2026 PCA: Prometheus Certified Associate Exam Marvelous Reliable Exam Camp

Are you on the way to pass the PCA exam? Our PCA exam questions will be the best choice for you. And if you still feel uncertain about the content, wondering whether it is the exact PCA exam material that you want, you can free download the demo to check it out. You will be quite surprised by the convenience to have an overview just by clicking into the link, and you can experience all kinds of PCA versions.

Linux Foundation PCA Exam Syllabus Topics:

Topic	Details
Topic 1	• PromQL: This section of the exam measures the skills of Monitoring Specialists and focuses on Prometheus Query Language (PromQL) concepts. It covers data selection, calculating rates and derivatives, and performing aggregations across time and dimensions. Candidates also study the use of binary operators, histograms, and timestamp metrics to analyze monitoring data effectively, ensuring accurate interpretation of system performance and trends.
Topic 2	• Alerting and Dashboarding: This section of the exam assesses the competencies of Cloud Operations Engineers and focuses on monitoring visualization and alert management. It covers dashboarding basics, alerting rules configuration, and the use of Alertmanager to handle notifications. Candidates also learn the core principles of when, what, and why to trigger alerts, ensuring they can create reliable monitoring dashboards and proactive alerting systems to maintain system stability.
Topic 3	• Prometheus Fundamentals: This domain evaluates the knowledge of DevOps Engineers and emphasizes the core architecture and components of Prometheus. It includes topics such as configuration and scraping techniques, limitations of the Prometheus system, data models and labels, and the exposition format used for data collection. The section ensures a solid grasp of how Prometheus functions as a monitoring and alerting toolkit within distributed environments.
Topic 4	• Instrumentation and Exporters: This domain evaluates the abilities of Software Engineers and addresses the methods for integrating Prometheus into applications. It includes the use of client libraries, the process of instrumenting code, and the proper structuring and naming of metrics. The section also introduces exporters that allow Prometheus to collect metrics from various systems, ensuring efficient and standardized monitoring implementation.

Topic 5	• Observability Concepts: This section of the exam measures the skills of Site Reliability Engineers and covers the essential principles of observability used in modern systems. It focuses on understanding metrics, logs, and tracing mechanisms such as spans, as well as the difference between push and pull data collection methods. Candidates also learn about service discovery processes and the fundamentals of defining and maintaining SLOs, SLAs, and SLIs to monitor performance and reliability.

100% Pass Quiz Linux Foundation - PCA –High Hit-Rate Reliable Exam Camp

Our website Prep4cram provide the PCA test guide to clients and help they pass the test PCA certification which is highly authorized and valuable. Our company is a famous company which bears the world-wide influences and our PCA test prep is recognized as the most representative and advanced study materials among the same kinds of products. Whether the qualities and functions or the service of our PCA Exam Questions, are leading and we boost the most professional expert team domestically.

Linux Foundation Prometheus Certified Associate Exam Sample Questions (Q54-Q59):

NEW QUESTION # 54
What's "wrong" with the myapp_filG_uploads_total{userid=„5123",status="failed"} metric?

- A. The userid should not be exposed as a label.
- B. The metric name should consist of dashes instead of underscores.
- C. The _total suffix should be omitted.
- D. The status should not be exposed as a label.

Answer: A

Explanation:
In Prometheus best practices, high-cardinality labels-especially those containing unique or user-specific identifiers-should be avoided. The metric myapp_filG_uploads_total{userid="5123",status="failed"} exposes the userid as a label, which is problematic. Each distinct value of a label generates a new time series in Prometheus. If there are thousands or millions of unique users, this would exponentially increase the number of time series, leading to cardinality explosion, degraded performance, and high memory usage. The _total suffix is actually correct and required for counters, as per the Prometheus naming convention. The use of underscores in metric names is also correct, as Prometheus does not support dashes in metric identifiers. The status label, however, is perfectly valid because it typically has a low number of possible values (e.g., "success", "failed").
Reference:
Verified from Prometheus official documentation sections Instrumentation - Metric and Label Naming Best Practices and Writing Exporters.

NEW QUESTION # 55
Which of the following PromQL queries is invalid?

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

Answer: D

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

NEW QUESTION # 56
Which of the following is an invalid @ modifier expression?

- A. sum(http_requests_total{method="GET"} @ 1609746000)
- B. go_goroutines @ end()
- C. go_goroutines @ start()
- D. sum(http_requests_total{method="GET"}) @ 1609746000

Answer: A

Explanation:
The @ modifier in PromQL allows querying data as it existed at a specific point in time rather than the evaluation time. It can be applied after a selector or an entire expression, but the syntax rules are strict.
 go_goroutines @ start() → Valid; queries value at the start of the evaluation range.
 sum(http_requests_total{method="GET"}) @ 1609746000 → Valid; applies the modifier after the full expression.
 go_goroutines @ end() → Valid; queries value at the end of the evaluation range.
 sum(http_requests_total{method="GET"} @ 1609746000) → Invalid, because the @ modifier cannot appear inside the selector braces; it must appear after the selector or aggregation expression.
This invalid placement violates PromQL's syntax grammar for subquery and modifier ordering.
Reference:
Verified from Prometheus documentation - PromQL @ Modifier Syntax, Evaluation Modifiers, and PromQL Expression Grammar sections.

NEW QUESTION # 57
How can you use Prometheus Node Exporter?

- A. You can use it to instrument applications with metrics.
- B. You can use it to collect metrics for hardware and OS metrics.
- C. You can use it to probe endpoints over HTTP, HTTPS.
- D. You can use it to collect resource metrics from the application HTTP server.

Answer: B

Explanation:
The Prometheus Node Exporter is a core system-level exporter that exposes hardware and operating system metrics from *nix-based hosts. It collects metrics such as CPU usage, memory, disk I/O, filesystem space, network statistics, and load averages.
It runs as a lightweight daemon on each host and exposes metrics via an HTTP endpoint (default: :9100/metrics), which Prometheus scrapes periodically.
Key clarification:
It does not instrument applications (A).
It does not collect metrics directly from application HTTP endpoints (B).
It is unrelated to HTTP probing tasks - those are handled by the Blackbox Exporter (D).
Thus, the correct use of the Node Exporter is to collect and expose hardware and OS-level metrics for Prometheus monitoring.
Reference:
Extracted and verified from Prometheus documentation - Node Exporter Overview, Host-Level Monitoring, and Exporter Usage Best Practices sections.

NEW QUESTION # 58
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

- A. max(apiserver_current_inflight_requests)
- B. sum(apiserver_current_inflight_requests)
- C. min(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.
Reference:
Verified from Prometheus documentation - Aggregation Operators and Summing Across Dimensions sections.

NEW QUESTION # 59
.....

myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, www.stes.tyc.edu.tw, Disposable vapes