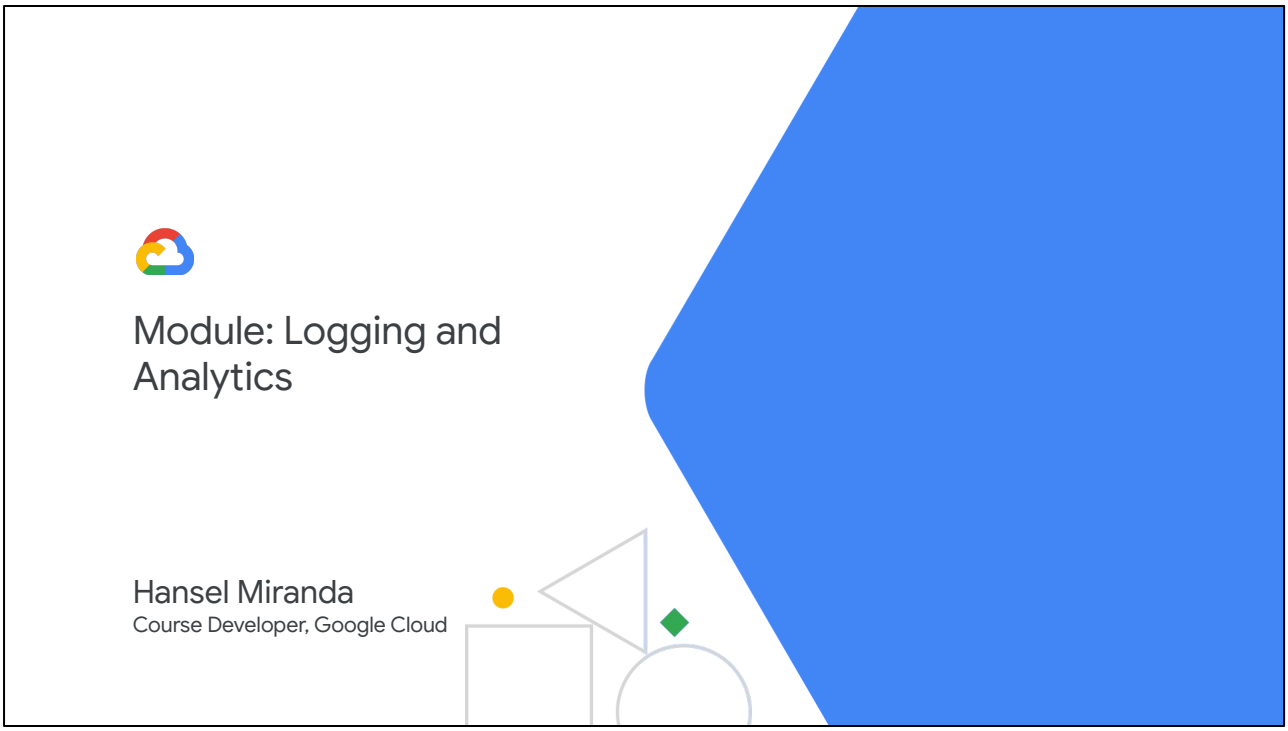
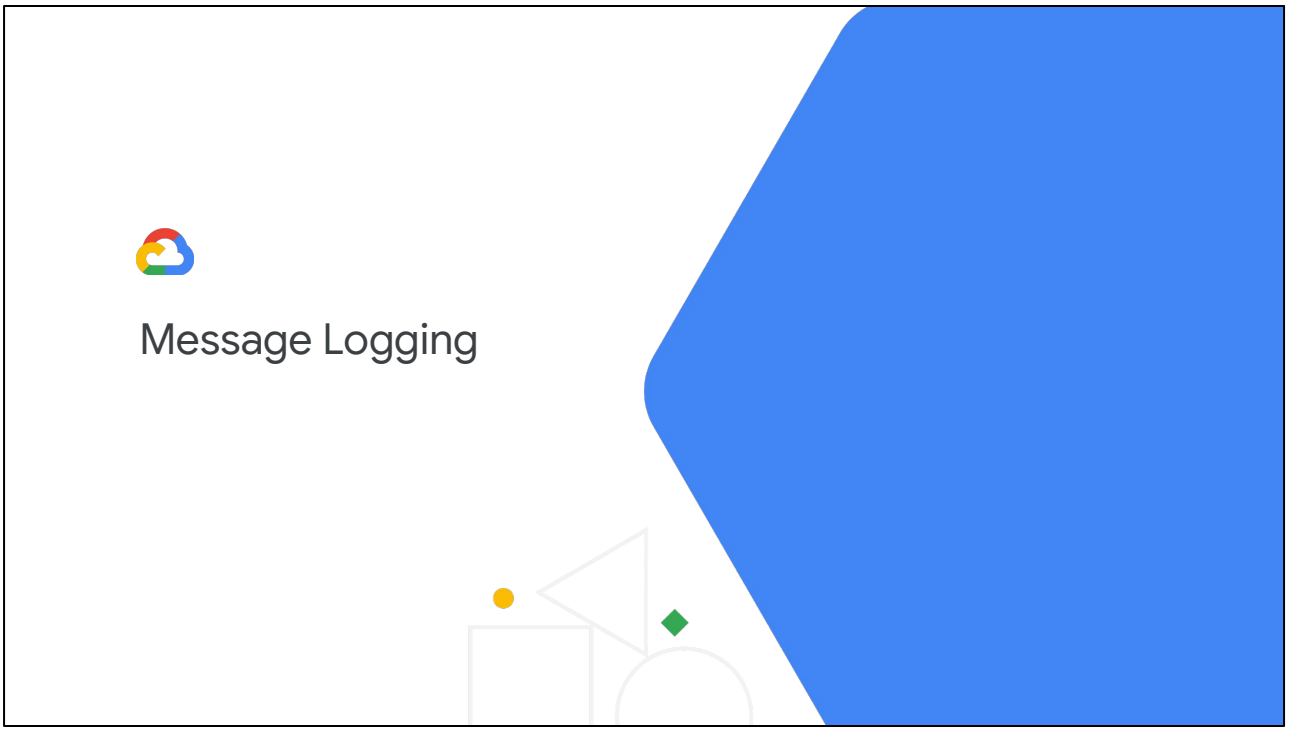


In this module, you'll learn how to use the MessageLogging policy to log messages in Apigee proxies.

You'll also learn about Apigee's Analytics, which you can use to gain insights into your API program and track API performance, as well as the DataCapture policy, which can be used to save custom data for use in Analytics reports.



This lecture will discuss how we write to logs from API proxies using the MessageLogging policy.

Message logging

- Use to log events that occur during the handling of an API call
- MessageLogging policy uses Syslog protocol
- Can log to Cloud Logging using ServiceCallout to call REST API

Logging messages is one of the best ways to track down issues in your APIs.

You can use a MessageLogging policy within an API proxy. This policy can be used to log errors that occur during the handling of an API call.

The MessageLogging policy can send a custom log entry to a remote Syslog server. Syslog is a standard for message logging. You can set up your own syslog server or use a third-party syslog server.

You can also log messages into Cloud Logging. Cloud Logging provides a REST API, and you can call that API using a ServiceCallout policy.

MessageLogging policy

```
<MessageLogging continueOnError="true" enabled="true" name="LOG-LogMessage">
  <Syslog>
    <Message>{organization.name}.{apiproxy.name}.{environment.name} | {error.status.code}
{error.content}</Message>
    <Host>syslog.example.com</Host>
    <Port>2900</Port>
    <Protocol>TCP</Protocol>
    <SSLInfo>
        <Enabled>true</Enabled>
    </SSLInfo>
  </Syslog>
</MessageLogging>
```

Here is an example of a MessageLogging policy using syslog.

The first thing to notice is that continueOnError is set to true. It generally does not make sense to throw an error if your message logging policy fails, because you want the API call to work even if you can't log a message.

When a policy is added, continueOnError always defaults to false. It is a best practice to set it to true for the MessageLogging policy.

MessageLogging policy

```
<MessageLogging continueOnError="true" enabled="true" name="LOG-LogMessage">
  <Syslog>
    <Message>{organization.name}.{apiproxy.name}.{environment.name} | {error.status.code}
{error.content}</Message>
    <Host>syslog.example.com</Host>
    <Port>2900</Port>
    <Protocol>TCP</Protocol>
    <SSLInfo>
        <Enabled>true</Enabled>
    </SSLInfo>
  </Syslog>
</MessageLogging>
```

The Syslog element contains the information necessary for sending the log entry using the syslog message logging standard.

MessageLogging policy

```
<MessageLogging continueOnError="true" enabled="true" name="LOG-LogMessage">
  <Syslog>
    <Message>{organization.name}.{apiproxy.name}.{environment.name} | {error.status.code}
{error.content}</Message>
    <Host>syslog.example.com</Host>
    <Port>2900</Port>
    <Protocol>TCP</Protocol>
    <SSLInfo>
        <Enabled>true</Enabled>
    </SSLInfo>
  </Syslog>
</MessageLogging>
```

The Message element contains the custom log message.

Variables can be used in the message by using curly braces around the variable names.

MessageLogging policy

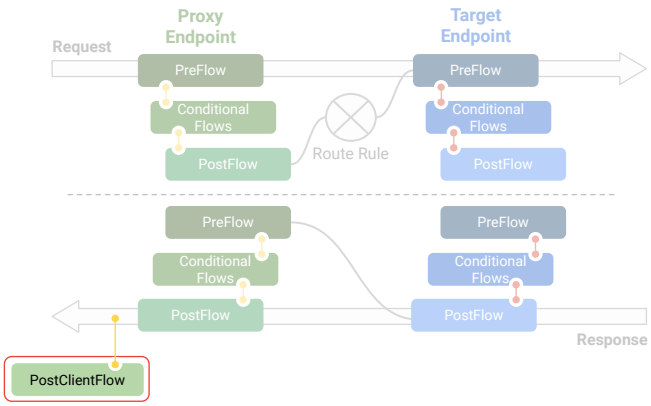
```
<MessageLogging continueOnError="true" enabled="true" name="LOG-LogMessage">
  <Syslog>
    <Message>{organization.name}.{apiproxy.name}.{environment.name} | {error.status.code}
{error.content}</Message>
    <Host>syslog.example.com</Host>
    <Port>2900</Port>
    <Protocol>TCP</Protocol>
    <SSLInfo>
        <Enabled>true</Enabled>
    </SSLInfo>
  </Syslog>
</MessageLogging>
```

The Host, Port, and Protocol specify the destination for the log message.

Log messages can use TCP or UDP for the protocol, and TCP messages can send the log message encrypted over TLS if SSLInfo is enabled.

PostClientFlow

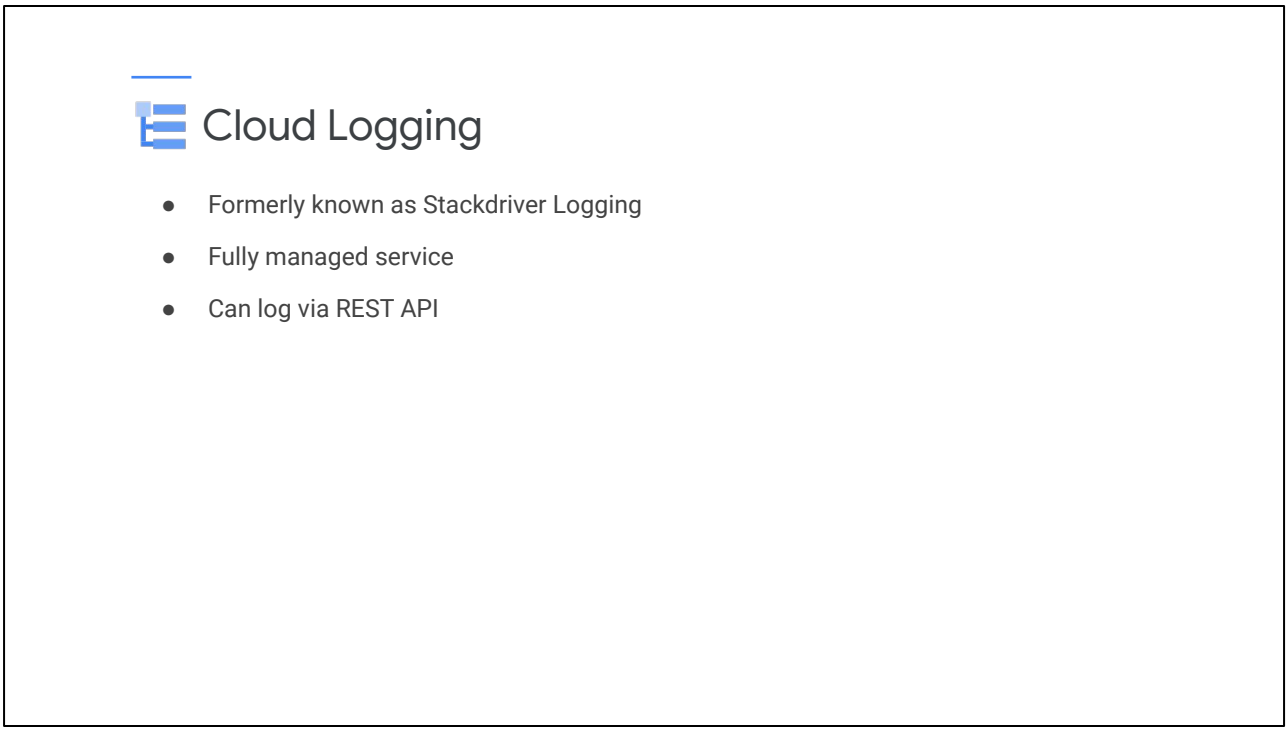
- MessageLogging policies in the *PostClientFlow* are executed after the response has been sent to the client.
- Logging in the PostClientFlow does not add to the call latency.
- Other types of policies in the PostClientFlow will be ignored.



There is another flow in the API proxy that we have not talked about yet. It is called the PostClientFlow, and is used for logging messages.

MessageLogging policies in the PostClientFlow are executed after the response is sent to the client. This allows logging to be used in your proxies without adding to the API call latency.

It is highly recommended that you put your logging policies in the PostClientFlow. Other types of policies that are placed in the PostClientFlow will be ignored.



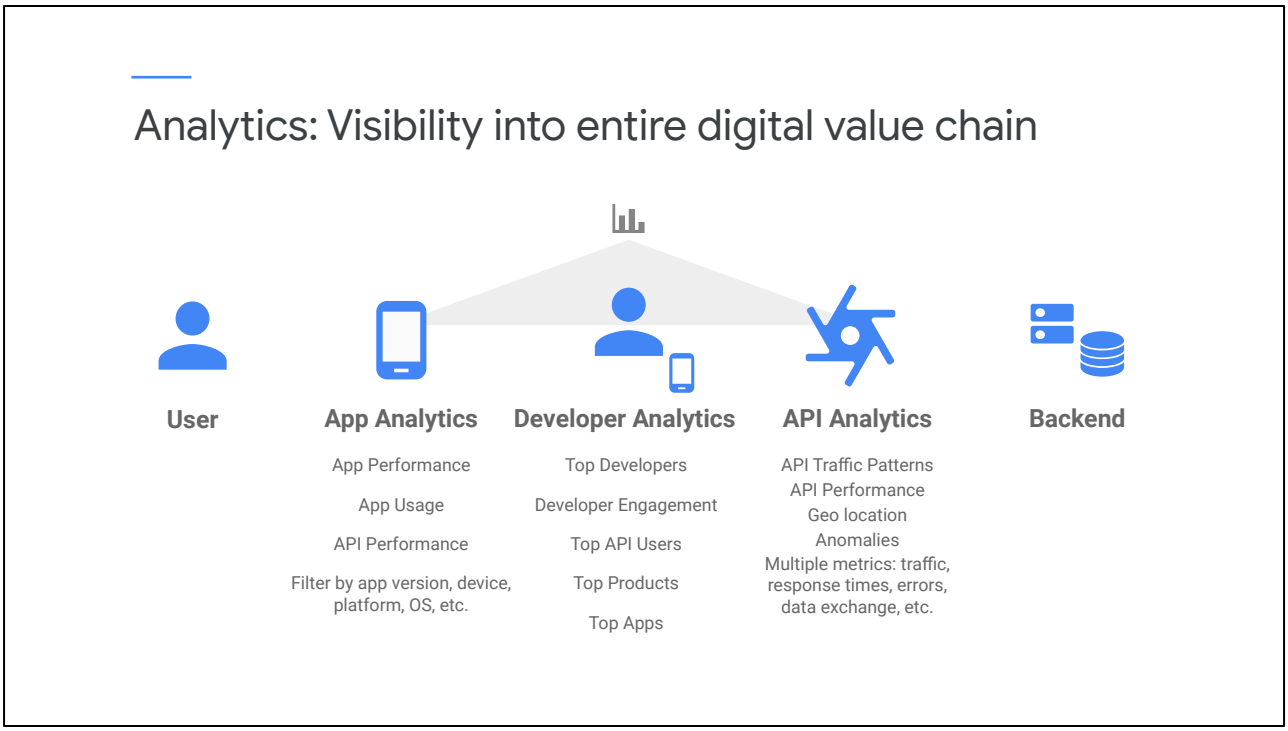
Google Cloud Logging, previously known as Stackdriver logging, is a fully managed logging service that allows you to store, search, analyze, monitor, and alert on log data.

You can send a log to Cloud Logging by using a REST API.

You may find it useful to create a shared flow to incorporate the steps of obtaining and caching a token and calling the Cloud Logging service.

Trends and metrics for APIs, apps, and app developers are important for driving improvements in API programs.

Apigee's analytics help give visibility into all aspects of your API program.



Apigee's analytics captures a wealth of information flowing through API proxies.

The analytics dashboards help answer questions about your APIs, like:

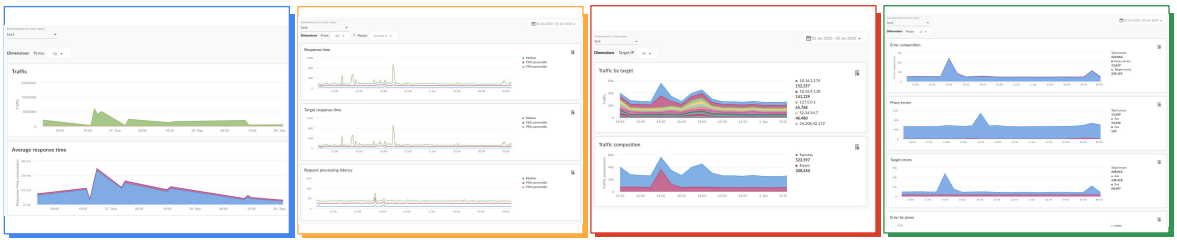
- "How is my API traffic trending over time?"
- "When is my API response time the fastest or the slowest?"
- "Geographically, where are my requests coming from?"
- "Which of my backends are returning the most errors?"

Analytics also captures information about the app developers and apps using your APIs, answering questions like:

- "Which app developers and apps are driving the most traffic?"
- "Which of my API products are used the most?"
- and, "How much traffic is being driven by specific devices, operating systems, and user agents?"

Visibility into API performance and usage

Many out-of-the-box dashboards



Proxy Performance | Latency Analysis | Target Performance | Error Code Analysis

There are many out-of-the-box dashboards that can help you analyze your APIs.

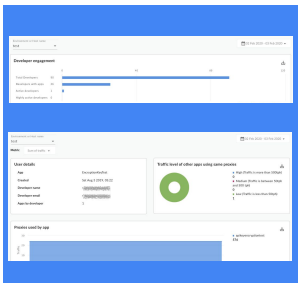
The proxy performance dashboard breaks down requests by API proxy and includes charts for total traffic, average response time, traffic by proxy, and average response time by proxy.

The latency analysis dashboard reports on overall response time, target response time, time taken by the proxy to process the request, and time taken by the proxy for the response. These metrics allow you to determine where most of the time is being taken during API calls, and can be analyzed by API proxy and region.

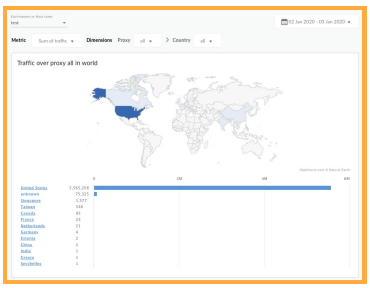
The target performance dashboard helps you determine the traffic and performance by backend target. The dashboard can show the total traffic by target, traffic by successful or unsuccessful response, response time for the proxy and target, target error composition by status code, and payload size.

The error code analysis dashboard specifies where errors are coming from. The dashboard shows how many errors are coming from the proxy and the target and the error composition for proxy and target errors by status code range and status code.

Measure and grow the API program



Engagement



Geomap

Devices

Dashboards and reports can also help with understanding metrics for app developers, apps, and users of the apps.

For example, you can track developer app traffic and errors, as well as the devices and locations for the users of apps using your APIs.

Improve API response times

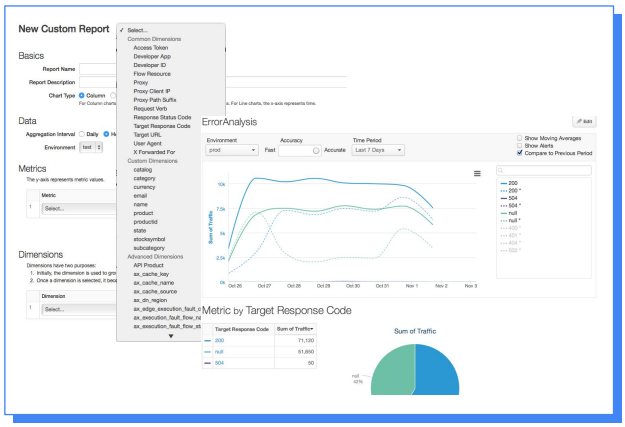
Measure cache performance

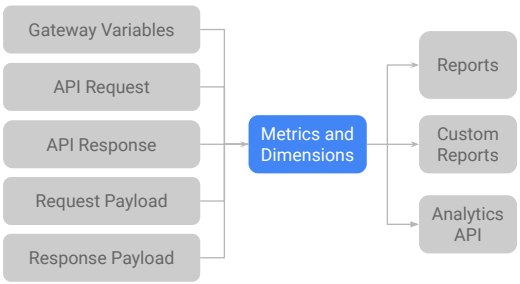
You can also use analytics to measure your cache performance.

You can analyze count and rate of cache hits, cache hits by app, average times for cache hits and for cache misses, and percentage improvement when there is a cache hit.

Build your own reports

Define custom metrics, dimensions, queries, reports, and dashboards





You can also build your own custom reports, when the out-of-the-box reports are not sufficient.

Metrics are numeric values that can be used as data points in the graph. For example, response time or error count can be specified as metrics.

String fields can only be specified as dimensions. Dimensions are used to group metrics in meaningful ways. For example, you can view traffic by API proxy name, API product name, or country in which the request originated.

Apigee automatically provides many metrics and dimensions for use in your custom reports, but you can also use your own custom metrics and dimensions in your reports by using the DataCapture policy in your API proxy.

Integrate analytics with enterprise tools

- Apigee API
  - Retrieve metrics by hostname over a period of time
  - Submit asynchronous queries
- Export raw analytics data by date range to BigQuery

In addition to using the Apigee console, there are some methods for integrating your API analytics with your enterprise tools.

The Apigee API can be used to retrieve metrics for an organization by hostname over a specific period of time. These metrics can be integrated into external tools and dashboards for easier visibility into important Apigee metrics. You can filter, sort, or group these metrics by dimension.

There is an asynchronous custom reports API for exporting custom report data. Asynchronous reports are useful for when the data sets are large, the time interval is large, or the query is complex due to a variety of dimensions and filters. You can specify a list of metrics, dimensions, and a filter for the asynchronous query. When the report is complete, you can download the data using the Apigee API or visualize the report in the Apigee console.

You can also retrieve analytics data by using the data export feature. The raw analytics data can be retrieved by date range into BigQuery.

Custom metrics and dimensions

- Common API metrics and dimensions are automatically collected.
- Cannot collect all metrics and dimensions that are important to business unless they are specified:
  - Product category
  - Business unit
  - Any custom value

One important feature for Apigee is the ability to capture custom metrics and dimensions for use in your analytics reports or company dashboards.

Apigee automatically captures common API metrics and dimensions.

Apigee cannot collect metrics and dimensions that are business-specific, though, unless you specify them in your proxies. For example, being able to filter a report based on product category or business unit may be very useful, but product category and business unit do not show up in a common location across APIs.

You can use the DataCapture policy to tell analytics to capture these important values.

Data Collector

- Captured value location and types are defined by Data Collector resources.
- *Name* is used in DataCapture policy.
- *Type* is string/integer/float/boolean/datetime.

Before using the DataCapture policy to save a value, you must create a Data Collector to hold the value.

A data collector contains a name, type, and optional description.

The name is used in the DataCapture policy to reference the data collector.

The type can be string, integer, floating point number, boolean, or a date/time.

DataCapture policy

- Each captured value must be stored in a data collector resource.
- The data collector determines the data type.
- Values may be captured from variables and message elements.
- If a value is captured into a data collector more than once during an API call, the data collector will contain the value from the last policy executed.

```
<DataCapture continueOnError="true"
 enabled="true" name="DC-CaptureData">
  <Capture>
    <DataCollector>dc_busUnit</DataCollector>
    <Collect default="UNK" ref="busUnit"/>
  </Capture>
  <Capture>
    <DataCollector>dc_urgency</DataCollector>
    <Collect default="0">
      <Source>request</Source>
      <JSONPayload>
        <JSONPath>$.urgency</JSONPath>
      </JSONPayload>
    </Collect>
  </Capture>
</DataCapture>
```

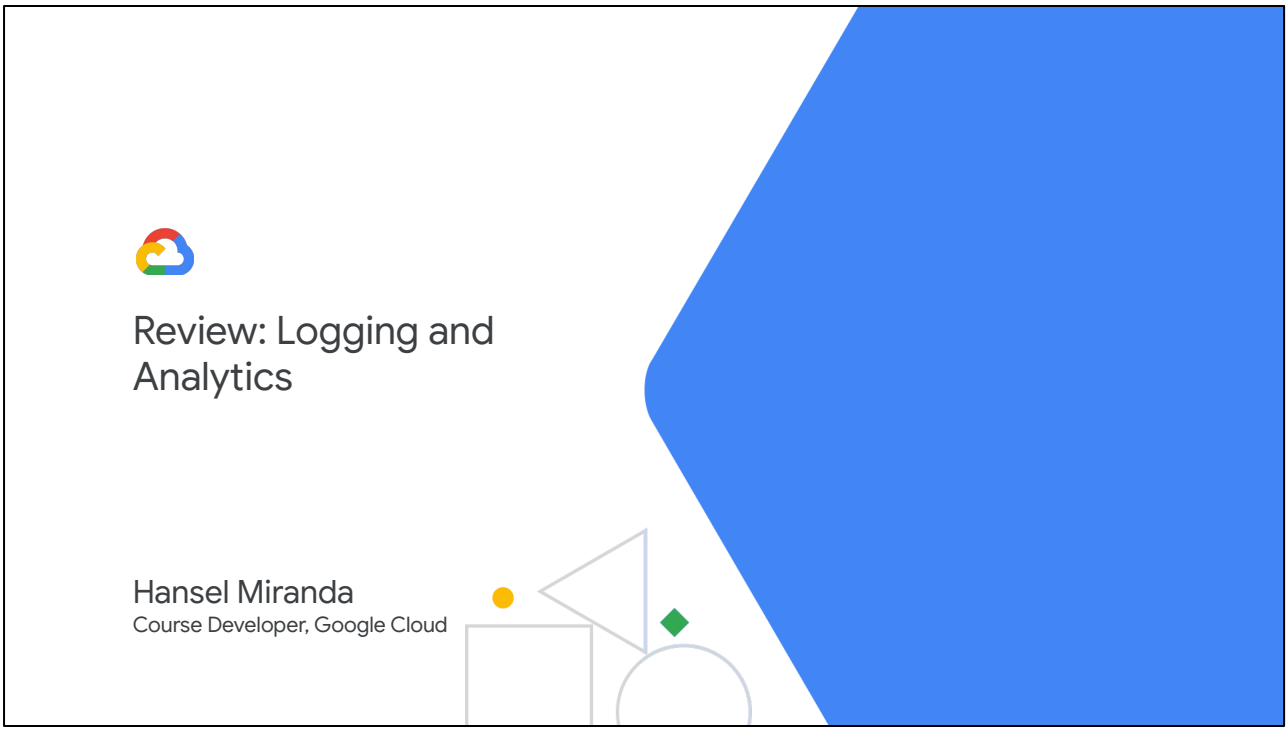
The DataCapture policy allows you to specify a list of values that should be captured into data collectors, for use as custom analytics metrics and dimensions.

Every captured value must be written into a data collector that already exists.

The specified data collector determines the data type of the captured value.

Values may be captured from variables or message elements like headers, query parameters, and JSON or XML payloads.

If data is captured into a data collector more than once during an API call, only the data from the last capture executed will be saved.



In this module, you learned about logging messages by using the MessageLogging policy.

And you learned about Analytics, and how to use the DataCapture policy to capture custom information for use in analytics reports.