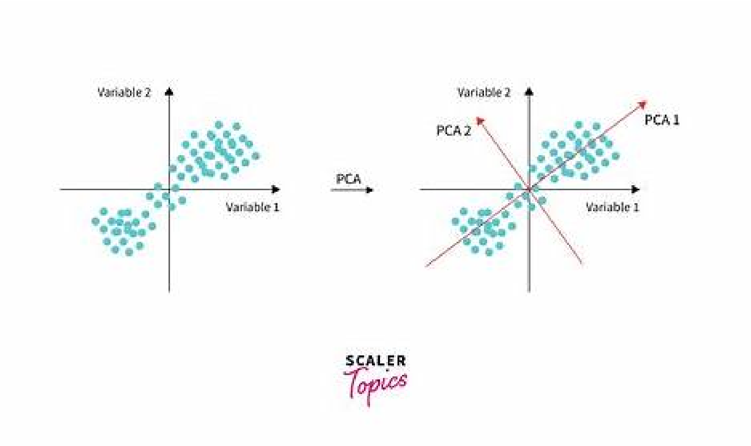
PCA New Soft Simulations & PCA Answers Free



BTW, DOWNLOAD part of Dumpkiller PCA dumps from Cloud Storage: https://drive.google.com/open?id=1FF1V3NzRC9CUcvRtpSqS4fmd3HZqSPqE

Our Prometheus Certified Associate Exam (PCA) prep material also includes web-based and desktop Prometheus Certified Associate Exam (PCA) practice tests for you to put your skills to the test. Our Prometheus Certified Associate Exam (PCA) practice exams simulate the real Prepare for your Prometheus Certified Associate Exam (PCA) exam environment, so you can experience the pressure and environment of the actual test before the day arrives. You'll receive detailed feedback on your performance, so you know what areas to focus on and improve.

Linux Foundation PCA Exam Syllabus Topics:

Topic	Details
Topic 1	• 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.
Topic 2	• 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 3	• 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 4	• 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 5	• 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.

>> PCA New Soft Simulations <<

PCA Answers Free & Exam PCA Guide

Now, I am glad to introduce a secret weapon for all of the candidates to pass the exam as well as get the related certification without any more ado-- our PCA study braindumps. You can only get the most useful and efficient PCA Guide materials with the most affordable price from our company, since we aim to help as many people as possible rather than earning as much money as possible. You will be much awarded with our PCA learning engine.

Linux Foundation Prometheus Certified Associate Exam Sample Questions (Q19-Q24):

NEW QUESTION # 19
Which Alertmanager feature prevents duplicate notifications from being sent?

- A. Deduplication
- B. Inhibition
- C. Silencing
- D. Grouping

Answer: A

Explanation:
Deduplication in Alertmanager ensures that identical alerts from multiple Prometheus servers or rule evaluations do not trigger duplicate notifications.
Alertmanager compares alerts based on their labels and fingerprints; if an alert with identical labels already exists, it merges or refreshes the existing one instead of creating a new notification.
This mechanism is essential in high-availability setups where multiple Prometheus instances monitor the same targets.

NEW QUESTION # 20
Which of the following signals belongs to symptom-based alerting?

- A. API latency
- B. Database availability
- C. CPU usage
- D. Disk space

Answer: A

Explanation:
Symptom-based alerting focuses on detecting user-visible or service-impacting issues rather than internal resource states. Metrics like API latency, error rates, and availability directly indicate degraded user experience and are therefore the preferred triggers for alerts.
In contrast, resource-based alerts (like CPU usage or disk space) often represent underlying causes, not symptoms. Alerting on them can produce noise and distract from actual service health problems.
For example, high API latency (http_request_duration_seconds) clearly reflects that users are experiencing delays, which is actionable and business-relevant.
This concept aligns with the RED (Rate, Errors, Duration) and USE (Utilization, Saturation, Errors) monitoring models promoted in Prometheus and SRE best practices.
Reference:
Verified from Prometheus documentation - Alerting Best Practices, Symptom vs. Cause Alerting, and RED/USE Monitoring Principles.

NEW QUESTION # 21
What is a difference between a counter and a gauge?

- A. Counters change value on each scrape and gauges remain static.
- B. Counters are only incremented, while gauges can go up and down.
- C. Counters and gauges are different names for the same thing.
- D. Counters have no labels while gauges can have many labels.

Answer: B

Explanation:
The key difference between a counter and a gauge in Prometheus lies in how their values change over time. A counter is a cumulative metric that only increases-it resets to zero only when the process restarts. Counters are typically used for metrics like total requests served, bytes processed, or errors encountered. You can derive rates of change from counters using functions like rate() or increase() in PromQL.
A gauge, on the other hand, represents a metric that can go up and down. It measures values that fluctuate, such as CPU usage, memory consumption, temperature, or active session counts. Gauges provide a snapshot of current state rather than a cumulative total.
This distinction ensures proper interpretation of time-series trends and prevents misrepresentation of one-time or fluctuating values as cumulative metrics.
Reference:
Extracted and verified from Prometheus official documentation - Metric Types section explaining Counters and Gauges definitions and usage examples.

NEW QUESTION # 22
Given the metric prometheus_tsdb_lowest_timestamp_seconds, how do you know in which month the lowest timestamp of your Prometheus TSDB belongs?

- A. month(prometheus_tsdb_lowest_timestamp_seconds)
- B. format_date(prometheus_tsdb_lowest_timestamp_seconds,"%M")
- C. (time() - prometheus_tsdb_lowest_timestamp_seconds) / 86400
- D. prometheus_tsdb_lowest_timestamp_seconds % month

Answer: C

Explanation:
The metric prometheus_tsdb_lowest_timestamp_seconds provides the oldest stored sample timestamp in Prometheus's local TSDB (in Unix epoch seconds). To determine the age or approximate date of this timestamp, you compare it with the current time (using time() in PromQL).
The expression:
(time() - prometheus_tsdb_lowest_timestamp_seconds) / 86400
converts the difference between the current time and the oldest timestamp from seconds into days (1 day = 86,400 seconds). This gives the number of days since the earliest sample was stored, allowing you to infer the time range and approximate month manually.
The other options are invalid because PromQL does not support direct date formatting (format_date) or month() extraction functions.
Reference:
Extracted and verified from Prometheus documentation - TSDB Internal Metrics, Time Functions in PromQL, and Using time() for Relative Calculations.

NEW QUESTION # 23
The following is a list of metrics exposed by an application:
http_requests_total{code="500"} 10
http_requests_total{code="200"} 20
http_requests_total{code="400"} 30
http_requests_total{verb="POST"} 30
http_requests_total{verb="GET"} 30
What is the issue with the metric family?

- A. The value represents two different things across the dimensions: code and verb.
- B. Metric names are missing a prefix to indicate which application is exposing the query.

Answer: A

Explanation:
Prometheus requires that a single metric name represents one well-defined thing, and all time series in that metric share the same set of label keys so the value's meaning is consistent across dimensions. The official guidance states that metrics should not "mix different dimensions under the same name," and that a metric name should have a consistent label schema; otherwise, "the same metric name would represent different things," making queries ambiguous and aggregations error-prone. In the example, http_requests_total{code="..."} expresses per-status-code request counts, while http_requests_total{verb="..."} expresses per-HTTP-method request counts. Because some series have only code and others only verb, the value changes its meaning across label sets, violating the consistency principle for a metric family. The correct approach is to expose one metric with both labels present on every series, e.g., http_requests_total{code="200", method="GET"}, ensuring every time series has the same label keys and the value always means "count of requests," sliced by the same dimensions. A missing application prefix is optional and not the core issue here.

NEW QUESTION # 24
......