DUMPSARENA

Prometheus Certified Associate Exam

Linux Foundation PCA

Version Demo

Total Demo Questions: 6

Total Premium Questions: 63

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Observability Concepts	6
Topic 2, Prometheus Fundamentals	25
Topic 3, PromQL	19
Topic 4, Instrumentation and Exporters	13
Total	63

QUESTION NO: 1

What are Inhibition rules?

A. Inhibition rules mute a set of alerts when another matching alert is firing.

B. Inhibition rules repeat a set of alerts when another matching alert is firing.

C. Inhibition rules inject a new set of alerts when a matching alert is firing.

D. Inhibition rules inspect alerts when a matching set of alerts is firing.

ANSWER: A

Explanation:

Inhibition rules mute notifications for target alerts when a source alert is firing and the alerts match on configured label values. This Alertmanager feature is intended to reduce redundant alert noise during a broader failure. For example, when a high-level alert such as a cluster, service, or datacenter outage is active, related lower-level alerts can be inhibited so responders receive the signal describing the likely root cause rather than many downstream symptom alerts.

An Alertmanager inhibition rule is configured with source matchers, target matchers, and an `equal` list. The source matchers identify the firing alert that causes inhibition; the target matchers identify alerts eligible to be muted; and labels named in `equal` must have identical values on both alerts. Inhibition affects alert notifications handled by Alertmanager, rather than preventing Prometheus from evaluating the alerting rules themselves. Alertmanager also treats an alert as inhibited if it would otherwise match both the source and target side of the same rule, avoiding self-inhibition.

See the official Alertmanager inhibition documentation and the inhibit_rule configuration reference.

QUESTION NO: 2

You'd like to monitor a short-lived batch job. What Prometheus component would you use?

A. PullProxy

B. PushGateway

C. PushProxy

D. PullGateway

ANSWER: B

Explanation:

The correct component is the **Pushgateway**. Prometheus generally collects metrics by scraping HTTP endpoints at configured intervals. A short-lived batch job can start and finish entirely between two scrapes, meaning its metrics endpoint may no longer exist when Prometheus attempts collection. The Pushgateway provides a stable, scrapeable endpoint to which a batch job can publish its metrics before it exits. Prometheus then retrieves those metrics from the Pushgateway during its normal scrape cycle.

This pattern is intended for service-level metrics produced by ephemeral jobs, such as the timestamp of the last successful backup, the duration of a completed data-processing task, or whether a scheduled job succeeded. Jobs should push metrics using an appropriate grouping key, and their metric lifecycle should be managed deliberately, because pushed metrics remain in the Pushgateway until they are replaced or explicitly deleted. The Pushgateway is therefore an intermediary for preserving results from jobs that Prometheus cannot reliably scrape directly, rather than a replacement for normal pull-based instrumentation of long-running services. See the official Prometheus guidance on pushing metrics and the Pushgateway project documentation.

QUESTION NO: 3

What is a rule group?

A. It is a set of rules that are executed sequentially.

B. It is the set (the group) of all the rules in a file.

C. It is a set of rules, split into groups by type.

D. It is a set of rules that are grouped by labels.

ANSWER: A

Explanation:

It is a set of rules that are executed sequentially. In Prometheus, recording rules and alerting rules are organized into named rule groups in rule-definition files. A group can define its own evaluation interval, and Prometheus evaluates the rules within that group in the order in which they are written. This sequential evaluation is important when a later rule uses the output of a recording rule defined earlier in the same group: the earlier rule's newly calculated result is available for the subsequent rule evaluation cycle within that group.

Rule groups are configured under the `groups` section of a YAML rule file. Each group normally includes a `name`, optionally an `interval`, and a `rules` list. The interval controls how often Prometheus evaluates that particular group; if omitted, Prometheus uses its global evaluation interval. Grouping rules therefore provides a clear scheduling and ordering boundary for related recording and alerting logic. Prometheus documentation explicitly states that rules in a group are evaluated sequentially at regular intervals. See the Prometheus recording rules documentation and the Prometheus alerting rules documentation.

QUESTION NO: 4

A scrape job must retain a `scrape_interval` of 15 seconds because short failures need to be observed promptly. Its configured `scrape_timeout` is 20 seconds. Which change makes the configuration valid while retaining the required collection cadence?

A. Set `scrape_timeout: 10s`.

B. Set `evaluation_interval: 20s`.

C. Set `scrape_timeout: 30s`.

D. Set `scrape_interval: 20s`.

ANSWER: A

Explanation:

Set `scrape_timeout` to a value no greater than 15 seconds. Prometheus requires a scrape timeout not to exceed the scrape interval for a job. A timeout such as 10 seconds allows each scrape attempt to finish or fail before the next scheduled 15-second scrape.

Increasing the interval to 20 seconds would also satisfy the configuration constraint, but violates the requirement to retain 15-second collection. Rule evaluation settings do not affect scrape scheduling, and a timeout longer than the interval remains invalid.

QUESTION NO: 5

What is the difference between client libraries and exporters?

A. Exporters are written in Go. Client libraries are written in many languages.

B. Exporters expose metrics for scraping. Client libraries push metrics via Remote Write.

C. Exporters run as separate processes to expose metrics from monitored services, while client libraries instrument applications directly.

D. Exporters and client libraries mean the same thing.

ANSWER: C

Explanation:

Exporters run as separate processes to expose metrics from monitored systems or services, while client libraries are integrated into an application's own code to instrument it directly. This distinction reflects Prometheus's pull-based model: both approaches ultimately expose an HTTP endpoint that Prometheus can scrape, but they are used at different integration points. A client library lets an application developer define counters, gauges, histograms, and other metrics as part of the application's implementation. It also handles the details of representing those metrics in the Prometheus exposition format. Exporters are appropriate when direct instrumentation is unavailable, impractical, or undesirable, such as for operating-system statistics, databases, hardware devices, or third-party software. An exporter collects or translates information from the target system and presents it as Prometheus metrics. Exporters commonly use a Prometheus client library internally to create and serve those metrics, but their essential role is to provide a scrapeable Prometheus interface for an external target rather than to add instrumentation to that target's source code. Prometheus documents client libraries as the usual method for application instrumentation and exporters as bridges for metrics from systems that do not natively expose Prometheus metrics. See the Prometheus client library documentation and the Prometheus exporter documentation.

QUESTION NO: 6

What is metamonitoring?

A. Metamonitoring is a monitoring that covers 100% of a service.

B. Metamonitoring is the monitoring of non-IT systems.

C. Metamonitoring is the monitoring of the monitoring infrastructure.

D. Metamonitoring is monitoring social networks for end user complaints about quality of service.

ANSWER: C

Explanation:

Metamonitoring is the monitoring of the monitoring infrastructure. In a Prometheus environment, this means observing the health and effectiveness of the components responsible for collecting, storing, evaluating, and delivering monitoring data and alerts. Typical targets include Prometheus servers, exporters, Alertmanager, remote-write endpoints, service discovery, and the rules and scrape operations that connect them. Useful signals include whether targets are being scraped successfully, scrape duration and sample volume, rule-evaluation failures or latency, time-series ingestion behavior, storage capacity, Alertmanager availability, and notification delivery health.

This practice is essential because an outage or degradation in the monitoring stack can create a dangerous blind spot: services may be failing while the team receives no trustworthy telemetry or alerts. Prometheus exposes internal metrics about its own operation, commonly scraped from its own metrics endpoint, so those metrics can be queried, recorded, and alerted on. For greater resilience, organizations commonly ensure that monitoring of critical monitoring-path components is itself visible from an independent or highly available monitoring path. Prometheus guidance discusses monitoring and alerting practices, including maintaining actionable alerting behavior, at Prometheus alerting best practices. The broader reliability rationale for monitoring the monitoring system is also covered in the Google SRE Book's monitoring chapter.