# WHAT IS PUB/SUB? PUBLISH/SUBSCRIBE MESSAGING EXPLAINED



Known as pub/sub, Publish/Subscribe messaging is an asynchronous service-to-service communication method used in <u>serverless and microservices architectures</u>. Basically, the Pub/Sub model involves:

- A publisher who sends a message
- A subscriber who receives the message via a message broker

In this article, we'll see how pub/sub works, look at pros, cons, and use cases, and share a tutorial for setting up simple pub/sub messaging.

## **Basics of pub/sub messaging**

With the popularity of decoupled and <u>microservices-based</u> applications, proper communication between components and services is crucial for overall application functionality. Pub/Sub messaging helps with this in two crucial ways:

- Allowing <u>developers</u> to create decoupled applications easily with a reliable communication method
- Enabling users to create event-driven architectures easily



A pub/sub model allows messages to be broadcasted asynchronously across multiple sections of the applications.

The core component that facilitates this functionality is something called a Topic. The publisher will push messages to a Topic, and the Topic will instantly push the message to all the subscribers. This is what differentiates the Pub/Sub model from traditional message brokers, where a message queue will batch individual messages until a user or service requests these messages and retrieves them.

Whatever the message is in the Pub/Sub model, it will be automatically pushed to all the subscribers. The only exception is user-created policies for subscribers that will filter out messages.

This approach makes it possible to create event-driven services without constantly querying a message queue for messages. It also enables developers to create different isolated functions using the same message (data) that can be executed parallelly with the ability to serve multiple subscribers.

The Pub/Sub pattern isolates publishers from subscribers so that publishers do not need to know where the message is being used while the subscriber does not need to know about the publisher. This helps to improve the overall security of the application organically.

## Advantages of publish/subscribe pattern

A distributed microservices-based application developed using the pub/sub pattern benefits a whole organization, from software architects to QA engineers.

Here are the advantages of pub/sub:

#### **Decoupled/loosely coupled components**

Pub/Sub allows you to separate the communication and application logic easily, thereby creating isolated components. This results in:

- Creating more modularized, robust, and secure software components or modules
- Improving code quality and maintainability

#### Greater system-wide visibility

The simplicity of the pub/sub pattern means that users can understand the flow of the application easily.

The pattern also allows creating decoupled components that help us get a bird's eye view of the information flow. We can know exactly where information is coming from and where it is delivered without explicitly defining origins or destinations within the source code.

#### **Real-time communication**

Pub/sub delivers messages to subscribers instantaneously with push-based delivery, making it the ideal choice for near real-time communication requirements. This eliminates the need for any polling to check for messages in queues and reduces the delivery latency of the application.

## Ease of development

Since pub/sub is not dependent on <u>programming language</u>, protocol, or a specific technology, any supported message broker can be easily integrated into it using any programming language. Additionally, Pub/Sub can be used as a bridge to enable communications between components built using different languages by managing inter-component communications.

This leads to easy integrations with external systems without having to create functionality to facilitate communications or worry about security implications. We can simply publish a message to a topic and let the external application subscribe to the topic, eliminating the need for direct interaction with the underlying application.

## **Increased scalability & reliability**

This messaging pattern is considered elastic—we do not have to pre-define a set number of publishers or subscribers. They can be added to a required topic depending on the usage.

The separation between communication and logic also leads to easier troubleshooting as developers can focus on the specific component without worrying about it affecting the rest of the application.

Pub/sub also improves the scalability of an application by allowing to change message brokers architecture, filters, and users without affecting the underlying components. With pub/sub, a new messaging implementation is simply a matter of changing the topic if the message formats are compatible even with complex architectural changes.

## **Testability improvements**

With the modularity of the overall application, tests can be targeted towards each module, creating a more streamlined testing pipeline. This drastically reduces the test case complexity by targeting tests for each component of the application.

The pub/sub pattern also helps to easily understand the origin and destination of the data and the information flow. It is particularly helpful in testing issues related to:

- Data corruption
- Formatting
- Security

## Disadvantages of pub/sub pattern

Pub/Sub is a robust messaging service, yet it is not the best option for all requirements. Next, let's look briefly at some shortcomings of this pattern.

#### **Unnecessary complexity in smaller systems**

Pub/sub needs to be properly configured and maintained. Where scalability and a decoupled nature are not vital factors to your app, implementing Pub/Sub will be a waste of resources and lead to unnecessary complexity for smaller systems

#### Media streaming

Pub/sub is not suitable when dealing with media such as audio or video as they require smooth <u>synchronous streaming</u> between the host and the receiver. Because it does not support synchronous end-to-end communications, pub/sub messaging is ill-suited for:

- Video conferencing
- VOIP
- General media streaming applications

## Use cases for publish/subscribe messaging

So, when is the right time to use pub/sub?

The Pub/Sub pattern can be used across different industries to facilitate real-time and distributed communications. For instance, <u>automation</u> is a key area that benefits from this pattern.

The following sections describe common use cases of Pub/Sub.

## loT (Internet of Things)

With smart devices, we need a reliable and efficient way to gather and distribute information. A control node or server can publish updates that will be automatically delivered to all the subscribed <u>IoT devices</u>.

End-user IoT devices can also act as publishers and publish notifications, sensor information, etc., to the cloud, which will then be notified to the user.

#### System monitoring & event notifications

Pub/sub allows users to create topics to gather system information and push them to visualization and notification frontends.

This is highly useful when dealing with large-scale deployments:

- 1. Messages can be categorized into different topics.
- 2. All servers or services can publish the data to these common topics without the need for separate notification pipelines.

We can extend this functionality further by subscribing maintenance or management functions to a topic. For example, if a server reports an error, it will trigger a function to automatically replace that server.

#### **Database backup & replication**

It's essential to make backups with multiple databases spread across different technologies and vendors. We can configure periodic backups or snapshots using cron jobs.

However, suppose that we need to move these backups to different regions or cloud storage. In that case, we can use Pub/Sub messaging to create a pipeline that will push a message informing of completed backup. Then, a subscribed function will use that message as the trigger to start the migration or copy process.

#### Log management

Pub/Sub can act as the go-between to aggregate and distribute logs. We can collect logs from multiple locations and push them to subscribed services like elastic search or simply store them across different designations.

Logs can be filtered by issues, audit trails, notification, background tasks, etc., and direct to different subscribers, enabling proper <u>log management</u>.

## Pub/sub messaging services

There are multitudes of Pub/Sub messaging services, from dedicated message brokers to cloud offerings. Following is a list of some common Pub/Sub services.

- **Apache Kafka**. Developed by Apache, Kafka has robust Pub/Sub messaging features with message logs.
- Faye. Simple Pub/Sub service designed to power web applications with servers designed for NodeJS and Ruby.
- **Redis.** This is one of the <u>most popular message brokers</u> with support for both traditional message queues as well as pub/sub pattern implementations.
- Amazon SNS. The Amazon Simple Notification Service is a fully managed service that offers Pub/Sub messages.
- Google Pub/Sub. GCP offering for pub/sub messaging service implementation.
- Azure Service Bus. A robust messaging service (MaaS) solution that offers Pub/Sub pattern.

## Simple example: Publish/subscribe messaging

Since we now understand the Pub/Sub concepts, let's look at a simple workflow using Google Pub/Sub. It will publish a message to a topic and trigger a subscribed <u>Google function</u> to print the pushed message.

#### Step 1. Creating the topic

The first step is to create a Topic in Google Pub/Sub so that we can publish messages to that topic.



#### Step 2. Set up the trigger

Navigate inside the created topic (Test\_Topic) and click on the "Trigger Google Function" option. It will let you create a Google Function with the created topic as the trigger.



#### Step 3. Create the Google function (print\_message\_pubsub\_test)

The first screen lets you name the Google function and set up the topic as the trigger. We will be using <u>Python</u> to create the function that simply captures the pushed data and send them to Webhook.site.

Also, we'll be utilizing the requests library to create a POST request to send the data.

	٩	>-	?	۰	:
(···) Cloud Functions					
1 Configuration — 2 Code					
Basics  Function name *  print_message_pubsub_test  Region us-central  Trigger  Cloud Pub/Sub  Trigger type Cloud Pub/Sub Select a Cloud Pub/Sub topic *  projects/test-applications-315905/topics/Test_Topic  Retry on failure CANCEL					
NEXT CANCEL					

Cloud function code block:



```
import base64
import requests
```

```
def get_quote(event, context):
    # Decode the Message Data
    message = base64.b64decode(event).decode('utf-8')

    # Create Request
    url = "https://webhook.site/xxxxxx-xxxx-xxxx-739c28ebd7ad"
    request_headers = {"Content-type": "application/json"}
    request_data = {"quote": message}

    response = requests.post(url, data=request_data, headers=request_headers)

    # Print Response
    print(response.status_code)
    print(response.text)
```

Once the function is deployed successfully, you will notice that it indicates the Test\_Topic as the trigger for the function.

≡	Goo	gle Cloud Platform 💲	Test Application	ons 🔻 🤇	<b>Q</b> Search product	ts and resources		× > ?	1 :
()	Clou	d Functions Funct	tions	+ CREATE FUN	CTION CREFRES	н			
ۍ Fil	ter Filt	er functions							0 III
		Name 🛧	Region	Trigger	Runtime	Memory allocated	Executed function	Last deployed	Authentication
	0	print_message_pubsub_test	us-central1	Topic: Test_Topic	Python 3.8	256 MiB	get_quote	16 Jul 2021, 21:13:56	

#### Step 4. Set up the publisher

In this step, let's create a simple Python program to act as the publisher.

We will utilize the google cloud pubsub\_v1 library to create a Publisher client and get a random inspirational quote from quotable.io. Then we will publish a concatenated string of the author and quote to the topic (Test\_Topic)

message\_publish.py

```
from google.oauth2 import service account
from google.cloud import pubsub v1
import requests
# Create Authentication Credentials
project id = "test-applications-xxxxx"
topic_id = "Test_Topic"
gcp credentials =
service account.Credentials.from service account file('test-applications-
xxxx-xxxxxxxxxxx.json')
# Create Publisher Client
publisher = pubsub v1.PublisherClient(credentials=gcp credentials)
topic path = publisher.topic path(project id, topic id)
# Get a Random Quote
response = requests.get("https://api.quotable.io/random")
json response = response.json()
message = f"{json response} - {json response}"
# Publish the Message
data = message.encode("utf-8")
future = publisher.publish(topic path, data)
# Print Result
print(f"Published messages to {topic path} - {future.result()}.")
```

That's it! We've successfully configured the messaging pipeline. When we run the "message\_publish" script, it will publish the data to the Test\_Topic and trigger the Google Cloud Function (print\_message\_pubsub\_test), which will send the data to the Webhook site.

We can see the messages published to the topic within the Pub/Sub topic.

	Google Cl	Messages					
*** ***	← Te	To view messages pub	lished to this tonic s	elect or create (recommanded for testing) a Dull subscription			
		To view messages pub	nisneu to this topic, se	elect of create (recommended for testing) a Pull subscription.			
Þ		Select a Cloud Pub/Su projects/test-	b subscription */subscription	intions/Test Tonic-sub			•
=		[···]····					
0		Click Pull to vie	w messages and terr	porarily delay message delivery to other subscribers.			
	Tenind	Select Enable A be pulled at a ti	ACK messages and th ime. Click Pull again t	en click ACK next to the message to permanently prevent message delivery to other subscribers. o retrieve more messages from the backlog. Use this option cautiously in production environmen	Only a ts. If yo	i few messag ou miss the	jes will
	торіс а	acknowledgem	ent deadline (10 seco	onds), the message will be sent again if no other subscribers of this subscription acknowledged the	he mes	ssage. <u>Learn</u>	more
≡	Topic nam	PULL 🗌 Enable :	ack messages				
		<b>= Filter</b> Filter messa	iges			0	III
	Exp	➡ Filter Filter messa Publish time	iges Attribute keys	Message body	On	₽ Ack ↑	ш
	Exp	<ul> <li>Filter Filter messa</li> <li>Publish time</li> <li>16 Jul 2021, 20:39:16</li> </ul>	ges Attribute keys	Message body Carl Jung - The shoe that fits one person pinches another; there is no recipe for living	Ori —	<pre> <b>Ack ↑</b> ACK </pre>	•
	Exp Creat	Filter         Filter messa           Publish time         16 Jul 2021, 20:39:16           16 Jul 2021, 21:09:34         16 Jul 2021, 21:09:34	ges Attribute keys	Message body         Carl Jung - The shoe that fits one person pinches another; there is no recipe for living         Napoleon Hill - Most great people have attained their greatest success just one step	On 	Image: Control of the second seco	■ > >
	Exp Creat EXPC	Filter         Filter messa           Publish time         16 Jul 2021, 20:39:16           16 Jul 2021, 21:09:34         16 Jul 2021, 21:14:13	ges Attribute keys	Message body         Carl Jung - The shoe that fits one person pinches another; there is no recipe for living         Napoleon Hill - Most great people have attained their greatest success just one step         Pablo Picasso - I begin with an idea and then it becomes something else.	On 	<pre>   Ack ↑   Ack   Ack   Ack   Ack   Ack   Ack </pre>	■ > >
	Exp Creat EXPC	Filter         Filter messa           Publish time         I           16 Jul 2021, 20:39:16         I           16 Jul 2021, 21:09:34         I           16 Jul 2021, 21:14:13         I           16 Jul 2021, 21:14:30         I	iges Attribute keys	Message body         Carl Jung - The shoe that fits one person pinches another; there is no recipe for living         Napoleon Hill - Most great people have attained their greatest success just one step         Pablo Picasso - I begin with an idea and then it becomes something else.         Virginia Woolf - Some people go to priests; others to poetry; I to my friends.	On   	Ack 个 ACK ACK ACK ACK	•
	Exp Creat EXPC	Filter       Filter messa         Publish time       16 Jul 2021, 20:39:16         16 Jul 2021, 21:09:34       16 Jul 2021, 21:14:30         16 Jul 2021, 21:14:30       16 Jul 2021, 21:14:30         16 Jul 2021, 21:14:30       16 Jul 2021, 21:14:30	ges Attribute keys	Message body         Carl Jung - The shoe that fits one person pinches another; there is no recipe for living         Napoleon Hill - Most great people have attained their greatest success just one step         Pablo Picasso - I begin with an idea and then it becomes something else.         Virginia Woolf - Some people go to priests; others to poetry; I to my friends.         Epictetus - It is the nature of the wise to resist pleasures, but the foolish to be a slave to	On 	<ul> <li>Ack ↑</li> <li>Ack</li> <li>ACK</li> <li>ACK</li> <li>ACK</li> <li>ACK</li> <li>ACK</li> <li>ACK</li> </ul>	■
	Exp Creat EXPC	Filter       Filter messa         Publish time       16 Jul 2021, 20:39:16         16 Jul 2021, 21:09:34       16 Jul 2021, 21:14:13         16 Jul 2021, 21:14:30       16 Jul 2021, 21:14:36         16 Jul 2021, 21:14:36       16 Jul 2021, 21:14:45	iges Attribute keys	Message body         Carl Jung - The shoe that fits one person pinches another; there is no recipe for living         Napoleon Hill - Most great people have attained their greatest success just one step         Pablo Picasso - I begin with an idea and then it becomes something else.         Virginia Woolf - Some people go to priests; others to poetry; I to my friends.         Epictetus - It is the nature of the wise to resist pleasures, but the foolish to be a slave to         Jean-Paul Sartre - Freedom is what you do with what's been done to you.	On 	Ack ↑           Ack	•
	Exp Creat ExPC Creat	Filter       Filter messa         Publish time       16 Jul 2021, 20:39:16         16 Jul 2021, 21:09:34       16 Jul 2021, 21:14:13         16 Jul 2021, 21:14:30       16 Jul 2021, 21:14:30         16 Jul 2021, 21:14:30       16 Jul 2021, 21:14:45         16 Jul 2021, 21:14:45       16 Jul 2021, 21:14:55	iges Attribute keys	Message body         Carl Jung - The shoe that fits one person pinches another; there is no recipe for living         Napoleon Hill - Most great people have attained their greatest success just one step         Pablo Picasso - I begin with an idea and then it becomes something else.         Virginia Woolf - Some people go to priests; others to poetry; I to my friends.         Epictetus - It is the nature of the wise to resist pleasures, but the foolish to be a slave to         Jean-Paul Sartre - Freedom is what you do with what's been done to you.         Walter Lippmann - Where all think alike, no one thinks very much.	On 	Ack ↑           Ack	•

The logs of the Google cloud function will indicate that the function was triggered.

E Google Cloud Platform 🕽 Test Applications 🗸 🔍 🔍 🔍 🔍 🕄 🕄 🕄 🕄 🕄
Cloud Functions
<pre>     print_message_pubsub_test     Version 6, deployed at 16 Jul 2021, 21:13:56 ▼ </pre>
METRICS DETAILS SOURCE VARIABLES TRIGGER PERMISSIONS LOGS TESTING
Logs Showing 37 messages Default - Filter logs
2021-07-16T15:39:39.521062427Z print_message_pubsub_test 8pjs70b5dhmo Function execution took 2973 ms, finished with st
2021-07-16T15:42:24.009623Z Cloud Functions UpdateFunction us-central1:print_message_pubsub_test
2021-07-16T15:43:56.843171Z Cloud Functions UpdateFunction us-central1:print_message_pubsub_test
λ 2021-07-16T15:44:14.794712338Z print_message_pubsub_test 4c4jhjtsreuo Function execution started
2021-07-16T15:44:15.450Z print_message_pubsub_test 4c4jhjtsreuo 200
λ 2021-07-16T15:44:15.457074558Z print_message_pubsub_test 4c4jhjtsreuo Function execution took 665 ms, finished with sta
λ 2021-07-16T15:44:30.487277841Z print_message_pubsub_test 4c4jydi1gn50 Function execution started
2021-07-16T15:44:31.085Z print_message_pubsub_test 4c4jydi1gn50 200
λ 2021-07-16T15:44:31.087262121Z print_message_pubsub_test 4c4jydi1gn50 Function execution took 601 ms, finished with sta
λ 2021-07-16T15:44:37.526701291Z print_message_pubsub_test jxcivhvfxs51 Function execution started
2021-07-16T15:44:38.982Z print_message_pubsub_test jxcivhvfxs51 200
2021-07-16T15:44:38.984629574Z print_message_pubsub_test jxcivhvfxs51 Function execution took 1460 ms, finished with st
2021-07-16T15:44:45.424777351Z print_message_pubsub_test jxci3gspvw0l Function execution started
2021-07-16T15:44:45.970Z print_message_pubsub_test jxci3gspvw01 200
λ 2021-07-16T15:44:45.970995666Z print_message_pubsub_test jxci3gspvw0l Function execution took 547 ms, finished with sta
2021-07-16T15:44:53.104150056Z print_message_pubsub_test jxciifjolq9t Function execution started
2021-07-16T15:44:53.562Z print_message_pubsub_test jxciifjolq9t 200
2021-07-16T15:44:53.563373097Z print_message_pubsub_test jxciifjolq9t Function execution took 460 ms, finished with sta
1 No newer entries found matching current filter.

Finally, we can see all the messages that were received by the Webhook.site as shown below.

REQUESTS (5/500) Newest First	Request Details		Permalink	Raw content	Export as 🗸	Delete	
	POST	https://webhook.site/	739c28ebd7ad				
POST #da31a	Host	2600:1900:2001:2::14 whois					
2600:1900:2001:2::14	Date	07/16/2021 9:14:53 PM (a few seconds ago	)				
07/16/2021 9:14:53 PM	Size	73 bytes					
	ID	da31abe4-					
POST #2a65d	Files						
07/16/2021 9:14:45 PM	Headers						
	connection	close					
POST #98871	content-length	73					
2600:1900:2001:2::14	content-type	application/json					
07/16/2021 9:14:38 PM	accept	*/*					
	accept-encoding	gzip, deflate					
POST #61e95	user-agent	python-requests/2.26.0					
2600:1900:2000:1b:400::1c	host	webhook.site					
07/16/2021 9:14:30 PM	Query strings						
POST #836bd	(empty)						
2600:1900:2000:1b:400::1c 07/16/2021 9:14:15 PM	Form values						
	(empty)						
	Raw Content		✓ Fo	ormat JSON	Word-Wrap	Сор	
	<pre>quote=Walter+Lippmann+-+Where+all+think+alike%2C+no+one+thinks+very+much.</pre>						

Above is the basic structure of any Pub/Sub workflow. We can use it as a simple template and extend it to facilitate any functionality.

## Simple, powerful communication

The Pub/Sub messaging pattern is a powerful yet simple communication method. It acts as the cornerstone of powering real-time distributed microservices-based applications by handling all the communication between internal and external components.

Pub/Sub can be used to create asynchronous scalable message flows with minimal delivery delays due to all the benefits it offers over traditional message brokers.

# **Related reading**

- <u>BMC DevOps Blog</u>
- 15 Best Practices for Building a Microservices Architecture
- <u>Service-Oriented Architecture vs Microservices Architecture: Comparing SOA to MSA</u>
- <u>3 Kubernetes Patterns for Cloud Native Applications</u>
- Anti-Patterns vs Patterns: What is an Anti-Pattern?