Querying Metrics with PromQL

Elton Stoneman
CONSULTANT & TRAINER

@EltonStoneman | blog.sixeyed.com

```
worker_jobs_total
{instance="i1",
 status="processed} 150
```

```
sum
without(instance, status)
(worker_jobs_total)
```

instance	job_status	job_count
i1	processed	150

```
SELECT
SUM(job_count) FROM
job_summaries
```

170K rows /24hr
- per status
- per instance

~ 50b per row

instance	job_status	job_count	timestamp
i1	processed	150	1592210327

```
worker_jobs_total
{instance="i1",
 status="processed}
```

```
150 @ 1592210327
158 @ 1592210357
210 @ 1592210387
235 @ 1592210417
```

One time series
- per status
- per instance

< 2b per sample

```
http_request_seconds
{status="200",le="5"} 2300

http_request_seconds
{status="200",le="10"} 3760
```

status	started	finished
200	1592210327	1592210334
200	1592210327	1592210332
200	1592210327	1592210329
200	1592210327	1592210338

...

12 time series
- per status

86M rows /24hr
- per status

```
histogram_quantile(
  0.90,
  sum without(code,instance)(
   rate(http_request_seconds[5m]
)))
```

```
select top (1) percentile_cont(0.90)
 within group (order by avg_duration)
 over () as percentile_90
from (select avg(duration) as avg_duration,
      percentile_cont(0.90)
        over (order by avg(duration))
       as percentile_90
      from t
      group by status_code, instance
     ) t;
```

Exploring PromQL Syntax

```
worker_jobs_active
```

→

```
worker_jobs_active
  {instance="i1",job="batch"} 84
worker_jobs_active
  {instance="i2",job="batch"} 51
```

```
worker_jobs_active
 {instance="i1"}
```

→

```
worker_jobs_active
  {instance="i1",job="batch"}  84
```

```
worker_jobs_active
 {job="batch",instance=~"i.*"}
```

→

```
worker_jobs_active
  {instance="i1",job="batch"} 84
worker_jobs_active
  {instance="i2",job="batch"} 51
```

```
worker_jobs_active[3m]
```

```
worker_jobs_active
  {instance="i1",job="batch"}
  70 @1592319615.353
  19 @1592319675.357
  34 @1592319735.352

worker_jobs_active
  {instance="i2",job="batch"}
  95 @1592319645.816
  56 @1592319705.818
  55 @1592319765.823
```

```
worker_jobs_active
 {instance="i1"}
 [3m]
```

→

```
worker_jobs_active
  {instance="i1",job="batch"}
  70 @1592319615.353
  19 @1592319675.357
  34 @1592319735.352
```

```
sum(worker_jobs_active)
```

→

```
135
```

```
sum without(job)
(worker_jobs_active)
```

→

```
worker_jobs_active
  {instance="i1"} 80

worker_jobs_active
  {instance="i2"} 55
```

```
delta(worker_jobs_active[1h])
```

```
worker_jobs_active
  {instance="i1"} 18.305058891159

worker_jobs_active
  {instance="i2"} 8.1355748348591
```

```
avg(delta(worker_jobs_active[1h]))
```

```
13.220316863009444
```

```
rate(worker_jobs_total[5m])
```

```
worker_jobs_active
  {instance="i1",status="p"} 47.4

worker_jobs_active
  {instance="i1 ",status="f"} 4.9

worker_jobs_active
  {instance="i2",status="p"} 46.2

worker_jobs_active
  {instance="i2 ",status="f"} 4.7
```

```
sum(rate(worker_jobs_total[5m]))
```

```
103.43267213350337
```

Operator

Metric name

Range

```
sum(rate(worker_jobs_total{instance="i1"}[5m]))
```

Function

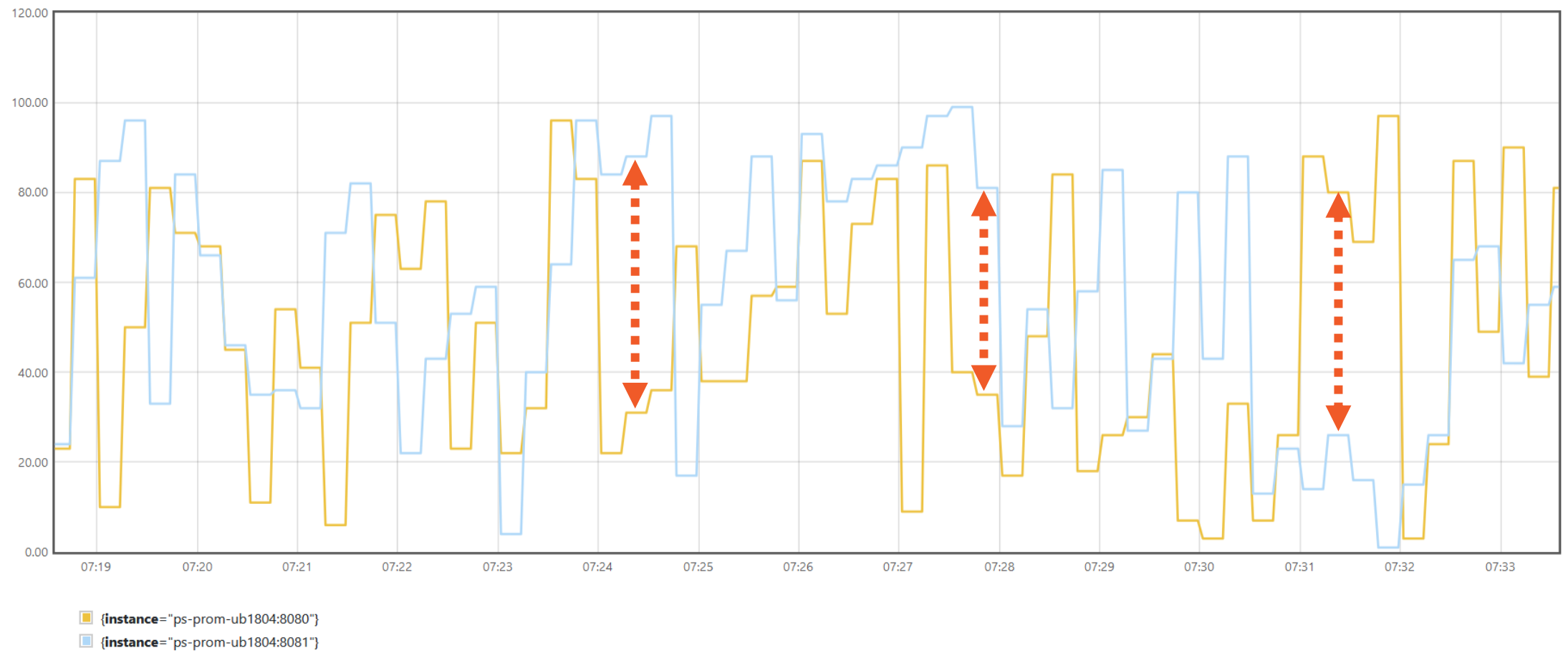
Selector

Demo

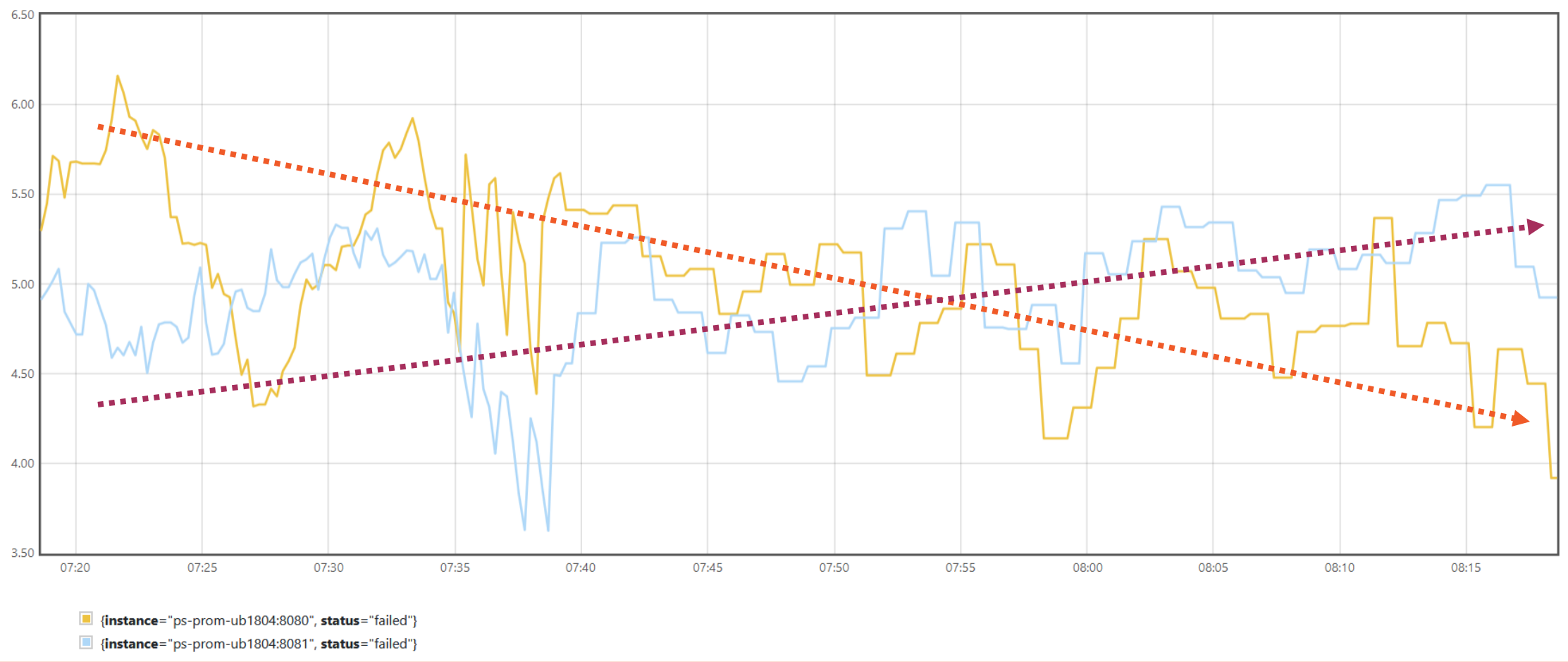
Querying gauges and counters

- PromQL expressions
- Selectors and ranges
- Operators and functions

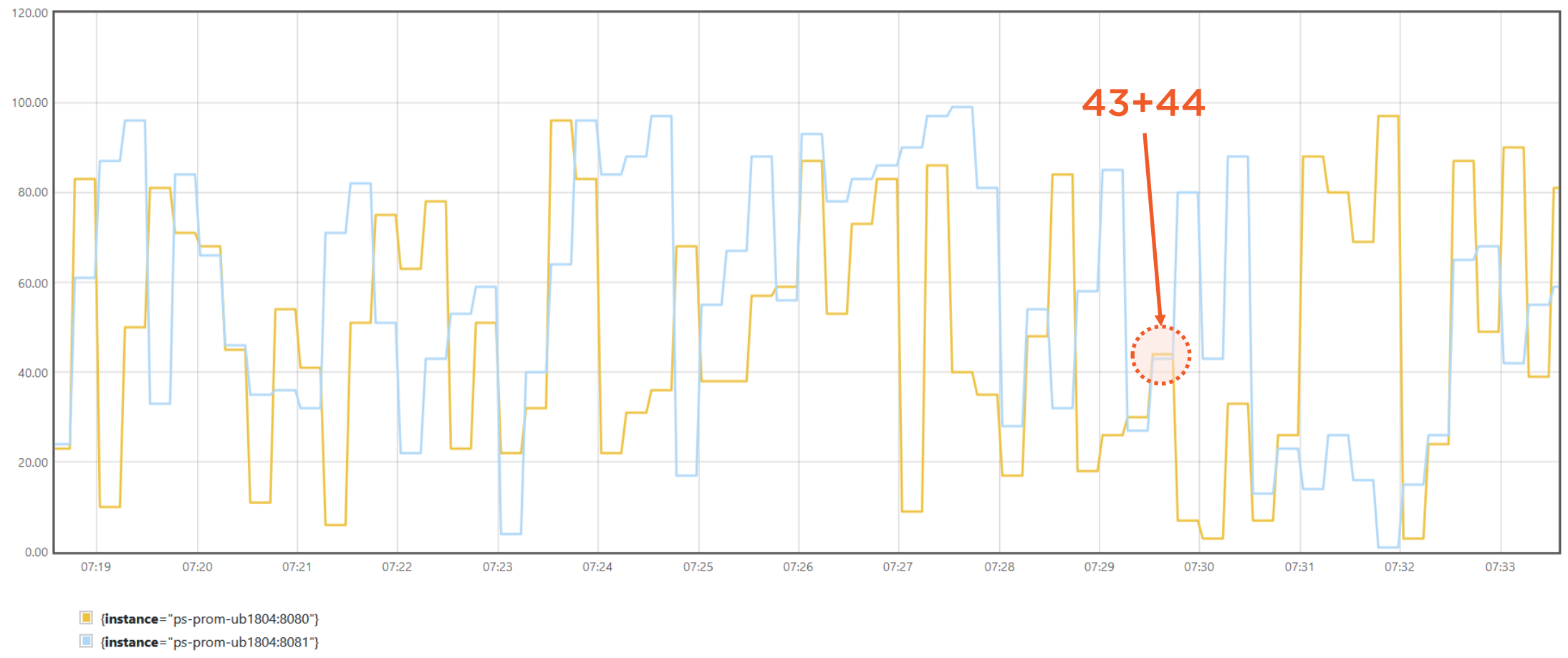
sum without(job, os, runtime) (worker_jobs_active)



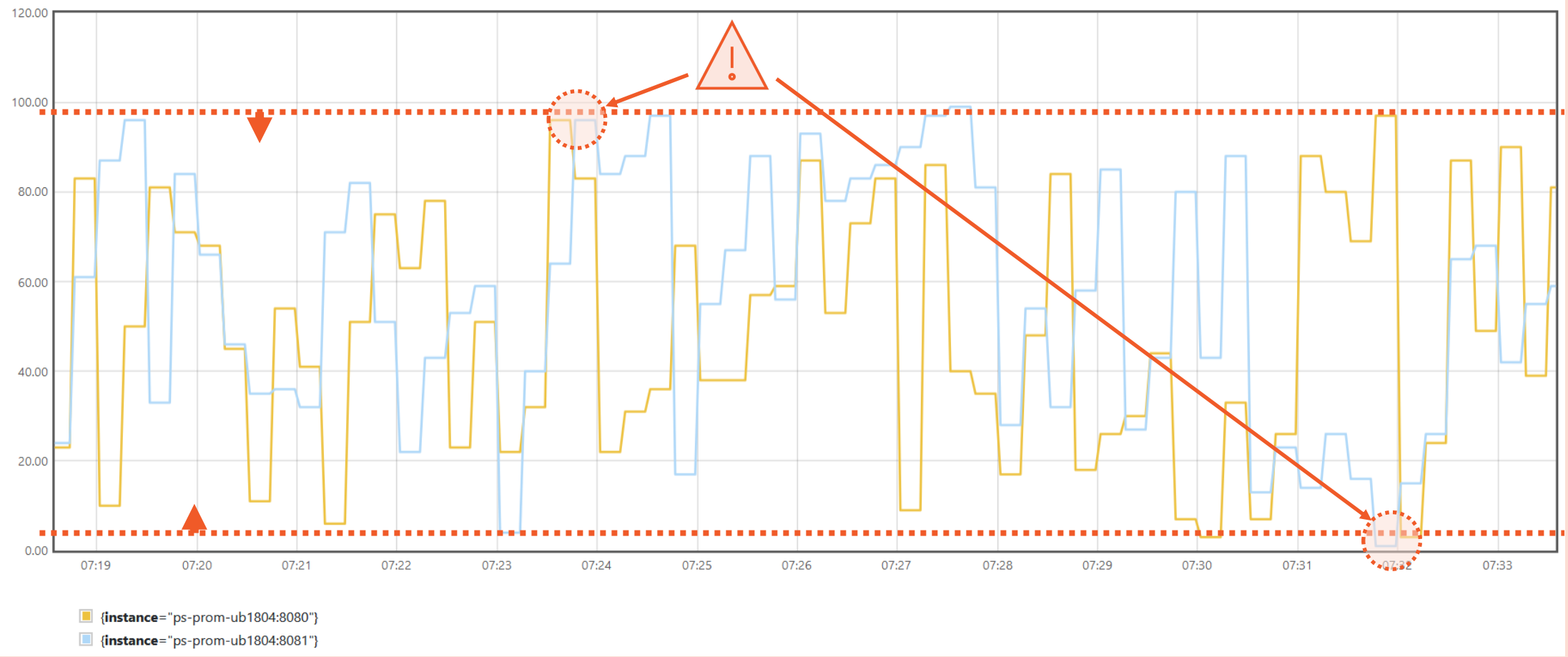
```
sum without(job, os, runtime) (rate(worker_jobs_total{status="failed"}[5m]))
```



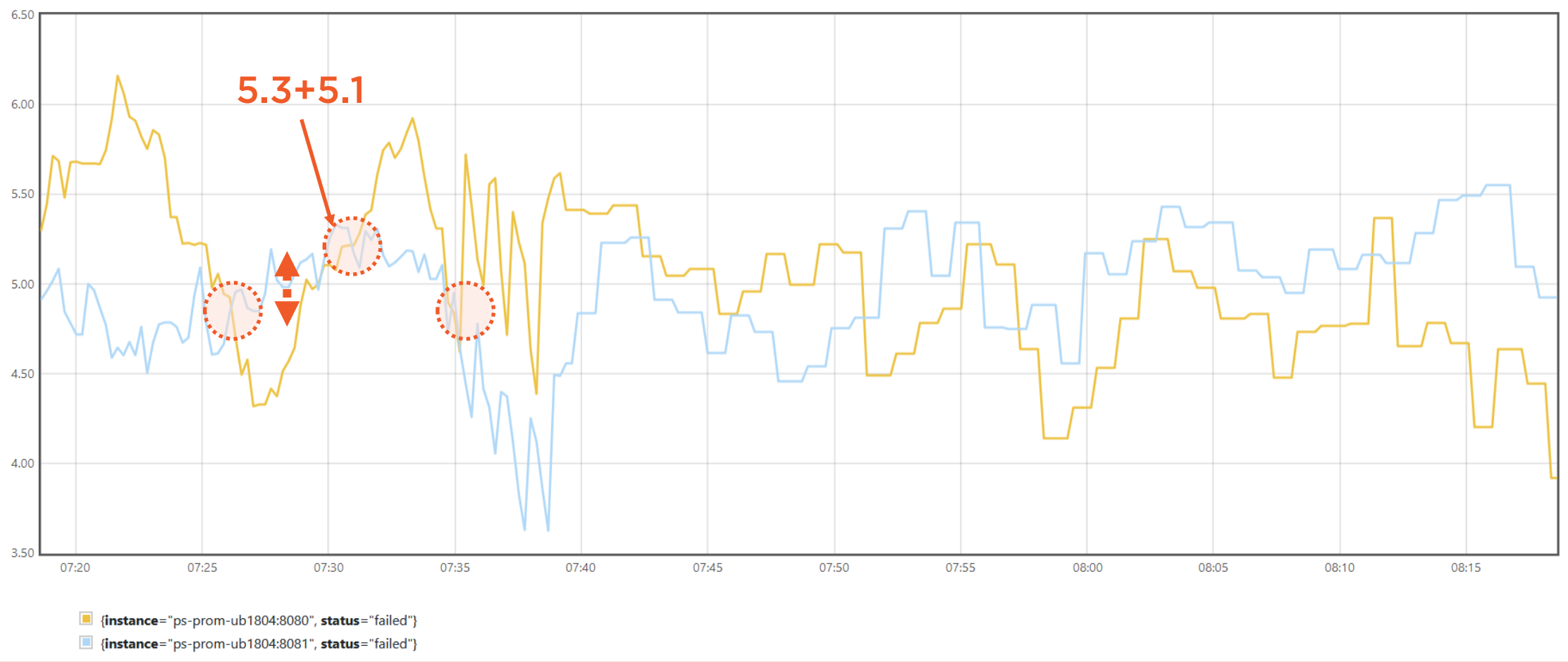
Active jobs over time



Active jobs over time



Failed jobs over time

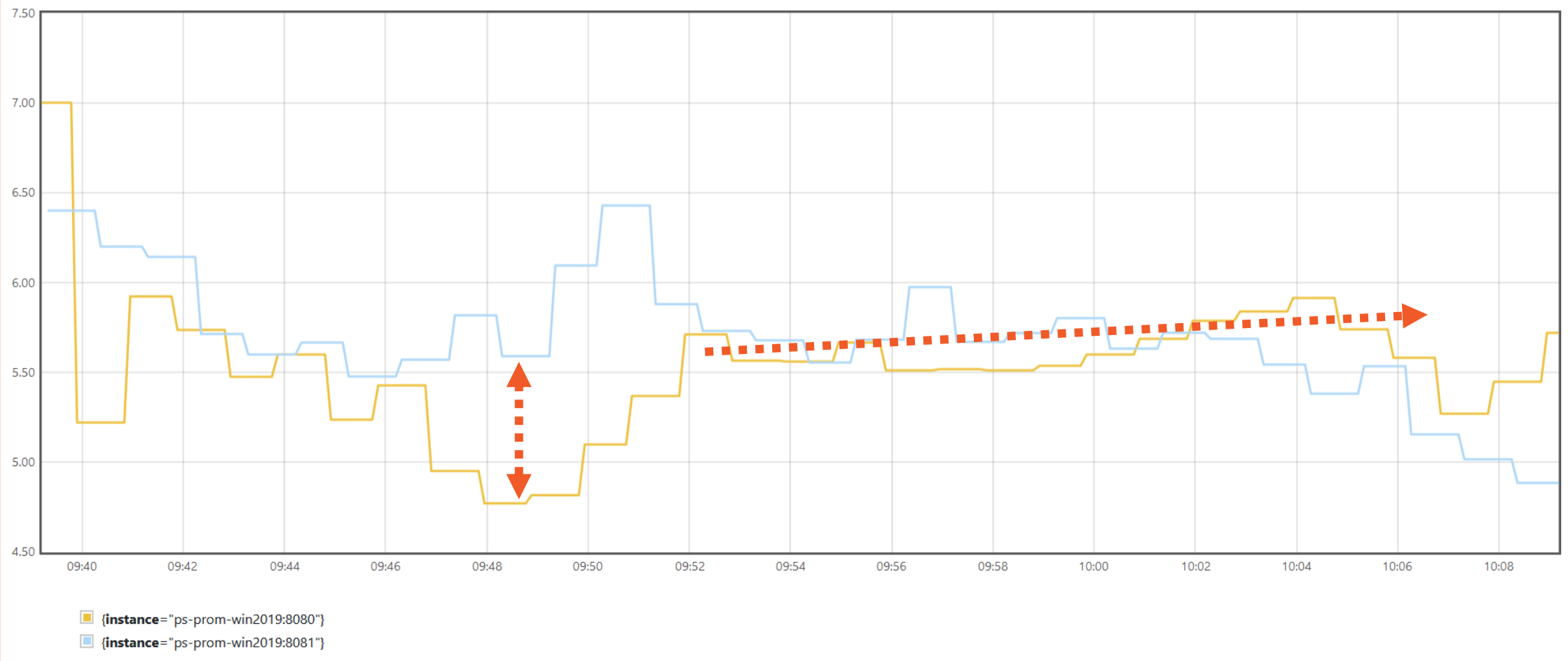


Demo

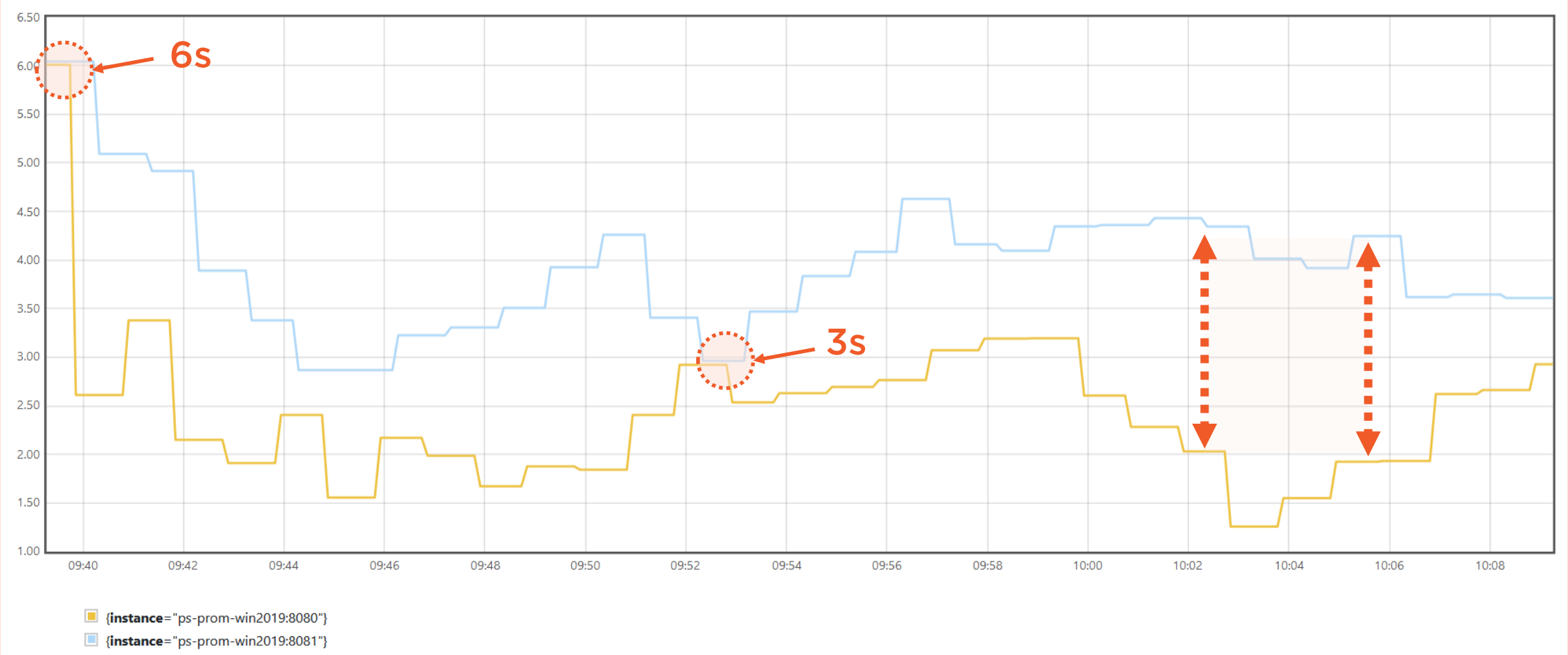
Querying summaries and histograms

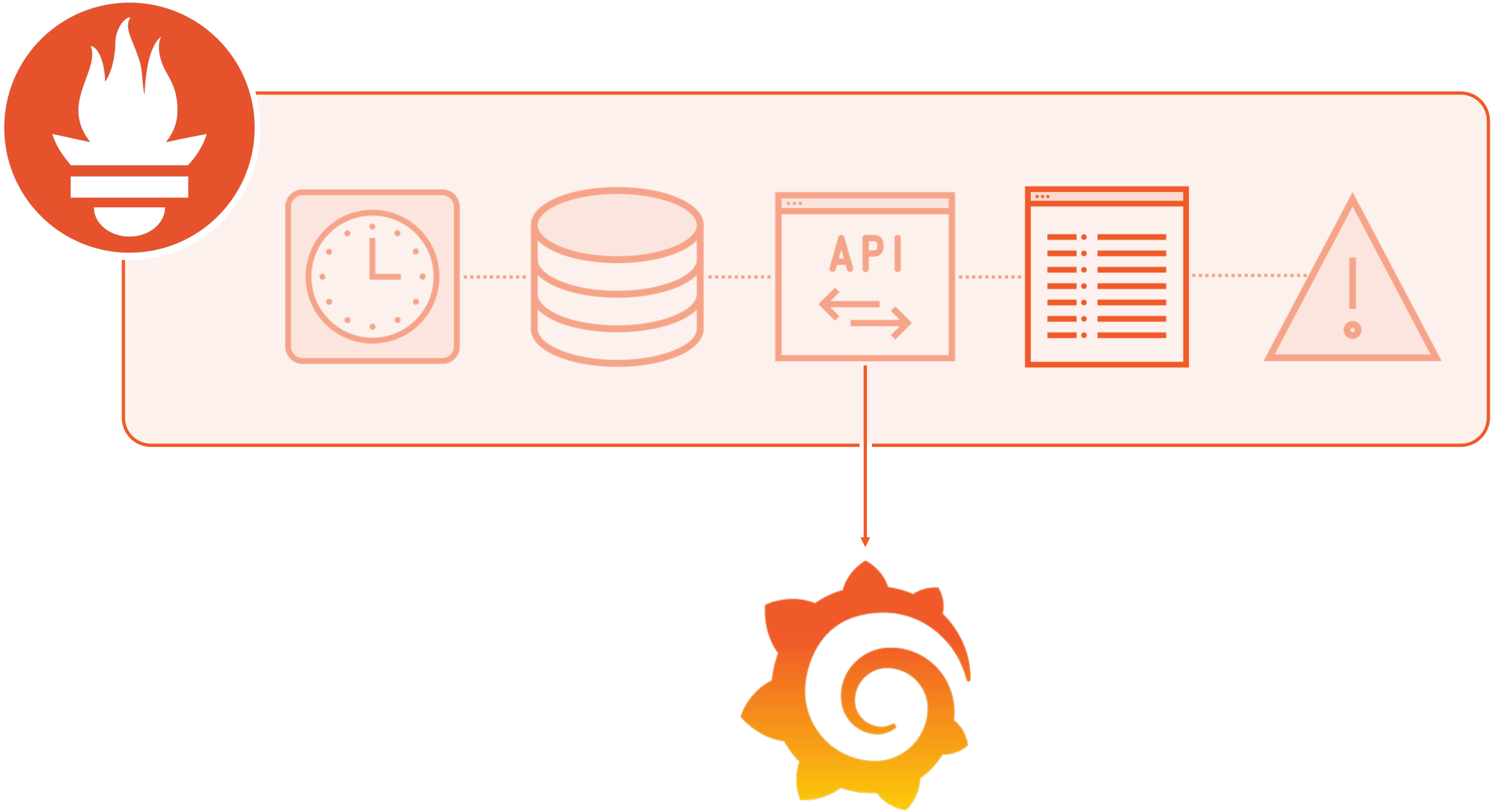
- Averaging summaries
- Percentiles from histograms
- Web app performance

```
sum without(job, os, runtime) (rate(web_delay_seconds_sum[5m])) /
sum without(job, os, runtime) (rate(web_delay_seconds_count[5m]))
```



```
histogram_quantile(0.90, sum without(code, job, method, os, runtime)
  (rate(http_request_duration_seconds_bucket{code="200"}[5m])))
```




API

Batch processor

Total Active Jobs

101

Active Jobs

45 ps-prom-ub1804:8080

57 ps-prom-ub1804:8081

Jobs Processed

55.0 52.5 50.0 47.5 45.0 42.5

12:16 12:17 12:18 12:19 12:20

— ps-prom-ub1804:8080 — ps-prom-ub1804:8081

Jobs Failed

5.50 5.25 5.00 4.75 4.50 4.25 4.00

12:16 12:17 12:18 12:19 12:20

— ps-prom-ub1804:8080 — ps-prom-ub1804:8081

Memory

5.665 MB

CPU

3.56 ms

Web App

Total Active Requests

2.7

Active Requests

4.0 3.0 2.0 1.0 0

12:16 12:17 12:18 12:19 12:20

— ps-prom-win2019:8080 — ps-prom-win2019:8081

90th Percentile Response

8 seconds 6 seconds 4 seconds 2 seconds 0 seconds

12:16 12:17 12:18 12:19 12:20

— ps-prom-win2019:8080 — ps-prom-win2019:8081

Average Slow Mode Delay

6 seconds, 500 milliseconds
6 seconds, 250 milliseconds
6 seconds, 0 milliseconds
5 seconds, 750 milliseconds
5 seconds, 500 milliseconds
5 seconds, 250 milliseconds

12:16 12:17 12:18 12:19 12:20

— ps-prom-win2019:8080 — ps-prom-win2019:8081

Memory

23.7 MB

CPU

3.28 ms

Info

Linux Server

linux	ps-prom-ub1804:9100	4.15.0-101-generic

Windows Server

windows	ps-prom-win2019:9182	10.0.17763

Batch app

Instance	App Version	Assembly Name	.NET Version
ps-prom-ub1804:8080	1.6.2	PrometheusDemo.Ba...	3.1.5
ps-prom-ub1804:8081	1.6.2	PrometheusDemo.Ba...	3.1.5

Web app

Instance	App Version	Assembly Name	.NET Version
ps-prom-win2019:8081	2.0.1	PrometheusDemo.Web	3.1.5
ps-prom-win2019:8080	2.0.1	PrometheusDemo.Web	3.1.5

Demo

Prometheus API and Grafana

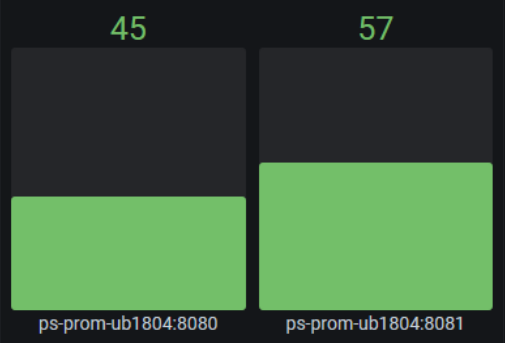
- Querying with the API
- Running Grafana
- Visualizing PromQL queries

Batch processor

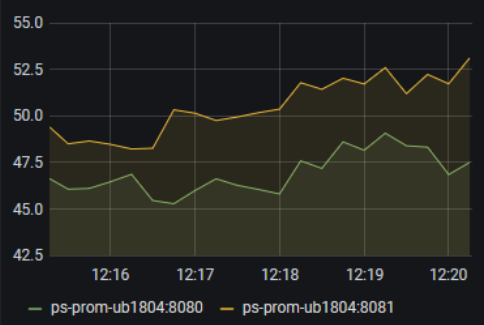
Total Active Jobs

101

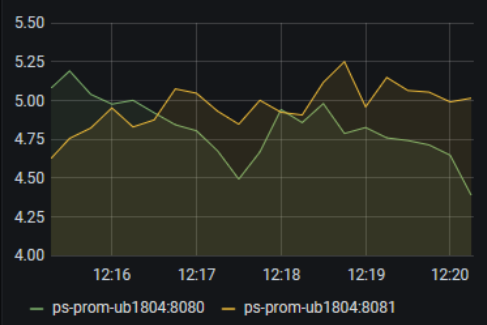
Active Jobs



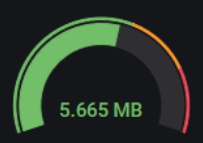
Jobs Processed



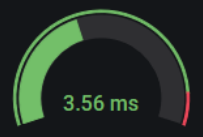
Jobs Failed



Memory



CPU

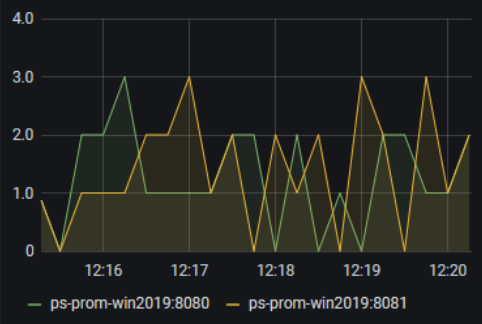


Web App

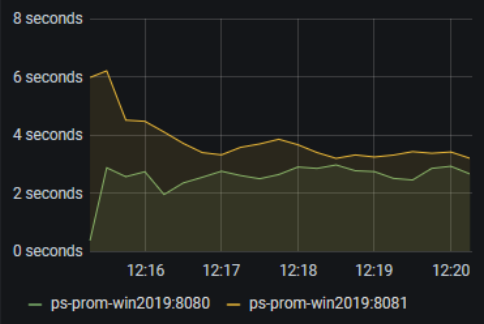
Total Active Requests

2.7

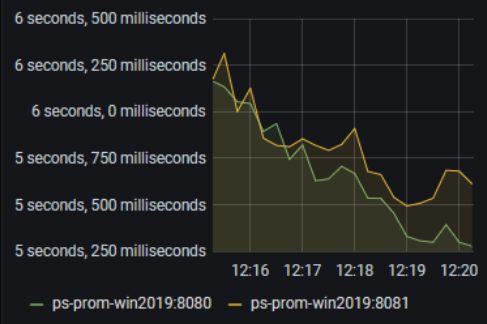
Active Requests



90th Percentile Response

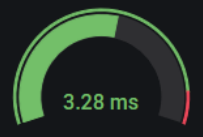


Average Slow Mode Delay



Memory

CPU



Info

Linux Server

linux	ps-prom-ub1804:9100	4.15.0-101-generic

Windows Server

windows	ps-prom-win2019:9182	10.0.17763

Batch app

Instance	App Version	Assembly Name	.NET Version
ps-prom-ub1804:8080	1.6.2	PrometheusDemo.Ba...	3.1.5
ps-prom-ub1804:8081	1.6.2	PrometheusDemo.Ba...	3.1.5

Web app

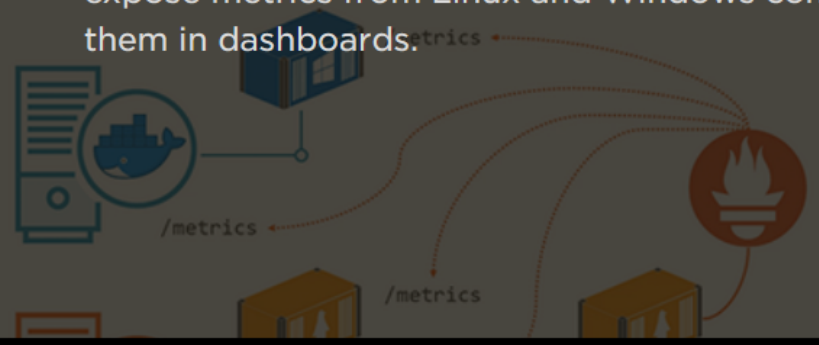
Instance	App Version	Assembly Name	.NET Version
ps-prom-win2019:8081	2.0.1	PrometheusDemo.Web	3.1.5
ps-prom-win2019:8080	2.0.1	PrometheusDemo.Web	3.1.5

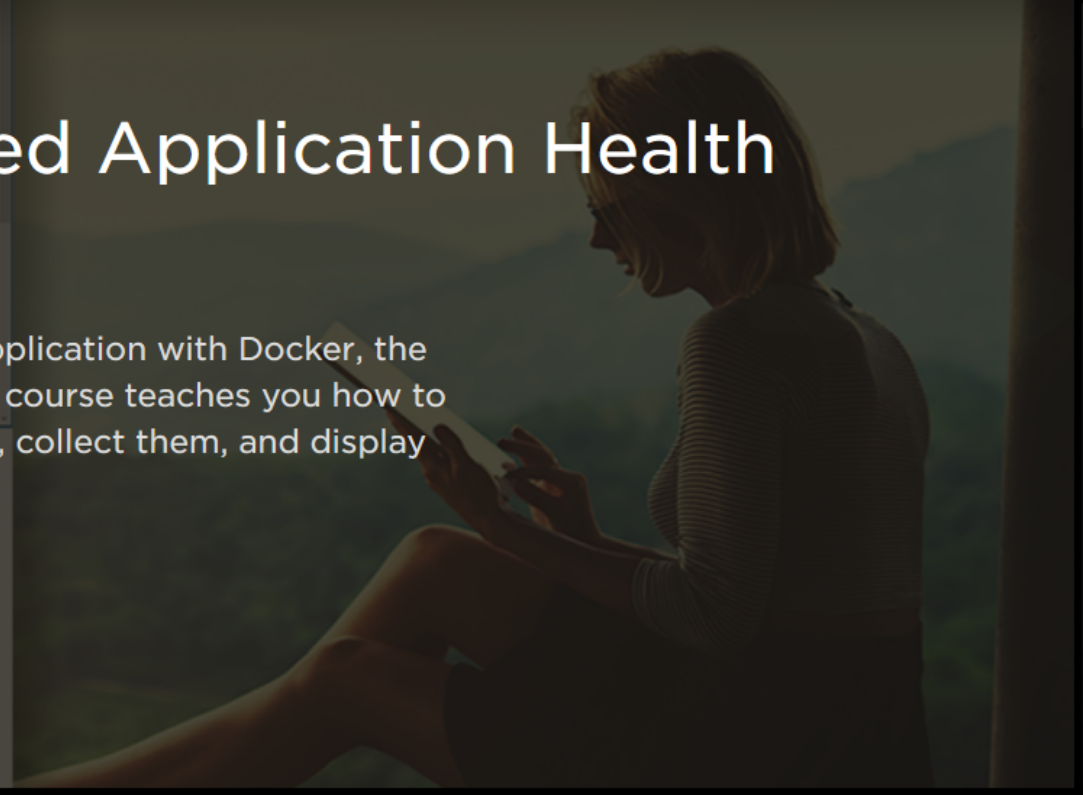

Batch processor
Total Active Jobs
101
Active Jobs
45
57
ps-prom-ub1804:8080
ps-prom-ub1804:8081
Jobs Processed
55.0
52.5
50.0
47.5
45.0
42.5
12:16
12:17
12:18
12:19
12:20
ps-prom-ub1804:8080
ps-prom-ub1804:8081
Jobs Failed
5.50
5.25
5.00
4.75
4.50
4.25
4.00
12:16
12:17
12:18
12:19
12:20
ps-prom-ub1804:8080
ps-prom-ub1804:8081
Memory
5.665 MB
CPU
3.56 ms
Web App
Total Active Requests
2.7
Active Requests
4.0
3.0
2.0
1.0
0
12:16
12:17
12:18
12:19
12:20
ps-prom-win2019:8080
ps-prom-win2019:8081
90th Percentile Response
8 seconds
6 seconds
4 seconds
2 seconds
0 seconds
12:16
12:17
12:18
12:19
12:20
ps-prom-win2019:8080
ps-prom-win2019:8081
Average Slow Mode Delay
6 seconds, 500 milliseconds
6 seconds, 250 milliseconds
6 seconds, 0 milliseconds
5 seconds, 750 milliseconds
5 seconds, 500 milliseconds
5 seconds, 250 milliseconds
12:16
12:17
12:18
12:19
ps-prom-win2019:8080
ps-prom-win2019:8081
Memory
23.7 MB
CPU
3.28 ms
Info
Linux Server
linux
ps-prom-ub1804:9100
4.15.0-101-generic
Windows Server
windows
ps-prom-win2019:9182
10.0.17763
Batch app
Instance
App Version
Assembly Name
.NET Version
ps-prom-ub1804:8080
1.6.2
PrometheusDemo.Ba...
3.1.5
ps-prom-ub1804:8081
1.6.2
PrometheusDemo.Ba...
3.1.5
Web app
Instance
App Version
Assembly Name
.NET Version
ps-prom-win2019:8081
2.0.1
PrometheusDemo.Web
3.1.5
ps-prom-win2019:8080
2.0.1
PrometheusDemo.Web
3.1.5

Monitoring Containerized Application Health with Docker

★★★★☆ By Elton Stoneman

You can add consistent monitoring to your whole application with Docker, the same for every container in every environment. This course teaches you how to expose metrics from Linux and Windows containers, collect them, and display them in dashboards.





Architecting Monitoring for Containerized Applications

- Course Intro and Module Overviews
- Understanding Monitoring for Containerized Apps
- Demo: Monitoring Containerized Applications
- Introducing Prometheus and Grafana
- Demo: Monitoring with Prometheus
- Consistent Monitoring with Containers
- Demo: Docker Platform Metrics

Course info

Rating	★★★★☆ (79)
Level	Intermediate
Updated	Aug 8, 2018
Duration	2h 43m

Description

It's easy to run new and old applications in Docker, but you can't put containerized apps into production without monitoring. In this course, Monitoring Containerized Application Health with Docker, you'll learn how to implement effective monitoring for Linux and Windows containers. First, you'll learn how to gather and visualize metrics from containers using Prometheus and Grafana. Next, you'll see how to add metrics to your application, and export metrics from the Java and .NET runtimes and from the Docker platform. Finally, you'll explore how to build an effective dashboard with a single view over the health of your whole application. When you're finished with this course, you'll be ready to add monitoring to your application and move confidently to production.

https://is.gd/eduweh

Site Reliability Engineering (SRE): The Big Picture

★★★★★ By Elton Stoneman

Site Reliability Engineering (SRE) is how Google runs production systems, promoting high availability with high velocity and removing operational toil. It achieves the same goals as DevOps without the culture shift.

Course info

Rating	★★★★★ (29)
Level	Beginner
Updated	Mar 5, 2020
Duration	1h 41m

Description

Site Reliability Engineering (SRE) is a set of principles and practices that supports software delivery - keeping production systems stable and still delivering new features at speed. In this course, Site Reliability Engineering (SRE): The Big Picture, you'll get a thorough overview of how SRE works and why it's a good choice for many organisations. First, you'll learn the differences between SRE, DevOps, and traditional operations. Next, you'll discover how engineering practices help to reduce toil and provide more time to focus on high value tasks. Finally, you'll learn how SRE approaches monitoring and alerting, and about the SRE approach to managing incidents. When you're finished with this course, you'll be able to evaluate SRE and see if it's a good fit for your organisation.

https://is.gd/veroto

Summary

PromQL

- Querying time-series data
- Expression syntax
- Selectors, ranges, functions, operators

Typical expressions

- `sum()` over gauges
- `rate()` over counters
- `sum() / sum()` for summaries
- `histogram_quantile()`

Query API and Grafana

- Visualizing PromQL results
- Building informative dashboards

We're Done!

So...

- Please leave a rating
- Follow `@EltonStoneman` on Twitter
- Check out `blog.sixeyed.com`
- Watch my other courses ☺