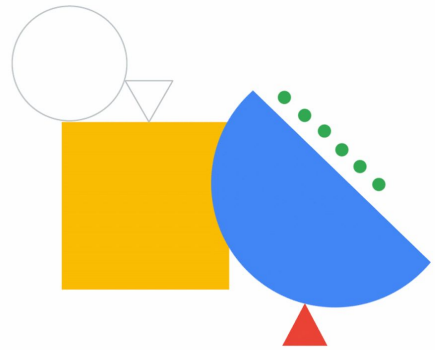


Alerting Policies



Alerting gives timely awareness to problems in your cloud applications so you can resolve the problems quickly.

Objectives

- 01 Explain why SLI, SLO, and SLA are important.
- 02 Explain alerting policies and alerting strategies.
- 03 Explain error budgets.
- 04 Identify types of alerts and common uses for each.
- 05 Use Cloud Monitoring to manage services.



In this module, you will learn how to:

- Explain why SLI, SLO and SLA are important.
- Explain alerting policies and alerting strategies.
- Explain error budgets.
- Identify types of alerts and common uses for each.
- Use Cloud Monitoring to manage services.

In this section, you explore



- ✓ SLI, SLO, and SLA
- ✓ Developing an alerting strategy
- ✓ Creating alerts
- ✓ Service Monitoring

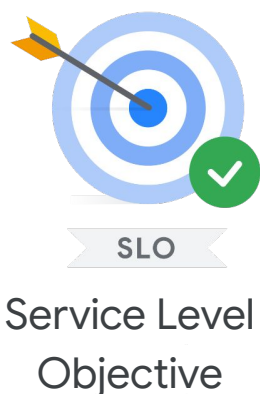
Before we cover alerting strategy, it is important to understand what SLI, SLO and SLA are. These terms are frequently used in this course.



Service Level Indicator

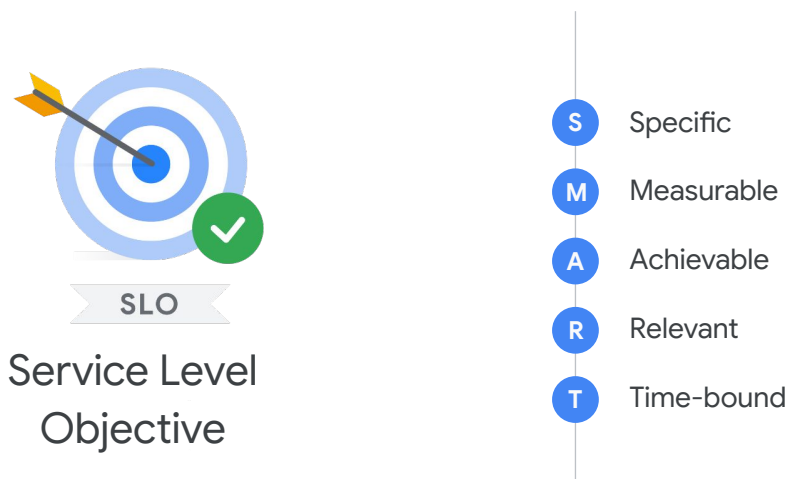
Carefully selected monitoring metrics that measure one aspect of a service's reliability

Service level indicators, or SLIs, are carefully selected monitoring metrics that measure one aspect of a service's reliability. Ideally, SLIs should have a close linear relationship with your users' experience of that reliability, and we recommend expressing them as the ratio of two numbers: the number of good events divided by the count of all valid events.



Combines a service level indicator with a target reliability and will generally be somewhere just short of 100%, for example, 99.9% ("three nines")

A **Service level objective, or SLO**, combines a service level indicator with a target reliability. If you express your SLIs as is commonly recommended, your SLOs will generally be somewhere just short of 100%, for example, 99.9%, or "three nines."



You can't measure everything, so when possible, you should choose SLOs that are **S.M.A.R.T.**

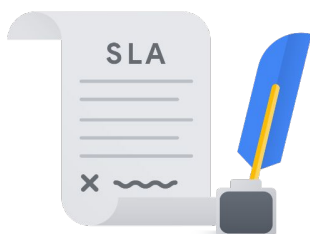
SLOs should be **specific**. "Hey everyone, is the site fast enough for you?" is not specific; it's subjective. "The 95th percentile of results are returned in under 100ms." That's specific.

They need to be based on indicators that are **measurable**. A lot of monitoring is numbers, grouped over time, with math applied. An SLI must be a number or a delta, something we can measure and place in a mathematical equation.

SLO goals should be **achievable**. "100% Availability" might sound good, but it's not possible to obtain, let alone maintain, over an extended window of time.

SLOs should be **relevant**. Does it matter to the user? Will it help achieve application-related goals? If not, then it's a poor metric.

And SLOs should be **time-bound**. You want a service to be 99% available? That's fine. Is that per year? Per month? Per day? Does the calculation look at specific windows of set time, from Sunday to Sunday for example, or is it a rolling period of the last seven days? If we don't know the answers to those types of questions, it can't be measured accurately.



Service Level Agreement

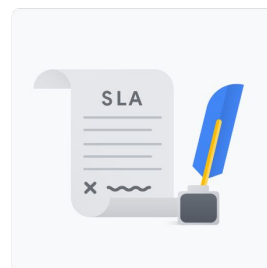
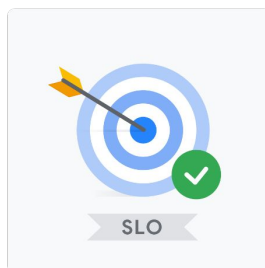
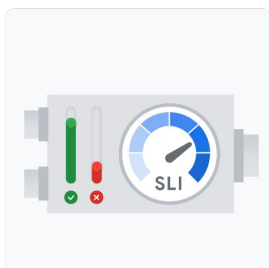
Commitments made to your customers that your systems and applications will have only a certain amount of down time

And then there are **Service Level Agreements, or SLAs**, which are commitments made to your customers that your systems and applications will have only a certain amount of “down time.”

An SLA describes the minimum levels of service that you promise to provide to your customers and what happens when you break that promise.

If your service has paying customers, an SLA may include some way of compensating them with refunds or credits when that service has an outage that is longer than this agreement allows.

To give you the opportunity to detect problems and take remedial action before your reputation is damaged, your alerting thresholds are often substantially higher than the minimum levels of service documented in your SLA.



To improve service reliability, all parts of the business must agree that these are an **accurate measure of user experience** and must agree to use them as a **primary driver for decision making**

For SLOs, SLIs and SLAs to help improve service reliability, all parts of the business must agree that they are an accurate measure of user experience and must also agree to use them as a primary driver for decision making.

Being out of SLO must have concrete, well-documented consequences, just as there are consequences for breaching SLAs.

For example, slowing down the rate of change and directing more engineering effort towards eliminating risks and improving reliability are actions that could be taken to get your product back to meeting its SLOs faster.

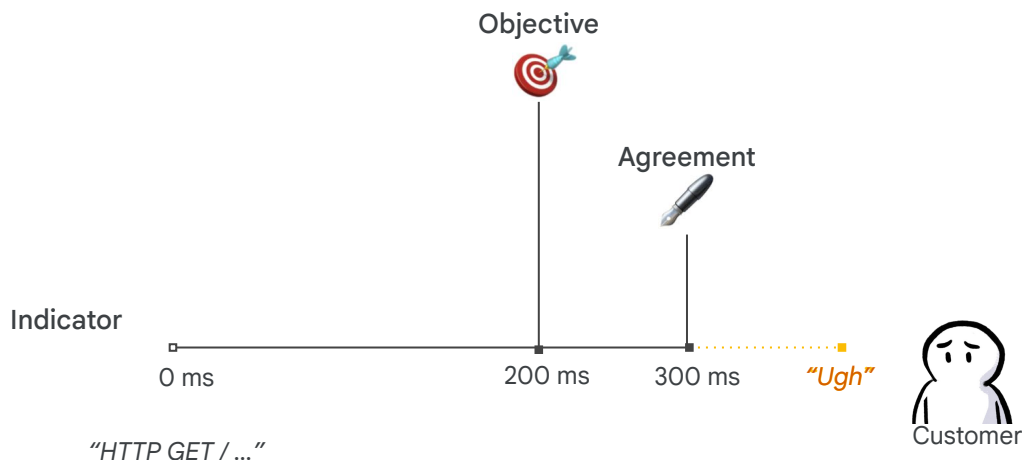
Operations teams need strong executive support to enforce these consequences and effect change in your development practice.

SLI, SLO and SLA example



Here is an example of a SLA, which is to maintain an error rate of less than 0.3% for the billing system. Here error rate is a quantifiable measure which is the SLI and 0.3 is the specific target set which is the SLO in this case.

Deciding the SLA, SLO and SLI



Google Cloud

If your service has paying customers, you probably have some way of compensating them with refunds or credits when that service has an outage.

Your criteria for compensation are usually written into a service level agreement, which describes the minimum levels of service that you promise to provide and what happens when you break that promise.

The problem with SLAs is that you're only incentivized to promise the minimum level of service and compensation that will stop your customers from replacing you with a competitor. When reliability falls far short of the levels of service that keep your customers happy, this contributes to a perception that your service is unreliable, customers often feel the impact of reliability problems before these promises are breached,

Compensating your customers all the time can get expensive, so what targets do you hold yourself to internally?

When does your monitoring system trigger an operational response?

To give you the breathing room to detect problems and take remedial action before your reputation is damaged, your alerting thresholds are often substantially higher than the minimum levels of service documented in your SLA.

SLOs provide another way of expressing these internal reliability targets.

For SLOs to help improve service reliability, all parts of the business must agree that they are an accurate measure of user experience and must also agree to use SLO as a primary driver for decision making.

Your customers probably don't need to be aware of them, but they should have a measurable impact on the priorities of your organization.

Being out of SLO must have concrete, well-documented consequences, just like there are consequences for breaching SLAs.

For example, slowing down the rate of change and directing more engineering effort towards eliminating risks and improving reliability will get you back into SLO faster.

Operations teams need strong executive support to enforce these consequences and effect change in your development practice.

In this section, you explore

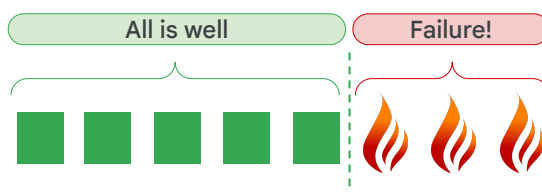


- ✓ SLI, SLO, and SLA
- ✓ **Developing an alerting strategy**
- ✓ Creating alerts
- ✓ Service Monitoring

Let us look at how to develop an alerting strategy next.

Goal: Person is notified when needed

- A service is down.
- SLOs or SLAs are heading toward not being met.
- Something needs to change.



Let's start by defining our term. An alert is an automated notification sent by Google Cloud through some notification channel to an external application, ticketing system, or person.

Why is the alert being sent? Perhaps a service is down, or an SLO isn't being met. Regardless, an alert is generated when something needs to change. The events are processed through a time series: a series of event data points broken into successive, equally spaced windows of time. Based on need, the duration of each window and the math applied to the member data points inside each window are both configurable.

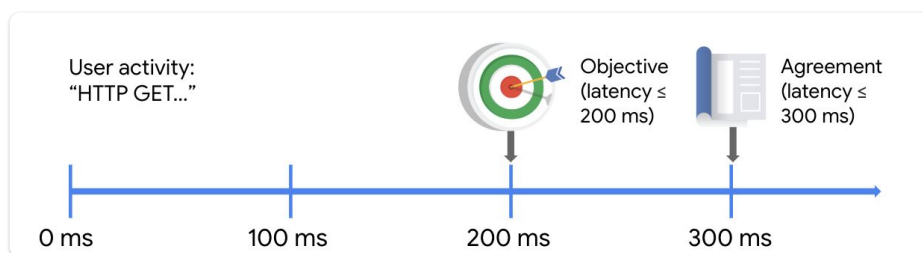
Because of the time series, events can be summarized, error rates can be calculated, and alerts can be triggered where appropriate.

When something puts the error budget in danger: Alert!

Error budget = Perfection - SLO

- SLIs are the things you measure.
- SLOs represent an achievable target.

If the SLO is: "90% of requests must return in 200 ms," then the error budget is: $100\% - 90\% = 10\%$



Google Cloud

A great time to generate alerts is when a system is heading to spend all of its error budget before the allocated time window.

An error budget is perfection minus SLO. SLIs are things that are measured, and SLOs represent achievable targets.

If the SLO target is "90% of requests must return in 200 ms," then the error budget is $100\% - 90\% = 10\%$.

Evaluating alerts

Precision	Recall	Detection time	Reset time
Relevant alerts = Relevant alerts + irrelevant alerts	Relevant alerts = Relevant alerts + missed alerts	How long it takes the system to notice an alert condition	How long alerts fire after an issue is resolved
Adversely affected by false positives	Striving for precision might cause events to be missed	Long detection times can negatively affect the error budget	Continued alerts on repaired systems can lead to confusion
	Recall is adversely affected by missed alerts	Raising alerts too fast may result in poor precision	

Several attributes should be considered when attempting to measure the accuracy or effectiveness of a particular alerting strategy.

Precision is the proportion of alerts detected that were relevant to the sum of relevant and irrelevant alerts. It's decreased by false alerts.

Recall is the proportion of alerts detected that were relevant to the sum of relevant alerts and missed alerts. It's decreased by missing alerts.

Precision can be seen as a measure of exactness, whereas recall is a measure of completeness.

Detection time can be defined as how long it takes the system to notice an alert condition. Long detection times can negatively affect the error budget, but alerting too fast can generate false positives.

Reset time measures how long alerts fire after an issue has been resolved. Continued alerts on repaired systems can lead to confusion.

When error count is trending greater than error budget: Alert!



Windows, as a concept related to SRE and alerting, is frequently used in two main ways:



An SLO of 99.9% is meaningless without a time period.



A **Window** is the period that the error calculation is made over.

Error budgeting 101 would state that when the error count, or whatever is being measured, is trending to be greater than the allowed error budget, an alert should be generated.

Both the SLO itself, and the idea of “trending toward” require windows of time over which they are calculated.

In this subject space, the window term is used in two main ways:

- The SLO itself will be measured over a window of time, say an availability SLO of 99.9% over every 30-day period.
- Alert triggering will have a window over which it watches for trends. For example, an alert might fire if the percentage of errors over any 60-minute period exceeds 0.1%.

Alert window lengths

Small windows

- Faster alert detection
- Shorter reset time
- Poor precision

For a 99.9% availability SLO over 30 days, a 10-minute window would alert in 0.6 seconds if a full outage occurs.

Consume only 0.02% of the error budget.

Longer windows

- Better precision
- Longer reset and detection times
- More error budget spent before alert

For a 99.9% availability SLO over 30 days, a 36-hour window would alert in 2 minutes 10 seconds if a full outage occurs.

Represent 5% of the error budget.

Google Cloud

One of the alerting decisions you and your team have to make is window length. The window is a regular-length subdivision of the SLO total time.

Imagine you set a Google Cloud spend budget of \$1,000 a month. When do you want to receive an alert? When the \$1,000 is spent? Or when the predicted spend is trending past the \$1,000? Of course, the latter.

Now, the same concept, but this time imagine a 99.9% availability SLO over 30 days. You don't want to get an alert when your error budget is already gone. By then it's too late to do anything about the problem.

One option is small windows. Smaller windows tend to yield faster alert detections and shorter reset times, but they also tend to decrease precision because of their tendency toward false positives.

In our 99.9% availability SLO for over 30 days, a 10-minute window would alert in 0.6 seconds if a full outage occurs and would consume only 0.02% of the error budget.

Longer windows tend to yield better precision, because they have longer to confirm that an error is really occurring. But reset and detection times are also longer. That means you spend more error budget before the alert triggers.

In our same 99.9% availability for SLO over 30 days, a 36-hour window would alert in 2 minutes and 10 seconds if a full outage occurs, but would consume a full 5% of the

error budget.

Try short windows with successive failure counts

An error is spotted quickly but treated as an anomaly until **three** windows fail in a row.

Recall becomes worse:

- If the duration is 10 minutes, a 100% outage for 5 minutes is not detected.
- If errors spike up and down, they might never be detected.

One trick might be to use short windows, but add a successive failure count.

One window failing won't trigger the alert, but when three fail in a row the error is triggered. This way, the error is spotted quickly but treated as an anomaly until the duration or error count is reached. This is what you do when your car starts making a sound. You don't immediately freak out, but you pay attention and try to determine whether it's a real issue or a fluke.

The downside is that precision typically has an inverse relationship to recall. As the precision goes up, as you avoid false positives, you let the problem continue to happen.

If the "pay attention but don't alert yet" duration is 10 minutes, a 100% outage for 5 minutes is not detected. As a result, if errors spike up and down, they might never be detected.

Use multiple conditions for better precision and recall

- Many variables can affect a good alerting strategy:
 - Amount of traffic
 - Error budget
 - Peak and slow periods
- You can define multiple conditions in an alerting policy to try to get better precision, recall, detection time, and rest time.
- You can also define multiple alerts through multiple channels:
 - Automated and human
- See the SRE Workbook for more information:
<https://landing.google.com/sre/workbook/chapters/alerting-on-slos/>

So how do we get good precision and recall? This is achieved with multiple conditions.

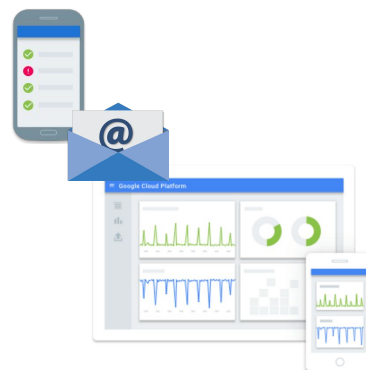
Many variables affect a good alerting strategy, including the amount of traffic, the error budget, peak and slow periods, to name a few.

The fallacy is believing that you have to choose a single option. Define multiple conditions in an alerting policy to get better precision, recall, detection time, and rest time.

You can also define multiple alerts through multiple channels. Perhaps a short window condition generates an alert, but it takes the form of a Pub/Sub message to a Cloud Run container, which then uses complex logic to check multiple other conditions before deciding whether a human gets a notification.

Prioritize alerts based on customer impact and SLA

- Involve humans only for critical alerts.
- Use severity level to assess the priority of an alert.
- Configure how to triage low and high priority alerts.



Google Cloud

And alerts should always be prioritized based on customer impact and SLA.

Don't involve humans unless the alert meets some threshold for criticality.

You can use severity levels as an important concept in alerting to aid you and your team in properly assessing which notifications should be prioritized. You can use these levels to focus on the issues deemed most critical for your operations and triage through the noise. You can create custom severity levels on your alert policies and have this data included in your notifications for more effective alerting and integration with downstream third-party services.

The notification channels were enhanced to accept this data—including Email, Webhooks, Cloud Pub/Sub, and PagerDuty. This enables further automation and customization based on importance wherever the notifications are consumed.

High-priority alerts might go to Slack, SMS, and/or maybe even a third-party solution like PagerDuty. You can even use multiple channels together for redundancy.

Low-priority alerts might be logged, sent through email, or inserted into a support ticket management system.

In this section, you explore



- ✓ SLI, SLO, and SLA
- ✓ Developing an alerting strategy
- ✓ **Creating alerts**
- ✓ Service Monitoring

We've discussed some of the alerting concepts and strategies. Let's look at how Google Cloud create alerts in Google Cloud.

Use alerting policies to define alerts

An alerting policy has:

- A name
- One or more conditions
- Notifications
- Documentation



Google Cloud

Google Cloud defines alerts by using alerting policies.

An alerting policy has:

- A name
- One or more alert conditions
- Notifications
- Documentation

For the name, use something descriptive so you can recognize alerts after the fact. Organizational naming conventions can be a great help.

<https://cloud.google.com/sdk/gcloud/reference/alpha/monitoring/channels/create>

Alert policies can also be created from the gcloud CLI, the API and Terraform. It starts with an alert policy definition in either a JSON or YAML format. One neat trick when learning the correct file format is to create an alert using the Google Cloud console. Then use the *gcloud monitoring policies list* and the *describe* commands to see the corresponding definition file.

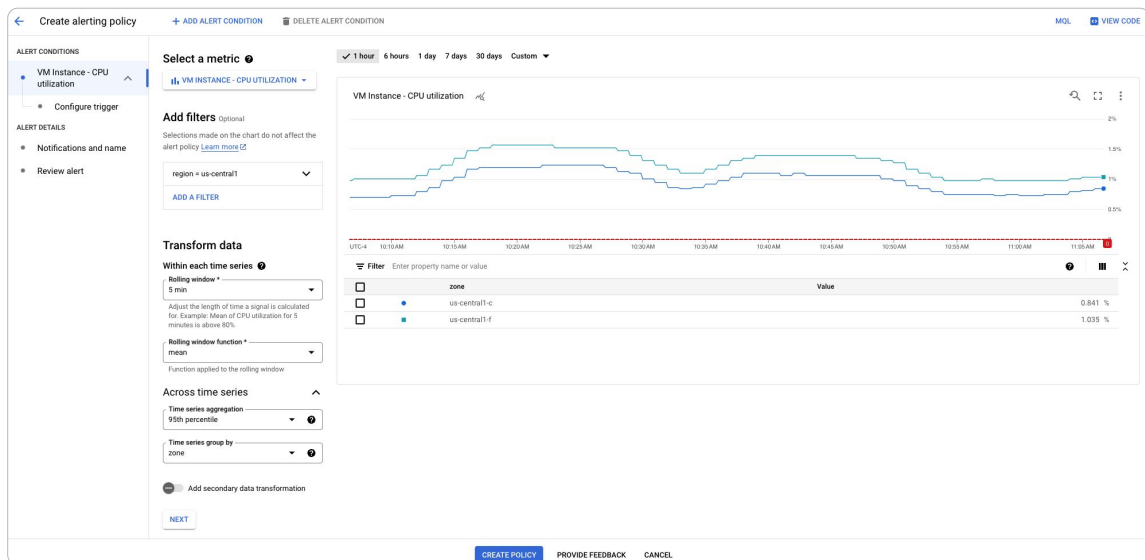
The alerting API and gcloud CLI can create, retrieve, and delete alerting policies.

Alerting policies are of two types

Metric-based alerting policies	Log-based alerting policies
Used to track metric data collected by Cloud Monitoring	Used to notify anytime a specific message occurs in a log ,
Add a metric-based alerting policy by starting from the Alerts page of Cloud Monitoring.	Add a log-based alerting policy by using the Logs Explorer in Cloud Logging or under Cloud Monitoring
Example: Notify when the application that runs on a VM has high latency for a significant time period.	Example: Notify when a human user accesses the security key of a service account.

The alerting policies are of two types:

- **Metric based alerting:** Policies used to track metric data collected by Cloud Monitoring are called metric-based alerting policies. You can add a metric-based alerting policy to your Google Cloud project by using the Google Cloud console. A classic example of a metric-based alerting policy to notify when the application running on a VM has high latency for a significant time period.
- **Log based alerting:** Notified anytime a specific message occurs in a log. You can add a log-based alerting policy to your Google Cloud project by using the Logs Explorer in Cloud Logging or by using the Cloud Monitoring API. An example of log-based alerting policy is to notify when a human user accesses the security key of a service account.



The alert condition is where you spend the most alerting policy time and make the most decisions. The alert condition is where you decide what's being monitored and under what condition an alert should be generated. Notice how the web interface combines the heart of the Metrics Explorer with a configuration condition.

You start with a target resource and metric you want the alert to monitor. You can filter, group by, and aggregate to the exact measure you require.

Configure alert triggers

Proprietary + Confidential

Configure alert trigger

Condition Types

- ☒ **Threshold**
Condition triggers if a time series rises above or falls below a value for a specific duration window
- ☐ **Metric absence**
Condition triggers if any time series in the metric has no data for a specific duration window
- ☐ **Forecast** PREVIEW
Condition triggers if any timeseries in the metric is projected to cross the threshold in the near future.

Alert trigger: Any time series violates

Threshold position: Above threshold

Threshold value: 3

Advanced Options

Condition name *: VM Instance - CPU utilization

1 hour 6 hours 1 day 7 days 30 days Custom

VM Instance - CPU utilization

UTC-4 10:10 AM 10:15 AM 10:20 AM 10:25 AM 10:30 AM 10:35 AM 10:40 AM 10:45 AM 10:50 AM 10:55 AM 11:00 AM

Filter Enter property name or value

	zone	Value
<input type="checkbox"/>	us-central1-c	
<input type="checkbox"/>	us-central1-f	

CREATE POLICY PROVIDE FEEDBACK CANCEL

Google Cloud

Then the yes-no decision logic for triggering the alert notification is configured. It includes the trigger condition, threshold, and duration.

There are three types of conditions for metric-based alerts:

- Metric-threshold conditions trigger when the values of a metric are more than, or less than, a threshold for a specific duration window.
- Metric-absence conditions trigger when there is an absence of measurements for a duration window.
- Forecast conditions predict the future behavior of the measurements by using previous data. These conditions trigger when there is a prediction that a time series will violate the threshold within a forecast window.

Configure notifications

Proprietary + Confidential

The screenshot shows the 'Create alerting policy' interface in Google Cloud. The left sidebar has 'ALERT CONDITIONS' with 'New condition' and 'Configure trigger', and 'ALERT DETAILS' with 'Notifications and name' (selected) and 'Review alert'. The main area is titled 'Configure notifications and finalize alert'. It includes a 'Configure notifications' section with a 'Use notification channel' toggle (checked) and a 'Notification Channels' dropdown menu. Below this is a recommendation box about creating multiple channels for redundancy. Further down is a checkbox for 'Notify on incident closure' and a dropdown for 'Incident autoclose duration' (set to 7 days). The 'Policy user labels' section explains that labels can be added for organization and severity. At the bottom, there is a 'Documentation' section for optional markdown. The footer contains 'CREATE POLICY', 'PROVIDE FEEDBACK', and 'CANCEL' buttons, along with a 'Show debug log' link.

Google Cloud

An alert might have zero to many notification options selected, and they each can be of a different type. There are direct-to-human notification channels (Email, SMS, Slack, Mobile Push), and for third-party integration use Webhook and Pub/Sub.

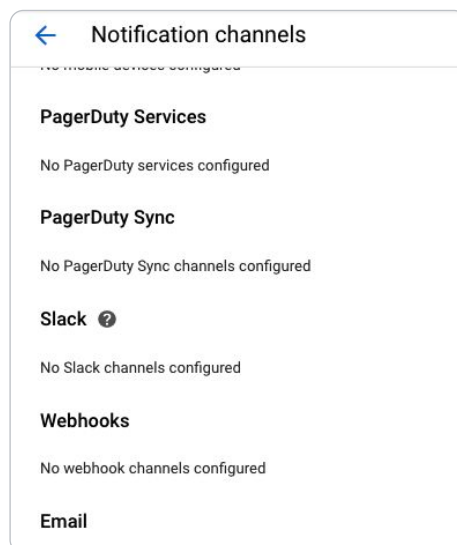
Manage your notifications and incidents by adding user-defined labels to an alerting policy. Because user-defined labels are included in notifications, if you add labels that indicate the severity of an incident, then the notification contains information that can help you prioritize your alerts for investigation.

If you send notifications to a third-party service like PagerDuty, Webhooks, or Pub/Sub then you can parse the JSON payload and route the notification according to its severity so that so that your team doesn't miss critical information.

Select notification channels

Supported notification channels include:

- Email
- SMS
- Slack
- Google Cloud app
- PagerDuty
- Webhooks
- Pub/Sub



A notification channel decides how the alert is sent to the recipient.

Alerts can be routed to any third-party service.

Email alerts are easy and informative, but they can become notification spam if you aren't careful.

SMS is a great option for fast notifications, but choose the recipient carefully.

Slack is very popular in support circles.

The Google Cloud app for mobile devices is a valid option.

PagerDuty is a third-party on-call management and incident response service.

Webhooks and Pub/Sub are excellent options when you want to alert users to external systems or code.

Documentation to guide troubleshooting

- Use the documentation section to guide your troubleshooting.
- Include internal playbooks, landing links and dynamic labels.
- Make it easy for the team to understand what is wrong.

Proprietary + Confidential

← Create alerting policy + ADD ALERT CONDITION DELETE ALERT CONDITION

7 days
If data is absent, select a duration after which Incident will automatically close.

ALERT CONDITIONS

- New condition ^
- Configure trigger

ALERT DETAILS

- Notifications and name
- Review alert

Policy user labels Recommended
Policy user labels allow you to add your own labels to alert policies for organization. The labels are included in the notification and incident details. It is recommended to use this for managing the [severity level](#) of the alert.

+ ADD LABEL

Documentation Optional
Enter any documentation you would like included with the notification. You can use markdown, [variables](#), and channel-specific controls. Markdown formatting may not apply to all notification channels.

Text Field

Name the alert policy

Google Cloud


The documentation option is designed to give the alert recipient additional information they might find helpful. Use the documentation section to guide your troubleshooting.

Include internal playbooks, landing links and dynamic labels

The default alert contains information about which alert is failing and why, so think of this more like an easy button. If there's a standard solution to this particular alert, adding a reference to it here might be a good example of proper documentation inclusion.

Then again, if it was that easy, automate it!

Email alert example

 **Alert firing**

Burn rate on 99.5% - Availability - Rolling 7 days

SLO Burn Rate for qwiklabs-gcp-be2e04b8437001a4 GAE Application is above the threshold of 1 with a value of 1.137.

Summary

Start time
Sep 21, 2020 at 12:45AM UTC (~4 minutes ago)

Project
[quwiklabs-gcp-be2e04b8437001a4](#)

Policy
[Burn rate on 99.5% - Availability - Rolling 7 days](#)

Condition
Burn rate on 99.5% - Availability - Rolling 7 days

Metric
`select_slo_burn_rate("projects/453214603901/services/gae:quwiklabs-gcp-be2e04b8437001a4_default/serviceLevelObjectives/srk0K6FTQIvooeeakGhlg", "600s")`

Threshold
above 1

Observed
1.137

Policy documentation

Easy Button

1. Do this
2. Do that

[VIEW INCIDENT](#)

Proprietary + Confidential

Google Cloud

Here, you see an alert notification sent out through an email. Notice how many details about exactly what went wrong are automatically included in the email body. The bottom documentation section can also be used to augment the provided information.

Alerting + CREATE POLICY EDIT NOTIFICATION CHANNELS

Summary

Incidents firing: 1

Incidents acknowledged: 1

Alert policies: 7 [View all](#)

Incidents [SHOW CLOSED INCIDENTS](#)

State	Policy name	Incident summary	Opened
	VM CPU Utiliz...	CPU utilization for lees-gmp VM Instance labels (project_id=lees-gmp, zone=us-central1-f) with system labels (region=us-ce...	Jul 3, 2023, 11:39:53 AM
	VM CPU Utiliz...	CPU utilization for lees-gmp VM Instance labels (project_id=lees-gmp, zone=us-central1-c) with system labels (region=us-ce...	Jul 3, 2023, 11:39:53 AM

[→ See all incidents](#)

Snoozes [CREATE SNOOZE](#) [Show past snoozes](#)

State	Name	Start time	End time	
Active	Log bytes	Jul 3, 11:39 AM	Jul 3, 12:39 PM	VIEW DETAILS

[→ See all snoozes](#)

When one or more alert policies are created, the alerting web interface provides a summary of incidents and alerting events. An event occurs when the conditions for an alerting policy are met. When an event occurs, Cloud Monitoring opens an incident.

In the **Alerting** window, the **Summary** pane lists the number of incidents, and the **Incidents** pane displays the ten most recent incidents. Each incident is in one of three states:

- **Incidents firing:** If an incident is open, the alerting policy's set of conditions is being met. Or there's no data to indicate that the condition is no longer met. Open incidents usually indicate a new or unhandled alert.
- **Acknowledged incidents:** A technician spots a new open alert. Before one starts to investigate, they mark it as acknowledged as a signal to others that someone is dealing with the issue.
- **Alert policies** displays the number of alerting policies created.

The **Incidents pane** displays the most recent open incidents. To list the most recent incidents in the table, including those that are closed, click **Show closed incidents**.

Snooze displays the recently configured snoozes. When you want to temporarily prevent alerts from being created and notifications from being sent, or to prevent repeated notifications from being sent for an open incident, you create a snooze. For example, you might create a snooze when you have an escalating outage and you want to reduce the number of new notifications.

Resource groups can monitor multiple resources

- Trigger based on the group instead of on individual resources.
- Groups can contain subgroups up to six levels deep.
- Resources can be members of more than one group.

Groups provide a mechanism for alerting on the behavior of a set of resources instead of individual resources. For example, you can create an alerting policy that is triggered if some resources in the group violate a condition (for example, CPU load), instead of having each resource inform you of violations individually.

Groups can contain subgroups and can be up to six levels deep. One application for groups and subgroups is the management of physical or logical topologies. For example, with groups, you can separate your monitoring of production resources from your monitoring of test or development resources. You can also create subgroups to monitor your production resources by zone.

Resources can belong to multiple groups.

Use multiple criteria to create resource groups

Criteria can include:

- Cloud Projects
- Resource name
- Resource type
- Tags and labels
- Security groups
- Regions
- App Engine apps and services

Groups let you define alerts on a set of resources.

Name *

Pets K8S Cluster

Criteria

Add criterion

Type *

Name

Operator *

Starts with

Value *

gke-pets-cluster-pool

CANCEL DONE

ADD CRITERION

CREATE CANCEL

You define the one-to-many membership criteria for your groups. A resource belongs to a group if the resource meets the membership criteria of the group. Membership criteria can be based on resource name or type, Cloud Projects, network tag, resource label, security group, region, or App Engine app or service. Resources can belong to multiple groups.

Log-based Metrics

About log-based metrics

Log-based metrics count the log entries that match a given filter. Project scoped log-based metrics only apply to logs generated in this project, and bucket scoped log-based metrics only apply to logs exported to buckets in this project.

User-defined metrics [CREATE METRIC](#) [DELETE](#)

System-defined metrics

Create alerting policy [+ ADD ALERT CONDITION](#) [DELETE ALERT CONDITION](#)

ALERT CONDITIONS

- New condition
 - Configure trigger

ALERT DETAILS

- Notifications and name
- Review alert

Select a metric

GLOBAL - LOG BYTES INGESTED

Add filters Optional

Selections made on the chart do not affect the alert policy [Learn more](#)

[ADD A FILTER](#)

Transform data

Within each time series

Rolling window * 10 min

Adjust the length of time a signal is calculated for. Example: Mean of CPU utilization for 5 minutes is above 80%

Rolling window function * delta

Function applied to the rolling window

View in Metrics Explorer

Create alert from metric

Logs-based metrics are extracted from Cloud Monitoring and are based on the content of log entries. For example, the metrics can record the number of log entries that contain particular messages, or they can extract latency information reported in log entries. You can use logs-based metrics in Cloud Monitoring charts and alerting policies.

As we covered earlier in this module, an alerting policy describes a set of conditions that you want to monitor. When you create an alerting policy, you must also specify its conditions: what is monitored and when to trigger an alert. The logs-based metrics serve as the basis for an alerting condition.

Create policies by using Terraform

```
resource "google_monitoring_alert_policy" "alert_policy" {  
  display_name = "My Alert Policy"  
  
  combiner = "OR"  
  
  conditions {  
    display_name = "test condition"  
  
    condition_threshold {  
  
      filter = "metric.type=\"compute.googleapis.com/instance/disk/write_bytes_count\" AND  
resource.type=\"gce_instance\""  
  
      duration = "60s"  
      comparison = "COMPARISON_GT"  
      aggregations {  
        alignment_period = "60s"  
        per_series_aligner = "ALIGN_RATE"  
      }  
    }  
  }  
}
```

Google Cloud

To create alerting policies by using Terraform, start by creating a description of the conditions under which some aspect of your system is considered to be "unhealthy" and the ways to notify people or services about this state.

Shown on slides is a basic monitoring alert policy with the name `alert_policy`. The code includes three required arguments:

1. `Display_name`: This argument helps identify the policy with a name that can be seen on the dashboard.
2. `Combiner`: This argument defines how the results of multiple conditions have to combined.
3. `Conditions`: This argument defines a list of conditions that are combined based on the combiner. A policy can have up to six conditions.

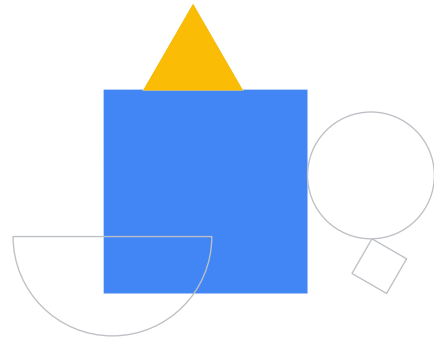
See documentation for more policy examples

- [Group Aggregate Policy](#)
- [Uptime Check Policy](#)
- [Process Health Policy](#)
- [Metric Ratio Policy](#)
- [Setting for Common Alerting Policies](#)
- [Alerting JSON files](#)

For some more policy examples, visit the links listed on this slide.

Lab Intro

Alerting in Google Cloud



Google Cloud

In this lab, you deploy an application to Google Cloud. You then create alerting policies that notify you if the application is not up or is generating errors.

In this section, you explore



- ✓ SLI, SLO, and SLA
- ✓ Developing an alerting strategy
- ✓ Creating alerts
- ✓ **Service Monitoring**

Now that we've examined alerts, their use, and their creation, let's see how the Service Monitoring console and API can help.

Service Monitoring helps with SLO and alert creation

What are the **commitments regarding the availability and performance** of those services, and are your services meeting them?

For microservices-based apps, what are the **inter-service dependencies**?

How to double **check new code rollouts and triage problems** if a service degradation occurs

Can you **look at all the monitoring signals for a service holistically** to reduce MTTR?

Google Cloud

Modern applications are composed of multiple services connected together, and when something fails, it often seems like many things fail at the same time. To help manage this complexity, SLO monitoring helps with SLO and alert creation.

With Service Monitoring, you get the answers to the following questions:

- What are your services? What functionality do those services expose to internal and external customers?
- What are your promises and commitments regarding the availability and performance of those services, and are your services meeting them?
- For microservices-based apps, what are the inter-service dependencies? How can you use that knowledge to double check new code rollouts and triage problems if a service degradation occurs?
- Can you look at all the monitoring signals for a service holistically to reduce mean time to repair (MTTR)?

Cloud Monitoring can identify potential or candidate services for the following types:

- GKE namespaces
- GKE services
- GKE workloads
- Cloud Run services

Consolidated services overview

Supported Services		Services				
Select service type		Filter Enter a service name or value				
All services	11	Name ↑	Type	SLOs out of error budget ?	SLOs with firing alert ?	Labels
		cartservice	Custom	0 / 0	0 / 0	GKE: Kubernetes Service project_id: sd-uxr-001
App Engine	4	Cloud Comp...	Custom	0 / 0	0 / 0	resource_name: project_id: sd-uxr-001
Istio	0	Define-New-S...	Custom	0 / 0	0 / 0	resource_name: project_id: sd-uxr-001
Kubernetes	1	frontend	Custom	0 / 1	0 / 1	GKE: Kubernetes Service project_id: sd-uxr-001
Anthos	0	kubernetes	GKE Service	0 / 0	0 / 0	cluster_name: autopilot-cluster-1 location: us-central1
Cloud Run	0	list-metrics	App Engine	0 / 1	0 / 1	project_id: sd-uxr-001 module_id: list-metrics
Custom	6	sd-uxr-001/d...	App Engine	0 / 0	0 / 0	project_id: sd-uxr-001 module_id: default
		sd-uxr-001/g...	App Engine	0 / 0	0 / 0	project_id: sd-uxr-001 module_id: get-timeseries
		sd-uxr-001/w...	App Engine	0 / 0	0 / 0	project_id: sd-uxr-001 module_id: write-metrics
		Test-Service	Custom	0 / 0	0 / 0	resource_name: project_id: sd-uxr-001
		User-API	Custom	0 / 0	0 / 0	resource_name: project_id: sd-uxr-001

Rows per page: 50 1 - 11 of 11 < >

The Service Monitoring consolidated services overview page is your point of entry. **Services** pane provides a summary of the health of your various services. Here, you can see the service name, type, SLO status, and whether any SLO-related alerts are firing.

To monitor or view details for a specific service, click the service name.

You can also filter by entering a value in the **Filter** text box to apply additional conditions.

Windows-based versus request-based SLOs

- Imagine you get 1,000,000 requests a month and your compliance period is a rolling 30 days.
- A 99.9% request-based SLO can allow 1,000 bad requests every 30 days.
- A 99.9% windows-based SLO based on a 1-minute window can allow a total of 43 bad windows.
 - $43,200 \text{ total windows} * 99.9\% = 43,157 \text{ good windows}$.
- Windows-based SLOs can be tricky, because they can hide burst-related failures.

Service Monitoring can approach SLO compliance calculations in two fundamental ways.

Request-based SLOs use a ratio of good requests to total requests.

For example, we want a request-based SLO with a latency below 100 ms for at least 95% of requests. So it is convenient for us if 98% of requests were faster than 100 ms.

Windows-based SLOs use a ratio of the number of good versus bad measurement intervals, or windows. So each window represents a data point, instead of all the data points that comprise the window.

For example, take a 95th percentile latency SLO that needs to be less than 100 ms for at least 99% of 10-minute windows. Here, a compliant window is a 10-minute period over which 95% of the requests were less than 100 ms. So it is convenient for us if 99% of 10-minute windows were compliant.

Let's look at another pair of window-based versus request-based SLO examples.

Imagine you get 1,000,000 requests a month, and your compliance period is rolling for 30 days.

If you are looking for a 99.9% request-based SLO, that translate to 1,000 total bad requests every 30 days.

However, a 99.9% windows-based SLO which is averaged across 1-minute windows, allows a total of 43 bad windows, or $43,200 \text{ total windows} * 99.9\% = 43,157$ good windows.

Windows-based SLOs can be tricky because they can hide burst-related failures. If the system returns nothing but errors, but only every Friday morning from 9:00-9:05, then you will never violate your SLO. However, not many people prefer to use the system first thing Friday morning.

Service Monitoring makes SLO creation easy

+ CREATE SLO

Create a Service Level Objective (SLO) [SEND FEEDBACK](#) [LEARN MORE](#)

- 1 Set your service-level indicator (SLI)
Choose the aspect of service health for which you want to set a performance goal
- 2 Define SLI details
Specify more details for the metric you've chosen
- 3 Set your service-level objective (SLO)
Set targets for how well your service should perform
- 4 Review and save
Review details and name your SLO

Set your SLI

First, choose the aspect of service health for which you want to set a performance goal. This is used to calculate the service level indicator, or SLI, because it indicates how your service is performing for your users.

Service details

NAME	TYPE	LABELS
default	App Engine	project_id: 882581100213 module_id: default

Choose a metric

☒ **Availability**
Measures how available your service was to users. You'll get a metric related to how many requests were successful within a time period that you define.

☐ **Latency**
Measures how quickly your service responded to users. You'll get a metric related to how many responses were faster than a threshold that you define.

☐ **Other (advanced)**
Configure your own metrics to measure the performance of your service.

Request-based or windows-based?
The method of evaluation you choose will affect how compliance is measured.

☒ **Request-based**
Counts individual events. This lets you know how well your service performed over the entire compliance period, no matter how the load was distributed.

☐ **Windows-based (advanced)**
Counts "good minutes" versus "bad minutes" according to criteria you define. This lets you measure performance in terms of time, regardless of how load is distributed.

← BACK **NEXT** →

Google Cloud

Service Monitoring makes SLO creation easy. On the **Services** overview page, select one of the listed services. If a service is built on a Google Cloud compute technology that supports Service Monitoring, it will be automatically listed.

Next, click **Create an SLO**.

Select an option from the SLI metric. The options include:

- **Availability** is a ratio of the number of successful responses to the number of all responses.
- **Latency** is the ratio of the number of calls that are below the specified **Latency Threshold** to the number of all calls.
- The option **Other** gives you the Metrics Explorer window and lets you create your own SLI from the beginning.

As previously discussed, you can also select request-based or windows-based SLOs here.

Set compliance period and goal

Update a Service Level Objective

[SEND FEEDBACK](#) [LEARN MORE](#)

- Set your service-level indicator (SLI)
Choose the aspect of service health for which you want to set a performance goal
- Define SLI details
Specify more details for the metric you've chosen
- Set your service-level objective (SLO)**
Set targets for how well your service should perform
- Review and save
Review details and name your SLO

Set your SLO

Define a time period for compliance and set a goal for "good service." This is called a service level objective (SLO).

Compliance period

Select the time period you want to use for evaluating your SLO.

Period type

Rolling

Period length

7 Days

Performance goal

Set a goal for the ratio of "good service" to "demanded service" over the compliance period. You can refine this later as you learn about a system's behavior. It's recommended to start with a loose goal that you tighten than an overly strict goal.

Goal

99.5 %

Preview

Based on current parameters, using historical data

← BACK

NEXT →

In the **Compliance Period** section, select the **Period Type** and the **Period Length**. The two compliance period types are calendar-based and rolling.

In the **Performance Goal** section, enter a percentage in the **Goal** field to set the performance target for the SLI. Service Monitoring uses this value to calculate the error budget you have for this SLO.

SLO linked alerts are easy to create

Current status of 1 SLO Status calculated at 3:51 PM GMT-5 + CREATE SLO

Status	Objective	Type	Alerts firing	Error budget
✓ Healthy	99.5% - Availability - Rolling 7 days	Availability SLI	EDIT	DELETE ^
Service level indicator 📈 99.92%		Error budget ✓ 79.91%	Alerts firing ✓ 0/1	
Policy name		Open incidents	Status	
Burn rate on 99.5% - Availability - Rolling 7 days		0	Enabled View policy	
+ CREATE ALERTING POLICY				

You can create an alert by just clicking Create alerting policy. Click an individual service on the Services Overview page to view its details. There, you can see existing SLOs and, by expanding them, their details. The SLI status, the error budget remaining, and the current level of SLO compliance are all displayed. If alerts have been set, their status is also displayed.

Alert if the burn rate is too high

×

Create SLO burn rate alert policy

1

Set SLO alert conditions

Creating an alert condition on your service-level objectives (SLOs) will let you know whether you are in danger of violating an SLO.

Target: 99.5% - Availability - Rolling 7 days

Select a burn rate threshold value that constitutes a violation, and a lookback duration period for which the violation is permitted. If the burn rate threshold is exceeded for more than the allowable period, an incident is created. [Learn more](#)

Display name *

Burn rate on 99.5% - Availability - Rolling 7 days

Lookback duration *

60minute(s)?

Burn rate threshold *

10?

NEXT

2

Who should be notified? (optional)

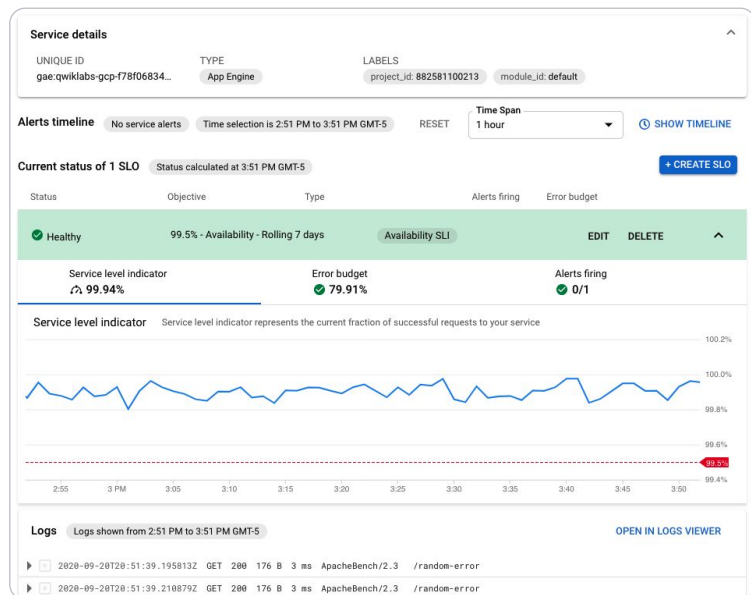
When alerting policy violations occur, you will be notified via these channels.

3

What are the steps to fix the issue? (optional)

Service Monitoring can trigger an alert when a service is heading to violate an SLO. The alerting policy uses a lookback window to examine a period for trends. The burn rate threshold is then used to determine whether an alert should be raised. In this example, we are using a 60-minute window. A burn rate threshold of 1 would use 100% of the error budget by the end of the 7-day period. In this case, if we see a trend that would burn through our total error budget in 1/10th of those seven days or faster, then we should raise an alert.

SLO status is easy to monitor

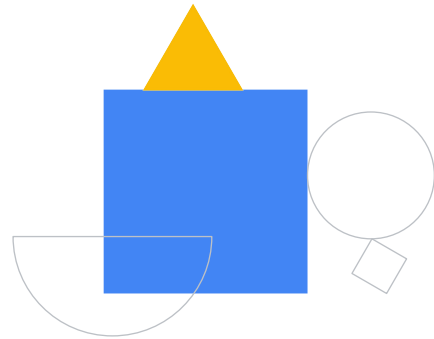


Google Cloud

After the SLO is created, it's easy to monitor the SLI status, error budget, compliance, and alert status. In this case, I've selected to view the **Service level indicator**, so it's displaying the current SLI values. You could also examine the **Error budget** or **Alerts firing** tabs for more detail on those.

Demo

Using Service Monitoring



Google Cloud

Google Cloud Service Monitoring streamlines the creation of Service Level Objectives based on latency- and availability-based Service Level Indicators. In this demo, you will learn how to use Service Monitoring to create a 99.5% availability SLO and corresponding alert.



Knowledge Check



Quiz | Question 1

Question

In evaluating your alerting policies, which option best describes precision?

- A. The proportion of events detected that were relevant to the sum of relevant and irrelevant alerts.
- B. How long it takes to send notifications in various conditions.
- C. How long alerts fire after an issue is resolved.
- D. How long alerts take to be acknowledged.

Quiz | Question 1

Answer

In evaluating your alerting policies, which option best describes precision?

- A. The proportion of events detected that were relevant to the sum of relevant and irrelevant alerts.
- B. How long it takes to send notifications in various conditions.
- C. How long alerts fire after an issue is resolved.
- D. How long alerts take to be acknowledged.



Quiz | Question 2

Question

Explain recall.

- A. The proportion of alerts detected that were relevant to the sum of relevant and irrelevant alerts.
- B. The proportion of alerts detected that were relevant to the sum of relevant alerts and missed alerts.
- C. How long it takes to send notifications in various conditions.
- D. How long alerts fire after an issue is resolved.

Quiz | Question 2

Answer

Explain recall.

- A. The proportion of alerts detected that were relevant to the sum of relevant and irrelevant alerts.
- B. The proportion of alerts detected that were relevant to the sum of relevant alerts and missed alerts.
- C. How long it takes to send notifications in various conditions.
- D. How long alerts fire after an issue is resolved.



Quiz | Question 3

Question

In the statement “Maintain an error rate of less than 0.3% for the billing system”, what is an SLI?

- A. 0.3%
- B. Error rate
- C. Billing system
- D. Less than 0.3%

Quiz | Question 3

Answer

In the statement “Maintain an error rate of less than 0.3% for the billing system”, what is an SLI?

- A. 0.3%
- B. Error rate
- C. Billing system
- D. Less than 0.3%



Recap

- 01 Explain why SLI, SLO, and SLA are important.
- 02 Explain alerting policies and alerting strategies.
- 03 Explain error budgets.
- 04 Identify types of alerts and common uses for each.
- 05 Use Cloud Monitoring to manage services.



In this module, you learned how to:

- Explain why SLI, SLO, and SLA are important.
- Explain alerting policies and alerting strategies.
- Explain error budgets.
- Identify types of alerts and common uses for each.
- Use Cloud Monitoring to manage services.

