Cloud Pub/Sub Concepts:

* Global Messaging and event ingestion.
* Serverless, no ops and fully-managed.
* Multiple publisher/subscriber patterns.
* At-least once delivery.
* Real-time or batch.
* Integrate with Cloud Dataflow – if you need time windowing or exactly once processing.

Cloud Pu/Sub Use Cases:

* Distributing Workloads – e,g you can queue a large no. of works and have them distributed among multiple workers.
* Asynchronous Workflows.
* Distributing Event Notifications.
* Distributed logging – rather than sending logging to a single source, they can be sent to a pub/sub topic to be consumed by multiple subscribers.
* Device data streaming.

Pub/Sub Basics:

1. Publishing Messages:

* Create a message containing your data – JSON payload, base64encoded, 10MB or less.
* Send the payload as a request to the Pub/Sub API specifying the topic to which the message should be published.

1. Receiving Messages:

* Create a subscription to a topic. - always associated to a single topic.
* There are two types of delivery methods to subscriptions
  + Pull is the default delivery method
    - You can make ad-hoc pull requests to the Pub/Sub API. Specifying your subscriptions to receive messages associated to that subscription.
    - Messages must be acknowledged. - or else the message remains in the top of the queue associated with your sub.
    - You won’t get next message or any messages after that until you acknowledge it.
  + Push will send messages to an endpoint:
    - Endpoint must use HTPPS with valid SSL cert.
    - Pub/Sub uses slow-start algorithm to moderate speed of messages sent to endpoints – it will gradually increase in speed but gradually back off if it receives any problem.
    - It accepts valid HTTP response code to acknowledge.

1. Integrations:

* Client libraries for popular languages.
* Cloud Dataflow.
* Cloud Functions.
* Cloud Run.
* Cloud IoT Core.

1. Developing for Pub/Sub:

* Local Pub/Sub emulator.
* You need Google Cloud SDK.
* Java Runtime Env 7+

Advanced Pub/Sub:

1. At least once delivery:

* Every message is delivered at least once for every subscription.
* Undelivered messages are deleted after the message retention duration. (default = 7 days; can config= 10s)
* Messages published before a subscription is created will not be delivered to that subscription.

1. Subscription Lifecycle:

* Subscriptions expire after 31 days of inactivity.
* New subscriptions with the same name have no relationship to the previous subscription.

1. Standard Model Limitations:

* Acknowledged messages are no longer available to subscribers.
* Every message must be processed by a subscription.

1. Seeking/Snapshot
2. Ordering Messages:

* Messages may not be received in order – that’s probably ok.
* Use timestamps when final order matters.
* Consider alternatives for transactional ordering.

1. Resource locations

* Messages are stored in the nearest region.
* Message storage policies allow you to control this.
* Additional egress fees may apply.

1. Monitoring:

Exam Tips:

Decouple Data:

* Spot where Pub/Sub would be a good fit to decouple components that would normally send data directly to each other. Pub/Sub can be a shock absorber, receiving data globally and allowing it to be consumed by other components at their own pace.

Decouple Services:

* Try to spot where Pub/Sub can add event logic to a stack. Pub/Sub can pass events from one system to another, using any of the patterns described to create asynchronous workflows.

Beware of Pub/Sub’s limitations:

* Pub/Sub has unrivalled capacity & latency globally but it has certain limitations. message data must be 10Mb or less in size and you need to be mindful of the expiry time of undelivered messages and unused subscriptions.

Apache Kafka:

* Pub/Sub maybe be a great cloud service o replace it with, esp if a case study calls for managed cloud services to be embraced. But Kafka has a much broader feature set, so make sure Pub/Sub still meets the necessary technical requirements.

Cloud IoT:

* Cloud IoT Core manages secure service device registration and connection, but all messages are handled by Cloud Pub/Sub.

Cloud Tasks:

* Other GCP asynchronous event system. Designed for explicit invocation by a publisher/controller as opposed to Pub/Sub’s decentralized approach.