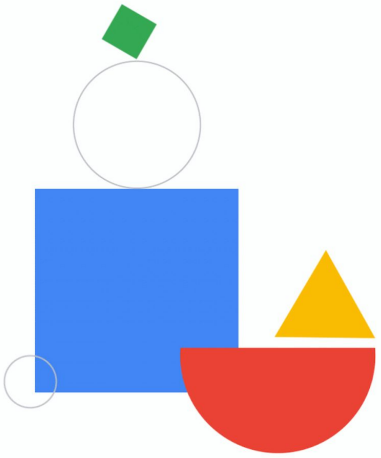
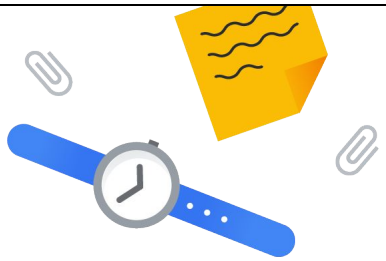
Service Orchestration and Choreography on Google Cloud

Module 2: Event-Driven Applications



Welcome to module 2 of Service Orchestration and Choreography on Google Cloud: Event-Driven Applications.



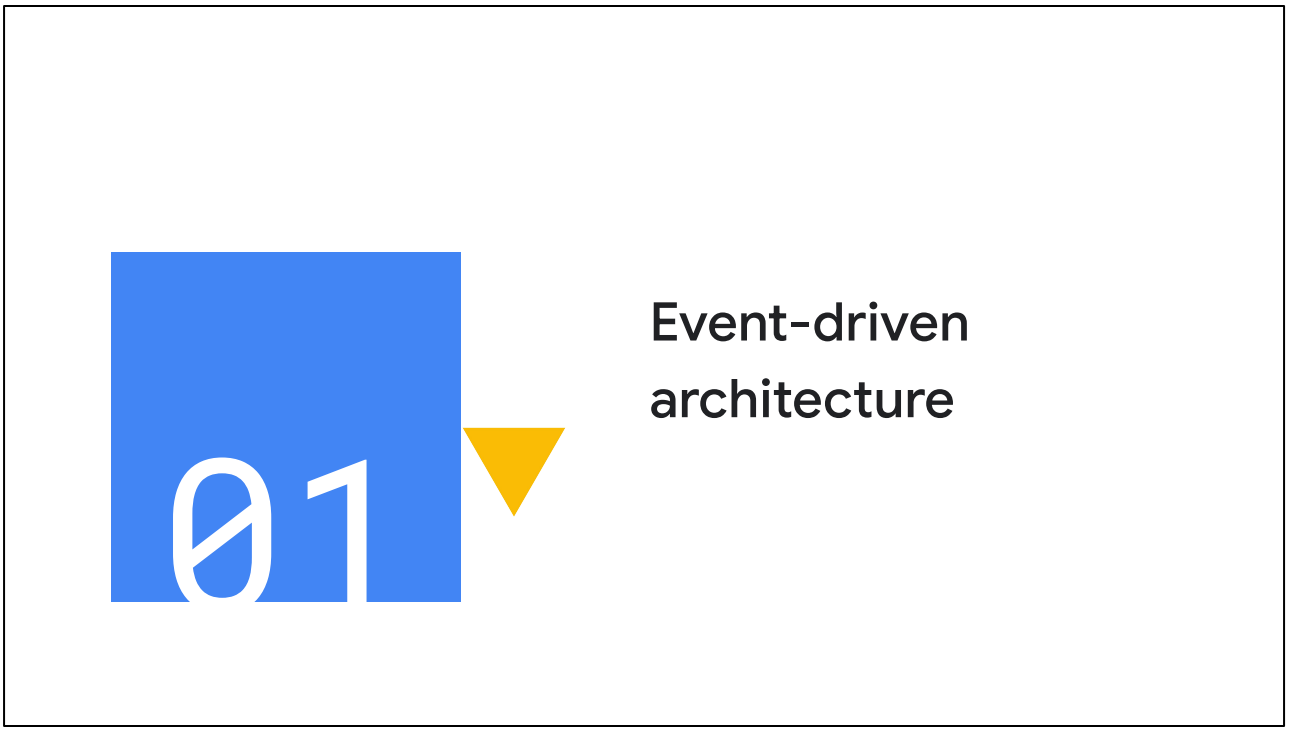
Module agenda

01 Event-driven architecture

02 Benefits of event-driven applications

In this module, you learn about event-driven architecture, a useful pattern for communication when using microservices.

You also learn about the benefits of using event-driven architecture for your applications.

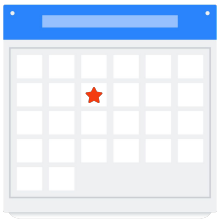


Many applications built using microservices will benefit from an event-driven architecture.

An event is a record of something that has happened

An event:

- Captures an immutable fact that cannot be changed or deleted.
- Occurs whether or not any service consumes it.
- Can be persisted indefinitely and consumed as many times as necessary.



Before we discuss event-driven architecture, we must understand events.

An event is a record of something that has happened. Examples of events are an employee logging in to an application or a product being added to a shopping cart.

That definition may seem obvious. However, when we are discussing event-driven architectures, there are other important attributes of events.

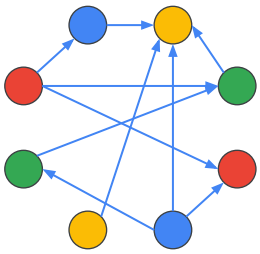
First, an event is typically treated as an immutable fact. It's an historical record of an occurrence, and it should not be modified or deleted.

Second, an event can be generated even if it's never consumed. Many applications that produce events don't know whether the events they generate are ever consumed.

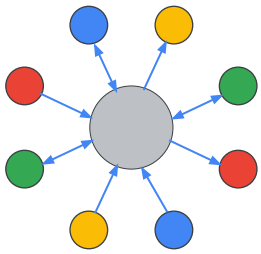
Third, an event can be persisted indefinitely, and can be consumed as many times as necessary. A single event can be consumed by many services, allowing event processing to occur in parallel.

Event-driven architectures:
https://cloud.google.com/eventarc/docs/event-driven-architectures

Event-driven architectures use an event intermediary



Point-to-point communication



Communication using event intermediary

As discussed earlier, the "spider web" of point-to-point communication between microservices can be a challenge. Each service must know how to communicate with all downstream services. Point-to-point communication tends to introduce coupling between the microservices.

An event-driven architecture inserts an event intermediary between the services. When a service acts as an event producer, it sends events to the intermediary. It isn't necessary for the service to know anything about the services that are consuming the events.

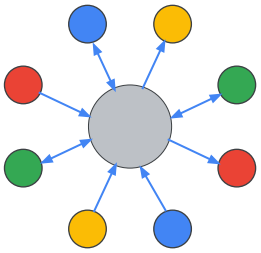
A service can also act as an event consumer, receiving events from the intermediary. Event consumers understand how to handle an event. The consumers do not have to know any details about the event producer.

Benefits of event-driven applications

There are several benefits of event-driven applications.

A centralized event service simplifies auditing and control

- A log of immutable events can be used for auditing purposes.
- The event service can limit who can send or receive specific events.



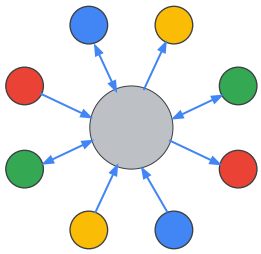
A centralized event service can simplify the auditing and control for a distributed application.

A log of immutable events can be used for auditing purposes. Events can provide a timed, ordered record of every change to the state of an application.

A centralized event service can also help you control access to particular services and data. By requiring authentication and authorization at the event service, you can limit access to each of your event-based services.

Producers and consumers of an event are decoupled

- Producers create events.
- Consumers handle events.
- Point-to-point spider web of communication is eliminated.
- New consumers can be added without modifying existing services.



When using an event intermediary, the producer and consumer of an event are decoupled.

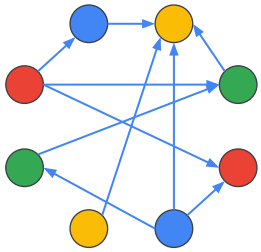
Services can create an event without having to send direct requests to any services that consume the event.

Services can also consume an event without knowing anything about the service that generated the event.

This decoupling means that there's no point-to-point spider web of communication. Each event travels through the event intermediary, which routes the event to the correct consumer or consumers. A producer or consumer is only required to know the format of a specific event.

An extra benefit of this decoupling is that new event consumers can be added to your application without modifying any existing services.

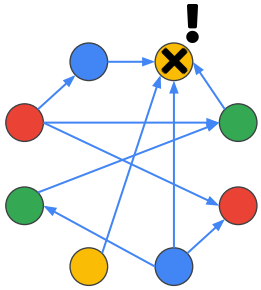
Event-driven applications are resilient



Synchronous request/response microservices architecture

Microservices applications are sometimes designed to use synchronous request/response calls. The health of a service is affected by the health of the services that it calls, directly or indirectly.

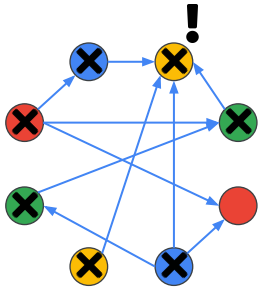
Event-driven applications are resilient



Synchronous request/response
microservices architecture

When a single service fails...

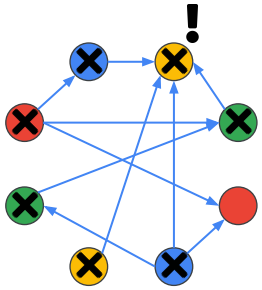
Event-driven applications are resilient



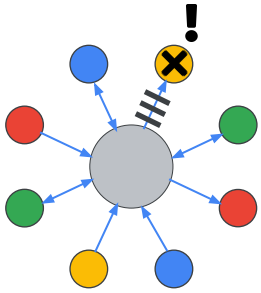
Synchronous request/response
microservices architecture

...it can bring down your entire application.

Event-driven applications are resilient



Synchronous request/response microservices architecture

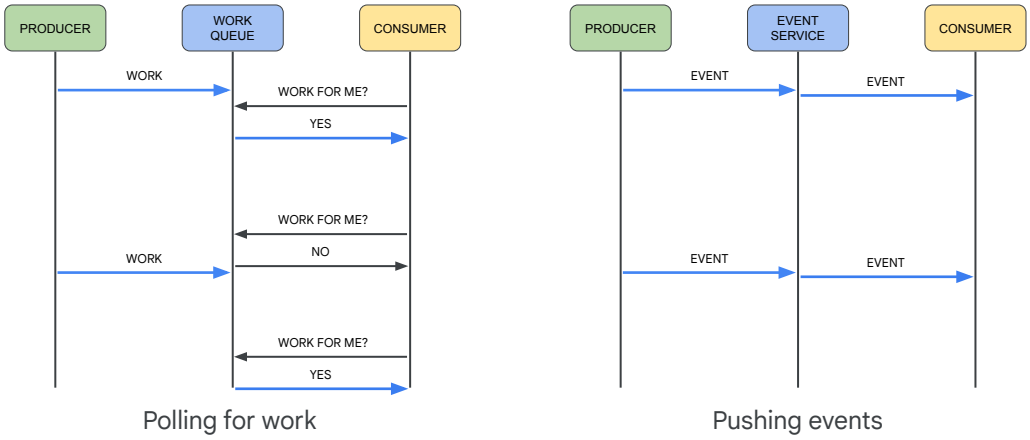


Asynchronous event-based microservices architecture

With an event-based architecture, events are generated asynchronously, and events are created without waiting for a response. An architecture can be designed to survive the temporary loss of a service.

Events sent to the unhealthy service can be replayed or redelivered when the service comes back up. This asynchronous handling of events leads to more resilient applications.

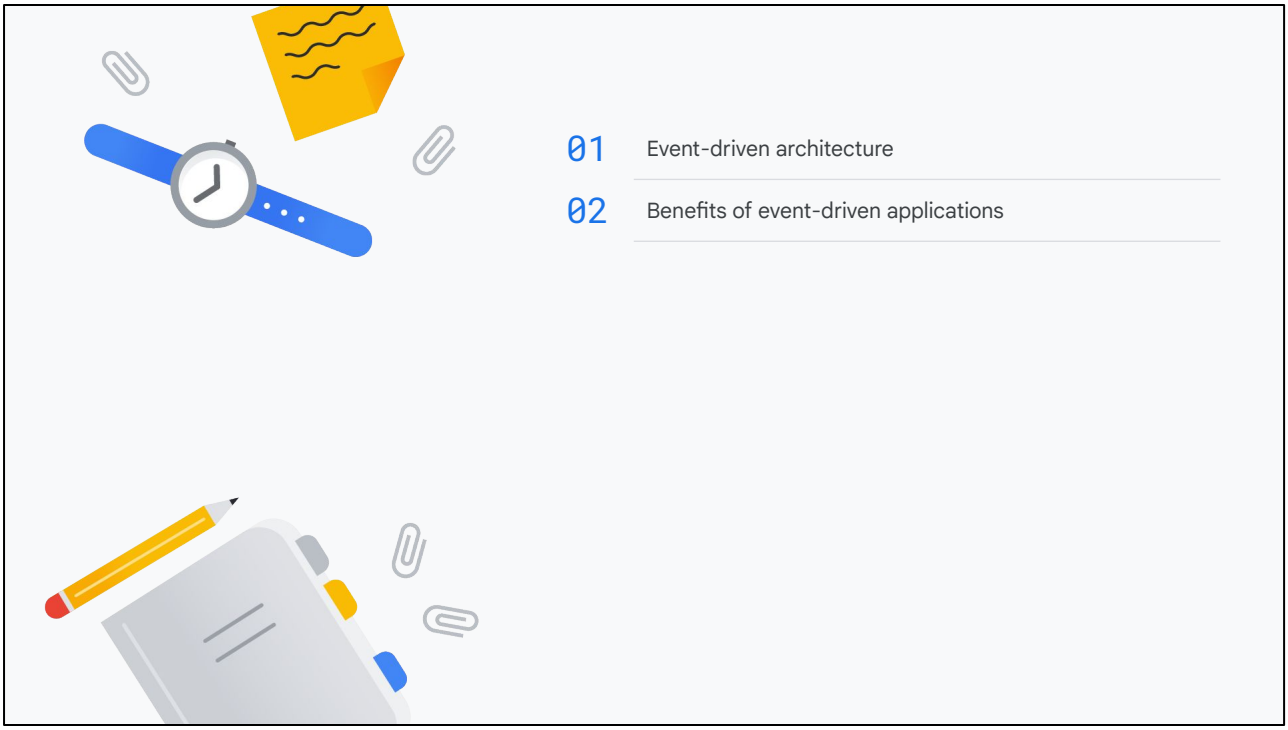
Push-based messaging is efficient



When services are asynchronous, push-based messaging allows clients to receive updates without needing to continuously poll remote services.

When using a polling model, consumers must continuously poll to determine when there's work to be done. Polling typically leads to increased network I/O and unnecessary delays in processing.

A push-based model allows consumers to automatically be notified when there are events to be consumed. Events are efficiently routed to consumers.



This is the end of the second module of the course "Service Orchestration and Choreography in Google Cloud."

In this module, "Event-Driven Applications," you learned about event-driven architecture. You also learned about the benefits of building event-driven applications.