Accelerate Your Certification with Linux Foundation PCA Practice Questions

Last chance to prepare smart! Get your hands on free Linux Foundation Prometheus Certified Associate exam PDF questions. Study real PCA dumps with verified answers and fast-track your certification success with PrepBolt PCA exam pdf questions and answers.

Thank you for Downloading PCA exam PDF Demo

https://prepbolt.com/PCA.html

Question 1

Question Type: MultipleChoice

Which PromQL expression computes how many requests in total are currently in-flight for the following time series data?

apiserver_current_inflight_requests{instance="1"} 5

apiserver_current_inflight_requests{instance="2"} 7

Options:

A- min(apiserver_current_inflight_requests)
B- sum(apiserver_current_inflight_requests)
C- max(apiserver_current_inflight_requests)
D- sum_over_time(apiserver_current_inflight_requests[10m])

Answer:

B

Explanation:

In Prometheus, when you have multiple time series that represent the same type of measurement across different instances, the sum() aggregation operator is used to compute their total value.

Here, each instance (1 and 2) exposes the metric apiserver_current_inflight_requests, indicating the number of active API requests currently being processed.

To find the total number of in-flight requests across all instances, the correct expression is:

sum(apiserver_current_inflight_requests)

This returns 5 + 7 = 12.

min() would return the lowest value (5).

max() would return the highest value (7).

sum_over_time() calculates the cumulative sum over a range vector, not the current value, so it's incorrect here.

Verified from Prometheus documentation -- Aggregation Operators and Summing Across Dimensions sections.

Question 2

Question Type: MultipleChoice

Which of the following metrics is unsuitable for a Prometheus setup?

Options:

A- prometheus_engine_query_log_enabled
B- promhttp_metric_handler_requests_total{code='500'}
C- http_response_total{handler='static/*filepath'}
D- user_last_login_timestamp_seconds{email='john.doe@example.com'}

Answer:

D

Explanation:

The metric user_last_login_timestamp_seconds{email='john.doe@example.com'} is unsuitable for Prometheus because it includes a high-cardinality label (email). Each unique email address would generate a separate time series, potentially numbering in the millions, which severely impacts Prometheus performance and memory usage.

Prometheus is optimized for low- to medium-cardinality metrics that represent system-wide behavior rather than per-user data. High-cardinality metrics cause data explosion, complicating queries and overwhelming the storage engine.

By contrast, the other metrics---prometheus_engine_query_log_enabled, promhttp_metric_handler_requests_total{code='500'}, and http_response_total{handler='static/*filepath'}---adhere to Prometheus best practices. They represent operational or service-level metrics with limited, manageable label value sets.

Extracted and verified from Prometheus documentation -- Metric and Label Naming Best Practices, Cardinality Management, and Anti-Patterns for Metric Design sections.

Question 3

Question Type: MultipleChoice

Which of the following PromQL queries is invalid?

Options:

A- max by (instance) up
B- max on (instance) (up)
C- max without (instance) up
D- max without (instance, job) up

Answer:

B

Explanation:

The max operator in PromQL is an aggregation operator, not a binary vector matching operator. Therefore, the valid syntax for aggregation uses by() or without(), not on().

max by (instance) up Valid; aggregates maximum values per instance.

max without (instance) up and max without (instance, job) up Valid; aggregates over all labels except those listed.

max on (instance) (up) Invalid; the keyword on() is only valid in binary operations (e.g., +, -, and, or, unless), where two vectors are being matched on specific labels.

Hence, max on (instance) (up) is a syntax error in PromQL because on() cannot be used directly with aggregation operators.

Verified from Prometheus documentation -- Aggregation Operators, Vector Matching -- on()/ignoring(), and PromQL Language Syntax Reference sections.

Question 4

Question Type: MultipleChoice

How do you calculate the average request duration during the last 5 minutes from a histogram or summary called http_request_duration_seconds?

Options:

A- rate(http_request_duration_seconds_sum[5m]) / rate(http_request_duration_seconds_count[5m])
B- rate(http_request_duration_seconds_total[5m]) / rate(http_request_duration_second$_count[5m])
C- rate(http_request_duration_seconds_total[5m]) / rate(http_request_duration_seconds_average[5m])

D- rate(http_request_duration_seconds_sum[5m]) / rate(http_request_duration_seconds_average[5m])

Answer:

A

Explanation:

In Prometheus, histograms and summaries expose metrics with _sum and _count suffixes to represent total accumulated values and sample counts, respectively. To compute the average request duration over a given time window (for example, 5 minutes), you divide the rate of increase of _sum by the rate of increase of _count:

\text{Average duration} = \frac{\text{rate(http_request_duration_seconds_sum[5m])}}{\text{rate(http_request_duration_seconds_count[5m])}}

Here,

http_request_duration_seconds_sum represents the total accumulated request time, and

http_request_duration_seconds_count represents the number of requests observed.

By dividing these rates, you obtain the average request duration per request over the specified time range.

Extracted and verified from Prometheus documentation -- Querying Histograms and Summaries, PromQL Rate Function, and Metric Naming Conventions sections.

Question 5

Question Type: MultipleChoice

What is an example of a single-target exporter?

Options:

A- Redis Exporter
B- SNMP Exporter
C- Node Exporter
D- Blackbox Exporter

Answer:

A

Explanation:

A single-target exporter in Prometheus is designed to expose metrics for a specific service instance rather than multiple dynamic endpoints. The Redis Exporter is a prime example --- it connects to one Redis server instance and exports its metrics (like memory usage, keyspace hits, or command statistics) to Prometheus.

By contrast, exporters like the SNMP Exporter and Blackbox Exporter can probe multiple targets dynamically, making them multi-target exporters. The Node Exporter, while often deployed per host, is considered a host-level exporter, not a true single-target one in configuration behavior.

The Redis Exporter is instrumented specifically for a single Redis endpoint per configuration, aligning it with Prometheus's single-target exporter definition. This design simplifies monitoring and avoids dynamic reconfiguration.

Verified from Prometheus documentation and official exporter guidelines -- Writing Exporters, Exporter Types, and Redis Exporter Overview sections.

Question 6

Question Type: MultipleChoice

Which function would you use to calculate the 95th percentile latency from histogram data?

Options:

A- quantile_over_time(0.95, http_request_duration_seconds[5m])
B- histogram_quantile(0.95, sum(rate(http_request_duration_seconds_bucket[5m])) by (le))
C- percentile(http_request_duration_seconds, 0.95)
D- topk(0.95, http_request_duration_seconds)

Answer:

B

Explanation:

To calculate a percentile (e.g., 95th percentile) from histogram data in Prometheus, the correct function is histogram_quantile(). It estimates quantiles based on cumulative bucket counts.

Example:

histogram_quantile(0.95, sum(rate(http_request_duration_seconds_bucket[5m])) by (le))

This computes the 95th percentile request duration across all observed instances over the last 5 minutes.

Thank You for trying PCA PDF Demo

To try our PCA practice exam software visit link below

https://prepbolt.com/PCA.html

Start Your PCA Preparation

Use Coupon “SAVE50” for extra 50% discount on the purchase of Practice Test Software. Test your PCA preparation with actual exam questions.