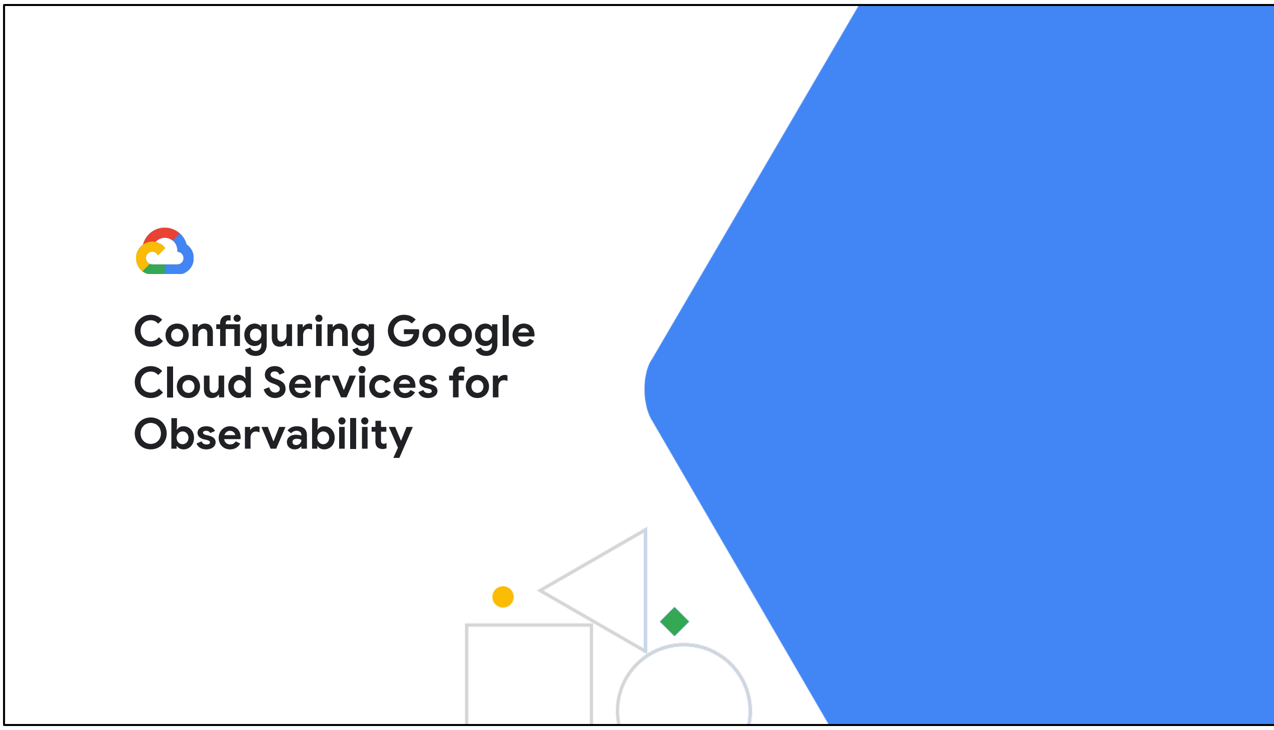


In the next part of our operations suite discussion, let’s take a little time to examine the art of configuring Google Cloud services for observability.

Agenda

Working with Agents

- Monitoring
- Logging
- Images and Agent Policies

Non-VM Resources

Exposing Custom Metrics

Google Cloud

In this module, we're going to spend a little time learning how to:

- Integrate logging and monitoring agents into Compute Engine VMs and images, using Agents.
- Enable and utilize Kubernetes monitoring.
- Extend and clarify Kubernetes monitoring with Prometheus.
- And expose custom metrics through code, and with the help of OpenCensus.

Agenda

Working with Agents

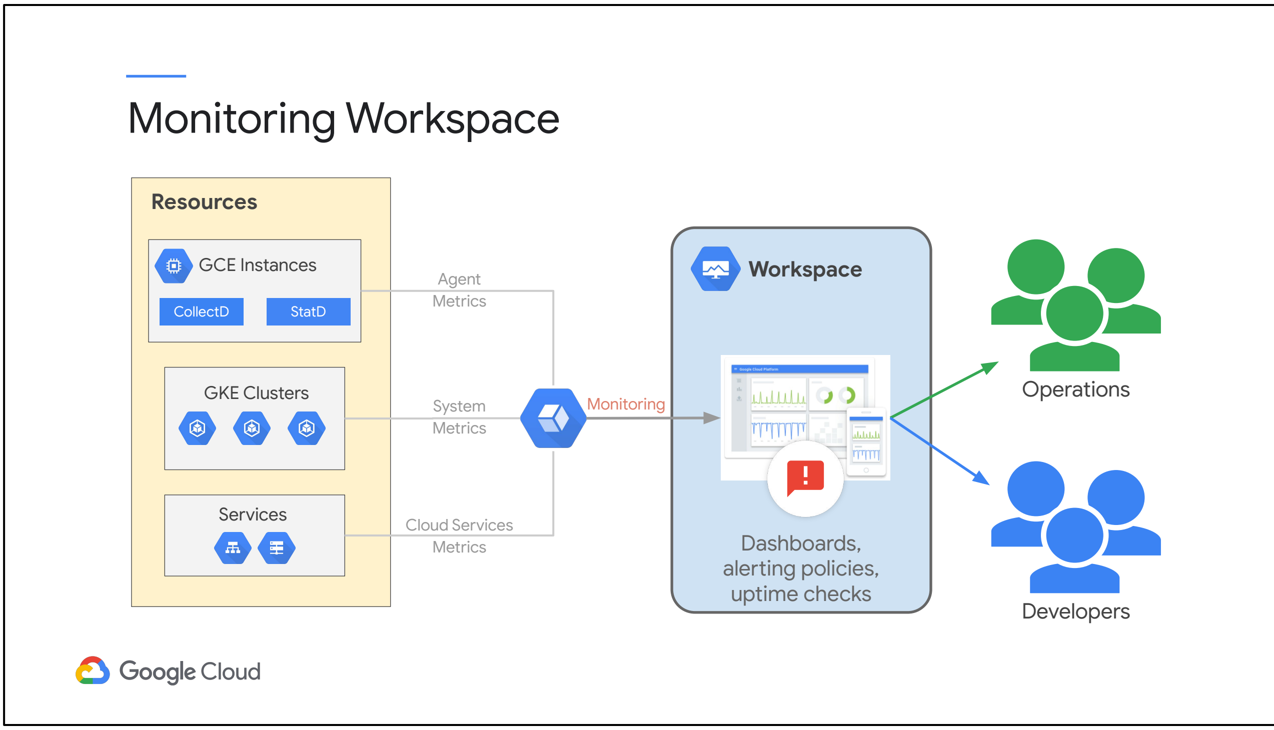
- Monitoring
- Logging
- Images and Agent Policies

Non-VM Resources

Exposing Custom Metrics

Google Cloud

Let's start with Logging and Monitoring agents for Compute Engine.



As we've discussed, monitoring data can originate at a number of different sources. With Google Compute Engine instances, since the VMs are running on Google hardware, Monitoring can access some instance metrics without the Monitoring agent, including CPU utilization, some disk traffic metrics, network traffic, and uptime information, but that information can be augmented by installing agents into the VM operating system.

These agents are required because for security reasons, the hypervisor cannot access some of the internal metrics inside a VM, for example, memory usage.

GKE clusters also send system metrics via an agent.

OS Monitoring agent

Gathers system and application metrics from VM instances and sends them to Monitoring

- Based on the open-source collectd
- Gathers additional system resources and application metrics
- Optional, but recommended
- Supports third-party applications, such as:
  - Apache/Nginx/MySQL
- Additional support offered through BindPlane from Blue Medora
- Supports major operating systems:
  - CentOS, Debian, Red Hat Enterprise Linux
  - Ubuntu LTS, SUSE Linux Enterprise Server
  - Windows server

Google Cloud

The Cloud Monitoring agent is a collectd-based open-source daemon that gathers system and application metrics from virtual machine instances and sends them to Monitoring.

By default, the optional but recommended Monitoring agent collects disk, CPU, network, and process metrics.

You can configure the Monitoring agent to monitor third-party applications like Apache, mySQL, and NGINX.

Additional support provided through integration through BindPlane from Blue Medora.

The Monitoring agent supports most major operating systems from CentOS, to Ubuntu, to Windows.

Lecture Notes:
VM that you create, they do not have monitoring agents. Monitoring agents are required for whitebox monitoring (instead of blackbox monitoring)

Services with “other” Monitoring support

Don’t try to manually install or configure the agent

- App Engine standard has monitoring built-in
- App Engine flex has agent pre-installed and configured
- GKE nodes has monitoring configurable and enabled by default
- Anthos GKE On-Prem agent collects system but not application metrics
- Cloud Run provides integrated monitoring support
- Cloud Function supports integrated monitoring

Google Cloud

When monitoring any of the following non-virtual machine systems in Google Cloud, the Monitoring agent is not required, and you should not try and install it:

- App Engine standard has monitoring built-in.
- App Engine flex is built on top of GKE and has the Monitoring agent pre-installed and configured.
- With Standard Google Kubernetes Engine nodes (VMs), logging and monitoring is an option which is enabled by default.
- Currently, the Anthos GKE On-Premises agent collects system but not application metrics.
- Cloud Run provides integrated monitoring support.
- And Cloud Function supports integrated monitoring.

Installing the Monitoring agent

SUSE

```
curl -sSO https://dl.google.com/cloudagents/add-monitoring-agent-repo.sh
sudo bash add-monitoring-agent-repo.sh
sudo zypper install stackdriver-agent
sudo service stackdriver-agent start
```

Debian 10

```
curl -sSO https://dl.google.com/cloudagents/add-monitoring-agent-repo.sh
sudo bash add-monitoring-agent-repo.sh
sudo apt-get update
sudo apt-get install stackdriver-agent
sudo service stackdriver-agent start
```

Google Cloud

Installing the Monitoring agent is well documented on the Google site. Here, you see examples of installing the agent on SUSE and in Debian 10.

Lecture Notes:
How to install?
(1) Use VM instances dashboard
(2) use this slide
(3) Agent policies (BETA) (https://cloud.google.com/stackdriver/docs/solutions/managing-agent-policies)
(4) IaC like Terraform
(5) Custom images

Today Google has a new agent coming in, which is called a "Unified agent" - which is an agent for both monitoring and logging. But that is not yet the default.

Installing the Monitoring agent

Centos 8

```
curl -sSO https://dl.google.com/cloudagents/add-monitoring-agent-repo.sh
sudo bash add-monitoring-agent-repo.sh
sudo yum install -y stackdriver-agent
sudo service stackdriver-agent start
```

Other

- All other Linux distros, see here
- Windows, see here

Google Cloud

Here's another example of installing the agent in Centos 8. Please see the Google site for installing the agent into other Linux distributions or into Windows. Note, if using an HTTP proxy, there are extra steps for both Linux and Windows.

Verifying Monitoring agent authorization

Execute on the VM

```
curl --silent --connect-timeout 1 -f -H "Metadata-Flavor: Google" \
http://169.254.169.254/computeMetadata/v1/instance/service-accounts/default/scopes
```

Check for one or more of the following

```
https://www.googleapis.com/auth/monitoring.write
https://www.googleapis.com/auth/monitoring.admin
https://www.googleapis.com/auth/cloud-platform
```

Google Cloud

Once the agent has been successfully installed and started, execute the curl command you see at the top of this slide to get a list of the VM's access scopes from the Compute Engine metadata service.

In the returned list, verify that you see at least one of the scopes listed at the bottom of this slide.

Adding credentials

Note: This is only required if the test on the previous slide failed

1. Create a service account.
2. Grant it the Monitoring Metric Writer role (add Logs Writer to support logging as well).
3. Generate and download a JSON key file.
4. Copy the file to:
   Linux: /etc/google/auth/application_default_credentials.json
   Windows: C:\ProgramData\Google\Auth\application_default_credentials.json
   Or place its path in environment as: GOOGLE_APPLICATION_CREDENTIALS
5. Restart the agent.

Google Cloud

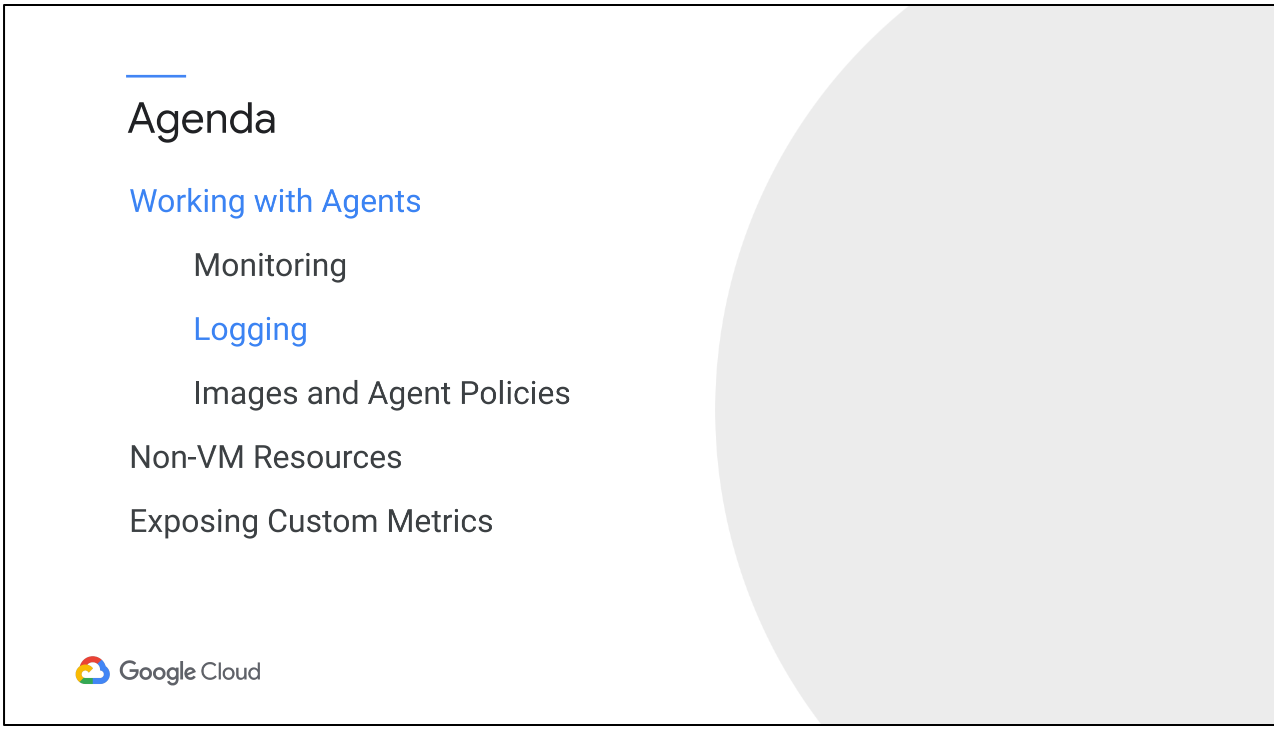
Normally, you won't have to perform this step, but if the test on the previous slide failed, then you are missing the scope required to allow your VM to write metrics into Monitoring.

Start by creating a service account, granting it the Monitoring Metric Writer role, and generating a JSON key file.

Take the downloaded file and copy it to the appropriate location, depending on whether you are using Windows or Linux.

Finally, restart the agent.

Make sure to check the Google documentation if you have questions.



In addition to the Monitoring agent, Google also recommends installing the Logging agent into your VMs.

OS Logging agent

Streams logs from common third-party applications and system software to Google Cloud Logging

- Supports third-party applications, such as:
  - Apache/Tomcat/Nginx
  - Chef/Jenkins/Puppet
  - Cassandra/Mongodb/MySQL
- Based on fluentd log data collector—can add own fluentd configuration files
- Supports major operating systems:
  - CentOS
  - Debian
  - Red Hat Enterprise Linux
  - Ubuntu LTS
  - SUSE
  - Windows Server

Google Cloud

By default, Google has little visibility into the logs that are created at the operating system level of your VM. If you need access to something like the Windows Event log, or the Linux syslog, you need to install the Google Logging agent.

Like the earlier discussed Monitoring agent, the Logging agent can stream logs from common third-party applications and system software to Google Cloud Logging.

It supports a number of third third-party applications, including: Apache, Nginx, and Jenkins, just to name a few.

Based on the open-source fluentd log data collector, it supports standard fluentd configuration files and options.

It supports many major operating systems, including CentOS, Ubuntu, SUSE, and Windows. Check the documentation for details.

Lecture Notes:
If you are going to install image, then use it as custom image. That way agent is always installed

Services with "other" Logging support

Don't try to manually install or configure the agent

- App Engine flex and standard have built-in support for logging
- GKE nodes can enable GKE logging
- Anthos GKE On-Prem agent collects system but not app logs
- Cloud Run has built-in logging support
- Cloud Functions have built-in logging support

Google Cloud

When collecting logging for any of the following non-virtual machine systems in Google Cloud, the Logging agent is not required, and you should not try and install it:

- App Engine standard and flex have logging support integrated, though there are extra logging options with flex.
- With Standard Google Kubernetes Engine nodes (VMs), Logging and Monitoring is an option which is enabled by default.
- Currently, the Anthos GKE On-Premises agent collects system but not application metrics.
- Cloud Run includes integrated logging support.
- Cloud Functions, both HTTP and background functions, include built-in support for logging.

Installing the Logging agent

Linux

```
curl -sSO https://dl.google.com/cloudagents/install-logging-agent.sh
sudo bash install-logging-agent.sh
```

Windows (PowerShell terminal)

```
cd $env:UserProfile;
(New-Object Net.WebClient).DownloadFile(
    "https://dl.google.com/cloudagents/windows/StackdriverLogging-v1-10.exe",
    ".\StackdriverLogging-v1-10.exe")
.\StackdriverLogging-v1-10.exe
```

Google Cloud

Installing the Logging agent is [well documented](#)on the Google site. Here, you see examples of installing the agent on Linux and using PowerShell to install it in Windows.

Verifying Logging agent authorization

Execute on the VM

```
curl --silent --connect-timeout 1 -f -H "Metadata-Flavor: Google" \
http://169.254.169.254/computeMetadata/v1/instance/service-accounts/default/scopes
```

Check for one or more of the following

```
https://www.googleapis.com/auth/logging.write
https://www.googleapis.com/auth/logging.admin
https://www.googleapis.com/auth/cloud-platform
```

Google Cloud

Once the agent has been successfully installed and started, execute the curl command you see at the top of this slide to get a list of the VM's access scopes from the Compute Engine metadata service.

In the returned list, verify that you see at least one of the scopes listed at the bottom of this slide.

Adding credentials

Note: This is only required if the test on the previous slide failed

1. Create a service account
2. Grant it the Logs Writer role (add Monitoring Metric Writer to support monitoring as well)
3. Generate and download a JSON key file
4. Copy the file to:
   Linux: /etc/google/auth/application_default_credentials.json
   Windows: C:\ProgramData\Google\Auth\application_default_credentials.json
   Or place its path in environment as: GOOGLE_APPLICATION_CREDENTIALS
5. Restart the agent

Google Cloud

Normally, you won't have to perform this step, but if the test on the previous slide failed, then you are missing the scope required to allow your VM to write log entries into Cloud Logging.

Start by creating a service account, granting it the Logs Writer role, and generating a JSON key file.

Take the downloaded file and copy it to the appropriate location, depending on whether you are using Windows or Linux.

Finally, restart the agent.

Make sure to check the Google documentation if you have questions.

Agenda

Working with Agents

GCE discussion

- Monitoring
- Logging
- Images and Agent Policies

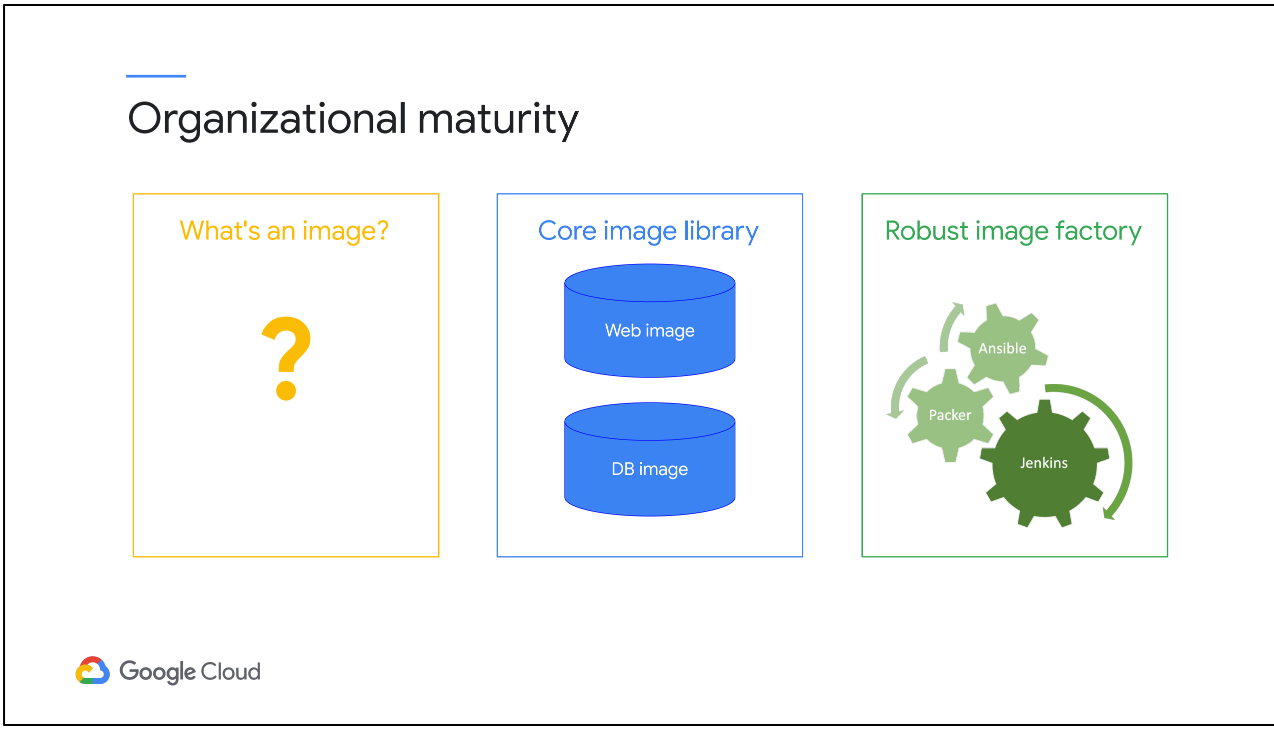
Non-VM Resources

Exposing Custom Metrics

Google Cloud

In the grand scheme of things, you don’t want to have to install the Logging and Monitoring agents each time you create a virtual machine.

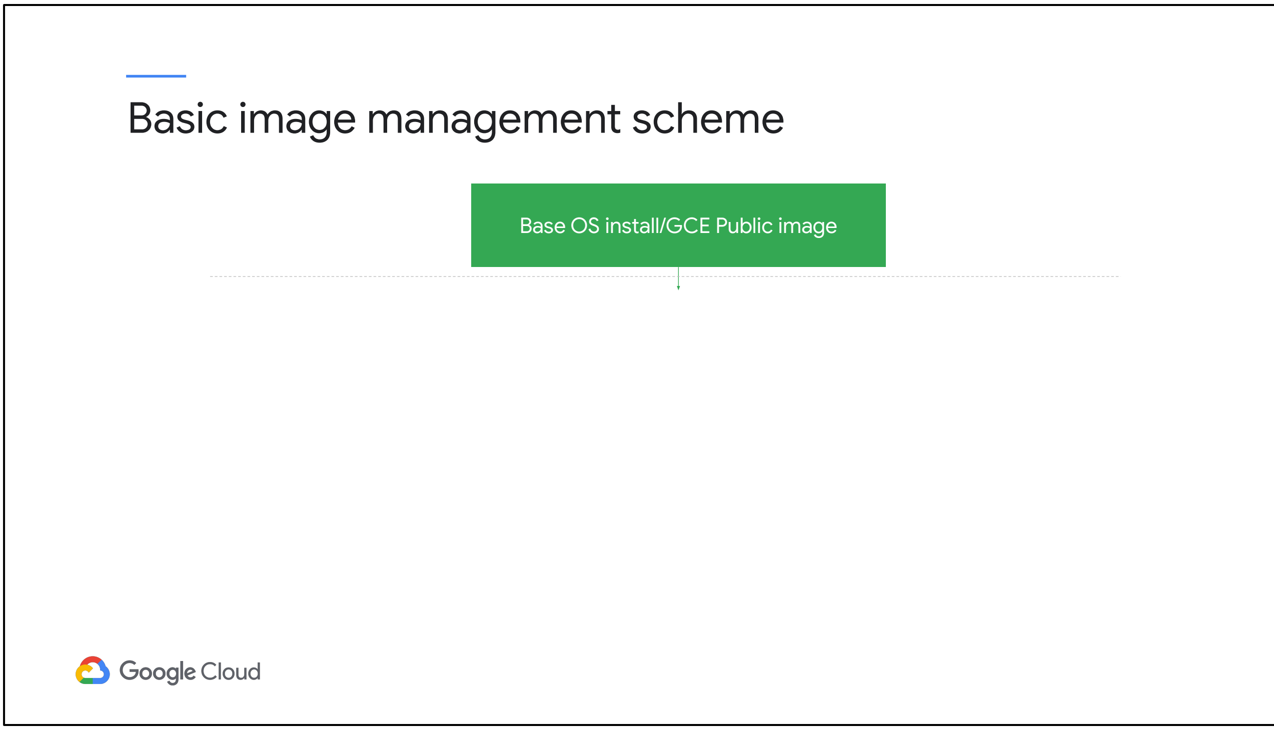
To ensure agent inclusion through a set of VMs in a project, folder of projects, or across the entire organization, two other good options are baking the agents into the virtual machine images, or using Agent Policies.



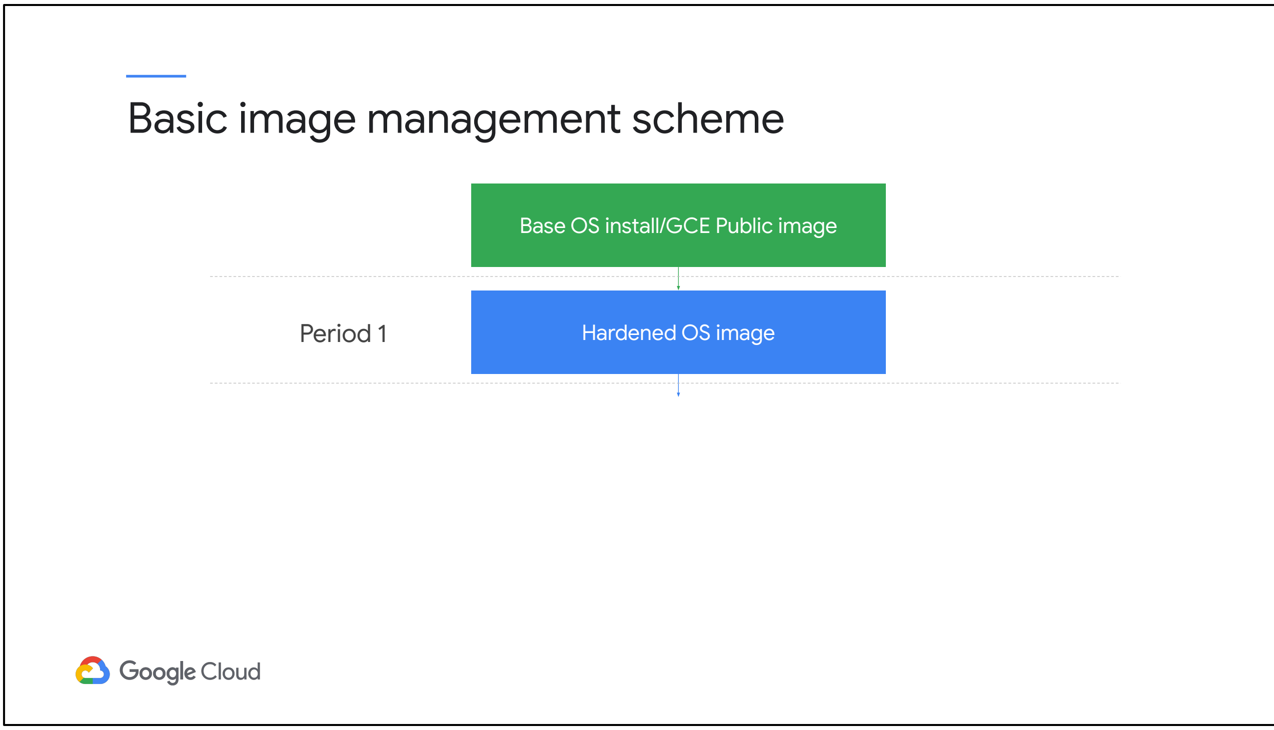
Some organizations are still in the habit of creating hand-crafted servers, and they have no existing support or process built around the idea of image automation.

Many organizations have some version of the second option. They have a set of images that they built manually or with partial automation, perhaps for particular workloads. They don't get built or updated often.

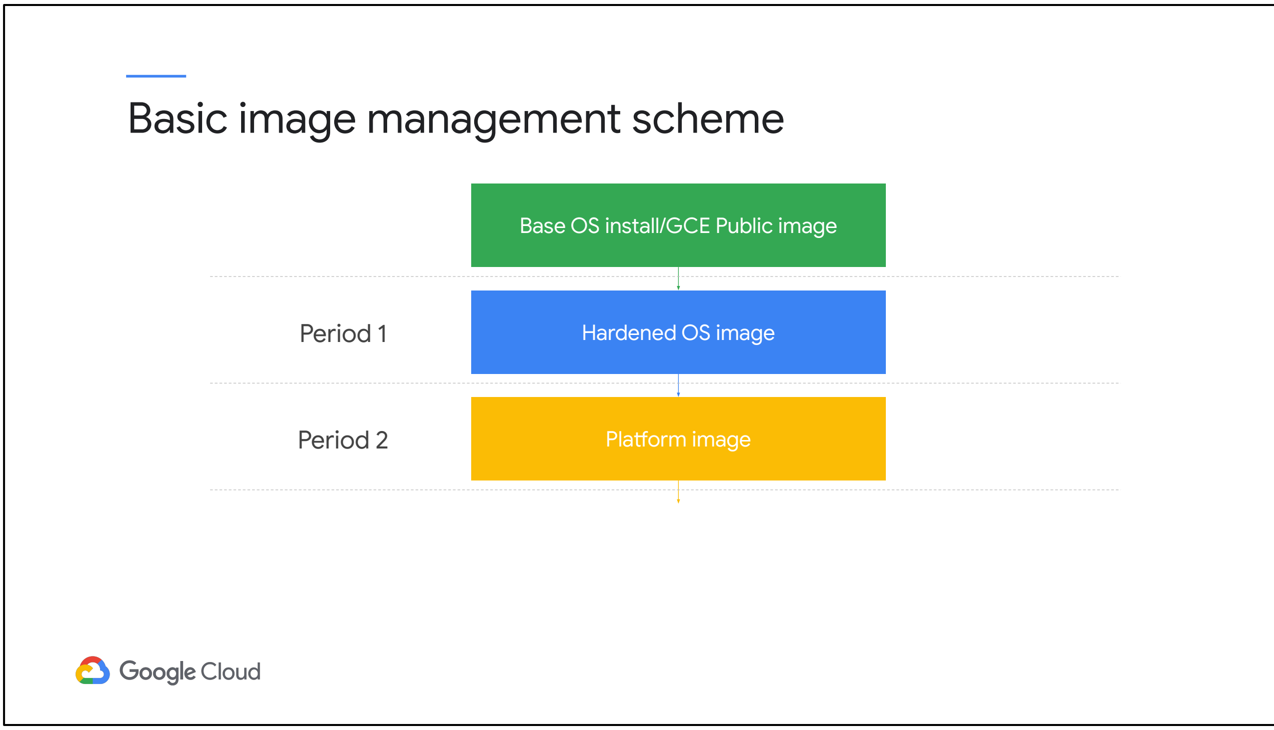
The goal is that organizations treat their image creation process as a standard DevOps pipeline. Commits to a codebase trigger build jobs, which create/test/deploy images with all requisite software and applications built-in, including the Logging and Monitoring agents.



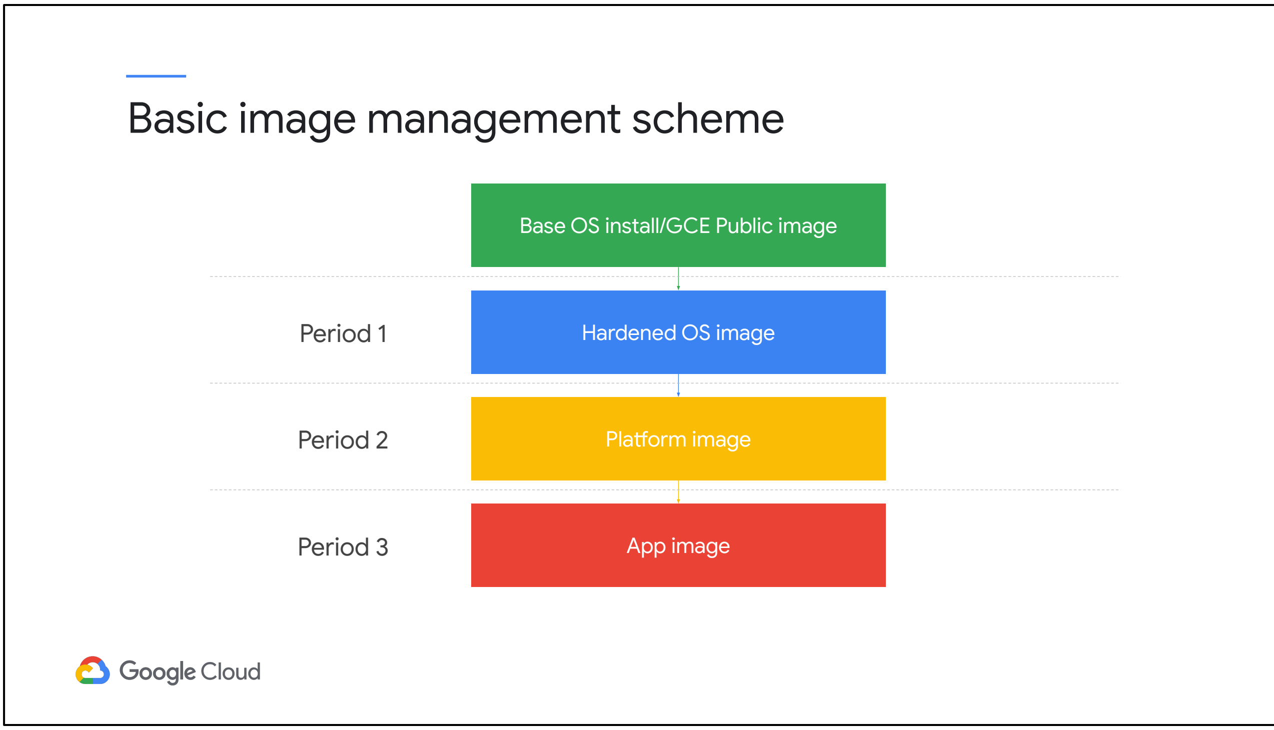
According to [Google's image management best practices](), image creation should start with a base OS installation, preferably from a golden (public) base.



Periodically, take the base image and have the security team harden it by removing services, changing settings, installing security components, etc. Build this hardened image every 90 days, or whatever frequency makes sense in the organization. This becomes the basis of subsequent builds.

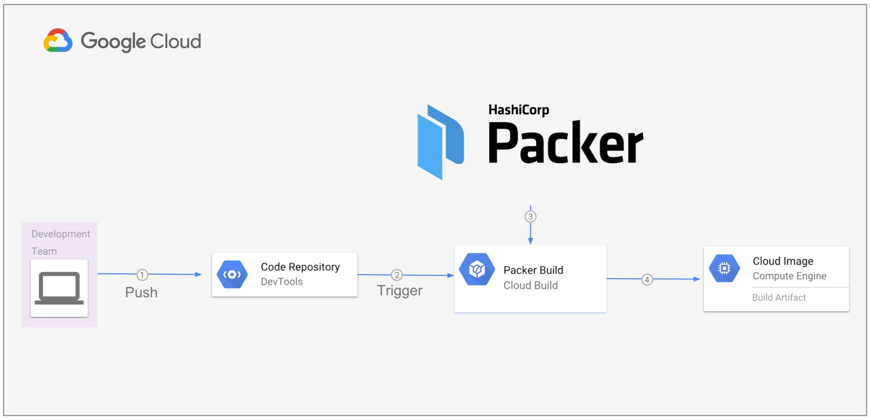


More frequently, build platform-specific images: one image for web servers, one for application servers, one for databases, etc. Build these images perhaps every 30 days. During this build process, you may wish to make a decision about including or excluding the Logging and Monitoring agents.



Finally, as frequently as you build an app, create new VM images for the new versions of the application. You might need to create new application images as often as once a day.

Packer can automate image builds



Google Cloud

HashiCorp's Packer is an open-source tool for creating virtual machine images. It integrates nicely with Google Cloud and can be used with Cloud Build to create images for Compute Engine. It really does a nice job of helping to automate image builds.

Leverage Agent Policies to aid with agent automation

- Automate installation and maintenance of Monitoring and Logging agents
- May apply to a fleet of VMs matching user-specified criteria
  - Current support for Linux VMs which support the agents

Here's an example policy that targets all CentOS 7 VMs with the labels *env=test* and *app=myproduct*

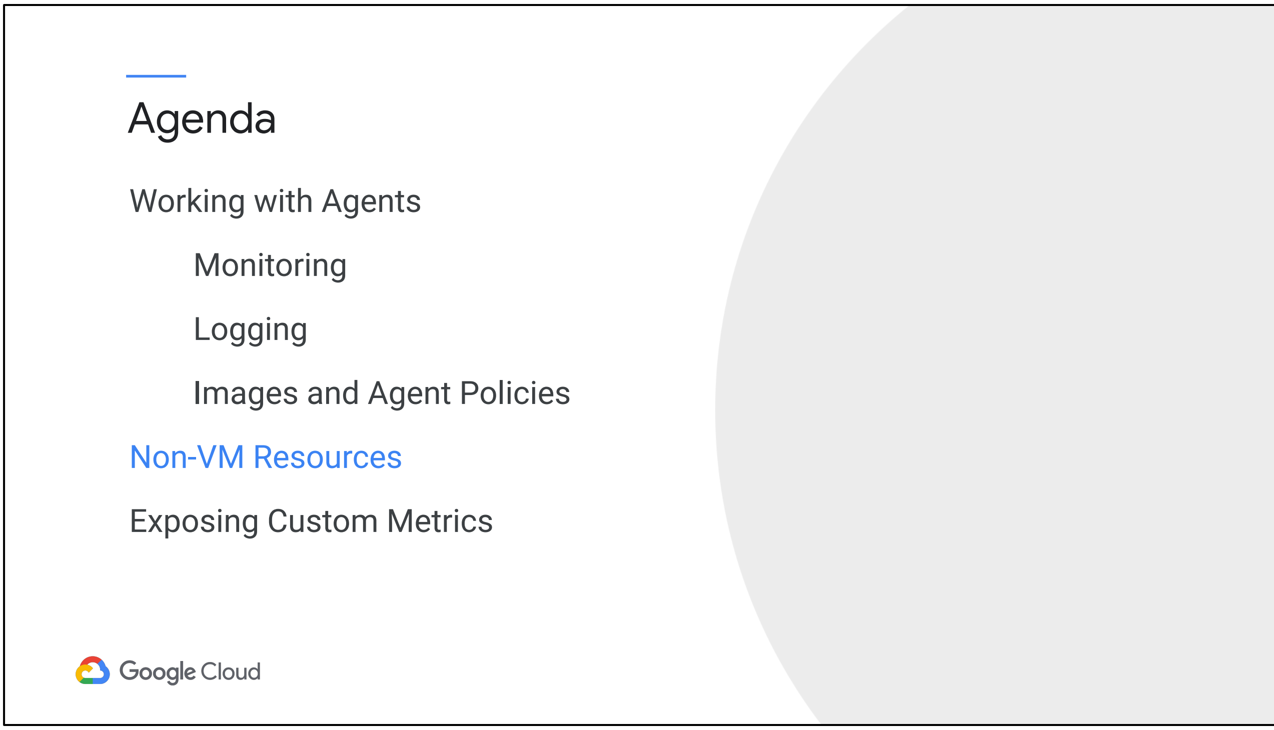
```
gcloud alpha compute instances ops-agents policies create ops-agents-policy-safe-rollout \
--agent-rules= "type=logging,version=current-major,package-state=installed,enable-autoupgrade=true; \
type=metrics,version=current-major, package-state=installed,enable-autoupgrade=true"    \
--os-types=short-name=centos,version=7  --group-labels=env=test,app=myproduct
```

Google Cloud

Agent Policies enable automated installation and maintenance of the Cloud Monitoring and Cloud Logging agents across a fleet of VMs that match user-specified criteria. With one command, you can create a Policy that governs existing and new VMs, ensuring proper installation and optional auto-upgrade of both agents.

To control the impact to production systems during rollout, it's recommended to use instance labels and zones to filter the instances that the Policy applies to.

Here is an example of creating a policy to target all VMs with the label *env=test* and *app=myproduct*.



In addition to Google Cloud virtual machines, there are a lot of Google Cloud resources that support some type of monitoring. Let's take a look at a few of these.

App Engine

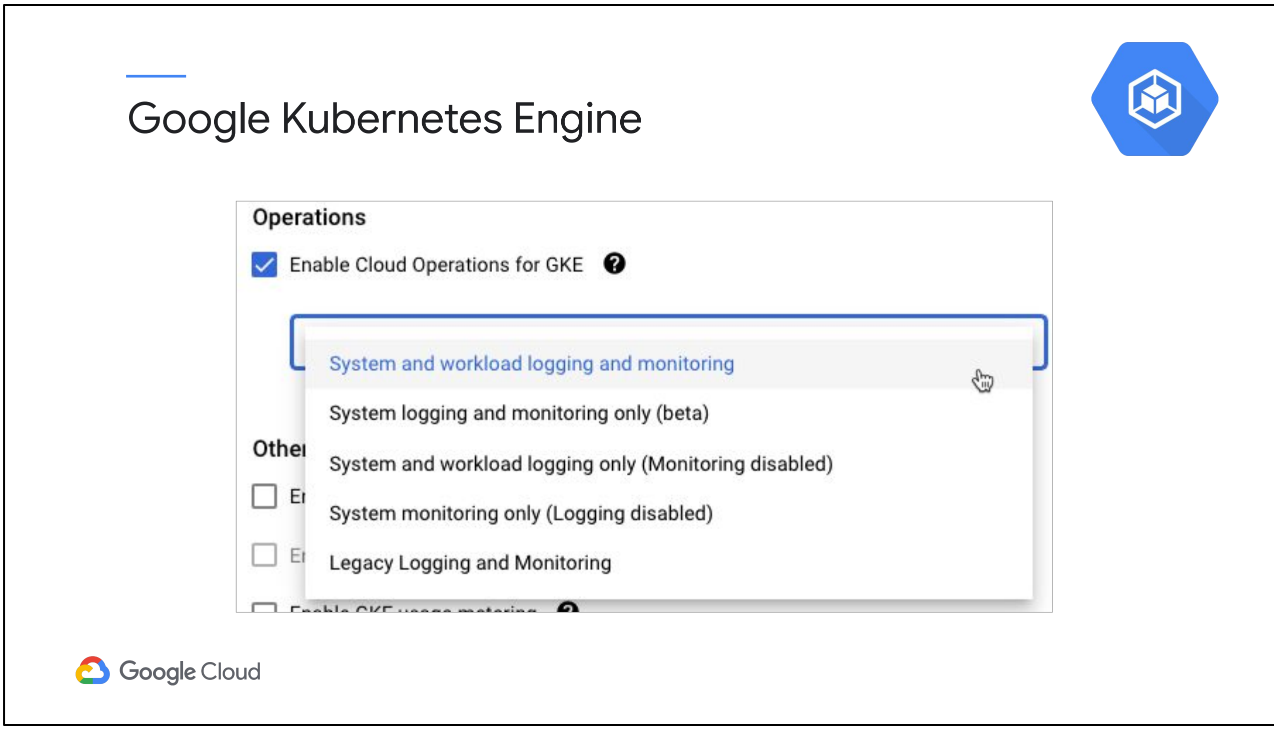
- Standard and Flex support Monitoring
  - Check documentation for metric details
- Standard and Flex support Logging
  - Write to stdout or stderr from code --> Best practice
  - May also use Logging APIs (like Winston on Node.js)
- Logs viewable under GAE Application resource

Google Cloud

Google's App Engine standard and flex both support monitoring. Make sure to check Google's documentation for the metric details.

They also both support logging by writing to standard out or error. For refined logging capabilities, check out the language-specific logging APIs, such as Winston for Node.js.

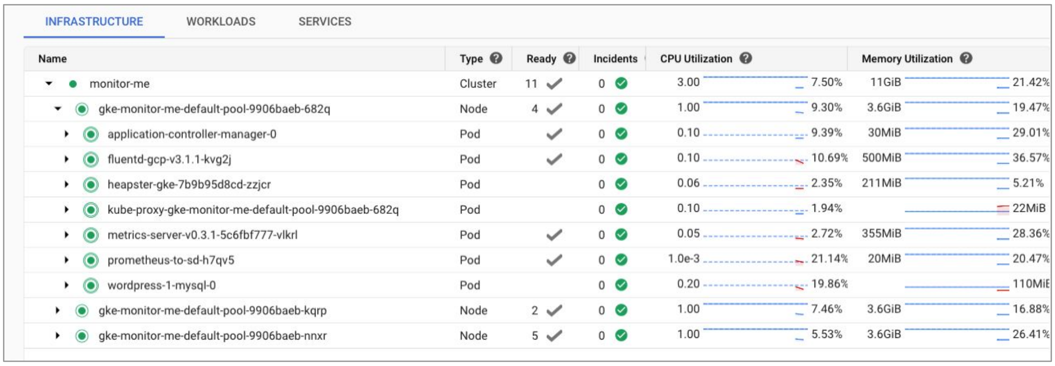
The logs are viewable under the GAE Application resource.



Google Kubernetes Engine supports several monitoring and logging configurations.

- GKE logging and monitoring integration can be disabled completely, though this will have an impact on Google's ability to support your cluster should problems arise.
- System and workload monitoring and logging can be enabled; this is currently the default, and Google's recommended best practice.
- In beta at the time of this writing, System logging and monitoring only (no workload) is an option. Logging data and monitoring metrics can incur spend. This option might lessen the cost by only capturing the system events. Think, "Someone created a service" (system) vs. "someone just visited the NGINX container in this pod" (workload).

GKE monitoring with the default dashboard



INFRASTRUCTURE | WORKLOADS | SERVICES

Name	Type	Ready	Incidents	CPU Utilization		Memory Utilization	
monitor-me	Cluster	11	0	3.00	7.50%	11GiB	21.42%
gke-monitor-me-default-pool-9906baeb-682q	Node	4	0	1.00	9.30%	3.6GiB	19.47%
application-controller-manager-0	Pod		0	0.10	9.39%	30MiB	29.01%
fluentd-gcp-v3.1.1-kvg2j	Pod		0	0.10	10.69%	500MiB	36.57%
heapster-gke-7b9b95d8cd-zzjcr	Pod		0	0.06	2.35%	211MiB	5.21%
kube-proxy-gke-monitor-me-default-pool-9906baeb-682q	Pod		0	0.10	1.94%		22MiB
metrics-server-v0.3.1-5c6fbf777-vlkrl	Pod		0	0.05	2.72%	355MiB	28.36%
prometheus-to-sd-h7qv5	Pod		0	1.0e-3	21.14%	20MiB	20.47%
wordpress-1-mysql-0	Pod		0	0.20	19.86%		110MiE
gke-monitor-me-default-pool-9906baeb-kqrp	Node	2	0	1.00	7.46%	3.6GiB	16.88%
gke-monitor-me-default-pool-9906baeb-nnxr	Node	5	0	1.00	5.53%	3.6GiB	26.41%

One of the benefits of GKE logging and monitoring is access to the Google created GKE Dashboard. The dashboard provides good resource visibility into your cluster from three different perspectives.

View the cluster from three perspectives

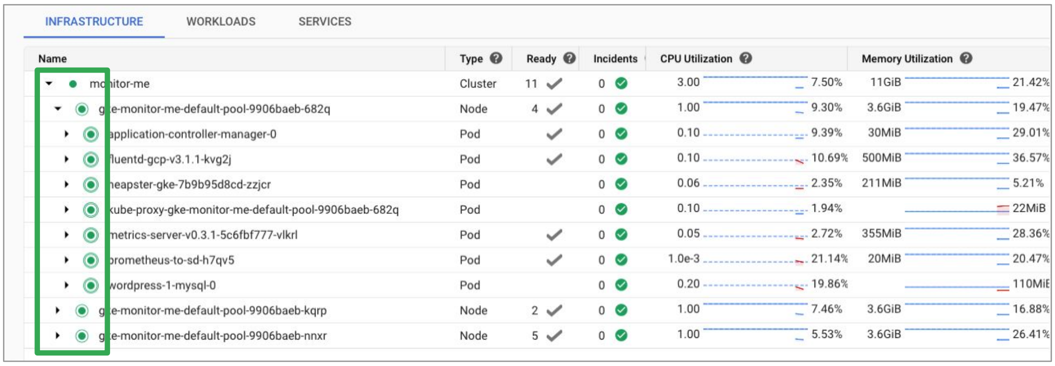
INFRASTRUCTURE | WORKLOADS | SERVICES

Name	Type	Ready	Incidents	CPU Utilization		Memory Utilization	
monitor-me	Cluster	11 ✓	0	3.00	7.50%	11GiB	21.42%
gke-monitor-me-default-pool-9906baeb-682q	Node	4 ✓	0	1.00	9.30%	3.6GiB	19.47%
application-controller-manager-0	Pod	✓	0	0.10	9.39%	30MiB	29.01%
fluentd-gcp-v3.1.1-kvg2j	Pod	✓	0	0.10	10.69%	500MiB	36.57%
heapster-gke-7b9b95d8cd-zzjcr	Pod		0	0.06	2.35%	211MiB	5.21%
kube-proxy-gke-monitor-me-default-pool-9906baeb-682q	Pod		0	0.10	1.94%		22MiB
metrics-server-v0.3.1-5c6fbf777-vlkrl	Pod	✓	0	0.05	2.72%	355MiB	28.36%
prometheus-to-sd-h7qv5	Pod	✓	0	1.0e-3	21.14%	20MiB	20.47%
wordpress-1-mysql-0	Pod		0	0.20	19.86%		110Mi
gke-monitor-me-default-pool-9906baeb-kqrp	Node	2 ✓	0	1.00	7.46%	3.6GiB	16.88%
gke-monitor-me-default-pool-9906baeb-nnxr	Node	5 ✓	0	1.00	5.53%	3.6GiB	26.41%

These three perspectives are:

- **Infrastructure**: which aggregates resources by **Cluster**, then **Node**, then **Pod**, and then by **Container**.
- **Workloads**: which aggregates resources by **Cluster**, then **Namespace**, then **Workload**, then **Pod**, and lastly by **Container**.
- **Services**: which aggregates resources by **Cluster**, then **Namespace**, then **Service**, then **Pod**, and lastly by **Container**.

Resources preceded by a status indicator

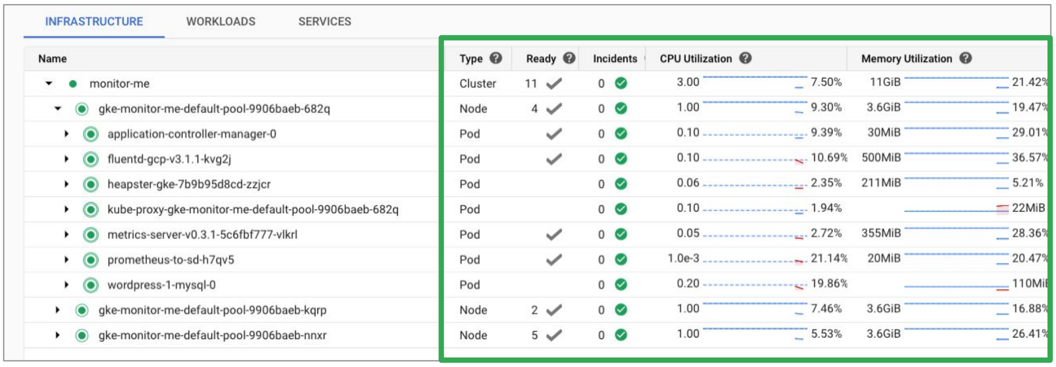


Name	Type	Ready	Incidents	CPU Utilization		Memory Utilization	
mo itor-me	Cluster	11	0	3.00	7.50%	11GiB	21.42%
g e-monitor-me-default-pool-9906baeb-682q	Node	4	0	1.00	9.30%	3.6GiB	19.47%
application-controller-manager-0	Pod		0	0.10	9.39%	30MiB	29.01%
fluentd-gcp-v3.1.1-kvg2j	Pod		0	0.10	10.69%	500MiB	36.57%
heapster-gke-7b9b95d8cd-zzjcr	Pod		0	0.06	2.35%	211MiB	5.21%
kube-proxy-gke-monitor-me-default-pool-9906baeb-682q	Pod		0	0.10	1.94%		22MiB
metrics-server-v0.3.1-5c6fbf777-vlkrl	Pod		0	0.05	2.72%	355MiB	28.36%
prometheus-to-sd-h7qv5	Pod		0	1.0e-3	21.14%	20MiB	20.47%
wordpress-1-mysql-0	Pod		0	0.20	19.86%		110MiB
g e-monitor-me-default-pool-9906baeb-kqrp	Node	2	0	1.00	7.46%	3.6GiB	16.88%
g e-monitor-me-default-pool-9906baeb-nnxr	Node	5	0	1.00	5.53%	3.6GiB	26.41%

Google Cloud

Each resource name in the list is preceded by a red or green indicator. A red indicator means that the resource, or a subcomponent of the resource, has an open incident. A green indicator means that there are no open incidents.

Status information provided for each GKE object



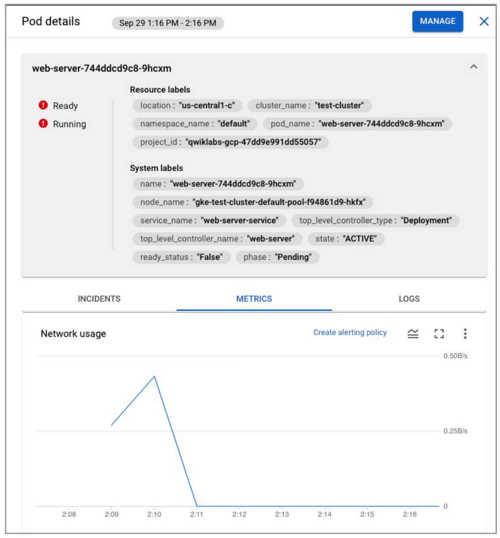
INFRASTRUCTURE WORKLOADS SERVICES

Name	Type	Ready	Incidents	CPU Utilization		Memory Utilization	
monitor-me	Cluster	11 ✓	0	3.00	7.50%	11GiB	21.42%
gke-monitor-me-default-pool-9906baeb-682q	Node	4 ✓	0	1.00	9.30%	3.6GiB	19.47%
application-controller-manager-0	Pod	✓	0	0.10	9.39%	30MiB	29.01%
fluentd-gcp-v3.1.1-kvg2j	Pod	✓	0	0.10	10.69%	500MiB	36.57%
heapster-gke-7b9b95d8cd-zzjcr	Pod		0	0.06	2.35%	211MiB	5.21%
kube-proxy-gke-monitor-me-default-pool-9906baeb-682q	Pod		0	0.10	1.94%		22MiB
metrics-server-v0.3.1-5c6fbf777-vlkrl	Pod	✓	0	0.05	2.72%	355MiB	28.36%
prometheus-to-sd-h7qv5	Pod	✓	0	1.0e-3	21.14%	20MiB	20.47%
wordpress-1-mysql-0	Pod		0	0.20	19.86%		110Mi
gke-monitor-me-default-pool-9906baeb-kqrp	Node	2 ✓	0	1.00	7.46%	3.6GiB	16.88%
gke-monitor-me-default-pool-9906baeb-nnxr	Node	5 ✓	0	1.00	5.53%	3.6GiB	26.41%

Google Cloud

Other columns display the Kubernetes object type, the number of pods, incidents, and percent of CPU and memory utilization as they relate to requested resources.

Select a pod to view details including metrics

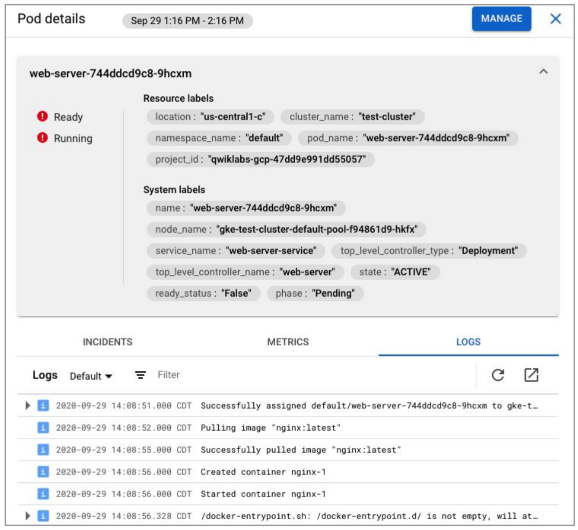


Google Cloud

Drilling down and then selecting a pod will display its details.

The **Metrics** tab includes status and label information, as well as charts for container restarts, CPU and memory utilization, networking, and storage.

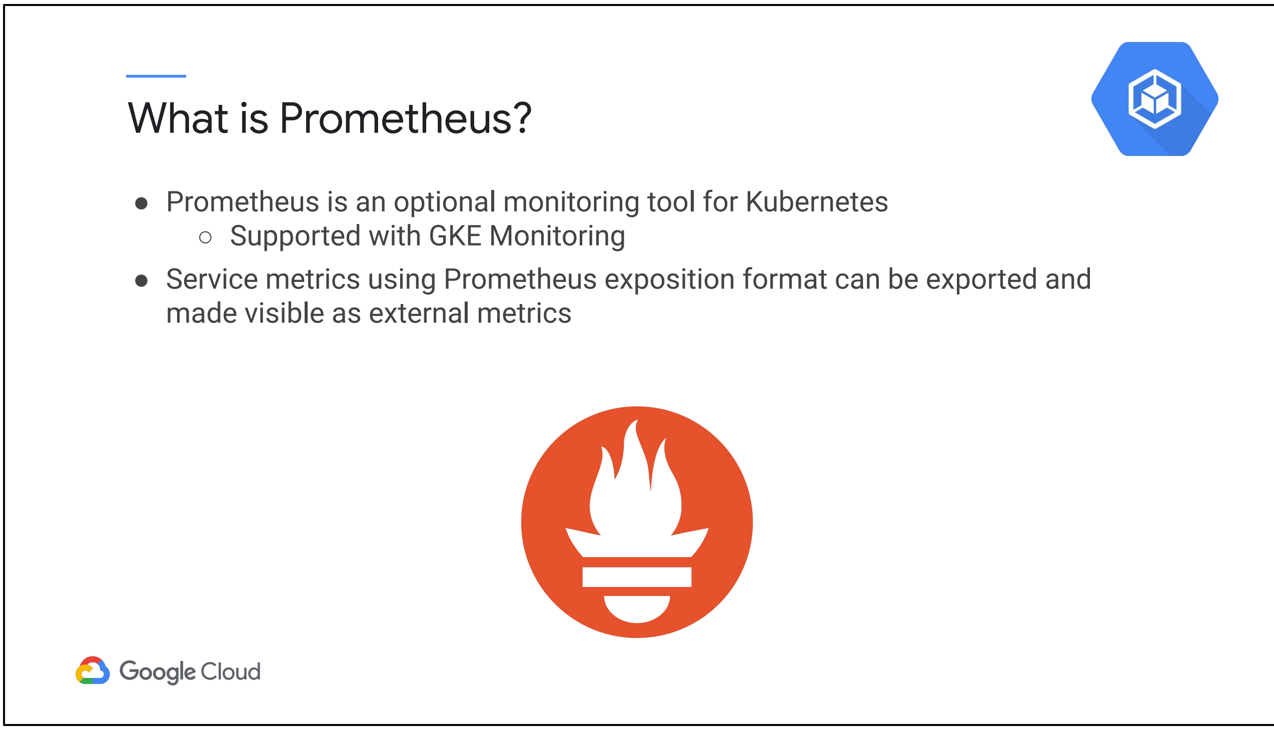
Logs tab displays the latest entries



Google Cloud

The **Logs** tab displays a logging rate histogram, the most recent log messages, and a link to the Google Cloud Log Viewer, which will auto-filter to the selected pod.

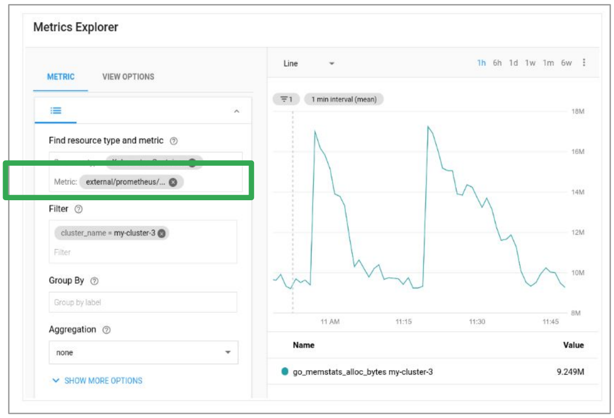
The **Details** tab contains pod information, including pod replica set type, namespace, and the node where the pod is running.



Prometheus is an open-source monitoring tool often used to extend Kubernetes' monitoring capabilities. If your cluster has standard Kubernetes Engine Monitoring enabled, you can add Prometheus support by installing the Prometheus server and collector. The Prometheus server scrapes your metrics and then exports them in Prometheus exposition format, through the collector, and then on to Google Cloud Monitoring.

Configure Prometheus for GKE

- Install Prometheus and the Collector
- Metrics can be viewed as external metrics
    - *external/prometheus/**

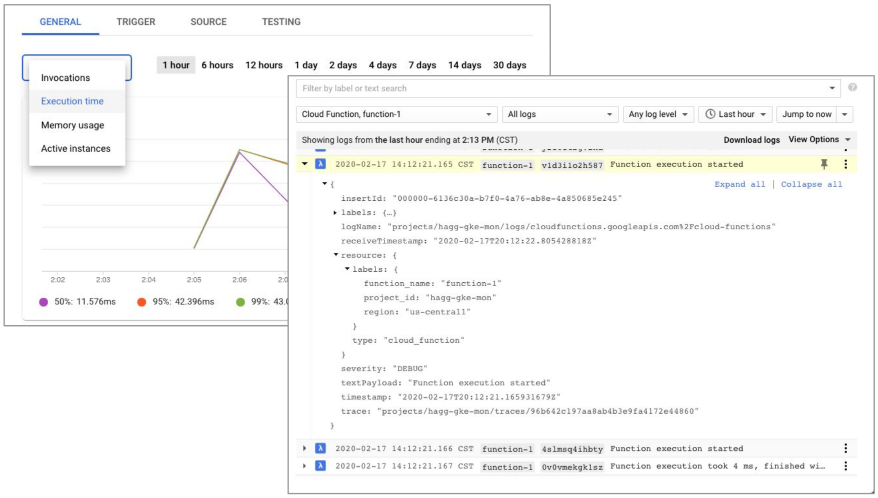


Google Cloud

Information on installing the Prometheus server can be found on the [Prometheus site](). Installing the collector, and accessing the metrics from Monitoring, is detailed in the [GKE documentation on Google]().

In Google Cloud Monitoring, the Prometheus-generated metric information will appear as *external/prometheus/** metrics.

Cloud Functions



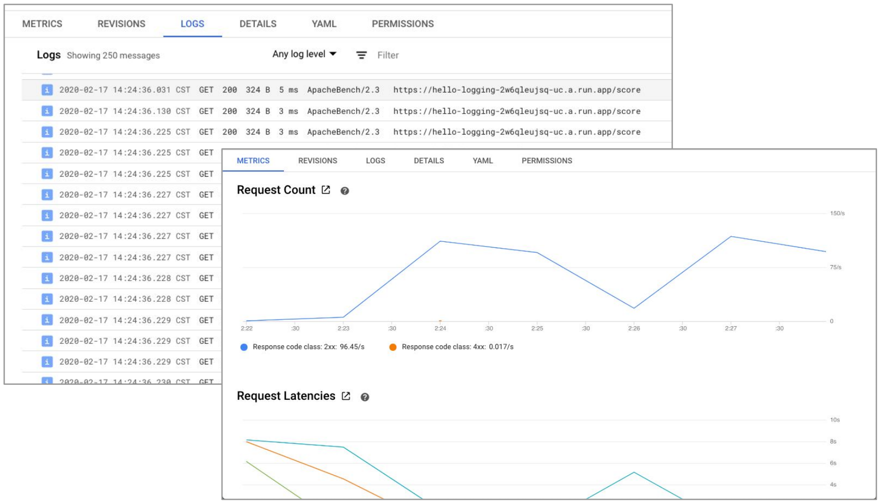
Google Cloud

Cloud Functions are lightweight, purpose-built functions, typically invoked in response to an event. For example, you might upload a PDF file to a Cloud Storage bucket, the new file triggers an event which invokes a Cloud Function, which translates the PDF from English to Spanish.

Cloud Functions monitoring is automatic and can provide you access to invocations, execution times, memory usage, and active instances in the Cloud Console. These metrics are also available in Cloud Monitoring, where you can set up custom alerting and dashboards for these metrics.

Cloud Functions also support simple logging by default. Logs written to standard out or standard error will appear automatically in the Cloud Console. The logging API can also be used to extend log support.

Cloud Run



Google Cloud

Cloud Run is Google's container service. It can run in a fully managed version, in which it acts as a sort of App Engine for containers, and it can also run on GKE, in which case it's a managed version of the open-source KNative. Cloud Run is automatically integrated with Cloud Monitoring **with no setup or configuration required**. This means that metrics of your Cloud Run services are captured automatically when they are running.

You can view metrics either in Cloud Monitoring or on the Cloud Run page in the console. Cloud Monitoring provides more charting and filtering options.

The resource type differs for fully managed Cloud Run and Cloud Run for Anthos:

- For fully managed Cloud Run, the monitoring resource name is "Cloud Run Revision" (*cloud_run_revision*).
- For Cloud Run for Anthos, the monitoring resource name is "Cloud Run on GKE Revision" (*knative_revision*).

Cloud Run has two types of logs which it automatically sends to Cloud Logging:

- Request logs: logs of requests sent to Cloud Run services.
- And Container logs: logs emitted from the container instances from your own code, written to standard out or standard error streams, or using the logging API.

Agenda

Working with Agents

- Monitoring
- Logging
- Images and Agent Policies

Non-VM Resources

Exposing Custom Metrics

Google Cloud

In addition to the more than 1,000 metrics that Google automatically collects, you can use code to create your own.

Exposing custom metrics

Two fundamental approaches:

- Use the Cloud Monitoring API
- Use OpenCensus

Google Cloud

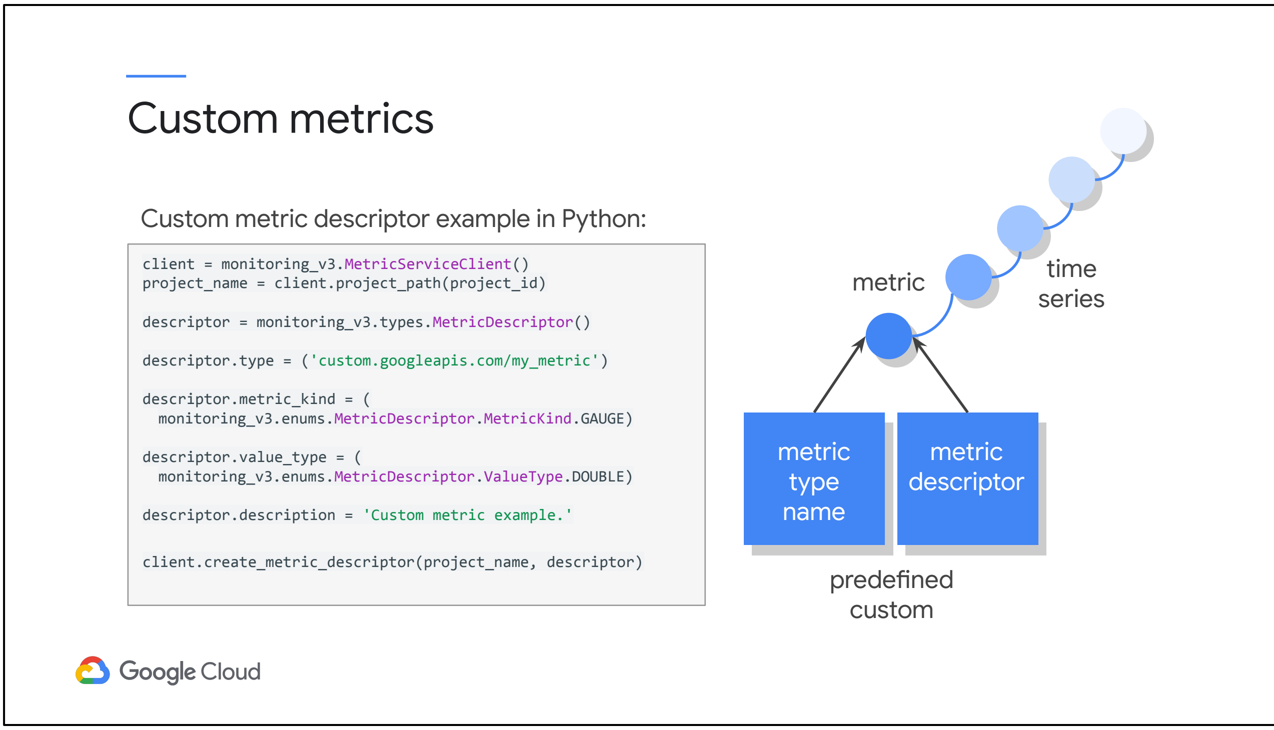
Application-specific metrics, also known as user or custom metrics, are metrics that you define and collect to capture information the built-in Cloud Monitoring metrics cannot. You capture such metrics by using an API provided by a library to instrument your code, and then you send the metrics to Cloud Monitoring.

Custom metrics can be used in the same way as built-in metrics. That is, you can create charts and alerts for your custom metric data.

There are two fundamental approaches to creating custom metrics for Google Cloud Monitoring:

- You can use the classic Cloud Monitoring API.
- Or you can use the OpenCensus open-source monitoring and tracing library.

Lecture Notes:
Google is changing from the old Stackdriver API to move towards OpenTelemetry

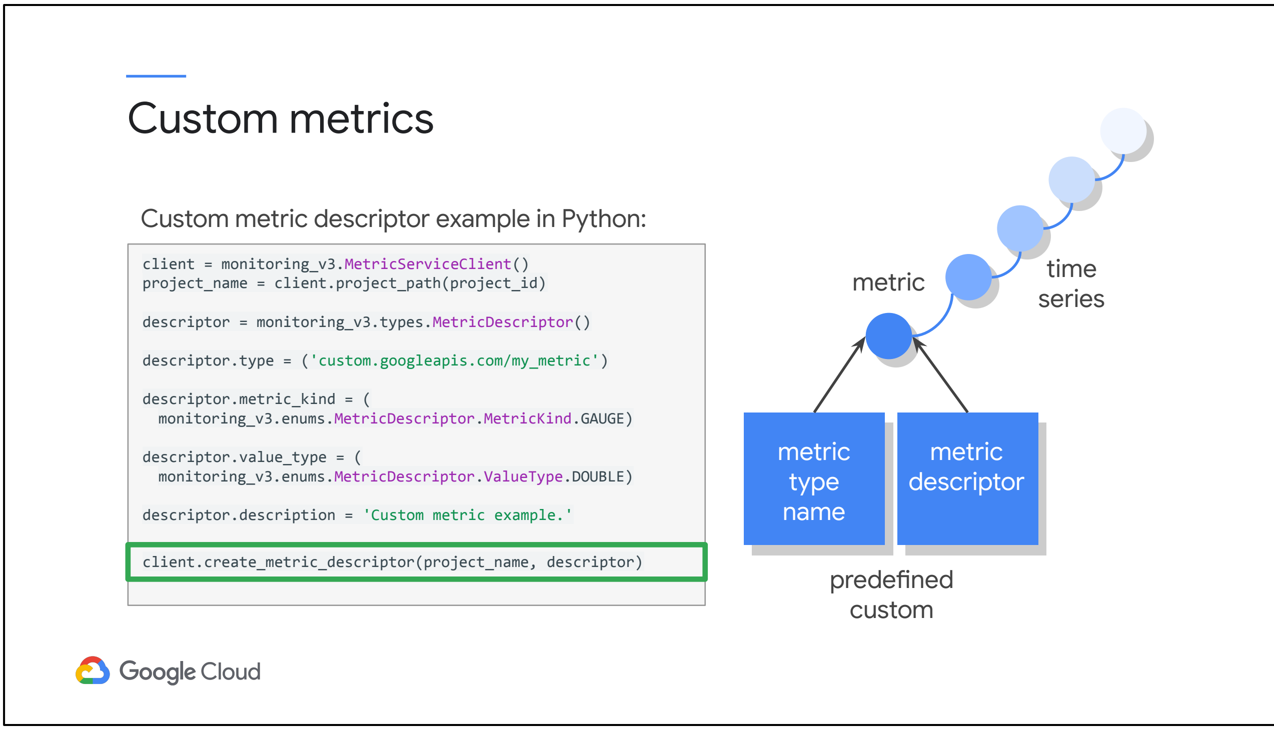


The steps used to create a custom metric using the API are [well documented].

To begin, the data you collect for a custom metric must be associated with a descriptor for a custom metric type.

Do you remember the example we saw earlier in this course, of the documentation for the cloud storage request count metric?

That's what we're creating here for our custom metric.

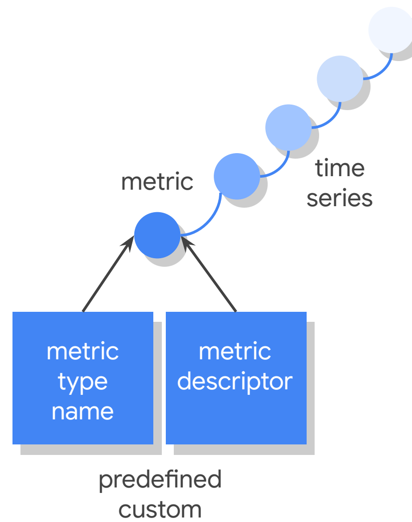


After you have collected the information you need for creating your custom metric type, call the create method, passing into a MetricDescriptor object.

Custom metrics

Custom metric descriptor example in Python:

```
client = monitoring_v3.MetricServiceClient()
project_name = client.project_path(project_id)

descriptor = monitoring_v3.types.MetricDescriptor()
descriptor.type = ('custom.googleapis.com/my_metric')

descriptor.metric_kind = (
  monitoring_v3.enums.MetricDescriptor.MetricKind.GAUGE)

descriptor.value_type = (
  monitoring_v3.enums.MetricDescriptor.ValueType.DOUBLE)
descriptor.description = 'Custom metric example.'

client.create_metric_descriptor(project_name, descriptor)
```



Google Cloud

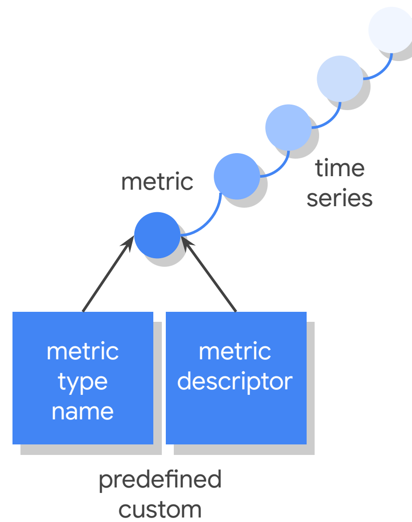
In this example, we are creating a gauge double metric named *my_metric*.

It's a gauge metric of type double, with the description "Custom metric example."

Writing metrics

Using our custom metric, again in Python:

```
client = monitoring_v3.MetricServiceClient()
project_name = client.project_path(project_id)

series = monitoring_v3.types.TimeSeries()
series.metric.type = ('custom.googleapis.com/my_metric’)
series.resource.type = 'gce_instance'
series.resource.labels['instance_id']='1267890123456789'
series.resource.labels['zone'] = 'us-central1-f'

point = series.points.add()
point.value.double_value = 3.14
now = time.time()
point.interval.end_time.seconds = int(now)

client.create_time_series(project_name, [series])
```

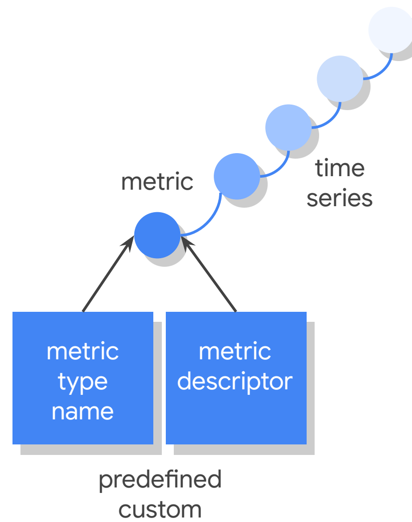


You write data points by passing a list of [TimeSeries] objects to *create_time_series*.

Writing metrics

Using our custom metric, again in Python:

```
client = monitoring_v3.MetricServiceClient()
project_name = client.project_path(project_id)

series = monitoring_v3.types.TimeSeries()
series.metric.type = ('custom.googleapis.com/my_metric')
series.resource.type = 'gce_instance'
series.resource.labels['instance_id']='1267890123456789'
series.resource.labels['zone'] = 'us-central1-f'

point = series.points.add()
point.value.double_value = 3.14
now = time.time()
point.interval.end_time.seconds = int(now)

client.create_time_series(project_name, [series])
```



Google Cloud

Each time series is identified by the metric and resource fields of the TimeSeries object.

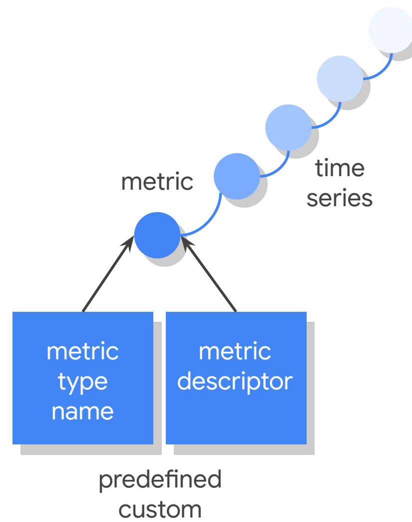
These fields represent the metric type and the monitored resource from which the data was collected.

In this example, we are using the *my_metric* described on the last slide and linking our metric to the specified Compute Engine instance.

Writing metrics

Using our custom metric, again in Python:

```
client = monitoring_v3.MetricServiceClient()
project_name = client.project_path(project_id)

series = monitoring_v3.types.TimeSeries()
series.metric.type = ('custom.googleapis.com/my_metric’)
series.resource.type = 'gce_instance'
series.resource.labels['instance_id']='1267890123456789'
series.resource.labels['zone'] = 'us-central1-f'

point = series.points.add()
point.value.double_value = 3.14
now = time.time()
point.interval.end_time.seconds = int(now)

client.create_time_series(project_name, [series])
```



Google Cloud

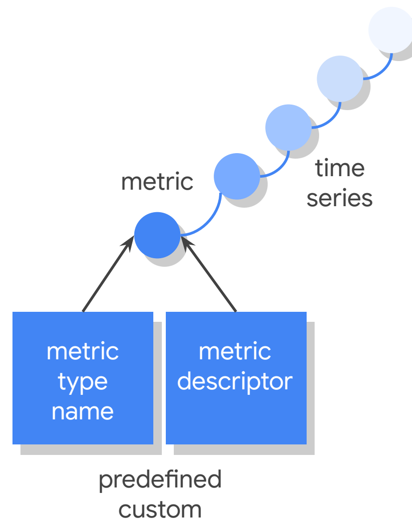
Next, we create the point by adding it to the series and adding the details.

Each [TimeSeries] object must contain a single [Point] object.

Writing metrics

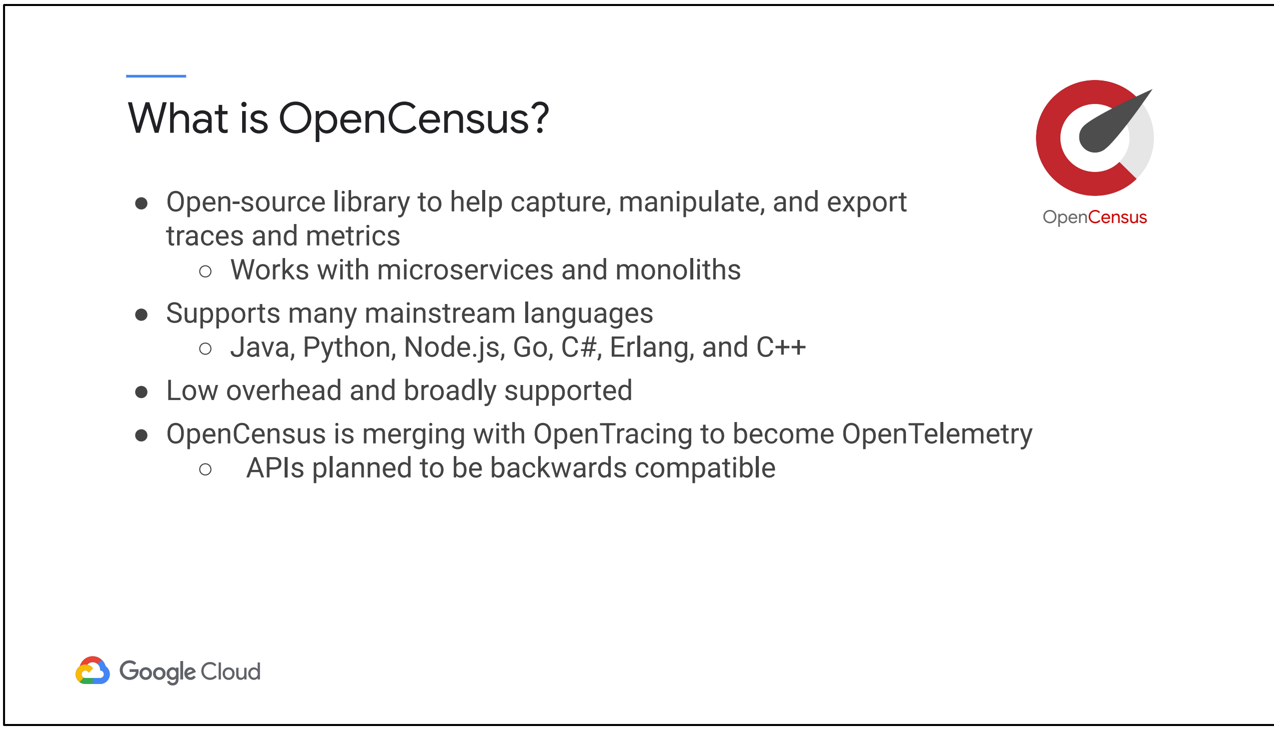
Using our custom metric, again in Python:

```
client = monitoring_v3.MetricServiceClient()
project_name = client.project_path(project_id)

series = monitoring_v3.types.TimeSeries()
series.metric.type = ('custom.googleapis.com/my_metric’)
series.resource.type = 'gce_instance'
series.resource.labels['instance_id']='1267890123456789'
series.resource.labels['zone'] = 'us-central1-f'

point = series.points.add()
point.value.double_value = 3.14
now = time.time()
point.interval.end_time.seconds = int(now)

client.create_time_series(project_name, [series])
```



Google Cloud

Finally, we report our metric.



OpenCensus is an open-source library to help capture, manipulate, and export traces and metrics. It works well with microservices and monoliths.

Support is available for a wide variety of languages, including Java, Python, Node.js, Go, C#, Erlang, and C++.

OpenCensus is low overhead and is broadly supported by various environments and back-end applications, including Cloud Monitoring.

Metrics expressed as measures and measurements

- A Measure represents a metric being recorded
  - **Name**: unique identifier
  - **Description**: purpose of the measure
  - **Unit**: string unit specifier, like “By”, “1”, or “ms”
    - Unit codes
  - Two measure value types: Int64 or a Float64
- A Measurement is a data point recorded as a Measure

Google Cloud

OpenCensus metrics are expressed as Measures and Measurements.

A **Measure** represents a metric being recorded. It contains:

- Name: a unique identifier.
- Description: of the purpose for the measure.
- Unit: a string unit specifier, like “By”, “1”, or “ms”.
- A data type of: Int64 or a Float64.

A **Measurement** is a data point recorded as a Measure.

Views describe how measurements are collected

- A View represents the coupling of an Aggregation applied to a Measure and optionally Tags
- They contain:
  - Name: unique view name
  - Description
  - Measure: Measurement type
  - TagKeys: tagkeys used to group and filter metrics
  - Aggregation: How is the data gathered
    - Count, Distribution, Sum, or LastValue

Views describe how Measurements are collected.

A **View** represents the coupling of an Aggregation applied to a Measure and optionally, Tags.

They contain:

- Name: a unique view name.
- Description.
- Measure: Discussed on the last slide.
- TagKeys: which can be used to group and filter metrics.
- Aggregation: describes how is the data is gathered; options include Count, Distribution, Sum, or LastValue.

Load required libraries

```
const {globalStats, MeasureUnit, AggregationType} =
require('@opencensus/core');
const {StackdriverStatsExporter} =
require('@opencensus/exporter-stackdriver');

const EXPORT_INTERVAL = 60;
const LATENCY_MS = globalStats.createMeasureInt64(
  'task_latency',
  MeasureUnit.MS,  'The task latency in milliseconds'
);
```

Google Cloud

Over the next several slides, we'll explore a quick OpenCensus example created using Node.js. This example can be found in the [Google documentation for OpenCensus](#).

Load required libraries

```
const {globalStats, MeasureUnit, AggregationType} =
require('@opencensus/core');
const {StackdriverStatsExporter} =
require('@opencensus/exporter-stackdriver');

const EXPORT_INTERVAL = 60;
const LATENCY_MS = globalStats.createMeasureInt64(
  'task_latency',
  MeasureUnit.MS,  'The task latency in milliseconds'
);
```

Google Cloud

First, we load our core OpenCensus library and the Stackdriver exporter.

Load required libraries

```
const {globalStats, MeasureUnit, AggregationType} =
require('@opencensus/core');
const {StackdriverStatsExporter} =
require('@opencensus/exporter-stackdriver');

const EXPORT_INTERVAL = 60;
const LATENCY_MS = globalStats.createMeasureInt64(
  'task_latency',
  MeasureUnit.MS,  'The task latency in milliseconds'
);
```

Google Cloud

Next, we create a couple of variables containing the export interval and an Int64 millisecond measure named *task_latency*.

Set up the view

```
// Register the view or Metrics will be dropped
const view = globalStats.createView(
  'task_latency_distribution',
  LATENCY_MS,
  AggregationType.DISTRIBUTION,
  [], // Tags
  'The distribution of the task latencies.',
  // Latency in buckets:
  // [>=0ms, >=100ms, >=200ms, >=400ms, >=1s, >=2s, >=4s]
  [0, 100, 200, 400, 1000, 2000, 4000]
);

globalStats.registerView(view);
```

Google Cloud

Now, we register the view. If the view isn't registered, the metrics we create later will be dropped and won't ever show up in Monitoring.

Set up the view

```
// Register the view or Metrics will be dropped
const view = globalStats.createView(
  'task_latency_distribution',
  LATENCY_MS,
  AggregationType.DISTRIBUTION,
  [], // Tags
  'The distribution of the task latencies.',
  // Latency in buckets:
  // [>=0ms, >=100ms, >=200ms, >=400ms, >=1s, >=2s, >=4s]
  [0, 100, 200, 400, 1000, 2000, 4000]
);

globalStats.registerView(view);
```

Google Cloud

This particular view is named *task_latency_distribution*, and that’s what you would search for in the Metrics Explorer.

Set up the view

```
// Register the view or Metrics will be dropped
const view = globalStats.createView(
  'task_latency_distribution',
  LATENCY_MS,
  AggregationType.DISTRIBUTION,
  [], // Tags
  'The distribution of the task latencies.',
  // Latency in buckets:
  // [>=0ms, >=100ms, >=200ms, >=400ms, >=1s, >=2s, >=4s]
  [0, 100, 200, 400, 1000, 2000, 4000]
);
globalStats.registerView(view);
```

Google Cloud

The view is reporting a distribution of latencies using the specified bucket ranges.

Configure the exporter

```
if (!projectId || !process.env.GOOGLE_APPLICATION_CREDENTIALS) {
  throw Error('Unable to proceed without a Project ID');
}

// The minimum reporting period is 1 minute.
const exporter = new StackdriverStatsExporter({
  projectId: projectId,
  period: EXPORT_INTERVAL * 1000,
});

// Pass the created exporter to Stats
globalStats.registerExporter(exporter);
```

Google Cloud

Now, we verify that we have the project id and a Google Cloud service account key file,

Configure the exporter

```
if (!projectId || !process.env.GOOGLE_APPLICATION_CREDENTIALS) {
  throw Error('Unable to proceed without a Project ID');
}

// The minimum reporting period is 1 minute.
const exporter = new StackdriverStatsExporter({
  projectId: projectId,
  period: EXPORT_INTERVAL * 1000,
});

// Pass the created exporter to Stats
globalStats.registerExporter(exporter);
```

Google Cloud

And then enable OpenCensus to report through the exporter for Stackdriver. The export interval is set to the minimum, (60 * 1000) one minute.

Record the measurements

```
// Record 1000 fake latency values between 0 and 5 seconds.
for (let i = 0; i < 1000; i++) {
  const ms = Math.floor(Math.random() * 5);
  console.log(`Latency ${i}: ${ms}`);
  globalStats.record([
    {
      measure: LATENCY_MS,
      value: ms,
    },
  ]);
}
```

Google Cloud

With all the setup out of the way, we can now record measurements through our view.

Record the measurements

```
// Record 1000 fake latency values between 0 and 4 seconds.
for (let i = 0; i < 1000; i++) {
  const ms = Math.floor(Math.random() * 5);
  console.log(`Latency ${i}: ${ms}`);
  globalStats.record([
    {
      measure: LATENCY_MS,
      value: ms,
    },
  ]);
}
```

Google Cloud

The code seen here generates a series of 1,000 random values in the range 0-4.

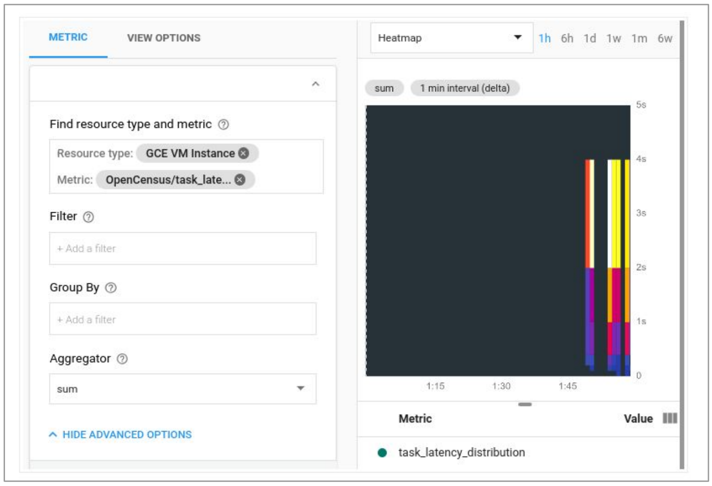
Record the measurements

```
// Record 1000 fake latency values between 0 and 5 seconds.
for (let i = 0; i < 1000; i++) {
  const ms = Math.floor(Math.random() * 5);
  console.log(`Latency ${i}: ${ms}`);
  globalStats.record([
    {
      measure: LATENCY_MS,
      value: ms,
    },
  ]);
}
```

Google Cloud

For each one, it records a measurement.

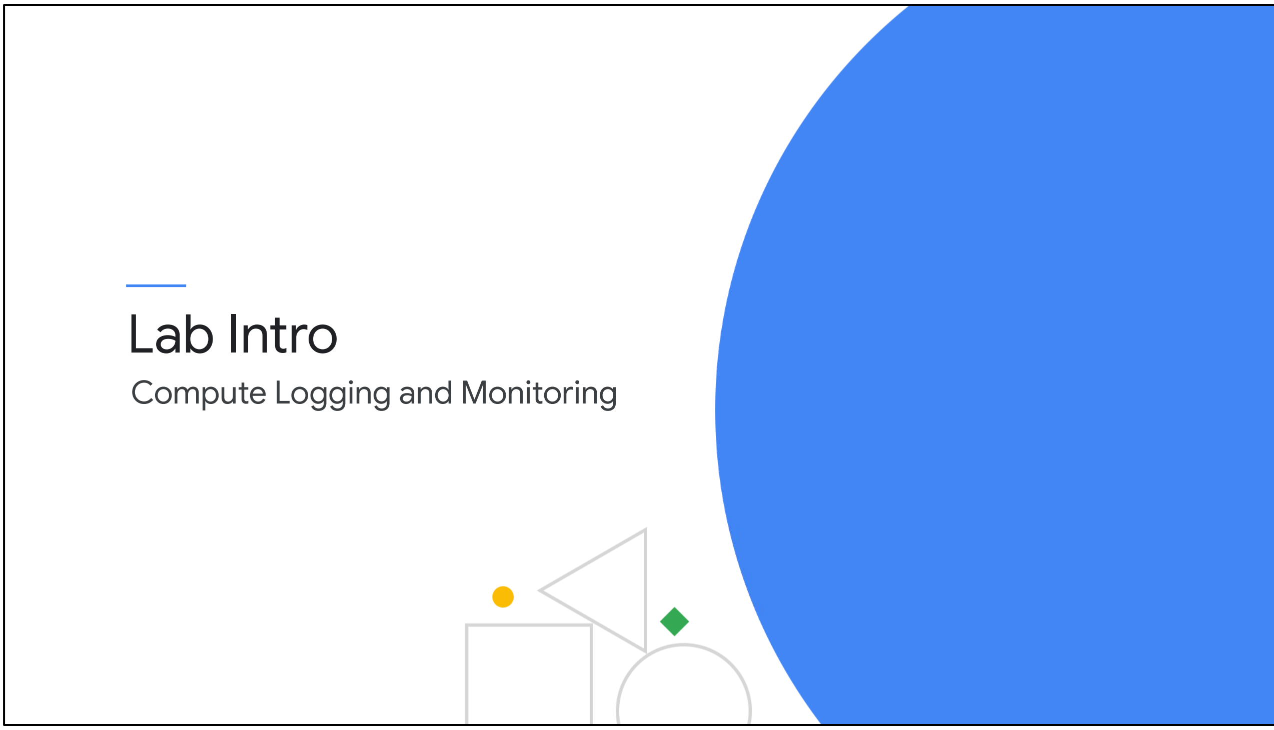
Exposing metrics from GCE using OpenCensus



Google Cloud

Running the example code on a Compute Engine VM yielded the pictured results. Notice the metric name, "OpenCensus/task_latency_distribution."

Each bar in the heatmap represents one run of the program, and the colored components of each bar represent buckets in the latency distribution.



Google Cloud has a number of compute-related resources, including Compute Engine, Kubernetes, and Cloud Run, just to name a few. In this lab, you install the Logging and Monitoring agents into a VM running an NGINX server to maximize the metrics we can easily view. You also set up a GKE cluster and monitor an application deployed into it.

Quiz

You’ve deployed an application to Compute Engine. The application writes to the logs, but you can’t find any of the logged messages in Cloud Logging. What might be the problem?

A. You need to allow access to all Google Cloud services under scopes
B. You need to turn off the VM machine logs
C. You need to install the Logging agent
D. All of the above

Quiz

You’ve deployed an application to Compute Engine. The application writes to the logs, but you can’t find any of the logged messages in Cloud Logging. What might be the problem?

A. You need to allow access to all Google Cloud services under scopes
B. You need to turn off the VM machine logs
C. You need to install the Logging agent
D. All of the above

Quiz

Management wants to see analysis of resources by development team, department, cost center, and application status. What could you do to make this easier?

A. Add appropriate labels to your Google Cloud resources
B. Add appropriate tags to your Google Cloud resources
C. Used standardized prefixes on the names of all resources
D. Use customized logging messages that include appropriate resource metadata

Quiz

Management wants to see analysis of resources by development team, department, cost center, and application status. What could you do to make this easier?

A. Add appropriate labels to your Google Cloud resources
B. Add appropriate tags to your Google Cloud resources
C. Used standardized prefixes on the names of all resources
D. Use customized logging messages that include appropriate resource metadata

Quiz

Which statement below is false?

A. Workspaces can include resources from multiple projects
B. Workspaces can include AWS resources
C. Each project can be in only one workspace
D. You can allow other users to view your workspace using IAM

Google Cloud

Quiz

Which statement below is false?

A. Workspaces can include resources from multiple projects
B. Workspaces can include AWS resources
C. Each project can be in only one workspace
D. You can allow other users to view your workspace using IAM

Learned how to...

- Integrate Logging and Monitoring agents into Compute Engine VMs and images
- Enable and utilize Kubernetes Monitoring
- Extend and clarify Kubernetes Monitoring with Prometheus
- Expose custom metrics through code, and with the help of OpenCensus

Google Cloud

Well done. In this module, you learned how to:

- Integrate Logging and Monitoring agents into Compute Engine VMs and images.
- Enable and utilize Kubernetes Monitoring.
- Extend and clarify Kubernetes Monitoring with Prometheus.
- Expose custom metrics through code, and with the help of OpenCensus.

Great job!

Google Cloud