

Linux Foundation PCA

Prometheus Certified Associate (PCA)

For More Information – Visit link below:

<https://www.examsempire.com/>

Product Version

1. Up to Date products, reliable and verified.
2. Questions and Answers in PDF Format.



<https://examsempire.com/>

Visit us at: <https://www.examsempire.com/pca>

Latest Version: 6.0

Question: 1

You are monitoring a system with Prometheus. You need to create a query that will return the average number of requests per second over the last 5 minutes for a specific endpoint called '/api/users'. Which Prometheus query achieves this?

A.

```
avg(rate(http_requests_total{method="GET", path="/api/users"}[5m]))
```

B.

```
avg(http_requests_total{method="GET", path="/api/users"})
```

C.

```
avg(increase(http_requests_total{method="GET", path="/api/users"}[5m]))
```

D.

```
rate(avg(http_requests_total{method="GET", path="/api/users"}[5m]))
```

E.

```
avg(http_requests_total{method="GET", path="/api/users"} / 5 60)
```

Answer: A

Explanation:

The correct answer is A. The function calculates the per.second average rate of change over the given time range (5 minutes in this case). The 'avg()' function then averages that rate across all samples within the time range. The other options are incorrect because they do not correctly use the 'rate()' function or do not consider the averaging over the time period.

Question: 2

You are monitoring a Kubernetes cluster with Prometheus. You want to create an alert that triggers when the number of pods in a specific deployment called 'my.app' falls below 3. Which PromQL alert rule configuration correctly achieves this?

A.

```
groups(kube_deployment_status_replicas{deployment="my-app", status="available"} == 3)"
```

B.

```
kube_deployment_status_replicas{deployment="my-app", status="available"} < 3
```

C.

`kube_deployment_status_replicas{deployment="my-app", status="available"} > 3`

D.

`avg(kube_deployment_status_replicas{deployment="my-app", status="available"}) < 3`

E.

`kube_deployment_status_replicas{deployment="my-app", status="available"} == 0`

Answer: B

Explanation:

The correct answer is B . The metric provides the number of replicas for a specific deployment. The alert rule checks if the metric value for the 'my.app' deployment with the status 'available' is less than 3.

Option A uses 'groups' which isn't useful for alerts. Options C, D, and E are incorrect because they don't match the desired condition of triggering the alert when the number of pods falls below 3.

Question: 3

You are using Prometheus to monitor a website's response time. You need to create a query that shows the 95th percentile response time for the last hour. Which PromQL query achieves this?

A.

`histogram_quantile(0.95, rate(http_request_duration_seconds_bucket{method="GET"}[1h]))`

B.

`quantile(0.95, http_request_duration_seconds{method="GET"})`

C.

`quantile(0.95, rate(http_request_duration_seconds{method="GET"}[1h]))`

D.

`topk(0.95, http_request_duration_seconds{method="GET"})`

E.

`avg(http_request_duration_seconds{method="GET"}) 0.95`

Answer: C

Explanation:

The 'histogram_quantile' function is used to calculate quantiles from histograms. In this case, we are using the 'http_request_duration_seconds_bucket' histogram to calculate the 95th percentile (0.95) of the response times over the last hour. Option A is incorrect because it uses 'rate()' function for histograms, Option B is incorrect because it is not specific to the time range, and Option D is incorrect because it doesn't consider the quantile value, Option E is incorrect because it calculates a simple average multiplied by 0.95, which is not the same as a 95th percentile.

Question: 4

You are using Prometheus to monitor a system's memory usage. You need to create a query that shows the maximum memory usage in the last 24 hours. Which PromQL query achieves this?

A.

```
max(node_memory_MemTotal_bytes{instance="server1", job="node-exporter"})
```

B.

```
max(node_memory_MemFree_bytes{instance="server1", job="node-exporter"})
```

C.

```
max(node_memory_MemUsed_bytes{instance="server1", job="node-exporter"})
```

D.

```
max_over_time(node_memory_MemUsed_bytes{instance="server1", job="node-exporter"}[24h])
```

E.

```
max(node_memory_MemFree_bytes{instance="server1", job="node-exporter"} node_memory_MemTotal_bytes{instance="server1", job="node-exporter"})
```

Answer: D

Explanation:

The function calculates the maximum value over a given time range. We're using it with the metric to get the maximum memory used over the last 24 hours. Option A, B, and C are incorrect because they return the absolute maximum value across all time, ignoring the 24. hour window Option E is incorrect because it multiplies MemFree_bytes with MemTotal_bytes, which doesn't represent maximum memory usage.

Question: 5

You have a service that reports metrics with different labels. You need to create a query that returns the average value of a metric for a specific label value across all instances. Which Prometheus query achieves this?

A.

```
avg(metric_name{label_name="label_value"})
```

B.

```
avg(metric_name{instance=~"instance_regex"})
```

C.

```
avg(metric_name{label_name="label_value", instance=~"instance_regex"})
```

D.

`avg(by(instance) (metric_name{label_name="label_value"}))`

E.

`avg(metric_name{label_name!="label_value"})`

Answer: D

Explanation:

The clause aggregates the metric 'metric_name' by the 'instance' label. This ensures that you get the average across all instances while still filtering for the specific label value 'label_name="label_value"'. Options A and B are incorrect because they do not consider the aggregation across instances. Option C is incorrect because it's unnecessary to filter by instance if you are already aggregating by it. Option E is incorrect because it excludes data points where label_name is equal to 'label_value'.

Question: 6

You are monitoring a system's CPU usage. You need to create a query that shows the average CPU usage for all instances over the last hour. Which PromQL query achieves this?

A.

`avg(node_cpu_seconds_total{mode="user"})`

B.

`avg(rate(node_cpu_seconds_total{mode="user"}[1h]))`

C.

`avg(increase(node_cpu_seconds_total{mode="user"}[1h]))`

D.

`avg(node_cpu_seconds_total{mode="user"}[1h])`

E.

`avg(by(instance) (node_cpu_seconds_total{mode="user"}[1h]))`

Answer: E

Explanation:

This query uses 'by(instance)' to group the metric by the 'instance' label. Then it calculates the average over the last hour for each instance. This gives you the average CPU usage across all instances. Option A is incorrect because it does not calculate the average over the last hour. Option B is incorrect because it calculates the average rate of change over the last hour, which is not what we want. Option C is incorrect because it calculates the total increase in CPU usage over the last hour. Option D is incorrect because it does not group by instances, so the average is taken across all instances.

Question: 7

You have a service that emits metrics with different labels. You need to create a query that returns the sum of a metric for a specific label value across all instances. Which PromQL query achieves this?

A.

```
sum(metric_name{label_name="label_value"})
```

B.

```
sum(by(instance) (metric_name{label_name="label_value"}))
```

C.

```
sum(metric_name{instance=~"instance_regex"})
```

D.

```
sum(by(instance) (metric_name{label_name="label_value", instance=~"instance_regex"}))
```

E.

```
sum(by(label_name) (metric_name{label_name="label_value"}))
```

Answer: A

Explanation:

The 'sum()' function directly calculates the sum of the metric 'metric_name' for all instances that have the label 'label_name' set to the value . While option B also works, its overkill as you are already filtering by the label. Option C and D are incorrect because they use instance_regex, which is unnecessary as we are interested in a specific label value. Option E is incorrect because it groups by label_name which will aggregate all metrics with the same label, not just the specific value we are interested in.

Question: 8

You need to create a PromQL query that returns the number of unique instances that have a specific metric value above a threshold for the last 5 minutes. Which query achieves this?

A.

```
count(metric_name{value>threshold}[5m])
```

B.

```
count(by(instance) (metric_name{value>threshold}[5m]))
```

C.

```
count(distinct(instance) (metric_name{value>threshold}[5m]))
```

D.

```
count(metric_name{value>threshold, instance=~"instance_regex"}[5m])
```

E.

```
count(by(instance) (metric_name{value>threshold}[5m]))
```

Answer: C

Explanation:

The function is used to count the unique instances within the specified time range. Option A is incorrect because it does not consider unique instances. Option B and D are incorrect because they do not use the 'distinct' function to identify unique instances. Option E is incorrect because it counts unique instances across all data points, ignoring the metric value threshold.

Question: 9

You are monitoring a system with Prometheus and want to create a query that returns the average value of a metric, but only for instances that have a specific label value set. Which PromQL query achieves this?

A.

```
avg(metric_name{label_name="label_value"})
```

B.

```
avg(metric_name{label_name!="label_value"})
```

C.

```
avg(by(instance) (metric_name{label_name="label_value"}))
```

D.

```
avg(metric_name) (label_name == "label_value")
```

E.

```
avg(metric_name) + (label_name == "label_value")
```

Answer: A

Explanation:

The 'avg()' function calculates the average of the 'metric_name' for all instances that have the label set to the value 'label_value'. The other options are incorrect. Option B would return the average for instances not having the specific label value. Option C is incorrect as it groups by instance, which would lead to separate averages for each instance, not a single average across all instances with the specified label. Option D and E are incorrect as they attempt to multiply or add a boolean condition to the average, which is not a valid operation.

Question: 10

You are monitoring a web application using Prometheus. You want to track the number of successful and failed login attempts over time. Which of the following Prometheus query approaches would be most effective to achieve this?

- A. Use a counter metric for both successful and failed login attempts, and increment the counter for each event.
- B. Use a gauge metric for both successful and failed login attempts, and set the gauge value to 1 for each event.
- C. Use a histogram metric to track the distribution of login attempts based on their success or failure.
- D. Use a summary metric to track the average and quantiles of successful and failed login attempts.
- E. Use a separate metric for successful and failed login attempts, with a counter for each metric incremented accordingly.

Answer: E

Explanation:

Option E is the most effective approach. Using separate counter metrics for successful and failed login attempts allows you to track both types of events independently and provides accurate counts over time. Counters are suitable for counting events and can be reset, making them ideal for tracking the total number of successful and failed login attempts.

Question: 11

You are setting up a Prometheus exporter for a custom application. The application emits logs in JSON format, and you want to extract specific metrics from the logs using a regular expression. How would you configure the Prometheus exporter to achieve this?

- A. Use the '`--text.file.collector`' flag with the Prometheus exporter, specifying the log file and a regular expression to extract metrics.
- B. Use the '`--json.file.collector`' flag with the Prometheus exporter, specifying the log file and a regular expression to extract metrics.
- C. Use the '`--log.collector`' flag with the Prometheus exporter, specifying the log file and a regular expression to extract metrics.
- D. Use a dedicated log processing tool like Fluentd or Logstash to extract metrics from the JSON logs and send them to Prometheus.
- E. Configure Prometheus to directly read the JSON logs using the '`--remote.write`' flag and define custom metrics based on the log data.

Answer: D

Explanation:

Option D is the most appropriate solution. Using a dedicated log processing tool like Fluentd or Logstash allows you to efficiently extract metrics from JSON logs and send them to Prometheus. These tools offer

flexible log parsing and filtering capabilities, enabling you to extract specific metrics based on regular expressions or other criteria. While options A, B, and C might seem relevant, they are not directly supported by Prometheus exporters for extracting metrics from JSON logs.

Question: 12

You are monitoring a service that generates a large number of logs. You want to use Prometheus to track the number of log messages per second. How can you achieve this efficiently using Prometheus metrics?

- A. Use a counter metric and increment it for each log message.
- B. Use a gauge metric and set it to the number of log messages per second.
- C. Use a histogram metric to track the distribution of log message counts per second.
- D. Use a summary metric to track the average and quantiles of log messages per second.
- E. Use a rate function in a Prometheus query to calculate the number of log messages per second based on a counter metric.

Answer: E

Explanation:

Option E is the most efficient approach. Using a counter metric to count log messages and applying the 'rate' function in a Prometheus query allows you to calculate the number of log messages per second. The 'rate' function calculates the rate of change of a counter over a specified time window, providing a clear understanding of the log message rate.

Thank You for Trying Our Product

Special 16 USD Discount Coupon: NSZUBG3X

Email: support@examsempire.com

**Check our Customer Testimonials and ratings
available on every product page.**

Visit our website.

<https://examsempire.com/>