**Using Cloud PubSub with Cloud Run [APPRUN]**

**Overview**

Pub/Sub enables applications to take advantage of efficient message queues. The service is compatible with a range of Google Cloud services, and in this lab, you learn how to integrate it with Cloud Run.

This lab is based on resolving a customer use case by using serverless infrastructure. The lab features three high level sections that resolve a technical problem:

* Situational Overview
* Requirements Gathering
* Developing a minimal viable product

Objectives

In this lab, you learn to:

* Enable the Cloud Run API.
* Deploy microservices to Cloud Run.
* Create a Pub/Sub topic.
* Invoke a Cloud Run service from a Pub/Sub subscription.

Prerequisites

These labs are based on intermediate knowledge of Google Cloud. While the steps required are covered in the content, it would be helpful to have familiarity with any of the following products:

* Pub/Sub
* Cloud Run

Situational overview



In this lab, you will help the development team at Critter Junction investigate the use of Pub/Sub for their requirements. The team would like to explore how to perform efficient queue processing within their applications.

Requirements gathering

The team at Critter Junction has a public web application and several microservices built on Google Cloud. Communication between the microservices is critical and needs a resilient form of messaging to be established between each application component.

The development team's previous attempts were unsuccessful due to the microservices needing to know a lot about each other ([High Coupling](https://en.wikipedia.org/wiki/Coupling_(computer_programming))). In addition, if a service was temporarily unavailable, messages would be lost.

The team needs a solution that includes a level of resilience without introducing additional service dependencies (Low Coupling) into their systems. Now that you know a bit more about Critter Junction and the issues they face, try to prioritize the key criteria for a solution.

Defining Critter Junction priorities

To ascertain the key use cases and priorities, initial discussions are held with the Critter Junction stakeholders. The results of the discussions are shown below:

|  |  |
| --- | --- |
| **Ref** | **User Story** |
| 1 | As a lead developer, I want to ensure that messaging is resilient, so service operations will be restored without needing manual intervention. |
| 2 | As a program manager, I want services to be capable of scaling seamlessly so additional transactional load does not lead to system instability. |
| 3 | As an operations lead, I want services to be managed so staff does not need to be reassigned from important maintenance work. |

From a discussion with the team leads, the following high level tasks are defined:

|  |  |
| --- | --- |
| **Ref** | **Definition of Done** |
| 1 | Establish an asynchronous component for inter-service communication. |
| 2 | Implement the proven scalability of the solution. |
| 3 | Services must run unsupervised. |

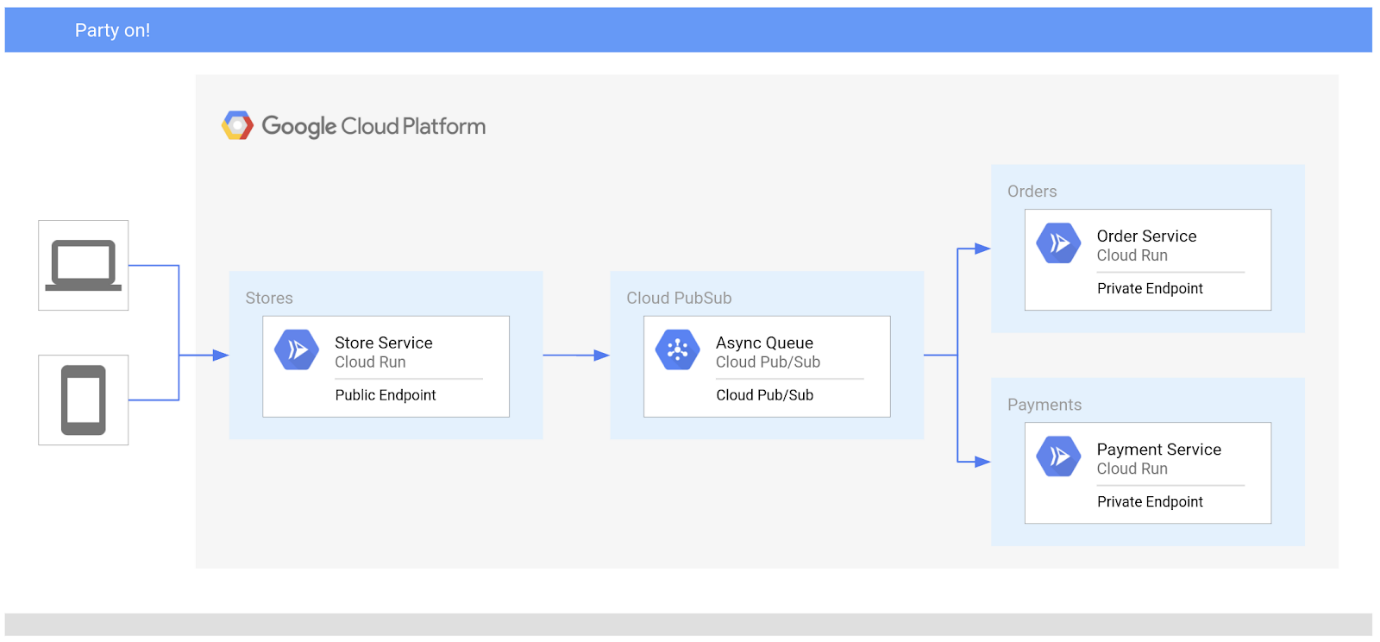
The team at Critter Junction is keen to define a solution that can be implemented quickly. In consideration of the requirements, the development team narrows their options down to:

* Pub/Sub
* Cloud Tasks

See [Pub/Sub versus Cloud Tasks](https://cloud.google.com/pubsub/docs/tasks-vs-pubsub) to learn more.

|  |  |  |
| --- | --- | --- |
| **Product** | **Use case** | **Choice** |
| ***Pub/Sub*** | ***"Optimal for more general event data ingestion and distribution patterns where some degree of control over execution can be sacrificed."*** | green check mark |
| Cloud Tasks | "Appropriate for use cases where a task producer needs to defer or control the execution timing of a specific webhook or remote procedure call." | red incorrect mark |

After considering the requirements, the development team chooses Pub/Sub because they only require a push based distribution pattern. The following high level architecture diagram summarizes the minimal viable product (MVP) that they need to investigate.



Critter Junction has multiple Cloud Run services that they would like integrated with Pub/Sub. To build an MVP, the following tasks are required:

* Deploy a producer service
* Deploy a consumer service
* Create a service account
* Create a Pub/Sub topic

Deploy a producer service

Critter Junction specifies that the externally facing *store* service should be configured as a public endpoint, indicating these requirements:

|  |  |  |
| --- | --- | --- |
| **Type** | **Permission** | **Description** |
| URL Access | --allow-unauthenticated | Make the service PUBLIC (Unauthenticated users can see it). |
| Invoke Permission | allUsers | Allow the service be invoked/triggered by anyone. |

The producer *store* service accepts public internet based connections for *purchase orders*. To do this, the service must not require authentication and must be able to be triggered by anyone.

Information collected by this service will be passed to the backend consumer services.

Deploy a consumer service

The development team also needs to configure the *order* service that can be accessed at a private endpoint. Unlike the **store** service, the **order** service is not meant to be publicly accessible over the internet, and should only be invoked by an account with the appropriate permissions.

For Cloud Run based services, this can be achieved by using the following settings:

|  |  |  |
| --- | --- | --- |
| **Type** | **Permission** | **Description** |
| URL Access | --no-allow-unauthenticated | Make the service PRIVATE (Only authenticated users can see it). |
| Invoke Role/Permission | Cloud Run Invoker | Only allow the service to be invoked by an account with the Cloud Run Invoker role. |

**Pub/Sub overview**

Pub/Sub is an asynchronous messaging service that decouples services that produce events from services that consume and process events.

Pub/Sub core concepts:

* Topic
* Subscription
* Message
* Message attribute

Pub/Sub requires a couple of options to be completed prior to successful deployment. In the Google Cloud console, Pub/Sub can be accessed under the Big Data menu option.

|  |  |
| --- | --- |
| **Field** | **Description** |
| Topic | A named resource to which messages are sent by publishers. |
| Subscription | A named resource representing the stream of messages from a single, specific topic, to be delivered to the subscribing application. For more details about subscriptions and message delivery semantics, see the [Subscriber Guide](https://cloud.google.com/pubsub/subscriber). |
| Message | The combination of data and (optional) attributes that a publisher sends to a topic and is eventually delivered to subscribers. |
| Message attribute | A key-value pair that a publisher can define for a message. For example, key ***iana.org/language\_tag*** and value **en** could be added to messages to mark them as readable by an English-speaking subscriber. |

Pub/Sub can be used in a wide variety of use cases, the most common of which are listed below:

|  |  |
| --- | --- |
| **Use Case** | **Example** |
| Balancing workloads in network clusters | For example, a large queue of tasks can be efficiently distributed among multiple workers, such as Compute Engine instances. |
| Implementing asynchronous workflows | For example, an order processing application can place an order on a topic, from which it can be processed by one or more workers. |
| Distributing event notifications | For example, a service that accepts user signups can send notifications whenever a new user registers, and downstream services can subscribe to receive notifications of the event. |
| Refreshing distributed caches | For example, an application can publish invalidation events to update the IDs of objects that have changed. |
| Logging to multiple systems | For example, a Google Compute Engine instance can write logs to the monitoring system, to a database for later querying, and so on. |
| Data streaming from various processes or devices | For example, a residential sensor can stream data to backend servers hosted in the cloud. |
| Reliability improvement | For example, a single-zone Compute Engine service can operate in additional zones by subscribing to a common topic, to recover from failures in a zone or region. |

To deliver a Pub/Sub message to a Cloud Run service, you need a Pub/Sub subscription. The subscription must be able to invoke the service using a service account with the appropriate permissions. In this lab, the consumer order service will be invoked by a subscription using the service account.

To achieve this functionality, the following activities are required:

* Create a Service Account
* Bind the Invoker Role permissions to the service account