Welcome to the “Google Kubernetes Engine Logging and Monitoring” module. In this module you will learn how logging is implemented in Kubernetes, and how GKE extends that basic functionality using Google Cloud’s operations suite, a set of multi-cloud resource reconnaissance tools provided by Google that includes monitoring, logging, and debugging for your applications and infrastructure.

Learn how to ...

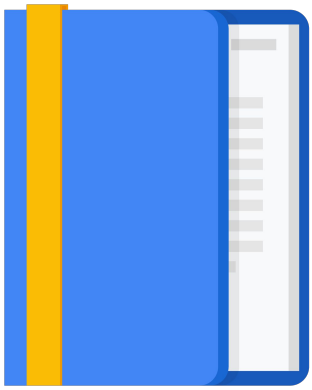
Use Google Cloud's operations suite to monitor and manage the availability and performance.

Locate and inspect Kubernetes logs.

Perform forensic analysis of logs.

Monitor performance.

Create probes for wellness checks on live applications.



Google Cloud

In this module, you'll learn how to use Google Cloud's operations suite to monitor and manage the availability and performance of your Google Cloud resources and applications that are built with those resources; locate and inspect Kubernetes logs produced by resources inside your GKE clusters; use Cloud Logging and BigQuery for longer term retention and forensic analysis of the logs that are produced by GKE and the Kubernetes resources inside your GKE clusters; how to monitor system and application performance from different vantage points and; how to create probes for wellness checks on the applications you're running.

Agenda

Google Cloud's Operations Suite

Logging

Monitoring

Lab: Configuring GKE-Native Monitoring and Logging

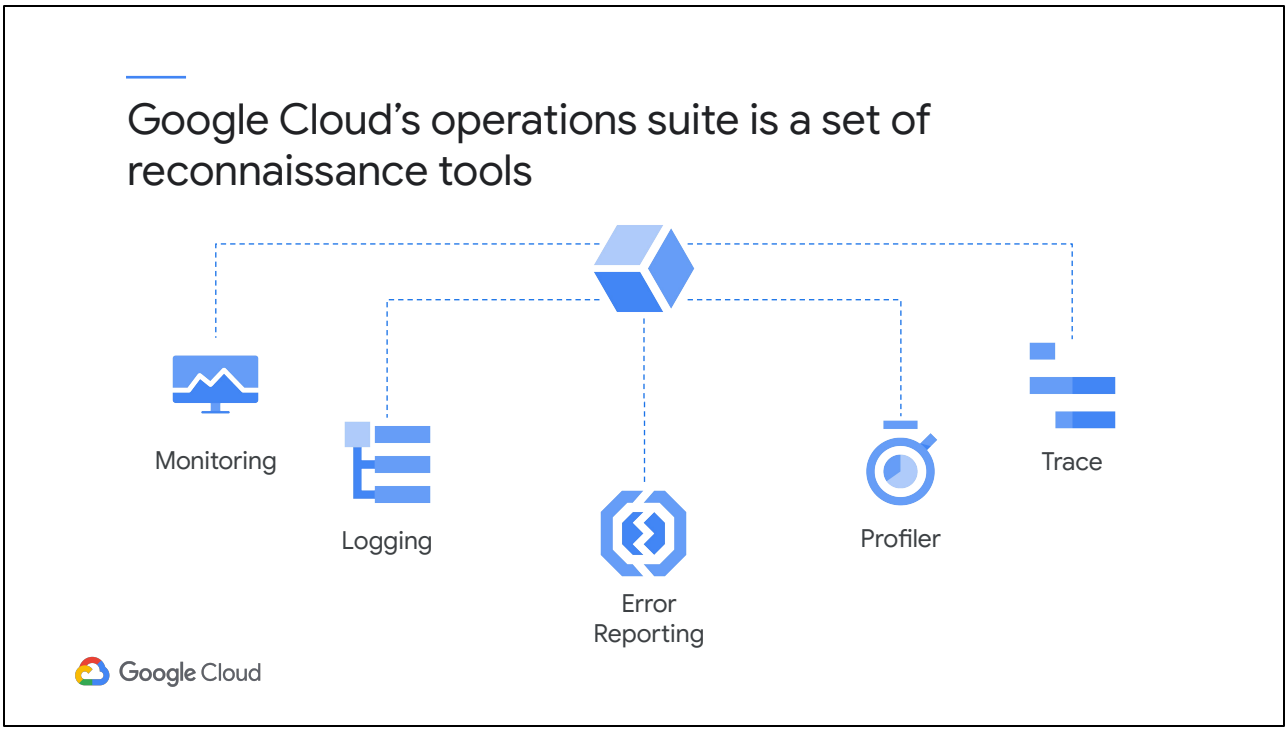
Probes

Lab: Configuring Liveness and Readiness Probes

Summary

Google Cloud

Let's start by introducing Google Cloud's operations suite. In this lesson, I'll introduce its key features and describe how you use them to monitor and troubleshoot Google Kubernetes Engine clusters and applications.



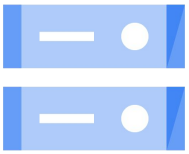
Google Cloud's operations suite is a set of reconnaissance tools that includes monitoring, logging, error reporting, tracing and profiling for your applications and infrastructure. It provides a single "pane of glass" through which you can view your infrastructure's health and manage alerts.

Why use Cloud Logging for logging? (1/2)

Log repository/aggregator

- Platforms
- Systems
- Applications

BigQuery



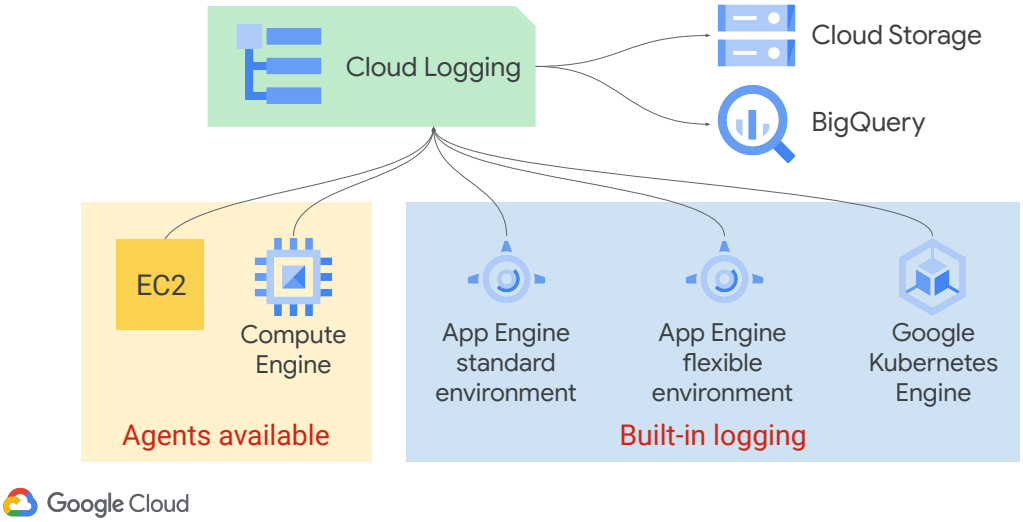
Cloud Storage

Google Cloud

Cloud Logging can serve as the log repository and aggregator for your platforms, systems, and applications. The logs can be searched with the power of BigQuery, which simplifies some of the most complex searches.

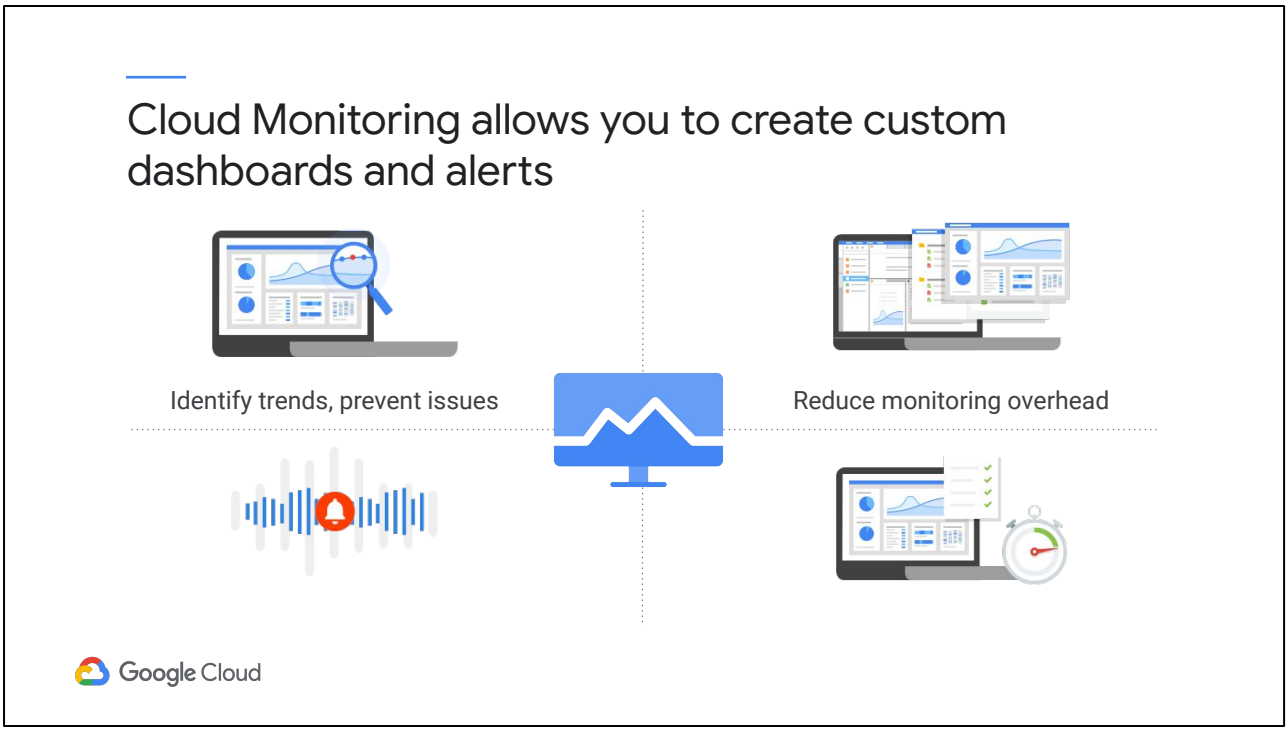
Additionally, you can move log data out of Cloud Logging and into Google Cloud storage services, namely Cloud Storage or BigQuery, for long-term archive and reference.

Why use Cloud Logging for logging? (2/2)



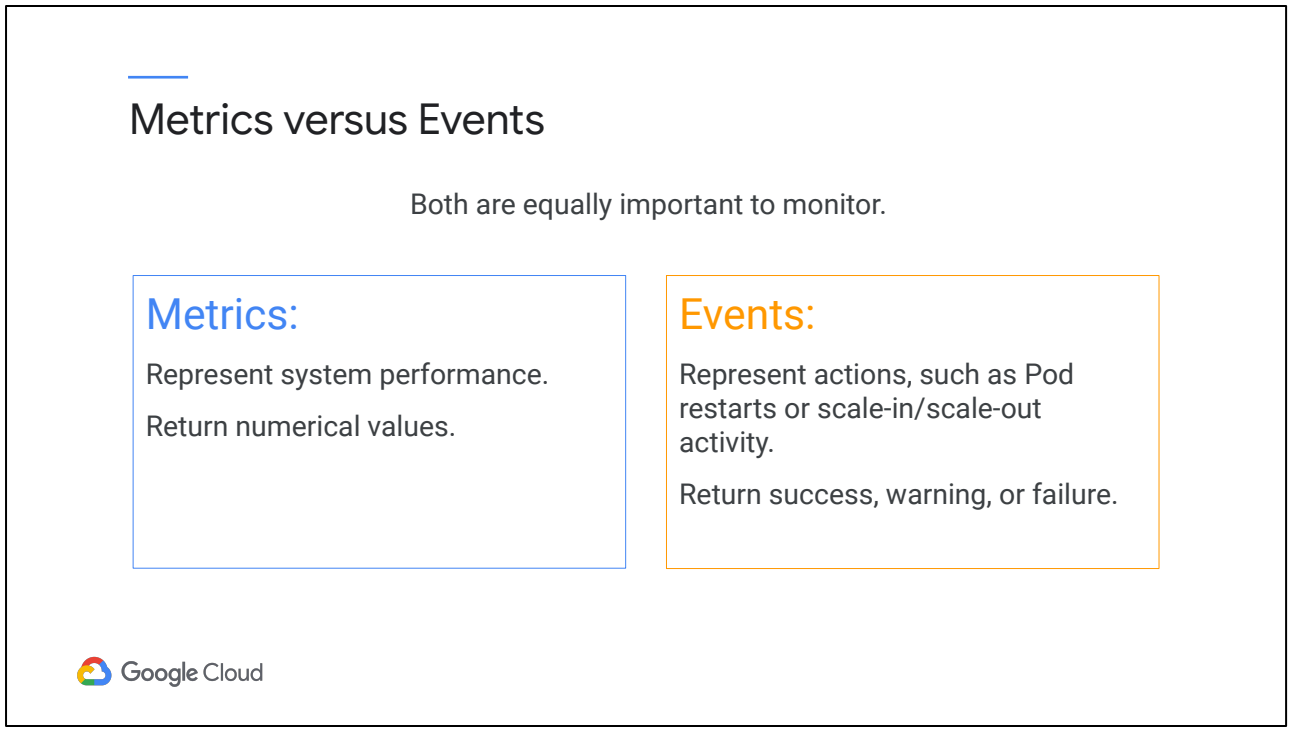
Google Cloud

Cloud Logging is the native logging solution for many Google Cloud products and services. It offers agent-based installation software for Compute Engine instances and Amazon EC2 instances.



The Cloud Monitoring engine allows you to create custom dashboards and alerts with metrics that represent the health of your application. You can customize metrics to gain further insight into your application's health, load, or behavior.

Cloud Monitoring's dashboards and graphs provide an at-a-glance method to ensure that systems are operating properly. More importantly, graphs give you a visual correlation cue for events, such as performance bottlenecks or service dependencies. You can also integrate Cloud Monitoring with many third-party products.



Events are different from metrics, although they are closely related.

A metric is a value that you can monitor, such as CPU or disk usage. These can be values that change up or down over time, called “gauge values,” or values that increase over time, called “counters.”

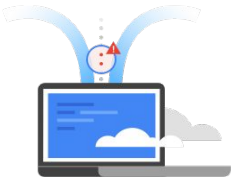
Events are things that happen to the cluster, Pod, or container. Here are a few examples of events: the restart of a Pod, node, or Service; scaling up or down of the number of Deployments in the cluster; or an application responding to a request.

Events typically report success, warning, or failure; while metrics report numerical values.

It’s important that you monitor metrics because they’re specific to a resource and can help you identify bottlenecks.

Events are important because they might include errors or warnings that appear when the cluster attempts to scale in or out.

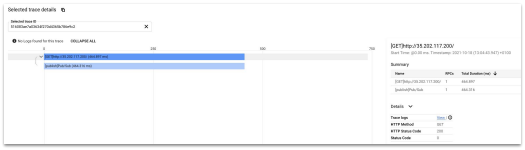
Cloud Trace provides latency reporting in near real-time



Find performance bottlenecks

Fast, automatic issue detection

Broad platform support



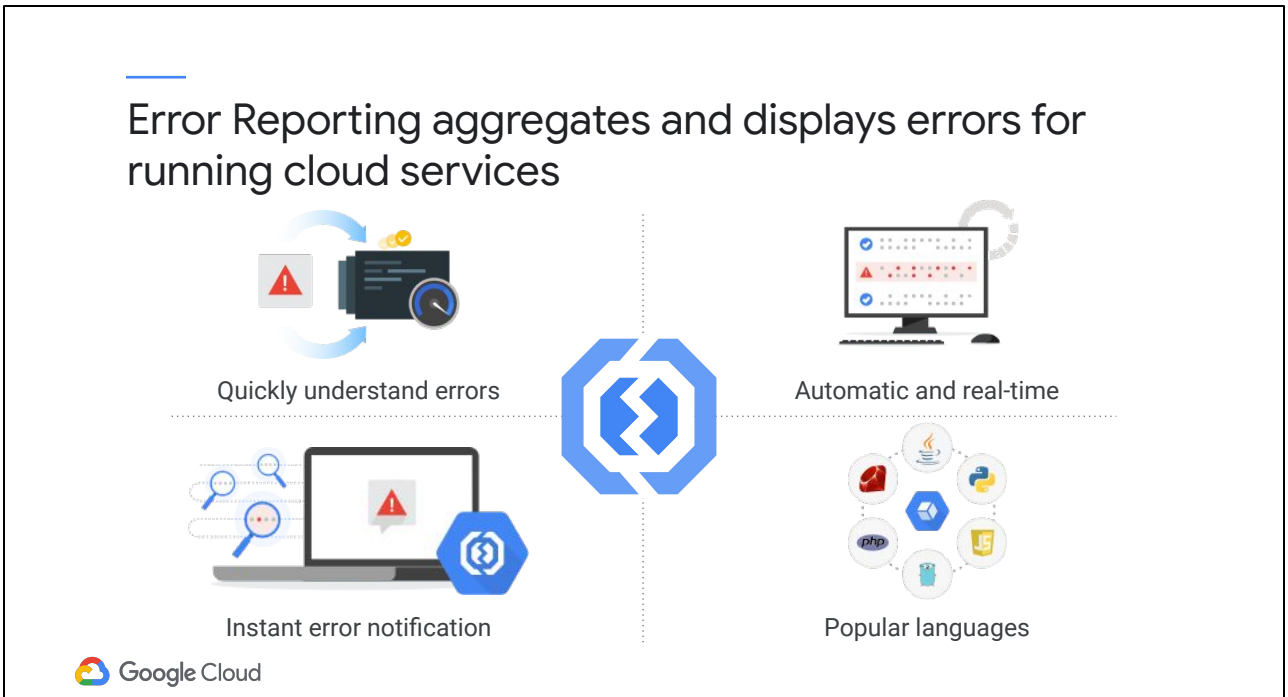
Stacked line charts

Google Cloud

Google Cloud's operations suite goes beyond simple monitoring and logging to provide deeper insight into the errors and performance issues that can delay or hinder application development.

Application Performance Management (APM) includes tools that provide you with easy access to debugging and code troubleshooting.

Cloud Trace allows you to quantify the true latency for each of the requests made by the components in your application, and the stacked line charts can help you visually detect bottlenecks.



Error Reporting notices when an error occurs within an application, captures the error information and call stack, and then provides that information to you for debugging. This provides more information for troubleshooting than is captured by Cloud Logging alone.

Cloud Profiler continually analyzes the performance of your code as it executes in production

Low-impact production profiling

Broad platform support

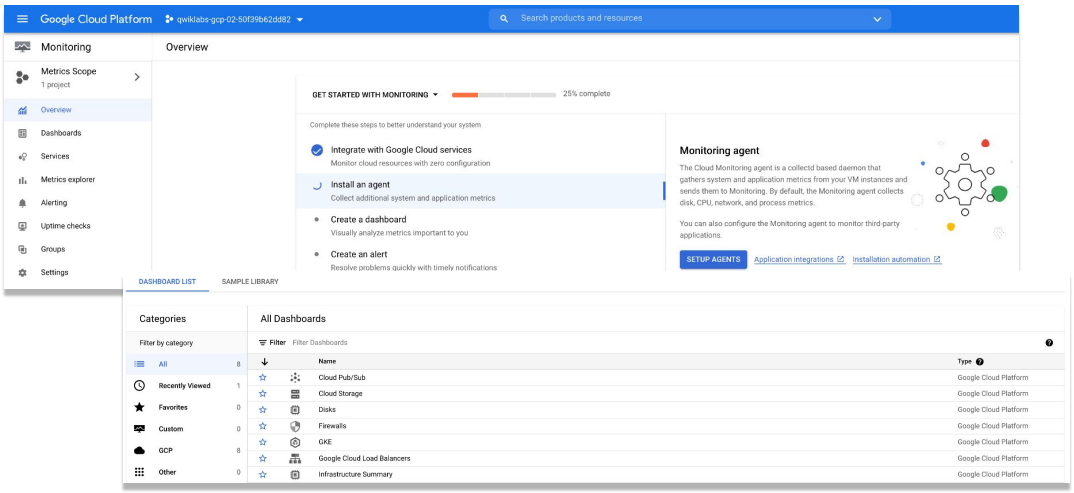
Google Cloud

Poorly performing code increases the latency and cost of applications and web services every day. Cloud Profiler continuously analyzes the performance of CPU or memory-intensive functions executed across an application.

While it's possible to measure code performance in development environments, the results generally don't map well to what's happening in production. Many production profiling techniques either slow down code execution or can only inspect a small subset of a codebase. Profiler uses statistical techniques and extremely low-impact instrumentation that runs across all production application instances to provide a complete picture of an application's performance without slowing it down.

Profiler allows developers to analyze applications running anywhere, including Google Cloud, other cloud platforms, or on-premises, with support for Java, Go, Node.js, and Python.

How do you start?

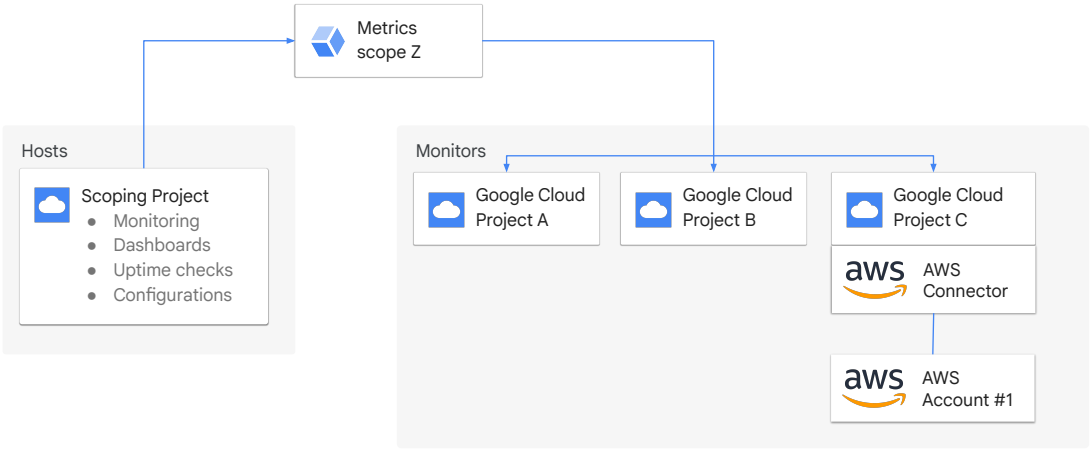


Google Cloud

Monitoring is the first tool listed in the Cloud Console under Operations. The landing page allows you to add Google Cloud Projects, install agents, and create uptime checks, alerting policies, and custom dashboards. Alternatively, the Debugger, Trace, Logging, Error Reporting, and Profiler tools are easily accessible.

Free usage allotments let you get started with no up-front fees or commitments. You control your own usage and spending, paying only for what you use.

Use Metrics Scope to organize monitoring information



A metrics scope is the root entity that holds monitoring and configuration information in Cloud Monitoring. Each metrics scope can have between 1 and 375 monitored projects. Now, monitoring data for all projects in that scope will be visible.

A metrics scope contains the custom dashboards, alerting policies, uptime checks, notification channels, and group definitions that you use with your monitored projects. A metrics scope can access metric data from its monitored projects, but the metrics data and log entries remain in the individual projects.

The first monitored Google Cloud project in a metrics scope is called the scoping project. The name of that project becomes the name of your metrics scope. To access an AWS account, you must configure a project in Google Cloud to hold the AWS Connector.

https://cloud.google.com/monitoring/settings#concept-scope

Agenda

Google Cloud's Operations Suite

Logging

Monitoring

Lab: Configuring GKE-Native Monitoring and Logging

Probes

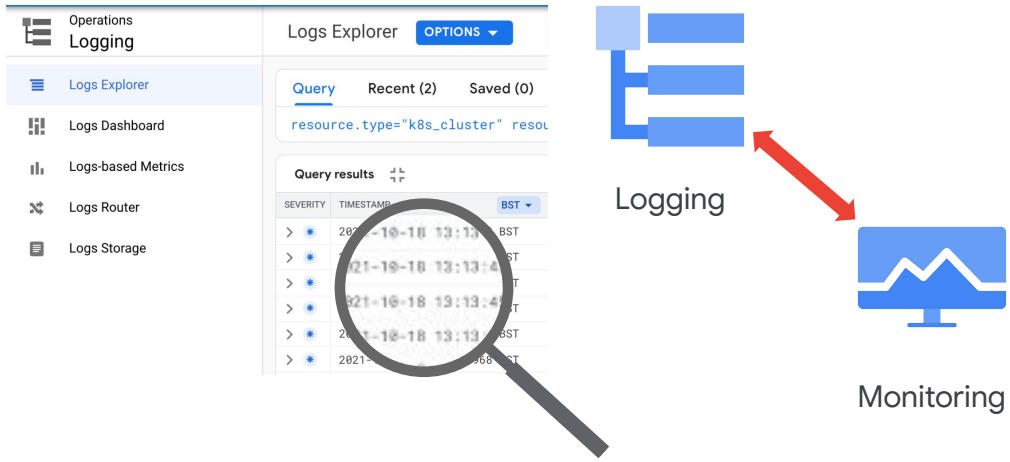
Lab: Configuring Liveness and Readiness Probes

Summary

Google Cloud

In this lesson, we'll look at logging. Logging provides the forensic history on what a program, process, or service has been doing. Whether a service or product is operating well or experiencing errors, logging provides visibility into the tasks and events that have occurred.

Logging is a passive form of systems monitoring

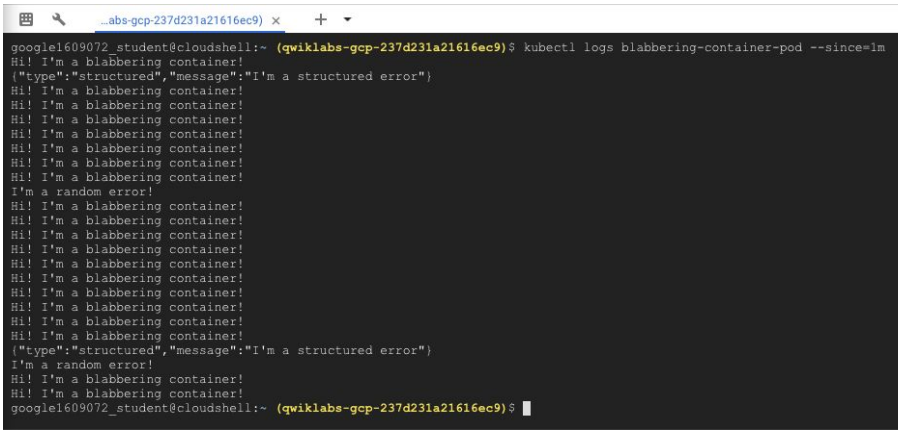


Google Cloud

Logging is often a passive form of systems monitoring. Instead of a monitoring service, which actively queries the health of a service, a logging service passively collects the event logs. These logs can then be used to identify patterns, or perhaps even help an operator track down the root cause of a system failure.

Logging should be used in conjunction with monitoring. Where monitoring checks end-user-facing metrics (in other words, your Service Level Indicators), logging can collect data on the internal systems in a much less intrusive manner.

Logs can be viewed from the Cloud Shell at no additional cost



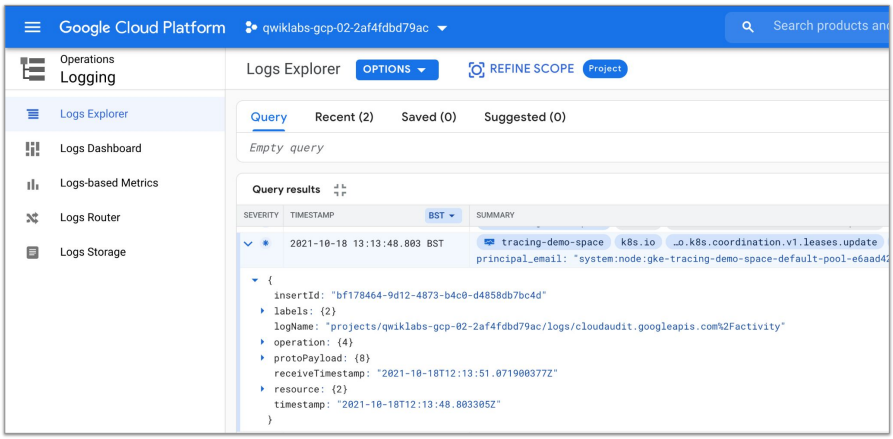
Google Cloud

There are two different ways to view logs.

You can view container logs using the kubectl command, or in the Cloud Logging web-based Cloud Console interface.

Viewing the logs with kubectl is a quick way to view the logs directly from the Pod. However, these logs aren't saved to a long-term database, and logs may be lost if containers are restarted or deleted. Cloud Logging saves the container logs for 30 days, although the log data may not be immediately available. Let's explore this next.

Cloud Logging provides a solution to collect and analyze product logs for GKE



Google Cloud

GKE automatically streams its logs to Cloud Logging to give you better visibility into the events and activity of your cluster.

Although Cloud Logging is a paid-for product, a Free Tier is available for a limited quantity of log storage. At the time this course was developed, storing the first 50 gibibytes of log information per project per month was free. You can export logs to Cloud Storage or BigQuery for long-term retention, if you need them for periods beyond 30 days, or to perform more complex analysis.

Why use one method instead of another?

Google Kubernetes Engine provides basic logs.

Cloud Logging provides much broader visibility and log correlation.

Google Cloud

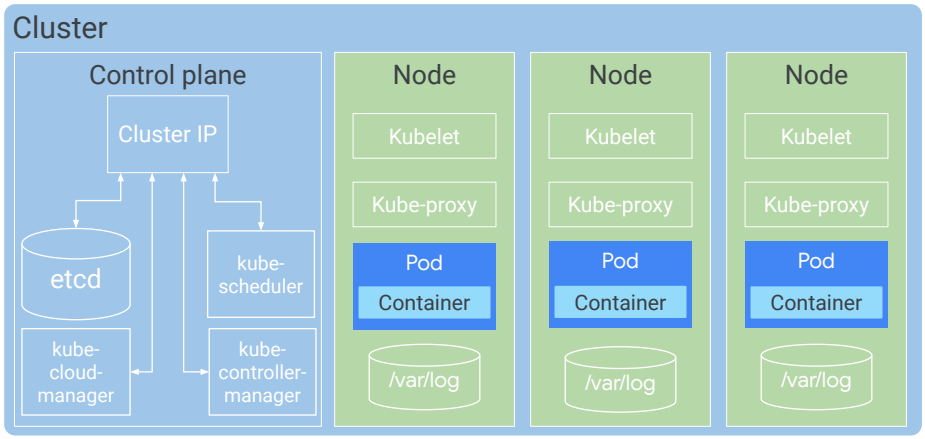
The basic GKE logs, such as the system component logs, standard output, and the standard error messages, are stored in the var log directory on the nodes and can be queried using the kubectl logs command or by directly accessing the var logs directory on the nodes. If you want the most recent logs for events that are happening right now then the kubectl logs command will give you those logs. However, if you want to find and examine logs for past events, or over a period of time in the last 30 days, then Cloud Logging gives you the tools to find and examine those logs.

If you let Cloud Logging handle the logs, you'll broaden the visibility of GKE events and be able to correlate issues better.

You'll have a single interface where you can review the logs from your containers, nodes, and other system services.

Note that although Cloud Logging does collect these logs, the data isn't kept in Cloud Logging forever. After 30 days it's purged, so if you want to keep the data longer, remember to export the logs to Cloud Storage or BigQuery for long-term analysis.

Kubernetes native logs are stored in each node's file system



Google Cloud

Let's start by discussing Kubernetes-native logging. Logs from Kubernetes system components, such as kubelet and kube-proxy, are stored in each node's file system, in its /var/log directory. Messages written by each container to its standard output and standard error are also logged in the same directory. You may be wondering whether, for a real-world application, those messages are useful. Yes. It's an increasingly common practice for applications to simply write log messages to standard output, with no buffering or filtering. Why? This way, these log messages can be captured and centrally managed.

You can view logs using the kubectl command

Viewing container logs

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

Container logs - Most recent 20 lines

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

Container logs - Most recent 3 hours

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

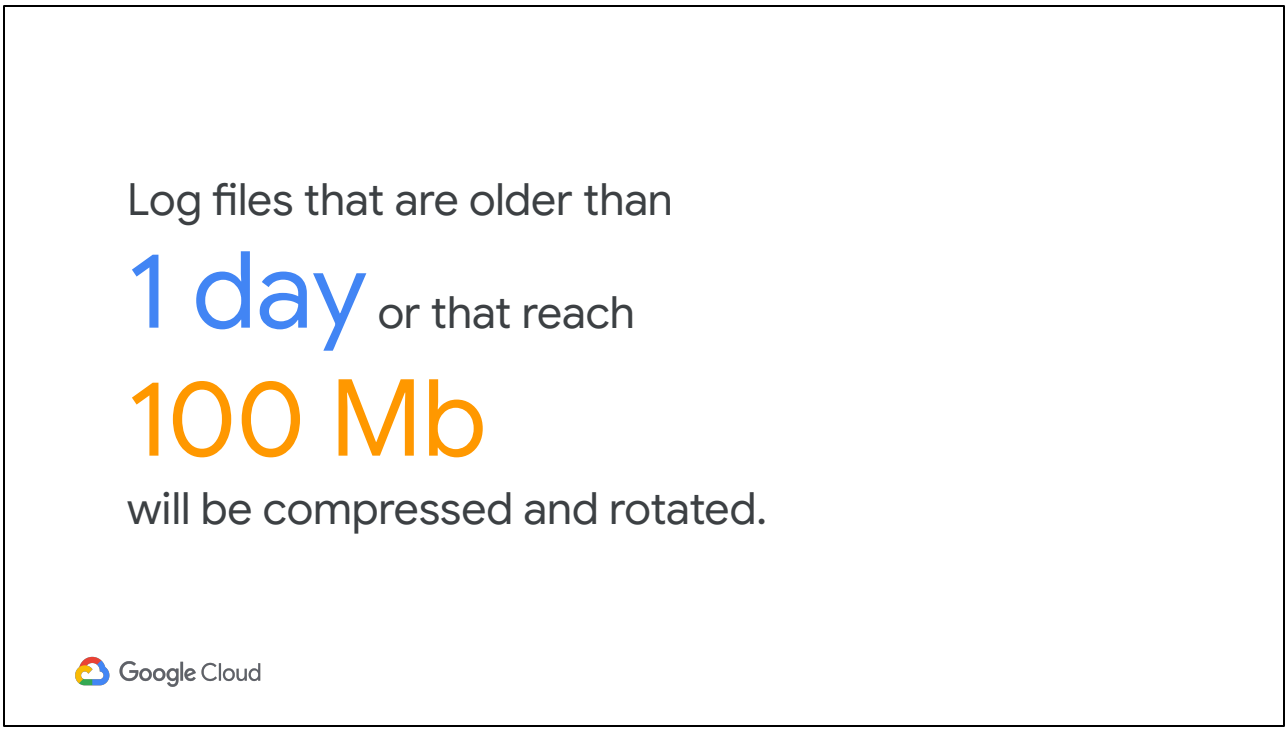
Google Cloud

To view the logs of a particular Pod, use “kubectl logs” followed by the name of the Pod. If you’re not sure of the Pod’s exact name, or if the Pod is part of a Deployment, you can run “kubectl get pods” to get a complete list of the Pods.

When troubleshooting recent issues, it’s much more convenient to retrieve only a handful of logs. By including the --tail option in the “kubectl logs” command, you can limit the number of logs that are returned. The example shown limits the output to the most recent 20 lines.

At other times, you might not know how many log messages to retrieve, but you know the problem occurred in the last few hours. Fortunately, you can also restrict the output using time as the criterion by adding the --since option, followed by a time period. In this example, kubectl will return all the logs for this Pod from the last 3 hours.

You can also view a previous instantiation of a container before it crashed and restarted using the --previous option. If your Pod has multiple containers, you can specify which container’s logs the kubectl command should return.



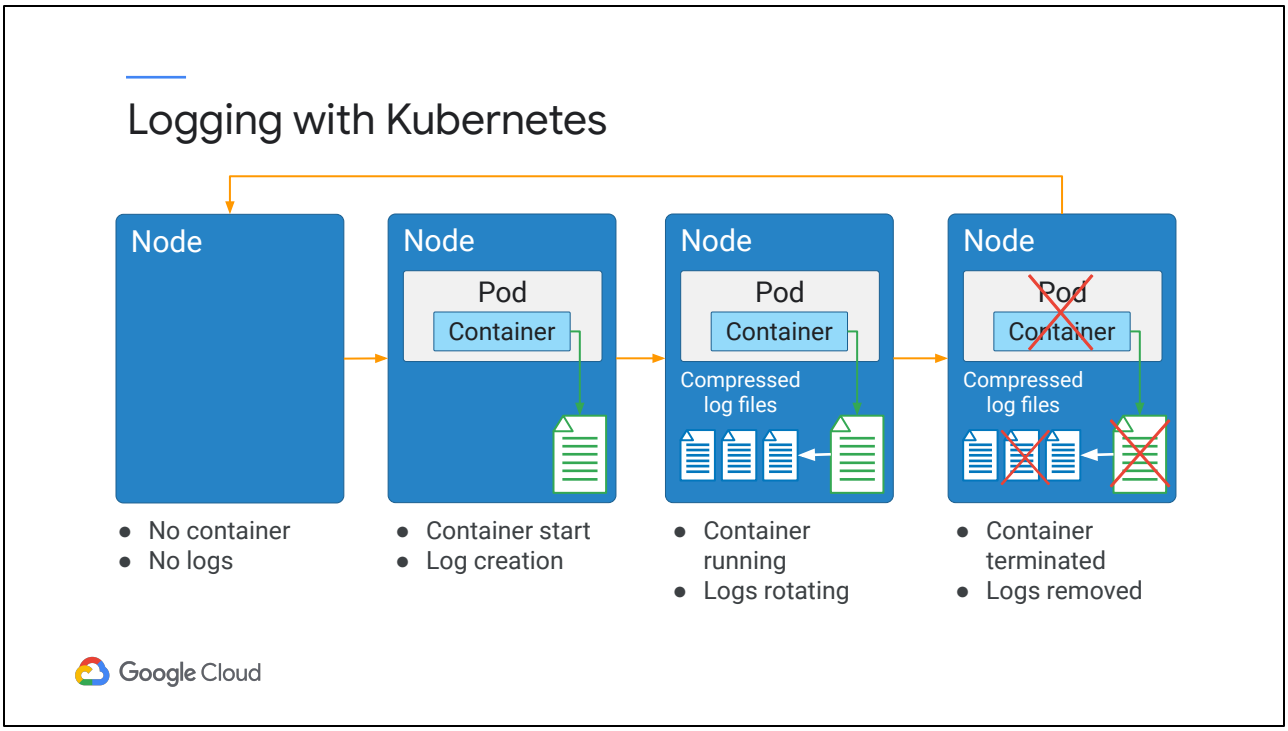
In Kubernetes, the container engine directs standard output and standard error streams from containers to a logging driver. This driver is configured to write these container logs in a JSON format and store them in the /var/log at the node level. As these log files grow, the node disk can easily become saturated. To avoid this, GKE streams these logs to Cloud Logging by default, and then regularly runs the Linux logrotate utility to clean up these files.

Any log file older than 1 day, or that has grown to 100 MB, will be compressed and copied into a archive file. Only the 5 most recent archived log files are kept on the node, previous versions are removed in order to avoid logs consuming too much disk space. However all log events are streamed to Cloud Logging, which retains all log event data 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.

Additional reference

This is a section of the “kube-up.sh” script that deploys GKE clusters within Google. This section contains the configuration for the log rotation. The documentation online is misleading in terms of the frequency and size thresholds; however the kube-up.sh script clearly shows that the logs are rotated DAILY -OR- at 100MB, whichever comes first.

```
/var/log/*.log {
    rotate ${LOGROTATE_FILES_MAX_COUNT:-5}
    copytruncate
    missingok
    notifempty
    compress
    maxsize ${LOGROTATE_MAX_SIZE:-100M}
    daily
    dateext
    dateformat -%Y%m%d-%s
    create 0644 root root
}
```

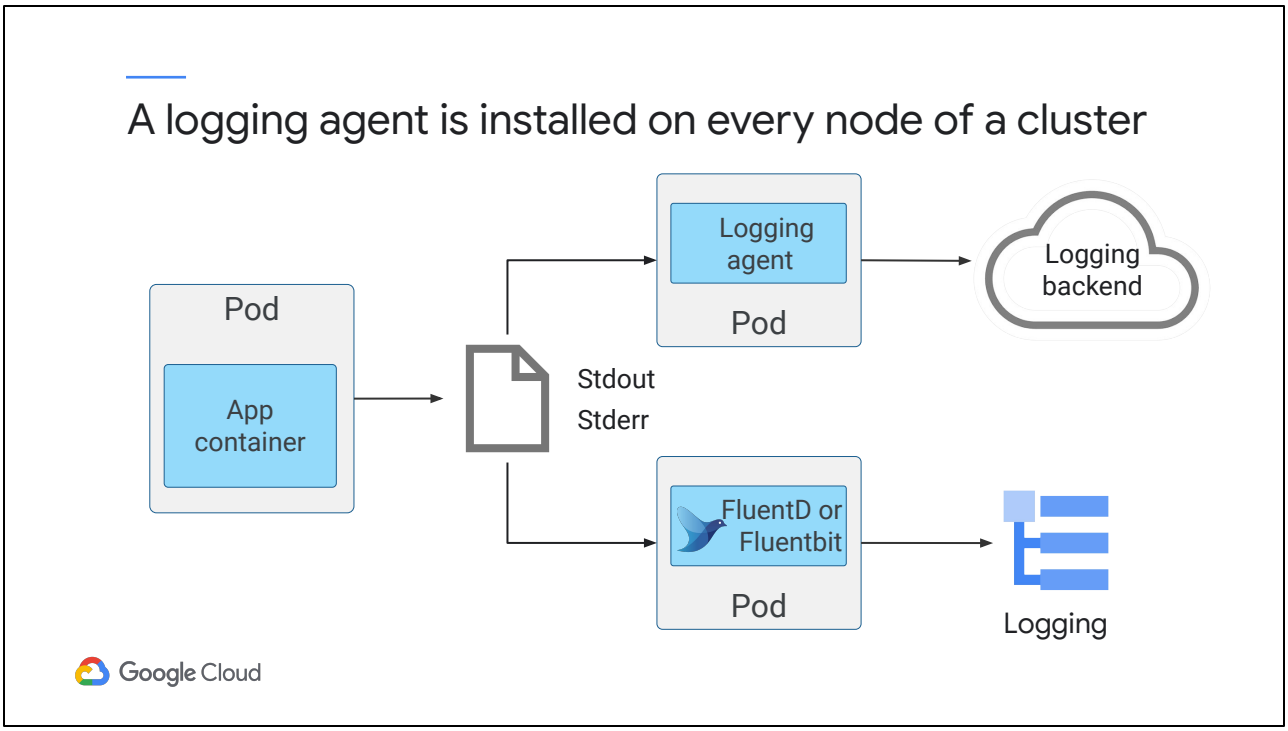


These native Kubernetes logs, and the compressed archives, aren't persistent. If, for any reason, a Pod is evicted or restarted, logs and the archives associated with that Pod are lost. As a result, there's a need to store logs outside a container, Pod, or node. This is called cluster-level logging. 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.

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. 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 the logs are deleted when the container is deleted from the node. If a Pod is deleted from the node, all corresponding containers are also deleted, along with their logs, which leaves you without any logs unless you have used a central log management utility such as Cloud Logging.



Now let's discuss Cloud Logging. As you've seen earlier, Logging is a fully managed service that manages these application and system logs. Logging is designed to automatically scale and ingest terabytes of log data per second. In GKE, Logging is enabled by default, but it can also be disabled on a cluster if required. Logging agents are preinstalled on the nodes and preconfigured to push your log data to Logging.

You can also use a public API to write a custom log and push it to Logging. Additionally, you can filter the logs displayed using Logging filter language, either in the Logs Viewer console, or directly through the Cloud Logging API.

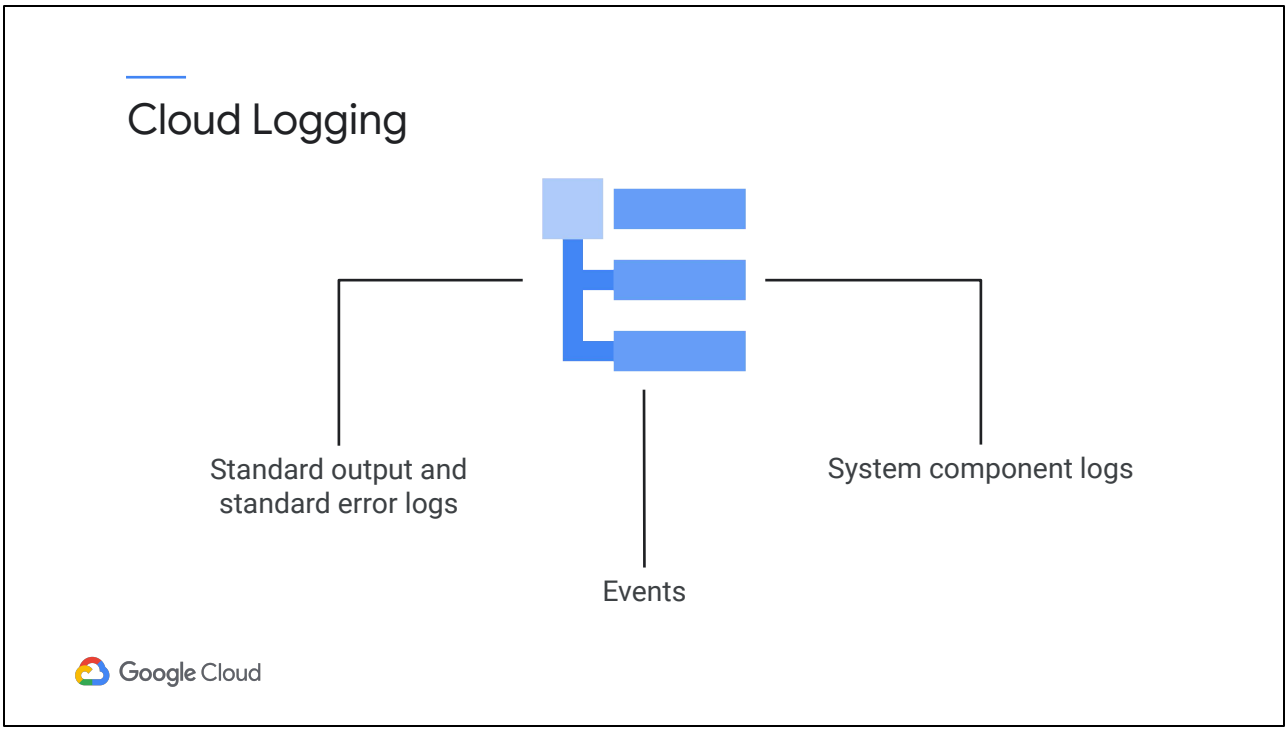
GKE installs a logging agent on every node of a cluster, and this agent collects and pushes the containers' logs and system component logs to the Cloud Logging backend.

Cloud Logging uses Fluentd or Fluentbit as the node logging agent. The agent is a log aggregator that will read all of the previously mentioned logs, adding helpful metadata, and then continuously push those logs into Cloud Logging. The agent is set up using a DaemonSet because DaemonSets can be used to ensure that every node in a cluster runs a copy of a specific Pod.

The configuration of the agent is managed through ConfigMaps. This increases the

scalability of the implementation by separating the application (the FluentD or Fluentbit DaemonSet) from the configuration elements (the ConfigMap).

Depending on your GKE cluster control plane version, either fluentd or fluentbit are used to collect logs. Starting from GKE 1.17, logs are collected using a fluentbit-based agent



Overall, in GKE, Cloud Logging monitors standard output and standard error logs from containerized processes, system component logs, for example kubelet, and events.

Events are all operations that take place in the cluster. Some examples include the deletion of a Pod, scaling of Deployments, and the creation of a container. These events are stored as API objects on the cluster control plane. Because these events are only stored temporarily in Kubernetes, GKE deploys an event exporter in the cluster control plane to capture these events and push them into Cloud Logging.

Google Cloud's operations suite also provides extensive features for customizing your monitoring, logging, and alerting solution.
Those features include the ability to create custom metrics that are based on Cloud Logging queries.

For example, you can create a custom metric, monitored using Cloud Monitoring, that counts the rate at which a specific event or group of events appears in the logs.

Custom metrics trigger automatic scaling of your infrastructure and you can also use them as part of custom dashboards and reports, and for alerts.

The alerting system is highly customizable and integrates with many third-party solutions so that you don't have to add another alerting tool to your arsenal.

The logs are kept in Cloud Logging for a period of 30 days. At that point, the logs are removed automatically, so you should export these logs to Cloud Storage or BigQuery if you need longer log retention periods.

Agenda

Google Cloud's Operations Suite

Logging

Monitoring

Lab: Configuring GKE-Native Monitoring and Logging

Probes

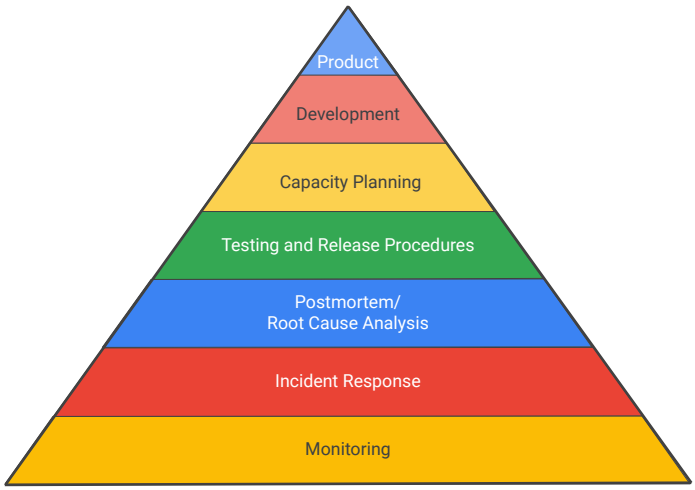
Lab: Configuring Liveness and Readiness Probes

Summary

Google Cloud

This lesson explores Monitoring. Monitoring allows you to visualize your service experience from different vantage points. You can monitor from outside the environment to view the service from the user's perspective, or you can monitor from within the environment to gain insights on key internal metrics.

Site Reliability Engineering is Google's approach to DevOps

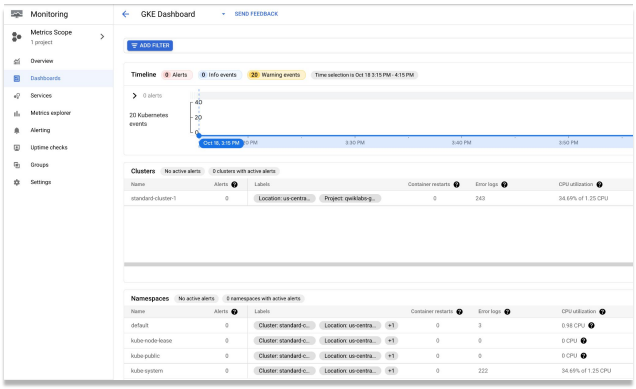


Google Cloud

You may have heard of Site Reliability Engineering, or "SRE." SRE is Google's approach to DevOps, and it's a fundamental part of how Google runs its services reliably and at massive scale. One important part of SRE is the Service Reliability Hierarchy. Here's a picture. This diagram shows what depends on what. The activity at a given level depends on what's beneath it. Notice that monitoring is the most fundamental layer of the Service Reliability Hierarchy. Everything else depends on it, all the way up to your service or product itself.

Monitoring lets you make decisions about your application based on data rather than emotions.

Why does monitoring matter?



- Provides a complete picture.
- Helps you size and scale systems.
- Provides focus on your application's current state.
- Helps you troubleshoot complex microservices solutions.

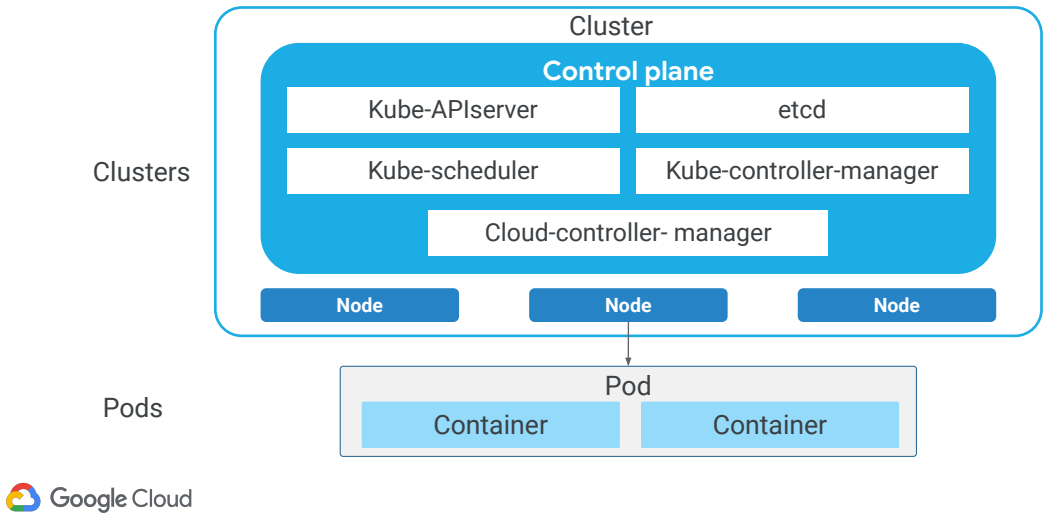
Google Cloud

Monitoring is your first line of reconnaissance. It's often coupled with logging to provide a complete picture of the system status and past trends. It can also reveals trends and patterns that can help you size and scale your systems over time.

Monitoring Kubernetes is actually less about watching the clusters and nodes, and more about monitoring your application's current state—and anything that may be impeding it.

Setting up a solid monitoring solution can really help when you are troubleshooting microservices-based applications. Because these services are loosely coupled by design, complex troubleshooting situations can arise when data is sent between different systems. If you're able to accurately monitor the state of these services, including the throughput and latency of the services, you can more easily track down performance bottlenecks. Through aggregated logging and debugging you can diagnose application code issues.

What do we monitor?



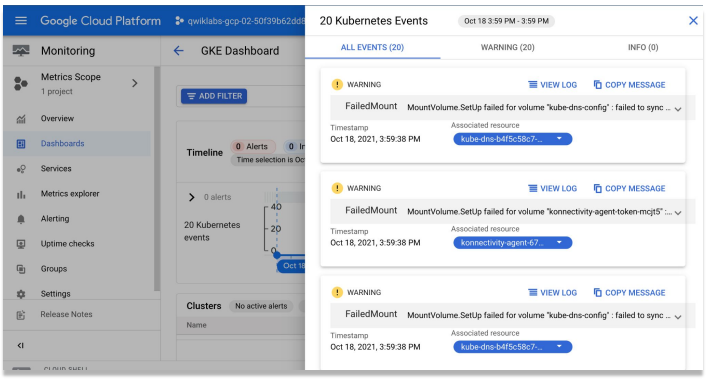
Google Cloud

In Kubernetes, monitoring can be broken into two domains.

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

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

Monitoring the cluster



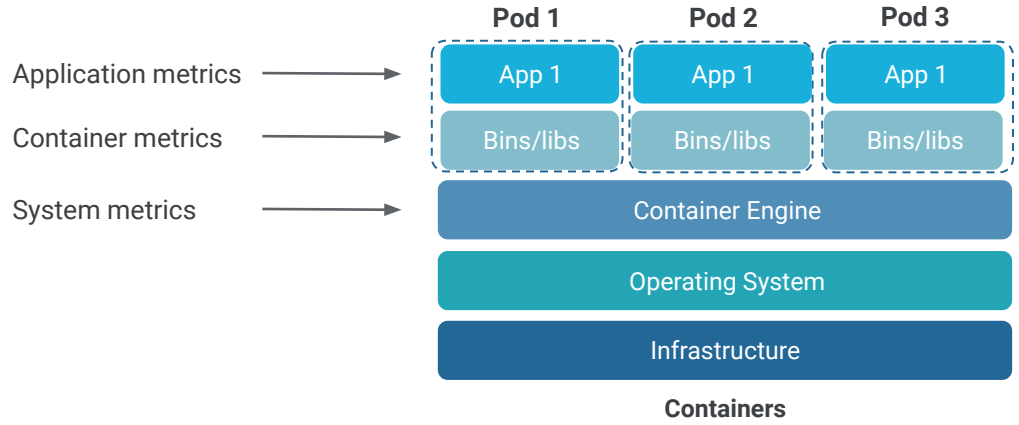
- Number of nodes
- Node utilization
- Pods/deployments running
- Errors and failures

Google Cloud

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.

Pod monitoring can be divided into several sub-categories



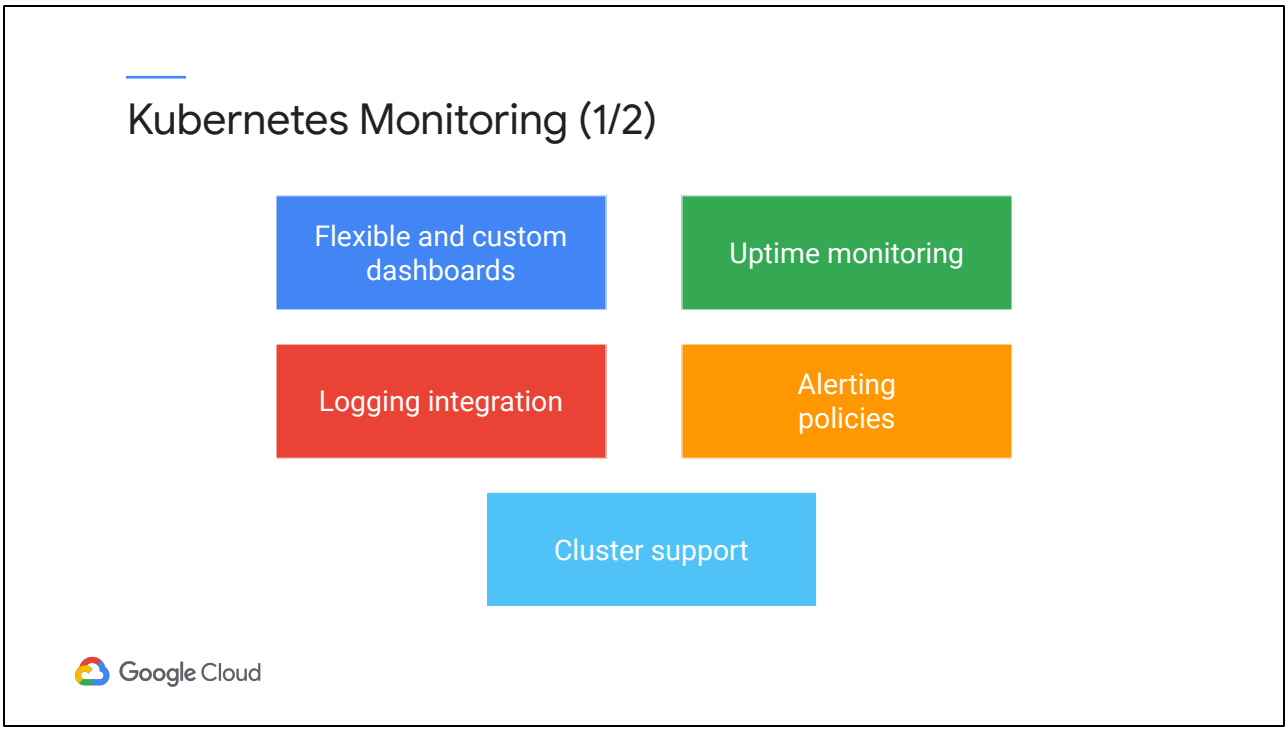
Google Cloud

System metrics, regarding 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.

Unlike traditional server monitoring, where you can specify a hostname to monitor, the abstraction layers in Kubernetes—well, containers in general—force you to change your approach. Instead of having a specific hostname or IP address to be monitored, all resources in Kubernetes have labels that you define to create a logical structure.

These labels give you a logical approach to organizing your resources. They also make it easier to for you to monitor specific systems or subsystems by selecting a combination of different labels. In Cloud Monitoring and other tools, you can filter the logs and focus the monitoring on components that match a given label. Here's an example. Remember that Kubernetes labels consist of a key and a value: You could apply a label consisting of the key "environment" and the value "production" to all the components of your production environment, and then use that label in Cloud Monitoring.



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

With a collection of metrics, events, and metadata, Monitoring provides the flexibility needed to monitor core metrics through custom dashboards.

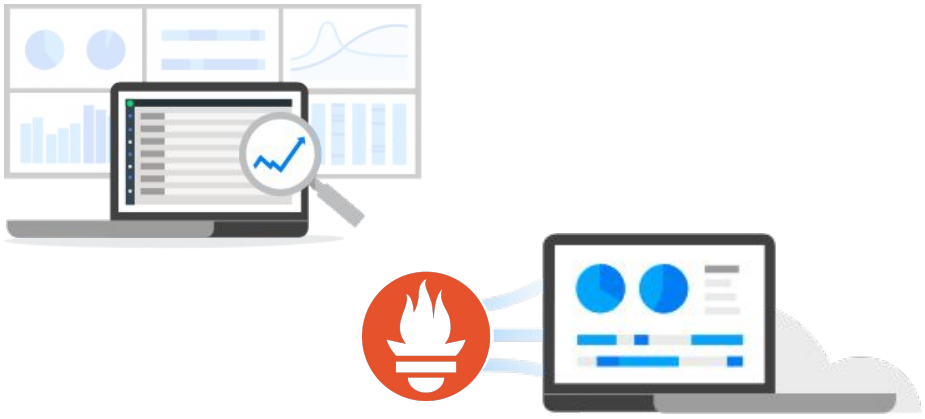
Through uptime checks, you can monitor the availability of your Kubernetes resources within the cluster and also the other Google Cloud resources used.

Logging also easily integrates with Monitoring using custom metrics; you can also monitor your logging-based custom metrics.

In addition, all these metrics can be configured to set up alerting policies to improve the signal-to-noise ratio of your alert messages.

All cluster resources are topologically aware and are appropriately labeled based on resource names, labels, projects, and more.

Kubernetes Monitoring (2/2)



Google Cloud

Cloud Monitoring is optimized for Kubernetes. It lets you observe your whole Kubernetes environment. You can see metric, logs, traces, events, and metadata in one place.

Imagine that your application has an error. Now you can analyze relevant information such as infrastructure metrics, application metrics, or logs directly from a single dashboard.

It supports multi-cluster monitoring within the Google Cloud environment and also in other cloud environments and on-premises Kubernetes environments. It has also extended direct support to Prometheus.

Cloud Monitoring can be extended by Prometheus

Google Cloud

Cloud Monitoring can only monitor what it can see. To get more information out of Cloud Monitoring, you need to put more information in. That's where an open source tool named 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, providing you with much more granular detail than Cloud Monitoring alone.

Agenda

Google Cloud's Operations Suite

Logging

Monitoring

Lab: Configuring GKE-Native Monitoring and Logging

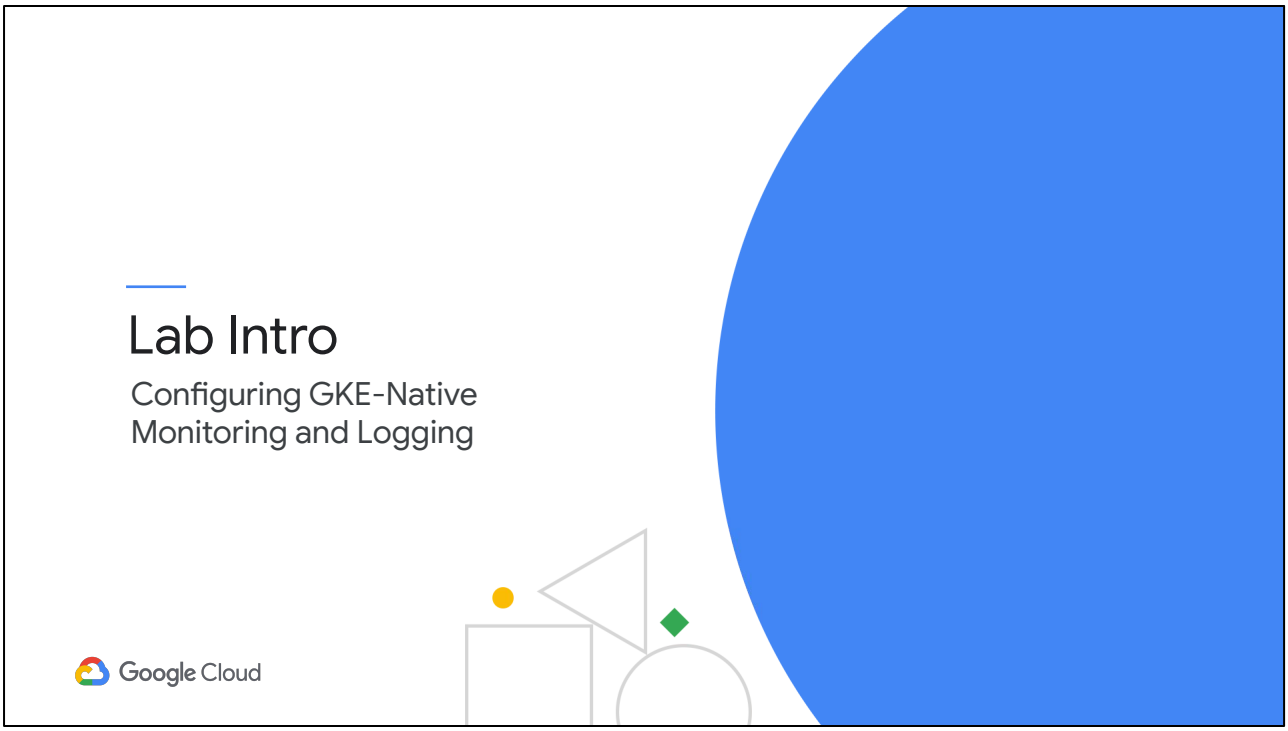
Probes

Lab: Configuring Liveness and Readiness Probes

Summary

Google Cloud

Let's start by introducing Google Cloud's operations suite. In this lesson, I'll introduce its key features and describe how you use them to monitor and troubleshoot Google Kubernetes Engine clusters and applications.



In this lab, you build a GKE cluster and then deploy pods for use with Kubernetes Engine Monitoring. You will create charts and a custom dashboard, work with custom metrics, and create and respond to alerts.

Agenda

Google Cloud's Operations Suite

Logging

Monitoring

Lab: Configuring GKE-Native Monitoring and Logging

Probes

Lab: Configuring Liveness and Readiness Probes

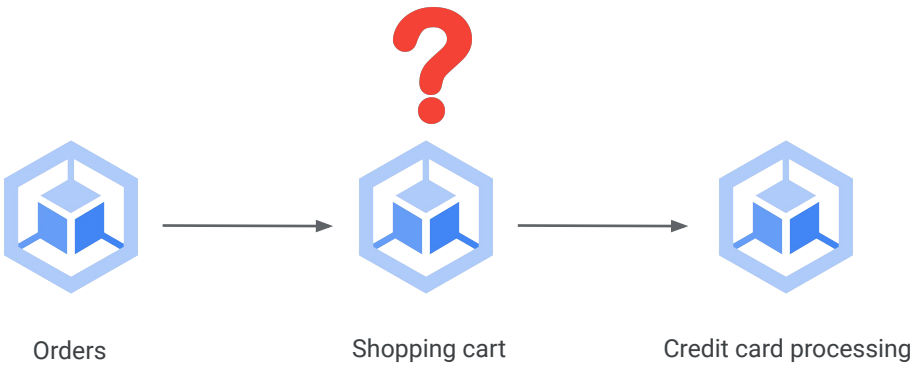
Summary

Google Cloud

In the last lesson of this module, we'll discuss probes. When working in a microservices environment, you'll have service dependencies. If a Service isn't ready, another Service might generate an error. Also, some containers might appear to be operational because they're consuming compute resources, but in fact they aren't able to service client requests. In both of these cases, the other microservices would generate errors that we can prevent by checking the status of a Pod, and if it's found to be non-responsive, redeploying it.

Setting up probes

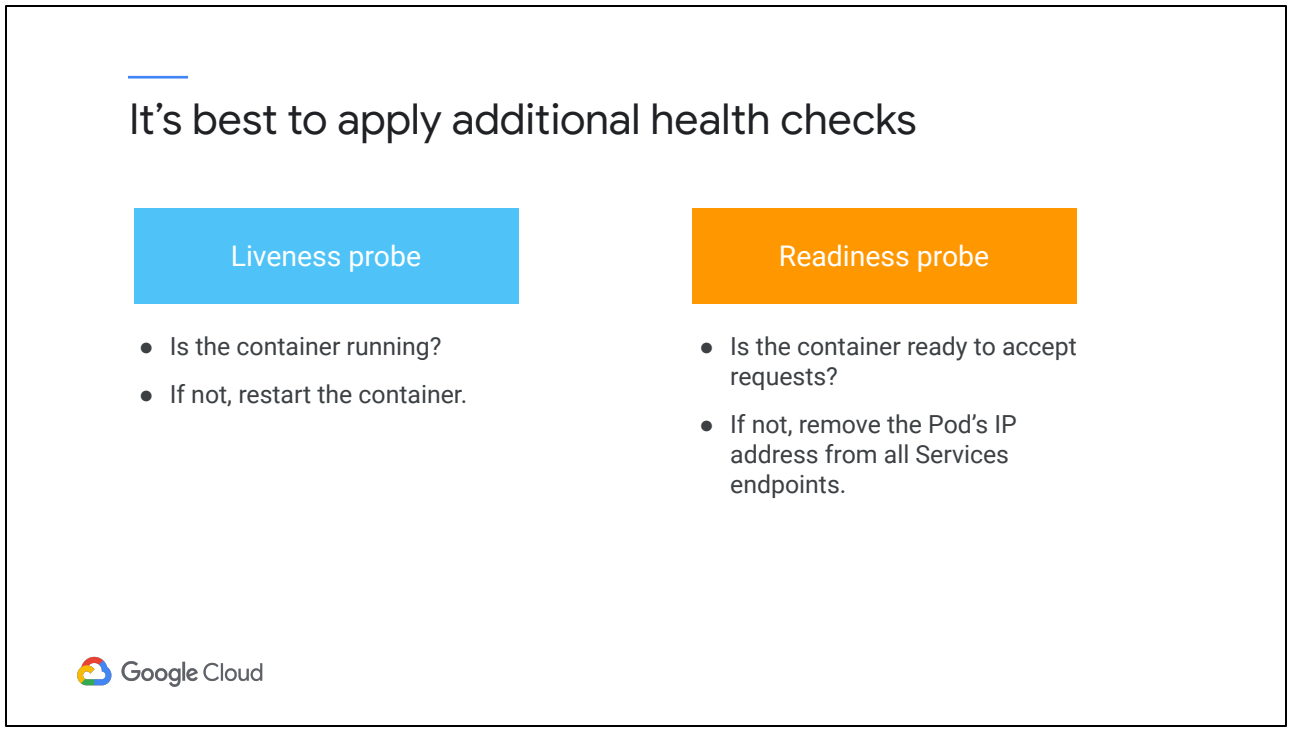
How do you know whether the shopping cart is functioning properly?



Google Cloud

In this example, the shopping process is broken down into various microservices such as orders, shopping cart, and credit card processing. It's vital that the orders are saved appropriately in the shopping cart. If the orders aren't saved correctly in the shopping cart, the credit card won't be processed correctly.

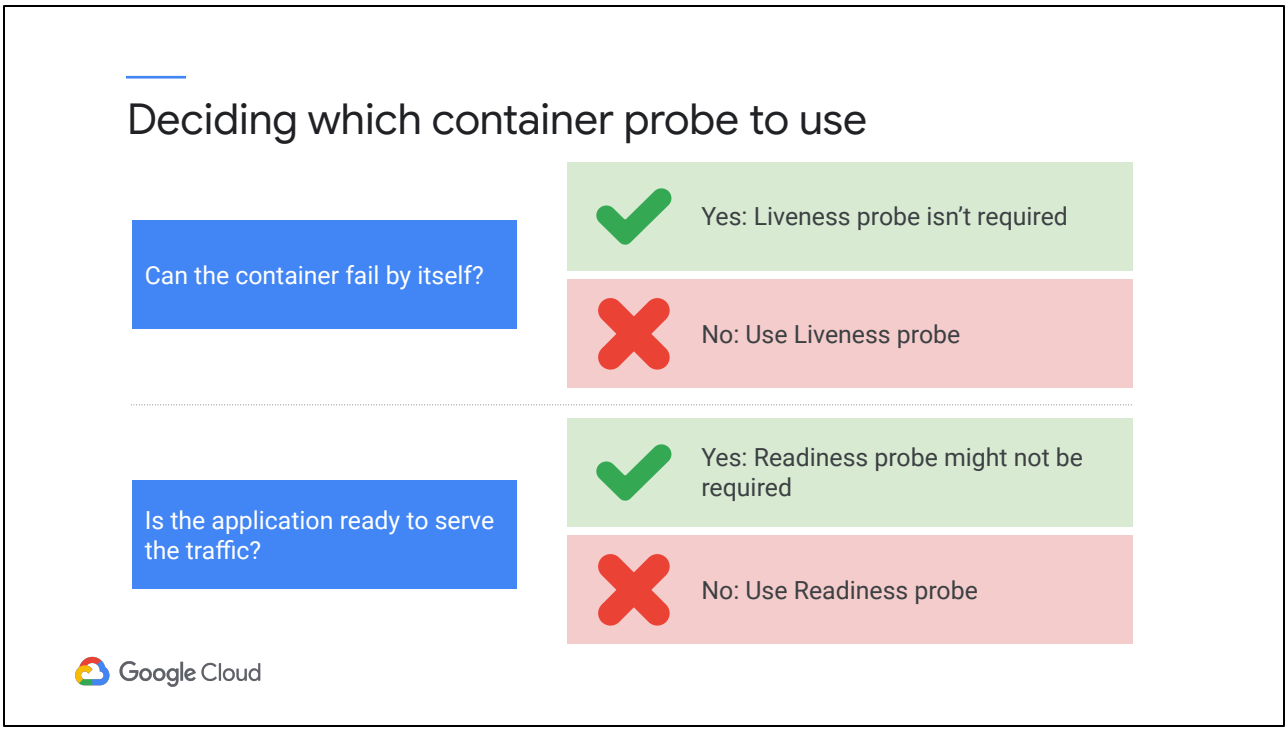
This might eventually lead to a lost conversion and ultimately, lost revenue. Overall, it's essential to have a functional shopping cart microservice. So how do you know whether the shopping cart is functioning properly?



It’s best to apply additional health checks such as liveness and readiness probes.

With a liveness probe, Kubernetes checks to see whether the container is running. If the liveness probe fails, and if the restartPolicy is set to Always or OnFailure, kubelet will restart the container.

With a readiness probe, Kubernetes checks whether the container is ready to accept requests. If a readiness probe fails, the Pod’s IP address is removed from all Services endpoints by the endpoint controller.



These probes can be defined using three type of handlers, Command, HTTP and TCP, to allow you to perform different types of diagnostic probe tests. How do you decide whether you need to use a liveness or readiness probe? Consider some common scenarios.

If your container can fail by itself and the restartPolicy is set to Always or OnFailure, you don't need a liveness probe or a readiness probe; kubelet will simply act based on the defined restartPolicy and restart the container.

However, if an application within a container is stuck in a broken state and requires a restart, you can set up a liveness probe to detect the broken state and restart the container.

The third case where a probe can help is where your container might be running, but your application isn't yet ready to serve the traffic.

By default, Kubernetes would send traffic to the container because it is running. Even if you've set up a liveness probe, the liveness probe will simply fail and restart the container. This just ends up in a continuous loop with the application never getting to a state where it is ready to receive traffic. In this case, you can add a readiness probe to make sure your application is ready and running before the traffic is sent. Note that

if you don’t specify any of these probes, by default Kubernetes assumes that the Pod executed successfully.

Command probe handler

```
apiVersion: v1
kind: Pod
metadata:
  name: demo-pod
  namespace: default
spec:
  containers:
  - name: liveness
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/ready
```

Liveness and Readiness probes are configured at the container level.

With the command probe handler, kubelet runs a command inside the container. If the command succeeds -- in other words, if its exit code is zero -- the container is considered healthy. Otherwise, kubelet will kill the container.

HTTP probe handler

```
[...]
spec:
  containers:
  - name: liveness
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
```

Google Cloud

The second type of probe handler, HTTP, uses an HTTP GET request. If the request returns with a code range from 200 to 400, kubelet considers the container healthy. Anything outside that range will kill the container. You can easily set up a lightweight HTTP server on the container to use this probe.

TCP probe handler

```
[...]
spec:
  containers:
  - name: liveness
    livenessProbe:
      tcpSocket:
        port: 8080
```

Google Cloud

The third probe handler type is TCP. Here, kubelet attempts to make a TCP connection. If the connection is established, the container is considered healthy. All these methods can also be used in a Readiness probe in exactly the same way.

Probes can be refined

```
[...]
spec:
  containers:
  - name: liveness
    livenessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 10
      timeoutSeconds: 1
      successThreshold: 1
      failureThreshold: 3
```

Google Cloud

You can refine these probes. The IntialDelaySeconds field sets the number of seconds to wait before liveness or readiness probes can be initiated. It's important to ensure that the probe isn't initiated before the application is ready; otherwise, the containers will be thrashed in a continuous loop. This value should be updated if your application boot time changes.

The PeriodSeconds field defines the interval between probe tests, the timeoutSeconds field defines the probe timeout, and success and failure threshold fields can also be set.

Agenda

Google Cloud's Operations Suite

Logging

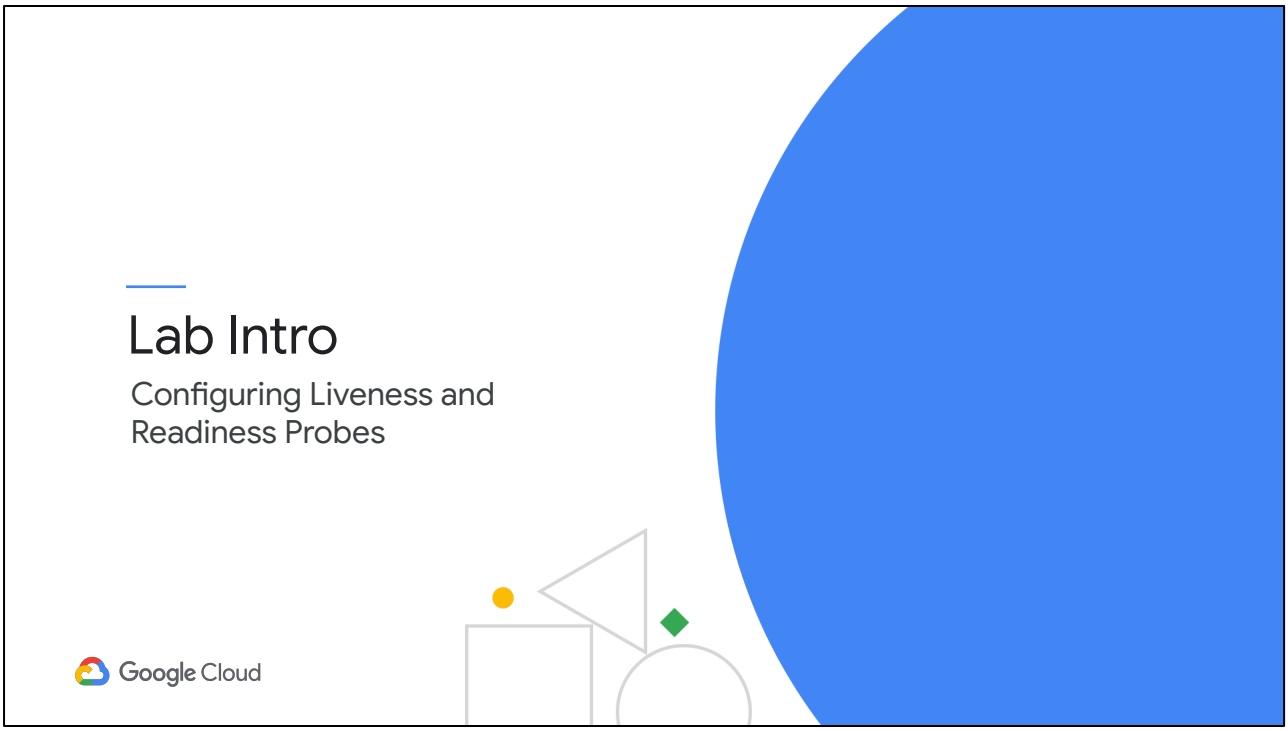
Monitoring

Lab: Configuring GKE-Native Monitoring and Logging

Probes

Lab: Configuring Liveness and Readiness Probes

Summary



In this lab, you will configure and test Kubernetes liveness and readiness probes for containers.

Agenda

Google Cloud's Operations Suite

Logging

Monitoring

Lab: Configuring GKE-Native Monitoring and Logging

Probes

Lab: Configuring Liveness and Readiness Probes

Summary

Google Cloud

Let's start by introducing Google Cloud's operations suite. In this lesson, I'll introduce its key features and describe how you use them to monitor and troubleshoot Google Kubernetes Engine clusters and applications.

Summary

Use Google Cloud's operations suite to monitor and manage availability and performance.

Locate and inspect Kubernetes logs.

Perform forensic analysis of logs.

Monitor performance.

Create probes for wellness checks on live applications.

Google Cloud

That concludes Google Kubernetes Engine Logging and Monitoring.

In this module, you learned how to use Google Cloud's operations suite to monitor and manage the availability and performance of your Google Cloud resources and applications that are built with those resources; locate and inspect Kubernetes logs produced by resources inside your GKE clusters; use Cloud Logging and BigQuery for longer term retention and forensic analysis of the logs produced by GKE and the Kubernetes resources inside your GKE clusters; how to monitor system and application performance from different vantage points; and create probes for wellness checks on live applications.


Google Cloud