish-subscribe, request-reply, streaming modes of operations
- Support at-most-once (NATS server) and at-least-once (NATS streaming) delivery




# NATs use case: monitor workflows, campaigns, datasets, exitCodes



# Use case scenarios

- CLI tool subscribes to a given topic: "cms.system.attr.pattern_or_value"
- Watch exit codes or specific exit code

```
nats-sub "cms.system.exitCode.>"     # watch all exit codes

nats-sub "cms.system.exitCode.8028"  # watch 8028 exit code only
```

- Watch production on all T2 sites

```
nats-sub "cms.system.site.T2.>"
```

- Watch production of specific campaign

```
nats-sub "cms.system.campaign.RunIISummer19UL17SIM"
```

- In CMS we use different systems to publish the data to certain channels
  - DBS yields info about dataset appearance
  - ReqMgr2 informs about dataset creation, new campaigns, task assignments
  - WMAgent/WMArchive systems report about production workflows, e.g. exit codes at a certain sites

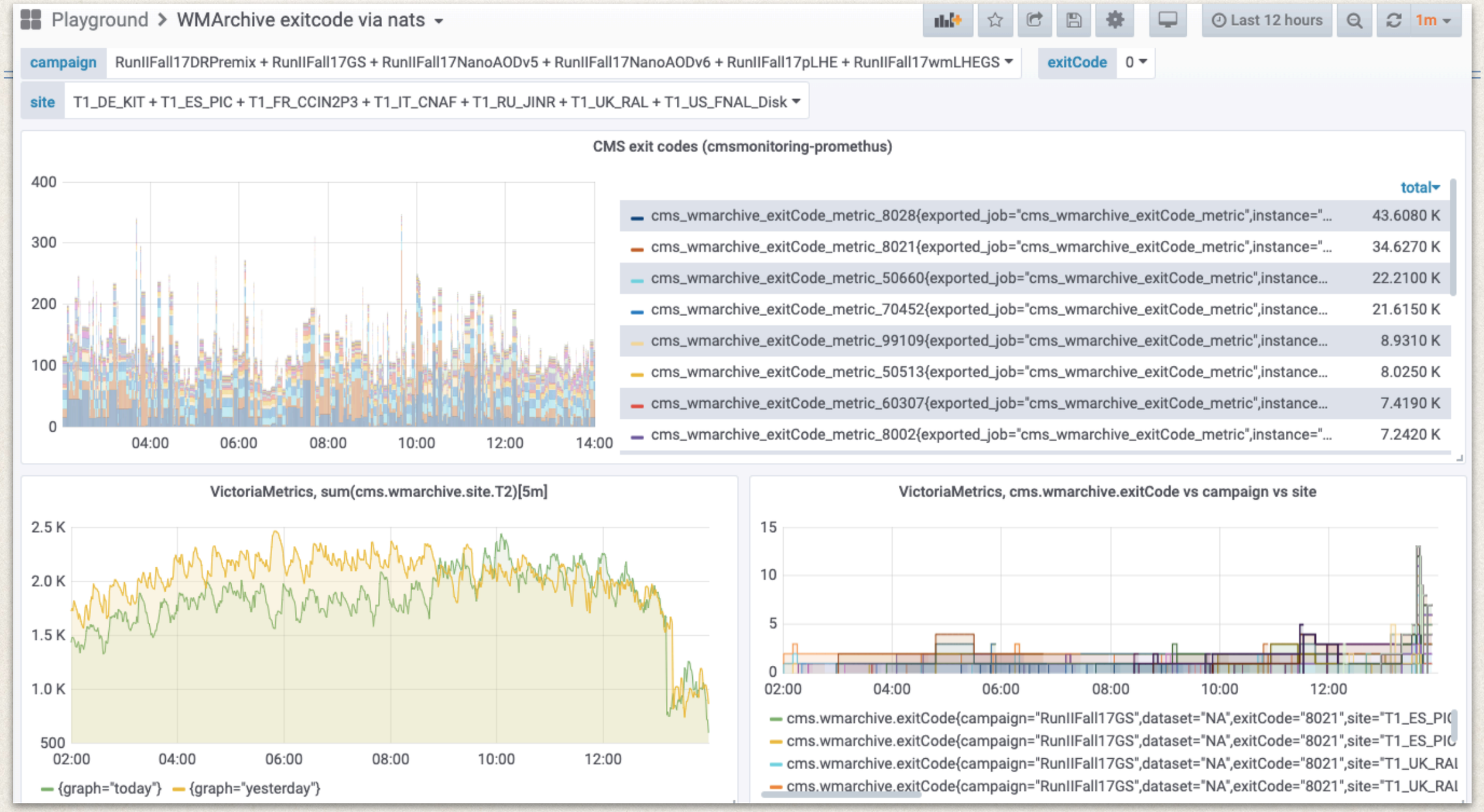
# Monitoring exit codes

```
# start new subscriber on exitCode channel and use timestamp prefix
nats-sub -t “cms.wmarchive.exitCode.>”

# now you can ONLY see exit code messages
2019/11/08 14:10:19 Listening on [exitCode]
2019/11/08 15:11:59 IDR_CMS_Home 8002  T3_CH_Volunteer /Data/sample
2019/11/08 15:27:02 IDR_CMS_Home 60307 T3_CH_Volunteer /RelVal/sample
2019/11/08 16:32:12 IDR_CMS_Home 99109 T3_CH_Volunteer /Another/sample
……

# start new subscriber on specific exitCode channel
nats-sub -t “cms.wmarchive.exitCode.8028”
2019/11/11 23:30:31 RunIIFall17GS 8028 T2_US_Site /Data/sample
……
```

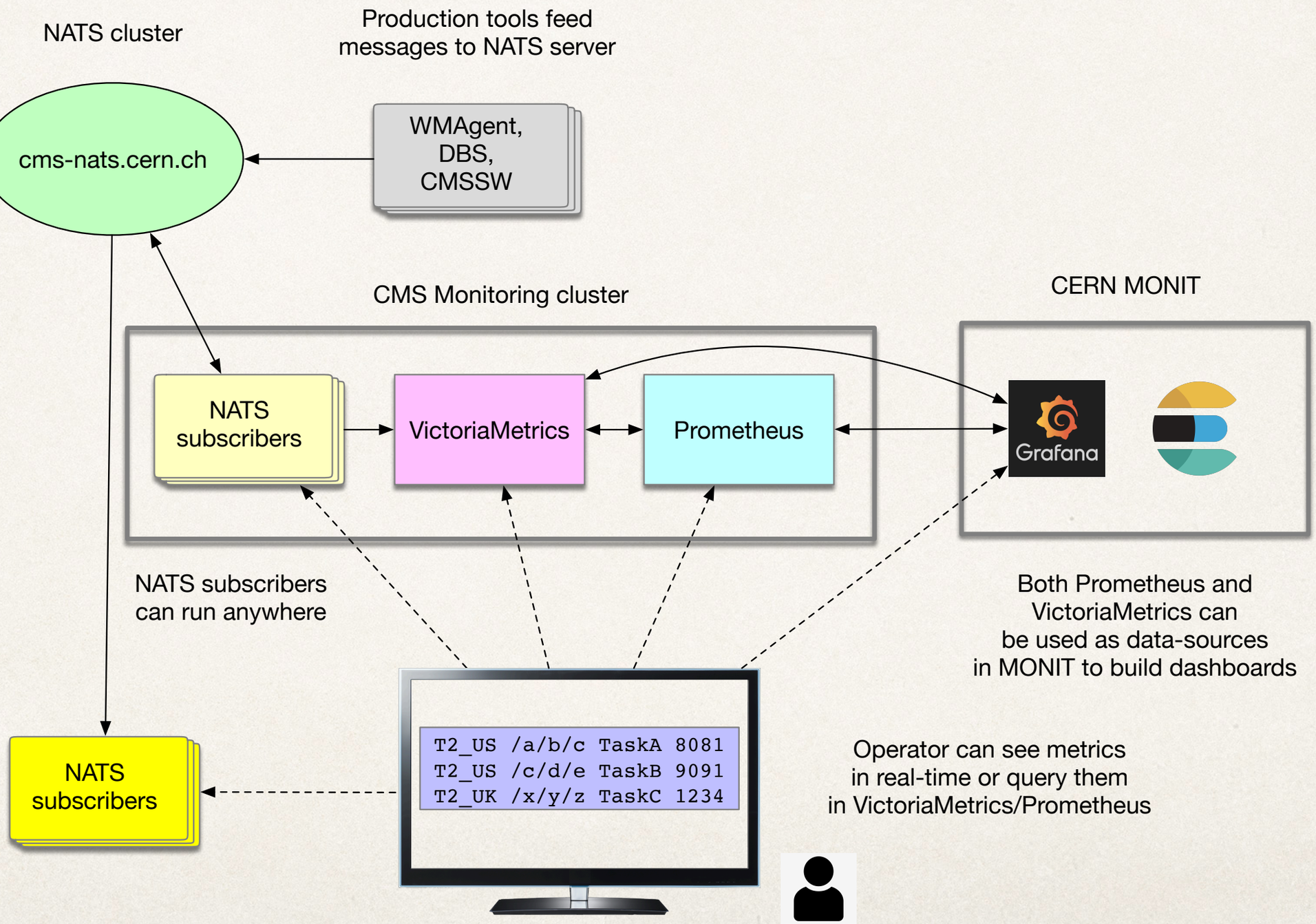
# NATS messages in MONIT




NATS messages can be visualized in MONIT if they're injected into VictoriaMetrics

VM is a fast IO back-end storage for Prometheus

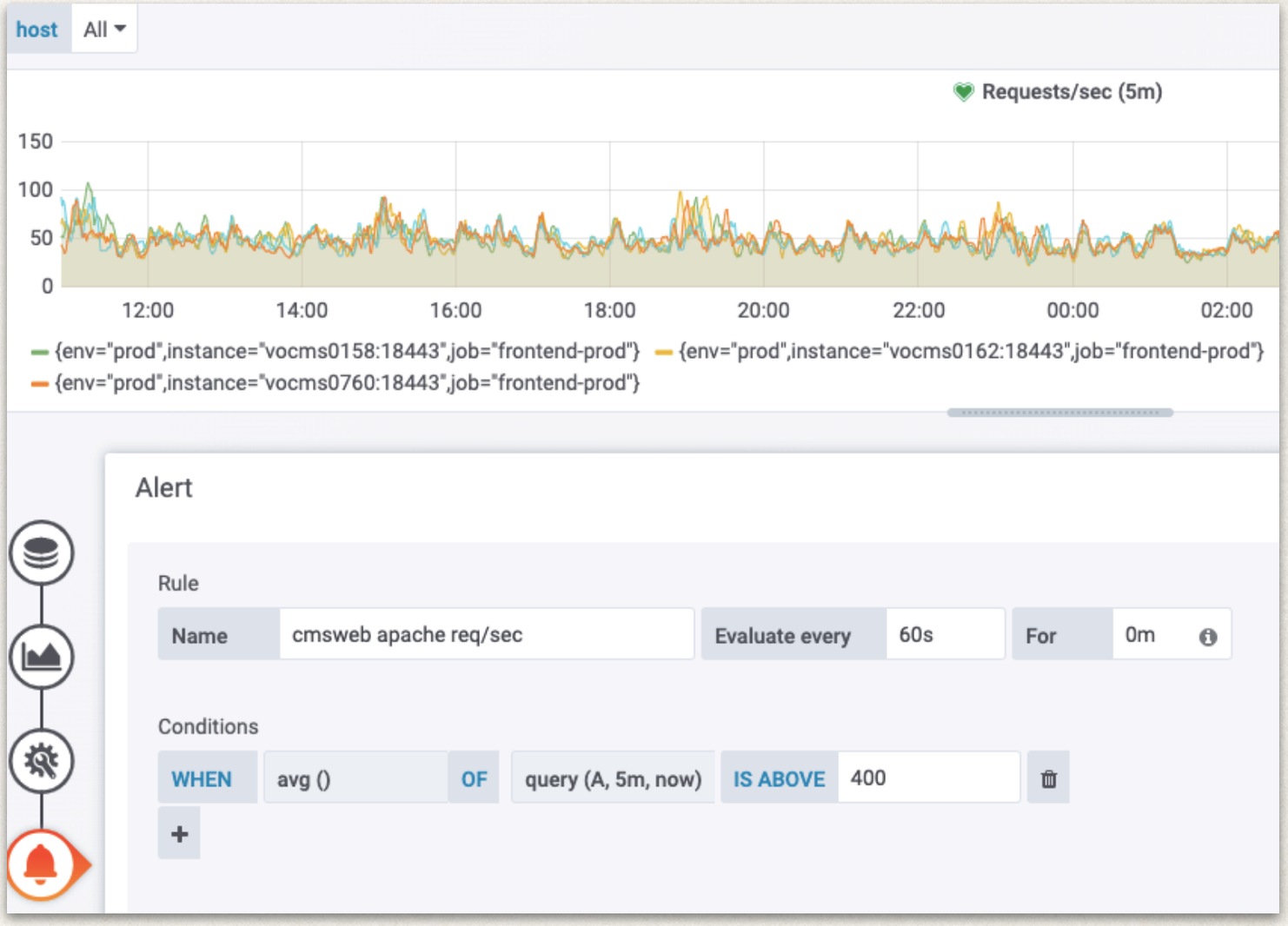
# CMS Monitoring k8s cluster



# Alerts

# Issue

- The Alerts are important mechanism to notify people about existing problem with a service
- So far we used Alerts defined in Grafana for a specific plot
  - alert is triggered upon query condition (ES, InfluxDB or Prometheus queries are supported)
- **If underlying backend goes down we observe flood of alerts**, e.g. the Grafana alerts are not aware of backend (Prometheus) service outages and start firing alarms because there is no metrics
- There is no mechanism to group, silence alerts
  - how we can distinguish real failure vs intermittent or planned outages




Alert is set in Grafana and relies on Prometheus/ES/InfluxDB queries

# Prometheus AlertManager

- Prometheus team provides an independent AlertManager middle-ware which interacts with Prometheus and handle alerts
- It provides grouping, silencing, inhibition as well as custom records, rules, etc.
- **Alerts can be treated as any other software codebase, i.e. we can write rules, chains, routes, etc., and implement unit tests for every alert**
- AlertManager is deployed to CMS Monitoring k8s cluster and integrated it with Prometheus server
- We keep all our alerts implementation in gitlab repository
  - each alert has its own set of unit tests
  - Email, Slack and log notification channels are tested

# Alert rules

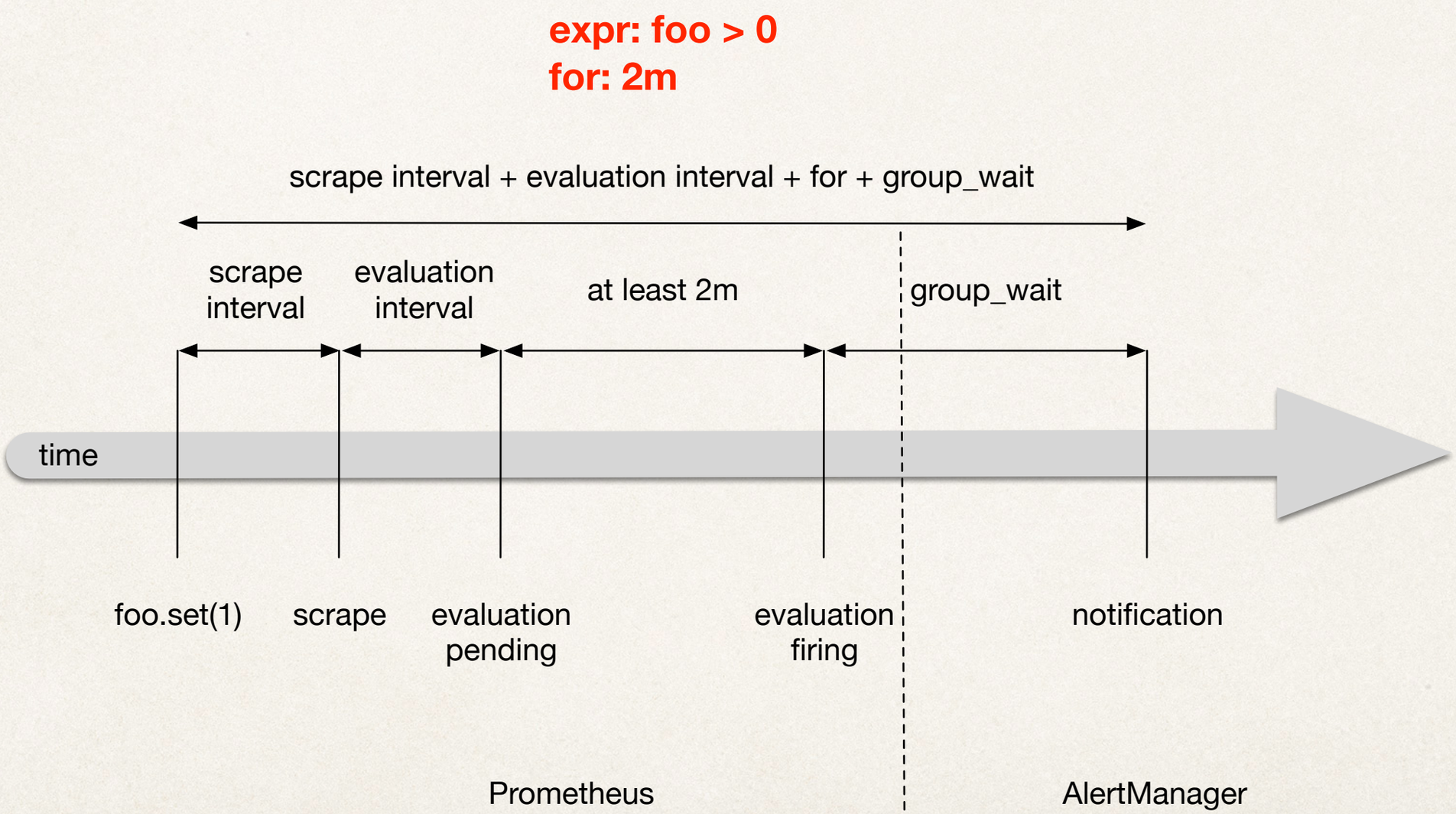
Records

Alerts

```
groups:
- name: dbs
  rules:
  - record: avg_cpu_total
    expr: avg_over_time(dbs_global_exporter_process_cpu_seconds_total[5m])
  - record: avg_open_fds
    expr: avg_over_time(dbs_global_exporter_process_open_fds[5m])

  - alert: high_fds
    expr: avg_open_fds > 300
    for: 1m
    labels:
      severity: dbs
    annotations:
      summary: "DBS global {{ $labels.env }} environment"
      description: "{{ $labels.env }} has high level of fds \
                    (above {{$value}} ) for more than 1m”
```

# Alert flow


expr: foo > 0
for: 2m
scrape interval + evaluation interval + for + group_wait
scrape interval
evaluation interval
at least 2m
group_wait
time
foo.set(1)
scrape
evaluation pending
evaluation firing
notification
Prometheus
AlertManager

# Alert routes

```
amtool config routes –alertmanager.url=http://alert-manager.url.com
Routing tree:
.
└── default-route  receiver: default
    ├── {severity=~"^(?:(.*,)?dbs(,.*)?)$"}  receiver: dbs
    ├── {severity=~"^(?:(.*,)?das(,.*)?)$"}  receiver: das
    ├── {severity=~"^(?:(.*,)?dmwm(,.*)?)$"}  receiver: dmwm
    ├── {severity=~"^(?:(.*,)?couchdb(,.*)?)$"}  receiver: couchdb
    ├── {severity=~"^(?:(.*,)?cmsweb(,.*)?)$"}  receiver: cmsweb
    ├── {severity=~"^(?:(.*,)?frontend(,.*)?)$"}  receiver: frontend
    ├── {severity=~"^(?:(.*,)?test(,.*)?)$"}  receiver: test
    ├── {severity=~"^(?:(.*,)?log(,.*)?)$"}  continue: true  receiver: log
    └── {severity=~"^(?:(.*,)?slack(,.*))$"}  continue: true  receiver: slack
```

```
- alert: service_down
  expr: avg_over_time(couchdb_httpd_up[5m]) < .9
  for: 5m
  labels:
    severity: couchdb, cmsweb, slack
```

# Alerts/routes testing

- Prometheus tools (promtool and amtool) provides comprehensive way to test Alert rules and routes
- Check alert rules

```
promtool check rules *.rules
```

- Perform unit test for set of alerts

```
promtool test rules couchdb.test
```

- Check AlerManager configuration

```
amtool check config a.yaml
```

- Check Alert behavior

```
amtool config routes --config.file=a.yaml
test severity=couchdb
```

```
# Couchdb alert rules
groups:
- name: couchdb
  rules:

  - alert: service_down
    expr: avg_over_time(couchdb_httpd_up[5m]) < .9
    for: 5m
    labels:
      severity: couchdb
    annotations:
      summary: "couchdb {{ $labels.env }} is down"
      description: "{{ $labels.env }} has been down for more than 5m"
```

```
# Unit test for CouchDB alert rules
rule_files:
  - couchdb.rules

evaluation_interval: 1m

tests:
- interval: 1m
  input_series:
  - series: 'couchdb_httpd_up{env="prod"}'
    values: '0+0x100000'
  alert_rule_test:
    - eval_time: 10m
      alertname: service_down
      exp_alerts:
        - exp_labels:
            severity: couchdb
            env: prod
          exp_annotations:
            summary: "couchdb prod is down"
            description: "prod has been down for more than 5m"
```

# Prometheus and AlertManager UI

Prometheus Alerts Graph Status Help

## Alerts

Inactive (25) Pending (0) Firing (1)

Show annotations

/etc/prometheus/couchdb.rules > couchdb

**service_down** (1 active)

**node_down** (0 active)

/etc/prometheus/crabserver.rules > crabserver

**service_down** (0 active)

/etc/prometheus/das.rules > das

**high_cpu_usage** (0 active)

```
alert: high_cpu_usage
expr: avg_over_time(das2go_cpu_percent[5m])
  > 90
for: 5m
labels:
  severity: das
annotations:
  description: '{{ $labels.env }} CPU usage above 90% for more than 5m'
  summary: DAS {{ $labels.env }} has high CPU usage
```

Alertmanager Alerts Silences Status Help New Silence

Filter Group Receiver: All Silenced Inhibited

+ Silence

Custom matcher, e.g. env="production"

+ Expand all groups

alertname="service_down" + 1 alert

12:38:35, 2020-01-07 (UTC) − Info Source Silence

**description:** prod has been down for more than 5m

**summary:** couchdb prod is down

env="prod" + instance="vocms0741:15984" + job="couchdb-prod" + severity="couchdb" +

# Alert log service

```
2020/01/06 15:43:00 POST / HTTP/1.1
Host: httpgo.default.svc.cluster.local:8888
Content-Length: 972
Content-Type: application/json
User-Agent: Alertmanager/0.20.0

{"receiver":"couchdb","status":"firing","alerts":
[{"status":"firing","labels":
{"alertname":"service_down","env":"prod","instance":"vocms0741:15984","job"
:"couchdb-prod","severity":"couchdb"},"annotations":{"description":"prod
has been down for more than 5m","summary":"couchdb prod is
down"},"startsAt":"2020-01-06T14:42:50.707209396Z","endsAt":"0001-01-01T00:
00:00Z","generatorURL":"http://prometheus-66b5d8cd58-9tq6b:9090/graph?
g0.expr=avg_over_time%28couchdb_httpd_up%5B5m%5D%29+
%3C+0.9\u0026g0.tab=1","fingerprint":"89201b12a5fd9544"}],"groupLabels":
{"alertname":"service_down"},"commonLabels":
{"alertname":"service_down","env":"prod","instance":"vocms0741:15984","job"
:"couchdb-prod","severity":"couchdb"},"commonAnnotations":
{"description":"prod has been down for more than 5m","summary":"couchdb
prod is down"},"externalURL":"http://cms-monitoring.cern.ch:
30093","version":"4","groupKey":"{}/{severity=~\"^(?:(.*,)?couchdb(,.*)?)$
\"}:{alertname=\"service_down\"}"}
```

# Email notification

```
groups:
- name: alert
  rules:

  - alert: service_down
    expr: das_requests > 10000
    for: 30s
    labels:
      severity: test
    annotations:
      summary: "Instance {{ $labels.instance }} down, value {{ $value }}"
      description: "{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 30 seconds."
```

- Supports annotations with templates
- Capture details of attributes (instance, job, severity levels, etc.)
- We can provide as much details in alerts as possible, e.g. current value of the metric, etc.

**3 alerts for alertname=service_down**

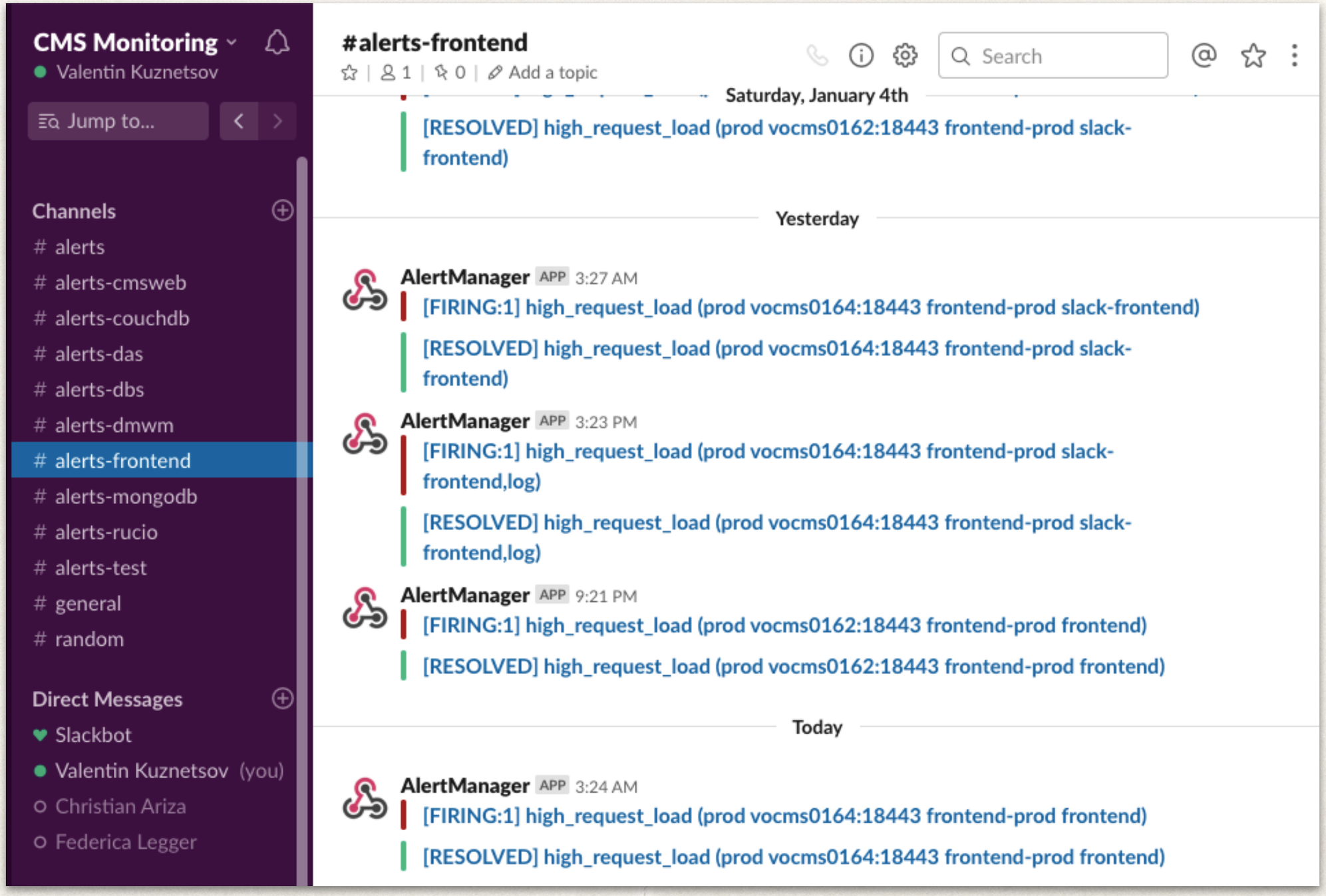
**View In AlertManager**

**[3] Firing**

**Labels**
alertname = service_down
env = preprod
instance = vocms0132:18217
job = das-preprod
severity = test
**Annotations**
description = vocms0132:18217 of job das-preprod has been down for more than 30 seconds.
summary = Instance vocms0132:18217 down, value 1.088847e+06
Source

**Labels**
alertname = service_down
env = preprod
instance = vocms0731:18217
job = das-preprod
severity = test
**Annotations**
description = vocms0731:18217 of job das-preprod has been down for more than 30

# Slack notifications



# Advanced features

- Define custom (complex) metrics via record definitions within alert rules
- Use templates to customize alert messages, provide annotations, details about system attributes (instance name, port, etc.)
- Define rules when to NOT send the alert, e.g. during public holidays
- We can fire alerts from CLI, e.g. support failing cron jobs (added to CMSSpark jobs)

```
amtool alert add ALERTNAME severity=test --annotation=summary="alert summary"
--alertmanager.url=http://alert-manager.url.com --end="`date --rfc-3339=ns`"
```

- Treat alerts as a code-base, e.g. with alert unit tests and simulate its behavior
- Perform alerts grouping and/or silencing, the former can be handy to send single notification about underlying problem with a system, while later can be used to perform maintenance tasks without alerting about infrastructure outages

# Summary

- CMS Monitoring group performs several R&D activities to enhance capability of CERN MONIT infrastructure
- We explored NATS service for real-time monitoring, VictoriaMetrics for long term storage of Prometheus, and AlertManager for our Alert needs
- We deployed and run all services in dedicated k8s clusters, one cluster for NATS and another for Monitoring tools (Prometheus, VM, AlertManager)
- Distributed agents can feed data about their workflows into NATS and client's subscribers can subscribe to any topic w/o any configuration
  - for our standard subscribers we capture messages and feed them back to VictoriaMetrics which is later used in MONIT as data-source for building various dashboards
- Alerts are treated as code-base, we can write sophisticated rules, e.g. avoid sending alerts on public holidays, calculate and evaluate custom metrics, tests all alerts via unit tests, etc.

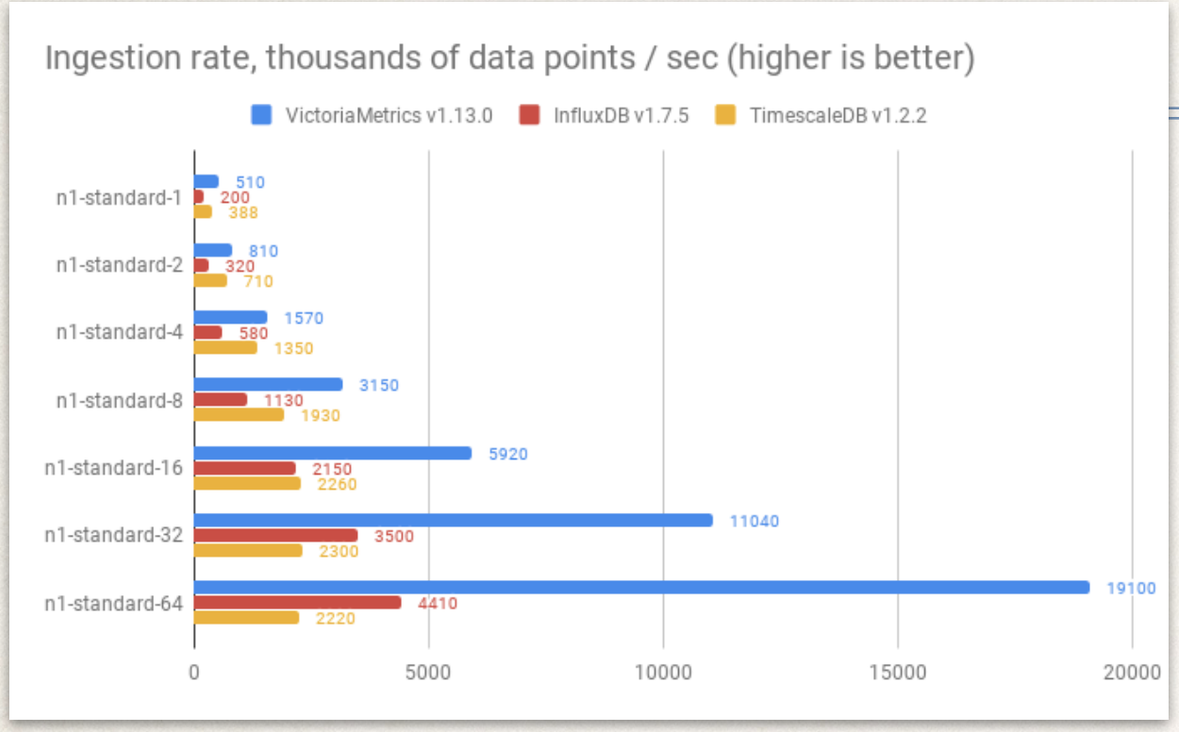
# Back-up slides

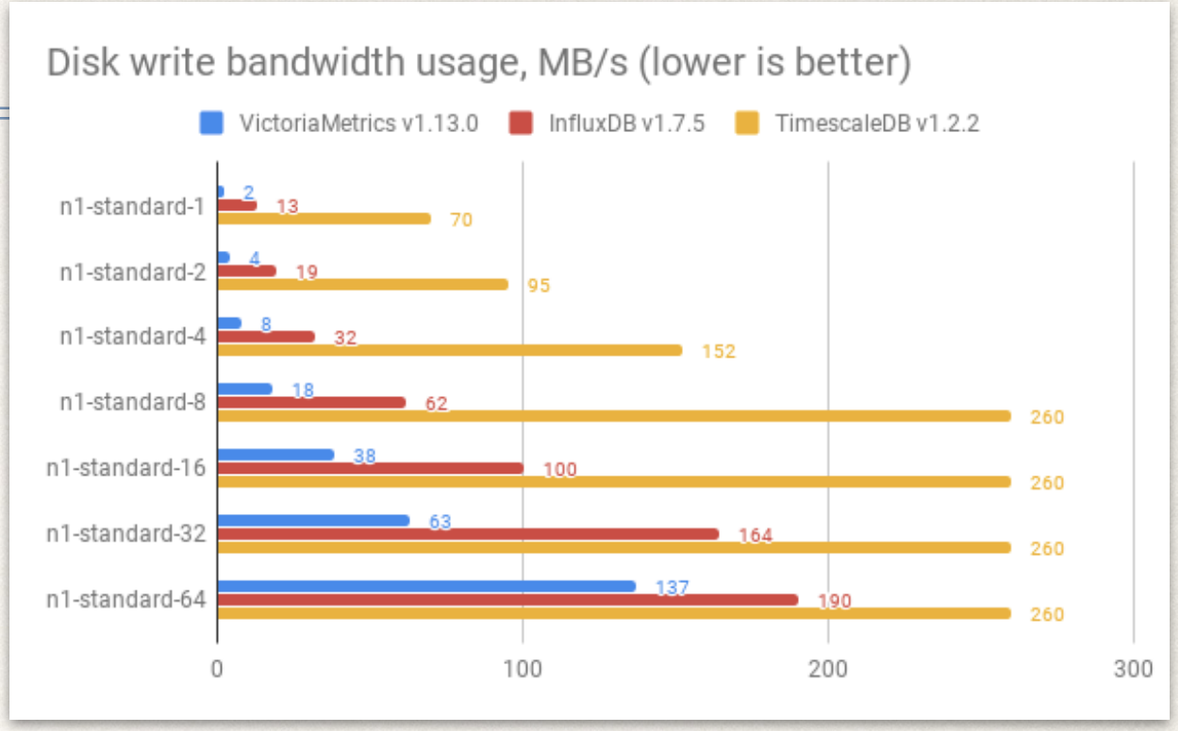
# Introducing VictoriaMetrics

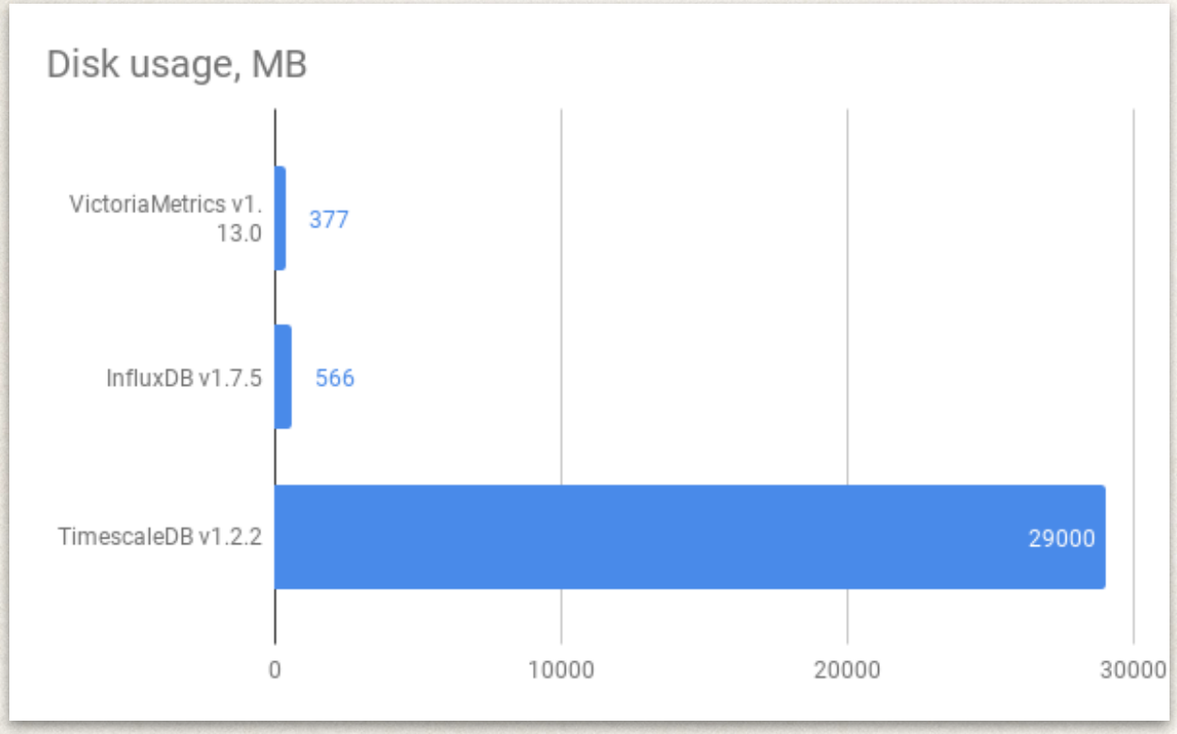
- VictoriaMetrics is fast, cost-effective and scalable time-series database. It can be used as long-term remote storage for Prometheus.
- High performance and good scalability for both inserts and selects. Outperforms InfluxDB and TimescaleDB by up to 20x, e.g. insert benchmarks based on 4M data-points injection:
  - VictoriaMetrics: 2.2M datapoints/sec; RAM usage: 6GB; final data size on disk: 3GB.
  - InfluxDB: 330K datapoints/sec; RAM usage: 20.5GB; final data size on disk: 18.4GB
- Uses 10x less RAM than InfluxDB when working with millions of unique time series (aka high cardinality)
- High data compression, so up to 70x more data points may be crammed into limited storage comparing to TimescaleDB
- Supports Prometheus querying API, so it can be used as Prometheus drop-in replacement in Grafana. Additionally, VictoriaMetrics extends PromQL with opt-in useful features

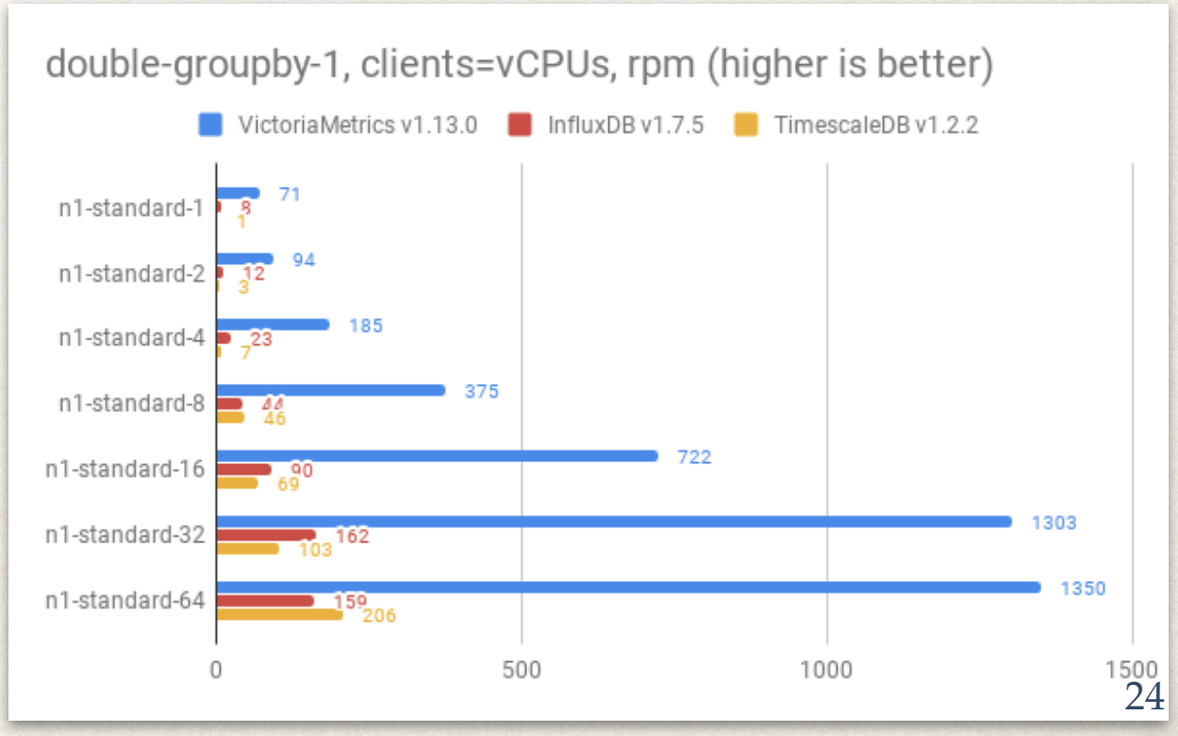
Ref

# VictoriaMetrics performance



# VictoriaMetrics use-case

- Use VictoriaMetrics as back-end for Prometheus server
  - drop-in replacement in Grafana, i.e. we can build dashboard based on VictoriaMetrics data-source
  - Extend retention period of our Prometheus metrics (cmsweb, NATS, etc.)
- We can inject NATS messages into VictoriaMetrics directly

```
curl -H 'Content-Type: application/json' -d '{"metric":"cms.test.exitCode", "value":1,
"tags":{"exitCode": "8021", "site":"T2_US", “task”:"test"}}' http://vm-url.com/api/put
```

- We can query VictoriaMetrics via HTTP APIs (to export **/api/v1/export**, to query **/api/v1/query**):

```
curl -G ‘http://vm-url.com/api/v1/export' -d
‘match={__name__="cms.wmarchive.exitCode",site="T1_ES_PIC"}'
```

- Results are records related to our sites:

```
{“metric”:
{“__name__”:”cms.wmarchive.exitCode”,”campaign":"RunIIFall17NanoAODv6","dataset":"NA",
"exitCode":"8021","site":"T1_ES_PIC","task":"/pdmvserv_task_HIG-
RunIIFall17NanoAODv6-01107__v1_T_191129_001440_134/HIG-
RunIIFall17NanoAODv6-01107_0"},"values":[1,1],"timestamps":[1575658878000,1575659828000]}
```

# VictoriaMetrics benefits in queries

- VictoriaMetrics extends Prometheus QL, see full list here
- Percentage of time with more than 10 errors per second for the last hour:

```
avg_over_time((rate(errors_total[5m]) > bool 10)[1h:1m])
```

- 95th percentile of network bandwidth for the last hour:

```
quantile_over_time(0.95, rate(node_network_receive_bytes_total[5m])[1h:1m])
```

- The increase rate of requests per second for the last 30 minutes:

```
increases_over_time(rate(requests_total[5m])[30m:])
```

- Number of cache requests for the previous day

```
rate(hits_total[5m] offset 1d) + rate(miss_total[5m] offset 1d)
```

- Request/sec (rps) graphs for “today”, “yesterday” and “week ago”

```
with ( rps = rate(requests_total[5m]) ) union( label_set(rps, “req”, “today”),
label_set(rps offset 1d, “req”, “yesterday”), label_set(rps offset 7d, “req”, “1w ago”) )
```