

## Google Kubernetes Engine Logging and Monitoring

Logging and monitoring are crucial for maintaining stable and healthy GKE deployments. They provide deep visibility into cluster and application behavior, helping you proactively identify any issues, debug them quickly, and optimize resource utilization.

# Google Kubernetes Engine Logging and Monitoring

- 01 Google Cloud Observability
- 02 Cloud Logging
- 03 Inspecting logs with the kubectl command
- 04 Inspecting logs with Cloud Logging and Logging Agents
- 05 Cloud Monitoring



Google Cloud

In this section of the course, titled “Google Kubernetes Engine Logging and Monitoring,” you’ll:

- Explore how Google Cloud Observability is used to manage the availability and performance of your GKE resources.
- Examine the benefits and features available with Cloud Logging and Cloud Monitoring.
- Inspect logs with the kubectl command.
- Inspect logs with Cloud Logging and Logging Agents.
- And get hands on practice configuring GKE-Native Monitoring and Logging.

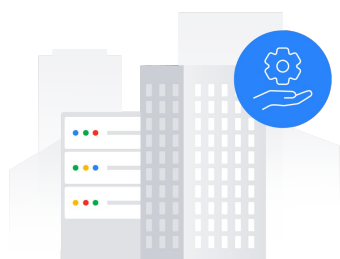
# Google Kubernetes Engine Logging and Monitoring

- 01 Google Cloud Observability
- 02 Cloud Logging
- 03 Inspecting logs with the kubectl command
- 04 Inspecting logs with Cloud Logging and Logging Agents
- 05 Cloud Monitoring



Let's begin with Google Cloud Observability.

## Observability on-premises versus in the cloud



On-premises

Physical check

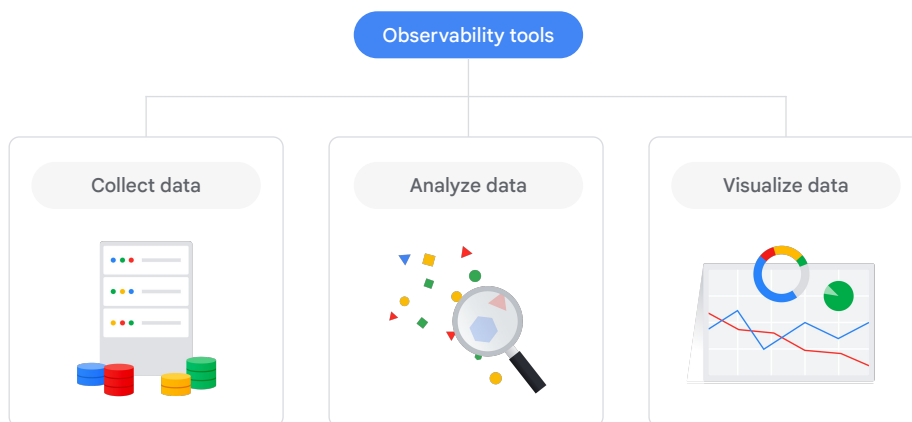


Cloud

Observability tools

If you've ever worked with on-premises environments, you know that you can physically touch the servers. If an application becomes unresponsive, someone can physically determine the cause. In the cloud though, the servers aren't yours—they belong to the cloud provider—and you can't physically inspect them.

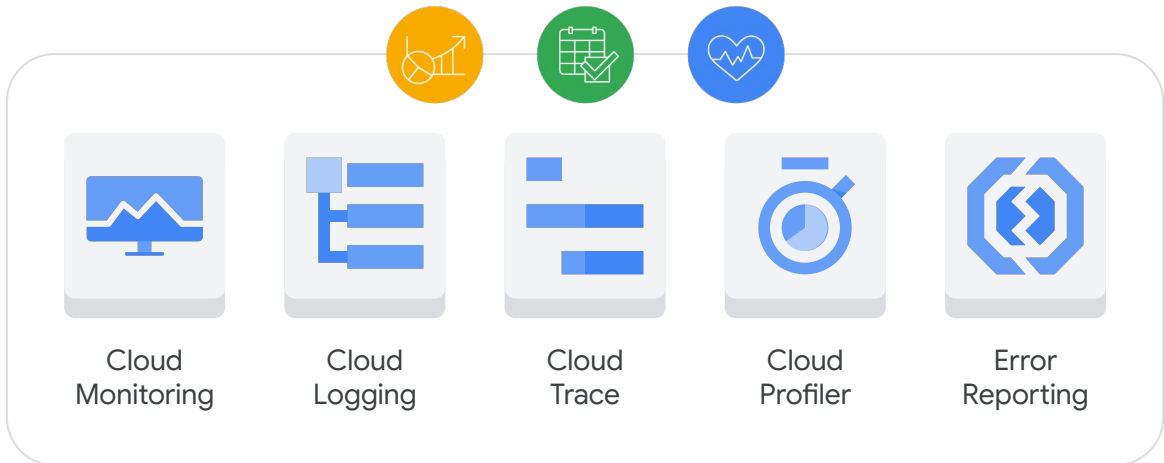
# How can you know what's happening with your apps?



So the question becomes: how can you know what's happening with your applications?

The answer is: by using Google's integrated observability tools. Observability involves collecting, analyzing, and visualizing data from various sources within a system to gain insights into its performance, health, and behavior.

# Google Cloud Observability

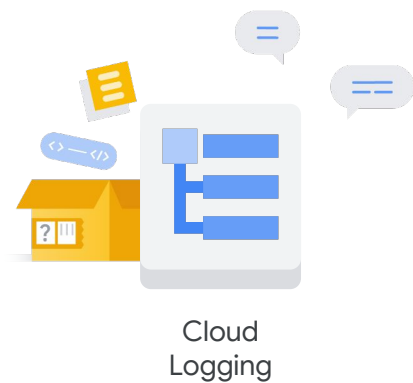


Google Cloud

To achieve this, Google Cloud offers observability tools for monitoring, logging, and diagnostics. It's a unified platform for managing and gaining insights into the performance, availability, and health of applications deployed on Google Cloud.

Let's explore some of the managed services that constitute Google Cloud Observability.

# Cloud Logging



Collects and stores all application and infrastructure logs.



Real-time insights help troubleshoot issues, identify trends, and comply with regulations.

The first is **Cloud Logging**, which collects and stores all application and infrastructure logs. With real-time insights, Cloud Logging can help to troubleshoot issues, identify trends, and comply with regulations.

# Cloud Monitoring



Cloud  
Monitoring

- ✓ Provides a comprehensive view of cloud infrastructure and applications.
- ✓ Collects metrics from applications and infrastructure, and provides insights into their performance, health, and availability.
- ✓ Lets you create alerting policies to notify when metrics, health check results, and uptime check results meet specified criteria.

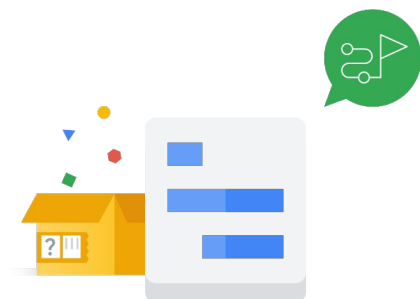
**Cloud Monitoring** provides a comprehensive view of your cloud infrastructure and applications. It collects metrics from your applications and infrastructure, and provides insights into their performance, health, and availability.

It can also be used to create alerting policies to notify you when metrics, health check results, and uptime check results meet specified criteria.

Graphs produced in Cloud Monitoring provide visual indication for when events have occurred, such as performance bottlenecks or service dependencies. Cloud Monitoring can also be integrated with many third-party products.



# Cloud Trace



Cloud  
Trace



Helps identify performance bottlenecks in applications.



Collects latency data from applications and provides insights into how they're performing.

**Cloud Trace** helps identify performance bottlenecks in applications. It collects latency data from applications, and provides insights into how they're performing.

# Cloud Profiler



Cloud Profiler



Identifies how much CPU power, memory, and other resources an application uses.



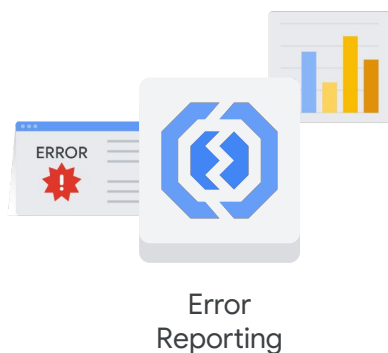
Continuously gathers CPU usage and memory-allocation information from production applications.



Provides insights into how applications are using resources.

**Cloud Profiler** identifies how much CPU power, memory, and other resources an application uses. It continuously gathers CPU usage and memory-allocation information from production applications and provides insights into how applications are using resources.

# Error Reporting



Error Reporting counts, analyzes, and aggregates the crashes in running cloud services in real time.



A centralized error management interface displays the results with sorting and filtering capabilities.



A dedicated view shows the error details: time chart, occurrences, affected user count, first- and last-seen dates.



Error Reporting supports email and mobile notifications through its API.

And finally, there's **Error Reporting**, which counts, analyzes, and aggregates the crashes in running cloud services in real-time. A centralized error management interface displays the results with sorting and filtering capabilities.

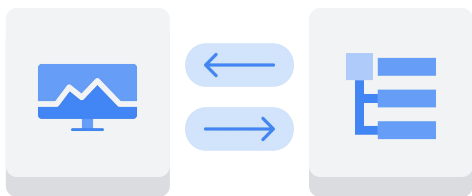
A dedicated view shows the error details, which include a time chart, occurrences, affected user count, first and last-seen dates, and a cleaned exception stack trace. Error Reporting supports email and mobile alerts through its API.

# Google Kubernetes Engine Logging and Monitoring

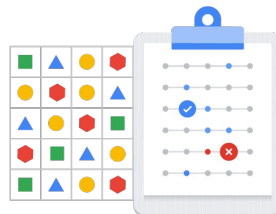
- 01 Google Cloud Observability
- 02 Cloud Logging
- 03 Inspecting logs with the kubectl command
- 04 Inspecting logs with Cloud Logging and Logging Agents
- 05 Cloud Monitoring



## Logging is a passive form of systems monitoring



The process of collecting and storing a record of events that occur in a GKE cluster.

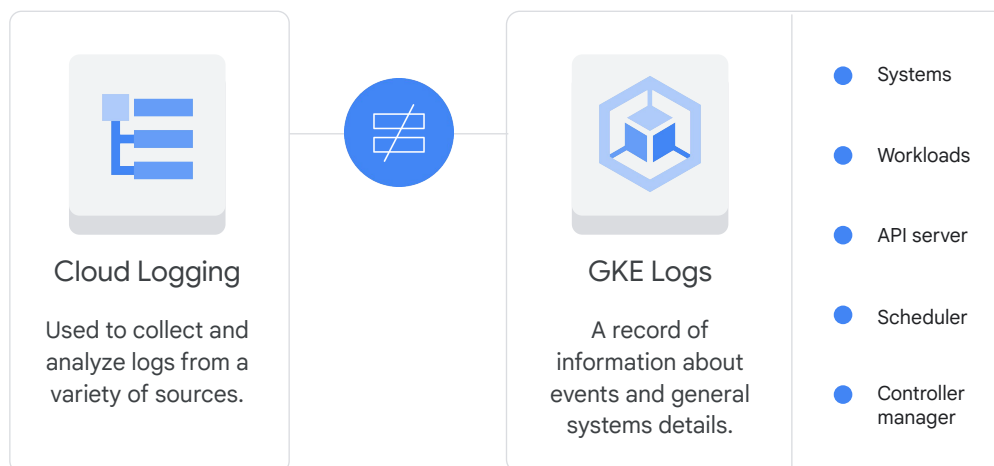


Example: collect events logs to identify patterns or the root cause of a system failure.

Logging with Google Kubernetes Engine is the process of collecting and storing a record of events that occur in a GKE cluster.

Often thought of as a passive form of systems monitoring, logging collects events logs that can be used, for example, to identify patterns or perhaps help track down the root cause of a system failure.

# GKE Logs and Cloud Logging are different



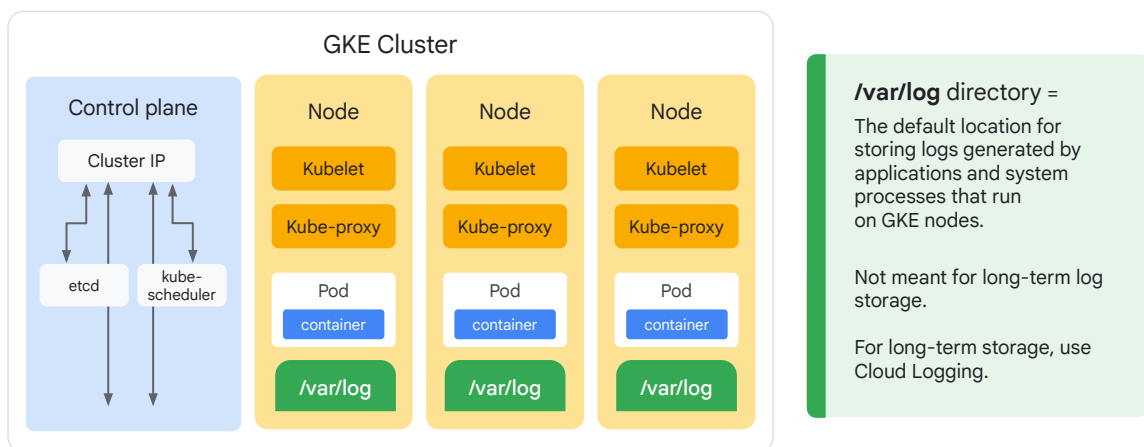
GKE logs, and logs produced by Cloud Logging both provide information to analyze your Google Cloud infrastructure. However, there are some key differences between the tools.

Cloud Logging is a fully managed logging service that can be used to collect logs from a *variety* of sources, including Google Cloud services, like GKE, on-premises infrastructure, and third-party applications. Cloud Logging can be used to consolidate all of your logs in one place and analyze them for trends.

Alternatively, a GKE log is a record of information about cluster events, container events, Pod events, node events, audit events, and general system details. The main types of GKE logs are

- System
- Workloads
- API server
- Scheduler
- Controller Manager

## Kubernetes logs are stored on node file systems



The `/var/log` directory in GKE is the default location for storing logs generated by applications and system processes that run on GKE nodes.

This directory is not meant for long-term storage of logs, because it might be deleted when the node is deleted or when the disk on which it is stored runs out of space.

For long-term storage and analysis of logs, it is recommended to use Cloud Logging.

## Options to view logs

1

### kubectl logs command

```
$ kubectl logs [POD_Name]
```

Display logs directly from a Pod.



Quick, but logs aren't saved to any database.

2

### Cloud Logging

Provides a single interface to review logs from containers, nodes, and other system services.



More options to broaden visibility of GKE events or correlate issues.

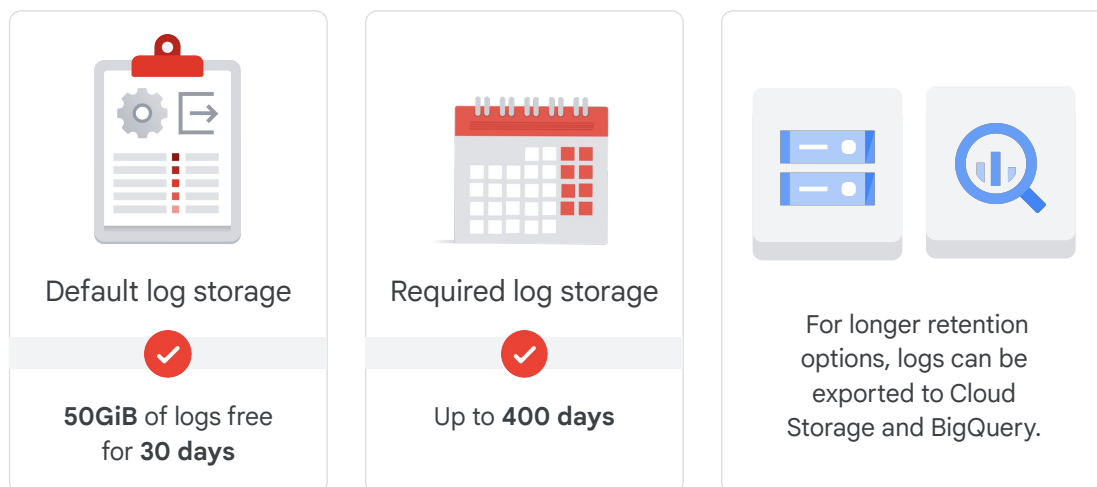
The first step to inspecting a log is to view it, for which there are two options.

The first option is to use the **kubectl** command to display logs directly from a Pod. This option is quick, but it's worth noting that logs aren't saved to any database and may be lost if containers are restarted or deleted.

The second option is to use Cloud Logging, which provides a single interface to review logs from containers, nodes, and other system services. This allows more options to broaden your visibility of GKE events to help correlate issues.



## Cloud Logging storage and retention periods



Google Cloud

Cloud Logging is a paid-for product, but offers a free tier that stores the first 50 gibibytes of log data for free for 30 days. This storage is in a bucket named `_Default`, which stores Data Access audit logs and Policy Denied audit logs.

Admin audit logs are held for 400 days in a storage bucket named `_Required`, which stores logs including Admin Activity audit logs, System Event audit logs, and Login Audit logs

For longer retention options, **user-defined log buckets** can be created, and can have a retention period of up to 3650 days.

Logs can also be exported to BigQuery.

# Google Kubernetes Engine Logging and Monitoring

- 01 Google Cloud Observability
- 02 Cloud Logging
- 03 Inspecting logs with the kubectl command**
- 04 Inspecting logs with Cloud Logging and Logging Agents
- 05 Cloud Monitoring

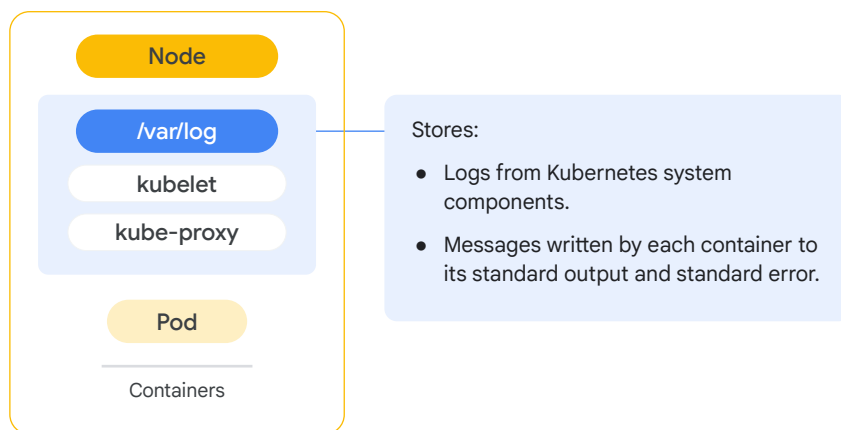


# Inspecting Logs with kubectl



Let's explore how to inspect Kubernetes logs with the **kubectl** command.

## Kubernetes system logs in /var/log



Regarding Kubernetes-native logging, logs from Kubernetes system components, such as **kubelet** and **kube-proxy**, are stored in the `/var/log` directory of each node's file system.

Messages written by each container to its *standard output* and *standard error* are also logged in the `/var/log` directory. This allows log messages to be captured and centrally managed.

# Producing logs with kubectl

```
$ kubectl logs [POD_NAME]
```

- Used to view the logs of a specific Pod.
- Run the `kubectl get pods` command to get a list of all the available Pods.

```
$ kubectl --tail=20 [POD_NAME]
```

- Used to limit the number of logs returned.
- By adding `-tail=20`, the output would be limited to the 20 most recent lines.

Let's explore some common **kubectl** commands that can be used to produce GKE logs.

To view the logs of a specific Pod, run the **kubectl logs** command, followed by the name of the Pod.

If you don't know the exact name of the Pod, or if the Pod is part of a Deployment, you can run the **kubectl get pods** command to get a list of all the available Pods.

When troubleshooting an issue, you might only need to inspect only the most recent log from a specific Pod or container. To limit the number of logs returned, add the **--tail** option to the **kubectl logs** command. By adding **-tail=20**, the output would be limited to the 20 most recent lines.

## Producing logs with kubectl

```
$ kubectl logs --since=3h [POD_NAME]
```

- Used to restrict the output based on time.
- `--since=3h` would return all logs for specific Pod or container from the last 3 hours.

```
$ kubectl logs --previous [POD_NAME]
```

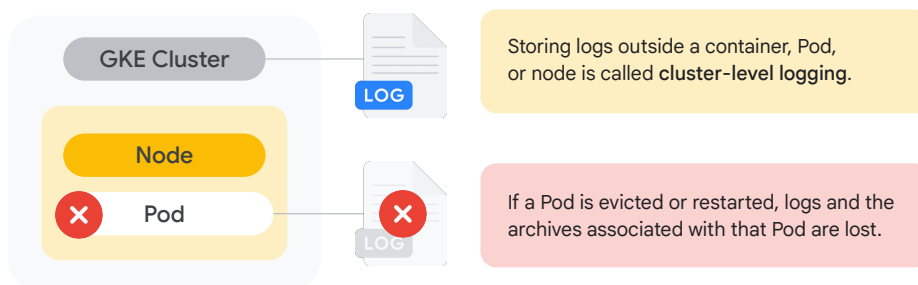
- Used to view a previous instantiation of a container.
- If your Pod has multiple containers, specify which container's log the `kubectl` command should return.

But there might be instances when you don't actually know how many log messages to retrieve, but you do know the problem occurred in the last few hours. To restrict the output based on time, you can add the `--since` option, followed by the time period. For example, `--since=3h` would return all logs for specific Pod or container from the last 3 hours.

Now let's say you need to view a previous instantiation of a container *before* it crashed. This can be accomplished by adding the `--previous` option.

If your Pod has multiple containers, you'll need to specify which container's log the `kubectl` command should return.

## Native Kubernetes Logs are not persistent

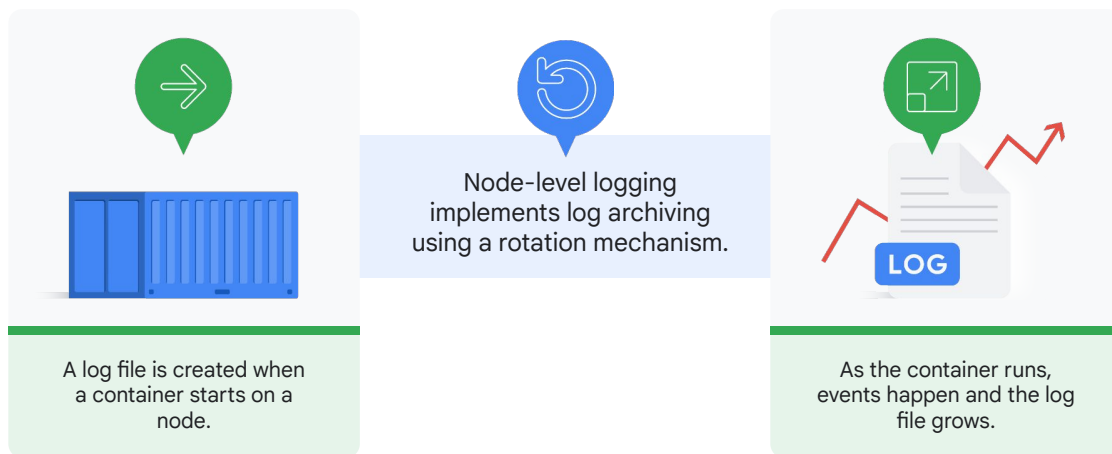


In GKE, log storage is handled by the integration with Cloud Logging.

These native Kubernetes logs, and the compressed archives aren't persistent. If, for any reason, a Pod is evicted or restarted, logs and the archives associated with that Pod are lost.

As a result, there's a need to store logs outside a container, Pod, or node. This is called cluster-level logging. As mentioned earlier, Kubernetes itself doesn't offer any log storage solution, but it does support various implementations. In GKE this is handled for you by the integration with Cloud Logging.

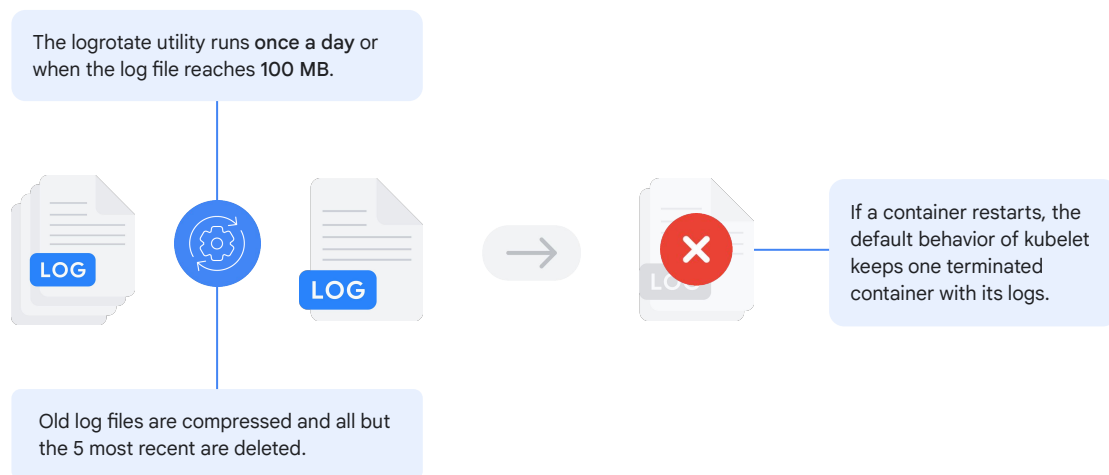
## Creating a log file



When a container starts on a node, a log file is created. Node-level logging implements log archiving using a rotation mechanism. As the container runs, events happen and the log file grows.

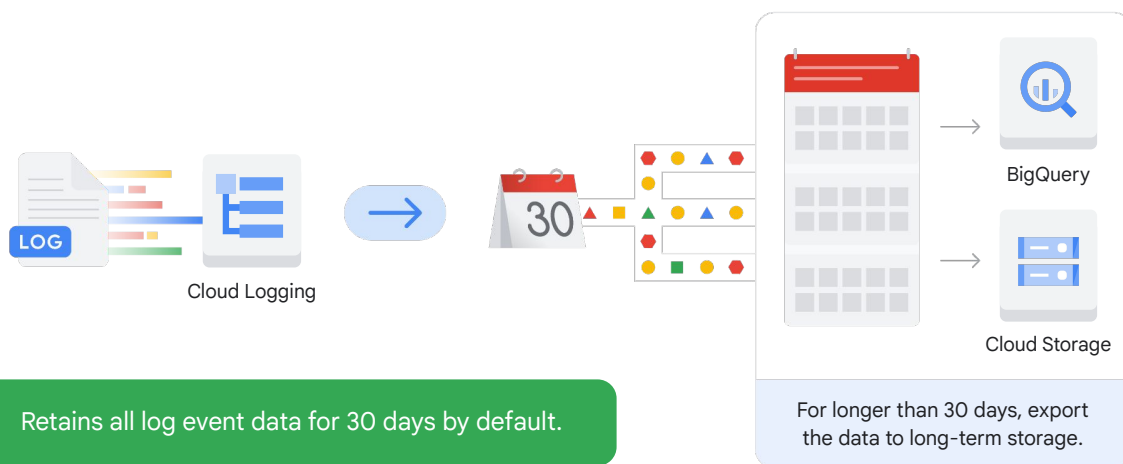


## The logrotate utility



Either once per day, or when the log file reaches 100 MB (whichever comes first) the logrotate utility creates a new log and compresses the old log file, saving it into an archive. It then deletes all but the 5 most recent compressed log archives. This ensures that logs don't consume all of the available storage on the node. If a container restarts, the default behavior of kubelet keeps one terminated container with its logs.

## All log events are streamed to Cloud Logging



However, all log events are streamed to Cloud Logging, which retains all log event data in the **\_default** bucket for 30 days by default. If your organization requires you to retain log data for longer than 30 days, configure Cloud Logging to export the data to long-term storage such as BigQuery or Cloud Storage.

# Google Kubernetes Engine Logging and Monitoring

- 01 Google Cloud Observability
- 02 Cloud Logging
- 03 Inspecting logs with the kubectl command
- 04 Inspecting logs with Cloud Logging and Logging Agents**
- 05 Cloud Monitoring



# Cloud Logging

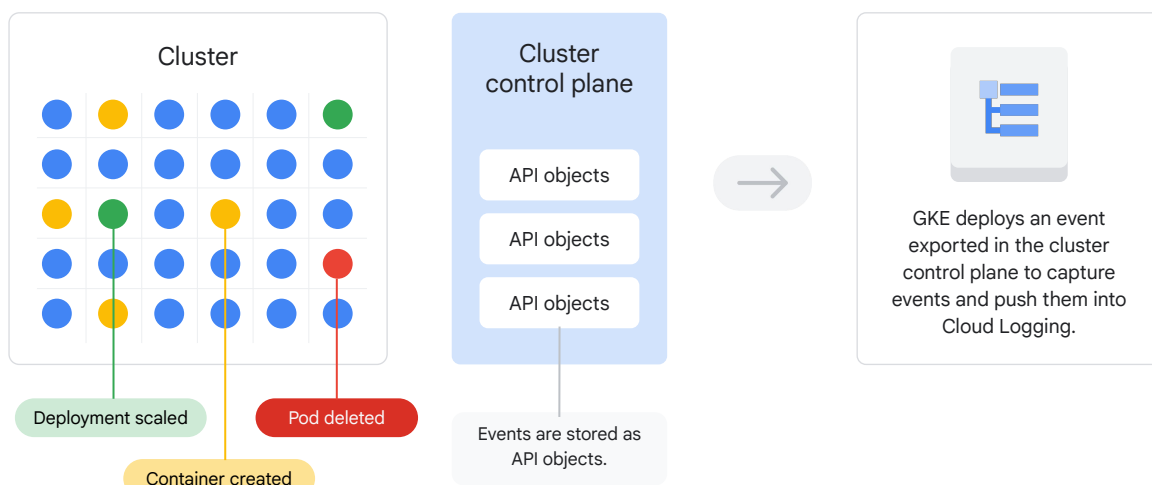


Cloud Logging

- Stores, searches, analyzes, monitors, and alerts on logging data and events from Google Cloud.
- Designed to automatically scale and ingest terabytes of log data per second.

Cloud Logging is a fully managed service that stores, searches, analyzes, monitors, and alerts on logging data and events from Google Cloud. It's designed to automatically scale and ingest terabytes of log data per second.

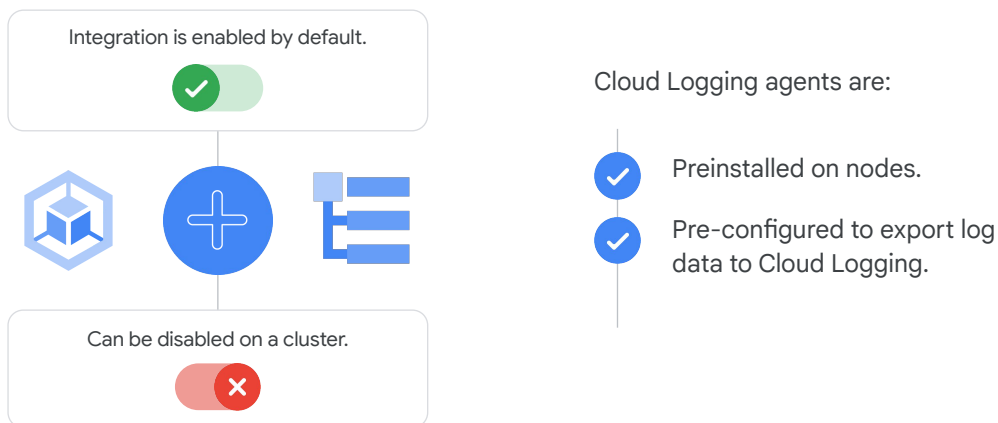
## Events are stored as API objects



An event describes an operation that takes place in a cluster. Some examples include deleting a Pod, scaling a Deployment, or creating a container.

Events are stored as API objects on the cluster control plan, and because events are only stored temporarily in Kubernetes, GKE deploys an event exported in the cluster control plane to capture events and push them into Cloud Logging.

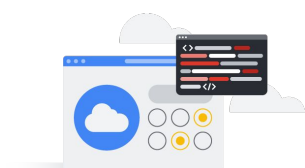
## GKE integration with Cloud Logging



GKE integration with Cloud Logging is enabled by default, though it can be disabled on a cluster if required.

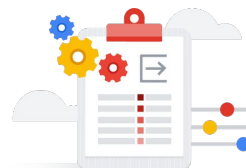
Cloud Logging agents are preinstalled on nodes and pre-configured to export log data to Cloud Logging.

## Custom log or filter logs



### Cloud Logging API

Write a custom log and push it to Cloud Logging



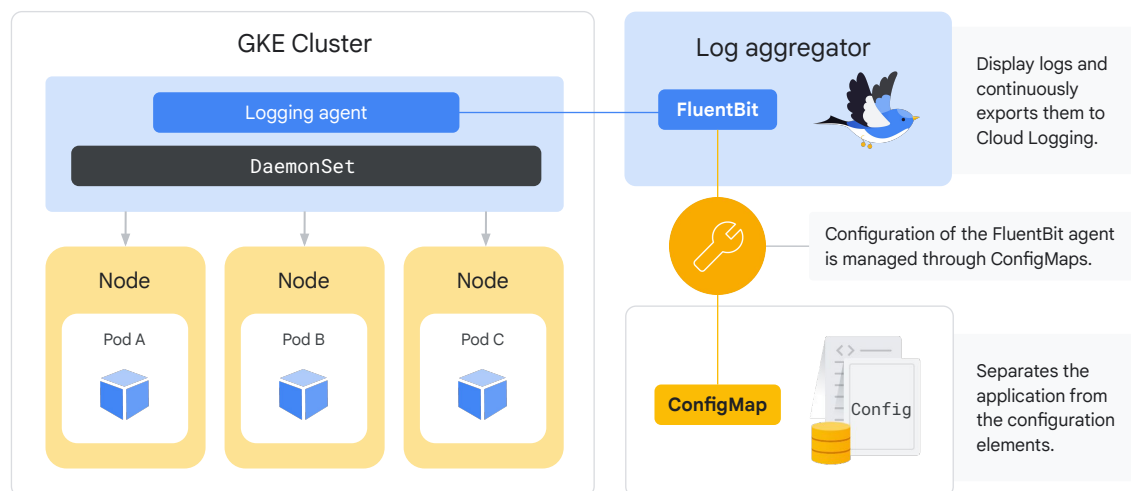
### Filter logs using Cloud filter language

- ☒ Logs Viewer console
- ☒ Cloud Logging API

Cloud Logging also offers an API that can be used to write a custom log and push it to Cloud Logging.

Additionally, logs can be filtered by using Cloud filter language, either in the Logs Viewer console or direction through the Cloud Logging API.

# FluentBit



Google Cloud

GKE installs a logging agent on every node of a cluster, and this agent collects and exports container logs and system component logs to the Cloud Logging backend.

The node logging agent used by Cloud Logging is FluentBit. FluentBit is a log aggregator that can display logs, adding helpful metadata, and then continuously export those logs to Cloud Logging.

FluentBit uses DaemonSet, as that ensures that every node in a cluster runs a copy of a specific Pod.

The configuration of the FluentBit agent is managed through ConfigMaps. This increases the scalability of the implementation by separating the application (the FluentBit DaemonSet) from the configuration elements (the ConfigMap).

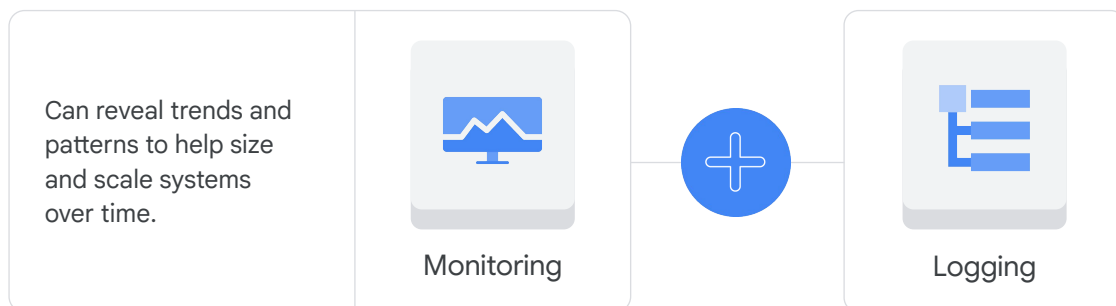


# Google Kubernetes Engine Logging and Monitoring

- 01 Google Cloud Observability
- 02 Cloud Logging
- 03 Inspecting logs with the kubectl command
- 04 Inspecting logs with Cloud Logging and Logging Agents
- 05 Cloud Monitoring**



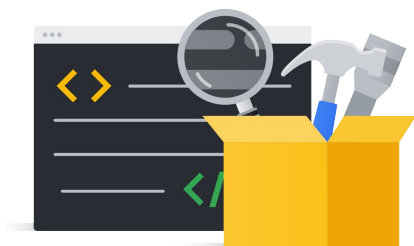
## Why does monitoring matter?



Monitoring and logging are often coupled to provide a complete picture of the system status and past trends.

Monitoring and logging are often coupled to provide a complete picture of the system status and past trends. Monitoring can also reveal trends and patterns that can help size and scale systems over time.

# Monitoring Kubernetes



Monitor your application's current state and anything that might be impeding it.

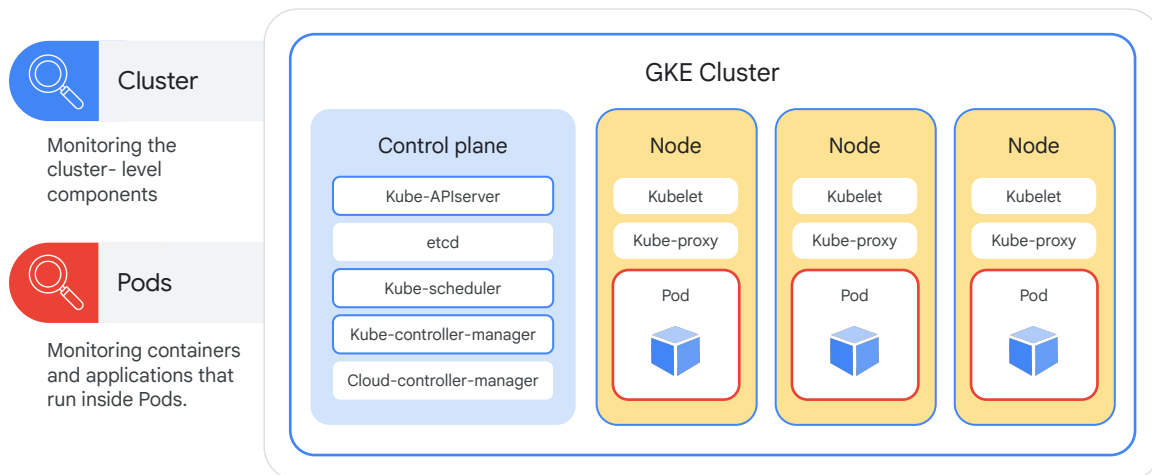


Identify performance bottlenecks by accurately monitoring the state of services like the throughput and latency.

When it comes to monitoring Kubernetes, it's less about watching the clusters and nodes, and more about monitoring your application's current state—and anything that might be impeding it.

If you're able to accurately monitor the state of these services, including the throughput and latency, you're more likely to identify performance bottlenecks. For example, through aggregated logging and debugging, you can diagnose application code issues.

# Kubernetes monitoring falls into two domains



Google Cloud

In Kubernetes, monitoring can be broken into two domains.

One domain is the **cluster**, which involves monitoring the cluster-level components like individual nodes, kube-APIserver, and kube-controller-manager.

The second monitoring domain is the **Pods**, including the containers and the applications that run inside them.

# Monitoring the cluster



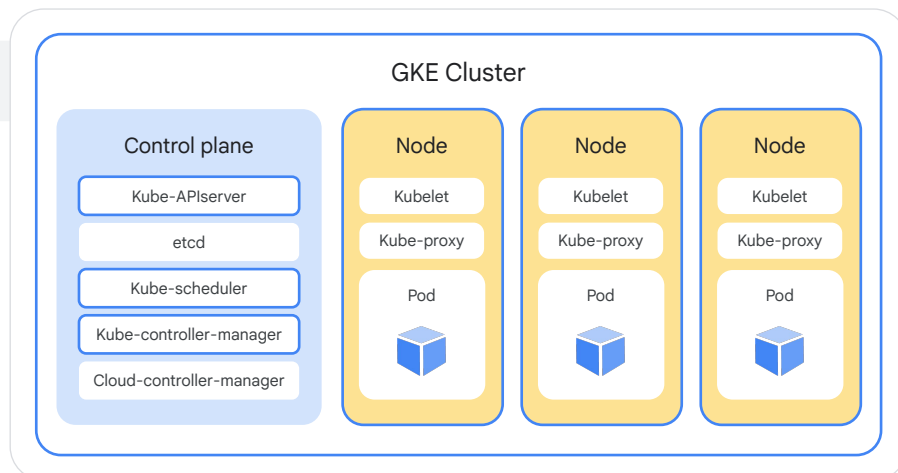
## Cluster

Cluster monitoring refers to the cluster services, nodes, and other infrastructure elements.

This can be accomplished using:

Cloud Console


Cloud Monitoring



Cluster monitoring refers to the cluster services, nodes, and other infrastructure elements.

This can be accomplished using the monitoring in the Cloud Console, or through Cloud Monitoring, using health checks and dashboards.


## Events and metrics



**Metrics**

A value that can be monitored, such as CPU or disk usage.

**Metrics** report numerical values.



**Events**

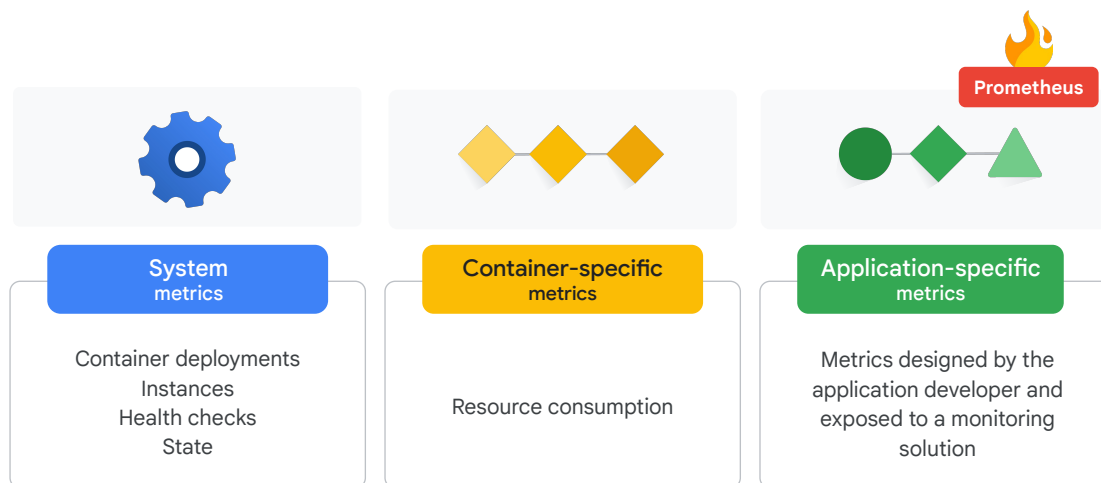
Represent occurrences that happen to a cluster, Pod, or container.

**Events** report successes, warnings, or failures.

Let's refer back to two important terms that were mentioned when introducing Cloud Monitoring, **events** and **metrics**, and take a moment to differentiate between the two.

- A **metric** represents a value that can be monitored, such as CPU or disk usage. They might be "gauge values," which fluctuate up or down over time, or "counters," which are values that increase over time. Monitoring metrics can help identify bottlenecks, as they're specific to a resource.
- And then there are **events**, which represent occurrences that happen to a cluster, Pod, or container. Examples include the restart of a Pod, node, or service; when the number of deployments in a cluster is scaled up or down; or when an application responds to a request. Events typically report successes, warnings, or failures, while metrics report numerical values.

## Pod monitoring metric sub-categories



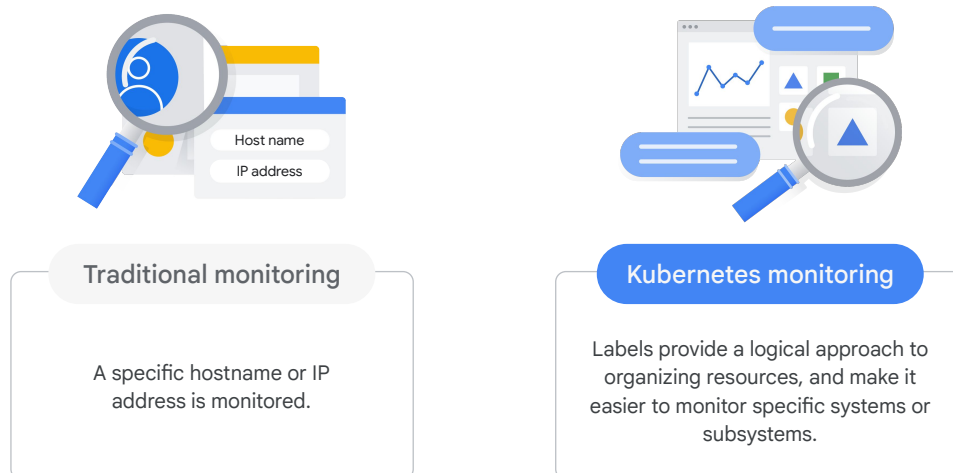
Pod Monitoring can be divided into several sub-categories.

**System metrics** include container deployments, instances, health checks, and state.

**Container-specific metrics** such as the resource consumption.

And **application-specific metrics**, which are designed by the application developer and exposed to a monitoring solution. This is an ideal use case for Google Cloud Managed Service for Prometheus. Users can export their metrics as Prometheus metrics to access the advanced monitoring features of Prometheus.

# Kubernetes monitoring requires a different approach



Google Cloud

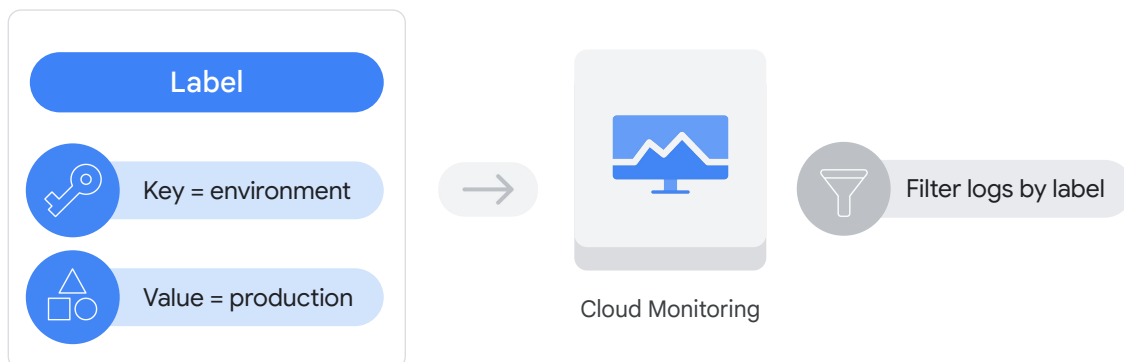
Unlike traditional server monitoring, where you specify a hostname to monitor, the abstraction layers in Kubernetes—well, containers in general—require a different approach.

Instead of having a specific hostname or IP address to be monitored, all resources in Kubernetes are labeled.

These labels provide a logical approach to organizing resources, and also make it easier to monitor specific systems or subsystems by selecting a combination of different labels.



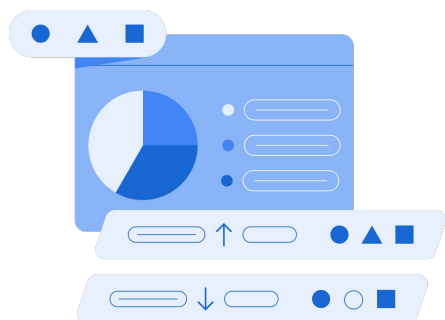
## Cloud Monitoring can filter log-based metrics by label



In Cloud Monitoring, as well as other tools, you can filter logs by label.

A Kubernetes label consists of a key and a value. This means that if you apply a label with the key defined as “environment” and the value defined as “production” to all the components of your production environment, you can use that label in Cloud Monitoring.

## Dashboards provide cluster health insights



Key features include:

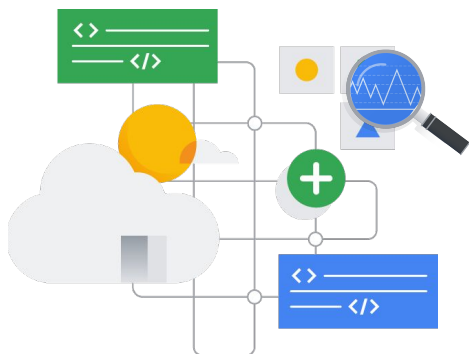
- ✓ Uptime checks to monitor cluster resources, control plane metrics, and Google Cloud resources.
- ✓ Integration with Cloud Logging to leverage custom metrics and monitor your logging-based custom metrics.
- ✓ Custom metrics for alerting policies, which can improve the signal-to-noise ratio of your alert messages.

Cloud Monitoring provides visibility into the overall health of your GKE cluster and the Pods it contains through custom dashboards.

Key features include:

- The ability to use uptime checks to monitor the availability of Kubernetes resources within a cluster as well as control plane metrics, along with other Google Cloud resources that are being used.
- Integration with Cloud Logging to leverage custom metrics and even monitor your logging-based custom metrics.
- The ability to use custom metrics to create alerting policies, which can improve the signal-to-noise ratio of your alert messages.

# Cloud Monitoring is optimized for Kubernetes



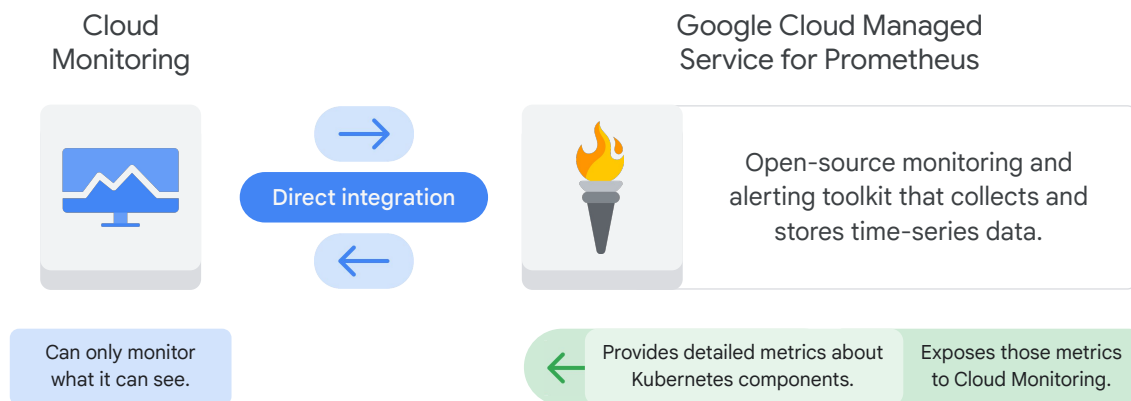
- ✓ View metrics, logs, traces, events, and metadata in one place.
- ✓ Analyze infrastructure metrics, application metrics, or logs from a single dashboard.
- ✓ Supports multi-cluster monitoring within the Google Cloud environment, other cloud environments, and on-premises Kubernetes environments.
- ✓ Offers extended direct support to Prometheus.

Cloud Monitoring is optimized for Kubernetes, because it lets you observe your entire Kubernetes environment to see metrics, logs, traces, events, and metadata in one place.

Imagine that your application has an error. With Cloud Monitoring, you can analyze infrastructure metrics, application metrics, or even logs directly from a single dashboard.

It supports multi-cluster monitoring within the Google Cloud environment, along with other cloud environments and on-premises Kubernetes environments. And it offers extended direct support to Prometheus.

# Cloud Monitoring + Prometheus



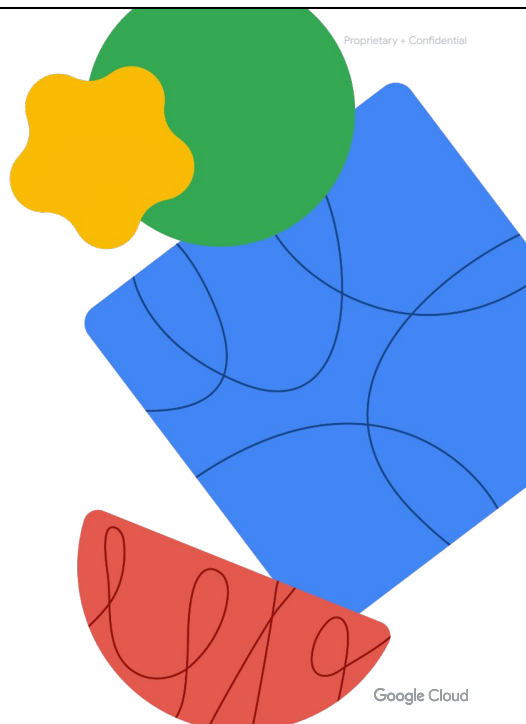
Cloud Monitoring also provides direct integration to **Google Cloud Managed Service for Prometheus**, which is an open-source monitoring and alerting toolkit that collects and stores time-series data.

Cloud Monitoring can only monitor what it can see. So, to extract more information from Cloud Monitoring, you need to add more information. That's where Prometheus can help.

Prometheus can provide detailed metrics about Kubernetes components, including metrics from within the applications running in your Pods, and then expose those metrics to Cloud Monitoring, which provides you with much more granular detail than Cloud Monitoring alone.

# Quiz questions

Let's pause for a quick check in.



# Quiz | Question 1

## Question

You are a systems administrator tasked with monitoring and maintaining a cluster of Google Kubernetes Engine (GKE) instances. Which tool would you use to collect and store logs from your GKE cluster?

- A. Cloud Logging
- B. Google Kubernetes Engine
- C. Error Reporting
- D. Kubernetes

## Quiz | Question 1

### Answer

You are a systems administrator tasked with monitoring and maintaining a cluster of Google Kubernetes Engine (GKE) instances. Which tool would you use to collect and store logs from your GKE cluster?

- A. Cloud Logging
- B. Google Kubernetes Engine
- C. Error Reporting
- D. Kubernetes



You are a systems administrator tasked with monitoring and maintaining a cluster of Google Kubernetes Engine (GKE) instances. Which tool would you use to collect and store logs from your GKE cluster?

- A. Cloud Logging: This is the **correct answer** because **Cloud Logging** is a fully managed service designed to collect, store, and analyze logs from various sources, including GKE clusters..
- B. Google Kubernetes Engine: This is the **incorrect answer** because **GKE** doesn't have built-in long-term storage or advanced analysis features..
- C. Error Reporting: This is the **incorrect answer** because **Error Reporting** focuses on aggregating and analyzing errors within applications, not general log collection and storage..
- D. Kubernetes This is the **incorrect answer** because **Kubernetes** is the container orchestration system itself and doesn't provide a native solution for log storage..

## Quiz | Question 2

### Question

You need to monitor specific applications running on your production GKE Clusters. How can you create a logical structure for your application that allows you to selectively monitor the application's components using Cloud Monitoring? Choose all responses that are correct (2 correct responses).



- A. Filter the Cloud Monitoring logs using the application name.
- B. Filter the Cloud Monitoring logs using Kubernetes labels.
- C. Add a prefix to all Pod names that identify the application.
- D. Add Labels to the Pods in Kubernetes that identify the applications.
- E. Filter the Cloud Monitoring logs using the application prefix.



## Quiz | Question 2

### Answer

You need to monitor specific applications running on your production GKE Clusters. How can you create a logical structure for your application that allows you to selectively monitor the application's components using Cloud Monitoring? Choose all responses that are correct (2 correct responses).

- A. Filter the Cloud Monitoring logs using the application name.
- B. Filter the Cloud Monitoring logs using Kubernetes labels. 
- C. Add a prefix to all Pod names that identify the application.
- D. Add Labels to the Pods in Kubernetes that identify the applications. 
- E. Filter the Cloud Monitoring logs using the application prefix.

Google Cloud

You need to monitor specific applications running on your production GKE Clusters. How can you create a logical structure for your application that allows you to selectively monitor the application's components using Cloud Monitoring? Choose all responses that are correct (2 correct responses).

A. Filter the Cloud Monitoring logs using the application name.: This is the **incorrect answer** because this is less flexible than using labels. Application names might not be unique or granular enough to isolate specific components..

**B. Filter the Cloud Monitoring logs using Kubernetes labels.**: This is the **correct answer** because Kubernetes labels are key-value pairs that can be attached to objects like Pods. Cloud Monitoring can filter logs based on these labels, allowing you to isolate logs for specific applications or components..

C. Add a prefix to all Pod names that identify the application.: This is the **incorrect answer** because prefixes are less flexible than labels. They also clutter Pod names and might not be easily searchable in Cloud Monitoring..

**D. Add Labels to the Pods in Kubernetes that identify the applications.**: This is the **correct answer** because this is how you would implement the filtering described in option B. By adding specific labels to your Pods (e.g., "app=myapp", "env=prod"), you create a logical structure that Cloud Monitoring can use to filter and monitor your applications.

E. Filter the Cloud Monitoring logs using the application prefix.: This is the **incorrect answer** because this depends on whether Cloud Monitoring can effectively parse and filter based on Pod name prefixes, which is not as reliable as using labels..

## Quiz | Question 3

### Question

You are troubleshooting an issue which happened in the last hour. You enter the command 'kubectl logs --since=3h demo-pod'. However, the events you are looking for do not appear in the output. What is the likely cause?

- A. The file has been archived to Cloud Logging and is no longer available locally.
- B. The log file contains more than 3 hours of data, and it has been archived.
- C. The log file is older than 2 hours and is not included in the results.
- D. The log file was greater than 100MB in size, and it has been rotated.

## Quiz | Question 3

### Answer

You are troubleshooting an issue which happened in the last hour. You enter the command 'kubectl logs --since=3h demo-pod'. However, the events you are looking for do not appear in the output. What is the likely cause?

- A. The file has been archived to Cloud Logging and is no longer available locally.
- B. The log file contains more than 3 hours of data, and it has been archived.
- C. The log file is older than 2 hours and is not included in the results.
- D. The log file was greater than 100MB in size, and it has been rotated.



You are troubleshooting an issue which happened in the last hour. You enter the command 'kubectl logs --since=3h demo-pod'. However, the events you are looking for do not appear in the output. What is the likely cause?

- A. The file has been archived to Cloud Logging and is no longer available locally.: This is the **incorrect answer** because archiving to Cloud Logging doesn't remove logs from the node, and can still access them..
- B. The log file contains more than 3 hours of data, and it has been archived.: This is the **incorrect answer** because this wouldn't prevent the last hour of logs from showing..
- C. The log file is older than 2 hours and is not included in the results.: This is the **incorrect answer** because xx.
- D. **The log file was greater than 100MB in size, and it has been rotated.:** This is the **correct answer** because Log rotation creates a new file.

## Quiz | Question 4

### Question

Which of the following is a logging agent used by GKE?

- A. DaemonSet
- B. FluentBit
- C. Cloud Logging
- D. Kubernetes

## Quiz | Question 4

### Answer

Which of the following is a logging agent used by GKE?

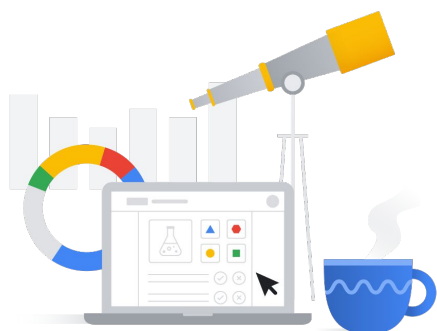
- A. DaemonSet
- B. FluentBit
- C. Cloud Logging
- D. Kubernetes



Which of the following is a logging agent used by GKE?

- A. DaemonSet: This is the **incorrect answer** because a **DaemonSet** ensures that all (or some) Nodes run a copy of a Pod..
- B. **FluentBit**: This is the **correct answer** because **FluentBit** is the specific logging agent that GKE deploys on each node to collect and forward logs to Cloud Logging..
- C. **Cloud Logging**: This is the **incorrect answer** because **Cloud Logging** is a cloud based service designed to collect, store, and analyze logs from various sources, not a logging agent.
- D. Kubernetes: This is the **incorrect answer** because **Kubernetes** is the container orchestration platform itself, not a logging agent.

## Lab: Configuring GKE-Native Monitoring and Logging



01 Use Kubernetes Engine Monitoring to view cluster and workload metrics.

02 Use Cloud Monitoring alerting to receive notifications about the cluster's health.

It's time for some hands-on practice with GKE.

In the lab titled "Configuring GKE-Native Monitoring and Logging", you'll:

- Use Kubernetes Engine Monitoring to view cluster and workload metrics
- Use Cloud Monitoring Alerting to receive notifications about the cluster's health