Faculty of Engineering and Applied Science

SOFE 4630U: Cloud Computing

Group: 2 CRN: 74293 Section: 001

Project Milestone 1: Data Ingestion System (Cloud Pub/Sub, Individual)

Date: January 27th, 2025

William Chamberlain (100846922)

Ontario Tech University

Oshawa, Ontario

william.chamberlain@ontariotechu.net

GitHub Repository Link

https://github.com/wzc-OntarioTechU/SOFE4630-Project.git

Video Links

These videos are set to allow accounts belonging to ontariotechu.net to view. Please contact me if this needs to be changed.

Smart Meter

https://youtu.be/2X3drm81mx8

Design

https://youtu.be/7xm8ZqhuGyU

What is EDA?

EDA is an acronym for Event Driven Architecture. Event Driven Architecture is a different strategy for inter-process communication than the more traditional Message Driven Architecture (MDA). In Message Driven Architecture, processes or services send deliberate messages with destinations to each other, often synchronously and unicast. Event Driven Architecture takes a different approach, often using a message queue or broker. Event occurrences are published, logged, or sent to the queue or broker and other services can subscribe and monitor for these events.

What are its Advantages and Disadvantages?

EDA provides superior scaling and logging capabilities compared to MDA. Since EDA decouples the messaging between service instances, scaling both the number of instances and services becomes much easier and makes on demand services more effective. Since the broker or queue can keep a persistent log of all or select events, this data can be stored for troubleshooting and analytics in a unified service rather than logging being distributed across each service instance. MDA often has services sending messages unicast and sometimes synchronously, which can reduce performance when services layer requests, when a message needs to be dispersed to many other services (since one may need to keep track of all the others), or when a request takes a long time to return it may deadlock the system; additionally, logging may not be unified and sorting and storing these logs can become difficult.

EDA does have some caveats, however. Since messages are not sent directly to services requests that require synchronicity may take longer or have additional mechanisms to be achieved. This issue doesn’t arise often when a service is properly architected using microservices, but critical paths less suited to this pattern can occur. Depending on the dependencies between services, additional topics or queues may be needed for pipelining, but this matter can be architected. Stateful operations are also less suited since the state will need to be passed service to service in some additional mechanism or data in the event. A broker, queue, or message bus is required, which can make the architecture less suitable for very small projects where the overhead is unnecessary, but this is uncommon, particularly in cloud deployments.

Push Vs. Pull Subscriptions

Push and Pull refer the methods in which subscribers discover new events published to a topic or queue.

Push

Push Subscriptions, Events, or Notifications describe the strategy where a subscriber will register itself to the broker or queue, and request that the queue message it or distribute the events to it. This allows for fewer overall requests on the network since polling is not required but is more delicate to a service dying not becoming unregistered and increased workload on the broker or queue to track subscribers and send messages. The broker or queue pushes messages to subscribers.

Pull

Pull Subscriptions, or Events describe the opposite strategy where a subscriber will regularly poll the broker or queue for new events. This can lighten the workload on the broker in some situations by removing the need to track subscribers, but at the cost of increased network traffic, requests, and increased latency due to polling frequency. Pull subscriptions are far more resilient however as they are fully stateless a service can come online or offline without registrations.

Ordering Keys and Topics

Topics are essential to a well functioning EDA. Topics allow events of different natures and destinations to be separated into their own queues so that subscribers who are only interested in those events can be notified specifically, and to better format logging and data collection.

Normally event messages are not strictly organized within the broker, especially when a queue (topic) becomes partitioned for scaling. By adding an ordering key to events (as produced or on arrival), the destination partition of events can be controlled, allowing the broker to direct subscriber requests to an appropriate partition for retrieval.