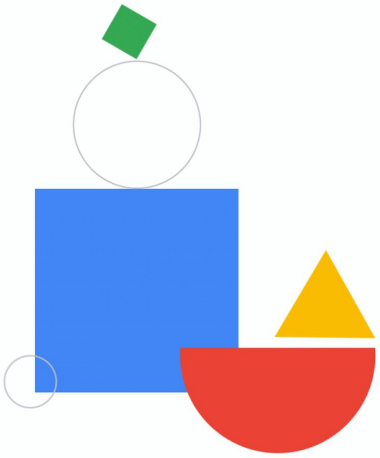
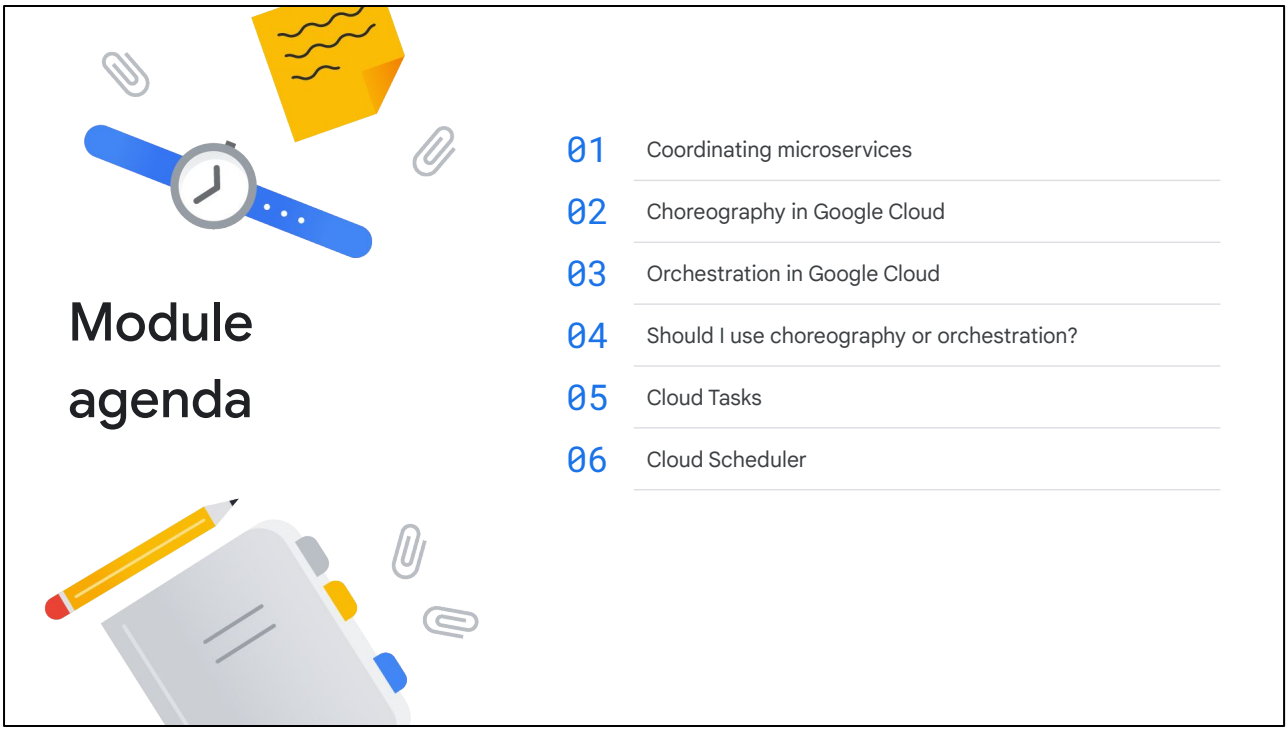
Service Orchestration and Choreography on Google Cloud

Module 3: Choreography and Orchestration



Welcome to module 3 of Service Orchestration and Choreography on Google Cloud: Choreography and Orchestration.



In this module, you learn about two primary patterns for coordinating microservices: choreography and orchestration.

We explore the choreography pattern. You learn how Pub/Sub and Eventarc can help you choreograph communication between microservices. We also discuss when you should use Eventarc for your applications.

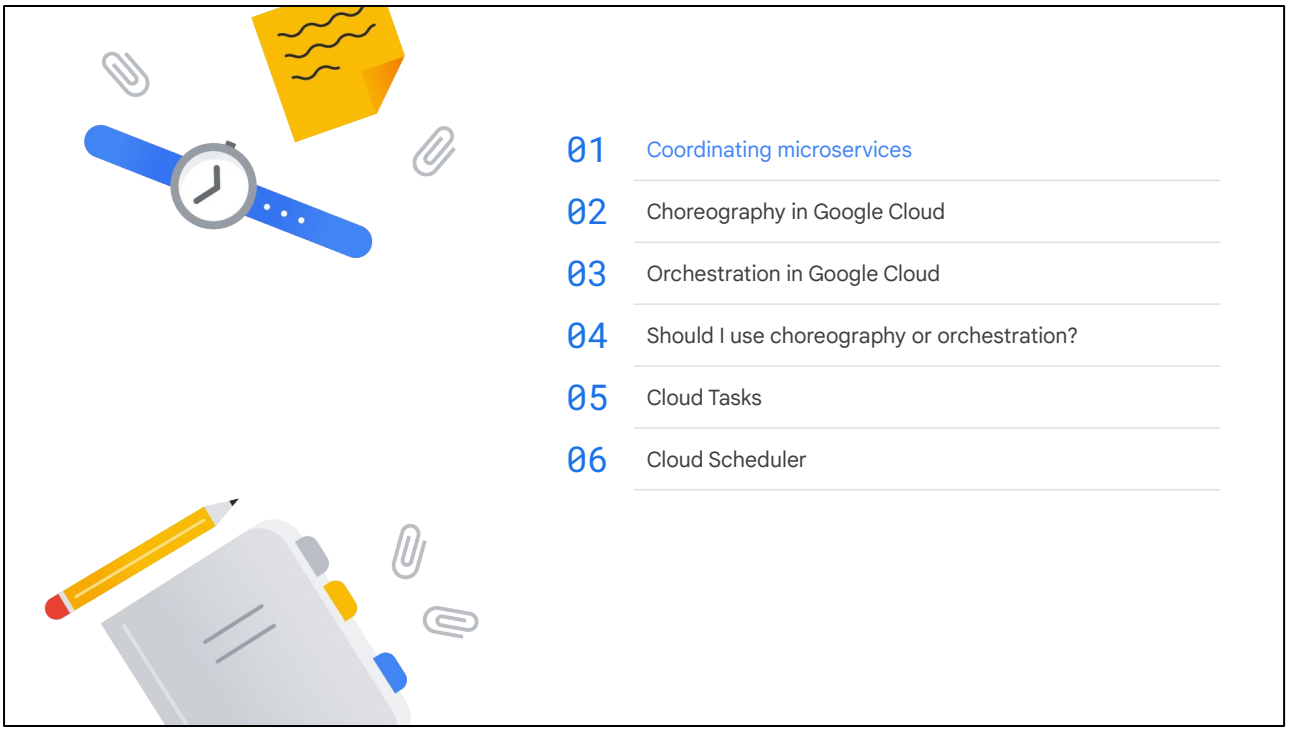
You also learn about Workflows, Google Cloud's orchestration service. We discuss when Workflows is the correct coordination solution.

We compare the use cases for choreography and orchestration, and you learn how each pattern can be used in your applications.

You learn about Cloud Tasks, a service that helps you manage asynchronous HTTP requests.

You learn about Cloud Scheduler, a service for managing scheduled jobs.

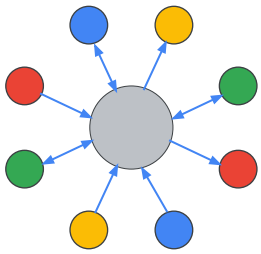
Finally, this module concludes with a lab where you use all of these Google Cloud services to create a machine learning-driven image application.



One challenging aspect of using a microservices architecture is coordinating the communication between microservices.

Coordinating microservices architectures

- A microservices architecture has significant benefits, but inter-service communication requires more focus.
- Business processes involve coordination of multiple microservices.
- Two useful coordination approaches are service choreography and service orchestration.

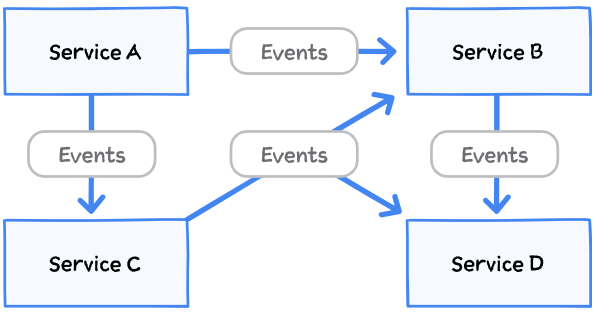


When you use a microservices architecture, you benefit from scalability, reusability, and ease of change. Each microservice is smaller and simpler to design than a monolithic application, but some complexity is shifted from the service to inter-service communications.

Multiple microservices are often combined to form business processes. The coordination of the communication becomes a key aspect of the design of your application.

There are two basic approaches to coordination in an event-driven architecture: service choreography and service orchestration.

Service choreography uses independent services



+ Each service works independently, interacting by sending asynchronous events.

+ Services are loosely coupled, and can be created, changed, and scaled independently.

- Business logic is distributed, which can be harder to understand.

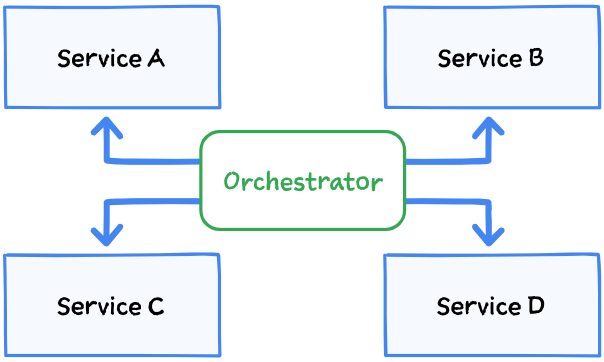
Service choreography has similarities to the choreography of a dance performance. When a dance is choreographed, the dancers are instructed how to perform the dance, but dancers are fully responsible for performing their parts during the performance.

With service choreography, each service works independently. Each service is responsible for receiving and sending events asynchronously, typically following defined rules of interaction between services. The event structures that are exchanged between services are specified, and each service must generate and expect services in the correct formats.

With choreography, services are loosely coupled, and can be created, changed, and scaled separately.

One choreography challenge is that the business logic is distributed. A distributed application can be harder to understand because there's no central source of truth.

Service orchestration uses a central orchestrator



- \+ A central orchestrator controls all interactions between the services.
- \+ Services are loosely coupled.
- \+ The orchestrator provides a high-level view of business processes to track execution and troubleshoot issues.
- \- The central orchestrator is a single point of failure.

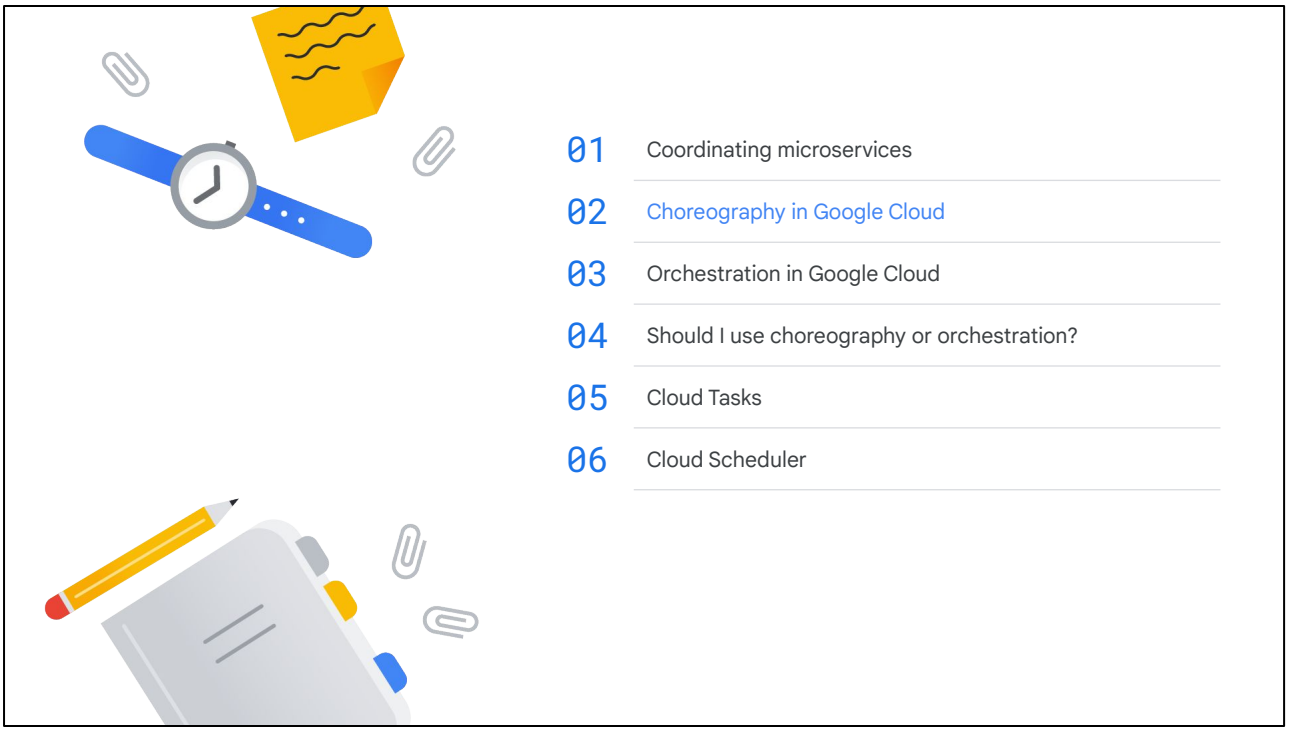
Service orchestration is similar to an orchestra performance. Each musician in the orchestra knows how to play their instrument, but the conductor takes an active role during the performance to ensure that the musicians are synchronized.

With service orchestration, each service performs its own tasks, but the central orchestrator controls all interactions between the services. Services do not need to know about or communicate with other services in the orchestration.

As with service choreography, the services are loosely coupled, and can be created, changed, and scaled separately.

Another benefit of orchestration is that it provides a high-level view of business processes which helps with understanding the application, tracking execution, and troubleshooting issues.

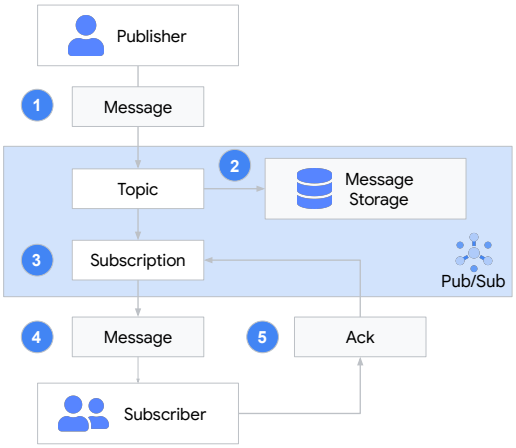
Unlike the fully distributed services in the service choreography pattern, service orchestration has a single point of failure. If the orchestrator is not operable, the orchestrated processes cannot run.



Now that you understand the basics of service choreography and orchestration, we will discuss how you can choreograph services in Google Cloud.

Create systems of message publishers and subscribers with Pub/Sub

- Pub/Sub is a fully managed real-time messaging service.
- A publisher sends a message to a Pub/Sub topic.
- Pub/Sub then delivers a message to a queue for each subscriber.
- Each subscriber receives the message and then acknowledges it to remove it from the queue.



Pub/Sub is one of the services on Google Cloud that can be used to choreograph services.

Pub/Sub is a fully managed real-time messaging service that lets you send and receive messages between independent services or applications.

A publisher sends a message to a Pub/Sub topic. This publisher is often a custom-built application, but it could also be a Google service.

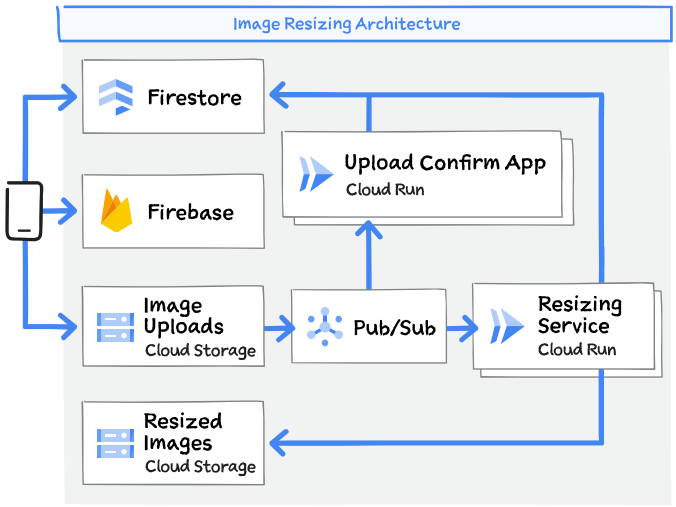
The message is stored within Pub/Sub and then delivered to the message queue for each subscriber for the topic.

Each Pub/Sub subscriber then receives the message. A subscriber with a pull subscription would occasionally poll for messages that have been sent to the topic. A push subscription would cause the message to be automatically sent to a configured endpoint for the subscription.

The subscriber would then acknowledge the message. Acknowledging a message removes it from the queue. The deletion of the queued message follows handling of the message, which guarantees that a message will be handled at least once.

Pub/Sub: https://cloud.google.com/pubsub

Using Pub/Sub to connect services



Here's an example of using Pub/Sub to connect services and build an application. This diagram shows an image resizing application.

A Pub/Sub topic is created for image uploads. A specific Cloud Storage bucket is configured to send a Pub/Sub message to that topic when a new image is received.

When the mobile device application uploads a new image to the Image Uploads bucket, a Pub/Sub message is automatically sent to Pub/Sub. This message is then forwarded to the two subscribers, the Resizing Service and the Upload Confirm application.

The Upload Confirm application updates Firestore, storing the fact that the image upload completed successfully.

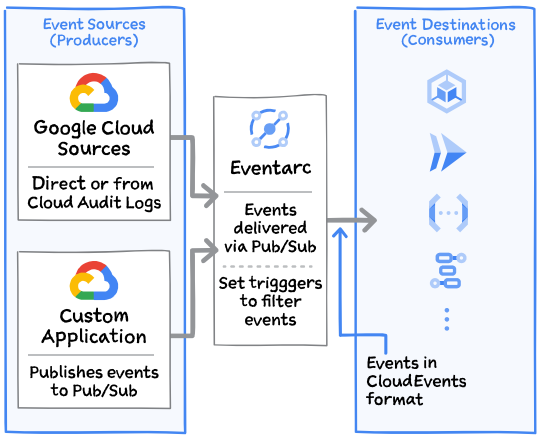
The Resizing Service resizes the image, storing it in the Resized Images bucket and also updating Firestore with the status.

The application can track status updates in Firestore using Firebase Realtime Database updates.

Each of the Cloud Run services can remain fairly simple, with Pub/Sub initiating processing when there's an image to act upon.

Eventarc is a managed eventing system

- Events can be sourced from Google Cloud sources, Cloud Audit Logs, and third-party providers.
- Events can also be generated from Pub/Sub messages.
- Triggers are used to filter events for a specific target.
- Pub/Sub is the transport layer.
- Events use the CloudEvents format.



Eventarc is Google Cloud's fully managed eventing system that makes building event-driven applications easy.

Eventarc supports many Google Cloud products as event sources. For some Google Cloud services, the source can directly send events to Eventarc. For Google services or event types that do not support direct access, Eventarc can seamlessly use Cloud Audit Logs entries to generate events. Third-party providers can also utilize the Eventarc API to create events. Applications don't need to write code to parse audit logs or poll for events.

Pub/Sub messages to specified topics may also be used as event sources. This integration allows custom applications to send events.

An event trigger is a rule-based filter to route specific event types from a particular event source to specified event consumers, or targets.

Eventarc uses Pub/Sub as its transport layer due to its superior reliability and observability. Eventarc automatically creates and manages Pub/Sub topics and subscriptions to facilitate event delivery. Your applications only need to accept HTTP requests that are automatically sent by Eventarc. Applications do not need to use Pub/Sub directly.

Events are delivered to targets using the standard CNCF CloudEvents format, regardless of event source. Whenever you create a trigger, you can see the format

and fields that will be used for that specific event.

Eventarc: https://cloud.google.com/eventarc
Eventarc third-party providers:
https://cloud.google.com/eventarc/docs/third-parties/third-party-entities

Using Eventarc CloudEvents in your applications

CloudEvents:

- Specifies a common format for describing event data.
- Allows developers to use the same event handling logic regardless of the event source or type.
- Provides SDKs for many common programming languages.

The CloudEvents format used by Eventarc specifies a common metadata format for describing event data, providing interoperability across services, platforms, and systems.

Publishers of events historically tended to use their own formats for their events. By standardizing on a specific format, CloudEvents allows developers to use the same event handling logic in their code regardless of the source or type of the event.

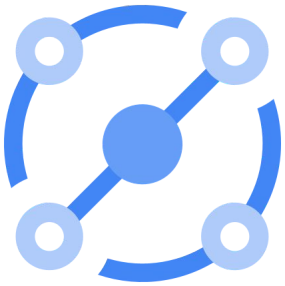
CloudEvents provides SDKs for many of the common programming languages in use today, like Python, JavaScript, Java, Go, C#, Ruby, and PHP. Developers can use these SDKs to easily parse incoming events, increasing code portability and developer productivity.

CloudEvents: https://cloudevents.io/
CNCF CloudEvents repository: https://github.com/cloudevents/spec
Google Cloud CloudEvents: https://github.com/googleapis/google-cloudevents

When to use Eventarc

Use Eventarc when you want to:

- Easily select events from a wide range of integrated sources.
- Use a simple rule-based interface to select source, filter, and destination, instead of using code to ingest events and manage topics and subscriptions.
- Consume events using a standard CloudEvents format.

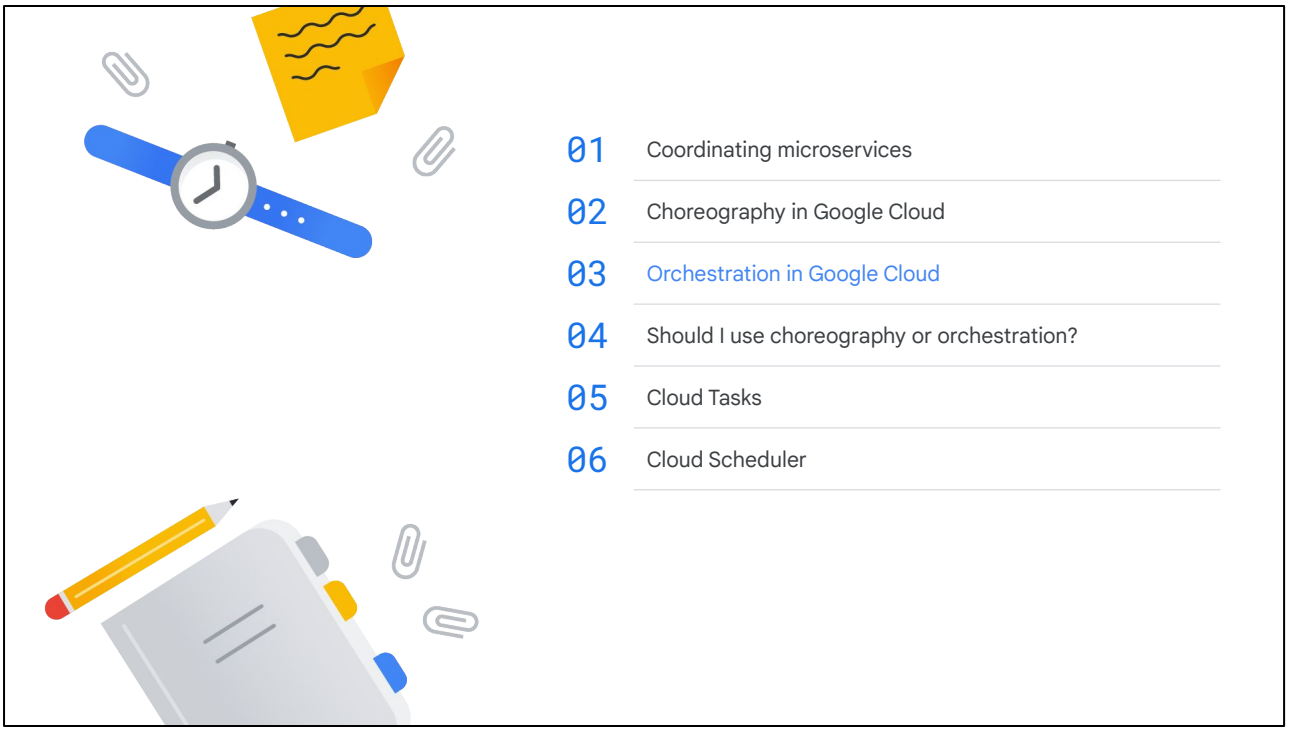


Eventarc is an abstraction layer on top of Pub/Sub that provides some significant benefits when designing an event-driven application.

Eventarc can use many built-in services as a source for events. Eventarc makes it easy to detect changes within many Google Cloud services and third-party applications, and automatically trigger code to run in response to those changes.

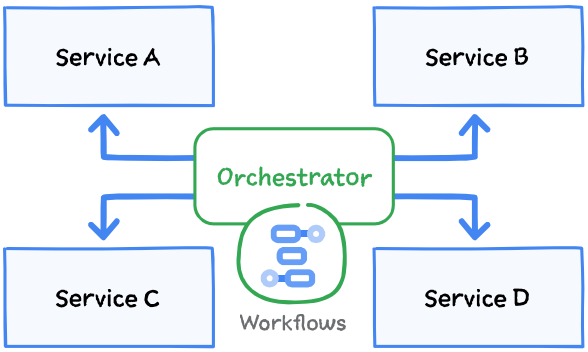
Eventarc provides a simple, rule-based interface to select the source, filter, and destination of a trigger. With Pub/Sub, you must write code to ingest events and manage topics and subscriptions.

Finally, you should use Eventarc when you want a standardized event format. By using the standard CloudEvents format, you can use CloudEvent SDKs to create and consume events.



Google Cloud's Workflows platform can be used to implement the service orchestration pattern.

Workflows orchestrates calls to multiple services to create business processes



- Workflows is a fully managed orchestration platform.
- A workflow combines Google Cloud services and APIs to create stateful, automated processes.
- Each execution of a workflow is observable.
- A workflow can hold state, retry, poll, or wait for up to a year.

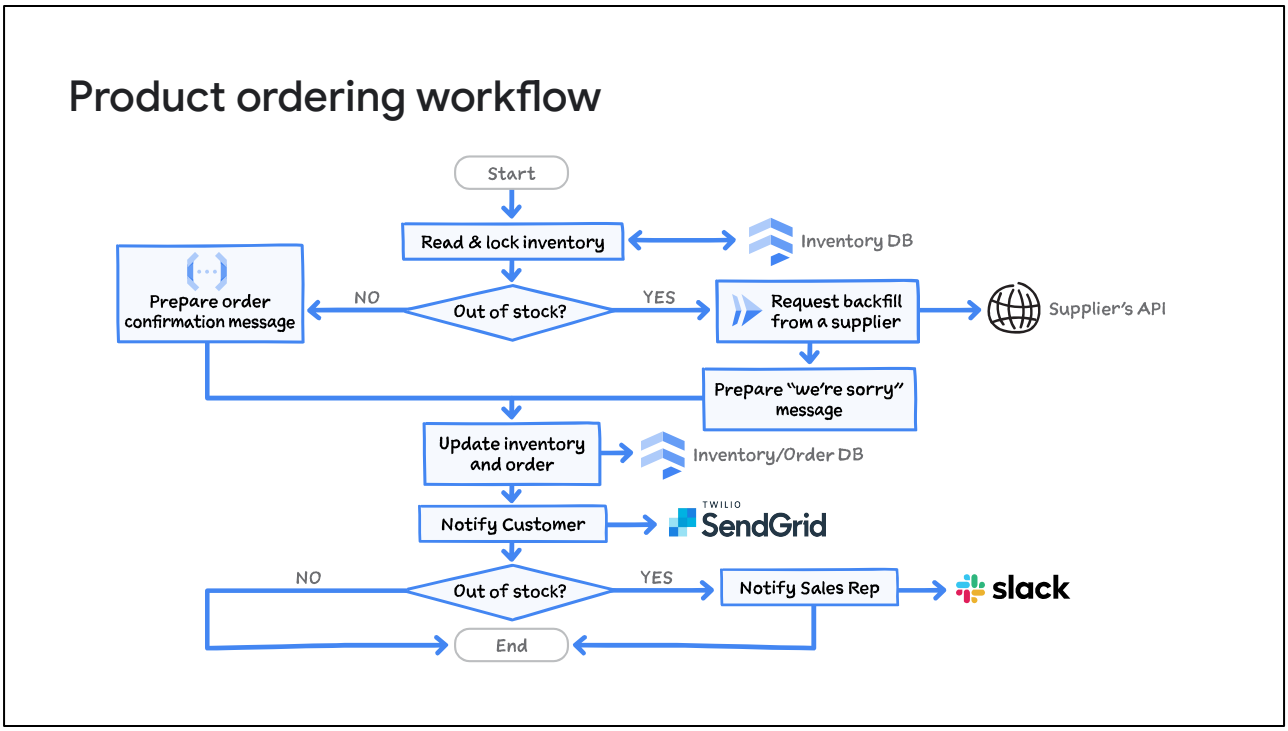
Workflows is a fully managed orchestration platform which acts as the central orchestrator for the service orchestration pattern.

You design and deploy workflows, which orchestrate Google Cloud services and API calls, to build stateful, automated processes.

A workflow provides a central source-of-truth for the application flow. Each execution of a workflow is logged and is observable, which makes it easier to understand the current state of the workflow and troubleshoot any issues.

A workflow can hold state, retry, poll, or wait for up to a year. This flexibility allows for creation of long-running business processes.

Workflows: https://cloud.google.com/workflows



Here's an example product ordering workflow.

The first step in the workflow process is to check the Firestore inventory database for availability of the ordered products. If the items are available, they are locked.

The workflow follows one of two paths depending on whether any of the items are out of stock. The out of stock boolean condition is available throughout the workflow, and is also used at the end of the process.

If every item in the order is available, the workflow calls a Cloud Run function to prepare the order confirmation message. If one or more items is out of stock, a Cloud Run service is triggered to request more inventory from the relevant suppliers. After the Cloud Run service is called, a "we're sorry" message is prepared for the customer.

Next, the inventory and order details are updated in the Firestore databases. The customer is then sent an email indicating whether the order was successful or not.

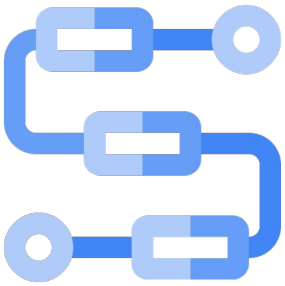
Finally, if one or more items is out of stock, a Slack message is posted to notify the sales rep.

Each workflow execution, which handles a single transaction, is logged for troubleshooting or tracing. The workflow handles retries or exceptions thrown by APIs, improving the reliability of the entire process.

When to use Workflows

Use Workflows when you want to:

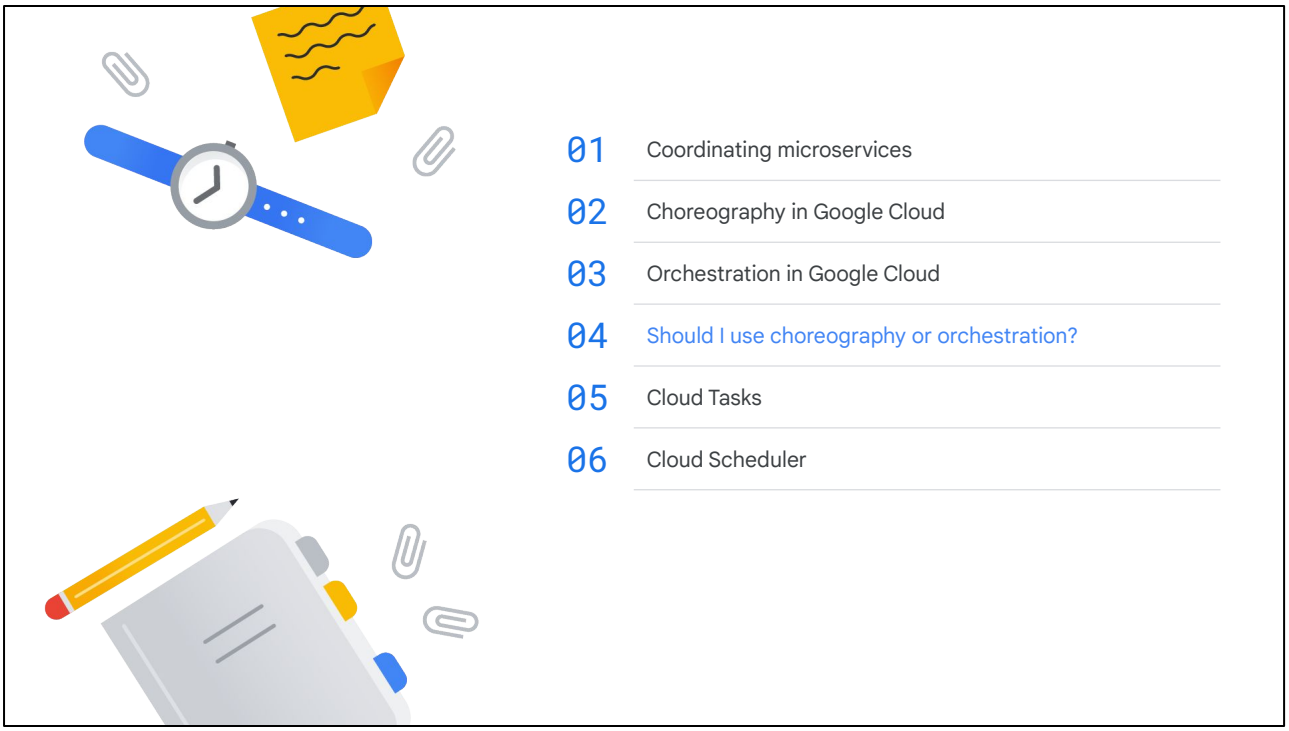
- Chain HTTP-based microservices and Google Cloud services into durable and stateful workflows.
- Maintain observability throughout the life of a long-running execution.
- Perform operations on a set of items or batch data.



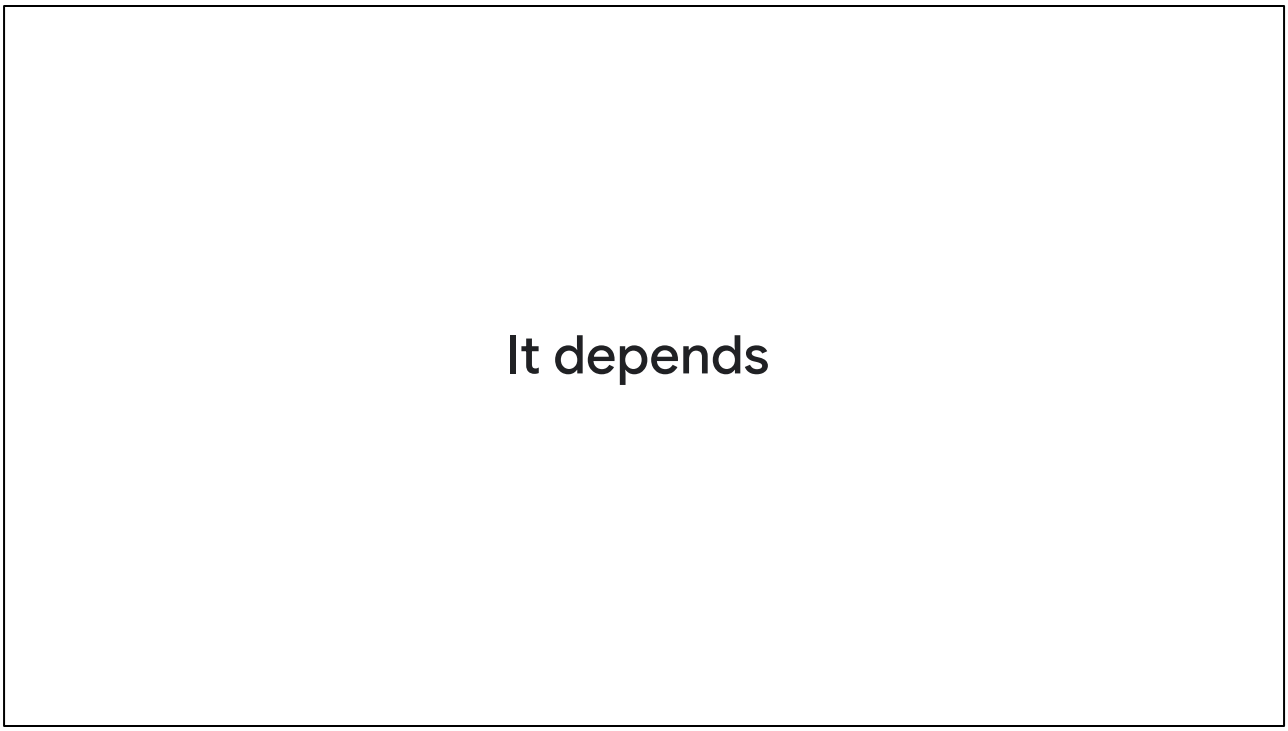
Workflows is an excellent choice when you want to chain HTTP-based microservices into durable and stateful workflows.

Workflows lets you implement long-running processes and maintain observability for each execution.

Workflows is also a great choice for performing operations on a set of items or batch data. Robust error handling can guarantee that each item is processed correctly.



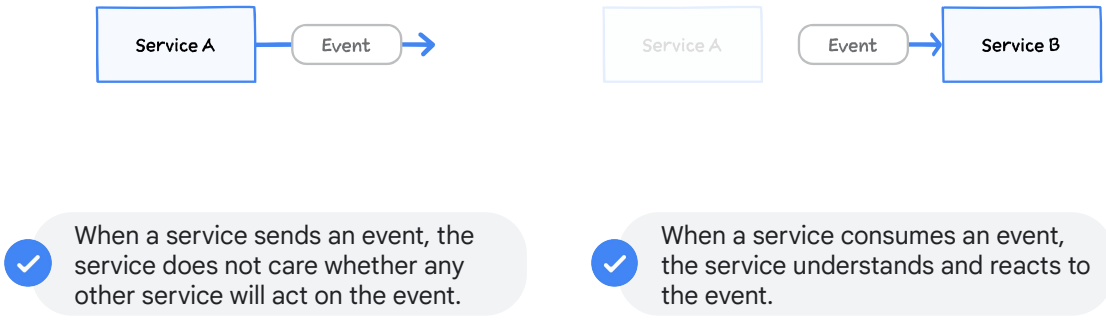
Now that you've learned how choreography and orchestration work in Google Cloud, which pattern is better, choreography or orchestration?



Like many application architecture decisions, the answer is "it depends."

The better question is "when should I use choreography, and when should I use orchestration?"

Event-based choreography is controlled by the event receiver

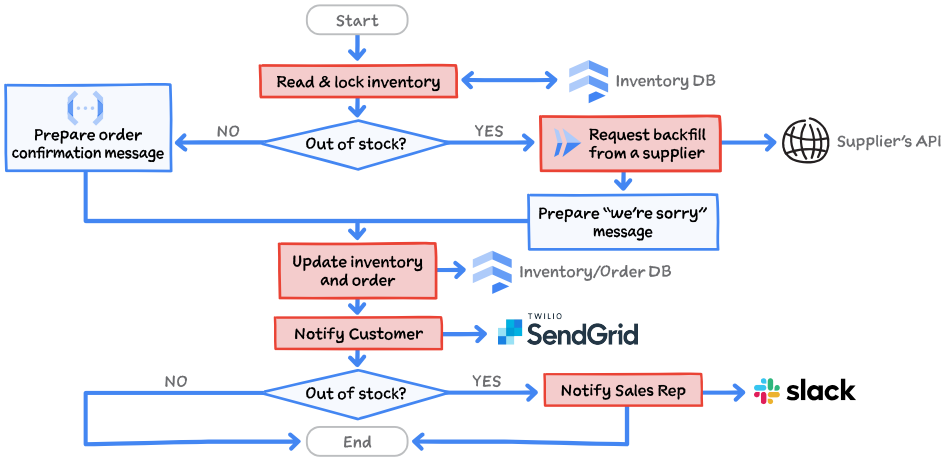


When you use event-based choreography, you rely on the receiving service to control the communication.

When Service A completes some work and sends an event, Service A has no idea whether any other service will act on that event. It is not the responsibility of Service A to know whether any downstream service or services are consuming the event. Using Eventarc, most Google Cloud or custom services can act as an event producer.

A receiver of an event, though, must understand the details of the event to be consumed. In the example shown here, Service B consumes the event that was sent by Service A. Service B understands the formatting of the event and how to act on the event. Service B might also know that the event was sent by Service A, but it is not directly coupled to Service A.

Orchestration is preferable for complex, compound processes

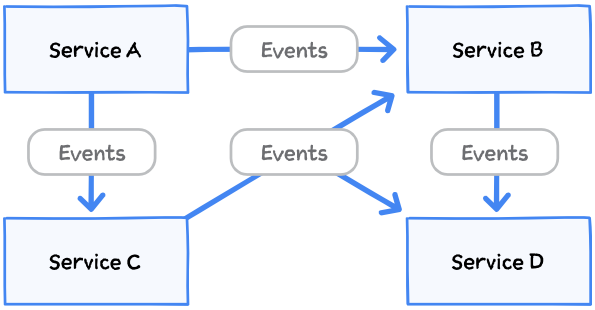


It's possible to implement the product ordering flow using choreography. Each service would send an event to indicate readiness for the next step in the process. The "out of stock" decision points could be implemented by specific services, deciding whether to send the "in stock" or "out of stock" events.

However, an enterprise order system requires visibility, error handling, and retries. Implementing these features in an event-based solution can be difficult. How would you troubleshoot issues with an event-based system? What happens if the process somehow aborts between locking the inventory in the database and updating the inventory and order? How do you make sure that requests are successfully sent to suppliers?

With orchestration, each workflow execution is tracked separately. The ordering process logic, which may be built using many services or functions, is defined in a single location. Retries and error handling can provide a reliable order handling flow.

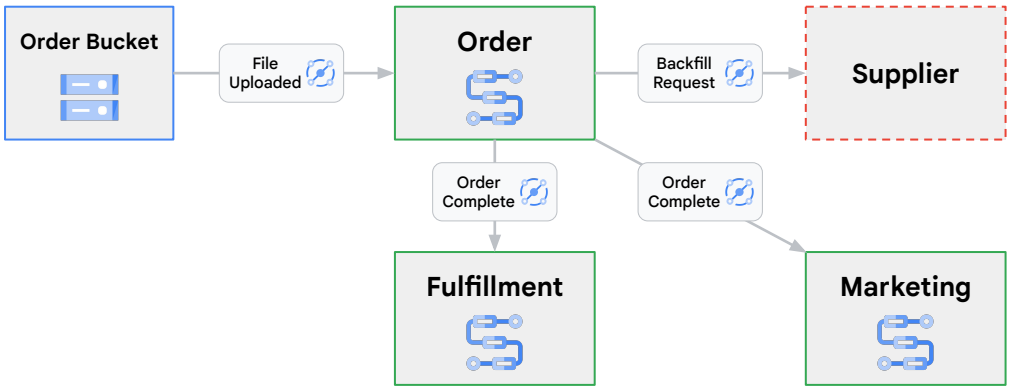
Choreography provides decentralized control



Sometimes, your architecture requires decentralized control which cannot be provided by a single orchestrated workflow. With orchestration, if these services are built and managed by different teams or organizations, shared management of the central orchestration process can be difficult.

Choreography provides decentralized control, where each of the services or applications connects to the others using events. Each service can be separately managed by a different team or organization.

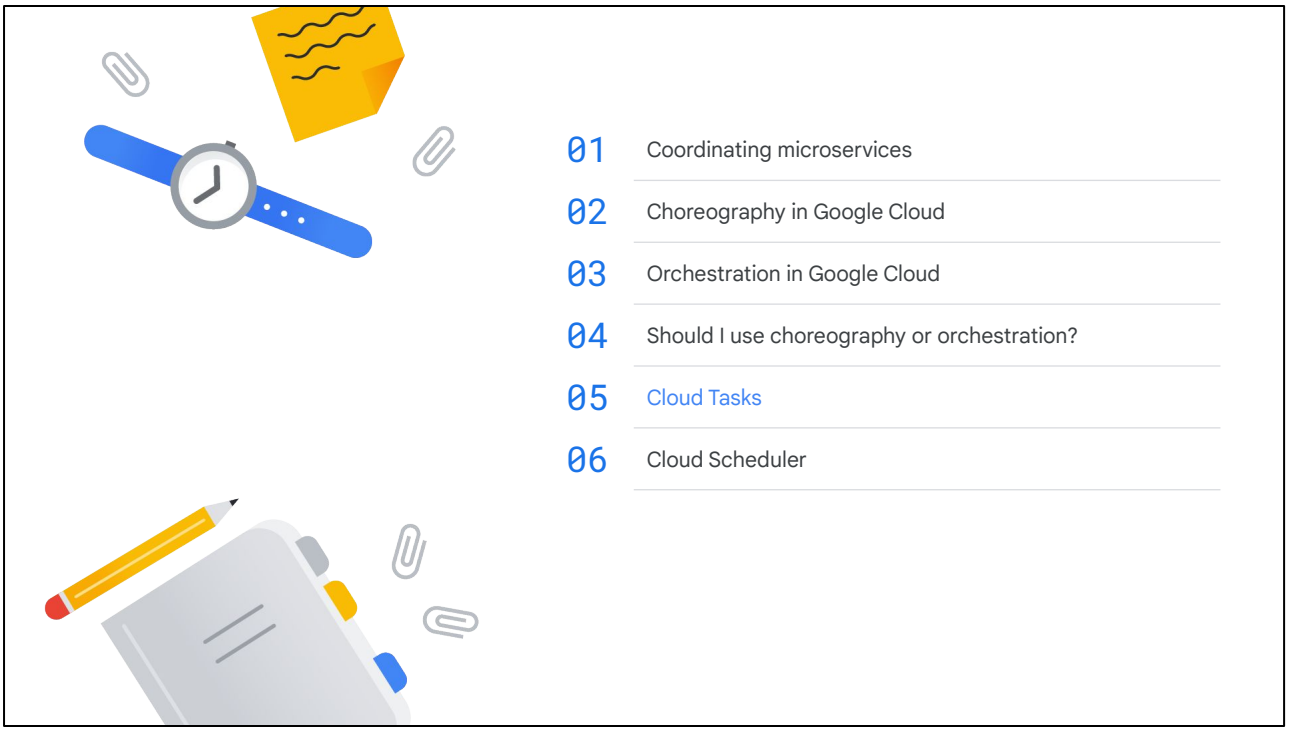
Choreography and orchestration can be used together



Orchestration is a strong pattern when you want to control a complex process that you can manage centrally. Choreography is more appropriate when combining separate, decentralized services and applications using events, or when you want to leverage events from Google Cloud services.

You may find that using Workflows and orchestration provides the visibility, error handling, and reliability required to create your complex services. You can use Eventarc to send events between the services.

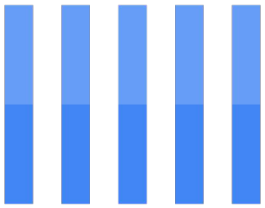
In the example shown here, the Order, Fulfillment, and Marketing services are implemented using Workflows. Event triggers between the services and the detection of new order files being uploaded into a Cloud Storage bucket are implemented using Eventarc.



Cloud Tasks lets you manage the execution, dispatch, and delivery of large numbers of distributed tasks.

Create distributed task queues with Cloud Tasks

- Pieces of work that can be processed independently are queued and then dispatched to HTTP services.
- Offloaded tasks are added to a queue, where they stay until the task is completed.
- Task delivery time, dispatch rate, and retry behavior can be specified for reliable processing.
- At-least-once delivery is guaranteed, and any task added multiple times is only sent once.
- OpenID Connect identity tokens or OAuth access tokens can be attached to the HTTP service call.



A task is a piece of work that can be processed independently by dispatching it to an HTTP service.

You can set up a queue for tasks that will be sent to a particular service. A task added to this queue will be automatically dispatched to your chosen HTTP service. The returned status code returned determines whether the task has been completed successfully or should be retried.

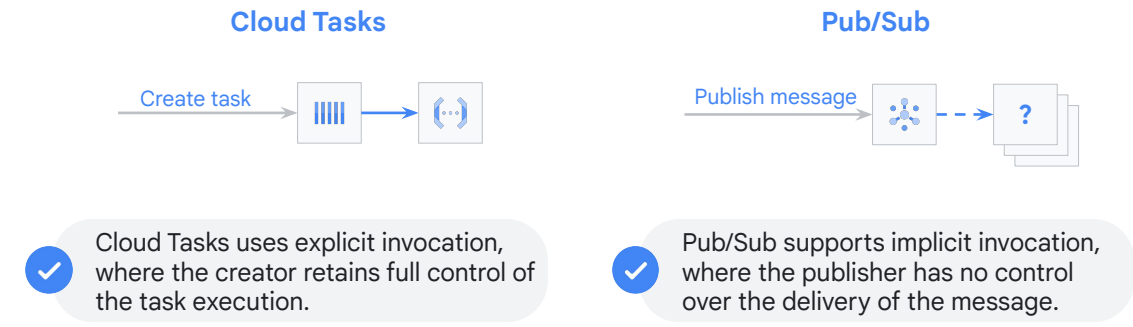
When sending a task to a queue, you can schedule the task to be dispatched at a specified future time. You can configure the queue with a maximum rate to dispatch tasks or a maximum number of tasks that can be dispatched concurrently. You can also specify the maximum number of attempts and the delay between attempts when tasks are retried.

Cloud Tasks also guarantees at-least-once delivery and eliminates any duplicate tasks that are created.

HTTP services that require authentication may be called by automatically attaching a token associated with the service account of the application creating the task.

Cloud Tasks: https://cloud.google.com/tasks
Using HTTP Target tasks with authentication tokens:
https://cloud.google.com/tasks/docs/creating-http-target-tasks#token

Cloud Tasks and Pub/Sub are conceptually similar, but designed for different use cases

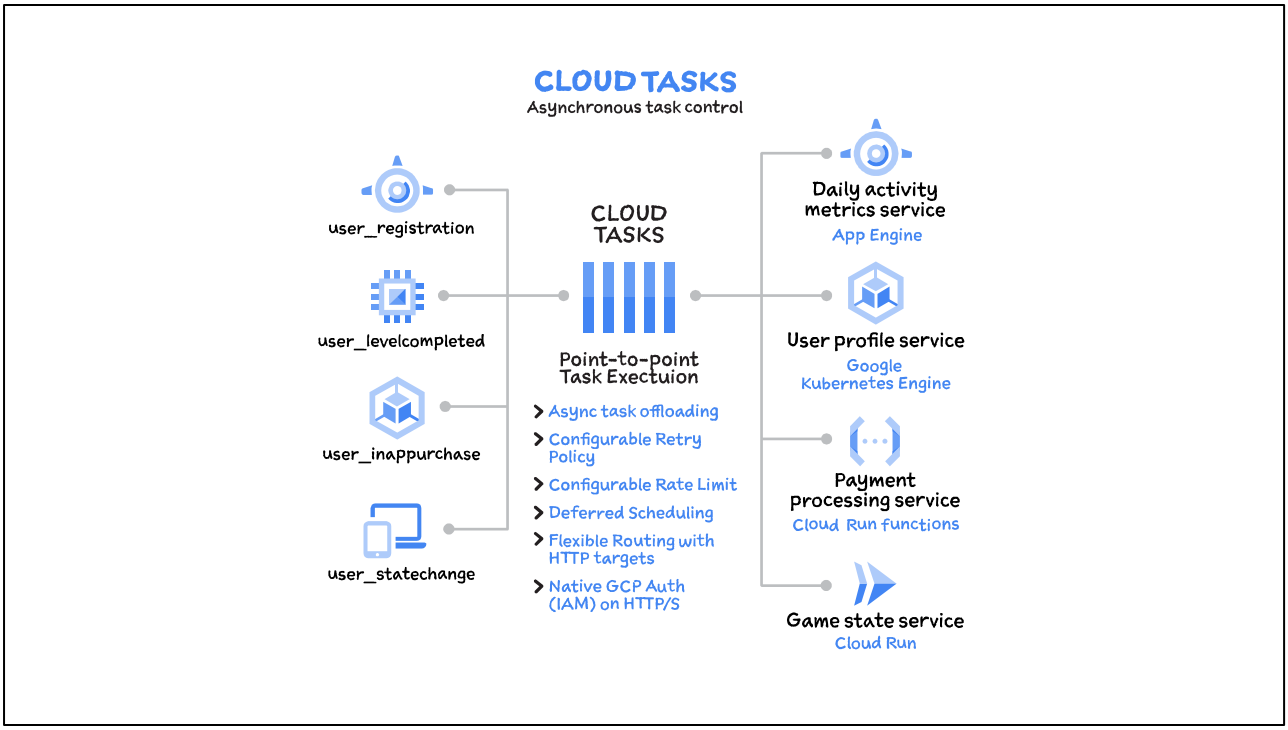


Although Cloud Tasks and Pub/Sub are conceptually similar, both implementing message passing and asynchronous integration, they are designed for different use cases.

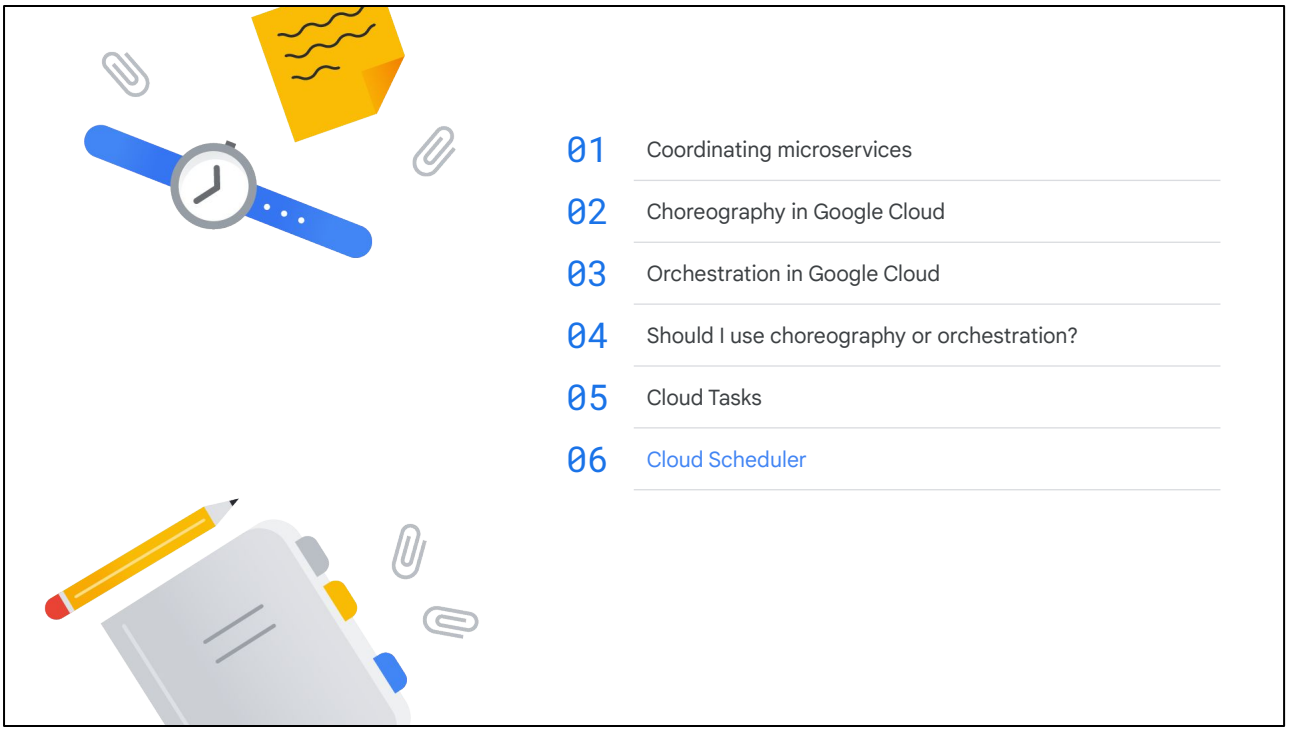
With Cloud Tasks, the task creator uses explicit invocation, where the creator retains full control over the execution and destination of the task. The creator places the task in a queue attached to a specified endpoint, and the creator can defer the dispatching of the task.

With Pub/Sub, the message publisher uses implicit invocation. The publisher implicitly causes any subscribers to execute by publishing the message, but the publisher has no control over which subscribing services receive the message.

Cloud Tasks is appropriate for use cases where an application wants to asynchronously execute a specific service, and possibly control the timing of the execution. Pub/Sub should be used for event-based architectures that rely on receiving services reacting to events generated by other services.



Cloud Tasks does a great job of separating out independent pieces of work to be processed asynchronously. Slow background operations can be offloaded to a separate worker, reducing the work required by the main application and improving response time. Unlike Pub/Sub, the application retains control of the execution, specifying the endpoint to handle the offloaded task. Cloud Tasks lets you configure retries, scheduling, and rate limiting.



Another managed service that can be used in your orchestration and choreography designs is Cloud Scheduler.

Schedule virtually any job with Cloud Scheduler

- Set up scheduled jobs to be executed at defined times or regular intervals.
- Job schedules are specified using a format based on Unix cron jobs.
- Jobs may be sent to Pub/Sub topics, App Engine applications, or publicly accessible HTTP endpoints.
- Jobs have guaranteed execution and can be retried when the execution fails.

Cloud Scheduler is a fully managed, enterprise-grade cron job scheduler which can be managed from a single dashboard.

Cloud Scheduler is used to schedule jobs that should be executed on a defined schedule or at regular intervals.

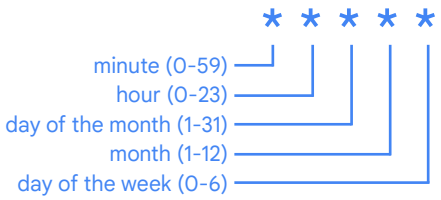
Jobs are specified using the familiar Unix cron job format, allowing a job to be run multiple times a day or on specific days and months.

Jobs can be sent to Pub/Sub topics, App Engine applications, or publicly accessible HTTP endpoints. Like Cloud Tasks, Cloud Scheduler can attach tokens associated with a specified service account to HTTP requests.

Jobs also have guaranteed execution, and failed job executions can be retried.

Cloud Scheduler: https://cloud.google.com/scheduler

Jobs are scheduled using the Unix cron string format



*** * * * ***	Every minute
45 23 * * 3,6	Every Wednesday and Saturday at 23:45 (11:45pm)
0 10 * * 1-5	Every weekday at 10:00
0 2 1 */2 *	2:00 on the first day of every other month

The time zone for evaluating the schedule may be specified for a job. UTC is the default time zone.

The Unix cron string format is a set of five space-separated fields on a single line that indicates when a job should be executed.

The first field is the minute. A 15 in this field means that the job would execute 15 minutes past the hour.

The second field is the hour. A 0 in this field indicates that the job executes within the hour following midnight.

The third and fourth fields are the day of the month and month respectively. These fields limit execution to the specified days of the month and months.

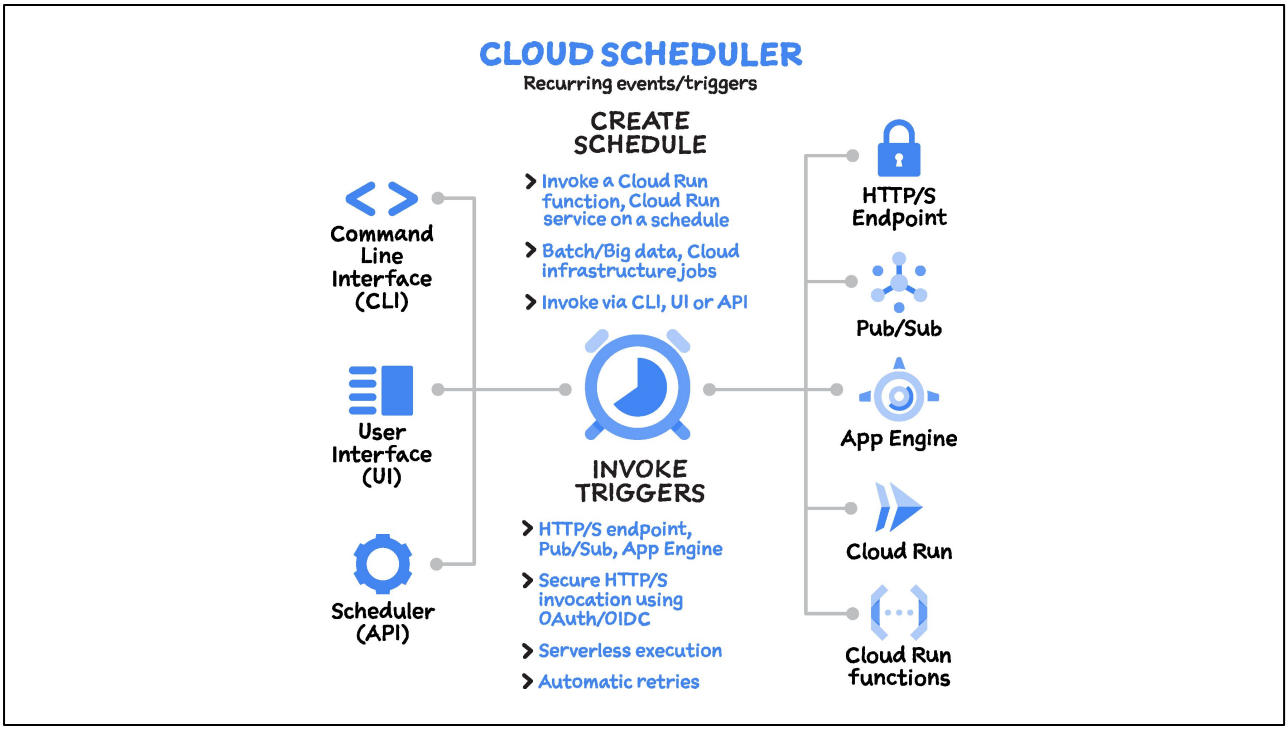
The last field is the day of the week, ranging between 0 for Sunday and 6 for Saturday.

A field can contain a single number, a range of numbers, or a list of numbers and ranges. Ranges are two numbers separated by a hyphen, and the range is inclusive. An asterisk indicates the entire allowed range.

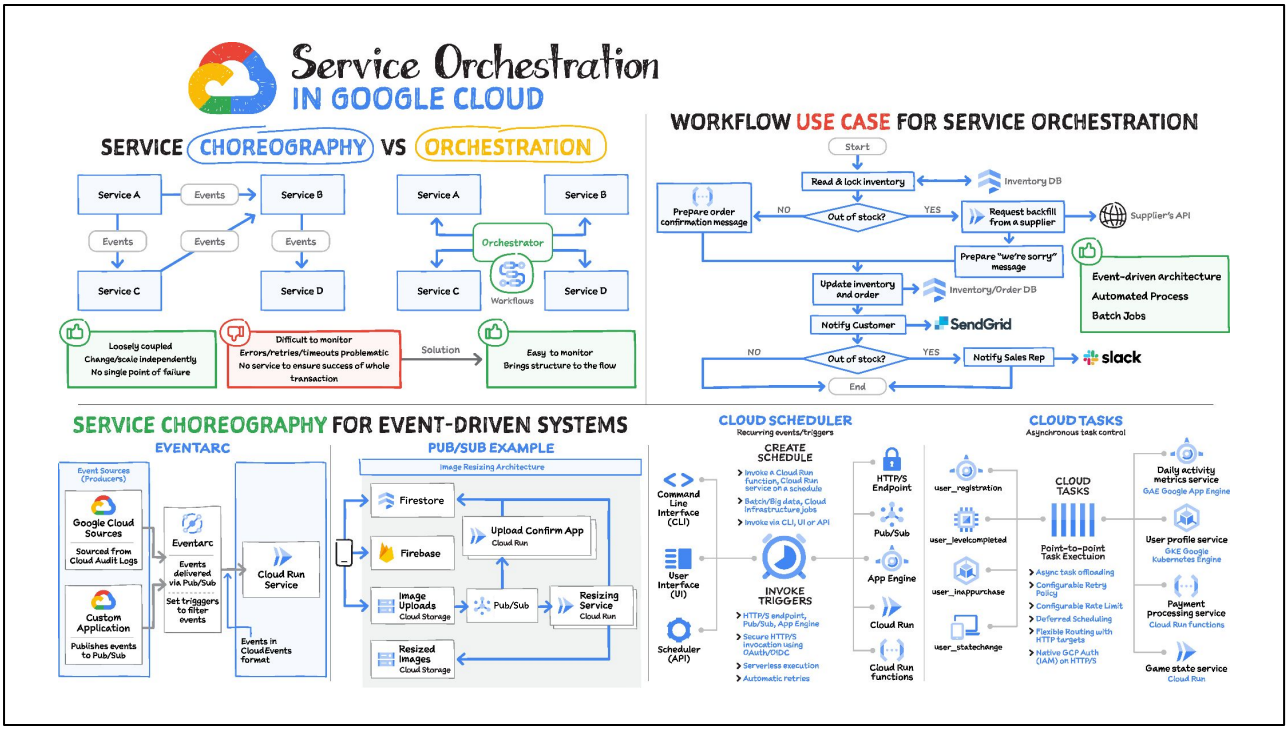
Following a range with a slash and a number will skip that number of values throughout the range. An hour setting of "* / 2" would indicate that the execution should occur every two hours. A list of numbers or ranges can be specified by separating the values with commas.

Your job schedule can be specified for a specific time zone. The default timezone is UTC. Time zones with daylight saving time can cause jobs to be skipped (when clocks are set forward) or run twice (when clocks are set backward). Using the UTC time zone is recommended to avoid this issue.

Configure cron job schedules:
https://cloud.google.com/scheduler/docs/configuring/cron-job-schedules



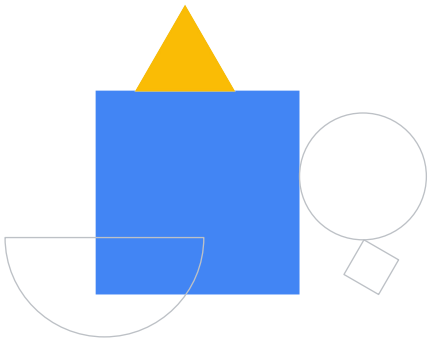
Cloud Scheduler is Google Cloud's solution for scheduling recurring jobs. Scheduled jobs are managed from a single dashboard. Jobs are created using the industry-standard unix-cron format, and can trigger Pub/Sub messages and securely call Cloud Run functions, Cloud Run services, or HTTP endpoints.



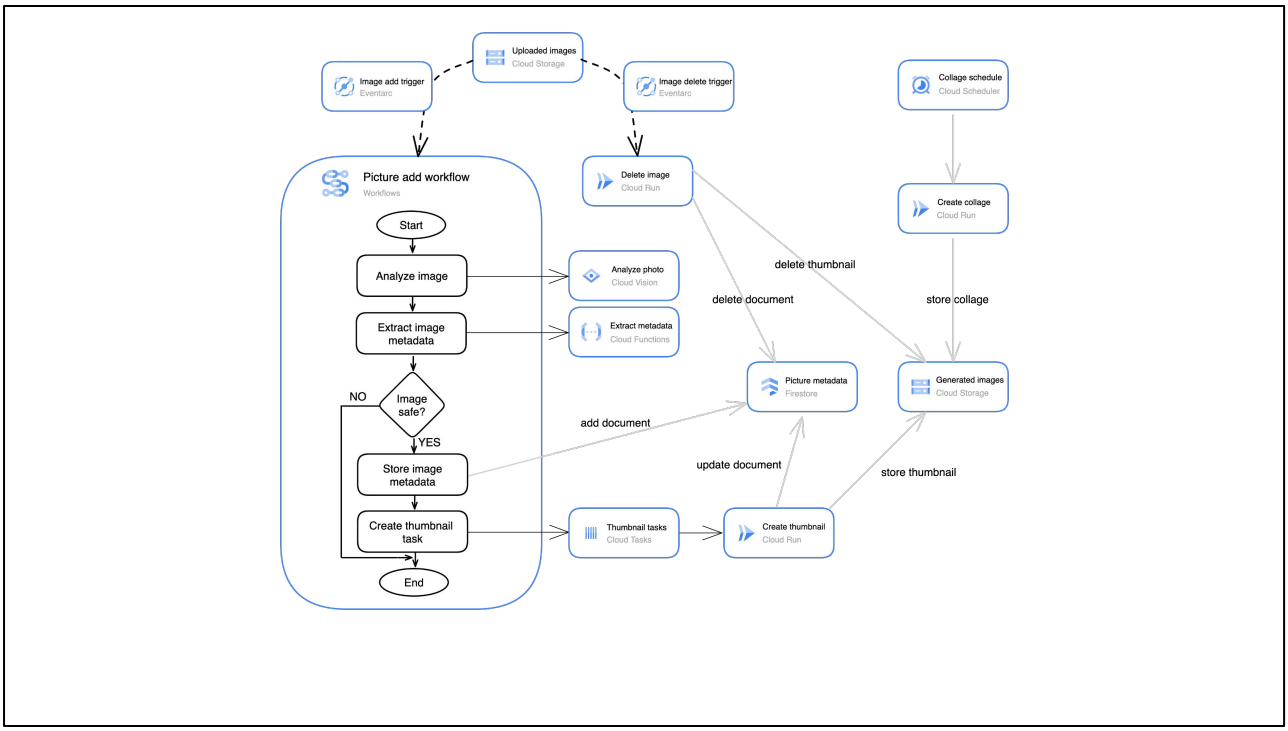
Many of the previous examples come from this sketch note "Service Orchestration in Google Cloud." This sketch note introduces the tools that make up the Application Integration toolbox. A full-featured application developed using a microservices architecture may benefit from any or all of these services. Now you understand when to use choreography and orchestration, and how these application integration products can help you create compelling and maintainable applications.

Lab Intro

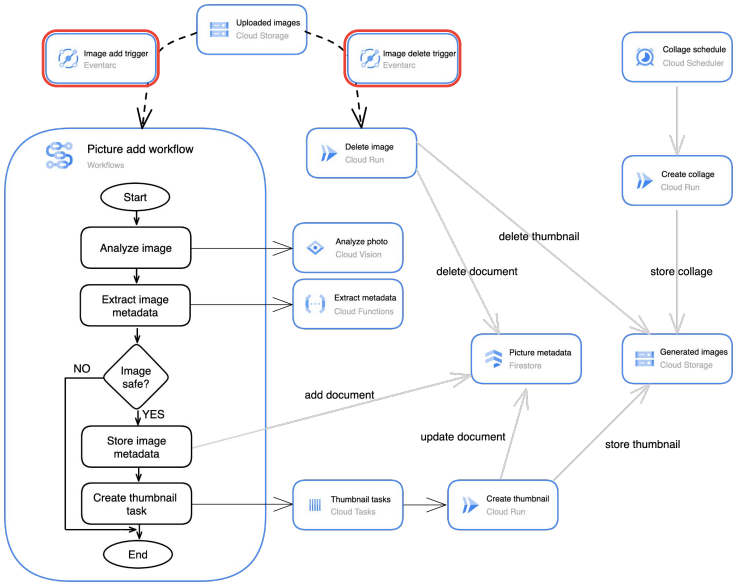
Building Service Orchestration
with Google Cloud



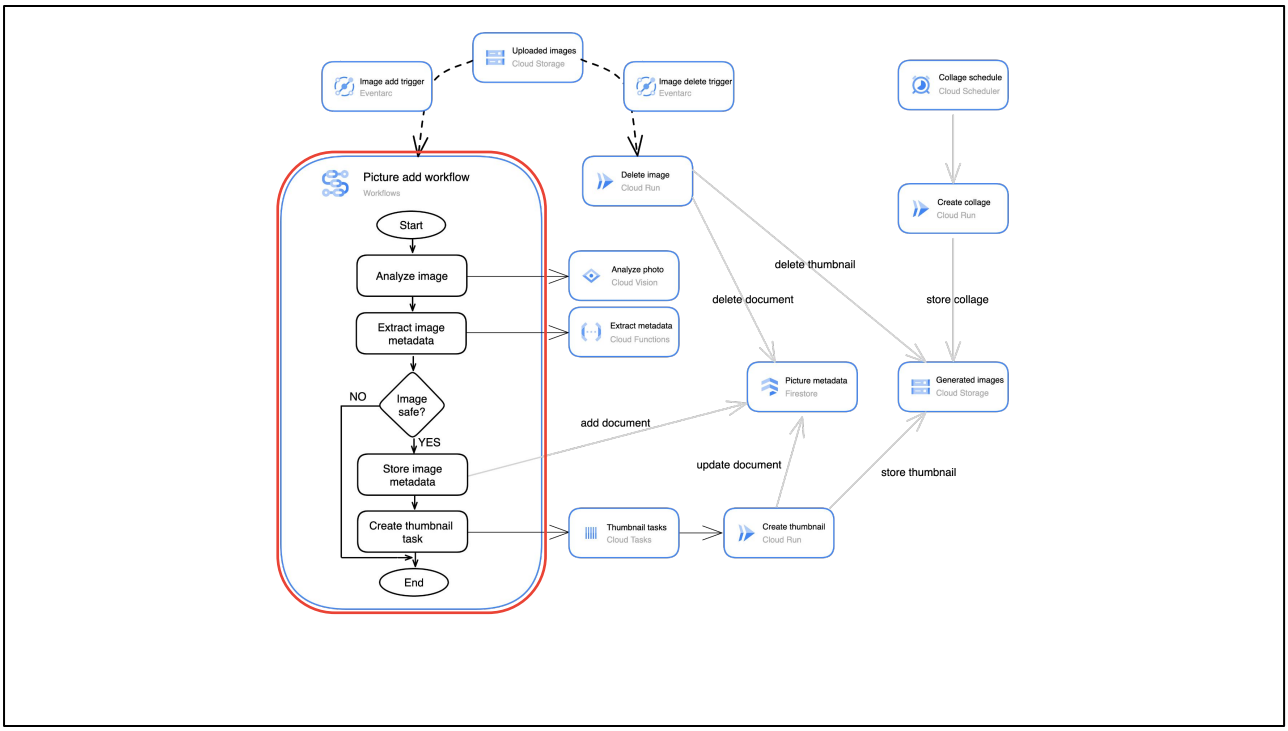
Workflows, Eventarc, Cloud Tasks, and Cloud Scheduler provide a powerful set of features that can be used to create exciting applications when using a microservices architecture.



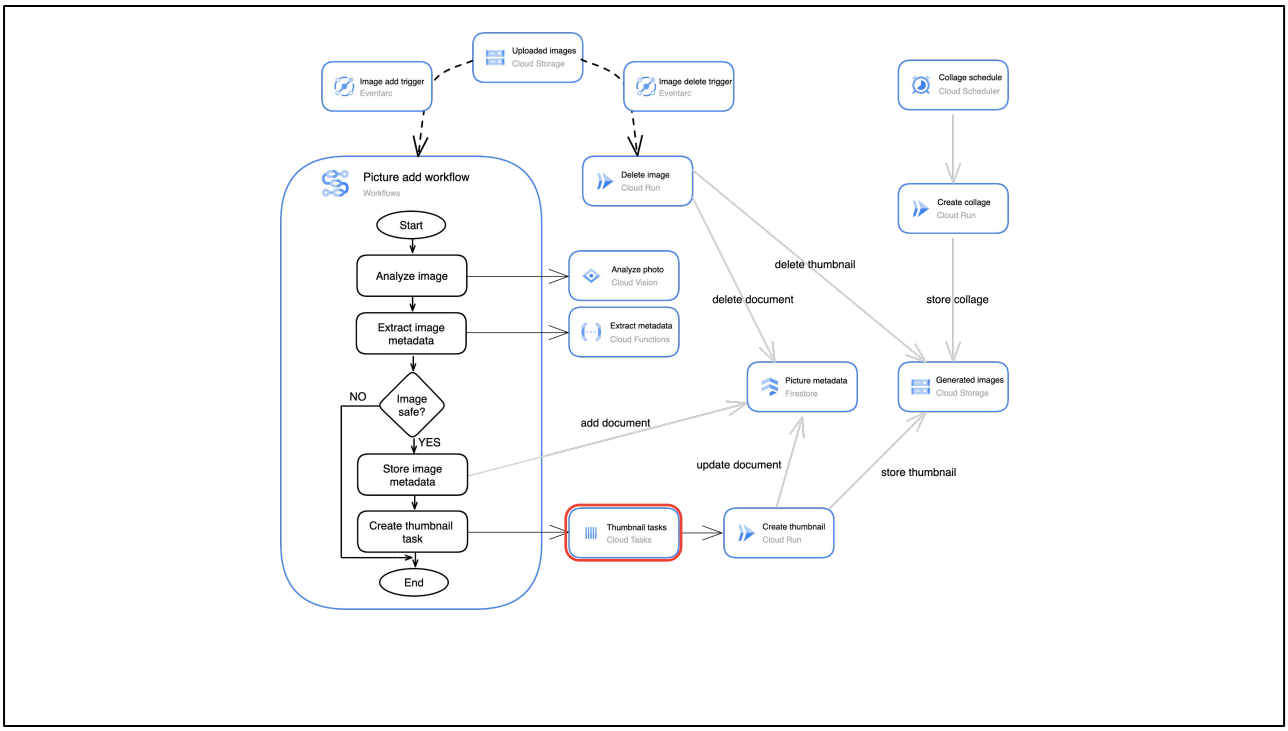
In this lab, you use these four services to implement a series of steps that occur when an image is added to a Cloud Storage bucket.



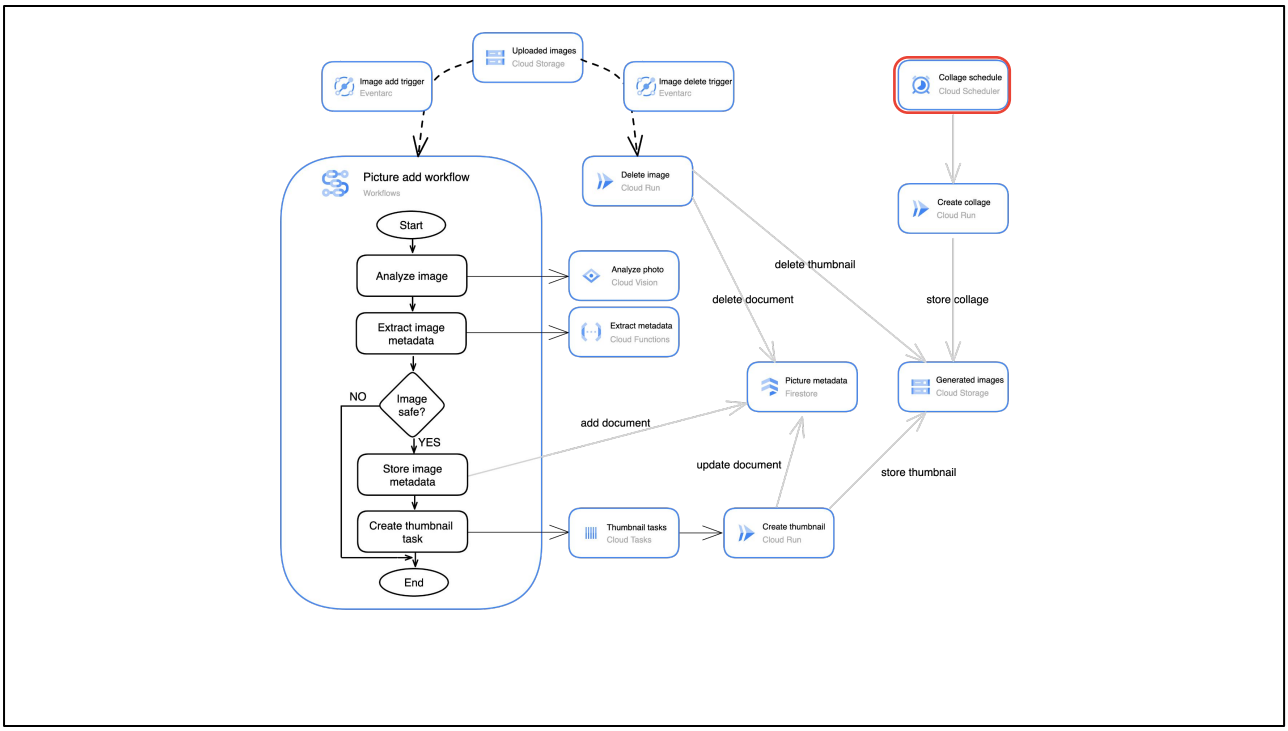
You create an Eventarc trigger that detects when an image file is uploaded to a Cloud Storage bucket. The trigger starts a workflow to analyze the image and create a thumbnail. Another Eventarc trigger will run a Delete Image service in Cloud Run when an image file is deleted from the bucket.



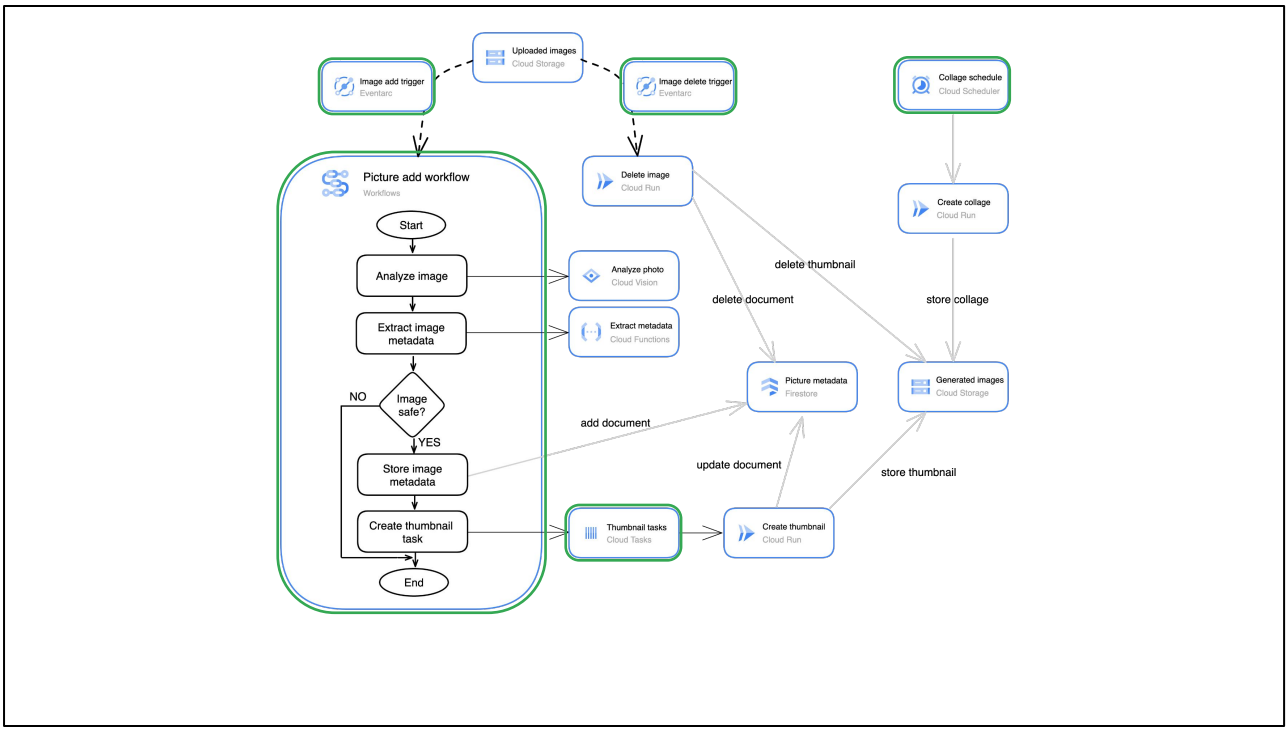
You create a workflow that receives the file creation event from Eventarc. This workflow calls Cloud Vision to analyze the photo, and then the workflow extracts metadata from the Cloud Vision response. If the image content is safe, the image metadata is stored in Firestore, and a Cloud Task is queued to create an image thumbnail.



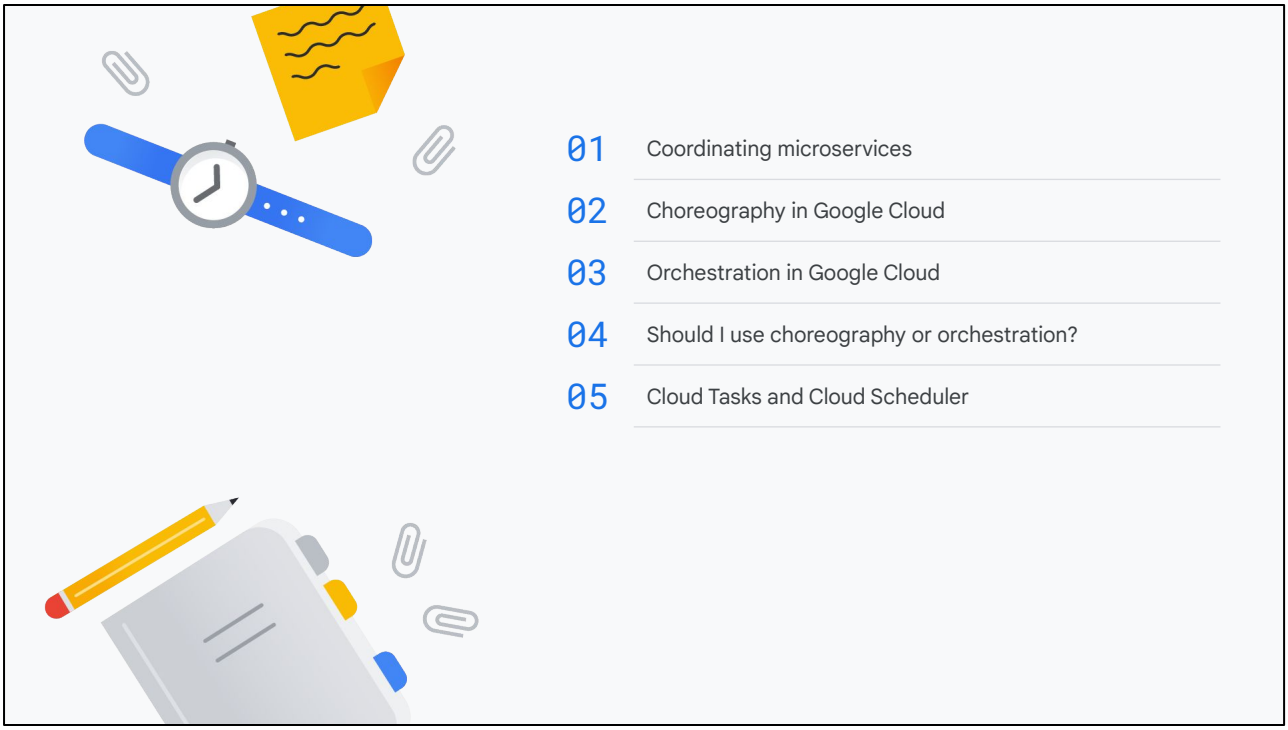
The Cloud Task starts a Cloud Run service that creates a thumbnail of the uploaded image.



Finally, Cloud Scheduler starts a Cloud Run service every minute to create a collage of the latest image thumbnails.



This lab shows how easy it is to build compelling applications in Google Cloud using Eventarc, Workflows, Cloud Tasks, and Cloud Scheduler.



You've reached the end of the course "Service Orchestration and Choreography in Google Cloud."

In this module, "Choreography and Orchestration," you learned about two popular microservices coordination patterns, choreography and orchestration, and you learned when you should use each pattern. You learned about how Pub/Sub and Eventarc support choreography, and how Workflows allows for service orchestration. We also discussed Cloud Tasks and Cloud Scheduler. We finished with a lab that demonstrated these patterns and used these application integration services to build an application.