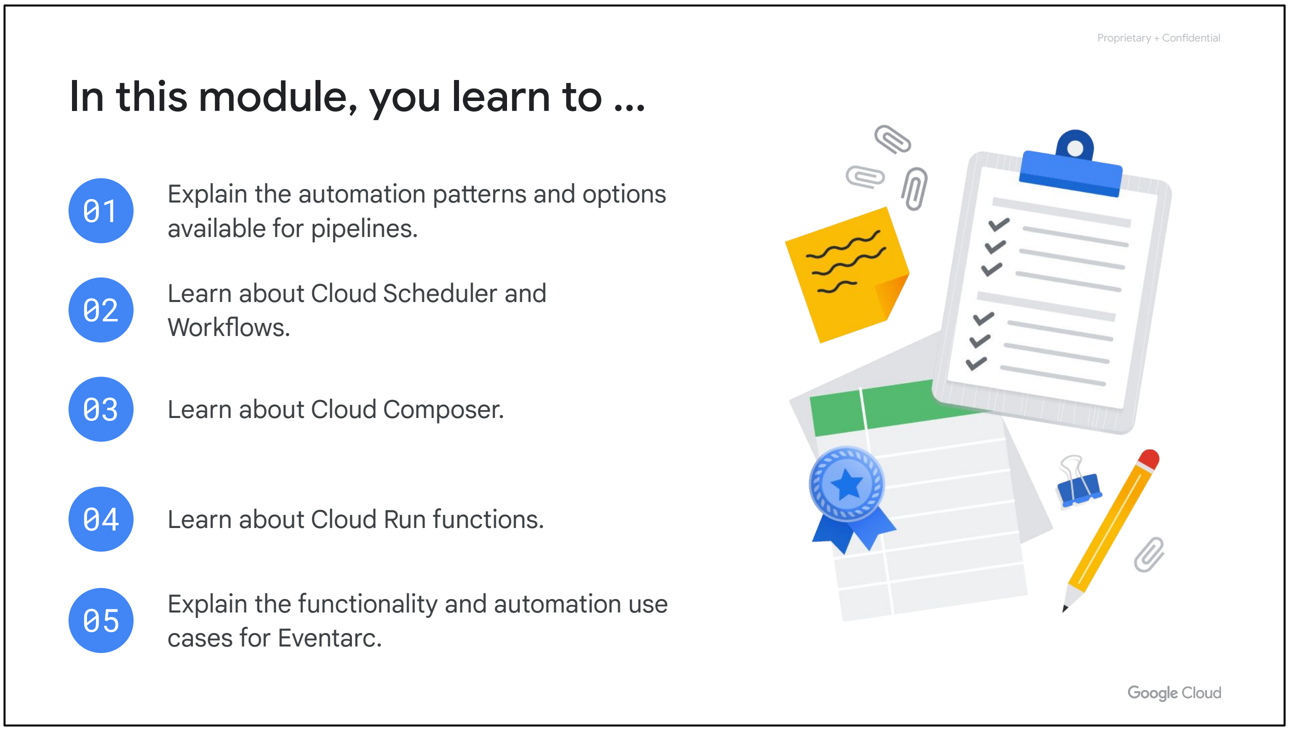
Automation Techniques



In this module, you learn to ...

01 Explain the automation patterns and options available for pipelines.

02 Learn about Cloud Scheduler and Workflows.

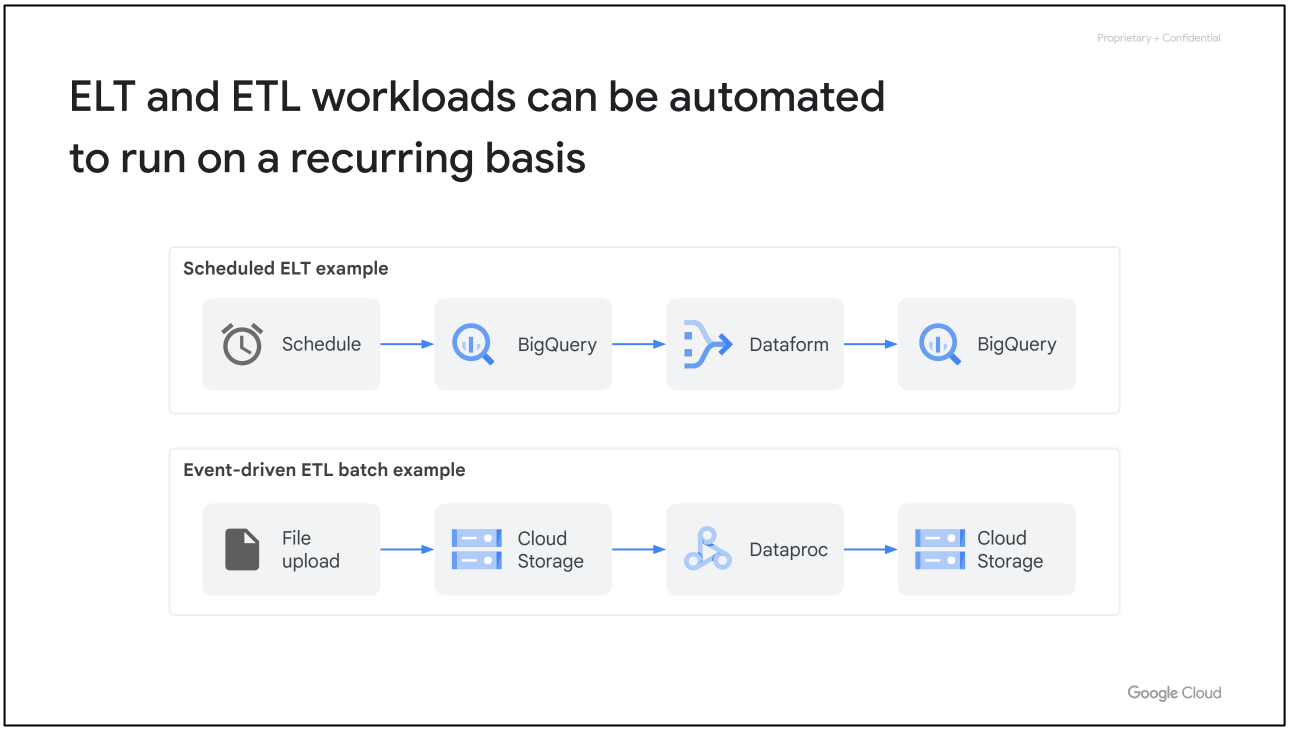
03 Learn about Cloud Composer.

04 Learn about Cloud Run functions.

05 Explain the functionality and automation use cases for Eventarc.

Google Cloud

In this module, first, you review the automation patterns and options available for pipelines. Second, you explore Cloud Scheduler and Workflows. Then, you review the functionality and use cases for Cloud Composer. Next, you review the capabilities of Cloud Run functions. Finally, you look at the functionality and automation use cases for Eventarc.


Proprietary + Confidential

ELT and ETL workloads can be automated to run on a recurring basis

On Google Cloud, ELT and ETL workloads can be automated for recurring execution.

For example, in a scheduled ELT, a defined schedule triggers data extraction from BigQuery, transformation via Dataform, and loading back into BigQuery.

Meanwhile, in the example shown for an event-driven ETL, a file upload to Cloud Storage initiates a batch process using Dataproc, culminating in data landing in Cloud Storage.

Instructor notes:

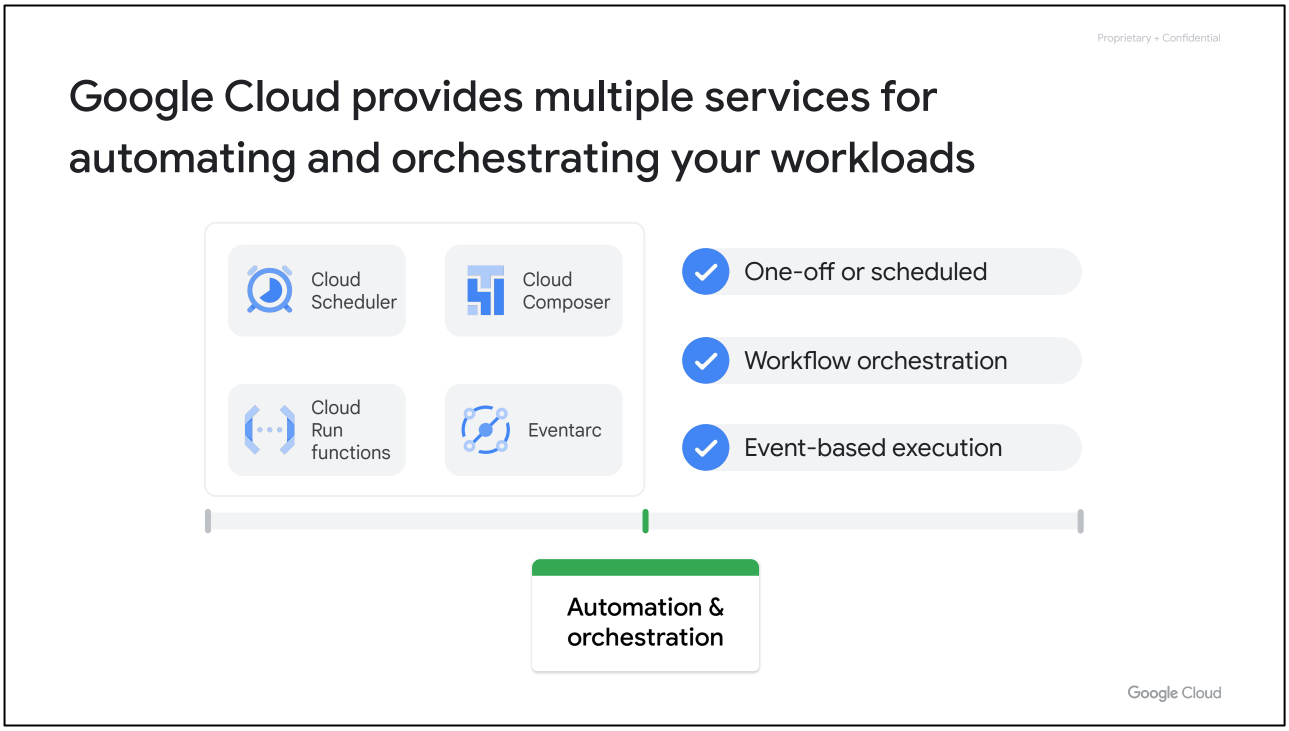
Scheduled ELT example:

- Scheduling system triggers the execution
- staging data is already loaded into BigQuery
- Dataform is triggered, pulls data from BigQuery, applies transformations
- BigQuery to store the end result

Event-driven ETL batch example:

- new file upload to Cloud Storage is the trigger
- Cloud Function receives this trigger event and calls Dataproc API with new file data (bucket and name)

- Dataproc executes a job on this file
- Dataproc stores the end result on Cloud Storage



Proprietary + Confidential

Google Cloud provides multiple services for automating and orchestrating your workloads

Cloud Scheduler
Cloud Composer
Cloud Run functions
Eventarc
One-off or scheduled
Workflow orchestration
Event-based execution
Automation & orchestration

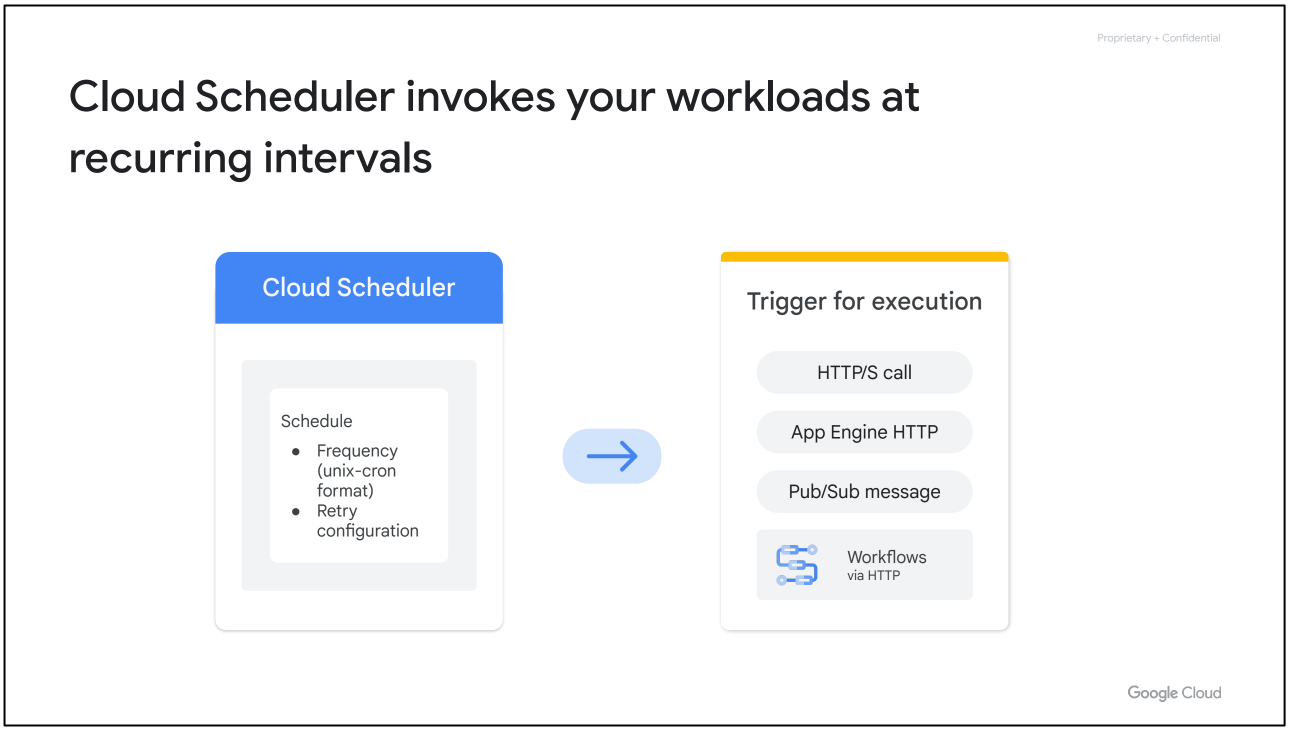
Google Cloud

Google Cloud offers a suite of services to automate and orchestrate your workloads.

For scheduled tasks or one-off jobs, you can leverage Cloud Scheduler and Cloud Composer.

If your workflows require orchestration, Cloud Composer is the ideal choice.

To trigger actions based on events, consider using Cloud Run functions or Eventarc.



Proprietary + Confidential

Cloud Scheduler invokes your workloads at recurring intervals

Google Cloud

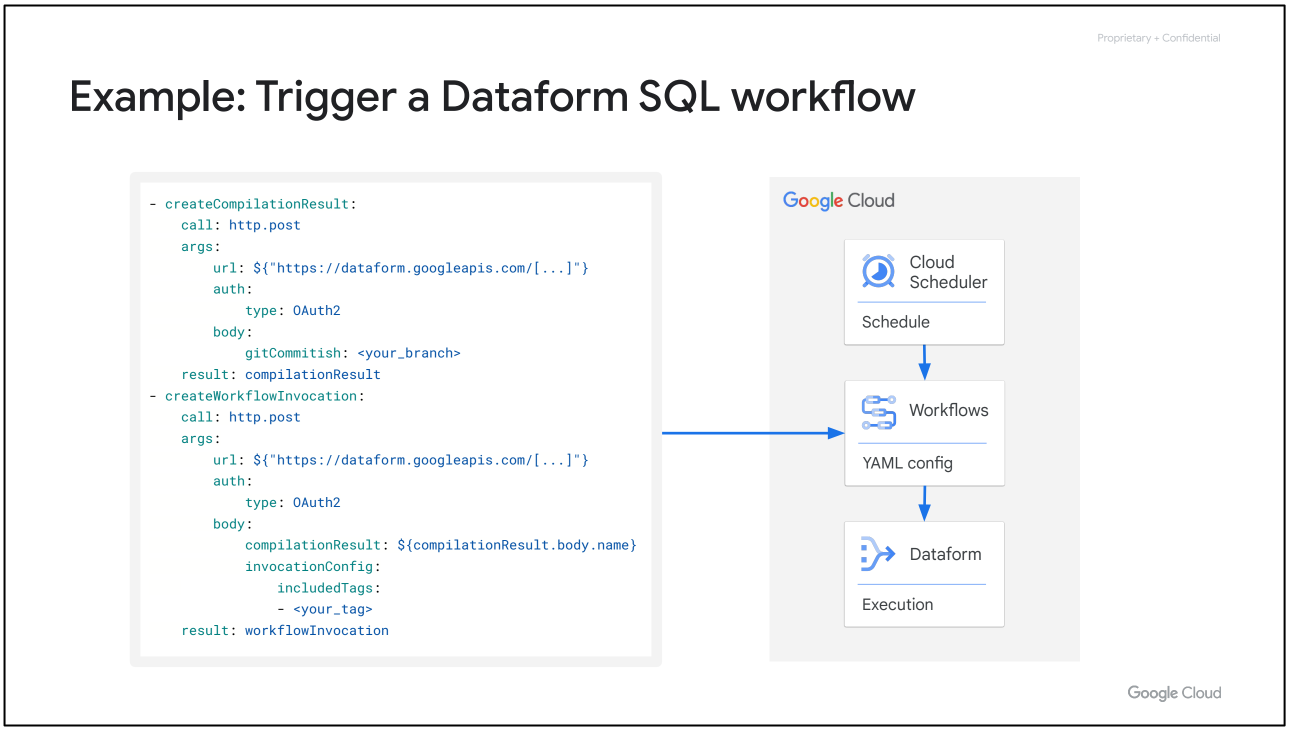
Cloud Scheduler empowers you to automate tasks by invoking your workloads at specified, recurring intervals.

It grants you the flexibility to define both the frequency and precise time of day for job execution.

Triggers can be based on HTTPS calls, App Engine HTTP calls, Pub/Sub messages, or Workflows.

References:

https://cloud.google.com/scheduler/docs/overview



Example: Trigger a Dataform SQL workflow

```
- createCompilationResult:
    call: http.post
    args:
        url: ${"https://dataform.googleapis.com/[...]"}
        auth:
            type: OAuth2
        body:
            gitCommitish: <your_branch>
    result: compilationResult
- createWorkflowInvocation:
    call: http.post
    args:
        url: ${"https://dataform.googleapis.com/[...]"}
        auth:
            type: OAuth2
        body:
            compilationResult: ${compilationResult.body.name}
            invocationConfig:
                includedTags:
                    - <your_tag>
    result: workflowInvocation
```

Cloud Scheduler can be used to trigger a Dataform SQL workflow.

In the example code, a scheduled job in Cloud Scheduler initiates the process, defined in a YAML config file.

The workflow involves two main steps: creating a compilation result from your Dataform code and then triggering a workflow invocation using that result, ensuring only specific parts of your Dataform project execute based on included tags.

References:

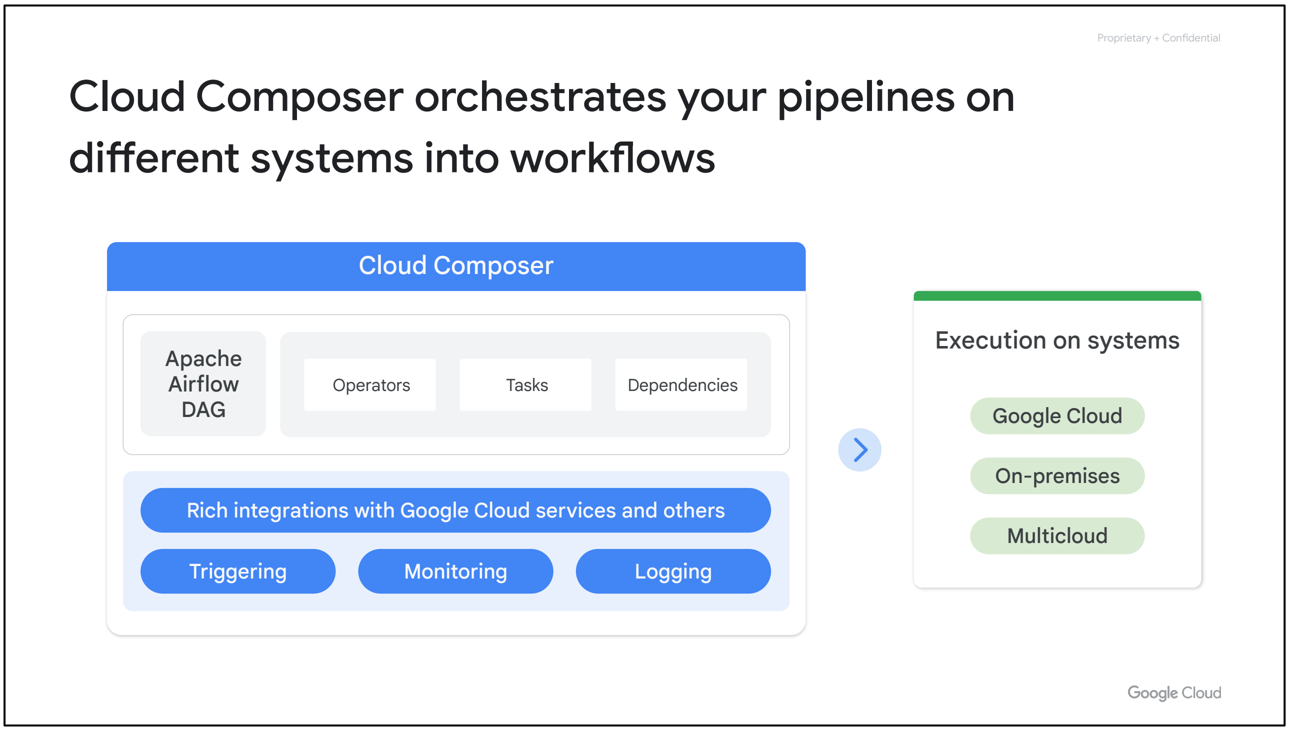
https://cloud.google.com/dataform/docs/schedule-executions-workflows

Instructor notes:

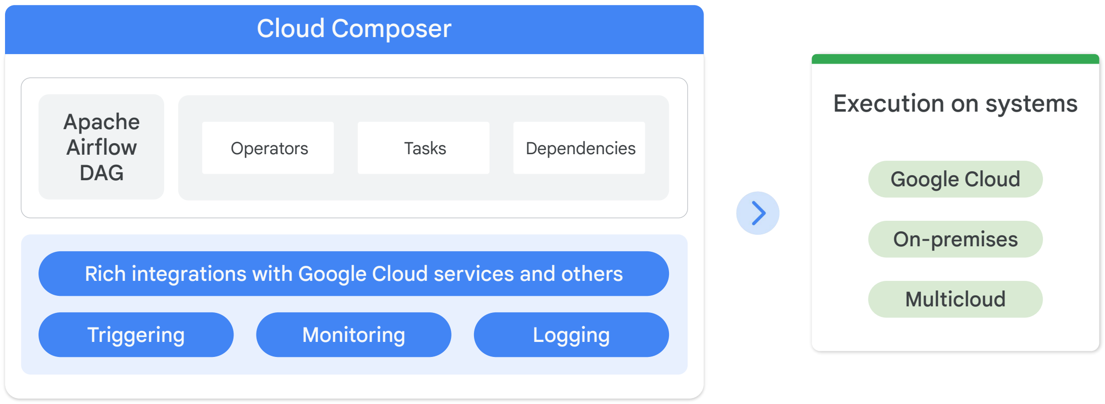
Main takeaways:

- You specify the execution of the Dataform workflow in Cloud Workflows
- API call and parameters need to be specified in YAML file as key-value pairs.
- Cloud Scheduler triggers Workflows using the schedule.
- Workflows then executes the API calls to Dataform

- Two API calls:
    - createCompilationResult: compile the SQLX files in the Dataform repository
    - createWorkflowInvocation: execute the workflow using the compilation result



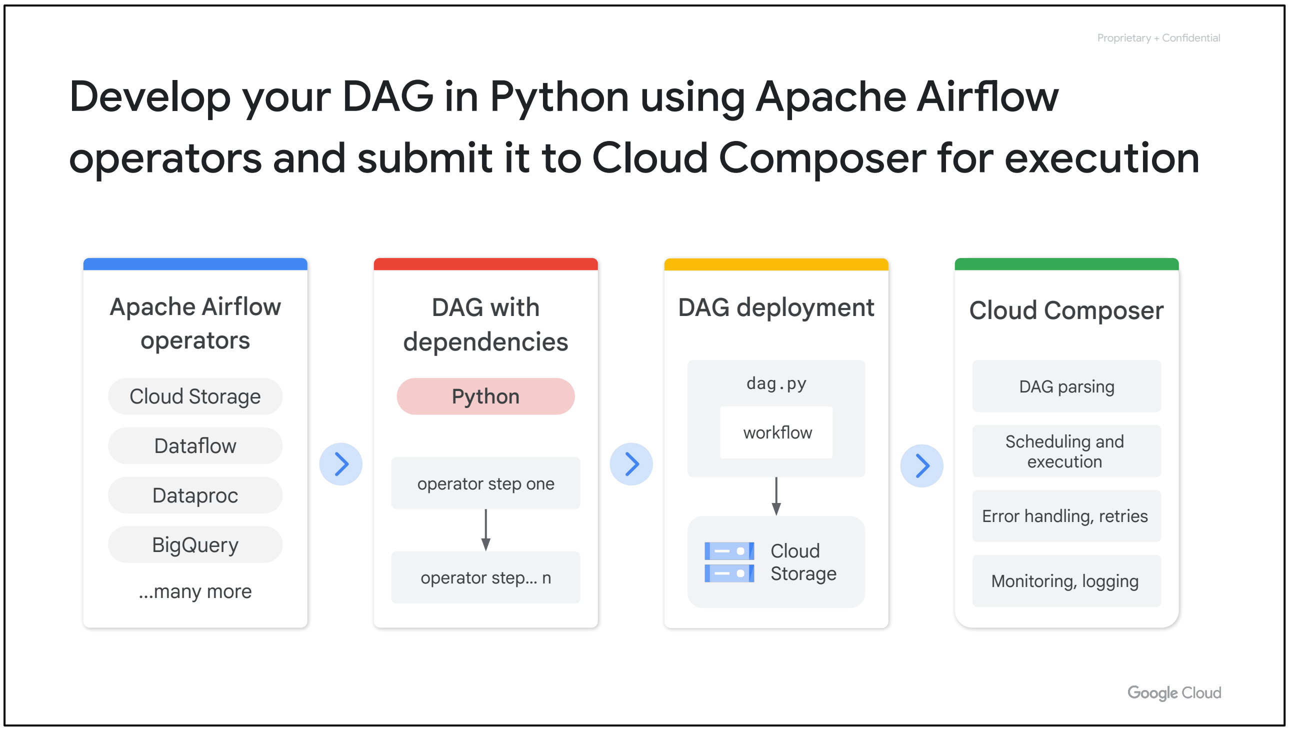
Cloud Composer orchestrates your pipelines on different systems into workflows



Cloud Composer acts as a central orchestrator, seamlessly integrating your pipelines across diverse systems—whether on Google Cloud, on-premises, or even multicloud environments.

Cloud Composer leverages Apache Airflow, incorporating essential elements like operators, tasks, and dependencies to define and manage your workflows.

Additionally, Cloud Composer offers robust features for triggering, monitoring, and logging, ensuring comprehensive control over your pipeline executions.



Develop your DAG in Python using Apache Airflow operators and submit it to Cloud Composer for execution

Developing and executing workflows using Apache Airflow and Cloud Composer is easily done using Python.

First, you leverage Apache Airflow operators to craft your directed acyclic graph or DAG, defining the tasks and their dependencies.

Next, the DAG is deployed to Cloud Composer, which handles the parsing and scheduling of your workflow.

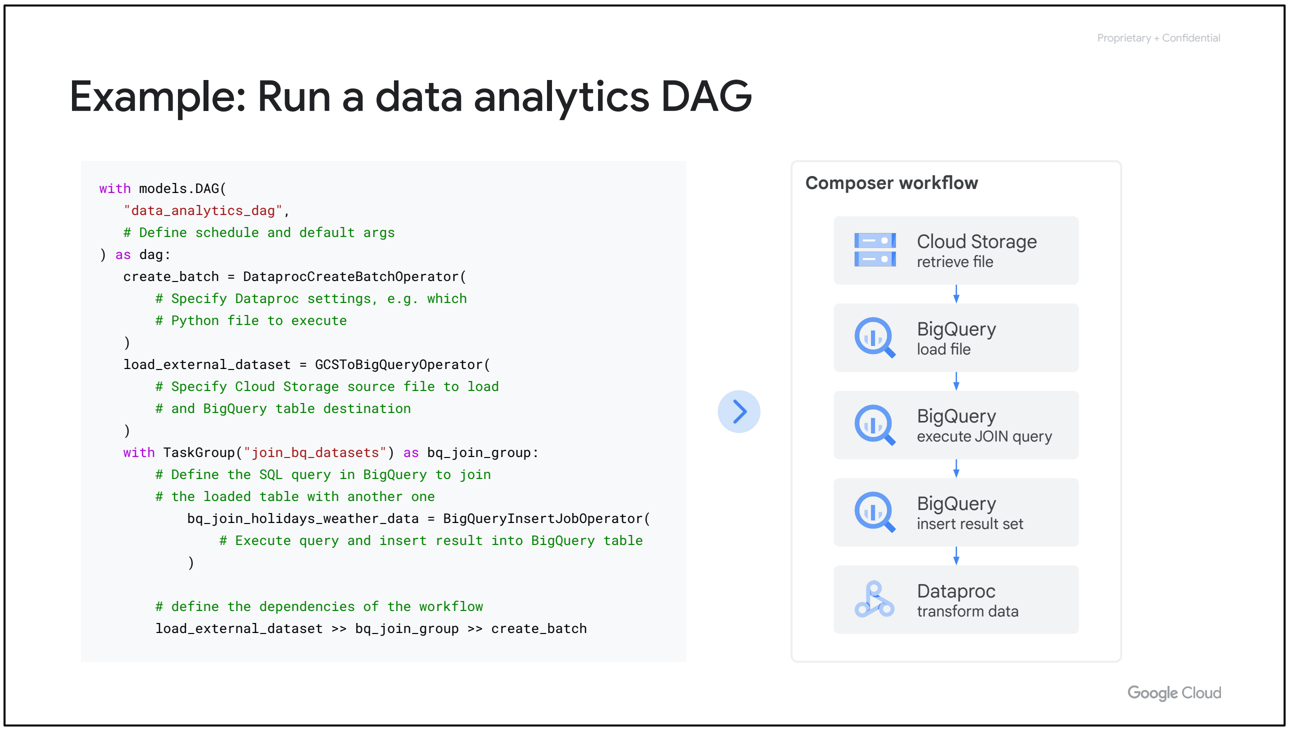
Cloud Composer further manages the execution of your tasks, incorporating features like error handling, retries, and monitoring to ensure smooth operation.

Instructor notes:

Step 4: Cloud Composer

- DAG parsing:
  - Airflow Scheduler periodically scans Cloud Storage bucket for updated or new python files (=DAGs)
  - Code gets parsed into tasks, dependencies, scheduled intervals
- Scheduling and execution
  - you can set a schedule in the Airflow UI as well, or execute it one-off

    - on-demand
    - the Airflow Executor is responsible for running the tasks
- Error handling, retries
    - If a task fails, Airflow can be configured to:
        - Retry the task a certain number of times
        - Send alerts (via email or other mechanisms)
        - Trigger downstream tasks that handle errors
- Monitoring, logging
    - using the Airflow UI:
        - Visualize your DAGs
        - Monitor task progress
        - View logs for troubleshooting



Example: Run a data analytics DAG

```
with models.DAG(
   "data_analytics_dag",
   # Define schedule and default args
) as dag:
   create_batch = DataprocCreateBatchOperator(
       # Specify Dataproc settings, e.g. which
       # Python file to execute
   )
   load_external_dataset = GCSToBigQueryOperator(
       # Specify Cloud Storage source file to load
       # and BigQuery table destination
   )
   with TaskGroup("join_bq_datasets") as bq_join_group:
       # Define the SQL query in BigQuery to join
       # the loaded table with another one
           bq_join_holidays_weather_data = BigQueryInsertJobOperator(
               # Execute query and insert result into BigQuery table
           )

       # define the dependencies of the workflow
       load_external_dataset >> bq_join_group >> create_batch
```

With minimal effort, Cloud Composer can be used to run a data analytics DAG.

In the example code, the workflow retrieves a file from Cloud Storage, loads it into BigQuery, and then performs a JOIN operation with an existing BigQuery table.

The joined results are then inserted into a new BigQuery table. Finally, Dataproc is used for further data transformation.

References:

https://cloud.google.com/composer/docs/composer-3/run-data-analytics-dag-googlecloud

Instructor notes:

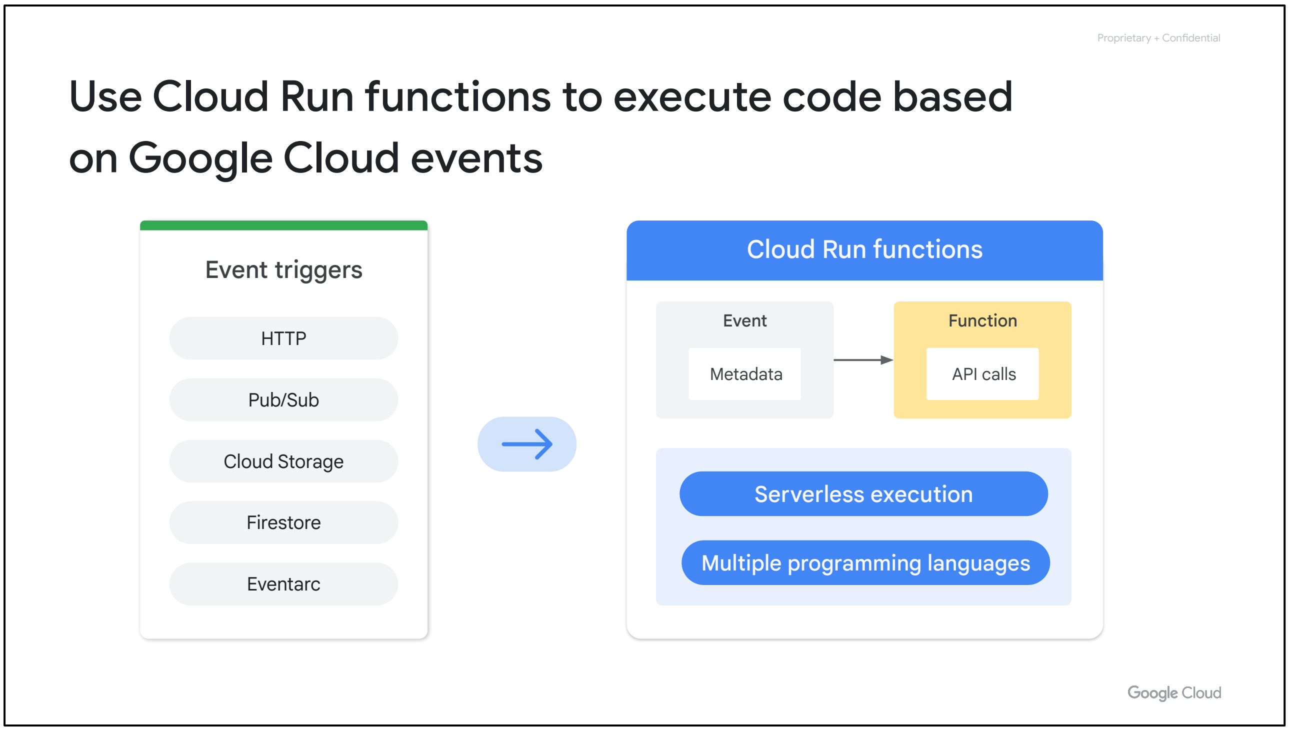
Main takeaways:

- DAG code here is just a snippet showing the outline of the DAG
- DAG code is easy to write and high level, since the underlying logic is performed by the Operator code itself
- you as the developer have to specify key-value pairs as properties only (e.g.

- which project to use)
- these properties are passed by the code to the Operator which then executes the task

Key steps:

- DAG is defined with name, but you can also define schedule and default args here
- create_batch is for Dataproc to execute a batch job, you would need to specify attributes here, e.g. which project and which file to use
- load_external_dataset accesses a Cloud Storage bucket and loads the data into BigQuery, here you need to specify source and destination properties
- bq_join_group defines the JOIN SQL query, and uses bq_join_holidays_weather_data to execute it and insert the results into a table, here you also have to define which project, dataset, and table
- dependencies between all the steps is defined in the last line of code



Use Cloud Run functions to execute code based on Google Cloud events

Google Cloud

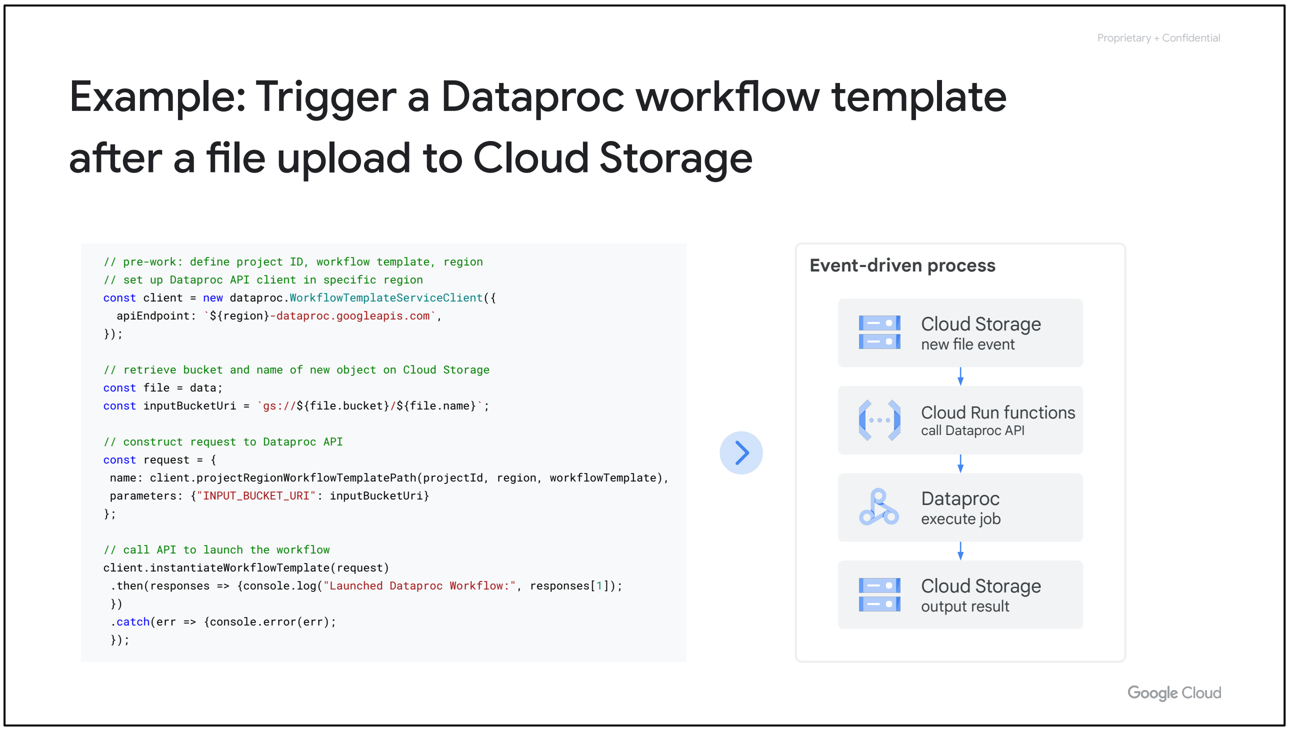
Cloud Run functions allow you to execute code in response to various Google Cloud events.

These events can originate from sources like HTTP requests, Pub/Sub messages, Cloud Storage changes, Firestore updates, or custom events through Eventarc.

When triggered, a Cloud Function provides a serverless execution environment where your code runs, supporting multiple programming languages for flexibility.

References:

Types of triggers:
https://cloud.google.com/functions/docs/calling



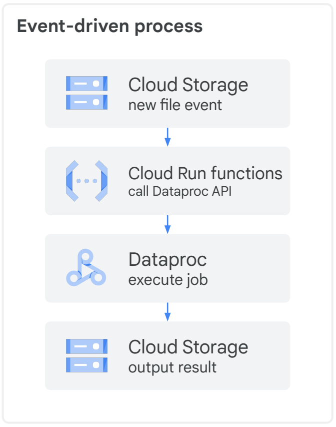
Example: Trigger a Dataproc workflow template after a file upload to Cloud Storage

```
// pre-work: define project ID, workflow template, region
// set up Dataproc API client in specific region
const client = new dataproc.WorkflowTemplateServiceClient({
  apiEndpoint: `${region}-dataproc.googleapis.com`,
});

// retrieve bucket and name of new object on Cloud Storage
const file = data;
const inputBucketUri = `gs://${file.bucket}/${file.name}`;

// construct request to Dataproc API
const request = {
 name: client.projectRegionWorkflowTemplatePath(projectId, region, workflowTemplate),
 parameters: {"INPUT_BUCKET_URI": inputBucketUri}
};

// call API to launch the workflow
client.instantiateWorkflowTemplate(request)
 .then(responses => {console.log("Launched Dataproc Workflow:", responses[1]);
 })
 .catch(err => {console.error(err);
 });
```



Google Cloud

Cloud Run functions easily automate routine tasks on Google Cloud.

In the example code, a Dataproc workflow template is triggered after a file is uploaded to Cloud Storage.

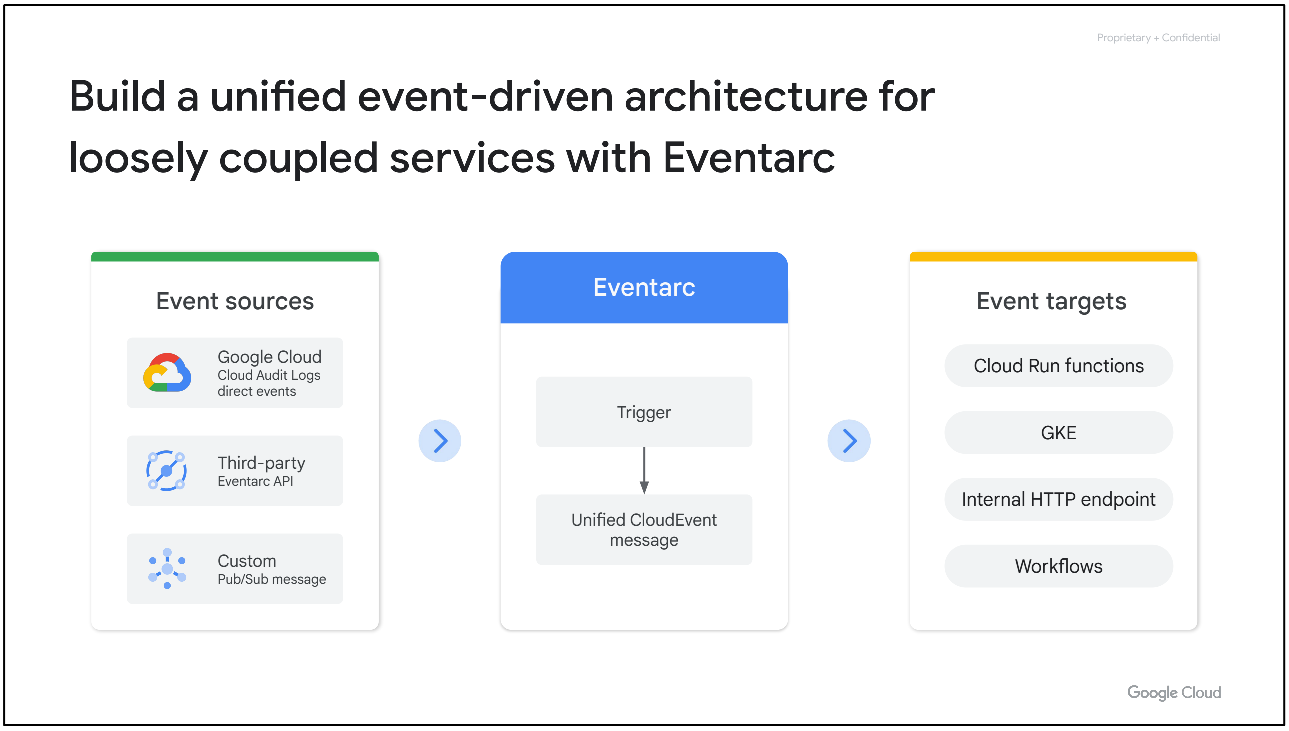
A Cloud Run function is used to capture the Cloud Storage new file event and call the Dataproc API.

The Dataproc API then executes the specified workflow template, using the uploaded file as an input parameter.

The final result of the workflow execution is stored in Cloud Storage.

References:

https://cloud.google.com/dataproc/docs/tutorials/workflow-function


Proprietary + Confidential

Build a unified event-driven architecture for loosely coupled services with Eventarc

Google Cloud

Eventarc enables the creation of a unified event-driven architecture for loosely coupled services.

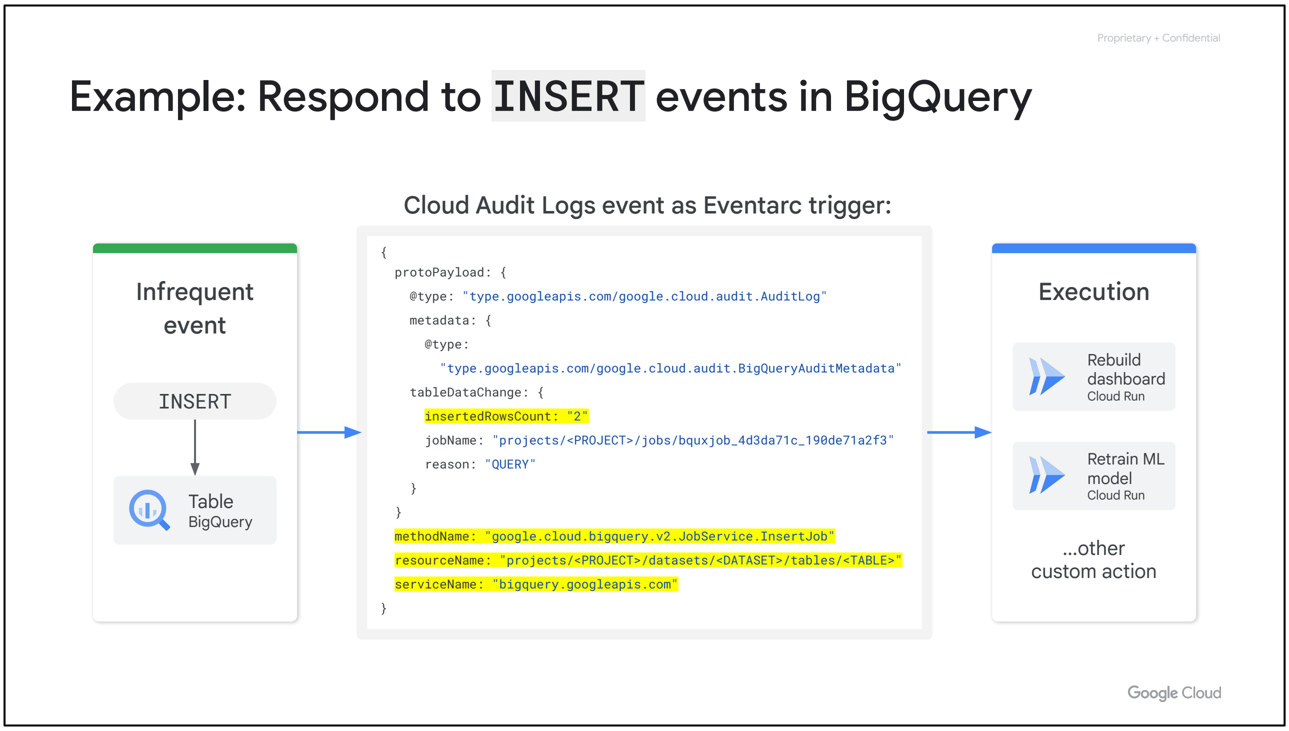
Eventarc connects various event sources, including Google Cloud services, third-party systems, and custom events via Pub/Sub, to a range of event targets like Cloud Run functions, and more.

By using a standardized CloudEvent message format, Eventarc simplifies the integration of diverse systems and facilitates the development of responsive, scalable applications.

References:

https://cloud.google.com/eventarc/docs/overview

List of triggers:
https://cloud.google.com/eventarc/docs/event-providers-targets#triggers



Example: Respond to `INSERT` events in BigQuery

Cloud Audit Logs event as Eventarc trigger:

```
{
  protoPayload: {
    @type: "type.googleapis.com/google.cloud.audit.AuditLog"
    metadata: {
      @type:
        "type.googleapis.com/google.cloud.audit.BigQueryAuditMetadata"
    tableDataChange: {
      insertedRowsCount: "2"
      jobName: "projects/<PROJECT>/jobs/bquxjob_4d3da71c_190de71a2f3"
      reason: "QUERY"
    }
  }
  methodName: "google.cloud.bigquery.v2.JobService.InsertJob"
  resourceName: "projects/<PROJECT>/datasets/<DATASET>/tables/<TABLE>"
  serviceName: "bigquery.googleapis.com"
}
```

Eventarc enables deep monitoring of logging and other events which occur less frequently on Google Cloud.

In the example code, Eventarc is used to trigger actions in response to data insertion events in BigQuery.

When an insert operation occurs in a BigQuery table, it generates a Cloud Audit Log event.

Eventarc can capture this event and initiate various actions such as rebuilding a dashboard, retraining an ML model, or executing any other custom action based on the specific requirements.

References:

https://cloud.google.com/blog/topics/developers-practitioners/how-trigger-cloud-run-actions-bigquery-events

Instructor notes:

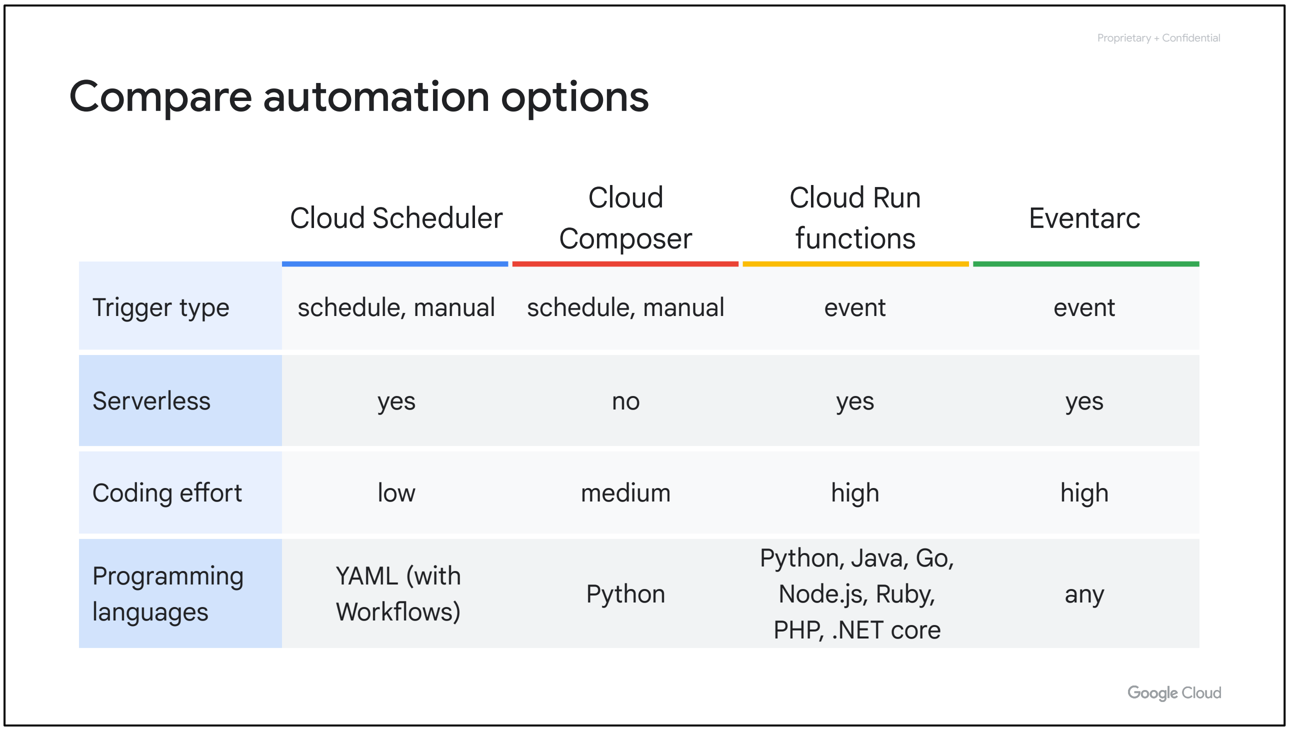
Background on why this is cool:

- You may have tables you build dashboards for or do other operations on based on INSERTs of new records
- tables may receive infrequent INSERTs, maybe just once a week at different times >> when to trigger your operation then?
    - schedule it to run daily? this may create overhead in the system, since it runs all the time no matter if there was an INSERT or not
    - schedule it to run weekly? you may miss the event so that execution takes place long time after INSERT happened
    - remember it's about loosely coupled services, so you don't have any insight on when the INSERT happens and which system executes it, so you cannot "chain" it together with the other system
- using Eventarc, you can trigger your operation exactly when the INSERT in BigQuery happened

Implementation:

- filters in Cloud Logs Explorer:
    - protoPayload.serviceName="bigquery.googleapis.com" AND
    - protoPayload.methodName="google.cloud.bigquery.v2.JobService.InsertJob" AND
    - protoPayload.resourceName="projects/<your_project>/datasets/<your_dataset>/tables/<your_table>"
- filters in Eventarc:
    - serviceName=bigquery.googleapis.com
    - methodName=google.cloud.bigquery.v2.JobService.InsertJob
    - resourceName=projects/<your_project>/datasets/<your_dataset>/tables/<your_table>
- these filters result in 2 events being triggered in Eventarc by one BigQuery INSERT (since BigQuery logs once tableDataChange and once tableDataRead)
    - the event with tableDataChange will show how many rows were inserted
    - use this indication in the code to run your operations

You can optionally show a demo here.
Demo steps are in the guide here:
https://docs.google.com/document/d/1hLoeHfGAnyp6tyjH3LtHsVg2rB_PhGYme47-BLJ5KqQ/edit?resourcekey=0-92Txnxw9r6ZzTPll4pGEzw&tab=t.0#heading=h.h8lmugn5880



Compare automation options

	Cloud Scheduler	Cloud Composer	Cloud Run functions	Eventarc
Trigger type	schedule, manual	schedule, manual	event	event
Serverless	yes	no	yes	yes
Coding effort	low	medium	high	high
Programming languages	YAML (with Workflows)	Python	Python, Java, Go, Node.js, Ruby, PHP, .NET core	any

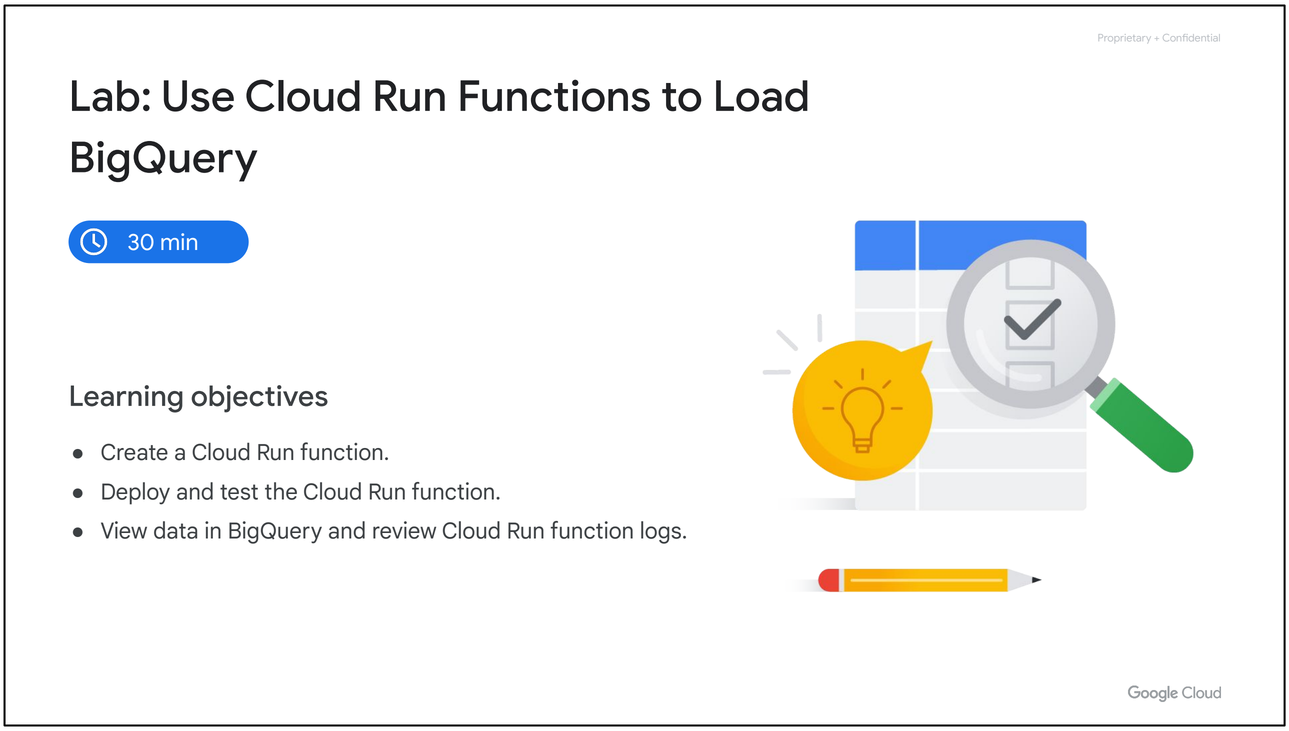
In summary, there are various Google Cloud data-related automation options.

Cloud Scheduler and Cloud Composer are suitable for scheduled or manual triggers, while Cloud Run functions and Eventarc are event-driven.

Cloud Scheduler offers low coding effort with YAML, and Cloud Composer requires medium effort with Python.

Cloud Run functions support multiple languages, while Eventarc is language-agnostic.

As a final note, all options except Cloud Composer are serverless.



Proprietary + Confidential

Lab: Use Cloud Run Functions to Load BigQuery

30 min

Learning objectives

- Create a Cloud Run function.
- Deploy and test the Cloud Run function.
- View data in BigQuery and review Cloud Run function logs.

Google Cloud

In this lab, you create a Cloud Run function to Load BigQuery.

You create a Cloud Run function using the Cloud SDK.

You then deploy and test the Cloud Run function.

Finally, you view data in BigQuery and review Cloud Run function logs.

reference: https://www.cloudskillsboost.google/authoring/labs/31673

Google Cloud